
Language Reference Guide

InterBase XE3

August, 2012

© 2012 Embarcadero Technologies, Inc. Embarcadero, the Embarcadero Technologies logos, and all other Embarcadero Technologies product or service names are trademarks or registered trademarks of Embarcadero Technologies, Inc. All other trademarks are property of their respective owners.

Embarcadero Technologies, Inc. is a leading provider of award-winning tools for application developers and database professionals so they can design systems right, build them faster and run them better, regardless of their platform or programming language. Ninety of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero products to increase productivity, reduce costs, simplify change management and compliance and accelerate innovation. The company's flagship tools include: Embarcadero® Change Manager™, RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® and Rapid SQL®. Founded in 1993, Embarcadero is headquartered in San Francisco, with offices located around the world. Embarcadero is online at www.embarcadero.com.

August 20, 2012

Contents

Tables	vii	COMMIT	2-33
Chapter 1		CONNECT	2-34
Using the InterBase		COUNT()	2-38
Language Reference		CREATE DATABASE	2-38
Who Should Use this Book	1-1	CREATE DOMAIN	2-42
Topics Covered in This Book	1-1	CREATE ENCRYPTION	2-45
Chapter 2		CREATE EXCEPTION	2-46
SQL Statement and		CREATE GENERATOR	2-47
Function Reference		CREATE INDEX	2-48
SQL Flavors	2-1	CREATE JOURNAL	2-49
Embedded SQL (ESQL)	2-2	CREATE JOURNAL ARCHIVE	2-51
Dynamic SQL (DSQL)	2-2	CREATE PROCEDURE	2-54
Stored Procedure and Trigger Language	2-2	CREATE ROLE	2-60
Interactive SQL (isql)	2-2	CREATE SHADOW	2-61
SQL Dialects	2-2	CREATE TABLE	2-63
Dialects	2-2	CREATE TRIGGER	2-71
Transition Features	2-3	CREATE USER	2-77
Database Object Naming Conventions	2-3	CREATE VIEW	2-78
Statement List	2-4	DECLARE CURSOR	2-81
Function List	2-5	DECLARE CURSOR (BLOB)	2-82
Datatypes	2-6	DECLARE EXTERNAL FUNCTION	2-83
Exact Numerics	2-7	DECLARE FILTER	2-85
Addition and Subtraction	2-8	DECLARE STATEMENT	2-86
Multiplication	2-8	DECLARE TABLE	2-87
Division	2-8	DELETE	2-88
Error Handling	2-9	DESCRIBE	2-90
Statement and Function Reference	2-10	DISCONNECT	2-91
ALTER DATABASE	2-10	DROP DATABASE	2-92
ALTER DOMAIN	2-13	DROP DOMAIN	2-93
ALTER EXCEPTION	2-15	DROP ENCRYPTION	2-93
ALTER INDEX	2-15	DROP EXCEPTION	2-94
ALTER PROCEDURE	2-16	DROP EXTERNAL FUNCTION	2-94
ALTER TABLE	2-18	DROP FILTER	2-95
ALTER TRIGGER	2-24	DROP GENERATOR	2-96
ALTER USER	2-25	DROP INDEX	2-96
AVG()	2-26	DROP JOURNAL	2-97
BASED ON	2-27	DROP JOURNAL ARCHIVE	2-97
BEGIN DECLARE SECTION	2-28	DROP PROCEDURE	2-97
CASE	2-29	DROP ROLE	2-98
CAST()	2-30	DROP SHADOW	2-99
CLOSE	2-31	DROP TABLE	2-99
CLOSE (BLOB)	2-32	DROP TRIGGER	2-100
COALESCE()	2-33	DROP USER	2-100
		DROP VIEW	2-100
		END DECLARE SECTION	2-101
		EVENT INIT	2-101
		EVENT WAIT	2-102

EXECUTE	2-103	FOR SELECT...DO	3-10
EXECUTE IMMEDIATE	2-104	IF...THEN ... ELSE	3-11
EXECUTE PROCEDURE	2-105	Input Parameters	3-12
EXTRACT()	2-107	NEW Context Variables	3-12
FETCH	2-108	OLD Context Variables	3-13
FETCH (BLOB)	2-109	Output Parameters	3-14
GEN_ID()	2-110	POST_EVENT	3-15
GRANT	2-111	SELECT	3-16
INSERT	2-115	SUSPEND	3-17
INSERT CURSOR (BLOB)	2-117	WHEN ... DO	3-18
MAX()	2-117	Handling Exceptions	3-19
MIN()	2-118	Handling SQL Errors	3-20
NULLIF()	2-119	Handling InterBase Error Codes	3-20
OPEN	2-119	WHILE ... DO	3-21
OPEN (BLOB)	2-120		
PREPARE	2-121		
RELEASE SAVEPOINT	2-123		
REVOKE	2-123		
ROLLBACK	2-126		
SAVEPOINT	2-127		
SELECT	2-127		
SET DATABASE	2-133		
SET GENERATOR	2-135		
SET NAMES	2-136		
SET SQL DIALECT	2-137		
SET STATISTICS	2-138		
SET TRANSACTION	2-139		
SHOW SQL DIALECT	2-141		
SUM()	2-142		
UPDATE	2-142		
UPPER()	2-145		
WHENEVER	2-145		

Chapter 3

Procedures and Triggers

Creating Triggers and Stored Procedures. . . .	3-1
Statement Types Not Supported	3-2
Nomenclature Conventions	3-2
Assignment Statement	3-3
BEGIN ... END	3-4
Comment	3-5
DECLARE VARIABLE	3-5
EXCEPTION	3-6
EXECUTE PROCEDURE	3-7
EXECUTE STATEMENT	3-8
No Rows or Data Returned	3-8
One Row of Data Returned	3-9
Any Number of Data Rows Returned	3-9
Requirements and Contraints	3-10

Chapter 4

Keywords

InterBase Keywords	4-1
------------------------------	-----

Chapter 5

Error Codes and Messages

Error Sources	5-1
Error Reporting and Handling	5-2
Trapping Errors with WHENEVER.	5-2
Checking SQLCODE Value Directly.	5-2
InterBase Status Array	5-3
For More Information	5-4
SQLCODE Error Codes and Messages	5-5
SQLCODE Error Messages Summary	5-5
SQLCODE Codes and Messages	5-5
InterBase Status Array Error Codes	5-20

Chapter 6

System Tables, Temporary Tables, and Views

Overview	6-1
System Tables.	6-2
RDB\$CHARACTER_SETS	6-3
RDB\$CHECK_CONSTRAINTS	6-4
RDB\$COLLATIONS	6-4
RDB\$DATABASE	6-5
RDB\$DEPENDENCIES	6-6
RDB\$ENCRYPTIONS	6-7
RDB\$EXCEPTIONS	6-9
RDB\$FIELD_DIMENSIONS	6-9
RDB\$FIELDS.	6-10
RDB\$FILES	6-15
RDB\$FILTERS	6-16

RDB\$FORMATS	6-16
RDB\$FUNCTION_ARGUMENTS	6-17
RDB\$FUNCTIONS	6-18
RDB\$GENERATORS	6-19
RDB\$INDEX_SEGMENTS	6-20
RDB\$INDICES	6-20
RDB\$JOURNAL_ARCHIVES	6-22
RDB\$LOG_FILES	6-22
RDB\$PAGES	6-22
RDB\$PROCEDURE_PARAMETERS	6-23
RDB\$PROCEDURES	6-23
RDB\$REF_CONSTRAINTS	6-25
RDB\$RELATION_CONSTRAINTS	6-25
RDB\$RELATION_FIELDS	6-26
RDB\$RELATIONS	6-28
RDB\$ROLES	6-30
RDB\$SECURITY_CLASSES	6-31
RDB\$TRANSACTIONS	6-32
RDB\$TRIGGER_MESSAGES	6-32
RDB\$TRIGGERS	6-33
RDB\$TYPES	6-34
RDB\$USER_PRIVILEGES	6-35
RDB\$USERS	6-36
RDB\$VIEW_RELATIONS	6-37
System Temporary Tables	6-38
TMP\$ATTACHMENTS	6-38
TMP\$DATABASE	6-39
TMP\$HEAPS	6-42
TMP\$POOL_BLOCKS	6-42

TMP\$POOLS	6-44
TMP\$PROCEDURES	6-45
TMP\$RELATIONS	6-46
TMP\$STATEMENTS	6-47
TMP\$TRANSACTIONS	6-48
TMP\$TRIGGERS	6-49
System Views	6-51
CHECK_CONSTRAINTS	6-52
CONSTRAINTS_COLUMN_USAGE	6-52
REFERENTIAL_CONSTRAINTS	6-53
TABLE_CONSTRAINTS	6-54

Chapter 7

Character Sets and Collation Orders

InterBase Character Sets and Collation Orders	7-2
Character Set Storage Requirements	7-5
Support for Paradox and dBASE	7-5
Additional Character Sets and Collations	7-6
Specifying Character Sets	7-7
Default Character Set for a Database	7-7
Character Set for a Column in a Table	7-8
Character Set for a Client Attachment	7-8
Collation Order for a Column	7-8
Collation Order in Comparison	7-9
Collation Order in ORDER BY	7-9
Collation Order in a GROUP BY clause	7-9
Index	I-1

Tables

1.1	<i>Language Reference Chapters</i>	1-2	6.17	RDB\$INDEX_SEGMENTS	6-20
2.1	SQL functions	2-5	6.18	RDB\$INDICES	6-20
2.2	Datatypes supported by InterBase	2-6	6.19	RDB\$JOURNAL_ARCHIVES	6-22
2.3	SQLCODE and message summary	2-9	6.20	RDB\$PAGES	6-23
2.4	Statement and function format	2-10	6.21	RDB\$PROCEDURE_PARAMETERS	6-23
2.5	The ALTER TABLE statement.	2-20	6.22	RDB\$PROCEDURES	6-24
2.6	Compatible datatypes for CAST()	2-30	6.23	RDB\$REF_CONSTRAINTS	6-25
2.7	Language extensions for stored procedures. 2-57		6.24	RDB\$RELATION_CONSTRAINTS	6-25
2.8	Language extensions for triggers	2-74	6.25	RDB\$RELATION_FIELDS	6-26
2.9	EXTRACT() date and time parts	2-107	6.26	RDB\$RELATIONS	6-28
2.10	SQL privileges.	2-125	6.27	RDB\$ROLES	6-31
2.11	SQL Dialects.	2-138	6.28	RDB\$SECURITY_CLASSES	6-31
2.12	SQL Dialects.	2-141	6.29	RDB\$TRANSACTIONS	6-32
3.1	SUSPEND, EXIT, and END	3-17	6.30	RDB\$TRIGGER_MESSAGES	6-32
5.1	Status array codes that require rollback and retry.	5-4	6.31	RDB\$TRIGGERS	6-33
5.2	Where to find error-handling topics	5-4	6.32	RDB\$TYPES	6-34
5.3	SQLCODE and messages summary.	5-5	6.33	RDB\$USER_PRIVILEGES	6-35
5.4	SQLCODE codes and messages	5-6	6.34	RDB\$USERS	6-36
5.5	InterBase status array error codes	5-20	6.35	RDB\$VIEW_RELATIONS	6-37
6.1	System tables	6-2	6.36	TMP\$ATTACHMENTS	6-38
6.2	RDB\$CHARACTER_SETS	6-3	6.37	TMP\$DATABASE	6-39
6.3	RDB\$CHECK_CONSTRAINTS	6-4	6.38	TMP\$HEAPS	6-42
6.4	RDB\$COLLATIONS	6-4	6.39	TMP\$POOL_BLOCKS	6-42
6.5	RDB\$DATABASE	6-5	6.40	TMP\$POOLS	6-45
6.6	RDB\$DEPENDENCIES	6-6	6.41	TMP\$PROCEDURES	6-45
6.7	RDB\$ENCRYPTIONS	6-8	6.42	TMP\$RELATIONS	6-46
6.8	RDB\$EXCEPTIONS	6-9	6.43	TMP\$STATEMENTS	6-47
6.9	RDB\$FIELD_DIMENSIONS	6-10	6.44	TMP\$TRANSACTIONS	6-48
6.10	RDB\$FIELDS	6-10	6.45	TMP\$TRIGGERS	6-49
6.11	RDB\$FILES	6-15	6.46	CHECK_CONSTRAINTS	6-52
6.12	RDB\$FILTERS	6-16	6.47	CONSTRAINTS_COLUMN_USAGE	6-53
6.13	RDB\$FORMATS	6-17	6.48	REFERENTIAL_CONSTRAINTS	6-53
6.14	RDB\$FUNCTION_ARGUMENTS	6-17	6.49	TABLE_CONSTRAINTS	6-54
6.15	RDB\$FUNCTIONS	6-18	7.1	Character sets and collation orders	7-2
6.16	RDB\$GENERATORS	6-19	7.2	Character sets corresponding to DOS code pages	7-6

Using the InterBase Language Reference

The InterBase *Language Reference* details the syntax and semantics of SQL and Dynamic SQL (DSQL) statements for embedded applications programming and for **isql**, the InterBase interactive SQL utility. It also describes additional language and syntax that is specific to InterBase stored procedures and triggers.

Who Should Use this Book

The *Language Reference* assumes a general familiarity with SQL, data definition, data manipulation, and programming practice. It is a syntax and usage resource for:

- Programmers writing embedded SQL and DSQL database applications.
- Programmers writing directly to the InterBase applications programming interface (API), who need to know supported SQL syntax.
- Database designers who create and maintain databases and tables with **isql**.
- Users who perform queries and data manipulation operations through **isql**.

Topics Covered in This Book

The following table lists the chapters in the *Language Reference*, and provides a brief description of them:

Table 1.1 *Language Reference Chapters*

Chapter	Description
Chapter 1, “Using the InterBase Language Reference”	Introduces the book, and describes its intended audience.
Chapter 2, “SQL Statement and Function Reference”	Provides syntax and usage information for SQL and DSQL language elements.
Chapter 3, “Procedures and Triggers”	Describes syntax and usage information for stored procedure and trigger language.
Chapter 4, “Keywords”	Lists keywords, symbols, and punctuation, that have special meaning to InterBase.
Chapter 5, “Error Codes and Messages”	Summarizes InterBase error messages and error codes.
Chapter 6, “System Tables, Temporary Tables, and Views”	Describes InterBase system tables and views that track metadata.
Chapter 7, “Character Sets and Collation Orders”	Explains all about character sets and corresponding collation orders for a variety of environments and uses.

Note For a listing of functions provided in the InterBase UDF library, see the “Working with UDFs and Blob Filters” chapter in the *Developer’s Guide*.

SQL Statement and Function Reference

This chapter provides the syntax and usage for InterBase SQL language elements. It includes the following topics:

- SQL variants and dialects
- Database object naming conventions
- Lists of SQL statements and functions
- A description of each InterBase datatype
- An introduction to using SQLCODE to handle errors
- How to use statement and function definitions
- A reference entry for each SQL statement supported by InterBase

SQL Flavors

Although InterBase SQL follows the ISO/IEC 9075:1992 standard closely, there are small differences. Differences also exist among the three major flavors of InterBase SQL: embedded SQL, dynamic SQL (DSQL), and the procedure and trigger language.

Embedded SQL (ESQL)

The embedded form of SQL is used in programs written in traditional languages such as C and Pascal. A preprocessor turns SQL statements into host language data structures and calls to the InterBase server. The embedded language is written into the program; its statements cannot be generated dynamically. Statements in embedded SQL are terminated with a semicolon.

Dynamic SQL (DSQL)

DSQL allows a program to create statements at run time. It can be used from conventional languages through the InterBase API. More often, it is used from modern development environments such as Delphi, which hide the nuts and bolts of the API. A completed DSQL statement is very much like the “embedded” language, without the “EXEC SQL” and without the terminating semicolon.

Stored Procedure and Trigger Language

Triggers and stored procedures are written in a variant of the embedded language, extended to provide flow control, conditional expressions, and error handling. Certain constructs, including all DDL (Data Definition Language) statements, are omitted. Within a trigger or stored procedure, statements are separated by semicolons.

Interactive SQL (isql)

The interactive query language, **isql**, is very similar to DSQL, with some omissions (cursors, for example) and a few additions (SET and SHOW statements). Like embedded SQL, **isql** statements must be terminated with a semicolon.

SQL Dialects

Starting with version 6, InterBase is closer to the ISO/IEC 9075:1992 standard than previous versions in several ways. Some of those ways are incompatible with earlier implementations of SQL. In the current InterBase, each client and database has a SQL *dialect*: an indicator that instructs an InterBase server how to interpret *transition features*: those features whose meanings have changed between InterBase versions. See the Migration appendix in the [Operations Guide](#) for information about using dialects and transition features.

Dialects

- Dialect 1: transition features are interpreted as in InterBase version 5.6 and earlier.

- Dialect 2: diagnostic mode, where transition features are recognized and flagged with a warning.
- Dialect 3: transition features are interpreted as SQL-92 compliant.

Transition Features

- Double quote ("): changed from a synonym for the single quote (') to the delimiter for an object name
- Large exact numerics: DECIMAL and NUMERIC datatypes with precision greater than 9 are stored at INT64 instead of DOUBLE PRECISION
- DATE, TIME, and TIMESTAMP datatypes:
 - DATE has changed from a 64-bit quantity containing both date and time information to a 32-bit quantity containing only date information
 - TIME is a 32-bit quantity containing only time information
 - TIMESTAMP is a 64-bit quantity containing both date and time information (same as DATE in InterBase 5 and older)

Database Object Naming Conventions

When an applications programmer or end user creates a database object or refers to it by name, case is unimportant. The following limitations on naming database objects must be observed:

- Start each name with an alphabetic character (A–Z or a–z).
- Restrict object names to 67 characters, including dollar signs (\$), underscores (_), 0 to 9, A to Z, and a to z. Some objects, such as constraint names, are restricted to 27 bytes in length.
- Keep object names unique. In all cases, objects of the same type—all tables, for example—*must* be unique. In most cases, object names must also be unique within the database.

To use keywords, ASCII characters, case-sensitive strings, or spaces (except for trailing spaces) in an object name, enclose the name in double quotes. It is then a *delimited identifier*. Delimited identifiers must always be referenced in double quotes. In InterBase dialect 3, names enclosed in double quotes are case sensitive. For example:

```
SELECT "CodAR" FROM MyTable
```

is different from:

```
SELECT "CODAR" FROM MyTable
```

This behavior conforms to ANSI SQL semantics for delimited identifiers.

For more information about naming database objects with CREATE or DECLARE statements, see the *Language Reference*.

Statement List

This chapter describes the following SQL statements:

ALTER DATABASE	ALTER DOMAIN	ALTER EXCEPTION
ALTER INDEX	ALTER PROCEDURE	ALTER TABLE
ALTER TRIGGER	ALTER USER	BASED ON
BEGIN DECLARE SECTION	CASE	CLOSE
CLOSE (BLOB)	COALESCE()	COMMIT
CONNECT	CREATE DATABASE	CREATE DOMAIN
CREATE ENCRYPTION	CREATE EXCEPTION	CREATE GENERATOR
CREATE INDEX	CREATE JOURNAL	CREATE JOURNAL ARCHIVE
CREATE PROCEDURE	CREATE ROLE	CREATE SHADOW
CREATE TABLE	CREATE TRIGGER	CREATE USER
CREATE VIEW	DECLARE CURSOR	DECLARE CURSOR (BLOB)
DECLARE EXTERNAL FUNCTION	DECLARE FILTER	DECLARE STATEMENT
DECLARE TABLE	DELETE	DESCRIBE
DISCONNECT	DROP DATABASE	DROP DOMAIN
DROP ENCRYPTION	DROP EXCEPTION	DROP EXTERNAL FUNCTION
DROP FILTER	DROP GENERATOR	DROP INDEX
DROP JOURNAL	DROP JOURNAL ARCHIVE	DROP PROCEDURE
DROP ROLE	DROP SHADOW	DROP TABLE
DROP TRIGGER	DROP VIEW	DROP USER
END DECLARE SECTION	EVENT INIT	EVENT WAIT

EXECUTE	EXECUTE IMMEDIATE	EXECUTE PROCEDURE
FETCH	FETCH (BLOB)	GRANT
INSERT	INSERT CURSOR (BLOB)	NULLIF()
OPEN	OPEN (BLOB)	PREPARE
RELEASE SAVEPOINT	REVOKE	ROLLBACK
SAVEPOINT	SELECT	SET DATABASE
SET GENERATOR	SET NAMES	SET SQL DIALECT
SET STATISTICS	SET TRANSACTION	SHOW SQL DIALECT
UPDATE	WHENEVER	

Function List

The following table lists the SQL functions described in this chapter:

Table 2.1 SQL functions

Function	Type	Purpose
AVG()	Aggregate	Calculates the average of a set of values
CAST()	Conversion	Converts a column from one datatype to another
COUNT()	Aggregate	Returns the number of rows that satisfy a query's search condition
EXTRACT()	Conversion	Extracts date and time information from DATE, TIME, and TIMESTAMP values
GEN_ID()	Numeric	Returns a system-generated value
MAX()	Aggregate	Retrieves the maximum value from a set of values
MIN()	Aggregate	Retrieves the minimum value from a set of values
SUM()	Aggregate	Totals the values in a set of numeric values
UPPER()	Conversion	Converts a string to all uppercase

Aggregate functions perform calculations over a series of values, such as the columns retrieved with a SELECT statement.

Conversion functions transform datatypes, either converting them from one type to another, or by changing the scale or precision of numeric values, or by converting CHARACTER datatypes to all uppercase.

The numeric function, GEN_ID(), produces a system-generated number that can be inserted into a column requiring a numeric datatype.

Datatypes

InterBase supports most SQL datatypes, a dynamically sizable datatype called a Blob, and arrays of datatypes. It does not support arrays of Blobs. The following table lists the datatypes available to SQL statements in InterBase:

Table 2.2 Datatypes supported by InterBase

Name	Size	Range/Precision	Description
BLOB	Variable	<ul style="list-style-type: none"> None Blob segment size is limited to 64K 	<ul style="list-style-type: none"> Dynamically sizable datatype for storing large data such as graphics, text, and digitized voice Basic structural unit is the segment Blob subtype describes Blob contents
BOOLEAN	16 bits	<ul style="list-style-type: none"> TRUE FALSE UNKNOWN 	<ul style="list-style-type: none"> Represents truth values TRUE, FALSE, and UNKNOWN Requires ODS 11 or higher, any dialect
CHAR(<i>n</i>)	<i>n</i> characters	<ul style="list-style-type: none"> 1 to 32,767 bytes Character set character size determines the maximum number of characters that can fit in 32K 	<ul style="list-style-type: none"> Fixed length CHAR or text string type Alternate keyword: CHARACTER
DATE	32 bits, signed ¹	1 Jan 100 a.d. to 29 Feb 32768 a.d.	ISC_DATE; stores a date as a 32-bit longword
DECIMAL (<i>precision</i> , <i>scale</i>)	Variable (16, 32, or 64 bits)	<ul style="list-style-type: none"> <i>precision</i> = 1 to 18; specifies at least <i>precision</i> digits of precision to store <i>scale</i> = 1 to 18; specifies number of decimal places for storage Must be less than or equal to <i>precision</i> 	<ul style="list-style-type: none"> Number with a decimal point <i>scale</i> digits from the right Example: DECIMAL(10, 3) holds numbers accurately in the following format: ppppppp.sss
DOUBLE PRECISION	64 bits ²	2.225×10^{-308} to 1.797×10^{308}	IEEE double precision: 15 digits

Table 2.2 Datatypes supported by InterBase (*continued*)

Name	Size	Range/Precision	Description
FLOAT	32 bits	1.175×10^{-38} to 3.402×10^{38}	IEEE single precision: 7 digits
INTEGER	32 bits	−2,147,483,648 to 2,147,483,647	Signed long (longword)
NUMERIC (<i>precision</i> , <i>scale</i>)	Variable (16, 32, or 64 bits)	<ul style="list-style-type: none"> <i>precision</i> = 1 to 18; specifies exactly <i>precision</i> digits of precision to store <i>scale</i> = 1 to 18; specifies number of decimal places for storage Must be less than or equal to <i>precision</i> 	<ul style="list-style-type: none"> Number with a decimal point <i>scale</i> digits from the right Example: NUMERIC(10,3) holds numbers accurately in the following format: ppppppp.sss
SMALLINT	16 bits	−32,768 to 32,767	Signed short (word)
TIME	32 bits, unsigned	0:00 AM to 23:59.9999 PM	ISC_TIME
TIMESTAMP	64 bits	1 Jan 100 a.d. to 29 Feb 32768 a.d.	Also includes time information
VARCHAR (<i>n</i>)	<i>n</i> characters	<ul style="list-style-type: none"> 1 to 32,765 bytes Character set character size determines the maximum number of characters that can fit in 32K 	<ul style="list-style-type: none"> Variable length CHAR or text string type Alternate keywords: CHAR VARYING, CHARACTER VARYING

1. InterBase version 5 had a DATE datatype that was 64 bits long and included both the date and time. InterBase version 6 and later recognizes that type if you have specified dialect 1; in dialect 3, that type is called TIMESTAMP.
2. Actual size of DOUBLE is platform-dependent. Most platforms support the 64-bit size.

Exact Numerics

All NUMERIC and DECIMAL datatypes are stored as exact numerics: 16, 32, or 64 bits, depending on the precision. NUMERIC and DECIMAL datatypes with precision greater than 9 are referred to as *large exact numerics*.

- If one operand is an approximate numeric, the result of any dyadic operation (addition, subtraction, multiplication, division) is DOUBLE PRECISION.
- Any value that can be stored in a DECIMAL(18,S) can also be specified as the default value for a column or a domain.

Addition and Subtraction

If both operands are exact numeric, adding or subtracting the operands produces an exact numeric with a precision of 18 and a scale equal to the larger of the two. For example:

```
CREATE TABLE t1 (n1 NUMERIC(16,2), n2 NUMERIC(16,3));
INSERT INTO t1 VALUES (12.12, 123.123);
COMMIT;
```

The following query returns the integer 135.243. The largest scale of the two operands is 3; therefore, the scale of the sum is 3.

```
SELECT n1 + n2 FROM t1;
```

Similarly, the following query returns the integer -111.003:

```
SELECT n1 - n2 FROM t1;
```

If either of the operands is approximate numeric (FLOAT, REAL, or DOUBLE PRECISION), the result is DOUBLE PRECISION.

Multiplication

If both operands are exact numeric, multiplying the operands produces an exact numeric with a precision of 18 and a scale equal to the sum of the scales of the operands. For example:

```
CREATE TABLE t1 (n1 NUMERIC(16,2), n2 NUMERIC(16,3));
INSERT INTO t1 VALUES (12.12, 123.123);
COMMIT;
```

the following query returns the integer 1492.25076 because n1 has a scale of 2 and n2 has a scale of 3. the sum of the scales is 5.

```
SELECT n1*n2 FROM t1
```

If one of the operands is approximate numeric (FLOAT, REAL, or DOUBLE PRECISION), the result is DOUBLE PRECISION.

Division

If both operands are exact numeric, dividing the operands produces an exact numeric with a precision of 18 and a scale equal to the sum of the scales of the operands. If at least one operand of a division operator has an approximate numeric type (FLOAT, REAL, or DOUBLE PRECISION), the result is DOUBLE PRECISION.

For example, in the following table, division operations produce a variety of results:

```
CREATE TABLE t1 (i1 INTEGER, i2 INTEGER, n1 NUMERIC(16,2)
  n2 NUMERIC(16,2));
INSERT INTO t1 VALUES (1, 3, 1.00, 3.00);
```

```
COMMIT;
```

The following query returns the integer 0 because each operand has a scale of 0, so the sum of the scales is 0:

```
SELECT i1/i2 FROM t1
```

The following query returns the NUMERIC(18,2) value 0.33, because the sum of the scales 0 (operand 1) and 2 (operand 2) is 2:

```
SELECT i1/n2 FROM t1
```

The following query returns the NUMERIC(18,4) value 0.3333, because the sum of the two operand scales is 4:

```
SELECT n1/n2 FROM t1
```

In InterBase 5 and earlier, any of the above division operations would have returned the DOUBLE PRECISION value 0.3333333333333333.

Error Handling

Every time an executable SQL statement is executed, the SQLCODE variable is set to indicate its success or failure. No SQLCODE is generated for declarative statements that are not executed, such as DECLARE CURSOR, DECLARE TABLE, and DECLARE STATEMENT.

The following table lists values that are returned to SQLCODE:

Table 2.3 SQLCODE and message summary

SQLCODE	Message	Meaning
< 0	SQLERROR	Error occurred; statement did not execute
0	SUCCESS	Successful execution
+1–99	SQLWARNING	System warning or informational message
+100	NOT FOUND	No qualifying rows found, or end of current active set of rows reached

When an error occurs in **isql**, InterBase displays an error message.

In embedded applications, the programmer must provide error handling by checking the value of SQLCODE.

To check SQLCODE, use one or a combination of the following approaches:

- Test for SQLCODE values with the WHENEVER statement.
- Check SQLCODE directly.
- Use the *isc_print_sqlerror()* routine to display specific error messages.

For more information about error handling, see the [Embedded SQL Guide](#).

Statement and Function Reference

The following is the reference of SQL statements and functions available in InterBase.

Each statement and function definition includes the following elements:

Table 2.4 Statement and function format

Element	Description
Title	Statement name
Definition	The statement's main purpose and availability
Syntax	Diagram of the statement and its parameters
Argument	Parameters available for use with the statement
Description	Information about using the statement
Examples	Examples of using the statement in a program and in isql
See also	Where to find more information about the statement or others related to it

Most statements can be used in SQL, DSQL, and **isql**. In many cases, the syntax is nearly identical, except that embedded SQL statements must always be preceded by the EXEC SQL keywords. EXEC SQL is omitted from syntax statements for clarity.

In other cases there are small, but significant differences among SQL, DSQL, and **isql** syntax. In these cases, separate syntax statements appear under the statement heading.

ALTER DATABASE

Changes the characteristics of the current database. Available in **gpre**, DSQL, and **isql**, but not in the trigger or stored procedure language.

Syntax

```
ALTER {DATABASE | SCHEMA}
    {ADD add_clause | DROP drop_clause | ENCRYPT key_name | DECRYPT
key_name | SET set_clause};

add_clause = FILE 'filespec' [fileinfo] [add_clause] | ADMIN
OPTION

fileinfo = LENGTH [=] int [PAGE[S]]
| STARTING [AT [PAGE]] int [fileinfo]
```

drop_clause = ADMIN OPTION

key_name = ENCRYPT / DECRYPT

set_clause = { FLUSH INTERVAL <number> / NO FLUSH INTERVAL /
GROUP COMMIT / NO GROUP COMMIT / LINGER INTERVAL <number> /
NO LINGER INTERVAL / PAGE CACHE <number> / RECLAIM INTERVAL
<number> / NO RECLAIM INTERVAL / SYSTEM ENCRYPTION PASSWORD
<255-character string> / NO SYSTEM ENCRYPTION PASSWORD }

Important In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	Description
SCHEMA	Alternative keyword for DATABASE
ADD FILE ' <i>filespec</i> '	Adds one or more secondary files to receive database pages after the primary file is filled; for a remote database, associate secondary files with the same node
LENGTH [=] <i>int</i> [PAGE[S]]	Specifies the range of pages for a secondary file by providing the number of pages in each file
STARTING [AT [PAGE]] <i>int</i>	Specifies a range of pages for a secondary file by providing the starting page number
ADD ADMIN OPTION	Enable embedded user authentication
DROP ADMIN OPTION	Disable embedded user authentication
ENCRYPT <i>key_name</i>	Uses the named encryption key to encrypt the database. Encrypting a database causes all pages to be encrypted. Only the database owner can encrypt a database.
DECRYPT <i>key_name</i>	Uses the named encryption key to decrypt the database. Decrypting a database causes all pages to be decrypted and rewritten in plaintext. Only the database owner can decrypt a database.
SET FLUSH INTERVAL <number>	Enables database flush. The interval <number> is interpreted in units of seconds.
SET NO FLUSH INTERVAL	Disables database flush.
SET GROUP COMMIT	Allows transactions to be committed by a background cache writer thread.
SET NO GROUP COMMIT	Disable group commit.
SET LINGER INTERVAL	Allows a database to remain in memory after the last user detaches. Interval is seconds

Argument	Description
SET NO LINGER INTERBAL	Disable database linger.
SET RECLAIM INTERVAL	Reclaim interval is in seconds. Determines how often the garbage collector thread will run to release memory from unused procedures, triggers, and internal system queries back to InterBase memory heap.
SET NO RECLAIM INTERVAL	Disable memory reclamation.
SET SYSTEM ENCRYPTION PASSWORD	Necessary to create encryption keys and perform encryption. InterBase uses a System Encryption Password (SEP) to protect the encryption keys that are used to encrypt the database and/or database columns. Note: Only the SYSDSO (Data Security Owner) can create this password. For more information about using InterBase encryption, see Chapter 13, “Encrypting Your Data” in the Data Definition Guide .
SET NO SYSTEM ENCRYPTION PASSWORD	Deletes the password if there are no existing encryption keys. NOTE: Only SYSDSO can delete a password.
SET PAGE CACHE	Sets database page buffer cache limit. Also, tries to expand cache to that limit.

Description ALTER DATABASE adds secondary files to an existing database. Secondary files permit databases to spread across storage devices, but they must remain on the same node as the primary database file. A database can be altered by its creator, the SYSDBA user, and any users with operating system root privileges.

ALTER DATABASE requires exclusive access to the database.

Note InterBase dynamically expands the last file in a database as needed. The maximum size of the last file is system-dependent. You should be aware that specifying a LENGTH for such files has no effect.

You cannot use ALTER DATABASE to split an existing database file. For example, if your existing database is 80,000 pages long and you add a secondary file STARTING AT 50000, InterBase starts the new database file at page 80,001.

Tip To split an existing database file into smaller files, back it up and restore it. When you restore a database, you are free to specify secondary file sizes at will, without reference to the number and size of the original files.

Example The following **isql** statement adds two secondary files to an existing database. The command creates a secondary database file called *employee2.ib* that is 10,000 pages long and another called *employee3.ib*. InterBase starts using *employee2.ib* only when the primary file reaches 10,000 pages.

```
ALTER DATABASE
  ADD FILE 'employee2.ib'
  STARTING AT PAGE 10001 LENGTH 10000
  ADD FILE 'employee3.ib';
```

See also CREATE DATABASE, DROP DATABASE

See the [Data Definition Guide](#) for more information about multi-file databases and the [Operations Guide](#) for more information about exclusive database access.

For detailed information on encryption and decryption, see the topics “Encrypting Data” (page 13-9) and “Decrypting Data” (page 13-11) in the [Data Definition Guide](#).

ALTER DOMAIN

Changes a domain definition. Available in **gpre**, **DSQL**, and **isql**, but not in the stored procedure or trigger language.

```
ALTER DOMAIN { name | old_name TO new_name }
  SET DEFAULT {literal | NULL | USER}
  | DROP DEFAULT
  | ADD [CONSTRAINT] CHECK (dom_search_condition)
  | DROP CONSTRAINT
  | new_col_name
  | TYPE datatype;

dom_search_condition =
  VALUE operator val
  | VALUE [NOT] BETWEEN val AND val
  | VALUE [NOT] LIKE val [ESCAPE val]
  | VALUE [NOT] IN (val [, val ...])
  | VALUE IS [NOT] NULL
  | VALUE [NOT] CONTAINING val
  | VALUE [NOT] STARTING [WITH] val
  | (dom_search_condition)
  | NOT dom_search_condition
  | dom_search_condition OR dom_search_condition
  | dom_search_condition AND dom_search_condition

operator = {= | < | > | <= | >= | !< | !> | <> | !=}
```

Important In SQL statements passed to **DSQL**, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	Description
<i>name</i>	Name of an existing domain
SET DEFAULT	Specifies a default column value that is entered when no other entry is made. Values: <ul style="list-style-type: none"> <i>literal</i>—Inserts a specified string, numeric value, or date value NULL—Enters a NULL value USER—Enters the user name of the current user; column must be of compatible text type to use the default Defaults set at column level override defaults set at the domain level
DROP DEFAULT	Drops an existing default
ADD [CONSTRAINT] CHECK <i>dom_search_condition</i>	Adds a CHECK constraint to the domain definition; a domain definition can include only one CHECK constraint
DROP CONSTRAINT	Drops CHECK constraint from the domain definition
<i>new_col_name</i>	Changes the domain name
TYPE <i>data_type</i>	Changes the domain datatype

Description ALTER DOMAIN changes any aspect of an existing domain except its NOT NULL setting. Changes made to a domain definition affect all column definitions based on the domain that have not been overridden at the table level.

Note To change the NOT NULL setting of a domain, drop the domain and recreate it with the desired combination of features.

The TYPE clause of ALTER DOMAIN does not allow you to make datatype conversions that could lead to data loss.

A domain can be altered by its creator, the SYSDBA user, and any users with operating system root privileges.

Example The following **isql** statements create a domain that must have a value > 1,000, then alter it by setting a default of 9,999:

```
CREATE DOMAIN CUSTNO
  AS INTEGER
  CHECK (VALUE > 1000);
ALTER DOMAIN CUSTNO SET DEFAULT 9999;
```

See also CREATE DOMAIN, CREATE TABLE, DROP DOMAIN

For a complete discussion of creating domains, and using them to create column definitions, see the *Data Definition Guide*.

ALTER EXCEPTION

Changes the message associated with an existing exception. Available in DSQL and **isql** but not the embedded language or stored procedure and trigger language.

Syntax ALTER EXCEPTION *name* 'message'

Argument	Description
<i>name</i>	Name of an existing exception message
'message'	Quoted string containing ASCII values

Description ALTER EXCEPTION changes the text of an exception error message.

An exception can be altered by its creator, the SYSDBA user, and any users with operating system root privileges.

Example This **isql** statement alters the message of an exception:

```
ALTER EXCEPTION CUSTOMER_CHECK 'Hold shipment for customer
remittance.';
```

See also ALTER PROCEDURE, ALTER TRIGGER, CREATE EXCEPTION, CREATE PROCEDURE, CREATE TRIGGER, DROP EXCEPTION

For more information on creating, raising, and handling exceptions, see the [Data Definition Guide](#).

ALTER INDEX

Activates or deactivates an index. Available in embedded SQL, DSQL, and **isql**, but not in the stored procedure or trigger language.

Syntax ALTER INDEX *name* {ACTIVE | INACTIVE};

Important In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	Description
<i>name</i>	Name of an existing index
ACTIVE	Changes an INACTIVE index to an ACTIVE one
INACTIVE	Changes an ACTIVE index to an INACTIVE one

Description ALTER INDEX makes an inactive index available for use, or disables the use of an active index. Deactivating an index is exactly like dropping it, except that the index definition remains in the database. Activating an index creates a new index structure.

Before inserting, updating, or deleting a large number of rows, deactivate a table's indexes to avoid altering the index incrementally. When finished, reactivate the index. A reasonable metric is that if you intend to add or delete more than 15% of the rows in a table, or update an indexed column in more than 10% of the rows, you should consider deactivating and reactivating the index.

If an index is in use, ALTER INDEX does not take effect until the index is no longer in use.

ALTER INDEX fails and returns an error if the index is defined for a UNIQUE, PRIMARY KEY, or FOREIGN KEY constraint. To alter such an index, use DROP INDEX to delete the index, then recreate it with CREATE INDEX.

An index can be altered by its creator, the SYSDBA user, and any users with operating system root privileges.

Note To add or drop index columns or keys, use DROP INDEX to delete the index, then recreate it with CREATE INDEX.

Example The following **isql** statements deactivate and reactivate an index to rebuild it:

```
ALTER INDEX BUDGETX INACTIVE;
ALTER INDEX BUDGETX ACTIVE;
```

See also ALTER TABLE, CREATE INDEX, DROP INDEX, SET STATISTICS

ALTER PROCEDURE

Changes the definition of an existing stored procedure. Available in DSQL and **isql** but not in the embedded language or in the stored procedures or triggers.

Syntax ALTER PROCEDURE *name*
 [(*param* datatype [, *param* datatype ...])]
 [RETURNS (*param* datatype [, *param* datatype ...])]
 AS *procedure_body*;

Argument	Description
<i>name</i>	Name of an existing procedure

Argument	Description
<i>param datatype</i>	Input parameters used by the procedure; legal datatypes are listed under CREATE PROCEDURE
RETURNS <i>param datatype</i>	Output parameters used by the procedure; legal datatypes are listed under CREATE PROCEDURE
<i>procedure_body</i>	<p>The procedure body includes:</p> <ul style="list-style-type: none"> • Local variable declarations • A block of statements in procedure and trigger language <p>See CREATE PROCEDURE for a complete description</p>

Description ALTER PROCEDURE changes an existing stored procedure without affecting its dependencies. It can modify a procedure's input parameters, output parameters, and body.

The complete procedure header and body must be included in the ALTER PROCEDURE statement. The syntax is exactly the same as CREATE PROCEDURE, except CREATE is replaced by ALTER.

Important Be careful about changing the type, number, and order of input and output parameters to a procedure, because existing application code may assume the procedure has its original format. Check for dependencies between procedures before changing parameters. Should you change parameters and find that another procedure can neither be altered to accept the new parameters or deleted, change the original procedure back to its original parameters, fix the calling procedure, then change the called procedure.

A procedure can be altered by its creator, the SYSDBA user, and any users with operating system root privileges. Procedures in use are not altered until they are no longer in use. ALTER PROCEDURE changes take effect when they are committed. Changes are then reflected in all applications that use the procedure without recompiling or relinking.

Example The following **isql** statements alter the GET_EMP_PROJ procedure, changing the return parameter to have a datatype of VARCHAR(20):

```
ALTER PROCEDURE GET_EMP_PROJ (EMP_NO SMALLINT)
RETURNS (PROJ_ID VARCHAR(20)) AS
BEGIN
    FOR SELECT PROJ_ID
    FROM EMPLOYEE_PROJECT
    WHERE EMP_NO = :emp_no
    INTO :proj_id
    DO
        SUSPEND;
END;
```

See also CREATE PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE

For more information on creating and using procedures, see the [Data Definition Guide](#).

For a complete description of the statements in procedure and trigger language, see [Chapter 3, “Procedures and Triggers.”](#)

ALTER TABLE

Changes a table by adding, dropping, or modifying columns or integrity constraints. Available in **gpre**, **DSQL**, and **isql**.

Important To alter a SQL global temporary table see: Altering a SQL global temporary table in the [Data Definition Guide](#).

Syntax `ALTER TABLE table operation [, operation ...];`

```

operation = ADD col_def
           | ADD tconstraint
           | ALTER [COLUMN] column_name alt_col_clause
           | DROP col
           | DROP CONSTRAINT constraint

alt_col_clause = TO new_col_name
                | TYPE new_col_datatype
                | POSITION new_col_position

col_def = col {datatype | COMPUTED [BY] (expr) | domain}
          [DEFAULT {literal | NULL | USER}]
          [NOT NULL]
          [col_constraint]
          [COLLATE collation]

datatype =
    {SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION}[array_dim]
    | {DATE | TIME | TIMESTAMP}[array_dim]
    | {DECIMAL | NUMERIC} [(precision [, scale])] [array_dim]
    | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(int)]
      [array_dim] [CHARACTER SET charname]
    | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
      [VARYING] [(int)] [array_dim]
    | BLOB [SUB_TYPE {int | subtype_name}] [SEGMENT SIZE int]
      [CHARACTER SET charname]
    | BLOB [(seglen [, subtype])] array_dim = [[x:]y [, [x:]y
...]]
    | BOOLEAN

expr = A valid SQL expression that results in a single
value.

col_constraint = [CONSTRAINT constraint]
    {
    | UNIQUE
    | PRIMARY KEY
  
```

```

| REFERENCES other_table [(other_col [, other_col ...])]
| [ON DELETE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
| [ON UPDATE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
| CHECK (search_condition)}

tconstraint = [CONSTRAINT constraint]
| {{PRIMARY KEY | UNIQUE} (col [, col ...])
| FOREIGN KEY (col [, col ...])
REFERENCES other_table [(other_col [, other_col ...])]
| [ON DELETE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
| [ON UPDATE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
| CHECK (search_condition)}

search_condition = val operator {val | (select_one)}
| val [NOT] BETWEEN val AND val
| val [NOT] LIKE val [ESCAPE val]
| val [NOT] IN (val [, val ...] | select_list)
| val IS [NOT] NULL
| val {>= | <=}
| val [NOT] {= | < | >}
| {ALL | SOME | ANY} (select_list)
| EXISTS (select_expr)
| SINGULAR (select_expr)
| val [NOT] CONTAINING val
| val [NOT] STARTING [WITH] val
| (search_condition)
| NOT search_condition
| search_condition OR search_condition
| search_condition AND search_condition

val = { col [array_dim] | :variable
| constant | expr | function
| udf ([val [, val ...]])
| NULL | USER | RDB$DB_KEY | ? }
[COLLATE collation]

constant = num | 'string' | charsetname 'string'

function = COUNT (*) | [ALL] val | DISTINCT val)
| SUM ([ALL] val | DISTINCT val)
| AVG ([ALL] val | DISTINCT val)
| MAX ([ALL] val | DISTINCT val)
| MIN ([ALL] val | DISTINCT val)
| CAST (val AS datatype)
| UPPER (val)
| GEN_ID (generator, val)

operator = {= | < | > | <= | >= | !< | !> | <> | !=}
```

select_one = SELECT on a single column; returns exactly one value.

select_list = SELECT on a single column; returns zero or more values.

select_expr = SELECT on a list of values; returns zero or more values.

Important In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

Notes on ALTER TABLE syntax

- The column constraints for referential integrity were new in InterBase 5. See *constraint_def* in Table 2.5 and the Description for ALTER TABLE on page 2-22.
- You cannot specify a COLLATE clause for Blob columns.
- When declaring arrays, you must include the outermost brackets, shown below in bold. For example, the following statement creates a 5 by 5 two-dimensional array of strings, each of which is 6 characters long:

```
my_array = varchar(6) [5,5]
```

- Use the colon (:) to specify an array with a starting point other than 1. The following example creates an array of integers that begins at 20 and ends at 30:

```
my_array = integer[20:30]
```

- For the full syntax of *search_condition*, see CREATE TABLE.

Table 2.5 The ALTER TABLE statement

Argument	Description
<i>table</i>	Name of an existing table to modify
<i>operation</i>	Action to perform on the table. Valid options are: <ul style="list-style-type: none"> • ADD a new column or table constraint to a table • DROP an existing column or constraint from a table
<i>col_def</i>	Description of a new column to add <ul style="list-style-type: none"> • Must include a column name and <i>datatype</i> • Can also include default values, column constraints, and a specific collation order
<i>col</i>	Name of the column to add or drop; column name must be unique within the table
<i>datatype</i>	Datatype of the column; see “Datatypes” on page 2-6.

Table 2.5 The ALTER TABLE statement

Argument	Description
ALTER [COLUMN]	Modifies column names, datatypes, and positions. Can also be used with ENCRYPT and DECRYPT options to encrypt and decrypt a column. For more information about encrypting databases and columns, see Chapter 13, “Encrypting Your Data” in the Data Definition Guide .
COMPUTED [BY] <i>expr</i>	<p>Specifies that the value of the column's data is calculated from <i>expr</i> at runtime and is therefore not allocated storage space in the database</p> <ul style="list-style-type: none"> • <i>expr</i> can be any arithmetic expression valid for the datatypes in the expression • Any columns referenced in <i>expr</i> must exist before they can be used in <i>expr</i> • <i>expr</i> cannot reference Blob columns • <i>expr</i> must return a single value, and cannot return an array
<i>domain</i>	Name of an existing domain
DEFAULT	<p>Specifies a default value for column data; this value is entered when no other entry is made; possible values are:</p> <ul style="list-style-type: none"> • <i>literal</i>: Inserts a specified string, numeric value, or date value • NULL: Enters a NULL value; this is the default DEFAULT • USER: Enters the user name of the current user; column must be of compatible text type to use the default <p>Defaults set at column level override defaults set at the domain level</p>
CONSTRAINT <i>constraint</i>	Name of a column or table constraint; the constraint name must be unique within the table
<i>constraint_def</i>	Specifies the kind of column constraint; valid options are UNIQUE, PRIMARY KEY, CHECK, and REFERENCES
CHECK <i>search_condition</i>	An attempt to enter a new value in the column fails if the value does not meet the <i>search_condition</i>
REFERENCES	Specifies that the column values are derived from column values in another table; if you do not specify column names, InterBase looks for a column with the same name as the referencing column in the referenced table

Table 2.5 The ALTER TABLE statement

Argument	Description
ON DELETE ON UPDATE	Used with REFERENCES: Changes a foreign key whenever the referenced primary key changes; valid options are: <ul style="list-style-type: none"> • [Default] NO ACTION: Does not change the foreign key; may cause the primary key update to fail due to referential integrity checks • CASCADE: For ON DELETE, deletes the corresponding foreign key; for ON UPDATE, updates the corresponding foreign key to the new value of the primary key • SET NULL: Sets all the columns of the corresponding foreign key to NULL • SET DEFAULT: Sets every column of the corresponding foreign key is set to its default value in effect when the referential integrity constraint is defined; when the default for a foreign column changes after the referential integrity constraint is defined, the change does not have an effect on the default value used in the referential integrity constraint
NOT NULL	Specifies that a column cannot contain a NULL value <ul style="list-style-type: none"> • If a table already has rows, a new column cannot be NOT NULL • NOT NULL is a column attribute only
DROP CONSTRAINT	Drops the specified table constraint
<i>table_constraint</i>	Description of the new table constraint; constraints can be PRIMARY KEY, UNIQUE, FOREIGN KEY, or CHECK
COLLATE <i>collation</i>	Establishes a default sorting behavior for the column; see Chapter 7, “Character Sets and Collation Orders” for more information

Description ALTER TABLE modifies the structure of an existing table. A single ALTER TABLE statement can perform multiple adds and drops.

- A table can be altered by its creator, the SYSDBA user, and any users with operating system superuser privileges.
- ALTER TABLE fails if the new data in a table violates a PRIMARY KEY or UNIQUE constraint definition added to the table. Dropping or altering a column fails if any of the following are true:
 - The column is part of a UNIQUE, PRIMARY, or FOREIGN KEY constraint
 - The column is used in a CHECK constraint
 - The column is used in the *value* expression of a computed column
 - The column is referenced by another database object such as a view

Important When a column is dropped, all data stored in it is lost.

Constraints

- Referential integrity constraints include optional ON UPDATE and ON DELETE clauses. They define the change to be made to the referencing column when the referenced column is updated or deleted. The values for these cascading referential integrity options are given in [Table 2.5, “The ALTER TABLE statement,” on page 20](#).
- To delete a column referenced by a computed column, you must drop the computed column before dropping the referenced column. To drop a column referenced in a FOREIGN KEY constraint, you must drop the constraint before dropping the referenced column. To drop a PRIMARY KEY or UNIQUE constraint on a column that is referenced by FOREIGN KEY constraints, drop the FOREIGN KEY constraint before dropping the PRIMARY KEY or UNIQUE key it references.
- You can create a FOREIGN KEY reference to a table that is owned by someone else only if that owner has explicitly granted you the REFERENCES privilege on that table using GRANT. Any user who updates your foreign key table must have REFERENCES or SELECT privileges on the referenced primary key table.
- You can add a check constraint to a column that is based on a domain, but be aware that changes to tables that contain CHECK constraints with subqueries may cause constraint violations.
- Naming column constraints is optional. If you do not specify a name, InterBase assigns a system-generated name. Assigning a descriptive name can make a constraint easier to find for changing or dropping, and more descriptive when its name appears in a constraint violation error message.
- When creating new columns in tables with data, do not use the UNIQUE constraint. If you use the NOT NULL constraint on a table with data, you should also specify a default value.

Example The following **isql** statement adds a column to a table and drops a column:

```
ALTER TABLE COUNTRY
    ADD CAPITAL VARCHAR(25) ,
    DROP CURRENCY;
```

This statement results in the loss of all data in the dropped CURRENCY column.

The next **isql** statement changes the name of the LARGEST_CITY column to BIGGEST_CITY:

```
ALTER TABLE COUNTRY ALTER LARGEST_CITY TO BIGGEST_CITY;
```

See also ALTER DOMAIN, CREATE DOMAIN, CREATE TABLE

For more information about altering tables, see the [Embedded SQL Guide](#).

For detailed information on encryption and decryption, see the topics “Encrypting Data” (page 13-9) and “Decrypting Data” (page 13-11) in the [Data Definition Guide](#).

ALTER TRIGGER

Changes an existing trigger. Available in DSQL and **isql**.

Syntax ALTER TRIGGER *name*
 [ACTIVE | INACTIVE]
 [{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
 [POSITION *number*]
 [AS *trigger_body*];

Argument	Description
<i>name</i>	Name of an existing trigger
ACTIVE	[Default] Specifies that a trigger action takes effect when fired
INACTIVE	Specifies that a trigger action does <i>not</i> take effect
BEFORE	Specifies the trigger fires before the associated operation takes place
AFTER	Specifies the trigger fires after the associated operation takes place
DELETE INSERT UPDATE	Specifies the table operation that causes the trigger to fire
POSITION <i>number</i>	Specifies order of firing for triggers before the same action or after the same action <ul style="list-style-type: none"> <i>number</i> must be an integer between 0 and 32,767, inclusive Lower-number triggers fire first Triggers for a table need not be consecutive; triggers on the same action with the same position number fire in random order
<i>trigger_body</i>	Body of the trigger: a block of statements in procedure and trigger language <ul style="list-style-type: none"> See CREATE TRIGGER for a complete description

Description ALTER TRIGGER changes the definition of an existing trigger. If any of the arguments to ALTER TRIGGER are omitted, then they default to their current values, that is the value specified by CREATE TRIGGER, or the last ALTER TRIGGER.

ALTER TRIGGER can change:

- Header information only, including the trigger activation status, when it performs its actions, the event that fires the trigger, and the order in which the trigger fires compared to other triggers.
- Body information only, the trigger statements that follow the AS clause.
- Header and trigger body information. In this case, the new trigger definition replaces the old trigger definition.

A trigger can be altered by its creator, the SYSDBA user, and any users with operating system root privileges.

Note To alter a trigger defined automatically by a CHECK constraint on a table, use ALTER TABLE to change the constraint definition.

Examples The following statement modifies the trigger, SET_CUST_NO, to be inactive:

```
ALTER TRIGGER SET_CUST_NO INACTIVE;
```

The next statement modifies the trigger, SET_CUST_NO, to insert a row into the table, NEW_CUSTOMERS, for each new customer.

```
ALTER TRIGGER SET_CUST_NO FOR CUSTOMER
BEFORE INSERT AS
BEGIN
    NEW.CUST_NO = GEN_ID(CUST_NO_GEN, 1);
    INSERT INTO NEW_CUSTOMERS(NEW.CUST_NO, TODAY)
END ;
```

See also CREATE TRIGGER, DROP TRIGGER

For a complete description of the statements in procedure and trigger language, see [Chapter 3, “Procedures and Triggers.”](#)

For more information about triggers, see the [Data Definition Guide](#) .

ALTER USER

Change an existing user. Available in **DSQL** and **isql**.

Syntax ALTER USER *name* SET
 [PASSWORD *password*]
 [[NO] DEFAULT ROLE *name*]
 [[NO] SYSTEM USER NAME *name*]
 [[NO] GROUP NAME *name*]
 [[NO] UID *number*]
 [[NO] GID *number*]
 [[NO] DESCRIPTION *string*]
 [[NO] FIRST NAME *string*]
 [[NO] MIDDLE NAME *string*]
 [[NO] LAST NAME *string*]
 [ACTIVE]
 [INACTIVE];

Argument	Description
PASSWORD	Password of user
[NO] DEFAULT ROLE	Default role
[NO] SYSTEM USER NAME	System user name for target user
[NO] GROUP NAME	Group name for target user
[NO] UID	Target user ID
[NO] GID	Group ID for target user
[NO] DESCRIPTION	Description
[NO] FIRST NAME	First name for target user
[NO] MIDDLE NAME	Middle name for target user
[NO] LAST NAME	Last name for target user
ACTIVE	Default. After inactive, reinstates selected user.
INACTIVE	Prevents a user from logging into database.

Description Alter user changes the definition of an existing user. Only used with database under embedded user authentication.

Note When NO is specified, an argument to the option must not be supplied. NO sets the option to a NULL state.

Examples The following statement modifies the user, JDOE, to be inactive:

```
ALTER USER JDOE SET INACTIVE;
```

The next statement modifies the user, JDOE, to be active:

```
ALTER USER JDOE SET ACTIVE;
```

See also CREATE USER, DROP USER

For more information about embedded user authentication, see the [Operations Guide](#).

AVG()

Calculates the average of numeric values in a specified column or expression. Available in **gpre**, **DSQL**, and **isql**.

Syntax AVG ([ALL] *value* | DISTINCT *value*)

Argument	Description
ALL	Returns the average of all values
DISTINCT	Eliminates duplicate values before calculating the average
<i>value</i>	A column or expression that evaluates to a numeric datatype

Description AVG() is an aggregate function that returns the average of the values in a specified column or expression. Only numeric datatypes are allowed as input to AVG().

If a field value involved in a calculation is NULL or unknown, it is automatically excluded from the calculation. Automatic exclusion prevents averages from being skewed by meaningless data.

AVG() computes its value over a range of selected rows. If the number of rows returned by a SELECT is zero, AVG() returns a NULL value.

Examples The following embedded SQL statement returns the average of all rows in a table:

```
EXEC SQL
    SELECT AVG (BUDGET) FROM DEPARTMENT INTO :avg_budget;
```

The next embedded SQL statement demonstrates the use of SUM(), AVG(), MIN(), and MAX() over a subset of rows in a table:

```
EXEC SQL
    SELECT SUM (BUDGET), AVG (BUDGET), MIN (BUDGET), MAX
    (BUDGET)
    FROM DEPARTMENT
    WHERE HEAD_DEPT = :head_dept
    INTO :tot_budget, :avg_budget, :min_budget, :max_budget;
```

See also COUNT(), MAX(), MIN(), SUM()

BASED ON

Declares a host-language variable based on a column. Available in **gpre**.

Syntax BASED [ON] [*dbhandle*.]*table.col*[.SEGMENT] *variable*;

Argument	Description
<i>dbhandle</i>	Handle for the database in which a table resides in a multi-database program; <i>dbhandle</i> must be previously declared in a SET DATABASE statement
<i>table.col</i>	Name of table and name of column on which the variable is based
.SEGMENT	Bases the local variable size on the segment length of the Blob column during BLOB FETCH operations; use only when <i>table.col</i> refers to a column of BLOB datatype
<i>variable</i>	Name of the host-language variable that inherits the characteristics of a database column

Description BASED ON is a preprocessor directive that creates a host-language variable based on a column definition. The host variable inherits the attributes described for the column and any characteristics that make the variable type consistent with the programming language in use. For example, in C, BASED ON adds one byte to CHAR and VARCHAR variables to accommodate the NULL character terminator.

Use BASED ON in a program's variable declaration section.

Note BASED ON does not require the EXEC SQL keywords.

To declare a host-language variable large enough to hold a Blob segment during FETCH operations, use the SEGMENT option of the BASED ON clause. The variable's size is derived from the segment length of a Blob column. If the segment length for the Blob column is changed in the database, recompile the program to adjust the size of host variables created with BASED ON.

Examples The following embedded statements declare a host variable based on a column:

```
EXEC SQL
    BEGIN DECLARE SECTION
        BASED_ON EMPLOYEE.SALARY salary;
EXEC SQL
    END DECLARE SECTION;
```

See also BEGIN DECLARE SECTION, CREATE TABLE, END DECLARE SECTION

BEGIN DECLARE SECTION

Identifies the start of a host-language variable declaration section. Available in **gpre**.

Syntax BEGIN DECLARE SECTION;

- Description** BEGIN DECLARE SECTION is used in embedded SQL applications to identify the start of host-language variable declarations for variables that will be used in subsequent SQL statements. BEGIN DECLARE SECTION is also a preprocessor directive that instructs **gpre** to declare SQLCODE automatically for the applications programmer.
- Important** BEGIN DECLARE SECTION must always appear within a module's global variable declaration section.
- Example** The following embedded SQL statements declare a section and a host-language variable:
- ```
EXEC SQL
 BEGIN DECLARE SECTION;
 BASED ON EMPLOYEE.SALARY salary;
EXEC SQL
 END DECLARE SECTION;
```
- See also** BASED ON, END DECLARE SECTION

## CASE

---

The CASE function allows you to evaluate a column value on a row against multiple criteria, where each criterion might return a different value.

**Syntax** CASE <expression>  
     WHEN <expression> THEN <expression> | NULL  
     [ELSE <expression> | NULL]  
     [COALESCE <expression>]  
     [NULLIF <expression, expression, ...>]  
 END

**Description** The CASE expression is a conditional value expression that consists of a list of value expressions, each of which is associated with a conditional expression. A CASE expression evaluates to the first value expression in the list for which its associated conditional expression evaluates to TRUE. The CASE expression has simple and searched forms of syntax.

The COALESCE and NULLIF expressions are common, shorthand forms of use for the CASE expression involving the NULL state. A COALESCE expression consists of a list of value expressions. It evaluates to the first value expression in the list that evaluates to non-NULL. If none of the value expressions in the list evaluates to non-NULL then the COALESCE expression evaluates to NULL.

The NULLIF expression consists of a list of two value expressions. If the two expressions are unequal then the NULLIF expression evaluates to the first value expression in the list. Otherwise, it evaluates to NULL.

**Example** The following example demonstrates the use of CASE using the sample employee.ib database:

```
select emp.first_name || ' ' || emp.last_name as name,
case proj.proj_name
 when 'DigiPizza' then 'Digital Pizza'
```

```

 when 'AutoMap' then 'AutoMobile Map'
 when 'Translator upgrade' then 'Universal Language
Translator'
 else 'Other'
end as project
from employee emp inner join employee_project emp_proj on
emp.emp_no = emp_proj.emp_no
inner join project proj on emp_proj.proj_id = proj.proj_id

```

## CAST()

Converts a column from one datatype to another. Available in **gpre**, DSQL, and **isql**.

**Syntax** CAST (*value AS datatype*)

| Argument        | Description                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------------------|
| <i>val</i>      | A column, constant, or expression; in SQL, <i>val</i> can also be a host-language variable, function, or UDF |
| <i>datatype</i> | Datatype to which to convert                                                                                 |

**Description** CAST() allows mixing of numerics and characters in a single expression by converting *val* to a specified datatype.

Normally, only similar datatypes can be compared in search conditions. CAST() can be used in search conditions to translate one datatype into another for comparison purposes.

Datatypes can be converted as shown in the following table:

**Table 2.6** Compatible datatypes for CAST()

| From datatype class          | To datatype class                        |
|------------------------------|------------------------------------------|
| Numeric                      | character, varying character, numeric    |
| Character, varying character | numeric, date, time, timestamp           |
| Date                         | character, varying character, timestamp  |
| Time                         | character, varying character, timestamp  |
| Timestamp                    | character, varying character, date, time |
| Blob, arrays                 | —                                        |
| Boolean                      | character, varying character             |

An error results if a given datatype cannot be converted into the datatype specified in `CAST()`. For example, you will get a string conversion error if you attempt to cast from a numeric type which is unable to represent in a date type to a date (e.g. a numeric type attempting to represent "year 99/12/31"(December) or "year 32768/3/1"(March))

**Example** In the following `WHERE` clause, `CAST()` is used to translate a `CHARACTER` datatype, `INTERVIEW_DATE`, to a `DATE` datatype to compare against a `DATE` datatype, `HIRE_DATE`:

```
WHERE HIRE_DATE = CAST (INTERVIEW_DATE AS DATE) ;
```

To cast a `VARCHAR` datatype, you must specify the length of the string, for example:

```
UPDATE client SET charef = CAST (clientref AS VARCHAR(20));
```

**See also** `UPPER()`

## CLOSE

Closes an open cursor. Available in **gpre**.

**Syntax** `CLOSE cursor;`

| Argument      | Description            |
|---------------|------------------------|
| <i>cursor</i> | Name of an open cursor |

**Description** `CLOSE` terminates the specified cursor, releasing the rows in its active set and any associated system resources. A cursor is a one-way pointer into the ordered set of rows retrieved by the select expression in the `DECLARE CURSOR` statement. A cursor enables sequential access to retrieved rows in turn and update in place.

There are four related cursor statements:

| Stage | Statement                   | Purpose                                                                                                                                                 |
|-------|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | <code>DECLARE CURSOR</code> | Declares the cursor; the <code>SELECT</code> statement determines rows retrieved for the cursor                                                         |
| 2     | <code>OPEN</code>           | Retrieves the rows specified for retrieval with <code>DECLARE CURSOR</code> ; the resulting rows become the cursor's <i>active set</i>                  |
| 3     | <code>FETCH</code>          | Retrieves the current row from the active set, starting with the first row; subsequent <code>FETCH</code> statements advance the cursor through the set |
| 4     | <code>CLOSE</code>          | Closes the cursor and releases system resources                                                                                                         |

FETCH statements cannot be issued against a closed cursor. Until a cursor is closed and reopened, InterBase does not reevaluate values passed to the search conditions. Another user can commit changes to the database while a cursor is open, making the active set different the next time that cursor is reopened.

**Note** In addition to CLOSE, COMMIT and ROLLBACK automatically close all cursors in a transaction.

**Example** The following embedded SQL statement closes a cursor:

```
EXEC SQL
 CLOSE BC;
```

**See also** CLOSE (BLOB), COMMIT, DECLARE CURSOR, FETCH, OPEN, ROLLBACK

## CLOSE (BLOB)

Terminates a specified Blob cursor and releases associated system resources. Available in **gpre**.

**Syntax** CLOSE *blob\_cursor*;

| Argument           | Description                 |
|--------------------|-----------------------------|
| <i>blob_cursor</i> | Name of an open Blob cursor |

**Description** CLOSE closes an opened read or insert Blob cursor. Generally a Blob cursor should only be closed only after:

- Fetching all the Blob segments for BLOB READ operations.
- Inserting all the segments for BLOB INSERT operations.

**Example** The following embedded SQL statement closes a Blob cursor:

```
EXEC SQL
 CLOSE BC;
```

**See also** DECLARE CURSOR (BLOB), FETCH (BLOB), INSERT CURSOR (BLOB), OPEN (BLOB)

## COALESCE( )

The COALESCE function evaluates to the first value expression in a list that evaluates to non-NULL. If none of the value expressions in the list evaluates to non-NULL then the COALESCE expression evaluates to NULL.

**Syntax** COALESCE(<expression1>,<expression2>,...<expression\_n>)

**Description** The COALESCE and NULLIF expressions are common, shorthand forms of use for the CASE expression involving the NULL state. A COALESCE expression consists of a list of value expressions. It evaluates to the first value expression in the list that evaluates to non-NULL. If none of the value expressions in the list evaluates to non-NULL then the COALESCE expression evaluates to NULL.

**Example** The following example demonstrates the use of CASE using the sample employee.ib database:

```
select coalesce(department, head_dept, location) from department
```

## COMMIT

Makes a transaction's changes to the database permanent, and ends the transaction. Available in **gpre**, DSQL, and **isql**.

**Syntax** COMMIT [WORK] [TRANSACTION *name*] [RELEASE] [RETAIN [SNAPSHOT]];

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                | Description                                                                                              |
|-------------------------|----------------------------------------------------------------------------------------------------------|
| WORK                    | An optional word used for compatibility with other relational databases that require it                  |
| TRANSACTION <i>name</i> | Commits transaction <i>name</i> to database. Without this option, COMMIT affects the default transaction |
| RELEASE                 | Available for compatibility with earlier versions of InterBase                                           |
| RETAIN [SNAPSHOT]       | Commits changes and retains current transaction context                                                  |

**Description** COMMIT is used to end a transaction and:

- Write all updates to the database.
- Make the transaction's changes visible to subsequent SNAPSHOT transactions or READ COMMITTED transactions.
- Close open cursors, unless the RETAIN argument is used.

A transaction ending with COMMIT is considered a successful termination. Always use COMMIT or ROLLBACK to end the default transaction.

**Tip** After read-only transactions, which make no database changes, use COMMIT rather than ROLLBACK. The effect is the same, but the performance of subsequent transactions is better and the system resources used by them are reduced.

**Important** The RELEASE argument is only available for compatibility with previous versions of InterBase. To detach from a database use DISCONNECT.

**Examples** The following **isql** statement makes permanent the changes to the database made by the default transaction:

```
COMMIT;
```

The next embedded SQL statement commits a named transaction:

```
EXEC SQL
 COMMIT TR1;
```

The following embedded SQL statement uses COMMIT RETAIN to commit changes while maintaining the current transaction context:

```
EXEC SQL
 COMMIT RETAIN;
```

**See also** DISCONNECT, ROLLBACK

For more information about handling transactions, see the [Embedded SQL Guide](#).

# CONNECT

Attaches to one or more databases. Available in **gpre**. A subset of CONNECT options is available in **isql**.

**Syntax** **isql form:**

```
CONNECT 'filespec' [USER 'username'] [PASSWORD 'password']
 [CACHE int] [ROLE 'rolename']
```

**SQL form:**

```
CONNECT [TO] {ALL | DEFAULT} config_opts
 | db_specs config_opts [, db_specs config_opts...];

<db_specs = dbhandle
 | {'filespec' | :variable} AS dbhandle

<config_opts = [USER {'username' | :variable}]
 [PASSWORD {'password' | :variable}]
 [ROLE {'rolename' | :variable}]
 [CACHE int [BUFFERS]]
```

| Argument           | Description                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------|
| {ALL   DEFAULT}    | Connects to all databases specified with SET DATABASE; options specified with CONNECT TO ALL affect all databases.        |
| <i>'filespec'</i>  | Database file name; can include path specification and node. The filespec must be in quotes if it includes spaces.        |
| <i>dbhandle</i>    | Database handle declared in a previous SET DATABASE statement; available in embedded SQL but not in <b>isql</b> .         |
| : <i>variable</i>  | Host-language variable specifying a database, user name, or password; available in embedded SQL but not in <b>isql</b> .  |
| AS <i>dbhandle</i> | Attaches to a database and assigns a previously declared handle to it; available in embedded SQL but not in <b>isql</b> . |

| Argument                          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| USER {'username'   :variable}     | String or host-language variable that specifies a user name for use when attaching to the database. The server checks the user name against the security database. User names are case insensitive on the server.                                                                                                                                                                                                                                                                                          |
| PASSWORD {'password'   :variable} | String or host-language variable, up to 8 characters in size, that specifies password for use when attaching to the database. The server checks the user name and password against the security database. Case sensitivity is retained for the comparison.                                                                                                                                                                                                                                                 |
| ROLE {'rolename'   :variable}     | String or host-language variable, up to 67 characters in size, which specifies the role that the user adopts on connection to the database. The user must have previously been granted membership in the role to gain the privileges of that role. Regardless of role memberships granted, the user has the privileges of a role at connect time only if a ROLE clause is specified in the connection. The user can adopt at most one role per connection, and cannot switch roles except by reconnecting. |
| CACHE int [BUFFERS]               | <p>Sets the number of cache buffers for a database, which determines the number of database pages a program can use at the same time. Values for <i>int</i>:</p> <ul style="list-style-type: none"> <li>• Default: 256</li> <li>• Maximum value: system-dependent</li> </ul> <p>Do not use the <i>filespec</i> form of database name with cache assignments.</p>                                                                                                                                           |

**Description** The CONNECT statement:

- Initializes database data structures.
- Determines if the database is on the originating node (a *local database*) or on another node (a *remote database*). An error message occurs if InterBase cannot locate the database.
- Optionally specifies one or more of a user name, password, or role for use when attaching to the database. PC clients must always send a valid user name and password. InterBase recognizes only the first 8 characters of a password.

If an InterBase user has ISC\_USER and ISC\_PASSWORD environment variables set and the user defined by those variables is not in the InterBase security database (*admin.ib* by default), the user receives the following error when attempting to view users from the local server manager connection: “undefined user name and password.” This applies only to the local connection; the automatic connection made through Server Manager bypasses user security.

- Attaches to the database and verifies the header page. The database file must contain a valid database, and the on-disk structure (ODS) version number of the database must be the one recognized by the installed version of InterBase on the server, or InterBase returns an error.
- Optionally establishes a database handle declared in a SET DATABASE statement.
- Specifies a cache buffer for the process attaching to a database.

In SQL programs before a database can be opened with CONNECT, it must be declared with the SET DATABASE statement. **isql** does not use SET DATABASE.

In SQL programs while the same CONNECT statement can open more than one database, use separate statements to keep code easy to read.

When CONNECT attaches to a database, it uses the default character set (NONE), or one specified in a previous SET NAMES statement.

In SQL programs the CACHE option changes the database cache size count (the total number of available buffers) from the default of 75. This option can be used to:

- Sets a new default size for all databases listed in the CONNECT statement that do not already have a specific cache size.
- Specifies a cache for a program that uses a single database.
- Changes the cache for one database without changing the default for all databases used by the program.

The size of the cache persists as long as the attachment is active. If a database is already attached through a multi-client server, an increase in cache size due to a new attachment persists until all the attachments end. A decrease in cache size does not affect databases that are already attached through a server.

A subset of CONNECT features is available in **isql**: database file name, USER, and PASSWORD. **isql** can only be connected to one database at a time. Each time CONNECT is used to attach to a database, previous attachments are disconnected.

**Examples** The following statement opens a database for use in **isql**. It uses all the CONNECT options available to **isql**:

```
CONNECT 'employee.ib' USER 'ACCT_REC' PASSWORD 'peanuts';
```

The next statements, from an embedded application, attach to a database file stored in the host-language variable and assign a previously declared database handle to it:

```
EXEC SQL
 SET DATABASE DB1 = 'employee.ib';
EXEC SQL
 CONNECT :db_file AS DB1;
```

The following embedded SQL statement attaches to *employee.ib* and allocates 150 cache buffers:

```
EXEC SQL
 CONNECT 'accounts.ib' CACHE 150;
```

The next embedded SQL statement connects the user to all databases specified with previous SET DATABASE statements:

```
EXEC SQL
 CONNECT ALL USER 'ACCT_REC' PASSWORD 'peanuts'
 CACHE 50;
```

The following embedded SQL statement attaches to the database, *employee.ib*, with 80 buffers and database *employee2.ib* with the default of 75 buffers:

```
EXEC SQL
 CONNECT 'employee.ib' CACHE 80, 'employee2.ib';
```

The next embedded SQL statement attaches to all databases and allocates 50 buffers:

```
EXEC SQL
 CONNECT ALL CACHE 50;
```

The following embedded SQL statement connects to EMP1 and v, setting the number of buffers for each to 80:

```
EXEC SQL
 CONNECT EMP1 CACHE 80, EMP2 CACHE 80;
```

The next embedded SQL statement connects to two databases identified by variable names, setting different user names and passwords for each:

```
EXEC SQL
 CONNECT
 :orderdb AS DB1 USER 'ACCT_REC' PASSWORD 'peanuts',
 :salesdb AS DB2 USER 'ACCT_PAY' PASSWORD 'payout';
```

**See also** DISCONNECT, SET DATABASE, SET NAMES

See the [Data Definition Guide](#) for more information about cache buffers and the [Operations Guide](#) for more information about database security and **isql**.

## COUNT( )

---

Calculates the number of rows that satisfy a query's search condition. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** COUNT ( \* | [ALL] *value* | DISTINCT *value*)

| Argument   | Description                                                              |
|------------|--------------------------------------------------------------------------|
| <i>*</i>   | Retrieves the number of rows in a specified table, including NULL values |
| ALL        | Counts all non-NULL values in a column                                   |
| DISTINCT   | Returns the number of unique, non-NULL values for the column             |
| <i>val</i> | A column or expression                                                   |

**Description** COUNT() is an aggregate function that returns the number of rows that satisfy a query's search condition. It can be used in views and joins as well as in tables.

**Example** The following embedded SQL statement returns the number of unique currency values it encounters in the COUNTRY table:

```
EXEC SQL
 SELECT COUNT (DISTINCT CURRENCY) INTO :cnt FROM COUNTRY;
```

**See also** AVG(), MAX(), MIN() SUM()

## CREATE DATABASE

Creates a new database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** CREATE {DATABASE | SCHEMA} '*filespec*'  
 [USER '*username*' [PASSWORD '*password*']]  
 [PAGE\_SIZE [=] *int*]  
 [LENGTH [=] *int* [PAGE[S]]]  
 [WITH ADMIN OPTION]  
 [DEFAULT CHARACTER SET *charset*]  
 [*secondary\_file*];

*secondary\_file* = FILE '*filespec*' [*fileinfo*] [*secondary\_file*]  
*fileinfo* = [LENGTH [=] *int* [PAGE[S]] | STARTING [AT [PAGE]]  
*int* ]  
 [*fileinfo*]

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                                   | Description                                                                                                                                                                                                                                                                                                                 |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>'filespec'</i>                          | <ul style="list-style-type: none"> <li>• A new database file specification</li> <li>• File naming conventions are platform-specific</li> </ul>                                                                                                                                                                              |
| USER <i>'username'</i>                     | <ul style="list-style-type: none"> <li>• Checks the <i>username</i> against valid user name and password combinations in the security database on the server where the database will reside</li> <li>• Windows client applications must provide a user name when attaching to a server</li> </ul>                           |
| PASSWORD<br><i>'password'</i>              | <ul style="list-style-type: none"> <li>• Checks the <i>password</i> against valid user name and password combinations in the security database on the server where the database will reside; can be up to 8 characters</li> <li>• Windows client applications must provide a password when attaching to a server</li> </ul> |
| PAGE_SIZE [=] <i>int</i>                   | <ul style="list-style-type: none"> <li>• Size, in bytes, for database pages</li> <li>• <i>int</i> can be 1024 (default), 2048, 4096, 8129, or 16384</li> </ul>                                                                                                                                                              |
| WITH ADMIN<br>OPTION                       | <ul style="list-style-type: none"> <li>• Create new database with embedded user authentication enabled.</li> </ul>                                                                                                                                                                                                          |
| DEFAULT<br>CHARACTER SET<br><i>charset</i> | <ul style="list-style-type: none"> <li>• Sets default character set for a database</li> <li>• <i>charset</i> is the name of a character set; if omitted, character set defaults to NONE</li> </ul>                                                                                                                          |
| FILE <i>'filespec'</i>                     | <ul style="list-style-type: none"> <li>• Names one or more secondary files to hold database pages after the primary file is filled</li> <li>• For databases created on remote servers, secondary file specifications cannot include a node name.</li> </ul>                                                                 |
| STARTING [AT [PAGE]]<br><i>int</i>         | Specifies the starting page number for a secondary file.                                                                                                                                                                                                                                                                    |
| LENGTH [=]<br><i>int</i> [PAGE[S]]         | <ul style="list-style-type: none"> <li>• Specifies the length of a primary or secondary database file</li> <li>• Use for primary file only if defining a secondary file in the same statement</li> </ul>                                                                                                                    |

**Description** CREATE DATABASE creates a new, empty database and establishes the following characteristics for it:

- The name of the primary file that identifies the database for users.

By default, databases are contained in single files.

- The name of any secondary files in which the database is stored.

A database can reside in more than one disk file if additional file names are specified as secondary files. If a database is created on a remote server, secondary file specifications cannot include a node name.

- The size of database pages.

Increasing page size can improve performance for the following reasons:

- Indexes work faster because the depth of the index is kept to a minimum.
- Keeping large rows on a single page is more efficient.
- Blob data is stored and retrieved more efficiently when it fits on a single page.

If most transactions involve only a few rows of data, a smaller page size might be appropriate, since less data needs to be passed back and forth and less memory is used by the disk cache.

- The number of pages in each database file.
- The dialect of the database.

The initial dialect of the database is the dialect of the client that creates it. For example, if you are using **isql**, either start it with the **-sql\_dialect *n*** switch or issue the **SET SQL DIALECT *n*** command before issuing the **CREATE DATABASE** command. Typically, you would create all databases in dialect 3. Dialect 1 exists to ease the migration of legacy databases.

To change the dialect of a database, use **gfix** or the Properties dialog in IBConsole. See the Migration appendix in the InterBase *Operations Guide* for information about migrating databases.

- The character set used by the database.

For a list of the character sets recognized by InterBase, see [Chapter 7, “Character Sets and Collation Orders.”](#)

Choice of **DEFAULT CHARACTER SET** limits possible collation orders to a subset of all available collation orders. Given a specific character set, a specific collation order can be specified when data is selected, inserted, or updated in a column.

If you do not specify a default character set, the character set defaults to **NONE**. Using character set **NONE** means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it. You can load any character set into a column defined with **NONE**, but you cannot load that same data into another column that has been defined with a different character set. In that case, no transliteration is performed between the source and destination character sets, and transliteration errors may occur during assignment.

- System tables that describe the structure of the database.

After creating the database, you define its tables, views, indexes, and system views as well as any triggers, generators, stored procedures, and UDFs that you need.

**Important** In DSQL, you must execute **CREATE DATABASE EXECUTE IMMEDIATE**. The database handle and transaction name, if present, must be initialized to zero prior to use.

**Read-only databases**

Databases are always created in *read-write mode*. You can change a table to *read-only mode* in one of two ways: you can specify **mode-read\_only** when you restore a backup, or you can use **gfix -mode read\_only** to change the mode of a table to read-only. See Chapter 6 in the [Operations Guide](#) for more information on database configuration and maintenance.

**About file sizes**

InterBase dynamically expands the last file in a database as needed. The maximum file size is system-dependent. This applies to single-file databases as well as to the last file of multifile databases. You should be aware that specifying a LENGTH for such files has no effect.

The total file size is the product of the number of database pages times the page size. The default page size is 4KB and the maximum page size is 16KB. However, InterBase files are small at creation time and increase in size as needed. The product of number of pages times page size represents a potential maximum size, not the size at creation.

**Examples** The following **isql** statement creates a database in the current directory using **isql**:

```
CREATE DATABASE 'employee.ib';
```

The next embedded SQL statement creates a database with a page size of 2048 bytes rather than the default of 4096:

```
EXEC SQL
 CREATE DATABASE 'employee.ib' PAGE_SIZE 2048;
```

The following embedded SQL statement creates a database stored in two files and specifies its default character set:

```
EXEC SQL
 CREATE DATABASE 'employee.ib'
 DEFAULT CHARACTER SET ISO8859_1
 FILE 'employee2.ib' STARTING AT PAGE 10001;
```

**See also** ALTER DATABASE, DROP DATABASE

See the [Data Definition Guide](#) for more information about secondary files, character set specification, and collation order; see the [Operations Guide](#) for more information about page size.

## CREATE DOMAIN

Creates a column definition that is global to the database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** CREATE DOMAIN *domain* [AS] datatype  
 [DEFAULT {*literal* | NULL | USER}]  
 [NOT NULL] [CHECK (*dom\_search\_condition*)]  
 [COLLATE *collation*];

```

datatype> =
 {SMALLINT|INTEGER|FLOAT|DOUBLE PRECISION}[array_dim]
 | {DATE|TIME|TIMESTAMP}[array_dim]
 | {DECIMAL | NUMERIC} [(precision [, scale])] [array_dim]
 | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(int)]
 [array_dim] [CHARACTER SET charname]
 | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
 [VARYING] [(int)] [array_dim]
 | BLOB [SUB_TYPE {int | subtype_name}] [SEGMENT SIZE int]
 [CHARACTER SET charname]
 | BLOB [(seglen [, subtype])]
 | BOOLEAN

array_dim> = [[x:]y [, [x:]y ...]]

dom_search_condition> =
 VALUE operator value
 | VALUE [NOT] BETWEEN value AND value
 | VALUE [NOT] LIKE value [ESCAPE value]
 | VALUE [NOT] IN (value [, value ...])
 | VALUE IS [NOT] NULL
 | VALUE [NOT] CONTAINING value
 | VALUE [NOT] STARTING [WITH] value
 | (dom_search_condition)
 | NOT dom_search_condition
 | dom_search_condition OR dom_search_condition
 | dom_search_condition AND dom_search_condition

operator> = {= | < | > | <= | >= | != | !< | !> | <> | !=}

```

**Note on the CREATE DOMAIN syntax**

- COLLATE is useful only for text data, not for numeric types. Also, you cannot specify a COLLATE clause for Blob columns.
- When declaring arrays, you must include the outermost brackets, shown below in bold. For example, the following statement creates a 5 by 5 two-dimensional array of strings, each of which is six characters long:

```
my_array = varchar(6) [5,5]
```

- Use the colon (:) to specify an array with a starting point other than 1. The following example creates an array of integers that begins at 20 and ends at 30:

```
my_array = integer[20:30]
```

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                              | Description                                                                                                                                                                                                                                                                                                                 |
|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>domain</i>                         | Unique name for the domain                                                                                                                                                                                                                                                                                                  |
| <i>datatype</i>                       | SQL datatype                                                                                                                                                                                                                                                                                                                |
| DEFAULT                               | Specifies a default column value that is entered when no other entry is made; possible values are:<br><i>literal</i> —Inserts a specified string, numeric value, or date value<br>NULL—Enters a NULL value<br>USER—Enters the user name of the current user; column must be of compatible character type to use the default |
| NOT NULL                              | Specifies that the values entered in a column cannot be NULL                                                                                                                                                                                                                                                                |
| CHECK ( <i>dom_search_condition</i> ) | Creates a single CHECK constraint for the domain                                                                                                                                                                                                                                                                            |
| VALUE                                 | Placeholder for the name of a column eventually based on the domain                                                                                                                                                                                                                                                         |
| COLLATE <i>collation</i>              | Specifies a collation sequence for the domain                                                                                                                                                                                                                                                                               |

**Description** CREATE DOMAIN builds an inheritable column definition that acts as a template for columns defined with CREATE TABLE or ALTER TABLE. The domain definition contains a set of characteristics, which include:

- Datatype
- An optional default value
- Optional disallowing of NULL values
- An optional CHECK constraint
- An optional collation clause

The CHECK constraint in a domain definition sets a *dom\_search\_condition* that must be true for data entered into columns based on the domain. The CHECK constraint cannot reference any domain or column.

**Note** Be careful not to create a domain with contradictory constraints, such as declaring a domain NOT NULL and assigning it a DEFAULT value of NULL.

The datatype specification for a CHAR or VARCHAR text domain definition can include a CHARACTER SET clause to specify a character set for the domain. Otherwise, the domain uses the default database character set. For a complete list of character sets recognized by InterBase, see [Chapter 7, “Character Sets and Collation Orders.”](#)

If you do not specify a default character set, the character set defaults to NONE. Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it. You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a different character set. In these cases, no transliteration is performed between the source and destination character sets, so errors can occur during assignment.

The COLLATE clause enables specification of a particular collation order for CHAR, VARCHAR, and NCHAR text datatypes. Choice of collation order is restricted to those supported for the domain's given character set, which is either the default character set for the entire database, or a different set defined in the CHARACTER SET clause as part of the datatype definition. For a complete list of collation orders recognized by InterBase, see [Chapter 7, “Character Sets and Collation Orders.”](#)

Columns based on a domain definition inherit all characteristics of the domain. The domain default, collation clause, and NOT NULL setting can be overridden when defining a column based on a domain. A column based on a domain can add additional CHECK constraints to the domain CHECK constraint.

**Examples** The following **isql** statement creates a domain that must have a positive value greater than 1,000, with a default value of 9,999. The keyword VALUE substitutes for the name of a column based on this domain.

```
CREATE DOMAIN CUSTNO
 AS INTEGER
 DEFAULT 9999
 CHECK (VALUE > 1000);
```

The next **isql** statement limits the values entered in the domain to four specific values:

```
CREATE DOMAIN PRODTYPE
 AS VARCHAR(12)
 CHECK (VALUE IN ('software', 'hardware', 'other', 'N/A'));
```

The following **isql** statement creates a domain that defines an array of CHAR datatype:

```
CREATE DOMAIN DEPTARRAY AS CHAR(67) [4:5];
```

In the following **isql** example, the first statement creates a domain with USER as the default. The next statement creates a table that includes a column, ENTERED\_BY, based on the USERNAME domain.

```
CREATE DOMAIN USERNAME AS VARCHAR(20)
 DEFAULT USER;

CREATE TABLE ORDERS (ORDER_DATE DATE, ENTERED_BY USERNAME,
 ORDER_AMT DECIMAL(8,2));

INSERT INTO ORDERS (ORDER_DATE, ORDER_AMT)
 VALUES ('1-MAY-93', 512.36);
```

The INSERT statement does not include a value for the ENTERED\_BY column, so InterBase automatically inserts the user name of the current user, JSMITH:

```
SELECT * FROM ORDERS;

1-MAY-93 JSMITH 512.36
```

The next **isql** statement creates a BLOB domain with a TEXT subtype that has an assigned character set:

```
CREATE DOMAIN DESCRIPT AS
 BLOB SUB_TYPE TEXT SEGMENT SIZE 80
 CHARACTER SET SJIS;
```

**See also** ALTER DOMAIN, ALTER TABLE, CREATE TABLE, DROP DOMAIN

For more information about character set specification and collation orders, see the [Data Definition Guide](#).

## CREATE ENCRYPTION

Creates encryption keys for use during the encryption process.

**Syntax** CREATE ENCRYPTION *key-name* for AES | for DES

| Argument        | Description                                                                                                                                                                                                                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Key-name</i> | Name associated with the encryption key. Name must be unique.                                                                                                                                                                                                                                                   |
| For AES DES     | Indicates the level of encryption InterBase will apply to the encrypted data. Advanced Encryption Standard (AES) is considered a strong encryption scheme and requires a license to use with InterBase. Data Encryption Standard (DES) is considered a weak encryption scheme that requires no special license. |

**Description** CREATE ENCRYPTION creates an encryption key. Only a SYSDSO (Data Security Owner) can create an encryption key. An encryption key is used to encrypt a database's pages and/or the database's columns. The database owner uses an encryption key to perform encryption on a specific database or column. InterBase stores encryption keys in the RDB\$ENCRYPTIONS system table.

Three new columns have been added to the RDB\$RELATIONS\_FIELDS table: RDB\$ENCRYPTION\_ID, RDB\$DECRYPT\_DEFAULT\_VALUE and RDB\$DECRYPT\_DEFAULT\_SOURCE to support the database page and column-level encryption as well.

**Example** The following isql statement creates an encryption key called revenue\_key using the AES encryption scheme and a length of 192 bits:

```
CREATE ENCRYPTION revenue_key FOR AES WITH LENGTH 192 BITS
```

**See also** DROP ENCRYPTION, GRANT, REVOKE, ALTER DATABASE, ALTER TABLE.

For detailed information on encryption and decryption, see the topics “Encrypting Data” (page 13-9) and “Decrypting Data” (page 13-11) in the [Data Definition Guide](#).

## CREATE EXCEPTION

---

Creates a user-defined error and associated message for use in stored procedures and triggers. Available in DSQL and **isql**.

**Syntax** `CREATE EXCEPTION name 'message';`

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument           | Description                                                                                       |
|--------------------|---------------------------------------------------------------------------------------------------|
| <i>name</i>        | Name associated with the exception message; must be unique among exception names in the database  |
| ' <i>message</i> ' | Quoted string containing alphanumeric characters and punctuation; maximum length = 78 characters. |

**Description** `CREATE EXCEPTION` creates an exception, a user-defined error with an associated message. Exceptions may be raised in triggers and stored procedures.

Exceptions are global to the database. The same message or set of messages is available to all stored procedures and triggers in an application. For example, a database can have English and French versions of the same exception messages and use the appropriate set as needed.

When raised by a trigger or a stored procedure, an exception:

- Terminates the trigger or procedure in which it was raised and undoes any actions performed (directly or indirectly) by it.
- Returns an error message to the calling application. In **isql**, the error message appears on the screen, unless output is redirected.

Exceptions may be trapped and handled with a `WHEN` statement in a stored procedure or trigger.

**Examples** This **isql** statement creates the exception, `UNKNOWN_EMP_ID`:

```
CREATE EXCEPTION UNKNOWN_EMP_ID 'Invalid employee number or
project id.';
```

The following statement from a stored procedure raises the previously created exception when `SQLCODE -530` is set, which is a violation of a `FOREIGN KEY` constraint:

```
. . .
```

```
WHEN SQLCODE -530 DO
 EXCEPTION UNKNOWN_EMP_ID;
. . .
```

**See also** ALTER EXCEPTION, ALTER PROCEDURE, ALTER TRIGGER, CREATE PROCEDURE, CREATE TRIGGER, DROP EXCEPTION

For more information on EXECUTE STATEMENTS, see [Chapter 3: “Procedures and Triggers”](#)

For more information on creating, raising, and handling exceptions, see the [Data Definition Guide](#).

## CREATE GENERATOR

---

Declares a generator to the database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** CREATE GENERATOR *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description            |
|-------------|------------------------|
| <i>name</i> | Name for the generator |

**Description** CREATE GENERATOR declares a generator to the database and sets its starting value to zero. A generator is a sequential number that can be automatically inserted in a column with the GEN\_ID() function. A generator is often used to ensure a unique value in a PRIMARY KEY, such as an invoice number, that must uniquely identify the associated row.

A database can contain any number of generators. Generators are global to the database, and can be used and updated in any transaction. InterBase does not assign duplicate generator values across transactions.

You can use SET GENERATOR to set or change the value of an existing generator when writing triggers, procedures, or SQL statements that call GEN\_ID().

**See also** GEN\_ID(), SET GENERATOR

## CREATE INDEX

---

Creates an index on one or more columns in a table. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]] INDEX *index*  
ON *table* (*col* [, *col* ...]);

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument     | Description                                                              |
|--------------|--------------------------------------------------------------------------|
| UNIQUE       | Prevents insertion or updating of duplicate values into indexed columns  |
| ASC[ENDING]  | Sorts columns in ascending order, the default order if none is specified |
| DESC[ENDING] | Sorts columns in descending order                                        |
| <i>index</i> | Unique name for the index                                                |
| <i>table</i> | Name of the table on which the index is defined                          |
| <i>col</i>   | Column in <i>table</i> to index                                          |

**Description** Creates an index on one or more columns in a table. Use CREATE INDEX to improve speed of data access. Using an index for columns that appear in a WHERE clause may improve search performance.

**Important** You cannot index Blob columns or arrays.

A UNIQUE index cannot be created on a column or set of columns that already contains duplicate or NULL values.

ASC and DESC specify the order in which an index is sorted. For faster response to queries that require sorted values, use the index order that matches the query's ORDER BY clause. Both an ASC and a DESC index can be created on the same column or set of columns to access data in different orders.

**Tip** To improve index performance, use SET STATISTICS to recompute index selectivity, or rebuild the index by making it inactive, then active with sequential calls to ALTER INDEX.

**Examples** The following **isql** statement creates a unique index:

```
CREATE UNIQUE INDEX PRODTYPEX ON PROJECT (PRODUCT,
PROJ_NAME) ;
```

The next **isql** statement creates a descending index:

```
CREATE DESCENDING INDEX CHANGEX ON SALARY_HISTORY
(CHANGE_DATE) ;
```

The following **isql** statement creates a two-column index:

```
CREATE INDEX NAMEX ON EMPLOYEE (LAST_NAME, FIRST_NAME) ;
```

**See also** ALTER INDEX, DROP INDEX, SELECT, SET STATISTICS

## CREATE JOURNAL

---

Creates a journal file and activates journaling.

**Syntax** `CREATE JOURNAL [<journal-file-specification>] [LENGTH <number-of-pages>]  
 [CHECKPOINT LENGTH <number-of-pages> [PAGES]]  
 [CHECKPOINT INTERVAL <number-of-seconds> [SECONDS]]  
 [PAGE SIZE <number-of-bytes> [BYTES]]  
 [PAGE CACHE <number-of-buffers> [BUFFERS]]  
 [[NO] TIMESTAMP NAME]  
 [[NO] PREALLOCATE <number-of-pages> [PAGES]];`

| Argument                          | Description                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>journal-file-specification</i> | Specifies a quoted string containing the full path and base file name of the journal file. The base journal file name is used as a template for the actual journal file names as they are created. The default is the full database path and file name                                                                                                  |
| LENGTH                            | This clause specifies the number of pages that can be written to the journal file before rolling over to a new journal file. The maximum length is 2GB or 4000 pages.                                                                                                                                                                                   |
| CHECKPOINT LENGTH                 | This clause specifies the number of pages that can be written to the journal file before checkpoint occurs. The default is 500.                                                                                                                                                                                                                         |
| CHECKPOINT INTERVAL               | Determines the number of seconds between database checkpoints. The checkpoint interval determines how long it will take to recover after a server crash. The default is 0.<br><br>Note: If both CHECKPOINT LENGTH and CHECKPOINT INTERVAL are specified, whichever event occurs first will initiate a database checkpoint.                              |
| PAGE SIZE                         | Determines the size of a journal page in bytes. A journal page size must be at least twice the size of a database page size. If a journal page size of less is specified, it will be rounded up to twice the database page size and a warning will be returned. The journal page size need not be a power of 2. The default is twice the database size. |

---

| Argument               | Description                                                                                                                                                                                                                                                                                                   |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PAGE CACHE             | Determines the number of journal pages that are cached to memory. This number must be large enough to provide buffers for worker threads to write to when the cache writer is writing other buffers. If the number is too small, the worker threads wait and performance suffers. The default is 100 buffers. |
| [NO] TIMESTAMP<br>NAME | Determines whether or not to append the file creation timestamp to the base journal file name. The default is enabled.<br>If used, the base journal file name will be appended with a timestamp in the following format:<br><code>YYYY_MM_DDTHH_MM_SSZ.sequence_number.journal</code>                         |
| [NO] PREALLOCATE       | Determines journal file space requirements while simultaneously guaranteeing that the space is allocated in advance. The default is twice the database size.                                                                                                                                                  |

**Description** A journal consists of one or more journal files. A journal file records each database transaction as it occurs. To save changed journal pages in the database cache to the hard disk, you set up journaling checkpoints to occur automatically. A checkpoint specifies the time at which InterBase must save all the changed pages in the database cache to the database file.

The `CREATE JOURNAL` statement causes all subsequent write operations on a database to be done asynchronously. The journal file I/O is always synchronous and cannot be altered. All transaction changes are safely recorded on durable storage before the transaction is committed.

Journaling can be used with journal archiving to provide more complete disaster recovery.

**Example** In the following example:

```
CREATE JOURNAL 'e:\database\test'
LENGTH 4000
CHECKPOINT LENGTH 10000
PAGE CACHE 2500;
```

The `LENGTH` parameter of 65000 will cause rollover to a new journal file every 1GB (65000 x 16KB). A `CHECKPOINT LENGTH` parameter of 10000 means the database checkpoint will occur every 160MB (10000 x 16KB). The 2500 journal buffer configuration will leave 2000 spare buffers for the worker threads to dump their journal changes. At the built-in `PAGE CACHE` default of 100, the worker threads can stall due to a high rate of journal buffer wait states.

**See also** `DROP JOURNAL`, `CREATE JOURNAL ARCHIVE`, `DROP JOURNAL ARCHIVE`.

For more information about journals, journal files, and journal archives, see the [Operations Guide](#).

## CREATE JOURNAL ARCHIVE

---

Activities journal archiving and performs the initial database dump to the archive directory.

**Syntax** `CREATE JOURNAL ARCHIVE <journal archive directory>`

| Argument                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>journal archive directory</i> | The location in which InterBase stores the journal archive. If the directory does not exist or is not accessible, InterBase returns an error message. The directory path can be a local drive, a mapped drive, or an UNC path (as long as the underlying file APIs can open the file using that specification). If you do not specify a journal archive directory in the <code>CREATE JOURNAL ARCHIVE</code> statement, InterBase uses the journal directory created with the <code>CREATE JOURNAL</code> statement. |

**Description** The `CREATE JOURNAL ARCHIVE` command performs two functions: it activates journal archiving in an InterBase database, *and* it automatically performs the initial full, physical dump of the database. InterBase stores the dump in the journal archive directory you specify in the `CREATE` statement. A journal archive enables you to recover to the last committed transaction in the most recently archived and completed journal file.

**Important** `CREATE JOURNAL ARCHIVE` creates the archive and performs an initial dump. However, you must issue a specific `gbak` command to copy completed journal files to the journal archive. You use another `gbak` command to perform subsequent dumps to the archive. For information about the `gbak` archive commands, and about how to implement journaling and journal archiving, see the InterBase [Operations Guide](#).

**See also** `DROP JOURNAL ARCHIVE`, `CREATE JOURNAL`, `DROP JOURNAL`.

### Journal Archive Management

You can manage the Journal Archive feature of InterBase V8. The archive is a directory that holds journal files, which have been archived from the local journal directory associated with a database. In addition, to storing copies of the local journal files, the archive also stores database dumps that are periodically backed up to the archive.

**Description** Archived database dumps represent the starting point from which long-term database recovery is initiated. A set of archive journal files are applied to a copy of the archive database in the same way that local journal files are applied to a production database during short-term recovery. Also, an InterBase timestamp can be specified to indicate a point-in-time until which the journal files will be applied.

When the archive is used to recover a database, the resulting database is not a journaled database. This means that RDB\$LOG\_FILES, RDB\$JOURNAL\_FILES and the log page of the database are empty. This prevents the database from accidentally using the journal and journal archive of an existing database. Database recovery is usually used when the original database is corrupted or unavailable due to hardware failures. However, it could be possible to recover a database on the same machine as the working production database or on a different machine where the journal and journal archive directories have no similarly-named directories. Therefore, if journaling and/or journal archiving is desired for the recovered database, it is necessary to execute the appropriate DDL commands to do so.

**Examples** Gbak is used to archive databases and journal files to the archive, and is also used to recover a database from the archive back to a specified local directory of the user's choice.

To archive a database:

**gbak -archive\_database <dbname>**

To archive local journal files:

**gbak -archive\_journals <dbname>**

To recover a database (optionally to a point-in-time)

**gbak -archive\_recover [-until <timestamp>] <archive\_dbname> <local\_dbname>**

If the -until command line switch is not given, the database recover applies as many journal files as possible to recover a database to the most recent point-in-time. If possible, the database recovery attempts to "jump" from the archive to the local journal directory to apply the journal files that were never copied to the archive. In this way, a database may be recovered to the most recently committed transaction of the original database.

If allowed, the archive grows in storage size infinitely as the database and the most current journal files are continually archived. Gfix is used to manage and garbage collect archived items that are no longer required. As the number of journal files grows in the archive without have created more recent archived database dumps, so does the time that will be needed to recover the database from the archive. Therefore, it is desirable to periodically create additional database dumps in the archive. At some point, you may decide that older database dumps and the journal files on which they depend on are no longer necessary, as the basis of recovery will be on more recent database dumps and journal files.

All archive items are denoted by an archive sequence number that corresponds to the order in which the items were created in the archive.

To garbage collect archive items less than an archive sequence number.

**gfix -archive\_sweep [-force] <archive\_sequence\_no>**

If an archive item cannot be swept for some reason, the sweep stops and returns an error status. In some cases, this could stop the command from ever succeeding. For example, if an archive is manually deleted with a shell OS command, the sweep always fails because it can't find the file to drop. The -force option continues regardless of errors to delete as much as possible. The -force switch logs errors to the InterBase error log instead of returning an error status.

To specify how many database dumps to allow in the archive:

**gfix -archive\_dumps <number>**

Once the number of database dumps in the archive exceeds the <number> given, all lower sequenced archive items are deleted from the archive. Sometimes all lower sequenced items cannot be deleted. For example, a database dump may depend on a lower sequenced journal file with which to start recovery. In that case, InterBase automatically adjusts the given sequence number lower so that this dependency is not lost.

To track that state of the archive, a new system table, **RDB\$JOURNAL\_ARCHIVES**, has been added for ODS 12 databases. The Gbak and Gfix commands listed above used this system table to decide which archive items are targets for the commands.

**Important** Listed below are the requirements and constraints for managing the Journal Archive.

- 1 The archive is platform-specific. An archive created with InterBase for Windows cannot be directly used to recover on InterBase for Unix. Instead, an archived database dump could be logically backed up in transportable format and then logically restored on the other platform.
- 2 The journal and journal archive are restricted to a single directory. The number of items allowed to be archived will be limited to the number of files that are allowed in a directory for a give file system.
- 3 Only full database dumps are archived. In particular, it is not possible to archive incremental database dumps.
- 4 Journaling must be enabled for a database before the database can be configured for journal archiving.

## CREATE PROCEDURE

---

Creates a stored procedure, its input and output parameters, and its actions.  
Available in DSQL, and **isql**.

**Syntax** `CREATE PROCEDURE name`  
           `[(param datatype [, param datatype ...])]`  
           `[RETURNS param datatype [, param datatype ...])]`  
           `AS procedure_body;`

*procedure\_body* =  
     `[variable_declaration_list]`  
     `block`

*variable\_declaration\_list* =  
     `DECLARE VARIABLE var datatype;`  
     `[DECLARE VARIABLE var datatype; ...]`

*block* =  
   `BEGIN`  
     `compound_statement`  
     `[compound_statement ...]`  
   `END`

*compound\_statement* = `block | statement;`

*datatype* = {SMALLINT| INTEGER| FLOAT| DOUBLE PRECISION}  
           | {DECIMAL | NUMERIC} [(*precision* [, *scale*])]  
           | {DATE | TIME | TIMESTAMP}  
           | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}  
           | [(*int*)] [CHARACTER SET *charname*]  
           | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]  
           | [(*int*)]  
           | BOOLEAN

| Argument                                | Description                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name</i>                             | Name of the procedure. Must be unique among procedure, table, and view names in the database                                                                                                                                                                                                                                               |
| <i>param datatype</i>                   | Input parameters that the calling program uses to pass values to the procedure:<br><i>param</i> : Name of the input parameter, unique for variables in the procedure<br><i>datatype</i> : An InterBase datatype                                                                                                                            |
| RETURNS <i>param datatype</i>           | Output parameters that the procedure uses to return values to the calling program:<br><i>param</i> : Name of the output parameter, unique for variables within the procedure<br><i>datatype</i> : An InterBase datatype<br>The procedure returns the values of output parameters when it reaches a SUSPEND statement in the procedure body |
| AS                                      | Keyword that separates the procedure header and the procedure body                                                                                                                                                                                                                                                                         |
| DECLARE VARIABLE<br><i>var datatype</i> | Declares local variables used only in the procedure; must be preceded by DECLARE VARIABLE and followed by a semicolon (;).<br><i>var</i> is the name of the local variable, unique for variables in the procedure.                                                                                                                         |
| <i>statement</i>                        | Any single statement in InterBase procedure and trigger language; must be followed by a semicolon (;) except for BEGIN and END statements                                                                                                                                                                                                  |

**Description** CREATE PROCEDURE defines a new stored procedure to a database. A stored procedure is a self-contained program written in InterBase procedure and trigger language, and stored as part of a database's metadata. Stored procedures can receive input parameters from and return values to applications.

InterBase procedure and trigger language includes all SQL data manipulation statements and some powerful extensions, including IF ... THEN ... ELSE, WHILE ... DO, FOR SELECT ... DO, exceptions, and error handling.

There are two types of procedures:

- *Select* procedures that an application can use in place of a table or view in a SELECT statement. A select procedure must be defined to return one or more values, or an error will result.
- *Executable* procedures that an application can call directly, with the EXECUTE PROCEDURE statement. An executable procedure need not return values to the calling program.

A stored procedure is composed of a *header* and a *body*.

The procedure header contains:

- The *name* of the stored procedure, which must be unique among procedure and table names in the database.
- An optional list of *input parameters* and their datatypes that a procedure receives from the calling program.
- RETURNS followed by a list of *output parameters* and their datatypes if the procedure returns values to the calling program.

The procedure body contains:

- An optional list of *local variables* and their datatypes.
- A *block* of statements in InterBase procedure and trigger language, bracketed by BEGIN and END. A block can itself include other blocks, so that there may be many levels of nesting.

InterBase does not allow database changes that affect the behavior of an existing stored procedure (for example, DROP TABLE or DROP EXCEPTION). To see all procedures defined for the current database or the text and parameters of a named procedure, use the **isql** internal commands SHOW PROCEDURES or SHOW PROCEDURE *procedure*.

InterBase procedure and trigger language is a complete programming language for stored procedures and triggers. It includes:

- SQL data manipulation statements: INSERT, UPDATE, DELETE, and singleton SELECT.
- SQL operators and expressions, including generators and UDFs that are linked with the database.
- Extensions to SQL, including assignment statements, control-flow statements, context variables (for triggers), event-posting statements, exceptions, and error-handling statements.

The following table summarizes language extensions for stored procedures. For a complete description of each statement, see [Chapter 3, “Procedures and Triggers.”](#)

**Table 2.7** Language extensions for stored procedures

| Statement                                                                                                                     | Description                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BEGIN ... END                                                                                                                 | Defines a block of statements that executes as one <ul style="list-style-type: none"> <li>• The BEGIN keyword starts the block; the END keyword terminates it</li> <li>• Neither should end with a semicolon</li> </ul>                                                                                    |
| <i>variable</i> = <i>expression</i>                                                                                           | Assignment statement: assigns the value of <i>expression</i> to <i>variable</i> , a local variable, input parameter, or output parameter                                                                                                                                                                   |
| <i>/* comment_text */</i>                                                                                                     | Programmer's comment, where <i>comment_text</i> can be any number of lines of text                                                                                                                                                                                                                         |
| EXCEPTION <i>exception_name</i>                                                                                               | Raises the named exception: an exception is a user-defined error that returns an error message to the calling application unless handled by a WHEN statement                                                                                                                                               |
| EXECUTE PROCEDURE<br><i>proc_name</i> [ <i>var</i> [, <i>var</i> ...]]<br>[RETURNING_VALUES<br><i>var</i> [, <i>var</i> ...]] | Executes stored procedure, <i>proc_name</i> , with the listed input arguments, returning values in the listed output arguments following RETURNING_VALUES; input and output arguments must be local variables                                                                                              |
| EXIT                                                                                                                          | Jumps to the final END statement in the procedure                                                                                                                                                                                                                                                          |
| FOR <i>select_statement</i><br>DO <i>compound_statement</i>                                                                   | Repeats the statement or block following DO for every qualifying row retrieved by <i>select_statement</i><br><i>select_statement</i> is like a normal SELECT statement                                                                                                                                     |
| <i>compound_statement</i>                                                                                                     | Either a single statement in procedure and trigger language or a block of statements bracketed by BEGIN and END                                                                                                                                                                                            |
| IF ( <i>condition</i> )<br>THEN <i>compound_statement</i><br>[ELSE <i>compound_statement</i> ]                                | Tests <i>condition</i> , and if it is TRUE, performs the statement or block following THEN; otherwise, performs the statement or block following ELSE, if present<br><br><i>condition</i> : a Boolean expression (TRUE, FALSE, or UNKNOWN), generally two expressions as operands of a comparison operator |
| NEW. <i>column</i>                                                                                                            | New context variable that indicates a new column value in an INSERT or UPDATE operation                                                                                                                                                                                                                    |
| OLD. <i>column</i>                                                                                                            | Old context variable that indicates a column value before an UPDATE or DELETE operation                                                                                                                                                                                                                    |
| POST_EVENT <i>event_name</i>   <i>col</i>                                                                                     | Posts the event, <i>event_name</i> , or uses the value in <i>col</i> as an event name                                                                                                                                                                                                                      |

**Table 2.7** Language extensions for stored procedures (*continued*)

| Statement                                                                       | Description                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SUSPEND                                                                         | In a SELECT procedure: <ul style="list-style-type: none"> <li>• Suspends execution of procedure until next FETCH is issued by the calling application</li> <li>• Returns output values, if any, to the calling application</li> <li>• Not recommended for executable procedures</li> </ul>                                                                                            |
| WHILE ( <i>condition</i> )<br>DO <i>compound_statement</i>                      | While <i>condition</i> is TRUE, keep performing <i>compound_statement</i> <ul style="list-style-type: none"> <li>• Tests <i>condition</i>, and performs <i>compound_statement</i> if <i>condition</i> is TRUE</li> <li>• Repeats this sequence until <i>condition</i> is no longer TRUE</li> </ul>                                                                                    |
| WHEN { <i>error</i> [, <i>error</i> ...]   ANY}<br>DO <i>compound_statement</i> | Error-handling statement: when one of the specified errors occurs, performs <i>compound_statement</i> <ul style="list-style-type: none"> <li>• WHEN statements, if present, must come at the end of a block, just before END</li> <li>• <i>error</i>: EXCEPTION <i>exception_name</i>, SQLCODE <i>errcode</i> or GDSCODE <i>errcode</i></li> <li>• ANY: Handles any errors</li> </ul> |

The stored procedure and trigger language does not include many of the statement types available in DSQL or **gpre**. The following statement types are not supported in triggers or stored procedures:

- Data definition language statements: CREATE, ALTER, DROP, DECLARE EXTERNAL FUNCTION, and DECLARE FILTER
- Transaction control statements: SET TRANSACTION, COMMIT, ROLLBACK
- Dynamic SQL statements: PREPARE, DESCRIBE, EXECUTE
- CONNECT/DISCONNECT, and sending SQL statements to another database
- GRANT/REVOKE
- SET GENERATOR
- EVENT INIT/WAIT
- BEGIN/END DECLARE SECTION
- BASED ON
- WHENEVER
- DECLARE CURSOR
- OPEN
- FETCH

**Examples** The following procedure, SUB\_TOT\_BUDGET, takes a department number as its input parameter, and returns the total, average, smallest, and largest budgets of departments with the specified HEAD\_DEPT.

```
CREATE PROCEDURE SUB_TOT_BUDGET (HEAD_DEPT CHAR(3))
 RETURNS (tot_bwludget DECIMAL(12, 2), avg_budget
DECIMAL(12, 2),
 min_budget DECIMAL(12, 2), max_budget DECIMAL(12, 2))
AS
BEGIN
 SELECT SUM(BUDGET) , AVG(BUDGET) , MIN(BUDGET) ,
MAX(BUDGET)
 FROM DEPARTMENT
 WHERE HEAD_DEPT = :head_dept
 INTO :tot_budget, :avg_budget, :min_budget,
:max_budget;
 EXIT;
END ;
```

The following SELECT procedure, ORG\_CHART, displays an organizational chart that shows the department name, the parent department, the department manager, the manager's job title, and the number of employees in the department.:

```
CREATE PROCEDURE ORG_CHART
 RETURNS (HEAD_DEPT CHAR(25), DEPARTMENT CHAR(25),
 MNGR_NAME CHAR(20), TITLE CHAR(5), EMP_CNT INTEGER)
AS
 DECLARE VARIABLE mngr_no INTEGER;
 DECLARE VARIABLE dno CHAR(3);
 BEGIN
 FOR SELECT H.DEPARTMENT, D.DEPARTMENT, D.MNGR_NO,
D.DEPT_NO
 FROM DEPARTMENT D
 LEFT OUTER JOIN DEPARTMENT H ON D.HEAD_DEPT =
H.DEPT_NO
 ORDER BY D.DEPT_NO
 INTO :head_dept, :department, :mngr_no, :dno
 DO
 BEGIN
 IF (:mngr_no IS NULL) THEN
 BEGIN
 MNGR_NAME = '---TBH---';
 TITLE = '';
 END
 ELSE
 SELECT FULL_NAME, JOB_CODE
 FROM EMPLOYEE
 WHERE EMP_NO = :mngr_no
 INTO :mngr_name, :title;
 SELECT COUNT(EMP_NO)
```

```
 FROM EMPLOYEE
 WHERE DEPT_NO = :dno
 INTO :emp_cnt;
 SUSPEND;
END
END ;
```

When `ORG_CHART` is invoked, for example in the following **isql** statement:

```
SELECT * FROM ORG_CHART
```

it displays the department name for each department, which department it is in, the department manager's name and title, and the number of employees in the department.

| HEAD_DEPT                | DEPARTMENT               | MGR_NAME           | TITLE | EMP_CNT |
|--------------------------|--------------------------|--------------------|-------|---------|
| =====                    | =====                    | =====              | ===== | =====   |
|                          | Corporate Headquarters   | Bender, Oliver H.  | CEO   | 2       |
| Corporate Headquarters   | Sales and Marketing      | MacDonald, Mary S. | VP    | 2       |
| Sales and Marketing      | Pacific Rim Headquarters | Baldwin, Janet ?   | Sales | 2       |
| Pacific Rim Headquarters | Field Office: Japan      | Yamamoto, Takashi  | SRep  | 2       |
| Pacific Rim Headquarters | Field Office: Singapore  | —TBH—              |       | 0       |

`ORG_CHART` must be used as a select procedure to display the full organization. If called with `EXECUTE PROCEDURE`, the first time it encounters the `SUSPEND` statement, it terminates, returning the information for Corporate Headquarters only.

**See also** `ALTER EXCEPTION`, `ALTER PROCEDURE`, `CREATE EXCEPTION`, `DROP EXCEPTION`, `DROP PROCEDURE`, `EXECUTE PROCEDURE`, `SELECT`

For more information on creating and using procedures, see the [Data Definition Guide](#).

For a complete description of the statements in procedure and trigger language, see [Chapter 3, “Procedures and Triggers.”](#)

## CREATE ROLE

---

Creates a role.

**Syntax** `CREATE ROLE rolename;`

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument        | Description                                                                    |
|-----------------|--------------------------------------------------------------------------------|
| <i>rolename</i> | Name associated with the role; must be unique among role names in the database |

**Description** Roles created with CREATE ROLE can be granted privileges just as users can. These roles can be granted to users, who then inherit the privilege list that has been granted to the role. Users must specify the role at connect time. Use GRANT to grant privileges (ALL, SELECT, INSERT, UPDATE, DELETE, EXECUTE, REFERENCES) to a role and to grant a role to users. Use REVOKE to revoke them.

**Example** The following statement creates a role called “administrator.”

```
CREATE ROLE administrator;
```

**See also** GRANT, REVOKE, DROP ROLE

## CREATE SHADOW

Creates one or more duplicate, in-sync copies of a database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** `CREATE SHADOW set_num [AUTO | MANUAL] [CONDITIONAL]  
'filespec' [LENGTH [=] int [PAGE[S]]]  
[secondary_file];`

`secondary_file = FILE 'filespec' [fileinfo] [secondary_file]  
fileinfo = LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]]  
int  
[fileinfo]`

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument       | Description                                                                                                                                                                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>set_num</i> | Positive integer that designates a shadow set to which all subsequent files listed in the statement belong                                                                                                                                                    |
| AUTO           | Specifies the default access behavior for databases in the event no shadow is available <ul style="list-style-type: none"> <li>• All attachments and accesses succeed</li> <li>• Deletes all references to the shadow and detaches the shadow file</li> </ul> |
| MANUAL         | Specifies that database attachments and accesses fail until a shadow becomes available, or until all references to the shadow are removed from the database                                                                                                   |

| Argument                        | Description                                                                                                                                               |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONDITIONAL                     | Creates a new shadow, allowing shadowing to continue if the primary shadow becomes unavailable or if the shadow replaces the database due to disk failure |
| <i>'filespec'</i>               | Explicit path name and file name for the shadow file; must be a local file system and must not include a node name or be on a networked file system       |
| LENGTH [=] <i>int</i> [PAGE[S]] | Length in database pages of an additional shadow file; page size is determined by the page size of the database itself                                    |
| <i>secondary_file</i>           | Specifies the length of a primary or secondary shadow file; use for primary file only if defining a secondary file in the same statement                  |
| STARTING [AT [PAGE]] <i>int</i> | Starting page number at which a secondary shadow file begins                                                                                              |

**Description** CREATE SHADOW is used to guard against loss of access to a database by establishing one or more copies of the database on secondary storage devices. Each copy of the database consists of one or more shadow files, referred to as a *shadow set*. Each shadow set is designated by a unique positive integer.

Disk shadowing has three components:

- A database to shadow.
- The RDB\$FILES system table, which lists shadow files and other information about the database.
- A shadow set, consisting of one or more shadow files.

When CREATE SHADOW is issued, a shadow is established for the database most recently attached by an application. A shadow set can consist of one or multiple files. In case of disk failure, the database administrator (DBA) activates the disk shadow so that it can take the place of the database. If CONDITIONAL is specified, then when the DBA activates the disk shadow to replace an actual database, a new shadow is established for the database.

If a database is larger than the space available for a shadow on one disk, use the *secondary\_file* option to define multiple shadow files. Multiple shadow files can be spread over several disks.

**Tip** To add a secondary file to an existing disk shadow, drop the shadow with DROP SHADOW and use CREATE SHADOW to recreate it with the desired number of files.

**Examples** The following **isql** statement creates a single, automatic shadow file for *employee.ib*:

```
CREATE SHADOW 1 AUTO 'employee.shd';
```

The next **isql** statement creates a conditional, single, automatic shadow file for *employee.ib*:

```
CREATE SHADOW 2 CONDITIONAL 'employee.shd' LENGTH 1000;
```

The following **isql** statements create a multiple-file shadow set for the *employee.ib* database. The first statement specifies starting pages for the shadow files; the second statement specifies the number of pages for the shadow files.

```
CREATE SHADOW 3 AUTO
 'employee.sh1'
 FILE 'employee.sh2'
 STARTING AT PAGE 1000
 FILE 'employee.sh3'
 STARTING AT PAGE 2000;
CREATE SHADOW 4 MANUAL 'employee.sdw'
 LENGTH 1000
 FILE 'employee.sh1'
 LENGTH 1000
 FILE 'employee.sh2';
```

**See also** DROP SHADOW

For more information about using shadows, see the [Operations Guide](#) or the [Data Definition Guide](#).

## CREATE TABLE

Creates a new table in an existing database. Available in **gpre**, **DSQL**, and **isql**.

**Important** To create a SQL Global Temporary table see: SQL global temporary tables in the [Data Definition Guide](#)

**Syntax** **CREATE TABLE** *table* [EXTERNAL [FILE] '*filespec*']  
 (*col\_def* [, *col\_def* | *tconstraint* ...]);

*col\_def* = *col* {datatype | COMPUTED [BY] (*expr*) | *domain*}  
 [DEFAULT {*literal* | NULL | USER}]  
 [NOT NULL]  
 [*col\_constraint*]  
 [COLLATE *collation*]

*datatype* =  
 {SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION} [*array\_dim*]  
 | {DATE | TIME | TIMESTAMP} [*array\_dim*]  
 | {DECIMAL | NUMERIC} [(*precision* [, *scale*))] [*array\_dim*]  
 | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(*int*)]  
 [*array\_dim*] [CHARACTER SET *charname*]  
 | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}  
 [VARYING] [(*int*)] [*array\_dim*]  
 | BLOB [SUB\_TYPE {*int* | *subtype\_name*}] [SEGMENT SIZE *int*]  
 [CHARACTER SET *charname*]

```

| BLOB [(seglen [, subtype])]
| BOOLEAN

array_dim = [[x:]y [, [x:]y ...]]

expr = A valid SQL expression that results in a single
value.

col_constraint = [CONSTRAINT constraint]
{
| UNIQUE
| PRIMARY KEY
| REFERENCES other_table [(other_col [, other_col ...])]
 [ON DELETE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
 [ON UPDATE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
| CHECK (search_condition)}

tconstraint = [CONSTRAINT constraint]
{{PRIMARY KEY | UNIQUE} (col [, col ...])
| FOREIGN KEY (col [, col ...])
 REFERENCES other_table [(other_col [, other_col ...])]
 [ON DELETE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
 [ON UPDATE {RESTRICT|NO ACTION|CASCADE|SET DEFAULT|SET
NULL}]
| CHECK (search_condition)}

search_condition = val operator {val | (select_one)}
| val [NOT] BETWEEN val AND val
| val [NOT] LIKE val [ESCAPE val]
| val [NOT] IN (val [, val ...] | select_list)
| val IS [NOT] NULL
| val {>= | <=}
| val [NOT] {= | < | >}
| {ALL | SOME | ANY} (select_list)
| EXISTS (select_expr)
| SINGULAR (select_expr)
| val [NOT] CONTAINING val
| val [NOT] STARTING [WITH] val
 (search_condition)
| NOT search_condition
| search_condition OR search_condition
| search_condition AND search_condition

val = { col [array_dim] | :variable
| constant | expr | function
| udf ([val [, val ...]])
| NULL | USER | RDB$DB_KEY | ? }
[COLLATE collation]

constant = num | 'string' | charsetname 'string'

```

```

function = COUNT (* | [ALL] val | DISTINCT val)
 SUM ([ALL] val | DISTINCT val)
 AVG ([ALL] val | DISTINCT val)
 MAX ([ALL] val | DISTINCT val)
 MIN ([ALL] val | DISTINCT val)
 CAST (val AS datatype)
 UPPER (val)
 GEN_ID (generator, val)

```

```
operator = {= | < | > | <= | >= | !< | !> | <> | !=}
```

*select\_one* = SELECT on a single column; returns exactly one value.

*select\_list* = SELECT on a single column; returns zero or more values.

*select\_expr* = SELECT on a list of values; returns zero or more values.

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

#### Notes on the CREATE TABLE statement

- When declaring arrays, you must include the outermost brackets, shown below in bold. For example, the following statement creates a 5 by 5 two-dimensional array of strings, each of which is 6 characters long:

```
my_array VARCHAR(6) [5,5]
```

- Use the colon (:) to specify an array with a starting point other than 1. The following example creates an array of integers that begins at 10 and ends at 20:

```
my_array INTEGER[10:20]
```

- In SQL and **isql**, you cannot use *val* as a parameter placeholder (like "?").
- In DSQL and **isql**, *val* cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.
- *expr* is any complex SQL statement or equation that produces a single value.

| Argument                            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>table</i>                        | Name for the table; must be unique among table and procedure names in the database                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| EXTERNAL [FILE] ' <i>filespec</i> ' | Declares that data for the table under creation resides in a table or file outside the database; <i>filespec</i> is the complete file specification of the external file or table                                                                                                                                                                                                                                                                                                                                                            |
| <i>col</i>                          | Name for the table column; unique among column names in the table. You can also encrypt/decrypt a column when you create a table. For instructions on how to encrypt and decrypt a column or database see Chapter 13, "Encrypting Your Data" in the <a href="#">Data Definition Guide</a> .                                                                                                                                                                                                                                                  |
| <i>datatype</i>                     | SQL datatype for the column; see "Datatypes" on page 2-6                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| COMPUTED [BY] ( <i>expr</i> )       | <p>Specifies that the value of the column's data is calculated from <i>expr</i> at runtime and is therefore not allocated storage space in the database</p> <ul style="list-style-type: none"> <li>• <i>expr</i> can be any arithmetic expression valid for the datatypes in the expression</li> <li>• Any columns referenced in <i>expr</i> must exist before they can be used in <i>expr</i></li> <li>• <i>expr</i> cannot reference Blob columns</li> <li>• <i>expr</i> must return a single value, and cannot return an array</li> </ul> |
| <i>domain</i>                       | Name of an existing domain                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| DEFAULT                             | <p>Specifies a default column value that is entered when no other entry is made; possible values are:</p> <ul style="list-style-type: none"> <li>• <i>literal</i>: Inserts a specified string, numeric value, or date value</li> <li>• NULL: Enters a NULL value</li> <li>• USER: Enters the user name of the current user. Column must be of compatible text type to use the default</li> </ul> <p>Defaults set at column level override defaults set at the domain level.</p>                                                              |
| CONSTRAINT <i>constraint</i>        | Name of a column or table constraint; the constraint name must be unique within the table                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>constraint_def</i>               | Specifies the kind of column constraint; valid options are UNIQUE, PRIMARY KEY, CHECK, and REFERENCES                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| REFERENCES                          | Specifies that the column values are derived from column values in another table; if you do not specify column names, InterBase looks for a column with the same name as the referencing column in the referenced table                                                                                                                                                                                                                                                                                                                      |

| Argument                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ON DELETE   ON UPDATE         | Used with REFERENCES: Changes a foreign key whenever the referenced primary key changes; valid options are: <ul style="list-style-type: none"> <li>• [Default] NO ACTION: Does not change the foreign key; may cause the primary key update to fail due to referential integrity checks</li> <li>• CASCADE: For ON DELETE, deletes the corresponding foreign key; for ON UPDATE, updates the corresponding foreign key to the new value of the primary key</li> <li>• SET NULL: Sets all the columns of the corresponding foreign key to NULL</li> <li>• SET DEFAULT: Sets every column of the corresponding foreign key is set to its default value in effect when the referential integrity constraint is defined. When the default for a foreign column changes <i>after</i> the referential integrity constraint is defined, the change does not have an effect on the default value used in the referential integrity constraint</li> </ul> |
| CHECK <i>search_condition</i> | An attempt to enter a new value in the column fails if the value does not meet the <i>search_condition</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| COLLATE <i>collation</i>      | Establishes a default sorting behavior for the column; see Chapter 7, “Character Sets and Collation Orders” for more information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

**Description** CREATE TABLE establishes a new table, its columns, and integrity constraints in an existing database. The user who creates a table is the table’s owner and has all privileges for it, including the ability to GRANT privileges to other users, triggers, and stored procedures.

- CREATE TABLE supports several options for defining columns:
  - Local columns specify the name and datatype for data entered into the column.
  - Computed columns are based on an expression. Column values are computed each time the table is accessed. If the datatype is not specified, InterBase calculates an appropriate one. Columns referenced in the expression must exist before the column can be defined.
  - Domain-based columns inherit all the characteristics of a domain, but the column definition can include a new default value, a NOT NULL attribute, additional CHECK constraints, or a collation clause that overrides the domain definition. It can also include additional column constraints.
  - The datatype specification for a CHAR, VARCHAR, or Blob text column definition can include a CHARACTER SET clause to specify a particular character set for the single column. Otherwise, the column uses the default database character set. If the database character set is changed, all columns subsequently defined have the new character set, but existing columns are

not affected. For a complete list of character sets recognized by InterBase, see [Chapter 7, “Character Sets and Collation Orders.”](#)

- If you do not specify a default character set, the character set defaults to NONE. Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it. You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a different character set. In this case, no transliteration is performed between the source and destination character sets, and errors may occur during assignment.
- The COLLATE clause enables specification of a particular collation order for CHAR, VARCHAR, and Blob text datatypes. Choice of collation order is restricted to those supported for the column's given character set, which is either the default character set for the entire database, or a different set defined in the CHARACTER SET clause as part of the datatype definition. For a complete list of collation orders recognized by InterBase, see [Chapter 7, “Character Sets and Collation Orders.”](#)
- NOT NULL is an attribute that prevents the entry of NULL or unknown values in column. NOT NULL affects all INSERT and UPDATE operations on a column.

**Important** A DECLARE TABLE must precede CREATE TABLE in embedded applications if the same SQL program both creates a table and inserts data in the table.

- The EXTERNAL FILE option creates a table whose data resides in an external file, rather than in the InterBase database. Use this option to:
  - Define an InterBase table composed of data from an external source, such as data in files managed by other operating systems or in non-database applications.
  - Transfer data to an existing InterBase table from an external file.

External files must either be placed in `<InterBase_home>/ext` or their location must be specified in the *ibconfig* configuration file using the EXTERNAL\_FILE\_DIRECTORY entry.

### Referential integrity constraints

- You can define integrity constraints at the time you create a table. These constraints are rules that validate data entries by enforcing column-to-table and table-to-table relationships. They span all transactions that access the database and are automatically maintained by the system. CREATE TABLE supports the following integrity constraints:
  - A PRIMARY KEY is one or more columns whose collective contents are guaranteed to be unique. A PRIMARY KEY column must also define the NOT NULL attribute. A table can have only one primary key.

- UNIQUE keys ensure that no two rows have the same value for a specified column or ordered set of columns. A unique column must also define the NOT NULL attribute. A table can have one or more UNIQUE keys. A UNIQUE key can be referenced by a FOREIGN KEY in another table.
- Referential constraints (REFERENCES) ensure that values in the specified columns (known as the *foreign key*) are the same as values in the referenced UNIQUE or PRIMARY KEY columns in another table. The UNIQUE or PRIMARY KEY columns in the referenced table must be defined before the REFERENCES constraint is added to the secondary table. REFERENCES has ON DELETE and ON UPDATE clauses that define the action on the foreign key when the referenced primary key is updated or deleted. The values for ON UPDATE and ON DELETE are as follows:

| Action specified | Effect on foreign key                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NO ACTION        | [Default] The foreign key does not change. This may cause the primary key update or delete to fail due to referential integrity checks.                                                                                                                                                                                                                                                                                                                                                                              |
| CASCADE          | The corresponding foreign key is updated or deleted as appropriate to the new value of the primary key.                                                                                                                                                                                                                                                                                                                                                                                                              |
| SET DEFAULT      | Every column of the corresponding foreign key is set to its default value. If the default value of the foreign key is not found in the primary key, the update or delete on the primary key fails.<br>The default value is the one in effect when the referential integrity constraint was defined. When the default for a foreign key column is changed after the referential integrity constraint is set up, the change does not have an effect on the default value used in the referential integrity constraint. |
| SET NULL         | Every column of the corresponding foreign key is set to NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

- You can create a FOREIGN KEY reference to a table that is owned by someone else only if that owner has explicitly granted you REFERENCES privilege on that table. Any user who updates your foreign key table must have REFERENCES or SELECT privileges on the referenced primary key table.
- CHECK constraints enforce a *search\_condition* that must be true for inserts or updates to the specified table. *search\_condition* can require a combination or range of values or can compare the value entered with data in other columns.

**Note** Specifying USER as the value for a *search\_condition* references the login of the user who is attempting to write to the referenced table.

- Creating PRIMARY KEY and FOREIGN KEY constraints requires exclusive access to the database.
- For unnamed constraints, the system assigns a unique constraint name stored in the RDB\$RELATION\_CONSTRAINTS system table.

**Note** Constraints are not enforced on expressions.

**Examples** The following **isql** statement creates a simple table with a PRIMARY KEY:

```
CREATE TABLE COUNTRY (COUNTRY COUNTRYNAME NOT NULL PRIMARY
KEY,
 CURRENCY VARCHAR(10) NOT NULL);
```

The next **isql** statement creates both a column-level and a table-level UNIQUE constraint:

```
CREATE TABLE STOCK (
 MODEL SMALLINT NOT NULL UNIQUE,
 MODELNAME CHAR(10) NOT NULL,
 ITEMID INTEGER NOT NULL,
 CONSTRAINT MOD_UNIQUE UNIQUE (MODELNAME, ITEMID));
```

The following **isql** statement illustrates table-level PRIMARY KEY, FOREIGN KEY, and CHECK constraints. The PRIMARY KEY constraint is based on three columns. This example also illustrates creating an array column of VARCHAR.

```
CREATE TABLE JOB (
 JOB_CODE JOBCODE NOT NULL,
 JOB_GRADE JOBGRADE NOT NULL,
 JOB_COUNTRY COUNTRYNAME NOT NULL,
 JOB_TITLE VARCHAR(25) NOT NULL,
 MIN_SALARY SALARY NOT NULL,
 MAX_SALARY SALARY NOT NULL,
 JOB_REQUIREMENT BLOB(400,1),
 LANGUAGE_REQ VARCHAR(15) [5],
 PRIMARY KEY (JOB_CODE, JOB_GRADE, JOB_COUNTRY),
 FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY),
 CHECK (MIN_SALARY < MAX_SALARY));
```

In the next example, the F2 column in table T2 is a foreign key that references table T1 through T1's primary key P1. When a row in T1 changes, that change propagates to all affected rows in table T2. When a row in T1 is deleted, all affected rows in the F2 column of table T2 are set to NULL.

```
CREATE TABLE T1 (P1 INTEGER NOT NULL PRIMARY KEY);

CREATE TABLE T2 (F2 INTEGER FOREIGN KEY (F2) REFERENCES T1
(P1)
 ON UPDATE CASCADE
 ON DELETE SET NULL);
```

The next **isql** statement creates a table with a calculated column:

```
CREATE TABLE SALARY_HISTORY (
 EMP_NO EMPNO NOT NULL,
 CHANGE_DATE DATE DEFAULT 'NOW' NOT NULL,
 UPDATER_ID VARCHAR(20) NOT NULL,
 OLD_SALARY SALARY NOT NULL,
 PERCENT_CHANGE DOUBLE PRECISION
```

```

 DEFAULT 0
 NOT NULL
 CHECK (PERCENT_CHANGE BETWEEN -50 AND 50),
 NEW_SALARY COMPUTED BY
 (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100),
 PRIMARY KEY (EMP_NO, CHANGE_DATE, UPDATER_ID),
 FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO));

```

In the following **isql** statement the first column retains the default collating order for the database's default character set. The second column has a different collating order, and the third column definition includes a character set and a collating order.

```

CREATE TABLE BOOKADVANCE (
 BOOKNO CHAR(6),
 TITLE CHAR(50) COLLATE ISO8859_1,
 EUROPUB CHAR(50) CHARACTER SET ISO8859_1 COLLATE FR_FR);

```

**See also** CREATE DOMAIN, DECLARE TABLE, GRANT, REVOKE

For more information on creating metadata, using integrity constraints, external tables, datatypes, collation order, and character sets, see the [Data Definition Guide](#).

For detailed information on encryption and decryption, see the topics “Encrypting Data” (page 13-9) and “Decrypting Data” (page 13-11) in the [Data Definition Guide](#).

## CREATE TRIGGER

---

Creates a trigger, including when it fires, and what actions it performs. Available in DSQL, and **isql**.

**Syntax** CREATE TRIGGER *name* FOR *table*

```

 [ACTIVE | INACTIVE]
 {BEFORE | AFTER}
 {DELETE | INSERT | UPDATE}
 [POSITION number]
 AS trigger_body;

```

*trigger\_body* = [*variable\_declaration\_list*] block

```

variable_declaration_list =
 DECLARE VARIABLE variable datatype;
 [DECLARE VARIABLE variable datatype; ...]

```

```

block =
 BEGIN
 compound_statement
 [compound_statement ...]
 END

```

```

datatype = SMALLINT
| INTEGER
| FLOAT
| DOUBLE PRECISION
| {DECIMAL | NUMERIC} [(precision [, scale])]
| {DATE | TIME | TIMESTAMP}
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
 [(int)] [CHARACTER SET charname]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
 [(int)]
| BOOLEAN

compound_statement = block | statement;

```

| Argument                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name</i>                             | Name of the trigger; must be unique in the database                                                                                                                                                                                                                                                                                                                                                                                     |
| <i>table</i>                            | Name of the table or view that causes the trigger to fire when the specified operation occurs on the table or view                                                                                                                                                                                                                                                                                                                      |
| ACTIVE INACTIVE                         | Optional. Specifies trigger action at transaction end: <ul style="list-style-type: none"> <li>• ACTIVE: [Default] Trigger takes effect</li> <li>• INACTIVE: Trigger does not take effect</li> </ul>                                                                                                                                                                                                                                     |
| BEFORE AFTER                            | Required. Specifies whether the trigger fires: <ul style="list-style-type: none"> <li>• BEFORE: Before associated operation</li> <li>• AFTER: After associated operation</li> </ul> Associated operations are DELETE, INSERT, or UPDATE                                                                                                                                                                                                 |
| DELETE INSERT<br> UPDATE                | Specifies the table operation that causes the trigger to fire                                                                                                                                                                                                                                                                                                                                                                           |
| POSITION <i>number</i>                  | Specifies firing order for triggers before the same action or after the same action; <i>number</i> must be an integer between 0 and 32,767, inclusive. <ul style="list-style-type: none"> <li>• Lower-number triggers fire first</li> <li>• Default: 0 = first trigger to fire</li> <li>• Triggers for a table need not be consecutive; triggers on the same action with the same position number will fire in random order.</li> </ul> |
| DECLARE VARIABLE<br><i>var datatype</i> | Declares local variables used only in the trigger. Each declaration must be preceded by DECLARE VARIABLE and followed by a semicolon (;). <ul style="list-style-type: none"> <li>• <i>var</i>: Local variable name, unique in the trigger</li> <li>• <i>datatype</i>: The datatype of the local variable</li> </ul>                                                                                                                     |
| <i>statement</i>                        | Any single statement in InterBase procedure and trigger language; each statement except BEGIN and END must be followed by a semicolon (;)                                                                                                                                                                                                                                                                                               |

**Description** CREATE TRIGGER defines a new trigger to a database. A trigger is a self-contained program associated with a table or view that automatically performs an action when a row in the table or view is inserted, updated, or deleted.

A trigger is never called directly. Instead, when an application or user attempts to INSERT, UPDATE, or DELETE a row in a table, any triggers associated with that table and operation automatically execute, or *fire*. Triggers defined for UPDATE on non-updatable views fire even if no update occurs.

A trigger is composed of a *header* and a *body*.

The trigger header contains:

- A *trigger name*, unique within the database, that distinguishes the trigger from all others.
- A *table name*, identifying the table with which to associate the trigger.
- *Statements* that determine when the trigger fires.

The trigger body contains:

- An optional list of *local variables* and their datatypes.
- A *block* of statements in InterBase procedure and trigger language, bracketed by BEGIN and END. These statements are performed when the trigger fires. A block can itself include other blocks, so that there may be many levels of nesting.

A trigger is associated with a table. The table owner and any user granted privileges to the table automatically have rights to execute associated triggers.

Triggers can be granted privileges on tables, just as users or procedures can be granted privileges. Use the GRANT statement, but instead of using TO *username*, use TO TRIGGER *trigger\_name*. Triggers' privileges can be revoked similarly using REVOKE.

When a user performs an action that fires a trigger, the trigger will have privileges to perform its actions if one of the following conditions is true:

- The trigger has privileges for the action.
- The user has privileges for the action.

InterBase procedure and trigger language is a complete programming language for stored procedures and triggers. It includes:

- SQL data manipulation statements: INSERT, UPDATE, DELETE, and singleton SELECT.
- SQL operators and expressions, including generators and UDFs that are linked with the calling application.
- Powerful extensions to SQL, including assignment statements, control-flow statements, context variables, event-posting statements, exceptions, and error-handling statements.

The following table summarizes language extensions for triggers. For a complete description of each statement, see [Chapter 3, “Procedures and Triggers.”](#)

**Table 2.8** Language extensions for triggers

| Statement                                                                                                                  | Description                                                                                                                                                                                                                                                           |
|----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BEGIN ... END                                                                                                              | Defines a block of statements that executes as one <ul style="list-style-type: none"> <li>The BEGIN keyword starts the block; the END keyword terminates it</li> <li>Neither should be followed by a semicolon</li> </ul>                                             |
| <i>variable</i> = <i>expression</i>                                                                                        | Assignment statement that assigns the value of <i>expression</i> to <i>variable</i> , a local variable, input parameter, or output parameter                                                                                                                          |
| <i>/* comment_text */</i>                                                                                                  | Programmer's comment, where <i>comment_text</i> can be any number of lines of text                                                                                                                                                                                    |
| EXCEPTION <i>exception_name</i>                                                                                            | Raises the named exception; an exception is a user-defined error that returns an error message to the calling application unless handled by a WHEN statement                                                                                                          |
| EXECUTE PROCEDURE<br><i>proc_name</i> [ <i>var</i> [, <i>var</i> ...]]<br>[RETURNING_VALUES <i>var</i> [, <i>var</i> ...]] | Executes stored procedure, <i>proc_name</i> , with the listed input arguments <ul style="list-style-type: none"> <li>Returns values in the listed output arguments following RETURNING_VALUES</li> <li>Input and output arguments must be local variables.</li> </ul> |
| EXIT                                                                                                                       | Jumps to the final END statement in the procedure                                                                                                                                                                                                                     |
| FOR <i>select_statement</i><br>DO <i>compound_statement</i>                                                                | Repeats the statement or block following DO for every qualifying row retrieved by <i>select_statement</i>                                                                                                                                                             |
| <i>select_statement</i>                                                                                                    | A normal SELECT statement                                                                                                                                                                                                                                             |
| <i>compound_statement</i>                                                                                                  | Either a single statement in procedure and trigger language or a block of statements bracketed by BEGIN and END                                                                                                                                                       |
| IF ( <i>condition</i> )<br>THEN <i>compound_statement</i><br>[ELSE <i>compound_statement</i> ]                             | Tests <i>condition</i> , and if it is TRUE, performs the statement or block following THEN; otherwise, performs the statement or block following ELSE, if present                                                                                                     |
| <i>condition</i>                                                                                                           | A Boolean expression (TRUE, FALSE, or UNKNOWN), generally two expressions as operands of a comparison operator                                                                                                                                                        |
| NEW. <i>column</i>                                                                                                         | New context variable that indicates a new column value in an INSERT or UPDATE operation                                                                                                                                                                               |

**Table 2.8** Language extensions for triggers (*continued*)

| Statement                                                                       | Description                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OLD. <i>column</i>                                                              | Old context variable that indicates a column value before an UPDATE or DELETE operation                                                                                                                                                                                                            |
| POST_EVENT <i>event_name</i>   <i>col</i>                                       | Posts the event, <i>event_name</i> , or uses the value in <i>col</i> as an event name                                                                                                                                                                                                              |
| WHILE ( <i>condition</i> )<br>DO <i>compound_statement</i>                      | While <i>condition</i> is TRUE, keep performing <i>compound_statement</i> <ul style="list-style-type: none"> <li>• Tests <i>condition</i>, and performs <i>compound_statement</i> if <i>condition</i> is TRUE</li> <li>• Repeats this sequence until <i>condition</i> is no longer TRUE</li> </ul> |
| WHEN { <i>error</i> [, <i>error</i> ...]   ANY}<br>DO <i>compound_statement</i> | Error-handling statement. When one of the specified errors occurs, performs <i>compound_statement</i> . WHEN statements, if present, must come at the end of a block, just before END <ul style="list-style-type: none"> <li>• ANY: Handles any errors</li> </ul>                                  |
| <i>error</i>                                                                    | EXCEPTION <i>exception_name</i> , SQLCODE <i>errcode</i> or GDSCODE <i>errcode</i>                                                                                                                                                                                                                 |

The stored procedure and trigger language does not include many of the statement types available in DSQL or **gpre**. The following statement types are not supported in triggers or stored procedures:

- Data definition language statements: CREATE, ALTER, DROP, DECLARE EXTERNAL FUNCTION, and DECLARE FILTER
- Transaction control statements: SET TRANSACTION, COMMIT, ROLLBACK
- Dynamic SQL statements: PREPARE, DESCRIBE, EXECUTE
- CONNECT/DISCONNECT, and sending SQL statements to another database
- GRANT/REVOKE
- SET GENERATOR
- EVENT INIT/WAIT
- BEGIN/END DECLARE SECTION
- BASED ON
- WHENEVER
- DECLARE CURSOR
- OPEN
- FETCH

**Examples** The following trigger, SAVE\_SALARY\_CHANGE, makes correlated updates to the SALARY\_HISTORY table when a change is made to an employee's salary in the EMPLOYEE table:

```
CREATE TRIGGER SAVE_SALARY_CHANGE FOR EMPLOYEE
 AFTER UPDATE AS
 BEGIN
 IF (OLD.SALARY <> NEW.SALARY) THEN
 INSERT INTO SALARY_HISTORY
 (EMP_NO, CHANGE_DATE, UPDATER_ID, OLD_SALARY,
 PERCENT_CHANGE)
 VALUES (OLD.EMP_NO, 'now', USER, OLD.SALARY,
 (NEW.SALARY - OLD.SALARY) * 100 / OLD.SALARY);
 END ;
```

The following trigger, SET\_CUST\_NO, uses a generator to create unique customer numbers when a new customer record is inserted in the CUSTOMER table.

```
CREATE TRIGGER SET_CUST_NO FOR CUSTOMER
 BEFORE INSERT AS
 BEGIN
 NEW.CUST_NO = GEN_ID(CUST_NO_GEN, 1);
 END ;
```

The following trigger, POST\_NEW\_ORDER, posts an event named "new\_order" whenever a new record is inserted in the SALES table.

```
CREATE TRIGGER POST_NEW_ORDER FOR SALES
 AFTER INSERT AS
 BEGIN
 POST_EVENT 'new_order';
 END ;
```

The following four fragments of trigger headers demonstrate how the POSITION option determines trigger firing order:

```
CREATE TRIGGER A FOR accounts
 BEFORE UPDATE
 POSITION 5 ... /*Trigger body follows*/

CREATE TRIGGER B FOR accounts
 BEFORE UPDATE
 POSITION 0 ... /*Trigger body follows*/

CREATE TRIGGER C FOR accounts
 AFTER UPDATE
 POSITION 5 ... /*Trigger body follows*/

CREATE TRIGGER D FOR accounts
 AFTER UPDATE
 POSITION 3 ... /*Trigger body follows*/
```

When this update takes place:

```
UPDATE accounts SET account_status = 'on_hold'
 WHERE account_balance <0;
```

The triggers fire in this order:

- 1 Trigger B fires.
- 2 Trigger A fires.
- 3 The update occurs.
- 4 Trigger D fires.
- 5 Trigger C fires.

**See also** ALTER EXCEPTION, ALTER TRIGGER, CREATE EXCEPTION, CREATE PROCEDURE, DROP EXCEPTION, DROP TRIGGER, EXECUTE PROCEDURE

For more information on creating and using triggers, see the [Data Definition Guide](#).

For a complete description of the statements in procedure and trigger language, see [Chapter 3, “Procedures and Triggers.”](#)

## CREATE USER

---

Create a new user. Available in **DSQL** and **isql**.

**Syntax** CREATE USER *name* SET  
 [PASSWORD *password*]  
 [[NO] DEFAULT ROLE *name*]  
 [[NO] SYSTEM USER NAME *name*]  
 [[NO] GROUP NAME *name*]  
 [[NO] UID *number*]  
 [[NO] GID *number*]  
 [[NO] DESCRIPTION *string*]  
 [[NO] FIRST NAME *string*]  
 [[NO] MIDDLE NAME *string*]  
 [[NO] LAST NAME *string*]  
 [ACTIVE]  
 [INACTIVE];

| Argument               | Description                      |
|------------------------|----------------------------------|
| PASSWORD               | Password of user                 |
| [[NO] DEFAULT ROLE     | Default role                     |
| [[NO] SYSTEM USER NAME | System user name for target user |

---

| Argument         | Description                                        |
|------------------|----------------------------------------------------|
| [NO] GROUP NAME  | Group name for target user                         |
| [NO] UID         | Target user ID                                     |
| [NO] GID         | Group ID for target user                           |
| [NO] DESCRIPTION | Description                                        |
| [NO] FIRST NAME  | First name for target user                         |
| [NO] MIDDLE NAME | Middle name for target user                        |
| [NO] LAST NAME   | Last name for target user                          |
| ACTIVE           | Default. After inactive, reinstates selected user. |
| INACTIVE         | Prevents a user from logging into database.        |

**Description** CREATE USER creates a new user. Only used with database under embedded user authentication.

**Note** When NO is specified, an argument to the option must not be supplied. NO sets the option to a NULL state.

**Examples** The following statement creates the user, JDOE and set password, jdoe:

```
CREATE USER JDOE SET PASSWORD 'jdoe';
```

The next statement creates the user, JDOE, and set password, first name and last name:

```
CREATE USER JDOE SET PASSWORD 'jdoe', FIRST NAME 'Jane', LAST NAME 'Doe';
```

**See also** ALTER USER, DROP USER

For more information about embedded user authentication, see the [Operations Guide](#).

## CREATE VIEW

Creates a new view of data from one or more tables. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** CREATE VIEW *name* [(*view\_col* [, *view\_col* ...])] AS *select* [WITH CHECK OPTION];

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument             | Description                                                                                                                                                                                                                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name</i>          | Name for the view; must be unique among all view, table, and procedure names in the database                                                                                                                                                                                                          |
| <i>view_col</i>      | Names the columns for the view <ul style="list-style-type: none"> <li>• Column names must be unique among all column names in the view</li> <li>• Required if the view includes columns based on expressions; otherwise optional</li> <li>• Default: Column name from the underlying table</li> </ul> |
| <i>select</i>        | Specifies the selection criteria for rows to be included in the view                                                                                                                                                                                                                                  |
| WITH CHECK<br>OPTION | Prevents INSERT or UPDATE operations on an updatable view if the INSERT or UPDATE violates the search condition specified in the WHERE clause of the view's SELECT clause                                                                                                                             |

**Description** CREATE VIEW describes a view of data based on one or more underlying tables in the database. The rows to return are defined by a SELECT statement that lists columns from the source tables. Only the view definition is stored in the database; a view does not directly represent physically stored data. It is possible to perform select, project, join, and union operations on views as if they were tables.

The user who creates a view is its owner and has all privileges for it, including the ability to GRANT privileges to other users, roles, triggers, views, and stored procedures. A user may have privileges to a view without having access to its base tables. When creating views:

- A read-only view requires SELECT privileges for any underlying tables.
- An updatable view requires ALL privileges to the underlying tables.

The *view\_col* option ensures that the view always contains the same columns and that the columns always have the same view-defined names.

View column names correspond in order and number to the columns listed in the SELECT clause, so specify *all* view column names or none.

A *view\_col* definition can contain one or more columns based on an expression that combines the outcome of two columns. The expression must return a single value, and cannot return an array or array element. If the view includes an expression, the *view-column* option is required.

**Note** Any columns used in the value expression must exist before the expression can be defined.

A SELECT statement clause cannot include the ORDER BY clause.

When SELECT \* is used rather than a column list, order of display is based on the order in which columns are stored in the base table.

WITH CHECK OPTION enables InterBase to verify that a row added to or updated in a view is able to be seen through the view before allowing the operation to succeed. Do not use WITH CHECK OPTION for read-only views.

**Note** You cannot select from a view that is based on the result set of a stored procedure.

**Note** An updatable view cannot have UNION clauses. To create such a view, use embedded SQL.

A view is updatable if:

- It is a subset of a single table or another updatable view.
- All base table columns excluded from the view definition allow NULL values.
- The view's SELECT statement does not contain subqueries, a DISTINCT predicate, a HAVING clause, aggregate functions, joined tables, user-defined functions, or stored procedures.

If the view definition does not meet these conditions, it is considered read-only.

**Note** Read-only views can be updated by using a combination of user-defined referential constraints, triggers, and unique indexes.

**Examples** The following **isql** statement creates an updatable view:

```
CREATE VIEW SNOW_LINE (CITY, STATE, SNOW_ALTITUDE) AS
 SELECT CITY, STATE, ALTITUDE
 FROM CITIES
 WHERE ALTITUDE > 5000;
```

The next **isql** statement uses a nested query to create a view:

```
CREATE VIEW RECENT_CITIES AS
 SELECT STATE, CITY, POPULATION
 FROM CITIES WHERE STATE IN
 (SELECT STATE FROM STATES WHERE STATEHOOD > '1-JAN-
1850');
```

In an updatable view, the WITH CHECK OPTION prevents any inserts or updates through the view that do not satisfy the WHERE clause of the CREATE VIEW SELECT statement:

```
CREATE VIEW HALF_MILE_CITIES AS
 SELECT CITY, STATE, ALTITUDE
 FROM CITIES
 WHERE ALTITUDE > 2500
 WITH CHECK OPTION;
```

The WITH CHECK OPTION clause in the view would prevent the following insertion:

```
INSERT INTO HALF_MILE_CITIES (CITY, STATE, ALTITUDE)
 VALUES ('Chicago', 'Illinois', 250);
```

On the other hand, the following UPDATE would be permitted:

```
INSERT INTO HALF_MILE_CITIES (CITY, STATE, ALTITUDE)
VALUES ('Truckee', 'California', 2736);
```

The WITH CHECK OPTION clause does not allow updates through the view which change the value of a row so that the view cannot retrieve it. For example, the WITH CHECK OPTION in the HALF\_MILE\_CITIES view prevents the following update:

```
UPDATE HALF_MILE_CITIES
SET ALTITUDE = 2000
WHERE STATE = 'NY';
```

The next **isql** statement creates a view that joins two tables, and so is read-only:

```
CREATE VIEW PHONE_LIST AS
SELECT EMP_NO, FIRST_NAME, LAST_NAME, PHONE_EXT, LOCATION,
PHONE_NO
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.DEPT_NO = DEPARTMENT.DEPT_NO;
```

**See also**

CREATE TABLE, DROP VIEW, GRANT, INSERT, REVOKE, SELECT, UPDATE

For a complete discussion of views, see the [Data Definition Guide](#).

## DECLARE CURSOR

Defines a cursor for a table by associating a name with the set of rows specified in a SELECT statement. Available in **gpre** and **DSQL**.

**Syntax SQL form:**

```
DECLARE cursor CURSOR FOR select [FOR UPDATE OF col [,
col...]];
```

**DSQL form:**

```
DECLARE cursor CURSOR FOR statement_id
```

**Blob form:** See DECLARE CURSOR (BLOB)

| Argument                                        | Description                                                                                                     |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <i>cursor</i>                                   | Name for the cursor                                                                                             |
| <i>select</i>                                   | Determines which rows to retrieve. SQL only                                                                     |
| FOR UPDATE OF <i>col</i><br>[, <i>col ...</i> ] | Enables UPDATE and DELETE of specified column for retrieved rows                                                |
| <i>statement_id</i>                             | SQL statement name of a previously prepared statement, which in this case must be a SELECT statement. DSQL only |

**Description** DECLARE CURSOR defines the set of rows that can be retrieved using the cursor it names. It is the first member of a group of table cursor statements that must be used in sequence.

*select* specifies a SELECT statement that determines which rows to retrieve. The SELECT statement cannot include INTO or ORDER BY clauses.

The FOR UPDATE OF clause is necessary for updating or deleting rows using the WHERE CURRENT OF clause with UPDATE and DELETE.

A cursor is a one-way pointer into the ordered set of rows retrieved by the select expression in the DECLARE CURSOR statement. It enables sequential access to retrieved rows in turn. There are four related cursor statements:

| Stage | Statement      | Purpose                                                                                                                                    |
|-------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | DECLARE CURSOR | Declares the cursor; the SELECT statement determines rows retrieved for the cursor                                                         |
| 2     | OPEN           | Retrieves the rows specified for retrieval with DECLARE CURSOR; the resulting rows become the cursor's <i>active set</i>                   |
| 3     | FETCH          | Retrieves the current row from the active set, starting with the first row; subsequent FETCH statements advance the cursor through the set |
| 4     | CLOSE          | Closes the cursor and releases system resources                                                                                            |

**Examples** The following embedded SQL statement declares a cursor with a search condition:

```
EXEC SQL
 DECLARE C CURSOR FOR
 SELECT CUST_NO, ORDER_STATUS
 FROM SALES
 WHERE ORDER_STATUS IN ('open', 'shipping');
```

The next DSQL statement declares a cursor for a previously prepared statement, QUERY1:

```
DECLARE Q CURSOR FOR QUERY1
```

**See also** CLOSE, DECLARE CURSOR (BLOB), FETCH, OPEN, PREPARE, SELECT

## DECLARE CURSOR (BLOB)

Declares a Blob cursor for read or insert. Available in **gpre**.

**Syntax** DECLARE *cursor* CURSOR FOR  
    {READ BLOB *column* FROM *table*  
    | INSERT BLOB *column* INTO *table*}  
    [FILTER [FROM *subtype*] TO *subtype*]  
    [MAXIMUM\_SEGMENT *length*];

| Argument                                              | Description                                                                                                                                                          |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>cursor</i>                                         | Name for the Blob cursor                                                                                                                                             |
| <i>column</i>                                         | Name of the Blob column                                                                                                                                              |
| <i>table</i>                                          | Table name                                                                                                                                                           |
| READ BLOB                                             | Declares a read operation on the Blob                                                                                                                                |
| INSERT BLOB                                           | Declares a write operation on the Blob                                                                                                                               |
| [FILTER [FROM <i>subtype</i> ]<br>TO <i>subtype</i> ] | Specifies optional Blob filters used to translate a Blob from one user-specified format to another; <i>subtype</i> determines which filters are used for translation |
| MAXIMUM_SEGMENT<br><i>length</i>                      | Length of the local variable to receive the Blob data after a FETCH operation                                                                                        |

**Description** Declares a cursor for reading or inserting Blob data. A Blob cursor can be associated with only one Blob column.

To read partial Blob segments when a host-language variable is smaller than the segment length of a Blob, declare the Blob cursor with the MAXIMUM\_SEGMENT clause. If *length* is less than the Blob segment, FETCH returns *length* bytes. If the same or greater, it returns a full segment (the default).

**Examples** The following embedded SQL statement declares a READ BLOB cursor and uses the MAXIMUM\_SEGMENT option:

```
EXEC SQL
 DECLARE BC CURSOR FOR
 READ BLOB JOB_REQUIREMENT FROM JOB MAXIMUM_SEGMENT 40;
```

The next embedded SQL statement declares an INSERT BLOB cursor:

```
EXEC SQL
 DECLARE BC CURSOR FOR
 INSERT BLOB JOB_REQUIREMENT INTO JOB;
```

**See also** CLOSE (BLOB), FETCH (BLOB), INSERT CURSOR (BLOB), OPEN (BLOB)

## DECLARE EXTERNAL FUNCTION

Declares an existing user-defined function (UDF) to a database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** DECLARE EXTERNAL FUNCTION *name* [*datatype*  
| CSTRING (*int*) [, *datatype* | CSTRING (*int*) ...]]  
RETURNS {*datatype* [BY VALUE] | CSTRING (*int*) | PARAMETER *n*}  
[FREE\_IT]  
ENTRY\_POINT '*entryname*' MODULE\_NAME '*modulename*';

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

**Note** Whenever a UDF returns a value by reference to dynamically allocated memory, you must declare it using the **FREE\_IT** keyword in order to free the allocated memory.

| Argument                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name</i>                   | Name of the UDF to use in SQL statements; can be different from the name of the function specified after the <b>ENTRY_POINT</b> keyword                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <i>datatype</i>               | Datatype of an input or return parameter <ul style="list-style-type: none"> <li>• All input parameters are passed to a UDF by reference</li> <li>• Return parameters can be passed by value</li> <li>• Cannot be an array element</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>CSTRING</b> ( <i>int</i> ) | Specifies a UDF that returns a null-terminated string <i>int</i> bytes in length                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>RETURNS</b>                | Specifies the return value of a function                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>BY VALUE</b>               | Specifies that a return value should be passed by value rather than by reference                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>PARAMETER</b> <i>n</i>     | <ul style="list-style-type: none"> <li>• Specifies that the <i>n</i>th input parameter is to be returned.</li> <li>• Used when the return datatype is <b>BLOB</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>FREE_IT</b>                | Frees memory of the return value after the UDF finishes running <ul style="list-style-type: none"> <li>• Use only if the memory is allocated dynamically in the UDF</li> <li>• See also <i>Language Reference</i>, Chapter 5</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| ' <i>entryname</i> '          | Quoted string that contains the function name as it is stored in the library that is referenced by the UDF                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| ' <i>modulename</i> '         | Quoted specification identifying the library that contains the UDF <ul style="list-style-type: none"> <li>• The library must reside on the same machine as the InterBase server</li> <li>• On any platform, the module can be referenced with no path name if it is in <i>&lt;InterBase_home&gt;/UDF</i> or <i>&lt;InterBase_home&gt;/intl</i></li> <li>• If the library is in a directory other than <i>&lt;InterBase_home&gt;/UDF</i> or <i>&lt;InterBase_home&gt;/intl</i>, you <i>must</i> specify its location in InterBase's configuration file (<i>ibconfig</i>) using the <b>EXTERNAL_FUNCTION_DIRECTORY</b> parameter</li> <li>• It is not necessary to supply the extension to the module name</li> </ul> |

**Description** **DECLARE EXTERNAL FUNCTION** provides information about a UDF to a database: where to find it, its name, the input parameters it requires, and the single value it returns. Each UDF in a library must be declared once to each database where it will be used. As long as the entry point and module name do not change, there is no need to redeclare a UDF, even if the function itself is modified.

*entryname* is the actual name of the function as stored in the UDF library. It does not have to match the name of the UDF as stored in the database.

**Important** The module name does not need to include a path. However, the module must either be placed in *<InterBase\_home>/UDF* or be listed in the InterBase configuration file using the `EXTERNAL_FUNCTION_DIRECTORY` parameter.

To specify a location for UDF libraries in the InterBase configuration file, enter a line of the following form for Windows platforms:

```
EXTERNAL_FUNCTION_DIRECTORY D:\Mylibraries\InterBase
```

For UNIX, the line does not include a drive letter:

```
EXTERNAL_FUNCTION_DIRECTORY \Mylibraries\InterBase
```

The InterBase configuration file is called *ibconfig* on all platforms.

**Examples** The following **isql** statement declares the `TOPS()` UDF to a database:

```
DECLARE EXTERNAL FUNCTION TOPS
 CHAR(256), INTEGER, BLOB
 RETURNS INTEGER BY VALUE
 ENTRY_POINT 'tel' MODULE_NAME 'tml';
```

This example does not need the `FREE_IT` keyword because only `cstrings`, `CHAR`, and `VARCHAR` return types require memory allocation.

The next example declares the `LOWERS()` UDF and frees the memory allocated for the return value:

```
DECLARE EXTERNAL FUNCTION LOWERS VARCHAR(256)
 RETURNS CSTRING(256) FREE_IT
 ENTRY POINT 'fn_lower' MODULE_NAME 'udflib';
```

**See also** `DROP EXTERNAL FUNCTION`

For more information about writing UDFs and for a complete list of UDFs supplied by InterBase, see “Working with UDFs and Blob Filters” in the [Developer's Guide](#).

## DECLARE FILTER

Declares an existing Blob filter to a database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** `DECLARE FILTER filter`  
           `INPUT_TYPE subtype OUTPUT_TYPE subtype`  
           `ENTRY_POINT 'entryname' MODULE_NAME 'modulename';`

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                   | Description                                                                           |
|----------------------------|---------------------------------------------------------------------------------------|
| <i>filter</i>              | Name of the filter; must be unique among filter names in the database                 |
| INPUT_TYPE <i>subtype</i>  | Specifies the Blob subtype from which data is to be converted                         |
| OUTPUT_TYPE <i>subtype</i> | Specifies the Blob subtype into which data is to be converted                         |
| ' <i>entryname</i> '       | Quoted string specifying the name of the Blob filter as stored in a linked library    |
| ' <i>modulename</i> '      | Quoted file specification identifying the object module in which the filter is stored |

**Description** DECLARE FILTER provides information about an existing Blob filter to the database: where to find it, its name, and the Blob subtypes it works with. A Blob filter is a user-written program that converts data stored in Blob columns from one subtype to another.

INPUT\_TYPE and OUTPUT\_TYPE together determine the behavior of the Blob filter. Each filter declared to the database should have a unique combination of INPUT\_TYPE and OUTPUT\_TYPE integer values. InterBase provides a built-in type of 1, for handling text. User-defined types must be expressed as negative values.

*entryname* is the name of the Blob filter stored in the library. When an application uses a Blob filter, it calls the filter function with this name.

**Example** The following **isql** statement declares a Blob filter:

```
DECLARE FILTER DESC_FILTER
 INPUT_TYPE 1
 OUTPUT_TYPE -4
 ENTRY_POINT 'desc_filter'
 MODULE_NAME 'FILTERLIB';
```

**See also** DROP FILTER

For instructions on writing Blob filters, see the [Embedded SQL Guide](#).

For more information about Blob subtypes, see the [Data Definition Guide](#).

## DECLARE STATEMENT

Identifies dynamic SQL statements before they are prepared and executed in an embedded program. Available in **gpre**.

**Syntax** DECLARE *statement* STATEMENT;

| Argument         | Description                                                                                |
|------------------|--------------------------------------------------------------------------------------------|
| <i>statement</i> | Name of a SQL variable for a user-supplied SQL statement to prepare and execute at runtime |

**Description** DECLARE STATEMENT names a SQL variable for a user-supplied SQL statement to prepare and execute at run time. DECLARE STATEMENT is not executed, so it does not produce run-time errors. The statement provides internal documentation.

**Example** The following embedded SQL statement declares Q1 to be the name of a string for preparation and execution.

```
EXEC SQL
 DECLARE Q1 STATEMENT;
```

**See also** EXECUTE, EXECUTE IMMEDIATE, PREPARE

## DECLARE TABLE

Describes the structure of a table to the preprocessor, **gpre**, before it is created with CREATE TABLE. Available in **gpre**.

**Syntax** DECLARE *table* TABLE (*table\_def*);

| Argument         | Description                                                                     |
|------------------|---------------------------------------------------------------------------------|
| <i>table</i>     | Name of the table; table names must be unique within the database               |
| <i>table_def</i> | Definition of the table; for complete table definition syntax, see CREATE TABLE |

**Description** DECLARE TABLE causes **gpre** to store a table description. You must use it if you both create and populate a table with data in the same program. If the declared table already exists in the database or if the declaration contains syntax errors, **gpre** returns an error.

When a table is referenced at run time, the column descriptions and datatypes are checked against the description stored in the database. If the table description is not in the database and the table is not declared, or if column descriptions and datatypes do not match, the application returns an error.

DECLARE TABLE can include an existing domain in a column definition, but must give the complete column description if the domain is not defined at compile time.

DECLARE TABLE cannot include integrity constraints and column attributes, even if they are present in a subsequent CREATE TABLE statement.

**Important** DECLARE TABLE cannot appear in a program that accesses multiple databases.

**Example** The following embedded SQL statements declare and create a table:

```

EXEC SQL
 DECLARE STOCK TABLE
 (MODEL SMALLINT,
 MODELNAME CHAR(10),
 ITEMID INTEGER);
EXEC SQL
 CREATE TABLE STOCK
 (MODEL SMALLINT NOT NULL UNIQUE,
 MODELNAME CHAR(10) NOT NULL,
 ITEMID INTEGER NOT NULL,
 CONSTRAINT MOD_UNIQUE UNIQUE (MODELNAME, ITEMID));

```

**See also** CREATE DOMAIN, CREATE TABLE

## DELETE

---

Removes rows in a table or in the active set of a cursor. Available in **gpre**, DSQL, and **isql**.

**Syntax** **SQL and DSQL form:**

**Important** Omit the terminating semicolon for DSQL.

```

DELETE [TRANSACTION transaction] FROM table
 {[WHERE search_condition] | WHERE CURRENT OF cursor}
 [ORDER BY order_list]
 [ROWS value [TO upper_value] [BY step_value] [PERCENT] [WITH
TIES]]];

```

*search\_condition* = Search condition as specified in SELECT.

**isql form:**

```
DELETE FROM TABLE [WHERE search_condition];
```

| Argument                          | Description                                                                                                                     |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| TRANSACTION<br><i>transaction</i> | Name of the transaction under control of which the statement is executed; SQL only                                              |
| <i>table</i>                      | Name of the table from which to delete rows                                                                                     |
| WHERE <i>search_condition</i>     | Search condition that specifies the rows to delete; without this clause, DELETE affects all rows in the specified table or view |

---

| Argument                                                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WHERE CURRENT OF <i>cursor</i>                                                                   | Specifies that the current row in the active set of <i>cursor</i> is to be deleted                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| ORDER BY <i>order_list</i>                                                                       | Specifies columns to order, either by column name or ordinal number in the query, and the sort order (ASC or DESC) for the returned rows                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| ROWS <i>value</i><br>[TO <i>upper_value</i> ]<br>[BY <i>step_value</i> ]<br>[PERCENT][WITH TIES] | <ul style="list-style-type: none"> <li>• <i>value</i> is the total number of rows to return if used by itself</li> <li>• <i>value</i> is the starting row number to return if used with TO</li> <li>• <i>value</i> is the percent if used with PERCENT</li> <li>• <i>upper_value</i> is the last row or highest percent to return</li> <li>• If <i>step_value</i> = <i>n</i>, returns every <i>n</i>th row, or <i>n</i> percent rows</li> <li>• PERCENT causes all previous ROWS values to be interpreted as percents</li> <li>• WITH TIES returns additional duplicate rows when the <i>last</i> value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with ORDER BY</li> </ul> |

DELETE specifies one or more rows to delete from a table or updatable view. DELETE is one of the database privileges controlled by the GRANT and REVOKE statements.

The TRANSACTION clause can be used in multiple transaction SQL applications to specify which transaction controls the DELETE operation. The TRANSACTION clause is not available in DSQL or **isql**.

For searched deletions, the optional WHERE clause can be used to restrict deletions to a subset of rows in the table.

**Important** Without a WHERE clause, a searched delete removes all rows from a table.

When performing a positioned delete with a cursor, the WHERE CURRENT OF clause must be specified to delete one row at a time from the active set.

**Examples** The following **isql** statement deletes all rows in a table:

```
DELETE FROM EMPLOYEE_PROJECT;
```

The next embedded SQL statement is a searched delete in an embedded application. It deletes all rows where a host-language variable equals a column value.

```
EXEC SQL
 DELETE FROM SALARY_HISTORY
 WHERE EMP_NO = :emp_num;
```

The following embedded SQL statements use a cursor and the WHERE CURRENT OF option to delete rows from CITIES with a population less than the host variable, *min\_pop*. They declare and open a cursor that finds qualifying cities, fetch rows into the cursor, and delete the current row pointed to by the cursor.

```
EXEC SQL
 DECLARE SMALL_CITIES CURSOR FOR
 SELECT CITY, STATE
 FROM CITIES
 WHERE POPULATION < :min_pop;

EXEC SQL
 OPEN SMALL_CITIES;

EXEC SQL
 FETCH SMALL_CITIES INTO :cityname, :statecode;
 WHILE (!SQLCODE)
 {EXEC SQL
 DELETE FROM CITIES
 WHERE CURRENT OF SMALL_CITIES;
 EXEC SQL
 FETCH SMALL_CITIES INTO :cityname, :statecode;}
EXEC SQL
 CLOSE SMALL_CITIES;
```

**See also** DECLARE CURSOR, FETCH, GRANT, OPEN, REVOKE, SELECT  
For more information about using cursors, see the [Embedded SQL Guide](#).

## DESCRIBE

Provides information about columns that are retrieved by a dynamic SQL (DSQL) statement, or information about the dynamic parameters that statement passes. Available in **gpre**.

**Syntax** DESCRIBE [OUTPUT | INPUT] *statement*  
          {INTO | USING} SQL DESCRIPTOR *xsqlda*;

| Argument                                    | Description                                                                                                                                         |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| OUTPUT                                      | [Default] Indicates that column information should be returned in the XSQLDA                                                                        |
| INPUT                                       | Indicates that dynamic parameter information should be stored in the XSQLDA                                                                         |
| <i>statement</i>                            | <ul style="list-style-type: none"><li>• A previously defined alias for the statement to DESCRIBE.</li><li>• Use PREPARE to define aliases</li></ul> |
| {INTO   USING} SQL DESCRIPTOR <i>xsqlda</i> | Specifies the XSQLDA to use for the DESCRIBE statement                                                                                              |

**Description** DESCRIBE has two uses:

- As a *describe output* statement, DESCRIBE stores into an XSQLDA a description of the columns that make up the select list of a previously prepared statement. If the PREPARE statement included an INTO clause, it is unnecessary to use DESCRIBE as an output statement.
- As a *describe input* statement, DESCRIBE stores into an XSQLDA a description of the dynamic parameters that are in a previously prepared statement.

DESCRIBE is one of a group of statements that process DSQL statements.

| Statement         | Purpose                                                      |
|-------------------|--------------------------------------------------------------|
| PREPARE           | Readies a DSQL statement for execution                       |
| DESCRIBE          | Fills in the XSQLDA with information about the statement     |
| EXECUTE           | Executes a previously prepared statement                     |
| EXECUTE IMMEDIATE | Prepares a DSQL statement, executes it once, and discards it |

Separate DESCRIBE statements must be issued for input and output operations. The INPUT keyword must be used to store dynamic parameter information.

**Important** When using DESCRIBE for output, if the value returned in the *sql/d* field in the XSQLDA is larger than the *sqln* field, you must:

- Allocate more storage space for XSQLVAR structures.
- Reissue the DESCRIBE statement.

**Note** The same XSQLDA structure can be used for input and output if desired.

**Example** The following embedded SQL statement retrieves information about the output of a SELECT statement:

```
EXEC SQL
 DESCRIBE Q INTO xsqllda
```

The next embedded SQL statement stores information about the dynamic parameters passed with a statement to be executed:

```
EXEC SQL
 DESCRIBE INPUT Q2 USING SQL DESCRIPTOR xsqllda;
```

**See also** EXECUTE, EXECUTE IMMEDIATE, PREPARE

For more information about DSQL programming and the XSQLDA, see the [Embedded SQL Guide](#).

## DISCONNECT

Detaches an application from a database. Available in **gpre**.

**Syntax** DISCONNECT {{ALL | DEFAULT} | *dbhandle* [, *dbhandle*] ...}};

| Argument        | Description                                                         |
|-----------------|---------------------------------------------------------------------|
| ALL DEFAULT     | Either keyword detaches all open databases                          |
| <i>dbhandle</i> | Previously declared database handle specifying a database to detach |

**Description** DISCONNECT closes a specific database identified by a database handle or all databases, releases resources used by the attached database, zeroes database handles, commits the default transaction if the **gpre -manual** option is not in effect, and returns an error if any non-default transaction is not committed.

Before using DISCONNECT, commit or roll back the transactions affecting the database to be detached.

To reattach to a database closed with DISCONNECT, reopen it with a CONNECT statement.

**Examples** The following embedded SQL statements close all databases:

```
EXEC SQL
 DISCONNECT DEFAULT;
```

```
EXEC SQL
 DISCONNECT ALL;
```

The next embedded SQL statements close the databases identified by their handles:

```
EXEC SQL
 DISCONNECT DB1;
```

```
EXEC SQL
 DISCONNECT DB1, DB2;
```

**See also** COMMIT, CONNECT, ROLLBACK, SET DATABASE

## DROP DATABASE

Deletes the currently attached database. Available in **isql**.

**Syntax** DROP DATABASE;

**Description** DROP DATABASE deletes the currently attached database, including any associated secondary, shadow, and log files. Dropping a database deletes any data it contains.

A database can be dropped by its creator, the SYSDBA user, and any users with operating system root privileges.

**Example** The following **isql** statement deletes the current database:

DROP DATABASE ;

**See also** ALTER DATABASE, CREATE DATABASE

## DROP DOMAIN

---

Deletes a domain from a database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** DROP DOMAIN *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description                |
|-------------|----------------------------|
| <i>name</i> | Name of an existing domain |

**Description** DROP DOMAIN removes an existing domain definition from a database.

If a domain is currently used in any column definition in the database, the DROP operation fails. To prevent failure, use ALTER TABLE to delete the columns based on the domain before executing DROP DOMAIN.

A domain may be dropped by its creator, the SYSDBA, and any users with operating system root privileges.

**Example** The following **isql** statement deletes a domain:

```
DROP DOMAIN COUNTRYNAME ;
```

**See also** ALTER DOMAIN, ALTER TABLE, CREATE DOMAIN

## DROP ENCRYPTION

---

Used to delete an encryption key from a database.

**Syntax** DROP ENCRYPTION *key-name*

| Argument        | Description                                                         |
|-----------------|---------------------------------------------------------------------|
| <i>key-name</i> | Specifies the name of the encryption key to drop from the database. |

**Description** An encryption key can be dropped (deleted) from the database. Only the SYSDSO can execute this command. The command fails if the encryption key is still being used to encrypt the database. If any any table columns are encrypted when "restrict" is specified, which is the default drop behavior, the command also fails. If "cascade" is specified, then all columns using that encryption are decrypted and the encryption is dropped "Restrict" and "Cascade" are the only options available for this command.

**Example** The following example uses the cascade option to decrypt all columns using the revenue\_key and to delete the key:

```
drop encryption revenue_key cascade
```

**See also** CREATE ENCRYPTION, GRANT, REVOKE, ALTER DATABASE, ALTER TABLE.  
For more information about creating and dropping encryption keys and performing encryption, see the [Data Definition Guide](#).

## DROP EXCEPTION

Deletes an exception from a database. Available in DSQL and **isql**.

**Syntax** DROP EXCEPTION *name*

| Argument    | Description                           |
|-------------|---------------------------------------|
| <i>name</i> | Name of an existing exception message |

**Description** DROP EXCEPTION removes an exception from a database.  
Exceptions used in existing procedures and triggers cannot be dropped.

**Tip** In **isql**, SHOW EXCEPTION displays a list of exceptions' *dependencies*, the procedures and triggers that use the exceptions.  
An exception can be dropped by its creator, the SYSDBA user, and any user with operating system root privileges.

**Example** This **isql** statement drops an exception:

```
DROP EXCEPTION UNKNOWN_EMP_ID;
```

**See also** ALTER EXCEPTION, ALTER PROCEDURE, ALTER TRIGGER, CREATE EXCEPTION, CREATE PROCEDURE, CREATE TRIGGER

## DROP EXTERNAL FUNCTION

Removes a user-defined function (UDF) declaration from a database. Available in **gpre**, DSQL, and **isql**.

**Syntax** DROP EXTERNAL FUNCTION *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description             |
|-------------|-------------------------|
| <i>name</i> | Name of an existing UDF |

**Description** DROP EXTERNAL FUNCTION deletes a UDF declaration from a database. Dropping a UDF declaration from a database does *not* remove it from the corresponding UDF library, but it does make the UDF inaccessible from the database. Once the definition is dropped, any applications that depend on the UDF will return run-time errors.

A UDF can be dropped by its declarer, the SYSDBA user, or any users with operating system root privileges.

**Example** This **isql** statement drops a UDF:

```
DROP EXTERNAL FUNCTION TOPS;
```

**See also** DECLARE EXTERNAL FUNCTION

## DROP FILTER

Removes a Blob filter declaration from a database. Available in **gpre**, DSQL, and **isql**.

**Syntax** DROP FILTER *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description                     |
|-------------|---------------------------------|
| <i>name</i> | Name of an existing Blob filter |

**Description** DROP FILTER removes a Blob filter declaration from a database. Dropping a Blob filter declaration from a database does *not* remove it from the corresponding Blob filter library, but it does make the filter inaccessible from the database. Once the definition is dropped, any applications that depend on the filter will return run-time errors.

DROP FILTER fails and returns an error if any processes are using the filter.

A filter can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

**Example** This **isql** statement drops a Blob filter:

```
DROP FILTER DESC_FILTER;
```

**See also** DECLARE FILTER

## DROP GENERATOR

---

Drops a generator from the database. Available in DSQL, and isql.

**Syntax** DROP GENERATOR *generator\_name*

| Argument              | Description            |
|-----------------------|------------------------|
| <i>generator_name</i> | Name of the generator. |

---

**Description** This command checks for any existing dependencies on the generator (as in triggers or UDFs) and fails if such dependencies exist. The statement fails if *generator\_name* is not the name of a generator defined on the database. An application that tries to call a deleted generator returns runtime errors.

**Note** In previous versions of InterBase that lacked the DROP GENERATOR command, users issued a SQL statement to delete the generator from the appropriate system table. This approach is strongly discouraged now that the DROP GENERATOR command is available, since modifying system tables always carries with it the possibility of rendering the entire database unusable as a result of even a slight error or miscalculation.

**See also** GEN\_ID(), CREATE GENERATOR, SET GENERATOR

## DROP INDEX

---

Removes an index from a database. Available in **gpre**, DSQL, and **isql**.

**Syntax** DROP INDEX *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description               |
|-------------|---------------------------|
| <i>name</i> | Name of an existing index |

---

**Description** DROP INDEX removes a user-defined index from a database.

An index can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

**Important** You cannot drop system-defined indexes, such as those for UNIQUE, PRIMARY KEY, and FOREIGN KEY.

An index in use is not dropped until it is no longer in use.

**Example** The following **isql** statement deletes an index:

```
DROP INDEX MINSALX;
```

**See also** ALTER INDEX, CREATE INDEX

For more information about integrity constraints and system-defined indexes, see the [Data Definition Guide](#).

## DROP JOURNAL

---

Discontinues the use of journaling and deletes existing journal files in the database.

**Syntax** DROP JOURNAL

**Description** The **DROP JOURNAL** statement discontinues the use of write-ahead logging and deletes all journal files. This operation does not delete any journal files in the journal archive but does discontinue maintenance of the journal archive. Dropping journal files requires exclusive access to the database.

**See also** CREATE JOURNAL, CREATE JOURNAL ARCHIVE, DROP JOURNAL ARCHIVE

For more information about journaling, journal files, and journal archiving, see the [Operations Guide](#).

## DROP JOURNAL ARCHIVE

---

Discontinues journal archiving on the database.

**Syntax** DROP JOURNAL ARCHIVE

**Description** DROP JOURNAL ARCHIVE disables journal archiving for the database. It causes all journal files and database file dumps to be deleted in all journal archive directories. The file system directories themselves are not deleted.

**Important** This command does not discontinue journaling and the creation of journal files.

**See also** CREATE JOURNAL ARCHIVE, DROP JOURNAL ARCHIVE, CREATE JOURNAL, DROP JOURNAL.

For more information about journaling, journal files, and journal archiving, see the [Operations Guide](#).

## DROP PROCEDURE

---

Deletes an existing stored procedure from a database. Available in DSQL, and **isql**.

**Syntax** DROP PROCEDURE *name*

| Argument    | Description                          |
|-------------|--------------------------------------|
| <i>name</i> | Name of an existing stored procedure |

- Description** DROP PROCEDURE removes an existing stored procedure definition from a database.
- Procedures used by other procedures, triggers, or views cannot be dropped. Procedures currently in use cannot be dropped.
- Tip** In **isql**, SHOW PROCEDURE displays a list of procedures' *dependencies*, the procedures, triggers, exceptions, and tables that use the procedures.
- A procedure can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.
- Example** The following **isql** statement deletes a procedure:
- ```
DROP PROCEDURE GET_EMP_PROJ;
```
- See also** ALTER PROCEDURE, CREATE PROCEDURE, EXECUTE PROCEDURE

DROP ROLE

Deletes a role from a database. Available in **gpre**, **DSQL**, and **isql**.

- Syntax** DROP ROLE *rolename*;
- Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	Description
<i>rolename</i>	Name of an existing role

- Description** DROP ROLE deletes a role that was previously created using CREATE ROLE. Any privileges that users acquired or granted through their membership in the role are revoked.
- A role can be dropped by its creator, the SYSDBA user, or any user with superuser privileges.
- Example** The following **isql** statement deletes a role from its database:
- ```
DROP ROLE administrator;
```
- See also** CREATE ROLE, GRANT, REVOKE

## DROP SHADOW

---

Deletes a shadow from a database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** DROP SHADOW *set\_num*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument       | Description                                         |
|----------------|-----------------------------------------------------|
| <i>set_num</i> | Positive integer to identify an existing shadow set |

**Description** DROP SHADOW deletes a shadow set and detaches from the shadowing process. The **isql** SHOW DATABASE command can be used to see shadow set numbers for a database.

A shadow can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

**Example** The following **isql** statement deletes a shadow set from its database:

```
DROP SHADOW 1;
```

**See also** CREATE SHADOW

## DROP TABLE

---

Removes a table from a database. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** DROP TABLE *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description               |
|-------------|---------------------------|
| <i>name</i> | Name of an existing table |

**Description** DROP TABLE removes a table's data, metadata, and indexes from a database. It also drops any triggers that reference the table.

A table referenced in a SQL expression, a view, integrity constraint, or stored procedure cannot be dropped. A table used by an active transaction is not dropped until it is free.

**Note** When used to drop an external table, DROP TABLE only removes the table definition from the database. The external file is not deleted.

A table can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

**Example** The following embedded SQL statement drops a table:

```
EXEC SQL
 DROP TABLE COUNTRY;
```

**See also** ALTER TABLE, CREATE TABLE

## DROP TRIGGER

---

Deletes an existing user-defined trigger from a database. Available in DSQL and **isql**.

**Syntax** DROP TRIGGER *name*

| Argument    | Description                 |
|-------------|-----------------------------|
| <i>name</i> | Name of an existing trigger |

---

**Description** DROP TRIGGER removes a user-defined trigger definition from the database. System-defined triggers, such as those created for CHECK constraints, cannot be dropped. Use ALTER TABLE to drop the CHECK clause that defines the trigger.

Triggers used by an active transaction cannot be dropped until the transaction is terminated.

A trigger can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

**Tip** To inactivate a trigger temporarily, use ALTER TRIGGER and specify INACTIVE in the header.

**Example** The following **isql** statement drops a trigger:

```
DROP TRIGGER POST_NEW_ORDER;
```

**See also** ALTER TRIGGER, CREATE TRIGGER

## DROP USER

---

Deletes an existing user from an embedded user authentication database. Available in DSQL, and **isql**.

**Syntax** DROP USER *name*

## DROP VIEW

---

Removes a view definition from the database. Available in **gpre**, DSQL, and **isql**.

**Syntax** DROP VIEW *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description                                 |
|-------------|---------------------------------------------|
| <i>name</i> | Name of an existing view definition to drop |

**Description** DROP VIEW enables a view's creator to remove a view definition from the database if the view is not used in another view, stored procedure, or CHECK constraint definition.

A view can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

**Example** The following **isql** statement removes a view definition:

```
DROP VIEW PHONE_LIST;
```

**See also** CREATE VIEW

## END DECLARE SECTION

Identifies the end of a host-language variable declaration section. Available in **gpre**.

**Syntax** END DECLARE SECTION;

**Description** END DECLARE SECTION is used in embedded SQL applications to identify the end of host-language variable declarations for variables used in subsequent SQL statements.

**Example** The following embedded SQL statements declare a section, and single host-language variable:

```
EXEC SQL
 BEGIN DECLARE SECTION;
 BASED_ON EMPLOYEE.SALARY salary;
EXEC SQL
 END DECLARE SECTION;
```

**See also** BASED ON, BEGIN DECLARE SECTION

## EVENT INIT

Registers interest in one or more events with the InterBase event manager. Available in **gpre**.

**Syntax** EVENT INIT *request\_name* [*dbhandle*]  
 [('string' | :*variable* [, 'string' | :*variable* ...]);

| Argument            | Description                                                                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>request_name</i> | Application event handle                                                                                                                                              |
| <i>dbhandle</i>     | Specifies the database to examine for occurrences of the events; if omitted, <i>dbhandle</i> defaults to the database named in the most recent SET DATABASE statement |
| ' <i>string</i> '   | Unique name identifying an event associated with <i>event_name</i>                                                                                                    |
| : <i>variable</i>   | Host-language character array containing a list of event names to associate with                                                                                      |

**Description** EVENT INIT is the first step in the InterBase two-part synchronous event mechanism:

- 1 EVENT INIT registers an application's interest in an event.
- 2 EVENT WAIT causes the application to wait until notified of the event's occurrence.

EVENT INIT registers an application's interest in a list of events in parentheses. The list should correspond to events posted by stored procedures or triggers in the database. If an application registers interest in multiple events with a single EVENT INIT, then when one of those events occurs, the application must determine which event occurred.

Events are posted by a POST\_EVENT call within a stored procedure or trigger.

The event manager keeps track of events of interest. At commit time, when an event occurs, the event manager notifies interested applications.

**Example** The following embedded SQL statement registers interest in an event:

```
EXEC SQL
 EVENT INIT ORDER_WAIT EMPDB ('new_order');
```

**See also** CREATE PROCEDURE, CREATE TRIGGER, EVENT WAIT, SET DATABASE

For more information about events, see the [Embedded SQL Guide](#).

## EVENT WAIT

Causes an application to wait until notified of an event's occurrence. Available in **gpre**.

**Syntax** EVENT WAIT *request\_name*;

| Argument            | Description                                                          |
|---------------------|----------------------------------------------------------------------|
| <i>request_name</i> | Application event handle declared in a previous EVENT INIT statement |

**Description** EVENT WAIT is the second step in the InterBase two-part synchronous event mechanism. After a program registers interest in an event, EVENT WAIT causes the process running the application to sleep until the event of interest occurs.

**Examples** The following embedded SQL statements register an application event name and indicate the program is ready to receive notification when the event occurs:

```
EXEC SQL
 EVENT INIT ORDER_WAIT EMPDB ('new_order');

EXEC SQL
 EVENT WAIT ORDER_WAIT;
```

**See also** EVENT INIT

For more information about events, see the [Embedded SQL Guide](#).

## EXECUTE

Executes a previously prepared dynamic SQL (DSQL) statement. Available in **gpre**.

**Syntax** EXECUTE [TRANSACTION *transaction*] *statement*  
[USING SQL DESCRIPTOR *xsqlda*] [INTO SQL DESCRIPTOR *xsqlda*];

| Argument                       | Description                                                                                                          |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------|
| TRANSACTION <i>transaction</i> | Specifies the transaction under which execution occurs                                                               |
| <i>statement</i>               | Alias of a previously prepared statement to execute                                                                  |
| USING SQL DESCRIPTOR           | Specifies that values corresponding to the prepared statement's parameters should be taken from the specified XSQLDA |
| INTO SQL DESCRIPTOR            | Specifies that return values from the executed statement should be stored in the specified XSQLDA                    |
| <i>xsqlda</i>                  | XSQLDA host-language variable                                                                                        |

**Description** EXECUTE carries out a previously prepared DSQL statement. It is one of a group of statements that process DSQL statements.

| Statement         | Purpose                                                      |
|-------------------|--------------------------------------------------------------|
| PREPARE           | Readies a DSQL statement for execution                       |
| DESCRIBE          | Fills in the XSQLDA with information about the statement     |
| EXECUTE           | Executes a previously prepared statement                     |
| EXECUTE IMMEDIATE | Prepares a DSQL statement, executes it once, and discards it |

Before a statement can be executed, it must be prepared using the PREPARE statement. The statement can be any SQL data definition, manipulation, or transaction management statement. Once it is prepared, a statement can be executed any number of times.

The TRANSACTION clause can be used in SQL applications running multiple, simultaneous transactions to specify which transaction controls the EXECUTE operation.

USING DESCRIPTOR enables EXECUTE to extract a statement's parameters from an XSQLDA structure previously loaded with values by the application. It need only be used for statements that have dynamic parameters.

INTO DESCRIPTOR enables EXECUTE to store return values from statement execution in a specified XSQLDA structure for application retrieval. It need only be used for DSQL statements that return values.

**Note** If an EXECUTE statement provides both a USING DESCRIPTOR clause and an INTO DESCRIPTOR clause, then two XSQLDA structures must be provided.

**Example** The following embedded SQL statement executes a previously prepared DSQL statement:

```
EXEC SQL
 EXECUTE DOUBLE_SMALL_BUDGET;
```

The next embedded SQL statement executes a previously prepared statement with parameters stored in an XSQLDA:

```
EXEC SQL
 EXECUTE Q USING DESCRIPTOR xsqllda;
```

The following embedded SQL statement executes a previously prepared statement with parameters in one XSQLDA, and produces results stored in a second XSQLDA:

```
EXEC SQL
 EXECUTE Q USING DESCRIPTOR xsqllda_1 INTO DESCRIPTOR
 xsqllda_2;
```

**See also** DESCRIBE, EXECUTE IMMEDIATE, PREPARE

For more information about DSQL programming and the XSQLDA, see the [Embedded SQL Guide](#).

## EXECUTE IMMEDIATE

---

Prepares a dynamic SQL (DSQL) statement, executes it once, and discards it. Available in **gpre**.

**Syntax** EXECUTE IMMEDIATE [TRANSACTION *transaction*]  
 {:*variable* | '*string*'} [USING SQL DESCRIPTOR *xsqllda*];

| Argument                       | Description                                                                                                 |
|--------------------------------|-------------------------------------------------------------------------------------------------------------|
| TRANSACTION <i>transaction</i> | Specifies the transaction under which execution occurs                                                      |
| : <i>variable</i>              | Host variable containing the SQL statement to execute                                                       |
| ' <i>string</i> '              | A string literal containing the SQL statement to execute                                                    |
| USING SQL DESCRIPTOR           | Specifies that values corresponding to the statement's parameters should be taken from the specified XSQLDA |
| <i>xsqllda</i>                 | XSQLDA host-language variable                                                                               |

**Description** EXECUTE IMMEDIATE prepares a DSQL statement stored in a host-language variable or in a literal string, executes it once, and discards it. To prepare and execute a DSQL statement for repeated use, use PREPARE and EXECUTE instead of EXECUTE IMMEDIATE.

The TRANSACTION clause can be used in SQL applications running multiple, simultaneous transactions to specify which transaction controls the EXECUTE IMMEDIATE operation.

The SQL statement to execute must be stored in a host variable or be a string literal. It can contain any SQL data definition statement or data manipulation statement that does not return output.

USING DESCRIPTOR enables EXECUTE IMMEDIATE to extract the values of a statement's parameters from an XSQLDA structure previously loaded with appropriate values.

**Example** The following embedded SQL statement prepares and executes a statement in a host variable:

```
EXEC SQL
 EXECUTE IMMEDIATE :insert_date;
```

**See also** DESCRIBE, EXECUTE IMMEDIATE, PREPARE

For more information about DSQL programming and the XSQLDA, see the [Embedded SQL Guide](#).

## EXECUTE PROCEDURE

Calls a stored procedure. Available in **gpre**, DSQL, and **isql**.

**Syntax SQL form:**

```
EXECUTE PROCEDURE [TRANSACTION transaction]
 name [:param [[INDICATOR]:indicator]]
 [, :param [[INDICATOR]:indicator] ...]
 [RETURNING_VALUES :param [[INDICATOR]:indicator]
 [, :param [[INDICATOR]:indicator] ...]];
```

**DSQL form:**

```
EXECUTE PROCEDURE name [param [, param ...]]
 [RETURNING_VALUES param [, param ...]]
```

**isql form:**

```
EXECUTE PROCEDURE name [param [, param ...]]
```

| Argument                          | Description                                                     |
|-----------------------------------|-----------------------------------------------------------------|
| TRANSACTION <i>transaction</i>    | Specifies the transaction under which execution occurs          |
| <i>name</i>                       | Name of an existing stored procedure in the database            |
| <i>param</i>                      | Input or output parameter; can be a host variable or a constant |
| RETURNING_VALUES:<br><i>param</i> | Host variable which takes the values of an output parameter     |
| [INDICATOR] : <i>indicator</i>    | Host variable for indicating NULL or unknown values             |

**Description** EXECUTE PROCEDURE calls the specified stored procedure. If the procedure requires input parameters, they are passed as host-language variables or as constants. If a procedure returns output parameters to a SQL program, host variables must be supplied in the RETURNING\_VALUES clause to hold the values returned.

In **isql**, do not use the RETURN clause or specify output parameters. **isql** will automatically display return values.

**Note** In DSQL, an EXECUTE PROCEDURE statement requires an input descriptor area if it has input parameters and an output descriptor area if it has output parameters.

In embedded SQL, input parameters and return values may have associated indicator variables for tracking NULL values. Indicator variables are integer values that indicate unknown or NULL values of return values.

An indicator variable that is less than zero indicates that the parameter is unknown or NULL. An indicator variable that is zero or greater indicates that the associated parameter is known and not NULL.

**Examples** The following embedded SQL statement demonstrates how the executable procedure, DEPT\_BUDGET, is called from embedded SQL with literal parameters:

```
EXEC SQL
 EXECUTE PROCEDURE DEPT_BUDGET 100 RETURNING_VALUES :sumb;
```

The next embedded SQL statement calls the same procedure using a host variable instead of a literal as the input parameter:

```
EXEC SQL
 EXECUTE PROCEDURE DEPT_BUDGET :rdno RETURNING_VALUES
 :sumb;
```

**See also** ALTER PROCEDURE, CREATE PROCEDURE, DROP PROCEDURE

For more information about indicator variables, see the [Embedded SQL Guide](#).

## EXTRACT()

Extracts date and time information from DATE, TIME, and TIMESTAMP values. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** EXTRACT (*part* FROM *value*)

| Argument     | Description                                                                                                         |
|--------------|---------------------------------------------------------------------------------------------------------------------|
| <i>part</i>  | YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, WEEKDAY, or YEARDAY; see the table below for datatypes and ranges of values |
| <i>value</i> | DATE, TIME, or TIMESTAMP value                                                                                      |

**Description** The value passed to the EXTRACT() expression must be a DATE, a TIME, or a TIMESTAMP. Extracting a part that does not exist in a datatype results in an error. For example, a statement such as tEXTRACT (YEAR from aTime) would fail.

**Note** The datatype of *part* depends on which part is extracted.

**Table 2.9** EXTRACT() date and time parts

| Part extracted | Datatype     | Range                                 |
|----------------|--------------|---------------------------------------|
| YEAR           | SMALLINT     | 0–5400                                |
| MONTH          | SMALLINT     | 1–12                                  |
| DAY            | SMALLINT     | 1–31                                  |
| HOUR           | SMALLINT     | 0–23                                  |
| MINUTE         | SMALLINT     | 0–59                                  |
| SECOND         | DECIMAL(6,4) | 0–59.9999                             |
| WEEKDAY        | SMALLINT     | 0–6<br>(0 = Sunday, 1 = Monday, etc.) |
| YEARDAY        | SMALLINT     | 0–365                                 |

**Example** EXTRACT(HOUR FROM StartTime);

## FETCH

---

Retrieves the next available row from the active set of an opened cursor. Available in **gpre** and DSQL.

### Syntax **SQL form:**

```
FETCH cursor
 [INTO :hostvar [[INDICATOR] :indvar]
 [, :hostvar [[INDICATOR] :indvar] ...];
```

### DSQL form:

```
FETCH cursor {INTO | USING} SQL DESCRIPTOR xsqlda
```

**Blob form:** See FETCH (BLOB).

| Argument                    | Description                                                                                                                                                                                                                               |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>cursor</i>               | Name of the opened cursor from which to fetch rows                                                                                                                                                                                        |
| : <i>hostvar</i>            | A host-language variable for holding values retrieved with the FETCH <ul style="list-style-type: none"> <li>• Optional if FETCH gets rows for DELETE or UPDATE</li> <li>• Required if row is displayed before DELETE or UPDATE</li> </ul> |
| : <i>indvar</i>             | Indicator variable for reporting that a column contains an unknown or NULL value                                                                                                                                                          |
| [INTO USING] SQL DESCRIPTOR | Specifies that values should be returned in the specified XSQLDA                                                                                                                                                                          |
| <i>xsqlda</i>               | XSQLDA host-language variable                                                                                                                                                                                                             |

---

**Description** FETCH retrieves one row at a time into a program from the active set of a cursor. The first FETCH operates on the first row of the active set. Subsequent FETCH statements advance the cursor sequentially through the active set one row at a time until no more rows are found and SQLCODE is set to 100.

A cursor is a one-way pointer into the ordered set of rows retrieved by the select expression in the DECLARE CURSOR statement. A cursor enables sequential access to retrieved rows. There are four related cursor statements:

| Stage | Statement      | Purpose                                                                                                                                   |
|-------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | DECLARE CURSOR | Declare the cursor; the SELECT statement determines rows retrieved for the cursor                                                         |
| 2     | OPEN           | Retrieve the rows specified for retrieval with DECLARE CURSOR; the resulting rows become the cursor's <i>active set</i>                   |
| 3     | FETCH          | Retrieve the current row from the active set, starting with the first row; subsequent FETCH statements advance the cursor through the set |
| 4     | CLOSE          | Close the cursor and release system resources                                                                                             |

The number, size, datatype, and order of columns in a FETCH must be the same as those listed in the query expression of its matching DECLARE CURSOR statement. If they are not, the wrong values can be assigned.

**Examples** The following embedded SQL statement fetches a column from the active set of a cursor:

```
EXEC SQL
 FETCH PROJ_CNT INTO :department, :hcmt;
```

**See also** CLOSE, DECLARE CURSOR, DELETE, FETCH (BLOB), OPEN

For more information about cursors and XSQLDA, see the [Embedded SQL Guide](#).

## FETCH (BLOB)

Retrieves the next available segment of a Blob column and places it in the specified local buffer. Available in **gpre**.

**Syntax** `FETCH cursor INTO  
[:buffer [[INDICATOR] :segment_length];`

| Argument                | Description                                                                                                                             |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>cursor</i>           | Name of an open Blob cursor from which to retrieve segments                                                                             |
| : <i>buffer</i>         | Host-language variable for holding segments fetched from the Blob column; user must declare the buffer before fetching segments into it |
| INDICATOR               | Optional keyword indicating that a host-language variable for indicating the number of bytes returned by the FETCH follows              |
| : <i>segment_length</i> | Host-language variable used to indicate the number of bytes returned by the FETCH                                                       |

**Description** FETCH retrieves the next segment from a Blob and places it into the specified buffer.

The host variable, *segment\_length*, indicates the number of bytes fetched. This is useful when the number of bytes fetched is smaller than the host variable, for example, when fetching the last portion of a Blob.

FETCH can return two SQLCODE values:

- SQLCODE = 100 indicates that there are no more Blob segments to retrieve.
- SQLCODE = 101 indicates that a partial segment was retrieved and placed in the local buffer variable.

**Note** To ensure that a host variable buffer is large enough to hold a Blob segment buffer during FETCH operations, use the SEGMENT option of the BASED ON statement.

To ensure that a host variable buffer is large enough to hold a Blob segment buffer during FETCH operations, use the SEGMENT option of the BASED ON statement.

**Example** The following code, from an embedded SQL application, performs a BLOB FETCH:

```
while (SQLCODE != 100)
{
 EXEC SQL
 OPEN BLOB_CUR USING :blob_id;
 EXEC SQL
 FETCH BLOB_CUR INTO :blob_segment :blob_seg_len;
 while (SQLCODE !=100 || SQLCODE == 101)
 {
 blob_segment[blob_seg_len + 1] = '\0';
 printf("%*.*s",blob_seg_len,blob_seg_len,blob_segment);
 blob_segment[blob_seg_len + 1] = ' ';
 EXEC SQL
 FETCH BLOB_CUR INTO :blob_segment :blob_seg_len;
 }
 . . .
}
```

**See also** BASED ON, CLOSE (BLOB), DECLARE CURSOR (BLOB), INSERT CURSOR (BLOB), OPEN (BLOB)

## GEN\_ID( )

Produces a system-generated integer value. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** *gen\_id (generator, step)*

| Argument         | Description                                                                                                                                                |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>generator</i> | Name of an existing generator                                                                                                                              |
| <i>step</i>      | Integer or expression specifying the increment for increasing or decreasing the current generator value. Values can range from $-(2^{63})$ to $2^{63} - 1$ |

**Description** The GEN\_ID() function:

- 1 Increments the current value of the specified generator by *step*.
- 2 Returns the new value of the specified generator.

GEN\_ID() is useful for automatically producing unique values that can be inserted into a UNIQUE or PRIMARY KEY column. To insert a generated number in a column, write a trigger, procedure, or SQL statement that calls GEN\_ID().

**Note** A generator is initially created with CREATE GENERATOR. By default, the value of a generator begins at zero. It can be set to a different value with SET GENERATOR.

**Examples** The following **isql** trigger definition includes a call to GEN\_ID():

```
CREATE TRIGGER CREATE_EMPNO FOR EMPLOYEES
 BEFORE INSERT
 POSITION 0
 AS BEGIN
 NEW.EMPNO = GEN_ID (EMPNO_GEN, 1);
 END
```

The first time the trigger fires, NEW.EMPNO is set to 1. Each subsequent firing increments NEW.EMPNO by 1.

**See also** CREATE GENERATOR, SET GENERATOR

## GRANT

Assigns privileges to users for specified database objects. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** GRANT *privileges* ON [TABLE] {*tablename* | *viewname*}  
           TO {*object*|*userlist* [WITH GRANT OPTION]]GROUP *UNIX\_group*}  
 | EXECUTE ON PROCEDURE *procname* TO {*object* | *userlist*}  
 | *role\_granted* TO {PUBLIC | *role\_grantee\_list*}[WITH ADMIN OPTION];

*privileges* = ALL [PRIVILEGES] | *privilege\_list*

*privilege\_list* = {  
     SELECT  
     | DELETE  
     | INSERT  
     | ENCRYPT ON ENCRYPTION  
     | DECRYPT  
     | UPDATE [(*col* [, *col* ...])]  
     | REFERENCES [(*col* [, *col* ...])]  
 }[, *privilege\_list* ...]

*object* = {  
     PROCEDURE *procname*  
     | TRIGGER *trigname*  
     | VIEW *viewname*

```

 | PUBLIC
 }[, object ...]
userlist = {
 [USER] username
 | rolename
 | UNIX_user
}[,userlist ...]
role_granted = rolename [, rolename ...]
role_grantee_list = [USER] username [, [USER] username ...]

```

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                 | Description                                                                                                                       |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>privilege_list</i>    | Name of privilege to be granted; valid options are SELECT, DELETE, INSERT, UPDATE, ENCRYPT ON ENCRYPTION, DECRYPT, and REFERENCES |
| <i>col</i>               | Column to which the granted privileges apply                                                                                      |
| <i>tablename</i>         | Name of an existing table for which granted privileges apply                                                                      |
| <i>viewname</i>          | Name of an existing view for which granted privileges apply                                                                       |
| GROUP <i>unix_group</i>  | On a UNIX system, the name of a group defined in <i>/etc/group</i>                                                                |
| <i>object</i>            | Name of an existing procedure, trigger, or view; PUBLIC is also a permitted value                                                 |
| <i>userlist</i>          | A user in the InterBase security database ( <i>admin.ib</i> by default) or a rolename created with CREATE ROLE                    |
| WITH GRANT OPTION        | Passes GRANT authority for privileges listed in the GRANT statement to <i>userlist</i>                                            |
| <i>rolename</i>          | An existing role created with the CREATE ROLE statement                                                                           |
| <i>role_grantee_list</i> | A list of users to whom <i>rolename</i> is granted; users must be in the InterBase security database                              |
| WITH ADMIN OPTION        | Passes grant authority for roles listed to <i>role_grantee_list</i>                                                               |

**Description** GRANT assigns privileges and roles for database objects to users, roles, or other database objects. When an object is first created, only its creator has privileges to it and only its creator can GRANT privileges for it to other users or objects.

The following table summarizes available privileges:

| Privilege             | Enables users to ...                                                                                                                                                                       |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ALL                   | Perform SELECT, DELETE, INSERT, UPDATE, and REFERENCES                                                                                                                                     |
| SELECT                | Retrieve rows from a table or view                                                                                                                                                         |
| DELETE                | Remove rows from a table or view                                                                                                                                                           |
| DECRYPT               | After encrypting a column, the database owner or the individual table owner can grant decrypt permission to users who need to access the values in an encrypted column.                    |
| ENCRYPT ON ENCRYPTION | Enables the database owner or individual table owner to use a specific encryption key to encrypt a database or column. Only the SYSDSO (Data Security Owner) can grant encrypt permission. |
| INSERT                | Store new rows in a table or view                                                                                                                                                          |
| UPDATE                | Change the current value in one or more columns in a table or view; can be restricted to a specified subset of columns                                                                     |
| EXECUTE               | Execute a stored procedure                                                                                                                                                                 |
| REFERENCES            | Reference the specified columns with a foreign key; at a minimum, this must be granted to all the columns of the primary key if it is granted at all                                       |

**Note** ALL does not include REFERENCES in code written for InterBase 4.0 or earlier.

- To access a table or view, a user or object needs the appropriate SELECT, INSERT, UPDATE, DELETE, or REFERENCES privileges for that table or view. SELECT, INSERT, UPDATE, DELETE, and REFERENCES privileges can be assigned as a unit with ALL.
- A user or object must have EXECUTE privilege to call a stored procedure in an application.
- For more information about the GRANT ENCRYPT ON ENCRYPTION and GRANT DECRYPT permissions, see Chapter 13, “Encrypting Your Data” in the [Data Definition Guide](#).
- To grant privileges to a group of users, create a role using CREATE ROLE. Then use GRANT *privilege* TO *rolename* to assign the desired privileges to that role and use GRANT *rolename* TO *user* to assign that role to users. Users can be added or removed from a role on a case-by-case basis using GRANT and REVOKE. A user must specify the role at connection time to actually have those privileges. See “ANSI SQL 3 roles” in the [Operations Guide](#) for more information about invoking a role when connecting to a database.
- On UNIX systems, privileges can be granted to groups listed in */etc/groups* and to any UNIX user listed in */etc/passwd* on both the client and server, as well as to individual users and to roles.

- To allow another user to reference a columns from a foreign key, grant REFERENCES privileges on the primary key table or on the table's primary key columns to the owner of the foreign key table. You must also grant REFERENCES or SELECT privileges on the primary key table to any user who needs to write to the foreign key table.

**Tip** Make it easy: if read security is not an issue, GRANT REFERENCES on the primary key table to PUBLIC.

- If you grant the REFERENCES privilege, it must, at a minimum, be granted to all columns of the primary key. When REFERENCES is granted to the entire table, columns that are not part of the primary key are not affected in any way.
- When a user defines a foreign key constraint on a table owned by someone else, InterBase checks that the user has REFERENCES privileges on the referenced table.
- The privilege is used at runtime to verify that a value entered in a foreign key field is contained in the primary key table.
- You can grant REFERENCES privileges to roles.
- To give users permission to grant privileges to other users, provide a *userlist* that includes the WITH GRANT OPTION. Users can grant to others only the privileges that they themselves possess.
- To grant privileges to all users, specify PUBLIC in place of a list of user names. Specifying PUBLIC grants privileges only to users, not to database objects.

Privileges can be removed only by the user who assigned them, using REVOKE. If ALL privileges are assigned, then ALL privileges must be revoked. If privileges are granted to PUBLIC, they can be removed only for PUBLIC.

**Examples** The following **isql** statement grants SELECT and DELETE privileges to a user. The WITH GRANT OPTION gives the user GRANT authority.

```
GRANT SELECT, DELETE ON COUNTRY TO CHLOE WITH GRANT OPTION;
```

The next embedded SQL statement, from an embedded program, grants SELECT and UPDATE privileges to a procedure for a table:

```
EXEC SQL
 GRANT SELECT, UPDATE ON JOB TO PROCEDURE GET_EMP_PROJ;
```

This embedded SQL statement grants EXECUTE privileges for a procedure to another procedure and to a user:

```
EXEC SQL
 GRANT EXECUTE ON PROCEDURE GET_EMP_PROJ
 TO PROCEDURE ADD_EMP_PROJ, LUIS;
```

The following example creates a role called "administrator", grants UPDATE privileges on table1 to that role, and then grants the role to user1, user2, and user3. These users then have UPDATE and REFERENCES privileges on table1.

```
CREATE ROLE administrator;
GRANT UPDATE ON table1 TO administrator;
GRANT administrator TO user1, user2, user3;
```

**See also** REVOKE  
For more information about privileges, see the [Data Definition Guide](#)..

# INSERT

Adds one or more new rows to a specified table. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** `INSERT [TRANSACTION transaction] INTO object [(col [, col ...])] {VALUES (val [, val ...]) | select_expr};`

*object* = *tablename* | *viewname*

*val* = {*:variable* | *constant* | *expr*  
| *function* | *udf* ([*val* [, *val* ...]])  
| NULL | USER | RDB\$DB\_KEY | ?  
} [*COLLATE collation*]

*constant* = *num* | '*string*' | *charsetname* '*string*'

*function* = CAST (*val* AS *datatype*)  
| UPPER (*val*)  
| GEN\_ID (*generator*, *val*)

| Argument           | Description                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>expr</i>        | A valid SQL expression that results in a single column value                                                                          |
| <i>select_expr</i> | A SELECT that returns zero or more rows and where the number of columns in each row is the same as the number of items to be inserted |

- Notes on the INSERT statement**
- In SQL and **isql**, you cannot use *val* as a parameter placeholder (like “?”).
  - In DSQL and **isql**, *val* cannot be a variable.
  - You cannot specify a COLLATE clause for Blob columns.

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                                | Description                                                                                                  |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------|
| TRANSACTION <i>transaction</i>          | Name of the transaction that controls the execution of the INSERT                                            |
| INTO <i>object</i>                      | Name of an existing table or view into which to insert data                                                  |
| <i>col</i>                              | Name of an existing column in a table or view into which to insert values                                    |
| VALUES ( <i>val</i> [, <i>val</i> ...]) | Lists values to insert into the table or view; values must be listed in the same order as the target columns |
| <i>select_expr</i>                      | Query that returns row values to insert into target columns                                                  |

**Description** INSERT stores one or more new rows of data in an existing table or view. INSERT is one of the database privileges controlled by the GRANT and REVOKE statements.

Values are inserted into a row in column order unless an optional list of target columns is provided. If the target list of columns is a subset of available columns, default or NULL values are automatically stored in all unlisted columns.

If the optional list of target columns is omitted, the VALUES clause must provide values to insert into all columns in the table.

To insert a single row of data, the VALUES clause should include a specific list of values to insert.

To insert multiple rows of data, specify a *select\_expr* that retrieves existing data from another table to insert into this one. The selected columns must correspond to the columns listed for insert.

**Important** It is legal to select from the same table into which insertions are made, but this practice is not advised because it may result in infinite row insertions.

The TRANSACTION clause can be used in multiple transaction SQL applications to specify which transaction controls the INSERT operation. The TRANSACTION clause is not available in DSQL or **isql**.

**Examples** The following statement, from an embedded SQL application, adds a row to a table, assigning values from host-language variables to two columns:

```
EXEC SQL
 INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID)
 VALUES (:emp_no, :proj_id);
```

The next **isql** statement specifies values to insert into a table with a SELECT statement:

```
INSERT INTO PROJECTS
 SELECT * FROM NEW_PROJECTS
 WHERE NEW_PROJECTS.START_DATE > '6-JUN-1994';
```

**See also** GRANT, REVOKE, SET TRANSACTION, UPDATE

## INSERT CURSOR (BLOB)

---

Inserts data into a Blob cursor in units of a Blob segment-length or less in size. Available in **gpre**.

**Syntax**    `INSERT CURSOR cursor  
              VALUES (:buffer [INDICATOR] :bufferlen);`

| Argument           | Description                                                            |
|--------------------|------------------------------------------------------------------------|
| <i>cursor</i>      | Name of the Blob cursor                                                |
| VALUES             | Clause containing the name and length of the buffer variable to insert |
| : <i>buffer</i>    | Name of host-variable buffer containing information to insert          |
| INDICATOR          | Indicates that the length of data placed in the buffer follows         |
| : <i>bufferlen</i> | Length, in bytes, of the buffer to insert                              |

---

**Description**    INSERT CURSOR writes Blob data into a column. Data is written in units equal to or less than the segment size for the Blob. Before inserting data into a Blob cursor:

- Declare a local variable, *buffer*, to contain the data to be inserted.
- Declare the length of the variable, *bufferlen*.
- Declare a Blob cursor for INSERT and open it.

Each INSERT into the Blob column inserts the current contents of *buffer*. Between statements fill *buffer* with new data. Repeat the INSERT until each existing *buffer* is inserted into the Blob.

**Important**    INSERT CURSOR requires the INSERT privilege, a table privilege controlled by the GRANT and REVOKE statements.

**Example**    The following embedded SQL statement shows an insert into the Blob cursor:

```
EXEC SQL
 INSERT CURSOR BC VALUES (:line INDICATOR :len);
```

**See also**    CLOSE (BLOB), DECLARE CURSOR (BLOB), FETCH (BLOB), OPEN (BLOB)

## MAX( )

---

Retrieves the maximum value in a column. Available in **gpre**, **DSQL**, and **isql**.

**Syntax**    `MAX ([ALL] val | DISTINCT val)`

| Argument   | Description                                                                            |
|------------|----------------------------------------------------------------------------------------|
| ALL        | Searches all values in a column                                                        |
| DISTINCT   | Eliminates duplicate values before finding the largest                                 |
| <i>val</i> | A column, constant, host-language variable, expression, non-aggregate function, or UDF |

**Description** MAX() is an aggregate function that returns the largest value in a specified column, excluding NULL values. If the number of qualifying rows is zero, MAX() returns a NULL value.

When MAX() is used on a CHAR, VARCHAR, or Blob text column, the largest value returned varies depending on the character set and collation in use for the column. A default character set can be specified for an entire database with the DEFAULT CHARACTER SET clause in CREATE DATABASE, or specified at the column level with the COLLATE clause in CREATE TABLE.

**Example** The following embedded SQL statement demonstrates the use of SUM(), AVG(), MIN(), and MAX():

```
EXEC SQL
 SELECT SUM (BUDGET), AVG (BUDGET), MIN (BUDGET), MAX
 (BUDGET)
 FROM DEPARTMENT
 WHERE HEAD_DEPT = :head_dept
 INTO :tot_budget, :avg_budget, :min_budget, :max_budget;
```

**See also** AVG(), COUNT(), CREATE DATABASE, CREATE TABLE, MIN(), SUM()

## MIN( )

Retrieves the minimum value in a column. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** MIN ([ALL] *val* | DISTINCT *val*)

| Argument   | Description                                                                            |
|------------|----------------------------------------------------------------------------------------|
| ALL        | Searches all values in a column                                                        |
| DISTINCT   | Eliminates duplicate values before finding the smallest                                |
| <i>val</i> | A column, constant, host-language variable, expression, non-aggregate function, or UDF |

**Description** MIN() is an aggregate function that returns the smallest value in a specified column, excluding NULL values. If the number of qualifying rows is zero, MIN() returns a NULL value.

When MIN() is used on a CHAR, VARCHAR, or Blob text column, the smallest value returned varies depending on the character set and collation in use for the column. Use the DEFAULT CHARACTER SET clause in CREATE DATABASE to specify a default character set for an entire database, or the COLLATE clause in CREATE TABLE to specify a character set at the column level.

**Example** The following embedded SQL statement demonstrates the use of SUM(), AVG(), MIN(), and MAX():

```
EXEC SQL
 SELECT SUM (BUDGET), AVG (BUDGET), MIN (BUDGET), MAX
 (BUDGET)
 FROM DEPARTMENT
 WHERE HEAD_DEPT = :head_dept
 INTO :tot_budget, :avg_budget, :min_budget, :max_budget;
```

**See also** AVG(), COUNT(), CREATE DATABASE, CREATE TABLE, MAX(), SUM()

## NULLIF()

---

The NULLIF function returns a null value if the arguments are equal, otherwise it returns the value of the first argument.

**Syntax** NULLIF (<expression1>, <expression2>)

**Description** The COALESCE and NULLIF expressions are common, shorthand forms of use for the CASE expression involving the NULL state. A COALESCE expression consists of a list of value expressions. It evaluates to the first value expression in the list that evaluates to non-NULL. If none of the value expressions in the list evaluates to non-NULL then the COALESCE expression evaluates to NULL.

The NULLIF expression consists of a list of two value expressions. If the two expressions are unequal then the NULLIF expression evaluates to the first value expression in the list. Otherwise, it evaluates to NULL.

**Example** The following example demonstrates the use of CASE using the sample employee.ib database:

```
select NULLIF(department, head_dept) from department
```

## OPEN

---

Retrieve specified rows from a cursor declaration. Available in **gpre** and DSQL.

**Syntax** **SQL form:**

```
OPEN [TRANSACTION transaction] cursor;
```

**DSQL form:**

```
OPEN [TRANSACTION transaction] cursor [USING SQL DESCRIPTOR
xsqlda]
```

Blob form: See OPEN (BLOB).

| Argument                          | Description                                                                                                          |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------|
| TRANSACTION <i>transaction</i>    | Name of the transaction that controls execution of OPEN                                                              |
| <i>cursor</i>                     | Name of a previously declared cursor to open                                                                         |
| USING DESCRIPTOR<br><i>xsqlda</i> | Passes the values corresponding to the prepared statement's parameters through the extended descriptor area (XSQLDA) |

**Description** OPEN evaluates the search condition specified in a cursor's DECLARE CURSOR statement. The selected rows become the *active set* for the cursor.

A cursor is a one-way pointer into the ordered set of rows retrieved by the SELECT in a DECLARE CURSOR statement. It enables sequential access to retrieved rows in turn. There are four related cursor statements:

| Stage | Statement      | Purpose                                                                                                                                                                                     |
|-------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | DECLARE CURSOR | Declares the cursor; the SELECT statement determines rows retrieved for the cursor                                                                                                          |
| 2     | OPEN           | Retrieves the rows specified for retrieval with DECLARE CURSOR; the resulting rows become the cursor's <i>active set</i>                                                                    |
| 3     | FETCH          | Retrieves the current row from the active set, starting with the first row <ul style="list-style-type: none"> <li>Subsequent FETCH statements advance the cursor through the set</li> </ul> |
| 4     | CLOSE          | Closes the cursor and release system resources                                                                                                                                              |

**Examples** The following embedded SQL statement opens a cursor:

```
EXEC SQL
 OPEN C;
```

**See also** CLOSE, DECLARE CURSOR, FETCH

## OPEN (BLOB)

Opens a previously declared Blob cursor and prepares it for read or insert. Available in **gpre**.

**Syntax** OPEN [TRANSACTION *name*] *cursor*  
 {INTO | USING} *:blob\_id*;

| Argument                   | Description                                                                                       |
|----------------------------|---------------------------------------------------------------------------------------------------|
| TRANSACTION<br><i>name</i> | Specifies the transaction under which the cursor is opened<br>Default: The default transaction    |
| <i>cursor</i>              | Name of the Blob cursor                                                                           |
| INTO   USING               | Depending on Blob cursor type, use one of these:<br>INTO: For INSERT BLOB<br>USING: For READ BLOB |
| <i>blob_id</i>             | Identifier for the Blob column                                                                    |

**Description** OPEN prepares a previously declared cursor for reading or inserting Blob data. Depending on whether the DECLARE CURSOR statement declares a READ or INSERT BLOB cursor, OPEN obtains the value for Blob ID differently:

- For a READ BLOB, the *blob\_id* comes from the outer TABLE cursor.
- For an INSERT BLOB, the *blob\_id* is returned by the system.

**Examples** The following embedded SQL statements declare and open a Blob cursor:

```
EXEC SQL
 DECLARE BC CURSOR FOR
 INSERT BLOB PROJ_DESC INTO PRJOECT;

EXEC SQL
 OPEN BC INTO :blob_id;
```

**See also** CLOSE (BLOB), DECLARE CURSOR (BLOB), FETCH (BLOB), INSERT CURSOR (BLOB)

## PREPARE

Prepares a dynamic SQL (DSQL) statement for execution. Available in **gpre**.

**Syntax** PREPARE [TRANSACTION *transaction*] *statement*  
[INTO SQL DESCRIPTOR *xsqlda*] FROM {*:variable* | '*string*'};

| Argument                          | Description                                                              |
|-----------------------------------|--------------------------------------------------------------------------|
| TRANSACTION<br><i>transaction</i> | Name of the transaction under control of which the statement is executed |

| Argument                              | Description                                                                                                    |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <i>statement</i>                      | Establishes an alias for the prepared statement that can be used by subsequent DESCRIBE and EXECUTE statements |
| INTO <i>xsqlda</i>                    | Specifies an XSQLDA to be filled in with the description of the select-list columns in the prepared statement  |
| : <i>variable</i>   ' <i>string</i> ' | DSQL statement to PREPARE; can be a host-language variable or a string literal                                 |

**Description** PREPARE readies a DSQL statement for repeated execution by:

- Checking the statement for syntax errors.
- Determining datatypes of optionally specified dynamic parameters.
- Optimizing statement execution.
- Compiling the statement for execution by EXECUTE.

PREPARE is part of a group of statements that prepare DSQL statements for execution.

| Statement         | Purpose                                                      |
|-------------------|--------------------------------------------------------------|
| PREPARE           | Readies a DSQL statement for execution                       |
| DESCRIBE          | Fills in the XSQLDA with information about the statement     |
| EXECUTE           | Executes a previously prepared statement                     |
| EXECUTE IMMEDIATE | Prepares a DSQL statement, executes it once, and discards it |

After a statement is prepared, it is available for execution as many times as necessary during the current session. To prepare and execute a statement only once, use EXECUTE IMMEDIATE.

*statement* establishes a symbolic name for the actual DSQL statement to prepare. It is *not* declared as a host-language variable. Except for C programs, **gpre** does not distinguish between uppercase and lowercase in *statement*, treating “B” and “b” as the same character. For C programs, use the **gpre -either\_case** switch to activate case sensitivity during preprocessing.

If the optional INTO clause is used, PREPARE also fills in the extended SQL descriptor area (XSQLDA) with information about the datatype, length, and name of select-list columns in the prepared statement. This clause is useful only when the statement to prepare is a SELECT.

**Note** The DESCRIBE statement can be used instead of the INTO clause to fill in the XSQLDA for a select list.

The FROM clause specifies the actual DSQL statement to PREPARE. It can be a host-language variable, or a quoted string literal. The DSQL statement to PREPARE can be any SQL data definition, data manipulation, or transaction-control statement.

**Examples** The following embedded SQL statement prepares a DSQL statement from a host-variable statement. Because it uses the optional INTO clause, the assumption is that the DSQL statement in the host variable is a SELECT.

```
EXEC SQL
 PREPARE Q INTO xsqlda FROM :buf;
```

**Note** The previous statement could also be prepared and described in the following manner:

```
EXEC SQL
 PREPARE Q FROM :buf;

EXEC SQL
 DESCRIBE Q INTO SQL DESCRIPTOR xsqlda;
```

**See also** DESCRIBE, EXECUTE, EXECUTE IMMEDIATE

## RELEASE SAVEPOINT

---

**Syntax** RELEASE SAVEPOINT *savepoint\_name*

**Description** Releasing a savepoint destroys savepoint named by the identifier without affecting any work that has been performed subsequent to its creation.

**See also** SAVEPOINT, ROLLBACK

## REVOKE

---

Withdraws privileges from users for specified database objects. Available in **gpre**, **DSQL**, and **isql**.

```
REVOKE [GRANT OPTION FOR] privilege ON [TABLE] {tablename |
viewname}
 FROM {object | userlist | rolelist | GROUP UNIX_group}
 | EXECUTE ON PROCEDURE procname FROM {object | userlist}
 | role_granted FROM {PUBLIC | role_grantee_list}};

privileges = ALL [PRIVILEGES] | privilege_list

privilege_list = {
 SELECT
 | DELETE
 | INSERT
 | ENCRYPT ON ENCRYPTION
 | DECRYPT
 | UPDATE [(col [, col ...])]
 | REFERENCES [(col [, col ...])]
```

```

 }, privilege_list ...]
 object = {
 PROCEDURE procname
 | TRIGGER trigrname
 | VIEW viewname
 | PUBLIC
 }, object ...]
 userlist = [USER] username [, [USER] username ...]
 rolelist = rolename [, rolename ...]
 role_granted = rolename [, rolename ...]
 role_grantee_list = [USER] username [, [USER] username ...]

```

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                 | Description                                                                                                                        |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>privilege_list</i>    | Name of privilege to be granted; valid options are SELECT, DELETE, INSERT, ENCRYPT ON ENCRYPTION, DECRYPT, UPDATE, and REFERENCES  |
| GRANT OPTION FOR         | Removes grant authority for privileges listed in the REVOKE statement from <i>userlist</i> ; cannot be used with <i>object</i>     |
| <i>col</i>               | Column for which the privilege is revoked                                                                                          |
| <i>tablename</i>         | Name of an existing table for which privileges are revoked                                                                         |
| <i>viewname</i>          | Name of an existing view for which privileges are revoked                                                                          |
| GROUP <i>unix_group</i>  | On a UNIX system, the name of a group defined in <i>/etc/group</i>                                                                 |
| <i>object</i>            | Name of an existing database object from which privileges are to be revoked                                                        |
| <i>userlist</i>          | A list of users from whom privileges are to be revoked                                                                             |
| <i>rolename</i>          | An existing role created with the CREATE ROLE statement                                                                            |
| <i>role_grantee_list</i> | A list of users to whom <i>rolename</i> is granted; users must be in the InterBase security database ( <i>admin.ib</i> by default) |

**Description** REVOKE removes privileges from users or other database objects. Privileges are operations for which a user has authority. The following table lists SQL privileges:

**Table 2.10** SQL privileges

| Privilege             | Removes a user's privilege to ...                                                                                                                                                          |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ALL                   | Perform SELECT, DELETE, INSERT, UPDATE, REFERENCES, and EXECUTE                                                                                                                            |
| SELECT                | Retrieve rows from a table or view                                                                                                                                                         |
| DELETE                | Remove rows from a table or view                                                                                                                                                           |
| DECRYPT               | After encrypting a column, the database owner or the individual table owner can grant decrypt permission to users who need to access the values in an encrypted column.                    |
| ENCRYPT ON ENCRYPTION | Enables the database owner or individual table owner to use a specific encryption key to encrypt a database or column. Only the SYSDSO (Data Security Owner) can grant encrypt permission. |
| INSERT                | Store new rows in a table or view                                                                                                                                                          |
| UPDATE                | Change the current value in one or more columns in a table or view; can be restricted to a specified subset of columns                                                                     |
| REFERENCES            | Reference the specified columns with a foreign key; at a minimum, this must be granted to all the columns of the primary key if it is granted at all                                       |
| EXECUTE               | Execute a stored procedure                                                                                                                                                                 |

GRANT OPTION FOR revokes a user's right to GRANT privileges to other users.

The following limitations should be noted for REVOKE:

- Only the user who grants a privilege can revoke that privilege.
- A single user can be assigned the same privileges for a database object by any number of other users. A REVOKE issued by a user only removes privileges previously assigned by that particular user.
- Privileges granted to all users with PUBLIC can only be removed by revoking privileges from PUBLIC.
- When a role is revoked from a user, all privileges that granted by that user to others because of authority gained from membership in the role are also revoked.
- For more information about the REVOKE ENCRYPT ON ENCRYPTION and REVOKE DECRYPT permissions, see Chapter 13, "Encrypting Your Data" in the [Data Definition Guide](#).

**Examples** The following `isql` statement takes the SELECT privilege away from a user for a table:

```
REVOKE SELECT ON COUNTRY FROM MIREILLE;
```

The following **isql** statement withdraws EXECUTE privileges for a procedure from another procedure and a user:

```
REVOKE EXECUTE ON PROCEDURE GET_EMP_PROJ
FROM PROCEDURE ADD_EMP_PROJ, LUIS;
```

**See also** GRANT

## ROLLBACK

Restores the database to its state prior to the start of the current transaction or savepoint. Available in **gpre**, DSQL, and **isql**.

**Syntax** ROLLBACK [TRANSACTION *name*] [TO SAVEPOINT *name*][WORK][RELEASE];

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                 | Description                                                                                                               |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------|
| TRANSACTION <i>name</i>  | Specifies the transaction to roll back in a multiple-transaction application [Default: roll back the default transaction] |
| TO SAVEPOINT <i>name</i> | Specifies the savepoint to roll back to                                                                                   |
| WORK                     | Optional word allowed for compatibility                                                                                   |
| RELEASE                  | Detaches from all databases after ending the current transaction; SQL only                                                |

**Description** ROLLBACK undoes changes made to a database by the current transaction, then ends the transaction. It breaks the program's connection to the database and frees system resources. Use RELEASE in the last ROLLBACK to close all open databases. Wait until a program no longer needs the database to release system resources.

The TRANSACTION clause can be used in multiple-transaction SQL applications to specify which transaction to roll back. If omitted, the default transaction is rolled back. The TRANSACTION clause is not available in DSQL.

**Note** RELEASE, available only in SQL, detaches from all databases after ending the current transaction. In effect, this option ends database processing. RELEASE is supported for backward compatibility with older versions of InterBase. The preferred method of detaching is with DISCONNECT.

**Examples** The following **isql** statement rolls back the default transaction:

```
ROLLBACK;
```

The next embedded SQL statement rolls back a named transaction:

```
EXEC SQL
 ROLLBACK TRANSACTION MYTRANS;
```

**See also** COMMIT, DISCONNECT

For more information about controlling transactions, see the [Embedded SQL Guide](#).

## SAVEPOINT

---

**Syntax** SAVEPOINT *savepoint\_name*

**Description** A savepoint allows a transaction to be partially rolled back. Updates that are made after a named savepoint is established can be rolled back by issuing a ROLLBACK command of the following form:

```
ROLLBACK [TRANSACTION transaction_name] TO SAVEPOINT
savepoint_name;
```

If no transaction name is specified, the default transaction is used.

A savepoint name can be any valid SQL identifier. Savepoint names must be unique within their atomic execution context. If you assign a name that is already in use, the existing savepoint is released and the name is applied to the current savepoint. An application, for example, is an execution context, as is each trigger and stored procedure. Thus, if you have an application with several triggers, you can have a savepoint named SV1 within the application and also within each trigger and stored procedure.

**See also** RELEASE SAVEPOINT, ROLLBACK

## SELECT

---

Retrieves data from one or more tables. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** SELECT [TRANSACTION *transaction*]  
 [DISTINCT | ALL]  
 {\*} | *val* [, *val* ...]  
 [INTO :*var* [, :*var* ...]]  
 FROM *tableref* [, *tableref* ...]  
 [WHERE *search\_condition*]  
 [GROUP BY *col* [COLLATE *collation*] [, *col* [COLLATE *collation*] ...]  
 [HAVING *search\_condition*]  
 [UNION [ALL] *select\_expr*]  
 [PLAN *plan\_expr*]  
 [ORDER BY *order\_list*]  
 [ROWS *value* [TO *upper\_value*] [BY *step\_value*][PERCENT][WITH TIES]]  
 [FOR UPDATE [OF *col* [, *col* ...]]];

```
val = {
 col [array_dim] | :variable
```

## Statement and Function Reference

```
constant | expr | function
udf ([val [, val ...]])
NULL | USER | RDB$DB_KEY | ?
} [COLLATE collation] [AS alias]

array_dim = [[x:]y [, [x:]y ...]]

constant = num | 'string' | charsetname 'string'

function = COUNT (* | [ALL] val | DISTINCT val)
 SUM ([ALL] val | DISTINCT val)
 AVG ([ALL] val | DISTINCT val)
 MAX ([ALL] val | DISTINCT val)
 MIN ([ALL] val | DISTINCT val)
 CAST (val AS datatype)
 UPPER (val)
 GEN_ID (generator, val)

tableref = joined_table | table | view | procedure
 [(val [, val ...])] [alias]

joined_table = tableref join_type JOIN tableref
 ON search_condition | (joined_table)

join_type = [INNER] JOIN
 {LEFT | RIGHT | FULL } [OUTER] }

search_condition = val operator {val | (select_one)}
 val [NOT] BETWEEN val AND val
 val [NOT] LIKE val [ESCAPE val]
 val [NOT] IN (val [, val ...] | select_list)
 val IS [NOT] NULL
 val {>= | <=} val
 val [NOT] {= | < | >} val
 {ALL | SOME | ANY} (select_list)
 EXISTS (select_expr)
 SINGULAR (select_expr)
 val [NOT] CONTAINING val
 val [NOT] STARTING [WITH] val
 (search_condition)
 NOT search_condition
 search_condition OR search_condition
 search_condition AND search_condition

operator = {= | < | > | <= | >= | !< | !> | <> | !=}

plan_expr =
 [JOIN | [SORT] [MERGE]] ({plan_item | plan_expr}
 [, {plan_item | plan_expr} ...])

plan_item = {table | alias}
 {NATURAL | INDEX (index [, index ...]) | ORDER index}
```

```

order_list =
 {col | int} [COLLATE collation]
 [ASC [ENDING] | DESC [ENDING]]
 [, order_list ...]

```

| Argument           | Description                                                 |
|--------------------|-------------------------------------------------------------|
| <i>expr</i>        | A valid SQL expression that results in a single value       |
| <i>select_one</i>  | A SELECT on a single column that returns exactly one value  |
| <i>select_list</i> | A SELECT on a single column that returns zero or more rows  |
| <i>select_expr</i> | A SELECT on a list of values that returns zero or more rows |

| Argument                                     | Description                                                                                                                                                                                                                  |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TRANSACTION <i>transaction</i>               | Name of the transaction under control of which the statement is executed; SQL only                                                                                                                                           |
| SELECT [DISTINCT   ALL]                      | Specifies data to retrieve <ul style="list-style-type: none"> <li>DISTINCT prevents duplicate values from being returned</li> <li>ALL, the default, retrieves every value</li> </ul>                                         |
| {*  <i>val</i> [, <i>val</i> ...]}           | The asterisk (*) retrieves all columns for the specified tables<br><i>val</i> [, <i>val</i> ...] retrieves a list of specified columns, values, and expressions                                                              |
| INTO : <i>var</i> [, <i>var</i> ...]         | Singleton select in embedded SQL only; specifies a list of host-language variables into which to retrieve values                                                                                                             |
| FROM <i>tableref</i> [, <i>tableref</i> ...] | List of tables, views, and stored procedures from which to retrieve data; list can include joins and joins can be nested                                                                                                     |
| <i>table</i>                                 | Name of an existing table in a database                                                                                                                                                                                      |
| <i>view</i>                                  | Name of an existing view in a database                                                                                                                                                                                       |
| <i>procedure</i>                             | Name of an existing stored procedure that functions like a SELECT statement                                                                                                                                                  |
| <i>alias</i>                                 | Brief, alternate name for a table, view, or column; after declaration in <i>tableref</i> , <i>alias</i> can stand in for subsequent references to a table or view                                                            |
| <i>joined_table</i>                          | A table reference consisting of a JOIN                                                                                                                                                                                       |
| <i>join_type</i>                             | Type of join to perform. Default: INNER                                                                                                                                                                                      |
| WHERE <i>search_condition</i>                | <ul style="list-style-type: none"> <li>Specifies a condition that limits rows retrieved to a subset of all available rows</li> <li>A WHERE clause can contain its own SELECT statement, referred to as a subquery</li> </ul> |

| Argument                                                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GROUP BY <i>col</i> [, <i>col</i> ...]                                                           | Groups related rows based on common column values; used in conjunction with HAVING                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| COLLATE <i>collation</i>                                                                         | Specifies the collation order for the data retrieved by the query                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| HAVING <i>search_condition</i>                                                                   | Used with GROUP BY; specifies a condition that limits the grouped rows returned                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| UNION [ALL]                                                                                      | <ul style="list-style-type: none"> <li>Combines the results of two or more SELECT statements to produce a single, dynamic table without duplicate rows</li> <li>The ALL option keeps identical rows separate instead of folding them together into one</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| PLAN <i>plan_expr</i>                                                                            | Specifies the query plan that should be used by the query optimizer instead of one it would normally choose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <i>plan_item</i>                                                                                 | Specifies a table and index method for a plan                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| ORDER BY <i>order_list</i>                                                                       | Specifies columns to order, either by column name or ordinal number in the query, and the sort order (ASC or DESC) for the returned rows                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| ROWS <i>value</i><br>[TO <i>upper_value</i> ]<br>[BY <i>step_value</i> ]<br>[PERCENT][WITH TIES] | <ul style="list-style-type: none"> <li><i>value</i> is the total number of rows to return if used by itself</li> <li><i>value</i> is the starting row number to return if used with TO</li> <li><i>value</i> is the percent if used with PERCENT</li> <li><i>upper_value</i> is the last row or highest percent to return</li> <li>If <i>step_value</i> = <i>n</i>, returns every <i>n</i>th row, or <i>n</i> percent rows</li> <li>PERCENT causes all previous ROWS values to be interpreted as percents</li> <li>WITH TIES returns additional duplicate rows when the <i>last</i> value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with ORDER BY</li> </ul> |
| FOR UPDATE                                                                                       | Specifies columns listed after the SELECT clause of a DECLARE CURSOR statement that can be updated using a WHERE CURRENT OF clause                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

- Description** SELECT retrieves data from tables, views, or stored procedures. Variations of the SELECT statement make it possible to:
- Retrieve a single row, or part of a row, from a table. This operation is referred to as a *singleton select*.  
  
In embedded applications, all SELECT statements that occur outside the context of a cursor must be singleton selects.
  - Retrieve multiple rows, or parts of rows, from a table.

In embedded applications, multiple row retrieval is accomplished by embedding a SELECT within a DECLARE CURSOR statement.

In **isql**, SELECT can be used directly to retrieve multiple rows.

- Retrieve related rows, or parts of rows, from a join of two or more tables.
- Retrieve all rows, or parts of rows, from union of two or more tables.
- Return portions or sequential portions of a larger result set; useful for Web developers, among others.

All SELECT statements consist of two required clauses (SELECT, FROM), and possibly others (INTO, WHERE, GROUP BY, HAVING, UNION, PLAN, ORDER BY, ROWS).

Because SELECT is such a ubiquitous and complex statement, a meaningful discussion lies outside the scope of this reference. To learn how to use SELECT in **isql**, see the [Operations Guide](#). For a complete explanation of SELECT and its clauses, see the [Embedded SQL Guide](#).

#### Notes on SELECT syntax

- When declaring arrays, you must include the outermost brackets, shown below in bold. For example, the following statement creates a 5 by 5 two-dimensional array of strings, each of which is 6 characters long:

```
my_array = varchar(6) [5,5]
```

Use the colon (:) to specify an array with a starting point other than 1. The following example creates an array of integers that begins at 10 and ends at 20:

```
my_array = integer[20:30]
```

- In SQL and **isql**, you cannot use *val* as a parameter placeholder (like “?”).
- In DSQL and **isql**, *val* cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.
- You cannot specify a GROUP BY clause for Blob and array columns.

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

**Examples** The following **isql** statement selects columns from a table:

```
SELECT JOB_GRADE, JOB_CODE, JOB_COUNTRY, MAX_SALARY FROM
PROJECT;
```

The next **isql** statement uses the \* wildcard to select all columns and rows from a table:

```
SELECT * FROM COUNTRIES;
```

The following embedded SQL statement uses an aggregate function to count all rows in a table that satisfy a search condition specified in the WHERE clause:

```
EXEC SQL
 SELECT COUNT (*) INTO :cnt FROM COUNTRY
 WHERE POPULATION > 5000000;
```

The next **isql** statement establishes a table alias in the SELECT clause and uses it to identify a column in the WHERE clause:

```
SELECT C.CITY FROM CITIES C
 WHERE C.POPULATION < 1000000;
```

The following **isql** statement selects two columns and orders the rows retrieved by the second of those columns:

```
SELECT CITY, STATE FROM CITIES
 ORDER BY STATE;
```

The next **isql** statement performs a left join:

```
SELECT CITY, STATE_NAME FROM CITIES C
 LEFT JOIN STATES S ON S.STATE = C.STATE
 WHERE C.CITY STARTING WITH 'San';
```

The following **isql** statement specifies a query optimization plan for ordered retrieval, utilizing an index for ordering:

```
SELECT * FROM CITIES
 PLAN (CITIES ORDER CITIES_1);
 ORDER BY CITY
```

The next **isql** statement specifies a query optimization plan based on a three-way join with two indexed column equalities:

```
SELECT * FROM CITIES C, STATES S, MAYORS M
 WHERE C.CITY = M.CITY AND C.STATE = M.STATE
 PLAN JOIN (STATE NATURAL, CITIES INDEX DUPE_CITY,
 MAYORS INDEX MAYORS_1);
```

The next example queries two of the system tables, RDB\$CHARACTER\_SETS and RDB\$COLLATIONS to display all the available character sets, their ID numbers, number of bytes per character, and collations. Note the use of ordinal column numbers in the ORDER BY clause.

```
SELECT RDB$CHARACTER_SET NAME, RDB$CHARACTER_SET_ID,
 RDB$BYTES_PER_CHARACTER, RDB$COLLATION_NAME
 FROM RDB$CHARACTER_SETS JOIN RDB$COLLATIONS
 ON RDB$CHARACTER_SETS.RDB$CHARACTER_SET_ID =
 RDB$COLLATIONS.RDB$CHARACTER_SET_ID
 ORDER BY 1, 4;
```

The following examples reward the best performing sales people and terminate the least performing members of the sales team. The examples show how a Web developer, for example, could split the result set in half for display purposes.

```
SELECT SALESMAN, SALES_DOLLARS, SALES_REGION
 FROM SALESPeople
```

```

ORDER BY SALES_DOLLARS DESC
ROWS 1 TO 50;

SELECT SALESMAN, SALES_DOLLARS, SALES_REGION
FROM SALESPEOPLE
ORDER BY SALES_DOLLARS DESC
ROWS 50 TO 100 WITH TIES;

```

Reward the best 100 performing salesmen with a 15 percent bonus:

```

UPDATE SALESPEOPLE
SET SALES_BONUS = 0.15 * SALES_DOLLARS
ORDER BY SALES_DOLLARS DESC
ROWS 100 WITH TIES;

```

Eliminate the worst five percent of the sales force:

```

DELETE FROM SALESPEOPLE
ORDER BY SALES_DOLLARS
ROWS 5 PERCENT WITH TIES;

```

**See also** DECLARE CURSOR, DELETE, INSERT, UPDATE

For a full discussion of data retrieval in embedded programming using DECLARE CURSOR and SELECT, see the [Embedded SQL Guide](#).

## SET DATABASE

---

Declares a database handle for database access. Available in **gpre**.

**Syntax** SET {DATABASE | SCHEMA} *dbhandle* =  
 [GLOBAL | STATIC | EXTERN][COMPILETIME][FILENAME] '*dbname*'  
 [USER '*name*' PASSWORD '*string*']  
 [RUNTIME [FILENAME]  
 {'*dbname*' | :*var*}  
 [USER {'*name*' | :*var*} PASSWORD {'*string*' | :*var*}}];

| Argument        | Description                                                                                                                                                                                        |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dbhandle</i> | An alias for a specified database <ul style="list-style-type: none"> <li>• Must be unique within the program</li> <li>• Used in subsequent SQL statements that support database handles</li> </ul> |
| GLOBAL          | [Default] Makes this database declaration available to all modules                                                                                                                                 |
| STATIC          | Limits scope of this database declaration to the current module                                                                                                                                    |
| EXTERN          | References a database declaration in another module, rather than actually declaring a new handle                                                                                                   |

---

| Argument          | Description                                                                                                                                                                                                                                |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| COMPILETIME       | Identifies the database used to look up column references during preprocessing <ul style="list-style-type: none"> <li>If only one database is specified in SET DATABASE, it is used both at runtime and compiletime</li> </ul>             |
| 'dbname'          | Location and path name of the database associated with <i>dbhandle</i> ; platform-specific                                                                                                                                                 |
| RUNTIME           | Specifies a database to use at runtime if different than the one specified for use during preprocessing                                                                                                                                    |
| :var              | Host-language variable containing a database specification, user name, or password                                                                                                                                                         |
| USER 'name'       | A valid user name on the server where the database resides <ul style="list-style-type: none"> <li>Used with PASSWORD to gain database access on the server</li> <li>Required for PC client attachments, optional for all others</li> </ul> |
| PASSWORD 'string' | A valid password on the server where the database resides <ul style="list-style-type: none"> <li>Used with USER to gain database access on the server</li> <li>Required for PC client attachments, optional for all others.</li> </ul>     |

**Description** SET DATABASE declares a database handle for a specified database and associates the handle with that database. It enables optional specification of different compile-time and run-time databases. Applications that access multiple databases simultaneously must use SET DATABASE statements to establish separate database handles for each database.

*dbhandle* is an application-defined name for the database handle. Usually handle names are abbreviations of the actual database name. Once declared, database handles can be used in subsequent CONNECT, COMMIT, and ROLLBACK statements. They can also be used within transactions to differentiate table names when two or more attached databases contain tables with the same names.

*dbname* is a platform-specific file specification for the database to associate with *dbhandle*. It should follow the file syntax conventions for the server where the database resides.

GLOBAL, STATIC, and EXTERN are optional parameters that determine the scope of a database declaration. The default scope, GLOBAL, means that a database handle is available to all code modules in an application. STATIC limits database handle availability to the code module where the handle is declared. EXTERN references a global database handle in another module.

The optional COMPILETIME and RUNTIME parameters enable a single database handle to refer to one database when an application is preprocessed, and to another database when an application is run by a user. If omitted, or if only a COMPILETIME database is specified, InterBase uses the same database during preprocessing and at run time.

The USER and PASSWORD parameters are required for all PC client applications, but are optional for all other remote attachments. The user name and password are verified by the server in the security database before permitting remote attachments to succeed.

**Examples** The following embedded SQL statement declares a handle for a database:

```
EXEC SQL
 SET DATABASE DB1 = 'employee.ib';
```

The next embedded SQL statement declares different databases at compile time and run time. It uses a host-language variable to specify the run-time database.

```
EXEC SQL
 SET DATABASE EMDBP = 'employee.ib' RUNTIME :db_name;
```

**See also** COMMIT, CONNECT, ROLLBACK, SELECT

For more information on the security database, see the [Operations Guide](#).

## SET GENERATOR

Sets a new value for an existing generator. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** SET GENERATOR *name* TO *int*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description                                                                    |
|-------------|--------------------------------------------------------------------------------|
| <i>name</i> | Name of an existing generator                                                  |
| <i>int</i>  | Value to which to set the generator, an integer from $-2^{63}$ to $2^{63} - 1$ |

**Description** SET GENERATOR initializes a starting value for a newly created generator, or resets the value of an existing generator. A generator provides a unique, sequential numeric value through the GEN\_ID() function. If a newly created generator is not initialized with SET GENERATOR, its starting value defaults to zero.

*int* is the new value for the generator. When the GEN\_ID() function inserts or updates a value in a column, that value is *int* plus the increment specified in the GEN\_ID() step parameter. Any value that can be stored in a DECIMAL(18,0) can be specified as the value in a SET GENERATOR statement.

Generators return a 64-bit value, and wrap around only after  $2^{64}$  invocations (assuming an increment of 1). Use an ISC-INT64 variable to hold the value returned by a generator.

**Tip** To force a generator's first insertion value to 1, use SET GENERATOR to specify a starting value of 0, and set the step value of the GEN\_ID() function to 1.

**Important** When resetting a generator that supplies values to a column defined with PRIMARY KEY or UNIQUE integrity constraints, be careful that the new value does not enable duplication of existing column values, or all subsequent insertions and updates will fail.

**Example** The following **isql** statement sets a generator value to 1,000:

```
SET GENERATOR CUST_NO_GEN TO 1000;
```

If GEN\_ID() now calls this generator with a step value of 1, the first number it returns is 1,001.

**See also** CREATE GENERATOR, CREATE PROCEDURE, CREATE TRIGGER, GEN\_ID()

## SET NAMES

---

Specifies an active character set to use for subsequent database attachments. Available in **gpre**, and **isql**.

**Syntax** SET NAMES [*charset* | :*var*];

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument       | Description                                                                                                                                                                          |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>charset</i> | Name of a character set that identifies the active character set for a given process; default: NONE                                                                                  |
| : <i>var</i>   | Host variable containing string identifying a known character set name <ul style="list-style-type: none"><li>• Must be declared as a character set name</li><li>• SQL only</li></ul> |

**Description** SET NAMES specifies the character set to use for subsequent database attachments in an application. It enables the server to translate between the default character set for a database on the server and the character set used by an application on the client.

SET NAMES must appear before the SET DATABASE and CONNECT statements it is to affect.

**Tip** Use a host-language variable with SET NAMES in an embedded application to specify a character set interactively.

For a complete list of character sets recognized by InterBase, see [Chapter 7, “Character Sets and Collation Orders.”](#) Choice of character sets limits possible collation orders to a subset of all available collation orders. Given a specific character set, a specific collation order can be specified when data is selected, inserted, or updated in a column.

**Important** If you do not specify a default character set, the character set defaults to NONE. Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it. You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a different character set. No transliteration is performed between the source and destination character sets, so in most cases, errors occur during assignment.

**Example** The following statements demonstrate the use of SET NAMES in an embedded SQL application:

```
EXEC SQL
 SET NAMES ISO8859_1;

EXEC SQL
 SET DATABASE DB1 = 'employee.ib';

EXEC SQL
 CONNECT;
```

The next statements demonstrate the use of SET NAMES in **isql**:

```
SET NAMES LATIN1;
CONNECT 'employee.ib';
```

**See also** CONNECT, SET DATABASE

For more information about character sets and collation orders, see the [Data Definition Guide](#).

## SET SQL DIALECT

Declares the SQL Dialect for database access. Available in **gpre** and **isql**.

**Syntax** SET SQL DIALECT *n*;

| Argument | Description                             |
|----------|-----------------------------------------|
| <i>n</i> | The SQL Dialect type, either 1, 2, or 3 |

**Description** SET SQL DIALECT declares the SQL Dialect for database access.

*n* is the SQL Dialect type 1, 2, or 3. If no dialect is specified, the default dialect is set to that of the specified compile-time database. If the default dialect is different than the one specified by the user, a warning is generated and the default dialect is set to the user-specified value

**Table 2.11** SQL Dialects

| SQL Dialect | Used for                                                                                                            |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| 1           | InterBase 5 and earlier compatibility                                                                               |
| 2           | Transitional dialect used to flag changes when migrating from dialect 1 to dialect 3                                |
| 3           | Current InterBase; allows you to use delimited identifiers, exact numerics, and DATE, TIME, and TIMESTAMP datatypes |

**Examples** The following embedded SQL statement sets the SQL Dialect to 3:

```
EXEC SQL
 SET SQL DIALECT 3;
```

**See also** SHOW SQL DIALECT

## SET STATISTICS

Recomputes the selectivity of a specified index. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** SET STATISTICS INDEX *name*;

**Important** In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument    | Description                                                  |
|-------------|--------------------------------------------------------------|
| <i>name</i> | Name of an existing index for which to recompute selectivity |

**Description** SET STATISTICS enables the selectivity of an index to be recomputed. Index selectivity is a calculation, based on the number of distinct rows in a table, that is made by the InterBase optimizer when a table is accessed. It is cached in memory, where the optimizer can access it to calculate the optimal retrieval plan for a given query. For tables where the number of duplicate values in indexed columns radically increases or decreases, periodically recomputing index selectivity can improve performance.

Only the creator of an index can use SET STATISTICS.

**Note** SET STATISTICS does not rebuild an index. To rebuild an index, use ALTER INDEX.

**Example** The following embedded SQL statement recomputes the selectivity for an index:

```
EXEC SQL
 SET STATISTICS INDEX MINSALX;
```

**See also** ALTER INDEX, CREATE INDEX, DROP INDEX

## SET TRANSACTION

---

Starts a transaction and optionally specifies its behavior. Available in **gpre**, DSQL, and **isql**.

**Syntax**    **SET TRANSACTION** [NAME *transaction*]  
               [READ WRITE | READ ONLY]  
               [WAIT | NO WAIT]  
               [[ISOLATION LEVEL] {SNAPSHOT [TABLE STABILITY]  
               | READ COMMITTED [[NO] RECORD\_VERSION]]]  
               [RESERVING *reserving\_clause*  
               | USING *dbhandle* [, *dbhandle* ...]];  
  
*reserving\_clause* = table [, table ...]  
                   [FOR [SHARED | PROTECTED] {READ | WRITE}] [, *reserving\_clause*]

**Important**    In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in **isql**, the semicolon is a terminating symbol for the statement, so it must be included.

| Argument                                      | Description                                                                                                                                                                                    |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NAME <i>transaction</i>                       | Specifies the name for this transaction <ul style="list-style-type: none"> <li><i>transaction</i> is a previously declared and initialized host-language variable</li> <li>SQL only</li> </ul> |
| READ WRITE                                    | [Default] Specifies that the transaction can read and write to tables                                                                                                                          |
| READ ONLY                                     | Specifies that the transaction can only read tables                                                                                                                                            |
| WAIT                                          | [Default] Specifies that a transaction wait for access if it encounters a lock conflict with another transaction                                                                               |
| NO WAIT                                       | Specifies that a transaction immediately return an error if it encounters a lock conflict                                                                                                      |
| ISOLATION LEVEL                               | Specifies the isolation level for this transaction when attempting to access the same tables as other simultaneous transactions; default: SNAPSHOT                                             |
| RESERVING <i>reserving_clause</i>             | Reserves lock for tables at transaction start                                                                                                                                                  |
| USING <i>dbhandle</i> [, <i>dbhandle</i> ...] | Limits database access to a subset of available databases; SQL only                                                                                                                            |

---

**Description** SET TRANSACTION starts a transaction, and optionally specifies its database access, lock conflict behavior, and level of interaction with other concurrent transactions accessing the same data. It can also reserve locks for tables. As an alternative to reserving tables, multiple database SQL applications can restrict a transaction's access to a subset of connected databases.

**Important** Applications preprocessed with the **gpre -manual** switch must explicitly start each transaction with a SET TRANSACTION statement.

SET TRANSACTION affects the default transaction unless another transaction is specified in the optional NAME clause. Named transactions enable support for multiple, simultaneous transactions in a single application. All transaction names must be declared as host-language variables at compile time. In DSQL, this restriction prevents dynamic specification of transaction names.

By default a transaction has READ WRITE access to a database. If a transaction only needs to read data, specify the READ ONLY parameter.

When simultaneous transactions attempt to update the same data in tables, only the first update succeeds. No other transaction can update or delete that data until the controlling transaction is rolled back or committed. By default, transactions WAIT until the controlling transaction ends, then attempt their own operations. To force a transaction to return immediately and report a lock conflict error without waiting, specify the NO WAIT parameter.

ISOLATION LEVEL determines how a transaction interacts with other simultaneous transactions accessing the same tables. The default ISOLATION LEVEL is SNAPSHOT. It provides a repeatable-read view of the database at the moment the transaction starts. Changes made by other simultaneous transactions are not visible.

SNAPSHOT TABLE STABILITY provides a repeatable read of the database by ensuring that transactions cannot write to tables, though they may still be able to read from them.

READ COMMITTED enables a transaction to see the most recently committed changes made by other simultaneous transactions. It can also update rows as long as no update conflict occurs. Uncommitted changes made by other transactions remain invisible until committed. READ COMMITTED also provides two optional parameters:

- NO RECORD\_VERSION, the default, reads only the latest version of a row. If the WAIT lock resolution option is specified, then the transaction waits until the latest version of a row is committed or rolled back, and retries its read.
- RECORD\_VERSION reads the latest committed version of a row, even if more recent uncommitted version also resides on disk.

The RESERVING clause enables a transaction to register its desired level of access for specified tables when the transaction starts instead of when the transaction attempts its operations on that table. Reserving tables at transaction start can reduce the possibility of deadlocks.

The USING clause, available only in SQL, can be used to conserve system resources by limiting the number of databases a transaction can access.

**Examples** The following embedded SQL statement sets up the default transaction with an isolation level of READ COMMITTED. If the transaction encounters an update conflict, it waits to get control until the first (locking) transaction is committed or rolled back.

```
EXEC SQL
 SET TRANSACTION WAIT ISOLATION LEVEL READ COMMITTED;
```

The next embedded SQL statement starts a named transaction:

```
EXEC SQL
 SET TRANSACTION NAME T1 READ COMMITTED;
```

The following embedded SQL statement reserves three tables:

```
EXEC SQL
 SET TRANSACTION NAME TR1
 ISOLATION LEVEL READ COMMITTED
 NO RECORD_VERSION WAIT
 RESERVING TABLE1, TABLE2 FOR SHARED WRITE,
 TABLE3 FOR PROTECTED WRITE;
```

**See also** COMMIT, ROLLBACK, SET NAMES

For more information about transactions, see the [Embedded SQL Guide](#).

## SHOW SQL DIALECT

Returns the current client SQL Dialect setting and the database SQL Dialect value. Available in **gpre** and **isql**.

**Syntax** SHOW SQL DIALECT;

**Description** SHOW SQL DIALECT returns the current client SQL Dialect setting and the database SQL Dialect value, either 1, 2, or 3.

**Table 2.12** SQL Dialects

| SQL Dialect | Used for                                                                                                            |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| 1           | InterBase 5 and earlier compatibility                                                                               |
| 2           | Transitional dialect used to flag changes when migrating from dialect 1 to dialect 3                                |
| 3           | Current InterBase; allows you to use delimited identifiers, exact numerics, and DATE, TIME, and TIMESTAMP datatypes |

**Examples** The following embedded SQL statement returns the SQL Dialect:

```
EXEC SQL
 SHOW SQL DIALECT;
```

**See also** SET SQL DIALECT

## SUM( )

Totals the numeric values in a specified column. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** SUM ([ALL] *val* | DISTINCT *val*)

| Argument   | Description                                                                                                                 |
|------------|-----------------------------------------------------------------------------------------------------------------------------|
| ALL        | Totals all values in a column                                                                                               |
| DISTINCT   | Eliminates duplicate values before calculating the total                                                                    |
| <i>val</i> | A column, constant, host-language variable, expression, non-aggregate function, or UDF that evaluates to a numeric datatype |

**Description** SUM() is an aggregate function that calculates the sum of numeric values for a column. If the number of qualifying rows is zero, SUM() returns a NULL value.

**Example** The following embedded SQL statement demonstrates the use of SUM(), AVG(), MIN(), and MAX():

```
EXEC SQL
 SELECT SUM (BUDGET), AVG (BUDGET), MIN (BUDGET), MAX
 (BUDGET)
 FROM DEPARTMENT
 WHERE HEAD_DEPT = :head_dept
 INTO :tot_budget, :avg_budget, :min_budget, :max_budget;
```

**See also** AVG(), COUNT(), MAX(), MIN()

## UPDATE

Changes the data in all or part of an existing row in a table, view, or active set of a cursor. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** **SQL form:**

```
UPDATE [TRANSACTION transaction] {table | view}
 SET col = val [, col = val ...]
 [WHERE search_condition | WHERE CURRENT OF cursor]
 [ORDER BY order_list]
 [ROWS value [TO upper_value] [BY step_value] [PERCENT] [WITH
 TIES]];
```

**DSQL and isql form:**

```
UPDATE {table | view}
 SET col = val [, col = val ...]
 [WHERE search_condition
 [ORDER BY order_list]
 [ROWS value [TO upper_value] [BY step_value] [PERCENT] [WITH
TIES]]]
```

```
val = {
 col [array_dim]
 | :variable
 | constant
 | expr
 | function
 | udf ([val [, val ...]])
 | NULL
 | USER
 | ?}
[COLLATE collation]
```

```
array_dim = [[x:]y [, [x:]y ...]]
```

```
constant = num | 'string' | charsetname 'string'
```

```
function = CAST (val AS datatype)
 | UPPER (val)
 | GEN_ID (generator, val)
```

*expr* = A valid SQL expression that results in a single value.

*search\_condition* = See CREATE TABLE for a full description.

#### Notes on the UPDATE statement

- In SQL and **isql**, you cannot use *val* as a parameter placeholder (like “?”).
- In DSQL and **isql**, *val* cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.

| Argument                       | Description                                                                   |
|--------------------------------|-------------------------------------------------------------------------------|
| TRANSACTION <i>transaction</i> | Name of the transaction under control of which the statement is executed      |
| <i>table</i>   <i>view</i>     | Name of an existing table or view to update.                                  |
| SET <i>col</i> = <i>val</i>    | Specifies the columns to change and the values to assign to those columns     |
| WHERE <i>search_condition</i>  | Searched update only; specifies the conditions a row must meet to be modified |

| Argument                                                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WHERE CURRENT OF <i>cursor</i>                                                                   | Positioned update only; specifies that the current row of a cursor's active set is to be modified <ul style="list-style-type: none"> <li>Not available in DSQL and <b>isql</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| ORDER BY <i>order_list</i>                                                                       | Specifies columns to order, either by column name or ordinal number in the query, and the sort order (ASC or DESC) for the returned rows                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| ROWS <i>value</i><br>[TO <i>upper_value</i> ]<br>[BY <i>step_value</i> ]<br>[PERCENT][WITH TIES] | <ul style="list-style-type: none"> <li><i>value</i> is the total number of rows to return if used by itself</li> <li><i>value</i> is the starting row number to return if used with TO</li> <li><i>value</i> is the percent if used with PERCENT</li> <li><i>upper_value</i> is the last row or highest percent to return</li> <li>If <i>step_value</i> = <i>n</i>, returns every <i>n</i>th row, or <i>n</i> percent rows</li> <li>PERCENT causes all previous ROWS values to be interpreted as percents</li> <li>WITH TIES returns additional duplicate rows when the <i>last</i> value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with ORDER BY</li> </ul> |

**Description** UPDATE modifies one or more existing rows in a table or view. UPDATE is one of the database privileges controlled by GRANT and REVOKE.

For searched updates, the optional WHERE clause can be used to restrict updates to a subset of rows in the table. Searched updates cannot update array slices.

**Important** Without a WHERE clause, a searched update modifies all rows in a table.

When performing a positioned update with a cursor, the WHERE CURRENT OF clause must be specified to update one row at a time in the active set.

**Note** When updating a Blob column, UPDATE replaces the entire Blob with a new value.

**Examples** The following **isql** statement modifies a column for all rows in a table:

```
UPDATE CITIES
SET POPULATION = POPULATION * 1.03;
```

The next embedded SQL statement uses a WHERE clause to restrict column modification to a subset of rows:

```
EXEC SQL
UPDATE PROJECT
SET PROJ_DESC = :blob_id
WHERE PROJ_ID = :proj_id;
```

**See also** DELETE, GRANT, INSERT, REVOKE, SELECT

## UPPER( )

---

Converts a string to all uppercase. Available in **gpre**, **DSQL**, and **isql**.

**Syntax** UPPER (*val*)

| Argument   | Description                                                                                                     |
|------------|-----------------------------------------------------------------------------------------------------------------|
| <i>val</i> | A column, constant, host-language variable, expression, function, or UDF that evaluates to a character datatype |

---

**Description** UPPER() converts a specified string to all uppercase characters. If applied to character sets that have no case differentiation, UPPER() has no effect.

**Examples** The following **isql** statement changes the name, BMatthews, to BMATTHEWS:

```
UPDATE EMPLOYEE
 SET EMP_NAME = UPPER (BMatthews)
 WHERE EMP_NAME = 'BMatthews';
```

The next **isql** statement creates a domain called PROJNO with a CHECK constraint that requires the value of the column to be all uppercase:

```
CREATE DOMAIN PROJNO
 AS CHAR(5)
 CHECK (VALUE = UPPER (VALUE));
```

**See also** CAST()

## WHENEVER

---

Traps SQLCODE errors and warnings. Available in **gpre**.

**Syntax** WHENEVER {NOT FOUND | SQLERROR | SQLWARNING}  
{GOTO *label* | CONTINUE};

| Argument          | Description                                                                        |
|-------------------|------------------------------------------------------------------------------------|
| NOT FOUND         | Traps SQLCODE = 100, no qualifying rows found for the executed statement           |
| SQLERROR          | Traps SQLCODE < 0, failed statement                                                |
| SQLWARNING        | Traps SQLCODE > 0 AND < 100, system warning or informational message               |
| GOTO <i>label</i> | Jumps to program location specified by <i>label</i> when a warning or error occurs |
| CONTINUE          | Ignores the warning or error and attempts to continue processing                   |

---

**Description** WHENEVER traps for SQLCODE errors and warnings. Every executable SQL statement returns a SQLCODE value to indicate its success or failure. If SQLCODE is zero, statement execution is successful. A non-zero value indicates an error, warning, or not found condition.

If the appropriate condition is trapped for, WHENEVER can:

- Use GOTO *label* to jump to an error-handling routine in an application.
- Use CONTINUE to ignore the condition.

WHENEVER can help limit the size of an application, because the application can use a single suite of routines for handling all errors and warnings.

WHENEVER statements should precede any SQL statement that can result in an error. Each condition to trap for requires a separate WHENEVER statement. If WHENEVER is omitted for a particular condition, it is not trapped.

**Tip** Precede error-handling routines with WHENEVER ... CONTINUE statements to prevent the possibility of infinite looping in the error-handling routines.

**Example** In the following code from an embedded SQL application, three WHENEVER statements determine which label to branch to for error and warning handling:

```
EXEC SQL
 WHENEVER SQLERROR GO TO Error; /* Trap all errors. */
EXEC SQL
 WHENEVER NOT FOUND GO TO AllDone; /* Trap SQLCODE = 100 */
EXEC SQL
 WHENEVER SQLWARNING CONTINUE; /* Ignore all warnings. */
```

For a complete discussion of error-handling methods and programming, see the [\*Embedded SQL Guide\*](#).

# Procedures and Triggers

InterBase procedure and trigger language is a complete programming language for writing stored procedures and triggers in **isql** and DSQL. It includes:

- SQL data manipulation statements: INSERT, UPDATE, DELETE, and singleton SELECT.
- Powerful extensions to SQL, including assignment statements, control-flow statements, context variables, event-posting, exceptions, and error handling.

Although stored procedures and triggers are used in entirely different ways and for different purposes, they both use procedure and trigger language. Both triggers and stored procedures can use any statements in procedure and trigger language, with some exceptions:

- OLD and NEW context variables are unique to triggers.
- Input and output parameters, and the SUSPEND and EXIT statements are unique to stored procedures.

The [Data Definition Guide](#) explains how to create and use stored procedures and triggers. This chapter is a reference for the statements that are unique to trigger and procedure language or that have special syntax when used in triggers and procedures.

## Creating Triggers and Stored Procedures

Stored procedures and triggers are defined with the CREATE PROCEDURE and CREATE TRIGGER statements, respectively. Each of these statements is composed of a *header* and a *body*.

**The header contains:**

- The name of the procedure or trigger, unique within the database.

## Statement Types Not Supported

- For a trigger:
  - A table name, identifying the table that causes the trigger to fire.
  - Statements that determine *when* the trigger fires.
- For a stored procedure:
  - An optional list of *input parameters* and their datatypes.
  - If the procedure returns values to the calling program, a list of *output parameters* and their datatypes.

### **The body contains:**

- An optional list of *local variables* and their datatypes.
- A *block* of statements in InterBase procedure and trigger language, bracketed by BEGIN and END. A block can itself include other blocks, so that there may be many levels of nesting.

## Statement Types Not Supported

---

The stored procedure and trigger language does not include many of the statement types available in DSQL or **gpre**. The following statement types are not supported in triggers or stored procedures:

- Data definition language statements: CREATE, ALTER, DROP, DECLARE EXTERNAL FUNCTION, and DECLARE FILTER
- Transaction control statements: SET TRANSACTION, COMMIT, ROLLBACK
- Dynamic SQL statements: PREPARE, DESCRIBE, EXECUTE
- CONNECT/DISCONNECT, and sending SQL statements to another database
- GRANT/REVOKE
- SET GENERATOR
- EVENT INIT/WAIT
- BEGIN/END DECLARE SECTION
- BASED ON
- WHENEVER
- DECLARE CURSOR
- OPEN
- FETCH

## Nomenclature Conventions

---

This chapter uses the following nomenclature:

- A *block* is one or more compound statements enclosed by BEGIN and END.
- A *compound statement* is either a block or a statement.
- A *statement* is a single statement in procedure and trigger language.

To illustrate in a syntax diagram:

```

<block> =
BEGIN
 <compound_statement>
 [<compound_statement> ...]
END

<compound_statement> = <block> | statement;

```

## Assignment Statement

Assigns a value to an input or output parameter or local variable. Available in triggers and stored procedures.

**Syntax** *variable* = <expression>;

| Argument          | Description                                                                                                                |
|-------------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>variable</i>   | A local variable, input parameter, or output parameter                                                                     |
| <i>expression</i> | Any valid combination of variables, SQL operators, and expressions, including user-defined functions (UDFs) and generators |

**Description** An assignment statement sets the value of a local variable, input parameter, or output parameter. Variables must be declared before they can be used in assignment statements.

**Example** The first assignment statement below sets the value of *x* to 9. The second statement sets the value of *y* at twice the value of *x*. The third statement uses an arithmetic expression to assign *z* a value of 3.

```

DECLARE VARIABLE x INTEGER;
DECLARE VARIABLE y INTEGER;
DECLARE VARIABLE z INTEGER;
x = 9;
y = 2 * x;
z = 4 * x / (y - 6);

```

**See also** DECLARE VARIABLE, [Input Parameters](#), [Output Parameters](#)

## BEGIN ... END

---

Defines a block of statements executed as one. Available in triggers and stored procedures.

**Syntax**    *<block>* =  
               BEGIN  
                   *<compound\_statement>*  
                   [*<compound\_statement>* ...]  
               END

**Description**    *<compound\_statement>* = {*<block>* | *statement*;}  
 Each block of statements in the procedure body starts with a BEGIN statement and ends with an END statement. As shown in the above syntax diagram, a block can itself contain other blocks, so there may be many levels of nesting.

BEGIN and END are not followed by a semicolon. In **isql**, the final END in the procedure body is followed by the semicolon.

The final END statement in a trigger terminates the trigger. The final END statement in a stored procedure operates differently, depending on the type of procedure:

- In a select procedure, the final END statement returns control to the application and sets SQLCODE to 100, which indicates there are no more rows to retrieve.
- In an executable procedure, the final END statement returns control and current values of output parameters, if any, to the calling application.

**Example**    The following **isql** fragment of the DELETE\_EMPLOYEE procedure shows two examples of BEGIN ... END blocks.

```
CREATE PROCEDURE DELETE_EMPLOYEE (EMP_NUM INTEGER)
AS
 DECLARE VARIABLE ANY_SALES INTEGER;
BEGIN
 ANY_SALES = 0;
 . . .
 IF (ANY_SALES > 0) THEN
 BEGIN
 EXCEPTION REASSIGN_SALES;
 EXIT;
 END
 . . .
END
;
```

**See also**    No Rows or Data Returned, SUSPEND

# Comment

---

Allows programmers to add comments to procedure and trigger code. Available in triggers and stored procedures.

**Syntax** `/* comment_text */`

| Argument            | Description                         |
|---------------------|-------------------------------------|
| <i>comment_text</i> | Any number of lines of comment text |

---

**Description** Comments can be placed on the same line as code, or on separate lines.

It is good programming practice to state the input and output parameters of a procedure in a comment preceding the procedure. It is also often useful to comment local variable declarations to indicate what each variable is used for.

**Example** The following `isql` procedure fragment illustrates some ways to use comments:

```
/*
 *Procedure DELETE_EMPLOYEE : Delete an employee.
 *
 *Parameters:
 * employee number
 *Returns:
 * --
 */
CREATE PROCEDURE DELETE_EMPLOYEE (EMP_NUM INTEGER)
AS
 DECLARE VARIABLE ANY_SALES INTEGER; /* Number of sales for
emp. */
BEGIN
 . . .
```

# DECLARE VARIABLE

---

Declares a local variable. Available in triggers and stored procedures.

**Syntax** `DECLARE VARIABLE var datatype;`

| Argument        | Description                                                                   |
|-----------------|-------------------------------------------------------------------------------|
| <i>var</i>      | Name of the local variable, unique within the trigger or procedure            |
| <i>datatype</i> | Data type of the local variable; can be any InterBase data type except arrays |

---

## EXCEPTION

**Description** Local variables are declared and used within a stored procedure. They have no effect outside the procedure.

Local variables must be declared at the beginning of a procedure body before they can be used. Each local variable requires a separate `DECLARE VARIABLE` statement, followed by a semicolon (;).

**Example** The following header declares the local variable, `ANY_SALES`:

```
CREATE PROCEDURE DELETE_EMPLOYEE (EMP_NUM INTEGER)
AS
 DECLARE VARIABLE ANY_SALES INTEGER;
BEGIN
 . . .
```

**See also** [Input Parameters](#), [Output Parameters](#)

## EXCEPTION

Raises the specified exception. Available in triggers and stored procedures.

**Syntax** `EXCEPTION` *name*;

| Argument    | Description                        |
|-------------|------------------------------------|
| <i>name</i> | Name of the exception being raised |

**Description** An exception is a user-defined error that has a name and an associated text message. When raised, an exception:

- Terminates the procedure or trigger in which it was raised and undoes any actions performed (directly or indirectly) by the procedure or trigger.
- Returns an error message to the calling application. In **isql**, the error message is displayed to the screen.

Exceptions can be handled with the `WHEN` statement. If an exception is handled, it will behave differently.

**Example** The following **isql** statement defines an exception named `REASSIGN_SALES`:

```
CREATE EXCEPTION REASSIGN_SALES
 'Reassign the sales records before deleting this
 employee.' ;
```

Then these statements from a procedure body raise the exception:

```
IF (ANY_SALES > 0) THEN
 EXCEPTION REASSIGN_SALES;
```

**See also** `WHEN ... DO`

For more information on creating exceptions, see [CREATE EXCEPTION](#) on page 2-46.

## EXECUTE PROCEDURE

Executes a stored procedure. Available in triggers and stored procedures.

**Syntax** EXECUTE PROCEDURE *name* [:*param* [, :*param* ...]]  
[RETURNING\_VALUES :*param* [, :*param* ...]];

| Argument                                                | Description                                                                                                                                                                                                                |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name</i>                                             | Name of the procedure being executed. Must have been previously defined to the database with CREATE PROCEDURE                                                                                                              |
| [ <i>param</i> [, <i>param</i> ...]]                    | List of input parameters, if the procedure requires them <ul style="list-style-type: none"> <li>• Can be constants or variables</li> <li>• Precede variables with a colon, except NEW and OLD context variables</li> </ul> |
| [RETURNING_VALUES<br><i>param</i> [, <i>param</i> ...]] | List of output parameters, if the procedure returns values; precede each with a colon, except NEW and OLD context variables                                                                                                |

**Description** A stored procedure can itself execute a stored procedure. Each time a stored procedure calls another procedure, the call is said to be *nested* because it occurs in the context of a previous and still active call to the first procedure. A stored procedure called by another stored procedure is known as a *nested procedure*.

If a procedure calls itself, it is *recursive*. Recursive procedures are useful for tasks that involve repetitive steps. Each invocation of a procedure is referred to as an *instance*, since each procedure call is a separate entity that performs as if called from an application, reserving memory and stack space as required to perform its tasks.

**Note** Stored procedures can be nested up to 1,000 levels deep. This limitation helps to prevent infinite loops that can occur when a recursive procedure provides no absolute terminating condition. Nested procedure calls may be restricted to fewer than 1,000 levels by memory and stack limitations of the server.

**Example** The following example illustrates a recursive procedure, FACTORIAL, which calculates factorials. The procedure calls itself recursively to calculate the factorial of NUM, the input parameter.

```
CREATE PROCEDURE FACTORIAL (NUM INT)
 RETURNS (N_FACTORIAL DOUBLE PRECISION)
 AS
 DECLARE VARIABLE NUM_LESS_ONE INT;
```

## EXECUTE STATEMENT

```
BEGIN
 IF (NUM = 1) THEN
 BEGIN /**** Base case: 1 factorial is 1 ****/
 N_FACTORIAL = 1;
 EXIT;
 END
 ELSE
 BEGIN
 /**** Recursion: num factorial = num * (num-1) factorial
****/
 NUM_LESS_ONE = NUM - 1;
 EXECUTE PROCEDURE FACTORIAL NUM_LESS_ONE
 RETURNING_VALUES N_FACTORIAL;
 N_FACTORIAL = N_FACTORIAL * NUM;
 END
 END;
END;
```

**See also** CREATE PROCEDURE, [Input Parameters](#), [Output Parameters](#)

For more information on executing procedures, see EXECUTE PROCEDURE on page 2-105.

## EXECUTE STATEMENT

---

Embedding a variation of EXECUTE STATEMENTS within a Stored Procedure.

**Description** Store procedure developers can now embed three variations of EXECUTE STATEMENT within their Stored Procedures. The variations depend on the number of rows returned from the EXECUTE STATEMENT command. The variations are: No rows or data returned, One row of data returned, and Any number of data rows returned.

**See also** For information on requirements and constraints, see [Requirements and Constraints](#).

**See also** For information on error messages added with the function, see [Chapter 5: “Error Codes and Messages”](#)

### No Rows or Data Returned

---

**Syntax** EXECUTE STATEMENT <statement>

<statement> ::= a SQL statement returning no rows of data

**Examples** CREATE PROCEDURE EXEC\_STMT\_NO\_RET (proc\_name varchar(20))  
AS  
DECLARE VARIABLE EMPNO INTEGER;  
DECLARE VARIABLE EXECSTMT VARCHAR(150);  
BEGIN  
SELECT MAX(EMP\_NO) from EMPLOYEE into EMPNO;  
EXECSTMT = 'EXECUTE PROCEDURE' || proc\_name || '(' || cast (EMPNO as  
varchar(10)) || ')';  
EXECUTE STATEMENT EXECSTMT;  
END

## One Row of Data Returned

---

**Syntax** EXECUTE STATEMENT <select-statement> INTO :<var> [, :<var> ..]  
<select-statement> ::= SQL statement returning one or no rows of data  
<var> ::= valid procedure variable, the ":" is optional.

**Example** CREATE PROCEDURE EXEC\_STMT\_SINGLETON (TABLE\_NAME  
VARCHAR(50))  
AS  
DECLARE VARIABLE MAXEMPNO INTEGER;  
BEGIN  
EXECUTE STATEMENT 'SELECT MAX(EMP\_NO) FROM ' || TABLE\_NAME  
INTO :MAXEMPNO;  
END

## Any Number of Data Rows Returned

---

**Syntax** FOR EXECUTE STATEMENT <select-statement> INTO :<var> [, :<var> ..]  
DO <compound-statement>  
<select-statement> ::= SQL statement returning one or no rows of data  
<var> ::= valid procedure variable, the ":" is optional..

**Example** CREATE PROCEDURE EXEC\_STMT\_ANY (TABLE\_NAME VARCHAR(50),  
INT\_FIELD INTEGER)  
RETURNS  
(INT\_RETVAR INTEGER)  
AS  
DECLARE VARIABLE IFIELD INTEGER;  
BEGIN  
FOR EXECUTE STATEMENT  
'SELECT ' || INT\_FIELD || ' FROM ' || TABLE\_NAME INTO :IFIELD  
DO

```
FOR SELECT...DO
```

```
 IF (IFIELD = 0) THEN
 INT_RETVAR = 0;
 ELSE
 INT_RETVAR = INT_RETVAR + IFIELD;
 SUSPEND;
 END
```

## Requirements and Constraints

---

There are constraints and peculiarities with using EXECUTE STATEMENTS:

- The Statement is "prepared" every time it is executed which affects the performance of the Stored Procedure.
- No checks are done on the statement when the procedure is created; dependency checks are not done when the procedure is created, also the checks for existence of tables or column names referred in the execute statement are not performed. All these checks are done at execute time and results in errors if a error condition occurs.
- The feature can be used to perform DDL operations.
- All statements are executed based on the privileges of the user executing the Stored Procedure.
- SQL statements, "COMMIT:", "COMMIT RETAIN", "ROLLBACK", "ROLLBACK RETAIN" and "CREATE DATABASE" are not supported with EXECUTE STATEMENT. These statements return the error code isc\_exec\_stmt\_disallow.

## FOR SELECT...DO

---

Repeats a block or statement for each row retrieved by the SELECT statement. Available in triggers and stored procedures.

**Syntax**    `FOR <select_expr>`  
              `DO <compound_statement>`

| Argument                  | Description                                                                                            |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>select_expr</i>        | SELECT statement that retrieves rows from the database; the INTO clause is required and must come last |
| <i>compound_statement</i> | Statement or block executed once for each row retrieved by the SELECT statement                        |

---

**Description**    FOR SELECT is a loop statement that retrieves the row specified in the *select\_expr* and performs the statement or block following DO for each row retrieved.

The *select\_expr* is a normal SELECT, except the INTO clause is required and must be the last clause.

**Example** The following **isql** statement selects department numbers into the local variable, RDNO, which is then used as an input parameter to the DEPT\_BUDGET procedure:

```
FOR SELECT DEPT_NO
 FROM DEPARTMENT
 WHERE HEAD_DEPT = :DNO
 INTO :RDNO
DO
 BEGIN
 EXECUTE PROCEDURE DEPT_BUDGET :RDNO RETURNING_VALUES
 :SUMB;
 TOT = TOT + SUMB;
 END
```

**See also** SELECT

## IF...THEN ... ELSE

---

Conditional statement that performs a block or statement in the IF clause if the specified condition is TRUE, otherwise performs the block or statement in the optional ELSE clause. Available in triggers and stored procedures.

**Syntax** IF (<condition>  
           THEN <compound\_statement>  
           [ELSE <compound\_statement>]

| Argument                          | Description                                                                                   |
|-----------------------------------|-----------------------------------------------------------------------------------------------|
| <i>condition</i>                  | Boolean expression that evaluates to TRUE, FALSE, or UNKNOWN; must be enclosed in parentheses |
| THEN<br><i>compound_statement</i> | Statement or block executed if <i>condition</i> is TRUE                                       |
| ELSE<br><i>compound_statement</i> | Optional statement or block executed if <i>condition</i> is not TRUE                          |

**Description** The IF ... THEN ... ELSE statement selects alternative courses of action by testing a specified condition.

*condition* is an expression that must evaluate to TRUE to execute the statement or block following THEN. The optional ELSE clause specifies an alternative statement or block executed if *condition* is not TRUE.

**Example** The following lines of code illustrate the use of IF... THEN, assuming the variables LINE2, FIRST, and LAST have been previously declared:

```

. . .
IF (FIRST IS NOT NULL) THEN
 LINE2 = FIRST || ' ' || LAST;
ELSE
 LINE2 = LAST;
. . .

```

**See also** WHILE ... DO

## Input Parameters

---

Used to pass values from an application to a stored procedure. Available in stored procedures only.

**Syntax** CREATE PROCEDURE *name*  
[(*param datatype* [, *param datatype* ...])]

**Description** Input parameters are used to pass values from an application to a stored procedure. They are declared in a comma-delimited list in parentheses following the procedure name in the header of CREATE PROCEDURE. Once declared, they can be used in the procedure body anywhere a variable can appear.

Input parameters are passed *by value* from the calling program to a stored procedure. This means that if the procedure changes the value of an input variable, the change has effect only within the procedure. When control returns to the calling program, the input variable will still have its original value.

Input parameters can be of any InterBase data type. However, arrays of data types are not supported.

**Example** The following procedure header, from an **isql** script, declares two input parameters, EMP\_NO and PROJ\_ID:

```

CREATE PROCEDURE ADD_EMP_PROJ (EMP_NO SMALLINT, PROJ_ID
CHAR(5))
AS
. . .

```

**See also** DECLARE VARIABLE

For more information on declaring input parameters in a procedure header, see [CREATE PROCEDURE](#) on page 2-54.

## NEW Context Variables

---

Indicates a new column value in an INSERT or UPDATE operation. Available only in triggers.

**Syntax** NEW.*column*

| Argument      | Description                          |
|---------------|--------------------------------------|
| <i>column</i> | Name of a column in the affected row |

- Description** Triggers support two context variables: OLD and NEW. A NEW context variable refers to the new value of a column in an INSERT or UPDATE operation.
- Context variables are often used to compare the values of a column before and after it is modified. Context variables can be used anywhere a regular variable can be used.
- New values for a row can only be altered *before* actions. A trigger that fires after INSERT and tries to assign a value to NEW.*column* will have no effect. However, the actual column values are not altered until after the action, so triggers that reference values from their target tables will not see a newly inserted or updated value unless they fire after UPDATE or INSERT.

- Example** The following script is a trigger that fires after the EMPLOYEE table is updated, and compares an employee's old and new salary. If there is a change in salary, the trigger inserts an entry in the SALARY\_HISTORY table.

```
CREATE TRIGGER SAVE_SALARY_CHANGE FOR EMPLOYEE
AFTER UPDATE AS
BEGIN
 IF (OLD.SALARY <> NEW.SALARY) THEN
 INSERT INTO SALARY_HISTORY
 (EMP_NO, CHANGE_DATE, UPDATER_ID, OLD_SALARY,
 PERCENT_CHANGE)
 VALUES (OLD.EMP_NO, 'NOW', USER, OLD.SALARY,
 (NEW.SALARY - OLD.SALARY) * 100 / OLD.SALARY);
 END ;
```

- See also** [OLD Context Variables](#)

For more information on creating triggers, see [CREATE TRIGGER](#) on page 2-71.

## OLD Context Variables

Indicates a current column value in an UPDATE or DELETE operation. Available in triggers only.

- Syntax** OLD.*column*

| Argument      | Description                          |
|---------------|--------------------------------------|
| <i>column</i> | Name of a column in the affected row |

**Description** Triggers support two context variables: OLD and NEW. An OLD context variable refers to the current or previous value of a column in an INSERT or UPDATE operation.

Context variables are often used to compare the values of a column before and after it is modified. Context variables can be used anywhere a regular variable can be used.

**Example** The following script is a trigger that fires after the EMPLOYEE table is updated, and compares an employee's old and new salary. If there is a change in salary, the trigger inserts an entry in the SALARY\_HISTORY table.

```
CREATE TRIGGER SAVE_SALARY_CHANGE FOR EMPLOYEE
AFTER UPDATE AS
BEGIN
 IF (OLD.SALARY <> NEW.SALARY) THEN
 INSERT INTO SALARY_HISTORY
 (EMP_NO, CHANGE_DATE, UPDATER_ID, OLD_SALARY,
 PERCENT_CHANGE)
 VALUES (OLD.EMP_NO, 'NOW', USER, OLD.SALARY,
 (NEW.SALARY - OLD.SALARY) * 100 / OLD.SALARY);
 END ;
```

**See also** [NEW Context Variables](#)

For more information about creating triggers, see [CREATE TRIGGER](#) on page 2-71.

## Output Parameters

---

Used to return values from a stored procedure to the calling application. Available in stored procedures only.

**Syntax** CREATE PROCEDURE *name*  
[(*param datatype* [, *param datatype* ...])]  
[RETURNS (*param datatype* [, *param datatype* ...])]

**Description** Output parameters are used to return values from a procedure to the calling application. They are declared in a comma-delimited list in parentheses following the RETURNS keyword in the header of CREATE PROCEDURE. Once declared, they can be used in the procedure body anywhere a variable can appear. They can be of any InterBase data type. Arrays of data types are not supported.

If output parameters are declared in a procedure's header, the procedure must assign them values to return to the calling application. Values can be derived from any valid expression in the procedure.

A procedure returns output parameter values to the calling application with a `SUSPEND` statement. An application receives values of output parameters from a select procedure by using the `INTO` clause of the `SELECT` statement. An application receives values of output parameters from an executable procedure by using the `RETURNING_VALUES` clause.

In a `SELECT` statement that retrieves values from a procedure, the column names must match the names and datatypes of the procedure's output parameters. In an `EXECUTE PROCEDURE` statement, the output parameters need not match the names of the procedure's output parameters, but the datatypes must match.

**Example** The following `isql` script is a procedure header declares five output parameters, `HEAD_DEPT`, `DEPARTMENT`, `MNGR_NAME`, `TITLE`, and `EMP_CNT`:

```
CREATE PROCEDURE ORG_CHART RETURNS (HEAD_DEPT CHAR(25),
DEPARTMENT
CHAR(25), MNGR_NAME CHAR(20), TITLE CHAR(5), EMP_CNT
INTEGER)
```

**See also** For more information on declaring output parameters in a procedure, see [CREATE PROCEDURE](#) on page 2-54.

## POST\_EVENT

Posts an event. Available in triggers and stored procedures.

**Syntax** `POST_EVENT 'event_name' | col | variable;`

| Argument                  | Description                                                  |
|---------------------------|--------------------------------------------------------------|
| <code>'event_name'</code> | Name of the event being posted; must be enclosed in quotes   |
| <code>col</code>          | Name of a column whose value the posting will be based on    |
| <code>variable</code>     | Name of a string variable in the stored procedure or trigger |

**Description** `POST_EVENT` posts an event to the event manager. When an event occurs, this statement will notify the event manager, which alerts applications waiting for the named event.

**Example** The following statement posts an event named "new\_order":

```
POST_EVENT 'new_order';
```

The next statement posts an event based on the current value of a column:

```
POST_EVENT NEW.COMPANY;
```

The next statement posts an event based on a string variable previously declared:

```
myval = 'new_order:' || NEW.COMPANY;
POST_EVENT myval;
```

## SELECT

**See also** EVENT INIT, EVENT WAIT

For more information on events, see the [Embedded SQL Statement](#).

# SELECT

---

Retrieves a single row that satisfies the requirements of the search condition. The same as standard singleton SELECT, with some differences in syntax. Available in triggers and stored procedures.

```
<select_expr> = <select_clause> <from_clause>
 [<where_clause>] [<group_by_clause>]
 [<having_clause>]
 [<union_expression>] [<plan_clause>]
 [<ordering_clause>]
 <into_clause>;
```

**Description** In a stored procedure, use the SELECT statement with an INTO clause to retrieve a single row value from the database and assign it to a host variable. The SELECT statement must return at most one row from the database, like a standard singleton SELECT. The INTO clause is required and must be the last clause in the statement.

The INTO clause comes at the end of the SELECT statement to allow the use of UNION operators. UNION is not allowed in singleton SELECT statements in embedded SQL.

**Example** The following statement is a standard singleton SELECT statement in an embedded application:

```
EXEC SQL
 SELECT SUM(BUDGET) , AVG(BUDGET)
 INTO :TOT_BUDGET, :AVG_BUDGET
 FROM DEPARTMENT
 WHERE HEAD_DEPT = :HEAD_DEPT
```

To use the above SELECT statement in a procedure, move the INTO clause to the end as follows:

```
SELECT SUM(BUDGET) , AVG(BUDGET)
 FROM DEPARTMENT
 WHERE HEAD_DEPT = :HEAD_DEPT
 INTO :TOT_BUDGET, :AVG_BUDGET;
```

**See also** FOR SELECT...DO

For a complete explanation of the standard SELECT syntax, see [SELECT](#) on page 2-127.

# SUSPEND

Suspends execution of a select procedure until the next FETCH is issued and returns values to the calling application. Available in stored procedures only.

**Syntax**    `SUSPEND;`

**Description**    The SUSPEND statement:

- Suspends execution of a stored procedure until the application issues the next FETCH.
- Returns values of output parameters, if any.

A procedure should ensure that all output parameters are assigned values before a SUSPEND.

SUSPEND should not be used in an executable procedure. Use EXIT instead to indicate to the reader explicitly that the statement terminates the procedure.

The following table summarizes the behavior of SUSPEND, EXIT, and END.

**Table 3.1** SUSPEND, EXIT, and END

| Procedure type       | SUSPEND                                                                                                                                      | EXIT               | END                                                                                                                                   |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Select procedure     | <ul style="list-style-type: none"><li>• Suspends execution of procedure until next FETCH is issued</li><li>• Returns output values</li></ul> | Jumps to final END | <ul style="list-style-type: none"><li>• Returns control to application</li><li>• Sets SQLCODE to 100 (end of record stream)</li></ul> |
| Executable procedure | <ul style="list-style-type: none"><li>• Jumps to final END</li><li>• Not recommended</li></ul>                                               | Jumps to final END | <ul style="list-style-type: none"><li>• Returns values</li><li>• Returns control to application</li></ul>                             |

**Note**    If a SELECT procedure has executable statements following the last SUSPEND in the procedure, all of those statements are executed, even though no more rows are returned to the calling program. The procedure terminates with the final END statement, which sets SQLCODE to 100.

The SUSPEND statement also delimits atomic statement blocks in select procedures. If an error occurs in a select procedure—either a SQLCODE error, GDSCODE error, or exception—the statements executed since the last SUSPEND are undone. Statements before the last SUSPEND are never undone, unless the transaction comprising the procedure is rolled back.

**Example**    The following procedure illustrates the use of SUSPEND and EXIT:

```
CREATE PROCEDURE P RETURNS (R INTEGER)
AS
BEGIN
 R = 0;
```

## WHEN ... DO

```
WHILE (R < 5) DO
BEGIN
 R = R + 1;
 SUSPEND;
 IF (R = 3) THEN
 EXIT;
 END
END;
```

If this procedure is used as a select procedure in **isql**, for example,

```
SELECT * FROM P;
```

then it will return values 1, 2, and 3 to the calling application, since the SUSPEND statement returns the current value of *r* to the calling application until *r* = 3, when the procedure performs an EXIT and terminates.

If the procedure is used as an executable procedure in **isql**, for example,

```
EXECUTE PROCEDURE P;
```

then it will return 1, since the SUSPEND statement will terminate the procedure and return the current value of *r* to the calling application. Since SUSPEND should not be used in executable procedures, EXIT would be used instead, indicating that when the statement is encountered, the procedure is exited.

**See also** No Rows or Data Returned, BEGIN ... END

## WHEN ... DO

---

Error-handling statement that performs the statements following DO when the specified error occurs. Available in triggers and stored procedures.

**Syntax** WHEN {<error> [, <error> ...] | ANY}  
DO <compound\_statement>

<error>=  
{EXCEPTION *exception\_name* | SQLCODE *number* | GDSCODE  
*errcode*}

| Argument                           | Description                                      |
|------------------------------------|--------------------------------------------------|
| EXCEPTION<br><i>exception_name</i> | The name of an exception already in the database |
| SQLCODE <i>number</i>              | A SQLCODE error code number                      |

---

| Argument                  | Description                                                                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| GDSCODE <i>errcode</i>    | An InterBase error code. Use Table 5.5 and strip "isc_" before mentioning the errorcode with GDSCODE usage. For example: GDSCODE lock_conflict. |
| ANY                       | Keyword that handles any of the above types of errors                                                                                           |
| <i>compound_statement</i> | Statement or block executed when any of the specified errors occur.                                                                             |

**Important** If used, WHEN must be the last statement in a BEGIN...END block. It should come after SUSPEND, if present.

**Description** Procedures can handle three kinds of errors with a WHEN statement:

- Exceptions raised by EXCEPTION statements in the current procedure, in a nested procedure, or in a trigger fired as a result of actions by such a procedure.
- SQL errors reported in SQLCODE.
- InterBase error codes.

The WHEN ANY statement handles any of the three types.

## Handling Exceptions

Instead of terminating when an exception occurs, a procedure can respond to and perhaps correct the error condition by handling the exception. When an exception is raised, it:

- Terminates execution of the BEGIN ... END block containing the exception and undoes any actions performed in the block.
- Backs out one level to the next BEGIN ... END block and seeks an exception-handling (WHEN) statement, and continues backing out levels until one is found. If no WHEN statement is found, the procedure is terminated and all its actions are undone.
- Performs the ensuing statement or block of statements specified after WHEN, if found.
- Returns program control to the block or statement in the procedure following the WHEN statement.

**Note** An exception that is handled with WHEN does not return an error message.

## Handling SQL Errors

---

Procedures can also handle error numbers returned in SQLCODE. After each SQL statement executes, SQLCODE contains a status code indicating the success or failure of the statement. It can also contain a warning status, such as when there are no more rows to retrieve in a FOR SELECT loop.

## Handling InterBase Error Codes

---

Procedures can also handle InterBase error codes. For example, suppose a statement in a procedure attempts to update a row already updated by another transaction, but not yet committed. In this case, the procedure might receive an InterBase error code, `isc_lock_conflict`. Perhaps if the procedure retries its update, the other transaction may have rolled back its changes and released its locks. By using a WHEN GDSCODE statement, the procedure can handle lock conflict errors and retry its operation.

**Example** For example, if a procedure attempts to insert a duplicate value into a column defined as a PRIMARY KEY, InterBase will return SQLCODE -803. This error can be handled in a procedure with the following statement:

```
WHEN SQLCODE -803
DO
 BEGIN
 . . .
```

For example, the following procedure, from an **isql** script, includes a WHEN statement to handle errors that may occur as the procedure runs. If an error occurs and SQLCODE is as expected, the procedure continues with the new value of B. If not, the procedure cannot handle the error, and rolls back all actions of the procedure, returning the active SQLCODE.

```
CREATE PROCEDURE NUMBERPROC (A INTEGER) RETURNS (B INTEGER)
AS
 BEGIN
 B = 0;
 BEGIN
 UPDATE R SET F1 = F1 + :A;
 UPDATE R SET F2 = F2 * F2;
 UPDATE R SET F1 = F1 + :A;
 WHEN SQLCODE -803 DO
 B = 1;
 END
 EXIT;
 END;
```

**See also** EXCEPTION

For more information about InterBase error codes and SQLCODE values, see [Chapter 5, “Error Codes and Messages.”](#)

# WHILE ... DO

---

Performs the statement or block following DO as long as the specified condition is TRUE. Available in triggers and stored procedures.

**Syntax**    `WHILE (<condition>) DO`  
               `<compound_statement>`

| Argument                  | Description                                                                            |
|---------------------------|----------------------------------------------------------------------------------------|
| <i>condition</i>          | Boolean expression tested before each execution of the statement or block following DO |
| <i>compound_statement</i> | Statement or block executed as long as <i>condition</i> is TRUE                        |

**Description**    WHILE ... DO is a looping statement that repeats a statement or block of statements as long as a condition is true. The condition is tested at the start of each loop.

**Example**        The following procedure, from an **isql** script, uses a WHILE ... DO loop to compute the sum of all integers from one up to the input parameter:

```
CREATE PROCEDURE SUM_INT (I INTEGER) RETURNS (S INTEGER)
AS
BEGIN
 S = 0;
 WHILE (I > 0) DO
 BEGIN
 S = S + I;
 I = I - 1;
 END
END;
```

If this procedure is called from **isql** with the command:

```
EXECUTE PROCEDURE SUM_INT 4;
```

then the results will be:

```
S
=====
10
```

**See also**        IF...THEN ... ELSE, FOR SELECT...DO

WHILE ... DO

# Keywords

The table in this chapter lists *keywords*, words reserved from use in SQL programs and **isql** (Interactive SQL). The list includes DSQL, **isql**, and **gpre** keywords.

Keywords are defined for special purposes, and are sometimes called reserved words. A keyword cannot occur in a user-declared identifier or as the name of a table, column, index, trigger, or constraint, unless it is enclosed in double quotes. Keywords are:

- Part of statements
- Used as statements
- Names of standard data structures or datatypes

## InterBase Keywords

---

|           |                  |             |           |
|-----------|------------------|-------------|-----------|
| ACTION    | ACTIVE           | ADD         | ADMIN     |
| AFTER     | ALL              | ALTER       | AND       |
| ANY       | AS               | ASC         | ASCENDING |
| AT        | AUTO             | AUTODDL     | AVG       |
| BASED     | BASENAME         | BASE_NAME   | BEFORE    |
| BEGIN     | BETWEEN          | BLOB        | BLOBEDIT  |
| BOOLEAN   | BUFFER           | BY          | CACHE     |
| CASCADE   | CASE             | CAST        | CHAR      |
| CHARACTER | CHARACTER_LENGTH | CHAR_LENGTH | CHECK     |

## InterBase Keywords

|                 |                    |                        |             |
|-----------------|--------------------|------------------------|-------------|
| CHECK_POINT_LEN | CHECK_POINT_LENGTH | COALESCE               | COLLATE     |
| COLLATION       | COLUMN             | COMMIT                 | COMMITTED   |
| COMPILETIME     | COMPUTED           | CLOSE                  | CONDITIONAL |
| CONNECT         | CONSTRAINT         | CONTAINING             | CONTINUE    |
| COUNT           | CREATE             | CSTRING                | CURRENT     |
| CURRENT_DATE    | CURRENT_TIME       | CURRENT_TIMESTAMP      | CURSOR      |
| DATABASE        | DATE               | DAY                    | DB_KEY      |
| DEBUG           | DEC                | DECIMAL                | DECLARE     |
| DECRYPT         | DEFAULT            | DELETE                 | DESC        |
| DESCENDING      | DESCRIBE           | DESCRIPTOR             | DISCONNECT  |
| DISPLAY         | DISTINCT           | DO                     | DOMAIN      |
| DOUBLE          | DROP               | ECHO                   | EDIT        |
| ELSE            | ENCRYPT            | ENCRYPTION             | END         |
| ENTRY_POINT     | ESCAPE             | EVENT                  | EXCEPTION   |
| EXECUTE         | EXISTS             | EXIT                   | EXTERN      |
| EXTERNAL        | EXTRACT            | FALSE                  | FETCH       |
| FILE            | FILTER             | FLOAT                  | FOR         |
| FOREIGN         | FOUND              | FREE_IT                | FROM        |
| FULL            | FUNCTION           | GDSCODE                | GENERATOR   |
| GEN_ID          | GLOBAL             | GOTO                   | GRANT       |
| GROUP           | GROUP_COMMIT_WAIT  | GROUP_COMMIT_WAIT_TIME | HAVING      |
| HELP            | HOUR               | IF                     | IMMEDIATE   |
| IN              | INACTIVE           | INDEX                  | INDICATOR   |
| INIT            | INNER              | INPUT                  | INPUT_TYPE  |
| INSERT          | INT                | INTEGER                | INTO        |
| IS              | ISOLATION          | ISQL                   | JOIN        |
| KEY             | LC_MESSAGES        | LC_TYPE                | LEFT        |
| LENGTH          | LEV                | LEVEL                  | LIKE        |

|                  |                 |              |                 |
|------------------|-----------------|--------------|-----------------|
| LOGFILE          | LOG_BUFFER_SIZE | LOG_BUF_SIZE | LONG            |
| MANUAL           | MAX             | MAXIMUM      | MAXIMUM_SEGMENT |
| MAX_SEGMENT      | MERGE           | MESSAGE      | MIN             |
| MINIMUM          | MINUTE          | MODULE_NAME  | MONTH           |
| NAMES            | NATIONAL        | NATURAL      | NCHAR           |
| NO               | NOAUTO          | NOT          | NULL            |
| NULLIF           | NUMERIC         | NUM_LOG_BUFS | NUM_LOG_BUFFERS |
| OCTET_LENGTH     | OF              | ON           | ONLY            |
| OPEN             | OPTION          | OR           | ORDER           |
| OUTER            | OUTPUT          | OUTPUT_TYPE  | OVERFLOW        |
| PAGE             | PAGELength      | PAGES        | PAGE_SIZE       |
| PARAMETERS       | PASSWORD        | PERCENT      | PLAN            |
| POSITION         | POST_EVENT      | PRECISION    | PREPARE         |
| PRESERVE         | PROCEDURE       | PROTECTED    | PRIMARY         |
| PRIVILEGES       | PUBLIC          | QUIT         |                 |
| RAW_PARTITIONS   | RDB\$DB_KEY     | READ         | REAL            |
| RECORD_VERSION   | REFERENCES      | RELEASE      | RESERV          |
| RESERVING        | RESTRICT        | RETAIN       | RETURN          |
| RETURNING_VALUES | RETURNS         | REVOKE       | RIGHT           |
| ROLE             | ROLLBACK        | ROWS         | RUNTIME         |
| SCHEMA           | SECOND          | SEGMENT      | SELECT          |
| SET              | SHADOW          | SHARED       | SHELL           |
| SHOW             | SINGULAR        | SIZE         | SMALLINT        |
| SNAPSHOT         | SOME            | SORT         | SQLCODE         |
| SQLERROR         | SQLWARNING      | STABILITY    | STARTING        |
| STARTS           | STATEMENT       | STATIC       | SUSPEND         |
| TABLE            | TEMPORARY       | TERMINATOR   | THEN            |
| TIES             | TIME            | TIMESTAMP    | TO              |
| TRANSACTION      | TRANSLATE       | TRANSLATION  | TRIGGER         |

## InterBase Keywords

|         |          |          |             |
|---------|----------|----------|-------------|
| TRIM    | TRUE     | TYPE     | UNCOMMITTED |
| UNION   | UNIQUE   | UNKNOWN  | UPDATE      |
| UPPER   | USER     | USING    | VALUE       |
| VALUES  | VARCHAR  | VARIABLE | VARYING     |
| VERSION | VIEW     | WAIT     | WEEKDAY     |
| WHEN    | WHENEVER | WHERE    | WHILE       |
| WITH    | WORK     | WRITE    | YEAR        |
| YEARDAY |          |          |             |

**Note** The following keywords are specific to InterBase and are not part of the SQL standard:

WEEKDAY                      YEARDAY

# Error Codes and Messages

This chapter summarizes InterBase error-handling options and error codes. Tables in this chapter list SQLCODE and InterBase error codes and messages for embedded SQL, dynamic SQL (DSQL), and interactive SQL (**isql**). For a detailed discussion of error handling, see the [Embedded SQL Statement](#).

## Error Sources

Run-time errors occur at points of user input or program output. When you run a program or use **isql**, the following types of errors may occur:

| Error type                 | Description                                                                                                                       | Action                                                                                                                                                                |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Database error             | Database errors can result from any one of many problems, such as conversion errors, arithmetic exceptions, and validation errors | If you encounter one of these messages: <ul style="list-style-type: none"><li>• Check any messages</li><li>• Check the file name or path name and try again</li></ul> |
| Bugcheck or internal error | Bugchecks reflect software problems you should report                                                                             | If you encounter a bugcheck, execute a traceback and save the output; submit output and script along with a copy of the database to InterBase Software Corp.          |

## Error Reporting and Handling

---

For reporting and dealing with errors, InterBase utilizes the SQLCODE variable and InterBase codes returned in the status array.

Every executable SQL statement sets the SQLCODE variable, which can serve as a status indicator. During preprocessing, **gpre** declares this variable automatically. An application can test for and use the SQLCODE variable in one of three ways:

- Use the WHENEVER statement to check the value of SQLCODE and direct the program to branch to error-handling routines coded in the application.
- Test for SQLCODE directly.
- Combine WHENEVER and direct SQLCODE testing.

For SQL programs that must be portable between InterBase and other database management systems, limit error-handling routines to one of these methods.

The InterBase status array displays information about errors that supplements SQLCODE messages.

InterBase applications can check both the SQLCODE message and the message returned in the status array.

### Trapping Errors with WHENEVER

---

The WHENEVER statement traps SQL errors and warnings. WHENEVER tests SQLCODE return values and branches to appropriate error-handling routines in the application. Error routines can range from:

- Simple reporting of errors and transaction rollback, or a prompt to the user to reenter a query or data.
- More sophisticated routines that react to many possible error conditions in predictable ways.

WHENEVER helps limit the size of an application, since it can call on a single suite of routines for handling errors and warnings.

### Checking SQLCODE Value Directly

---

Applications can test directly for a particular SQLCODE after each SQL statement. If that SQLCODE occurs, the program can branch to a specific routine.

To handle specific error situations, combine checking for SQLCODE with general WHENEVER statements. These steps outline the procedure, which is described in detail in the [Embedded SQL Statement](#):

- 1 Override the WHENEVER branching by inserting a WHENEVER SQLERROR CONTINUE statement. The program now ignores SQLCODE.

- 2 Use a SQLCODE-checking statement to check for a particular SQLCODE and direct the program to an alternative procedure.
- 3 To return to WHENEVER branching, insert a new WHENEVER statement.

Where portability is not an issue, additional information may be available in the InterBase status array.

## InterBase Status Array

---

Since each SQLCODE value can result from more than one type of error, the InterBase *status array* (`isc_status`) provides additional messages that enable further inquiry into SQLCODE errors.

**gpre** automatically declares `isc_status`, an array of twenty 32-bit integers, for all InterBase applications during preprocessing. When an error occurs, the status array is loaded with InterBase error codes, message string addresses, and sometimes other numeric, interpretive, platform-specific error data.

This chapter lists all status array codes in “[SQLCODE Error Codes and Messages](#)” on page 5-5. To see the codes online, display the `ibase.h` file. The location of this file is system-specific.

### Access to Status Array Messages

InterBase provides the following library functions for retrieving and printing status array codes and messages.

#### **isc\_print\_sqlerror()**

When `SQLCODE < 0`, this function prints the returned SQLCODE value, the corresponding SQL error message, and any additional InterBase error messages in the status array to the screen. Use within an error-handling routine.

**Syntax** `isc_print_sqlerror (short SQLCODE, ISC_STATUS *status_vector);`

#### **isc\_sql\_interprete()**

This function retrieves a SQL error message and stores it in a user-supplied buffer for later printing, manipulation, or display. Allow a buffer length of 256 bytes to hold the message. Use when building error display routines or if you are using a windowing system that does not permit direct screen writes. Do not use this function when `SQLCODE > 0`.

**Syntax** `isc_sql_interprete(short SQLCODE, char *buffer, short length);`

### Responding to Error Codes

After any error occurs, you have the following options: ignore the error, log the error and continue processing, roll back the transaction and try again, or roll back the transaction and quit the application.

For the following errors, it is recommended that you roll back the current transaction and try the operation again:

**Table 5.1** Status array codes that require rollback and retry

| Status array code        | Action to take                                                                                           |
|--------------------------|----------------------------------------------------------------------------------------------------------|
| <i>isc_convert_error</i> | Conversion error: A conversion between datatypes failed; correct the input and retry the operation       |
| <i>isc_deadlock</i>      | Deadlock: Transaction conflicted with another transaction; wait and try again                            |
| <i>isc_integ_fail</i>    | Integrity check: Operation failed due to a trigger; examine the abort code, fix the error, and try again |
| <i>isc_lock_conflict</i> | Lock conflict: Transaction unable to obtain the locks it needed; wait and try again                      |
| <i>isc_no_dup</i>        | Duplicate index entry: Attempt to add a duplicate field; correct field with duplicate and try again      |
| <i>isc_not_valid</i>     | Validation error: Row did not pass validation test; correct invalid row and try again                    |

## For More Information

The following table is a guide to further information on planning and programming error-handling routines.

**Table 5.2** Where to find error-handling topics

| Topic                                | To find...                                                                                                                             | See...                                                                |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| SQLCODE and error handling           | Complete discussion and programming instructions                                                                                       | <a href="#">Embedded SQL Statement</a>                                |
| List of SQLCODEs                     | SQLCODEs and associated messages for embedded SQL, DSQL, <b>isql</b>                                                                   | This chapter: “SQLCODE Codes and Messages” on page 5-5.               |
| WHENEVER syntax                      | Usage and syntax                                                                                                                       | <a href="#">Chapter 2, “SQL Statement and Function Reference.”</a>    |
| Programming WHENEVER                 | Using and programming error-handling routines                                                                                          | <a href="#">Embedded SQL Statement</a>                                |
| InterBase status array and functions | Complete programming instructions                                                                                                      | <a href="#">Embedded SQL Statement</a>                                |
| List of status array codes           | <ul style="list-style-type: none"> <li>Status array error codes and associated messages for embedded SQL, DSQL, <b>isql</b></li> </ul> | This chapter: “ <a href="#">InterBase Status Array</a> ” on page 5-3. |

# SQLCODE Error Codes and Messages

---

This section lists SQLCODE error codes and associated messages in the following tables:

- SQLCODE error messages summary
- SQLCODE codes and messages

## SQLCODE Error Messages Summary

---

This table summarizes the types of messages SQLCODE can pass to a program:

**Table 5.3** SQLCODE and messages summary

| SQLCODE | Message    | Meaning                                                                                    |
|---------|------------|--------------------------------------------------------------------------------------------|
| <0      | SQLERROR   | Error: The statement did not complete; table 5.4 lists SQLCODE error numbers and messages. |
| 0       | SUCCESS    | Successful completion                                                                      |
| +1–99   | SQLWARNING | System warning or informational message                                                    |
| +100    | NOT FOUND  | No qualifying records found; end of file                                                   |

---

## SQLCODE Codes and Messages

---

The following table lists SQLCODEs and associated messages for SQL and DSQL. Some SQLCODE values have more than one text message associated with them. In these cases, InterBase returns the most relevant string message for the error that occurred.

When code messages include the name of a database object or object type, the name is represented by a code in the SQLCODE Text column:

- *<string>*: String value, such as the name of a database object or object type.
- *<long>*: Long integer value, such as the identification number or code of a database object or object type.
- *<digit>*: Integer value, such as the identification number or code of a database object or object type.
- The InterBase number in the right-hand column is the actual error number returned in the error status vector. You can use InterBase error-handling functions to report messages based on these numbers instead of SQL code, but doing so results in non-portable SQL programs.

**Table 5.4** SQLCODE codes and messages

| SQLCODE | SQLCODE text                                                                                                     | InterBase number |
|---------|------------------------------------------------------------------------------------------------------------------|------------------|
| 101     | Segment buffer length shorter than expected                                                                      | 335544366L       |
| 100     | No match for first value expression                                                                              | 335544338L       |
| 100     | Invalid database key                                                                                             | 335544354L       |
| 100     | Attempted retrieval of more segments than exist                                                                  | 335544367L       |
| 100     | Attempt to fetch past the last record in a record stream                                                         | 335544374L       |
| -84     | Table/procedure has non-SQL security class defined                                                               | 335544554L       |
| -84     | Column has non-SQL security class defined                                                                        | 335544555L       |
| -84     | Procedure <i>&lt;string&gt;</i> does not return any values                                                       | 335544668L       |
| -103    | Datatype for constant unknown                                                                                    | 335544571L       |
| -104    | Invalid request BLR at offset <i>&lt;long&gt;</i>                                                                | 335544343L       |
| -104    | BLR syntax error: expected <i>&lt;string&gt;</i> at offset <i>&lt;long&gt;</i> , encountered <i>&lt;long&gt;</i> | 335544390L       |
| -104    | Context already in use (BLR error)                                                                               | 335544425L       |
| -104    | Context not defined (BLR error)                                                                                  | 335544426L       |
| -104    | Bad parameter number                                                                                             | 335544429L       |
| -104    |                                                                                                                  | 335544440L       |
| -104    | Invalid slice description language at offset <i>&lt;long&gt;</i>                                                 | 335544456L       |
| -104    | Invalid command                                                                                                  | 335544570L       |
| -104    | Internal error                                                                                                   | 335544579L       |
| -104    | Option specified more than once                                                                                  | 335544590L       |
| -104    | Unknown transaction option                                                                                       | 335544591L       |
| -104    | Invalid array reference                                                                                          | 335544592L       |
| -104    | Token unknown—line <i>&lt;long&gt;</i> , char <i>&lt;long&gt;</i>                                                | 335544634L       |
| -104    | Unexpected end of command                                                                                        | 335544608L       |
| -104    | Token unknown                                                                                                    | 335544612L       |
| -150    | Attempted update of read-only table                                                                              | 335544360L       |
| -150    | Cannot update read-only view <i>&lt;string&gt;</i>                                                               | 335544362L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                 | InterBase number |
|---------|------------------------------------------------------------------------------|------------------|
| -150    | Not updatable                                                                | 335544446L       |
| -150    | Cannot define constraints on views                                           | 335544546L       |
| -151    | Attempted update of read-only column                                         | 335544359L       |
| -155    | <string> is not a valid base table of the specified view                     | 335544658L       |
| -157    | Must specify column name for view select expression                          | 335544598L       |
| -158    | Number of columns does not match select list                                 | 335544599L       |
| -162    | Dbkey not available for multi-table views                                    | 335544685L       |
| -170    | Parameter mismatch for procedure <string>                                    | 335544512L       |
| -170    | External functions cannot have more than 10 parameters                       | 335544619L       |
| -171    | Function <string> could not be matched                                       | 335544439L       |
| -171    | Column not array or invalid dimensions (expected <long>, encountered <long>) | 335544458L       |
| -171    | Return mode by value not allowed for this datatype                           | 335544618L       |
| -172    | Function <string> is not defined                                             | 335544438L       |
| -204    | Generator <string> is not defined                                            | 335544463L       |
| -204    | Encryption <string> has bad length of <string> bits                          | 336003096L       |
| -204    | Reference to invalid stream number                                           | 335544502L       |
| -204    | CHARACTER SET <string> is not defined                                        | 335544509L       |
| -204    | Procedure <string> is not defined                                            | 335544511L       |
| -204    | Status code <string> unknown                                                 | 335544515L       |
| -204    | Exception <string> not defined                                               | 335544516L       |
| -204    | Name of Referential Constraint not defined in constraints table.             | 335544532L       |
| -204    | Could not find table/procedure for GRANT                                     | 335544551L       |
| -204    | Implementation of text subtype <digit> not located.                          | 335544568L       |
| -204    | Datatype unknown                                                             | 335544573L       |
| -204    | Table unknown                                                                | 335544580L       |
| -204    | Procedure unknown                                                            | 335544581L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                                | InterBase number |
|---------|---------------------------------------------------------------------------------------------|------------------|
| -204    | COLLATION <string> is not defined                                                           | 335544588L       |
| -204    | COLLATION <string> is not valid for specified CHARACTER SET                                 | 335544589L       |
| -204    | Trigger unknown                                                                             | 335544595L       |
| -204    | Alias <string> conflicts with an alias in the same statement                                | 335544620L       |
| -204    | Alias <string> conflicts with a procedure in the same statement                             | 335544621L       |
| -204    | Alias <string> conflicts with a table in the same statement                                 | 335544622L       |
| -204    | There is no alias or table named <string> at this scope level                               | 335544635L       |
| -204    | There is no index <string> for table <string>                                               | 335544636L       |
| -204    | Invalid use of CHARACTER SET or COLLATE                                                     | 335544640L       |
| -204    | BLOB SUB_TYPE <string> is not defined                                                       | 335544662L       |
| -204    | EXECUTE STATEMENT could not prepare statement : <string>                                    | 335544850        |
| -204    | SQL statement invalid as it returns no records. SQL : <string>                              | 335544851        |
| -204    | Parameter mis-match for the statement : <string>                                            | 335544852        |
| -204    | Could not execute statement : <string>                                                      | 335544853        |
| -204    | EXECUTE STATEMENT fetch error                                                               | 335544854        |
| -204    | EXECUTE STATEMENT in this form must return single row, not multiple rows.                   | 335544855        |
| -204    | Sql statement not allowed in EXECUTE STATEMENT : <string>                                   | 335544857        |
| -204    | Statement evaluated to a NULL statement. EXECUTE STATEMENT cannot execute a NULL statement. | 335544858        |
| -205    | Column <string> is not defined in table <string>                                            | 335544396L       |
| -205    | Could not find column for GRANT                                                             | 335544552L       |
| -206    | Column unknown                                                                              | 335544578L       |
| -206    | Column is not a Blob                                                                        | 335544587L       |
| -206    | Subselect illegal in this context                                                           | 335544596L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                    | InterBase number |
|---------|---------------------------------------------------------------------------------|------------------|
| -208    | Invalid ORDER BY clause                                                         | 335544617L       |
| -219    | Table <string> is not defined                                                   | 335544395L       |
| -239    | Cache length too small                                                          | 335544691L       |
| -260    | Cache redefined                                                                 | 335544690L       |
| -281    | Table <string> is not referenced in plan                                        | 335544637L       |
| -282    | Table <string> is referenced more than once in plan; use aliases to distinguish | 335544638L       |
| -282    | The table <string> is referenced twice; use aliases to differentiate            | 335544643L       |
| -282    | Table <string> is referenced twice in view; use an alias to distinguish         | 335544659L       |
| -282    | View <string> has more than one base table; use aliases to distinguish          | 335544660L       |
| -283    | Table <string> is referenced in the plan but not the from list                  | 335544639L       |
| -284    | Index <string> cannot be used in the specified plan                             | 335544642L       |
| -291    | Column used in a PRIMARY/UNIQUE constraint must be NOT NULL.                    | 335544531L       |
| -292    | Cannot update constraints (RDB\$REF_CONSTRAINTS).                               | 335544534L       |
| -293    | Cannot update constraints (RDB\$CHECK_CONSTRAINTS).                             | 335544535L       |
| -294    | Cannot delete CHECK constraint entry (RDB\$CHECK_CONSTRAINTS)                   | 335544536L       |
| -295    | Cannot update constraints (RDB\$RELATION_CONSTRAINTS).                          | 335544545L       |
| -296    | Internal isc software consistency check (invalid RDB\$CONSTRAINT_TYPE)          | 335544547L       |
| -297    | Operation violates CHECK constraint <string> on view or table                   | 335544558L       |
| -313    | Count of column list and variable list do not match                             | 335544669L       |
| -314    | Cannot transliterate character between character sets                           | 335544565L       |
| -401    | Invalid comparison operator for find operation                                  | 335544647L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                             | InterBase number |
|---------|--------------------------------------------------------------------------|------------------|
| -402    | Attempted invalid operation on a Blob                                    | 335544368L       |
| -402    | Blob and array datatypes are not supported for <string> operation        | 335544414L       |
| -402    | Data operation not supported                                             | 335544427L       |
| -406    | Subscript out of bounds                                                  | 335544457L       |
| -407    | Null segment of UNIQUE KEY                                               | 335544435L       |
| -413    | Conversion error from string "<string>"                                  | 335544334L       |
| -413    | Filter not found to convert type <long> to type <long>                   | 335544454L       |
| -501    | Invalid request handle                                                   | 335544327L       |
| -501    | Attempt to reclose a closed cursor                                       | 335544577L       |
| -502    | Declared cursor already exists                                           | 335544574L       |
| -502    | Attempt to reopen an open cursor                                         | 335544576L       |
| -504    | Cursor unknown                                                           | 335544572L       |
| -508    | No current record for fetch operation                                    | 335544348L       |
| -510    | Cursor not updatable                                                     | 335544575L       |
| -518    | Request unknown                                                          | 335544582L       |
| -519    | The PREPARE statement identifies a prepare statement with an open cursor | 335544688L       |
| -530    | Violation of FOREIGN KEY constraint: "<string>"                          | 335544466L       |
| -530    | Cannot prepare a CREATE DATABASE/SCHEMA statement                        | 335544597L       |
| -532    | Transaction marked invalid by I/O error                                  | 335544469L       |
| -551    | No permission for <string> access to <string> <string>                   | 335544352L       |
| -552    | Only the owner of a table can reassign ownership                         | 335544550L       |
| -552    | User does not have GRANT privileges for operation                        | 335544553L       |
| -553    | Cannot modify an existing user privilege                                 | 335544529L       |
| -595    | The current position is on a crack                                       | 335544645L       |
| -596    | Illegal operation when at beginning of stream                            | 335544644L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                         | InterBase number |
|---------|--------------------------------------------------------------------------------------|------------------|
| -597    | Preceding file did not specify length, so <string> must include starting page number | 335544632L       |
| -598    | Shadow number must be a positive integer                                             | 335544633L       |
| -599    | Gen.c: node not supported                                                            | 335544607L       |
| -600    | A node name is not permitted in a secondary, shadow, cache or log file name          | 335544625L       |
| -600    | Sort error: corruption in data structure                                             | 335544680L       |
| -601    | Database or file exists                                                              | 335544646L       |
| -604    | Array declared with too many dimensions                                              | 335544593L       |
| -604    | Illegal array dimension range                                                        | 335544594L       |
| -605    | Inappropriate self-reference of column                                               | 335544682L       |
| -607    | Unsuccessful metadata update                                                         | 335544351L       |
| -607    | Cannot modify or erase a system trigger                                              | 335544549L       |
| -607    | Array/Blob/DATE/TIME/TIMESTAMP datatypes not allowed in arithmetic                   | 335544657L       |
| -615    | Lock on table <string> conflicts with existing lock                                  | 335544475L       |
| -615    | Requested record lock conflicts with existing lock                                   | 335544476L       |
| -615    | Refresh range number <long> already in use                                           | 335544507L       |
| -616    | Cannot delete PRIMARY KEY being used in FOREIGN KEY definition.                      | 335544530L       |
| -616    | Cannot delete index used by an integrity constraint                                  | 335544539L       |
| -616    | Cannot modify index used by an integrity constraint                                  | 335544540L       |
| -616    | Cannot delete trigger used by a CHECK Constraint                                     | 335544541L       |
| -616    | Cannot delete column being used in an integrity constraint.                          | 335544543L       |
| -616    | There are <long> dependencies                                                        | 335544630L       |
| -616    | Last column in a table cannot be deleted                                             | 335544674L       |
| -617    | Cannot update trigger used by a CHECK Constraint                                     | 335544542L       |
| -617    | Cannot rename column being used in an integrity constraint.                          | 335544544L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                                                | InterBase number |
|---------|-------------------------------------------------------------------------------------------------------------|------------------|
| -618    | Cannot delete index segment used by an integrity constraint                                                 | 335544537L       |
| -618    | Cannot update index segment used by an integrity constraint                                                 | 335544538L       |
| -625    | Validation error for column <i>&lt;string&gt;</i> , value " <i>&lt;string&gt;</i> "                         | 335544347L       |
| -637    | Duplicate specification of <i>&lt;string&gt;</i> not supported                                              | 335544664L       |
| -660    | Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY                                                | 335544533L       |
| -660    | Cannot create index <i>&lt;string&gt;</i>                                                                   | 335544628L       |
| -663    | Segment count of 0 defined for index <i>&lt;string&gt;</i>                                                  | 335544624L       |
| -663    | Too many keys defined for index <i>&lt;string&gt;</i>                                                       | 335544631L       |
| -663    | Too few key columns found for index <i>&lt;string&gt;</i> (incorrect column name?)                          | 335544672L       |
| -664    | key size exceeds implementation restriction for index " <i>&lt;string&gt;</i> "                             | 335544434L       |
| -677    | <i>&lt;string&gt;</i> extension error                                                                       | 335544445L       |
| -685    | Invalid Blob type for operation                                                                             | 335544465L       |
| -685    | Attempt to index Blob column in index <i>&lt;string&gt;</i>                                                 | 335544670L       |
| -685    | Attempt to index array column in index <i>&lt;string&gt;</i>                                                | 335544671L       |
| -689    | Page <i>&lt;long&gt;</i> is of wrong type (expected <i>&lt;long&gt;</i> , found <i>&lt;long&gt;</i> )       | 335544403L       |
| -689    | Wrong page type                                                                                             | 335544650L       |
| -690    | Segments not allowed in expression index <i>&lt;string&gt;</i>                                              | 335544679L       |
| -691    | New record size of <i>&lt;long&gt;</i> bytes is too big                                                     | 335544681L       |
| -692    | Maximum indexes per table ( <i>&lt;digit&gt;</i> ) exceeded                                                 | 335544477L       |
| -693    | Too many concurrent executions of the same request                                                          | 335544663L       |
| -694    | Cannot access column <i>&lt;string&gt;</i> in view <i>&lt;string&gt;</i>                                    | 335544684L       |
| -802    | Arithmetic exception, numeric overflow, or string truncation                                                | 335544321L       |
| -803    | Attempt to store duplicate value (visible to active transactions) in unique index " <i>&lt;string&gt;</i> " | 335544349L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                     | InterBase number |
|---------|----------------------------------------------------------------------------------|------------------|
| -803    | Violation of PRIMARY or UNIQUE KEY constraint:<br>“<string>”                     | 335544665L       |
| -804    | Wrong number of arguments on call                                                | 335544380L       |
| -804    | SQLDA missing or incorrect version, or incorrect number/<br>type of variables    | 335544583L       |
| -804    | Count of columns not equal count of values                                       | 335544584L       |
| -804    | Function unknown                                                                 | 335544586L       |
| -806    | Only simple column names permitted for VIEW WITH<br>CHECK OPTION                 | 335544600L       |
| -807    | No where clause for VIEW WITH CHECK OPTION                                       | 335544601L       |
| -808    | Only one table allowed for VIEW WITH CHECK OPTION                                | 335544602L       |
| -809    | DISTINCT, GROUP or HAVING not permitted for VIEW<br>WITH CHECK OPTION            | 335544603L       |
| -810    | No subqueries permitted for VIEW WITH CHECK OPTION                               | 335544605L       |
| -811    | Multiple rows in singleton select                                                | 335544652L       |
| -816    | External file could not be opened for output                                     | 335544651L       |
| -817    | Attempted update during read-only transaction                                    | 335544361L       |
| -817    | Attempted write to read-only Blob                                                | 335544371L       |
| -817    | Operation not supported                                                          | 335544444L       |
| -820    | Metadata is obsolete                                                             | 335544356L       |
| -820    | Unsupported on-disk structure for file <string>; found<br><long>, support <long> | 335544379L       |
| -820    | Wrong DYN version                                                                | 335544437L       |
| -820    | Minor version too high found <long> expected <long>                              | 335544467L       |
| -823    | Invalid bookmark handle                                                          | 335544473L       |
| -824    | Invalid lock level <digit>                                                       | 335544474L       |
| -825    | Invalid lock handle                                                              | 335544519L       |
| -826    | Invalid statement handle                                                         | 335544585L       |
| -827    | Invalid direction for find operation                                             | 335544655L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                            | InterBase number |
|---------|-------------------------------------------------------------------------|------------------|
| -828    | Invalid key position                                                    | 335544678L       |
| -829    | Invalid column reference                                                | 335544616L       |
| -830    | Column used with aggregate                                              | 335544615L       |
| -831    | Attempt to define a second PRIMARY KEY for the same table               | 335544548L       |
| -832    | FOREIGN KEY column count does not match PRIMARY KEY                     | 335544604L       |
| -833    | Expression evaluation not supported                                     | 335544606L       |
| -834    | Refresh range number <i>&lt;long&gt;</i> not found                      | 335544508L       |
| -835    | Bad checksum                                                            | 335544649L       |
| -836    | Exception <i>&lt;digit&gt;</i>                                          | 335544517L       |
| -837    | Restart shared cache manager                                            | 335544518L       |
| -838    | Database <i>&lt;string&gt;</i> shutdown in <i>&lt;digit&gt;</i> seconds | 335544560L       |
| -839    | journal file wrong format                                               | 335544686L       |
| -840    | Intermediate journal file full                                          | 335544687L       |
| -841    | Too many versions                                                       | 335544677L       |
| -842    | Precision should be greater than 0                                      | 335544697L       |
| -842    | Scale cannot be greater than precision                                  | 335544698L       |
| -842    | Short integer expected                                                  | 335544699L       |
| -842    | Long integer expected                                                   | 335544700L       |
| -842    | Unsigned short integer expected                                         | 335544701L       |
| -901    | Invalid database key                                                    | 335544322L       |
| -901    | Unrecognized database parameter block                                   | 335544326L       |
| -901    | Invalid Blob handle                                                     | 335544328L       |
| -901    | Invalid Blob ID                                                         | 335544329L       |
| -901    | Invalid parameter in transaction parameter block                        | 335544330L       |
| -901    | Invalid format for transaction parameter block                          | 335544331L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                           | InterBase number |
|---------|----------------------------------------------------------------------------------------|------------------|
| -901    | Invalid transaction handle (expecting explicit transaction start)                      | 335544332L       |
| -901    | Attempt to start more than <i>&lt;long&gt;</i> transactions                            | 335544337L       |
| -901    | Information type inappropriate for object specified                                    | 335544339L       |
| -901    | No information of this type available for object specified                             | 335544340L       |
| -901    | Unknown information item                                                               | 335544341L       |
| -901    | Action cancelled by trigger ( <i>&lt;long&gt;</i> ) to preserve data integrity         | 335544342L       |
| -901    | Lock conflict on no wait transaction                                                   | 335544345L       |
| -901    | Program attempted to exit without finishing database                                   | 335544350L       |
| -901    | Transaction is not in limbo                                                            | 335544353L       |
| -901    | Blob was not closed                                                                    | 335544355L       |
| -901    | Cannot disconnect database with open transactions ( <i>&lt;long&gt;</i> active)        | 335544357L       |
| -901    | Message length error (encountered <i>&lt;long&gt;</i> , expected <i>&lt;long&gt;</i> ) | 335544358L       |
| -901    | No transaction for request                                                             | 335544363L       |
| -901    | Request synchronization error                                                          | 335544364L       |
| -901    | Request referenced an unavailable database                                             | 335544365L       |
| -901    | Attempted read of a new, open Blob                                                     | 335544369L       |
| -901    | Attempted action on blob outside transaction                                           | 335544370L       |
| -901    | Attempted reference to Blob in unavailable database                                    | 335544372L       |
| -901    | Table <i>&lt;string&gt;</i> was omitted from the transaction reserving list            | 335544376L       |
| -901    | Request includes a DSRI extension not supported in this implementation                 | 335544377L       |
| -901    | Feature is not supported                                                               | 335544378L       |
| -901    | <i>&lt;string&gt;</i>                                                                  | 335544382L       |
| -901    | Unrecoverable conflict with limbo transaction <i>&lt;long&gt;</i>                      | 335544383L       |
| -901    | Internal error                                                                         | 335544392L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                              | InterBase number |
|---------|-----------------------------------------------------------|------------------|
| -901    | Database handle not zero                                  | 335544407L       |
| -901    | Transaction handle not zero                               | 335544408L       |
| -901    | Transaction in limbo                                      | 335544418L       |
| -901    | Transaction not in limbo                                  | 335544419L       |
| -901    | Transaction outstanding                                   | 335544420L       |
| -901    | Undefined message number                                  | 335544428L       |
| -901    | Blocking signal has been received                         | 335544431L       |
| -901    | Database system cannot read argument <i>&lt;long&gt;</i>  | 335544442L       |
| -901    | Database system cannot write argument <i>&lt;long&gt;</i> | 335544443L       |
| -901    | <i>&lt;string&gt;</i>                                     | 335544450L       |
| -901    | Transaction <i>&lt;long&gt;</i> is <i>&lt;string&gt;</i>  | 335544468L       |
| -901    | Invalid statement handle                                  | 335544485L       |
| -901    | Lock time-out on wait transaction                         | 335544510L       |
| -901    | Invalid service handle                                    | 335544559L       |
| -901    | Wrong version of service parameter block                  | 335544561L       |
| -901    | Unrecognized service parameter block                      | 335544562L       |
| -901    | Service <i>&lt;string&gt;</i> is not defined              | 335544563L       |
| -901    | INDEX <i>&lt;string&gt;</i>                               | 335544609L       |
| -901    | EXCEPTION <i>&lt;string&gt;</i>                           | 335544610L       |
| -901    | Column <i>&lt;string&gt;</i>                              | 335544611L       |
| -901    | Union not supported                                       | 335544613L       |
| -901    | Unsupported DSQL construct                                | 335544614L       |
| -901    | Illegal use of keyword VALUE                              | 335544623L       |
| -901    | Table <i>&lt;string&gt;</i>                               | 335544626L       |
| -901    | Procedure <i>&lt;string&gt;</i>                           | 335544627L       |
| -901    | Specified domain or source column does not exist          | 335544641L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                            | InterBase number |
|---------|-----------------------------------------------------------------------------------------|------------------|
| -901    | Variable <i>&lt;string&gt;</i> conflicts with parameter in same procedure               | 335544656L       |
| -901    | Server version too old to support all CREATE DATABASE options                           | 335544666L       |
| -901    | Cannot delete                                                                           | 335544673L       |
| -901    | Sort error                                                                              | 335544675L       |
| -902    | Internal isc software consistency check ( <i>&lt;string&gt;</i> )                       | 335544333L       |
| -902    | Database file appears corrupt ( <i>&lt;string&gt;</i> )                                 | 335544335L       |
| -902    | I/O error during " <i>&lt;string&gt;</i> " operation for file " <i>&lt;string&gt;</i> " | 335544344L       |
| -902    | Corrupt system table                                                                    | 335544346L       |
| -902    | Operating system directive <i>&lt;string&gt;</i> failed                                 | 335544373L       |
| -902    | Internal error                                                                          | 335544384L       |
| -902    | Internal error                                                                          | 335544385L       |
| -902    | Internal error                                                                          | 335544387L       |
| -902    | Block size exceeds implementation restriction                                           | 335544388L       |
| -902    | Incompatible version of on-disk structure                                               | 335544394L       |
| -902    | Internal error                                                                          | 335544397L       |
| -902    | Internal error                                                                          | 335544398L       |
| -902    | Internal error                                                                          | 335544399L       |
| -902    | Internal error                                                                          | 335544400L       |
| -902    | Internal error                                                                          | 335544401L       |
| -902    | Internal error                                                                          | 335544402L       |
| -902    | Database corrupted                                                                      | 335544404L       |
| -902    | Checksum error on database page <i>&lt;long&gt;</i>                                     | 335544405L       |
| -902    | Index is broken                                                                         | 335544406L       |
| -902    | Transaction--request mismatch (synchronization error)                                   | 335544409L       |
| -902    | Bad handle count                                                                        | 335544410L       |
| -902    | Wrong version of transaction parameter block                                            | 335544411L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                                                               | InterBase number |
|---------|------------------------------------------------------------------------------------------------------------|------------------|
| -902    | Unsupported BLR version (expected <i>&lt;long&gt;</i> , encountered <i>&lt;long&gt;</i> )                  | 335544412L       |
| -902    | Wrong version of database parameter block                                                                  | 335544413L       |
| -902    | Database corrupted                                                                                         | 335544415L       |
| -902    | Internal error                                                                                             | 335544416L       |
| -902    | Internal error                                                                                             | 335544417L       |
| -902    | Internal error                                                                                             | 335544422L       |
| -902    | Internal error                                                                                             | 335544423L       |
| -902    | Lock manager error                                                                                         | 335544432L       |
| -902    | SQL error code = <i>&lt;long&gt;</i>                                                                       | 335544436L       |
| -902    |                                                                                                            | 335544448L       |
| -902    |                                                                                                            | 335544449L       |
| -902    | Cache buffer for page <i>&lt;long&gt;</i> invalid                                                          | 335544470L       |
| -902    | There is no index in table <i>&lt;string&gt;</i> with id <i>&lt;digit&gt;</i>                              | 335544471L       |
| -902    | Your user name and password are not defined. Ask your database administrator to set up an InterBase login. | 335544472L       |
| -902    | Enable journal for database before starting online dump                                                    | 335544478L       |
| -902    | Online dump failure. Retry dump                                                                            | 335544479L       |
| -902    | An online dump is already in progress                                                                      | 335544480L       |
| -902    | No more disk/tape space. Cannot continue online dump                                                       | 335544481L       |
| -902    | Maximum number of online dump files that can be specified is 16                                            | 335544483L       |
| -902    | Database <i>&lt;string&gt;</i> shutdown in progress                                                        | 335544506L       |
| -902    | Long-term journaling already enabled                                                                       | 335544520L       |
| -902    | Database <i>&lt;string&gt;</i> shutdown                                                                    | 335544528L       |
| -902    | Database shutdown unsuccessful                                                                             | 335544557L       |
| -902    | Cannot attach to password database                                                                         | 335544653L       |
| -902    | Cannot start transaction for password database                                                             | 335544654L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                                  | InterBase number |
|---------|---------------------------------------------------------------|------------------|
| -902    | Long-term journaling not enabled                              | 335544564L       |
| -902    | Dynamic SQL Error                                             | 335544569L       |
| -904    | Invalid database handle (no active connection)                | 335544324L       |
| -904    | Unavailable database                                          | 335544375L       |
| -904    | Implementation limit exceeded                                 | 335544381L       |
| -904    | Too many requests                                             | 335544386L       |
| -904    | Buffer exhausted                                              | 335544389L       |
| -904    | Buffer in use                                                 | 335544391L       |
| -904    | Request in use                                                | 335544393L       |
| -904    | No lock manager available                                     | 335544424L       |
| -904    | Unable to allocate memory from operating system               | 335544430L       |
| -904    | Update conflicts with concurrent update                       | 335544451L       |
| -904    | Object <i>&lt;string&gt;</i> is in use                        | 335544453L       |
| -904    | Cannot attach active shadow file                              | 335544455L       |
| -904    | A file in manual shadow <i>&lt;long&gt;</i> is unavailable    | 335544460L       |
| -904    | Cannot add index, index root page is full.                    | 335544661L       |
| -904    | Sort error: not enough memory                                 | 335544676L       |
| -904    | Request depth exceeded. (Recursive definition?)               | 335544683L       |
| -904    | Size of optimizer block exceeded                              | 335544762L       |
| -906    | Product <i>&lt;string&gt;</i> is not licensed                 | 335544452L       |
| -909    | Drop database completed with errors                           | 335544667L       |
| -911    | Record from transaction <i>&lt;long&gt;</i> is stuck in limbo | 335544459L       |
| -913    | Deadlock                                                      | 335544336L       |
| -922    | File <i>&lt;string&gt;</i> is not a valid database            | 335544323L       |
| -923    | Connection rejected by remote interface                       | 335544421L       |
| -923    | Secondary server attachments cannot validate databases        | 335544461L       |
| -923    | Secondary server attachments cannot start journaling          | 335544462L       |

**Table 5.4** SQLCODE codes and messages (*continued*)

| SQLCODE | SQLCODE text                                | InterBase number |
|---------|---------------------------------------------|------------------|
| -924    | Bad parameters on attach or create database | 335544325L       |
| -924    | Database detach completed with errors       | 335544441L       |
| -924    | Connection lost to pipe server              | 335544648L       |
| -926    | No rollback performed                       | 335544447L       |
| -999    | InterBase error                             | 335544689L       |

## InterBase Status Array Error Codes

This section lists InterBase error codes and associated messages returned in the status array in the following tables. When code messages include the name of a database object or object type, the name is represented by a code in the Message column:

- *<string>*: String value, such as the name of a database object or object type.
- *<digit>*: Integer value, such as the identification number or code of a database object or object type.
- *<long>*: Long integer value, such as the identification number or code of a database object or object type.

The following table lists SQL Status Array codes for embedded SQL programs, DSQL, and `isql`.

**Table 5.5** InterBase status array error codes

| Error code                 | Number     | Message                                                      |
|----------------------------|------------|--------------------------------------------------------------|
| <i>isc_arith_except</i>    | 335544321L | arithmetic exception, numeric overflow, or string truncation |
| <i>isc_bad_dbkey</i>       | 335544322L | invalid database key                                         |
| <i>isc_bad_db_format</i>   | 335544323L | file <i>&lt;string&gt;</i> is not a valid database           |
| <i>isc_bad_db_handle</i>   | 335544324L | invalid database handle (no active connection)               |
| <i>isc_bad_dpb_content</i> | 335544325L | bad parameters on attach or create database                  |
| <i>isc_bad_dpb_form</i>    | 335544326L | unrecognized database parameter block                        |
| <i>isc_bad_req_handle</i>  | 335544327L | invalid request handle                                       |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                   | Number     | Message                                                           |
|------------------------------|------------|-------------------------------------------------------------------|
| <i>isc_bad_segstr_handle</i> | 335544328L | invalid Blob handle                                               |
| <i>isc_bad_segstr_id</i>     | 335544329L | invalid Blob ID                                                   |
| <i>isc_bad_tpb_content</i>   | 335544330L | invalid parameter in transaction parameter block                  |
| <i>isc_bad_tpb_form</i>      | 335544331L | invalid format for transaction parameter block                    |
| <i>isc_bad_trans_handle</i>  | 335544332L | invalid transaction handle (expecting explicit transaction start) |
| <i>isc_bug_check</i>         | 335544333L | internal isc software consistency check (<string>)                |
| <i>isc_convert_error</i>     | 335544334L | conversion error from string "<string>"                           |
| <i>isc_db_corrupt</i>        | 335544335L | database file appears corrupt (<string>)                          |
| <i>isc_deadlock</i>          | 335544336L | deadlock                                                          |
| <i>isc_excess_trans</i>      | 335544337L | attempt to start more than <long> transactions                    |
| <i>isc_from_no_match</i>     | 335544338L | no match for first value expression                               |
| <i>isc_infinap</i>           | 335544339L | information type inappropriate for object specified               |
| <i>isc_infona</i>            | 335544340L | no information of this type available for object specified        |
| <i>isc_infunk</i>            | 335544341L | unknown information item                                          |
| <i>isc_integ_fail</i>        | 335544342L | action cancelled by trigger (<long>) to preserve data integrity   |
| <i>isc_invalid_blr</i>       | 335544343L | invalid request BLR at offset <long>                              |
| <i>isc_io_error</i>          | 335544344L | I/O error during "<string>" operation for file "<string>"         |
| <i>isc_lock_conflict</i>     | 335544345L | lock conflict on no wait transaction                              |
| <i>isc_metadata_corrupt</i>  | 335544346L | corrupt system table                                              |
| <i>isc_not_valid</i>         | 335544347L | validation error for column <string>, value "<string>"            |
| <i>isc_no_cur_rec</i>        | 335544348L | no current record for fetch operation                             |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                   | Number     | Message                                                                                                     |
|------------------------------|------------|-------------------------------------------------------------------------------------------------------------|
| <i>isc_no_dup</i>            | 335544349L | attempt to store duplicate value (visible to active transactions) in unique index " <i>&lt;string&gt;</i> " |
| <i>isc_no_finish</i>         | 335544350L | program attempted to exit without finishing database                                                        |
| <i>isc_no_meta_update</i>    | 335544351L | unsuccessful metadata update                                                                                |
| <i>isc_no_priv</i>           | 335544352L | no permission for <i>&lt;string&gt;</i> access to <i>&lt;string&gt;</i> <i>&lt;string&gt;</i>               |
| <i>isc_no_recon</i>          | 335544353L | transaction is not in limbo                                                                                 |
| <i>isc_no_record</i>         | 335544354L | invalid database key                                                                                        |
| <i>isc_no_segstr_close</i>   | 335544355L | Blob was not closed                                                                                         |
| <i>isc_obsolete_metadata</i> | 335544356L | metadata is obsolete                                                                                        |
| <i>isc_open_trans</i>        | 335544357L | cannot disconnect database with open transactions ( <i>&lt;long&gt;</i> active)                             |
| <i>isc_port_len</i>          | 335544358L | message length error (encountered <i>&lt;long&gt;</i> , expected <i>&lt;long&gt;</i> )                      |
| <i>isc_read_only_field</i>   | 335544359L | attempted update of read-only column                                                                        |
| <i>isc_read_only_rel</i>     | 335544360L | attempted update of read-only table                                                                         |
| <i>isc_read_only_trans</i>   | 335544361L | attempted update during read-only transaction                                                               |
| <i>isc_read_only_view</i>    | 335544362L | cannot update read-only view <i>&lt;string&gt;</i>                                                          |
| <i>isc_req_no_trans</i>      | 335544363L | no transaction for request                                                                                  |
| <i>isc_req_sync</i>          | 335544364L | request synchronization error                                                                               |
| <i>isc_req_wrong_db</i>      | 335544365L | request referenced an unavailable database                                                                  |
| <i>isc_segment</i>           | 335544366L | segment buffer length shorter than expected                                                                 |
| <i>isc_segstr_eof</i>        | 335544367L | attempted retrieval of more segments than exist                                                             |
| <i>isc_segstr_no_op</i>      | 335544368L | attempted invalid operation on a Blob                                                                       |
| <i>isc_segstr_no_read</i>    | 335544369L | attempted read of a new, open Blob                                                                          |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                 | Number     | Message                                                                                                                |
|----------------------------|------------|------------------------------------------------------------------------------------------------------------------------|
| <i>isc_segstr_no_trans</i> | 335544370L | attempted action on Blob outside transaction                                                                           |
| <i>isc_segstr_no_write</i> | 335544371L | attempted write to read-only Blob                                                                                      |
| <i>isc_segstr_wrong_db</i> | 335544372L | attempted reference to Blob in unavailable database                                                                    |
| <i>isc_sys_request</i>     | 335544373L | operating system directive <i>&lt;string&gt;</i> failed                                                                |
| <i>isc_stream_eof</i>      | 335544374L | attempt to fetch past the last record in a record stream                                                               |
| <i>isc_unavailable</i>     | 335544375L | unavailable database                                                                                                   |
| <i>isc_unres_rel</i>       | 335544376L | Table <i>&lt;string&gt;</i> was omitted from the transaction reserving list                                            |
| <i>isc_uns_ext</i>         | 335544377L | request includes a DSRI extension not supported in this implementation                                                 |
| <i>isc_wish_list</i>       | 335544378L | feature is not supported                                                                                               |
| <i>isc_wrong_ods</i>       | 335544379L | unsupported on-disk structure for file <i>&lt;string&gt;</i> ; found <i>&lt;long&gt;</i> , support <i>&lt;long&gt;</i> |
| <i>isc_wronumarg</i>       | 335544380L | wrong number of arguments on call                                                                                      |
| <i>isc_imp_exc</i>         | 335544381L | Implementation limit exceeded                                                                                          |
| <i>isc_random</i>          | 335544382L | <i>&lt;string&gt;</i>                                                                                                  |
| <i>isc_fatal_conflict</i>  | 335544383L | unrecoverable conflict with limbo transaction <i>&lt;long&gt;</i>                                                      |
| <i>isc_badblk</i>          | 335544384L | internal error                                                                                                         |
| <i>isc_invpoolcl</i>       | 335544385L | internal error                                                                                                         |
| <i>isc_nopoolids</i>       | 335544386L | too many requests                                                                                                      |
| <i>isc_relbadblk</i>       | 335544387L | internal error                                                                                                         |
| <i>isc_blktoobig</i>       | 335544388L | block size exceeds implementation restriction                                                                          |
| <i>isc_bufexh</i>          | 335544389L | buffer exhausted                                                                                                       |
| <i>isc_syntaxerr</i>       | 335544390L | BLR syntax error: expected <i>&lt;string&gt;</i> at offset <i>&lt;long&gt;</i> , encountered <i>&lt;long&gt;</i>       |
| <i>isc_bufinuse</i>        | 335544391L | buffer in use                                                                                                          |
| <i>isc_bdbincon</i>        | 335544392L | internal error                                                                                                         |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code             | Number     | Message                                                           |
|------------------------|------------|-------------------------------------------------------------------|
| <i>isc_requise</i>     | 335544393L | request in use                                                    |
| <i>isc_badodsver</i>   | 335544394L | incompatible version of on-disk structure                         |
| <i>isc_relnotdef</i>   | 335544395L | table <string> is not defined                                     |
| <i>isc fldnotdef</i>   | 335544396L | column <string> is not defined in table <string>                  |
| <i>isc_dirtypage</i>   | 335544397L | internal error                                                    |
| <i>isc_waifortra</i>   | 335544398L | internal error                                                    |
| <i>isc_doubleloc</i>   | 335544399L | internal error                                                    |
| <i>isc_nodnotfnd</i>   | 335544400L | internal error                                                    |
| <i>isc_dupnodfnd</i>   | 335544401L | internal error                                                    |
| <i>isc_locnotmar</i>   | 335544402L | internal error                                                    |
| <i>isc_badpagtyp</i>   | 335544403L | page <long> is of wrong type (expected <long>, found <long>)      |
| <i>isc_corrupt</i>     | 335544404L | database corrupted                                                |
| <i>isc_badpage</i>     | 335544405L | checksum error on database page <long>                            |
| <i>isc_badindex</i>    | 335544406L | index is broken                                                   |
| <i>isc_dbbnotzer</i>   | 335544407L | database handle not zero                                          |
| <i>isc_tranotzer</i>   | 335544408L | transaction handle not zero                                       |
| <i>isc_trareqmis</i>   | 335544409L | transaction—request mismatch (synchronization error)              |
| <i>isc_badhndcnt</i>   | 335544410L | bad handle count                                                  |
| <i>isc_wrotpbver</i>   | 335544411L | wrong version of transaction parameter block                      |
| <i>isc_wroblrver</i>   | 335544412L | unsupported BLR version (expected <long>, encountered <long>)     |
| <i>isc_wrodpbver</i>   | 335544413L | wrong version of database parameter block                         |
| <i>isc_blobnotsup</i>  | 335544414L | Blob and array datatypes are not supported for <string> operation |
| <i>isc_badrelation</i> | 335544415L | database corrupted                                                |
| <i>isc_nodetach</i>    | 335544416L | internal error                                                    |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                 | Number     | Message                                                          |
|----------------------------|------------|------------------------------------------------------------------|
| <i>isc_notremote</i>       | 335544417L | internal error                                                   |
| <i>isc_trainlim</i>        | 335544418L | transaction in limbo                                             |
| <i>isc_notinlim</i>        | 335544419L | transaction not in limbo                                         |
| <i>isc_traoutsta</i>       | 335544420L | transaction outstanding                                          |
| <i>isc_connect_reject</i>  | 335544421L | connection rejected by remote interface                          |
| <i>isc_dbfile</i>          | 335544422L | internal error                                                   |
| <i>isc_orphan</i>          | 335544423L | internal error                                                   |
| <i>isc_no_lock_mgr</i>     | 335544424L | no lock manager available                                        |
| <i>isc_ctxinuse</i>        | 335544425L | context already in use (BLR error)                               |
| <i>isc_ctxnotdef</i>       | 335544426L | context not defined (BLR error)                                  |
| <i>isc_datnotsup</i>       | 335544427L | data operation not supported                                     |
| <i>isc_badmsgnum</i>       | 335544428L | undefined message number                                         |
| <i>isc_badparnum</i>       | 335544429L | bad parameter number                                             |
| <i>isc_virmemexh</i>       | 335544430L | unable to allocate memory from operating system                  |
| <i>isc_blocking_signal</i> | 335544431L | blocking signal has been received                                |
| <i>isc_lockmanerr</i>      | 335544432L | lock manager error                                               |
| <i>isc_journerr</i>        | 335544433L | communication error with journal "<string>"                      |
| <i>isc_keytoobig</i>       | 335544434L | key size exceeds implementation restriction for index "<string>" |
| <i>isc_nullsegkey</i>      | 335544435L | null segment of UNIQUE KEY                                       |
| <i>isc_sqlerr</i>          | 335544436L | SQL error code = <long>                                          |
| <i>isc_wrodynver</i>       | 335544437L | wrong DYN version                                                |
| <i>isc_funnotdef</i>       | 335544438L | function <string> is not defined                                 |
| <i>isc_funmismat</i>       | 335544439L | function <string> could not be matched                           |
| <i>isc_bad_msg_vec</i>     | 335544440L |                                                                  |
| <i>isc_bad_detach</i>      | 335544441L | database detach completed with errors                            |
| <i>isc_noargacc_read</i>   | 335544442L | database system cannot read argument <long>                      |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                    | Number     | Message                                                                      |
|-------------------------------|------------|------------------------------------------------------------------------------|
| <i>isc_noargacc_write</i>     | 335544443L | database system cannot write argument <long>                                 |
| <i>isc_read_only</i>          | 335544444L | operation not supported                                                      |
| <i>isc_ext_err</i>            | 335544445L | <string> extension error                                                     |
| <i>isc_non_updatable</i>      | 335544446L | not updatable                                                                |
| <i>isc_no_rollback</i>        | 335544447L | no rollback performed                                                        |
| <i>isc_bad_sec_info</i>       | 335544448L |                                                                              |
| <i>isc_invalid_sec_info</i>   | 335544449L |                                                                              |
| <i>isc_misc_interpreted</i>   | 335544450L | <string>                                                                     |
| <i>isc_update_conflict</i>    | 335544451L | update conflicts with concurrent update                                      |
| <i>isc_unlicensed</i>         | 335544452L | product <string> is not licensed                                             |
| <i>isc_obj_in_use</i>         | 335544453L | object <string> is in use                                                    |
| <i>isc_nofilter</i>           | 335544454L | filter not found to convert type <long> to type <long>                       |
| <i>isc_shadow_accessed</i>    | 335544455L | cannot attach active shadow file                                             |
| <i>isc_invalid_sdl</i>        | 335544456L | invalid slice description language at offset <long>                          |
| <i>isc_out_of_bounds</i>      | 335544457L | subscript out of bounds                                                      |
| <i>isc_invalid_dimension</i>  | 335544458L | column not array or invalid dimensions (expected <long>, encountered <long>) |
| <i>isc_rec_in_limbo</i>       | 335544459L | record from transaction <long> is stuck in limbo                             |
| <i>isc_shadow_missing</i>     | 335544460L | a file in manual shadow <long> is unavailable                                |
| <i>isc_cant_validate</i>      | 335544461L | secondary server attachments cannot validate databases                       |
| <i>isc_cant_start_journal</i> | 335544462L | secondary server attachments cannot start journaling                         |
| <i>isc_gennotdef</i>          | 335544463L | generator <string> is not defined                                            |
| <i>isc_cant_start_logging</i> | 335544464L | secondary server attachments cannot start logging                            |
| <i>isc_bad_segstr_type</i>    | 335544465L | invalid Blob type for operation                                              |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                    | Number     | Message                                                                                                    |
|-------------------------------|------------|------------------------------------------------------------------------------------------------------------|
| <i>isc_foreign_key</i>        | 335544466L | violation of FOREIGN KEY constraint: " <i>&lt;string&gt;</i> "                                             |
| <i>isc_high_minor</i>         | 335544467L | minor version too high found <i>&lt;long&gt;</i> expected <i>&lt;long&gt;</i>                              |
| <i>isc_tra_state</i>          | 335544468L | transaction <i>&lt;long&gt;</i> is <i>&lt;string&gt;</i>                                                   |
| <i>isc_trans_invalid</i>      | 335544469L | transaction marked invalid by I/O error                                                                    |
| <i>isc_buf_invalid</i>        | 335544470L | cache buffer for page <i>&lt;long&gt;</i> invalid                                                          |
| <i>isc_indexnotdefined</i>    | 335544471L | there is no index in table <i>&lt;string&gt;</i> with id <i>&lt;digit&gt;</i>                              |
| <i>isc_login</i>              | 335544472L | Your user name and password are not defined. Ask your database administrator to set up an InterBase login. |
| <i>isc_invalid_bookmark</i>   | 335544473L | invalid bookmark handle                                                                                    |
| <i>isc_bad_lock_level</i>     | 335544474L | invalid lock level <i>&lt;digit&gt;</i>                                                                    |
| <i>isc_relation_lock</i>      | 335544475L | lock on table <i>&lt;string&gt;</i> conflicts with existing lock                                           |
| <i>isc_record_lock</i>        | 335544476L | requested record lock conflicts with existing lock                                                         |
| <i>isc_max_idx</i>            | 335544477L | maximum indexes per table ( <i>&lt;digit&gt;</i> ) exceeded                                                |
| <i>isc_jrn_enable</i>         | 335544478L | enable journal for database before starting online dump                                                    |
| <i>isc_old_failure</i>        | 335544479L | online dump failure. Retry dump                                                                            |
| <i>isc_old_in_progress</i>    | 335544480L | an online dump is already in progress                                                                      |
| <i>isc_old_no_space</i>       | 335544481L | no more disk/tape space. Cannot continue online dump                                                       |
| <i>isc_num_old_files</i>      | 335544483L | maximum number of online dump files that can be specified is 16                                            |
| <i>isc_bad_stmt_handle</i>    | 335544485L | invalid statement handle                                                                                   |
| <i>isc_stream_not_defined</i> | 335544502L | reference to invalid stream number                                                                         |
| <i>isc_shutinprog</i>         | 335544506L | database <i>&lt;string&gt;</i> shutdown in progress                                                        |
| <i>isc_range_in_use</i>       | 335544507L | refresh range number <i>&lt;long&gt;</i> already in use                                                    |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                      | Number     | Message                                                          |
|---------------------------------|------------|------------------------------------------------------------------|
| <i>isc_range_not_found</i>      | 335544508L | refresh range number <long> not found                            |
| <i>isc_charset_not_found</i>    | 335544509L | character set <string> is not defined                            |
| <i>isc_lock_timeout</i>         | 335544510L | lock time-out on wait transaction                                |
| <i>isc_prcnotdef</i>            | 335544511L | procedure <string> is not defined                                |
| <i>isc_prcmismat</i>            | 335544512L | parameter mismatch for procedure <string>                        |
| <i>isc_codnotdef</i>            | 335544515L | status code <string> unknown                                     |
| <i>isc_xcpnotdef</i>            | 335544516L | exception <string> not defined                                   |
| <i>isc_except</i>               | 335544517L | exception <digit>                                                |
| <i>isc_cache_restart</i>        | 335544518L | restart shared cache manager                                     |
| <i>isc_bad_lock_handle</i>      | 335544519L | invalid lock handle                                              |
| <i>isc_shutdown</i>             | 335544528L | database <string> shutdown                                       |
| <i>isc_existing_priv_mod</i>    | 335544529L | cannot modify an existing user privilege                         |
| <i>isc_primary_key_ref</i>      | 335544530L | Cannot delete PRIMARY KEY being used in FOREIGN KEY definition.  |
| <i>isc_primary_key_notnull</i>  | 335544531L | Column used in a PRIMARY/UNIQUE constraint must be NOT NULL.     |
| <i>isc_ref_cnstrnt_notfound</i> | 335544532L | Name of Referential Constraint not defined in constraints table. |
| <i>isc_foreign_key_notfound</i> | 335544533L | Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY.    |
| <i>isc_ref_cnstrnt_update</i>   | 335544534L | Cannot update constraints (RDB\$REF_CONSTRAINTS).                |
| <i>isc_check_cnstrnt_update</i> | 335544535L | Cannot update constraints (RDB\$CHECK_CONSTRAINTS).              |
| <i>isc_check_cnstrnt_del</i>    | 335544536L | Cannot delete CHECK constraint entry (RDB\$CHECK_CONSTRAINTS)    |
| <i>isc_integ_index_seg_del</i>  | 335544537L | Cannot delete index segment used by an Integrity Constraint      |
| <i>isc_integ_index_seg_mod</i>  | 335544538L | Cannot update index segment used by an Integrity Constraint      |
| <i>isc_integ_index_del</i>      | 335544539L | Cannot delete index used by an Integrity Constraint              |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                     | Number     | Message                                                                |
|--------------------------------|------------|------------------------------------------------------------------------|
| <i>isc_integ_index_mod</i>     | 335544540L | Cannot modify index used by an Integrity Constraint                    |
| <i>isc_check_trig_del</i>      | 335544541L | Cannot delete trigger used by a CHECK Constraint                       |
| <i>isc_check_trig_update</i>   | 335544542L | Cannot update trigger used by a CHECK Constraint                       |
| <i>isc_cnstrnt_fld_del</i>     | 335544543L | Cannot delete column being used in an Integrity Constraint.            |
| <i>isc_cnstrnt_fld_rename</i>  | 335544544L | Cannot rename column being used in an Integrity Constraint.            |
| <i>isc_rel_cnstrnt_update</i>  | 335544545L | Cannot update constraints (RDB\$RELATION_CONSTRAINTS).                 |
| <i>isc_constraint_on_view</i>  | 335544546L | Cannot define constraints on views                                     |
| <i>isc_invl_d_cnstrnt_type</i> | 335544547L | internal isc software consistency check (invalid RDB\$CONSTRAINT_TYPE) |
| <i>isc_primary_key_exists</i>  | 335544548L | Attempt to define a second PRIMARY KEY for the same table              |
| <i>isc_systrig_update</i>      | 335544549L | cannot modify or erase a system trigger                                |
| <i>isc_not_rel_owner</i>       | 335544550L | only the owner of a table may reassign ownership                       |
| <i>isc_grant_obj_notfound</i>  | 335544551L | could not find table/procedure for GRANT                               |
| <i>isc_grant_fld_notfound</i>  | 335544552L | could not find column for GRANT                                        |
| <i>isc_grant_nopriv</i>        | 335544553L | user does not have GRANT privileges for operation                      |
| <i>isc_nonsql_security_rel</i> | 335544554L | table/procedure has non-SQL security class defined                     |
| <i>isc_nonsql_security_fld</i> | 335544555L | column has non-SQL security class defined                              |
| <i>isc_shutdown</i>            | 335544557L | database shutdown unsuccessful                                         |
| <i>isc_check_constraint</i>    | 335544558L | Operation violates CHECK constraint <string> on view or table          |
| <i>isc_bad_svc_handle</i>      | 335544559L | invalid service handle                                                 |
| <i>isc_shutdown</i>            | 335544560L | database <string> shutdown in <digit> seconds                          |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                         | Number     | Message                                                                   |
|------------------------------------|------------|---------------------------------------------------------------------------|
| <i>isc_wrospbver</i>               | 335544561L | wrong version of service parameter block                                  |
| <i>isc_bad_spb_form</i>            | 335544562L | unrecognized service parameter block                                      |
| <i>isc_svcnotdef</i>               | 335544563L | service <string> is not defined                                           |
| <i>isc_no_jrn</i>                  | 335544564L | long-term journaling not enabled                                          |
| <i>isc_transliteration_failed</i>  | 335544565L | Cannot transliterate character between character sets                     |
| <i>isc_text_subtype</i>            | 335544568L | Implementation of text subtype <digit> not located.                       |
| <i>isc_dsqli_error</i>             | 335544569L | Dynamic SQL Error                                                         |
| <i>isc_dsqli_command_err</i>       | 335544570L | Invalid command                                                           |
| <i>isc_dsqli_constant_err</i>      | 335544571L | Datatype for constant unknown                                             |
| <i>isc_dsqli_cursor_err</i>        | 335544572L | Cursor unknown                                                            |
| <i>isc_dsqli_datatype_err</i>      | 335544573L | Datatype unknown                                                          |
| <i>isc_dsqli_decl_err</i>          | 335544574L | Declared cursor already exists                                            |
| <i>isc_dsqli_cursor_update_err</i> | 335544575L | Cursor not updatable                                                      |
| <i>isc_dsqli_cursor_open_err</i>   | 335544576L | Attempt to reopen an open cursor                                          |
| <i>isc_dsqli_cursor_close_err</i>  | 335544577L | Attempt to reclose a closed cursor                                        |
| <i>isc_dsqli_field_err</i>         | 335544578L | Column unknown                                                            |
| <i>isc_dsqli_internal_err</i>      | 335544579L | Internal error                                                            |
| <i>isc_dsqli_relation_err</i>      | 335544580L | Table unknown                                                             |
| <i>isc_dsqli_procedure_err</i>     | 335544581L | Procedure unknown                                                         |
| <i>isc_dsqli_request_err</i>       | 335544582L | Request unknown                                                           |
| <i>isc_dsqli_sqlda_err</i>         | 335544583L | SQLDA missing or incorrect version, or incorrect number/type of variables |
| <i>isc_dsqli_var_count_err</i>     | 335544584L | Count of columns not equal count of values                                |
| <i>isc_dsqli_stmt_handle</i>       | 335544585L | Invalid statement handle                                                  |
| <i>isc_dsqli_function_err</i>      | 335544586L | Function unknown                                                          |
| <i>isc_dsqli_blob_err</i>          | 335544587L | Column is not a Blob                                                      |
| <i>isc_collation_not_found</i>     | 335544588L | COLLATION <string> is not defined                                         |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                           | Number     | Message                                                            |
|--------------------------------------|------------|--------------------------------------------------------------------|
| <i>isc_collation_not_for_charset</i> | 335544589L | COLLATION <string> is not valid for specified CHARACTER SET        |
| <i>isc_dsql_dup_option</i>           | 335544590L | Option specified more than once                                    |
| <i>isc_dsql_tran_err</i>             | 335544591L | Unknown transaction option                                         |
| <i>isc_dsql_invalid_array</i>        | 335544592L | Invalid array reference                                            |
| <i>isc_dsql_max_arr_dim_exceeded</i> | 335544593L | Array declared with too many dimensions                            |
| <i>isc_dsql_arr_range_error</i>      | 335544594L | Illegal array dimension range                                      |
| <i>isc_dsql_trigger_err</i>          | 335544595L | Trigger unknown                                                    |
| <i>isc_dsql_subselect_err</i>        | 335544596L | Subselect illegal in this context                                  |
| <i>isc_dsql_crdb_prepare_err</i>     | 335544597L | Cannot prepare a CREATE DATABASE/SCHEMA statement                  |
| <i>isc_specify_field_err</i>         | 335544598L | must specify column name for view select expression                |
| <i>isc_num_field_err</i>             | 335544599L | number of columns does not match select list                       |
| <i>isc_col_name_err</i>              | 335544600L | Only simple column names permitted for VIEW WITH CHECK OPTION      |
| <i>isc_where_err</i>                 | 335544601L | No WHERE clause for VIEW WITH CHECK OPTION                         |
| <i>isc_table_view_err</i>            | 335544602L | Only one table allowed for VIEW WITH CHECK OPTION                  |
| <i>isc_distinct_err</i>              | 335544603L | DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION |
| <i>isc_key_field_count_err</i>       | 335544604L | FOREIGN KEY column count does not match PRIMARY KEY                |
| <i>isc_subquery_err</i>              | 335544605L | No subqueries permitted for VIEW WITH CHECK OPTION                 |
| <i>isc_expression_eval_err</i>       | 335544606L | expression evaluation not supported                                |
| <i>isc_node_err</i>                  | 335544607L | gen.c: node not supported                                          |
| <i>isc_command_end_err</i>           | 335544608L | Unexpected end of command                                          |
| <i>isc_index_name</i>                | 335544609L | INDEX <string>                                                     |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                          | Number     | Message                                                                              |
|-------------------------------------|------------|--------------------------------------------------------------------------------------|
| <i>isc_exception_name</i>           | 335544610L | EXCEPTION <string>                                                                   |
| <i>isc_field_name</i>               | 335544611L | COLUMN <string>                                                                      |
| <i>isc_token_err</i>                | 335544612L | Token unknown                                                                        |
| <i>isc_union_err</i>                | 335544613L | union not supported                                                                  |
| <i>isc_dsql_construct_err</i>       | 335544614L | Unsupported DSQL construct                                                           |
| <i>isc_field_aggregate_err</i>      | 335544615L | column used with aggregate                                                           |
| <i>isc_field_ref_err</i>            | 335544616L | invalid column reference                                                             |
| <i>isc_order_by_err</i>             | 335544617L | invalid ORDER BY clause                                                              |
| <i>isc_return_mode_err</i>          | 335544618L | Return mode by value not allowed for this datatype                                   |
| <i>isc_extern_func_err</i>          | 335544619L | External functions cannot have more than 10 parameters                               |
| <i>isc_alias_conflict_err</i>       | 335544620L | alias <string> conflicts with an alias in the same statement                         |
| <i>isc_procedure_conflict_error</i> | 335544621L | alias <string> conflicts with a procedure in the same statement                      |
| <i>isc_relation_conflict_err</i>    | 335544622L | alias <string> conflicts with a table in the same statement                          |
| <i>isc_dsql_domain_err</i>          | 335544623L | Illegal use of keyword VALUE                                                         |
| <i>isc_idx_seg_err</i>              | 335544624L | segment count of 0 defined for index <string>                                        |
| <i>isc_node_name_err</i>            | 335544625L | A node name is not permitted in a secondary, shadow, cache or log file name          |
| <i>isc_table_name</i>               | 335544626L | TABLE <string>                                                                       |
| <i>isc_proc_name</i>                | 335544627L | PROCEDURE <string>                                                                   |
| <i>isc_idx_create_err</i>           | 335544628L | cannot create index <string>                                                         |
| <i>isc_dependency</i>               | 335544630L | there are <long> dependencies                                                        |
| <i>isc_idx_key_err</i>              | 335544631L | too many keys defined for index <string>                                             |
| <i>isc_dsql_file_length_err</i>     | 335544632L | Preceding file did not specify length, so <string> must include starting page number |
| <i>isc_dsql_shadow_number_err</i>   | 335544633L | Shadow number must be a positive integer                                             |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                         | Number     | Message                                                                         |
|------------------------------------|------------|---------------------------------------------------------------------------------|
| <i>isc_dsql_token_unk_err</i>      | 335544634L | Token unknown - line <long>, char <long>                                        |
| <i>isc_dsql_no_relation_alias</i>  | 335544635L | there is no alias or table named <string> at this scope level                   |
| <i>isc_indexname</i>               | 335544636L | there is no index <string> for table <string>                                   |
| <i>isc_no_stream_plan</i>          | 335544637L | table <string> is not referenced in plan                                        |
| <i>isc_stream_twice</i>            | 335544638L | table <string> is referenced more than once in plan; use aliases to distinguish |
| <i>isc_stream_not_found</i>        | 335544639L | table <string> is referenced in the plan but not the from list                  |
| <i>isc_collation_requires_text</i> | 335544640L | Invalid use of CHARACTER SET or COLLATE                                         |
| <i>isc_dsql_domain_not_found</i>   | 335544641L | Specified domain or source column does not exist                                |
| <i>isc_index_unused</i>            | 335544642L | index <string> cannot be used in the specified plan                             |
| <i>isc_dsql_self_join</i>          | 335544643L | the table <string> is referenced twice; use aliases to differentiate            |
| <i>isc_stream_bof</i>              | 335544644L | illegal operation when at beginning of stream                                   |
| <i>isc_stream_crack</i>            | 335544645L | the current position is on a crack                                              |
| <i>isc_db_or_file_exists</i>       | 335544646L | database or file exists                                                         |
| <i>isc_invalid_operator</i>        | 335544647L | invalid comparison operator for find operation                                  |
| <i>isc_conn_lost</i>               | 335544648L | Connection lost to pipe server                                                  |
| <i>isc_bad_checksum</i>            | 335544649L | bad checksum                                                                    |
| <i>isc_page_type_err</i>           | 335544650L | wrong page type                                                                 |
| <i>isc_ext_readonly_err</i>        | 335544651L | external file could not be opened for output                                    |
| <i>isc_sing_select_err</i>         | 335544652L | multiple rows in singleton select                                               |
| <i>isc_psw_attach</i>              | 335544653L | cannot attach to password database                                              |
| <i>isc_psw_start_trans</i>         | 335544654L | cannot start transaction for password database                                  |
| <i>isc_invalid_direction</i>       | 335544655L | invalid direction for find operation                                            |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                            | Number     | Message                                                                 |
|---------------------------------------|------------|-------------------------------------------------------------------------|
| <i>isc_dsql_var_conflict</i>          | 335544656L | variable <string> conflicts with parameter in same procedure            |
| <i>isc_dsql_no_blob_array</i>         | 335544657L | Array/Blob/DATE /TIME/TIMESTAMP datatypes not allowed in arithmetic     |
| <i>isc_dsql_base_table</i>            | 335544658L | <string> is not a valid base table of the specified view                |
| <i>isc_duplicate_base_table</i>       | 335544659L | table <string> is referenced twice in view; use an alias to distinguish |
| <i>isc_view_alias</i>                 | 335544660L | view <string> has more than one base table; use aliases to distinguish  |
| <i>isc_index_root_page_full</i>       | 335544661L | cannot add index, index root page is full.                              |
| <i>isc_dsql_blob_type_unknown</i>     | 335544662L | BLOB SUB_TYPE <string> is not defined                                   |
| <i>isc_req_max_clones_exceeded</i>    | 335544663L | Too many concurrent executions of the same request                      |
| <i>isc_dsql_duplicate_spec</i>        | 335544664L | duplicate specification of <string> - not supported                     |
| <i>isc_unique_key_violation</i>       | 335544665L | violation of PRIMARY or UNIQUE KEY constraint: "<string>"               |
| <i>isc_srvr_version_too_old</i>       | 335544666L | server version too old to support all CREATE DATABASE options           |
| <i>isc_drdb_completed_with_errors</i> | 335544667L | drop database completed with errors                                     |
| <i>isc_dsql_procedure_use_err</i>     | 335544668L | procedure <string> does not return any values                           |
| <i>isc_dsql_count_mismatch</i>        | 335544669L | count of column list and variable list do not match                     |
| <i>isc_blob_idx_err</i>               | 335544670L | attempt to index Blob column in index <string>                          |
| <i>isc_array_idx_err</i>              | 335544671L | attempt to index array column in index <string>                         |
| <i>isc_key_field_err</i>              | 335544672L | too few key columns found for index <string> (incorrect column name?)   |
| <i>isc_no_delete</i>                  | 335544673L | cannot delete                                                           |
| <i>isc_del_last_field</i>             | 335544674L | last column in a table cannot be deleted                                |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                               | Number     | Message                                                                  |
|------------------------------------------|------------|--------------------------------------------------------------------------|
| <i>isc_sort_err</i>                      | 335544675L | sort error                                                               |
| <i>isc_sort_mem_err</i>                  | 335544676L | sort error: not enough memory                                            |
| <i>isc_version_err</i>                   | 335544677L | too many versions                                                        |
| <i>isc_inval_key_posn</i>                | 335544678L | invalid key position                                                     |
| <i>isc_no_segments_err</i>               | 335544679L | segments not allowed in expression index<br><string>                     |
| <i>isc_crrp_data_err</i>                 | 335544680L | sort error: corruption in data structure                                 |
| <i>isc_rec_size_err</i>                  | 335544681L | new record size of <long> bytes is too big                               |
| <i>isc_dsql_field_ref</i>                | 335544682L | Inappropriate self-reference of column                                   |
| <i>isc_req_depth_exceeded</i>            | 335544683L | request depth exceeded. (Recursive definition?)                          |
| <i>isc_no_field_access</i>               | 335544684L | cannot access column <string> in view<br><string>                        |
| <i>isc_no_dbkey</i>                      | 335544685L | dbkey not available for multi-table views                                |
| <i>isc_dsql_open_cursor_reque<br/>st</i> | 335544688L | The prepare statement identifies a prepare statement with an open cursor |
| <i>isc_ib_error</i>                      | 335544689L | InterBase error                                                          |
| <i>isc_cache_redef</i>                   | 335544690L | Cache redefined                                                          |
| <i>isc_cache_too_small</i>               | 335544691L | Cache length too small                                                   |
| <i>isc_precision_err</i>                 | 335544697L | Precision should be greater than 0                                       |
| <i>isc_scale_nogt</i>                    | 335544698L | Scale cannot be greater than precision                                   |
| <i>isc_expec_short</i>                   | 335544699L | Short integer expected                                                   |
| <i>isc_expec_long</i>                    | 335544700L | Long integer expected                                                    |
| <i>isc_expec_ushort</i>                  | 335544701L | Unsigned short integer expected                                          |
| <i>isc_like_escape_invalid</i>           | 335544702L | Invalid ESCAPE sequence                                                  |
| <i>isc_svcnoexe</i>                      | 335544703L | service <string> does not have an associated executable                  |
| <i>isc_net_lookup_err</i>                | 335544704L | Network lookup failure for host "<string>"                               |
| <i>isc_service_unknown</i>               | 335544705L | Undefined service <string>/<string>                                      |
| <i>isc_host_unknown</i>                  | 335544706L | Host unknown                                                             |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                        | Number     | Message                                                                     |
|-----------------------------------|------------|-----------------------------------------------------------------------------|
| <i>isc_grant_nopriv_on_base</i>   | 335544707L | user does not have GRANT privileges on base table/view for operation        |
| <i>isc_dyn_fld_ambiguous</i>      | 335544708L | Ambiguous column reference.                                                 |
| <i>isc_dsqli_agg_ref_err</i>      | 335544709L | Invalid aggregate reference                                                 |
| <i>isc_complex_view</i>           | 335544710L | navigational stream <long> references a view with more than one base table. |
| <i>isc_unprepared_stmt</i>        | 335544711L | attempt to execute an unprepared dynamic SQL statement                      |
| <i>isc_expec_positive</i>         | 335544712L | Positive value expected.                                                    |
| <i>isc_dsqli_sqlda_value_err</i>  | 335544713L | Incorrect values within SQLDA structure                                     |
| <i>isc_invalid_array_id</i>       | 335544714L | invalid Blob id                                                             |
| <i>isc_ext_file_uns_op</i>        | 335544715L | operation not supported for EXTERNAL FILE table <string>                    |
| <i>isc_svc_in_use</i>             | 335544716L | service is currently busy: <string>                                         |
| <i>isc_err_stack_limit</i>        | 335544717L | stack size insufficient to execute current request                          |
| <i>isc_invalid_key</i>            | 335544718L | invalid key for find operation                                              |
| <i>isc_net_init_error</i>         | 335544719L | error initializing the network software                                     |
| <i>isc_loadlib_failure</i>        | 335544720L | unable to load required library <string>                                    |
| <i>isc_network_error</i>          | 335544721L | unable to complete network request to host "<string>"                       |
| <i>isc_net_connect_err</i>        | 335544722L | failed to establish a connection                                            |
| <i>isc_net_connect_listen_err</i> | 335544723L | error while listening for an incoming connection                            |
| <i>isc_net_event_connect_err</i>  | 335544724L | failed to establish a secondary connection for event processing             |
| <i>isc_net_event_listen_err</i>   | 335544725L | error while listening for an incoming event connection request              |
| <i>isc_net_read_err</i>           | 335544726L | error reading data from the connection                                      |
| <i>isc_net_write_err</i>          | 335544727L | error writing data to the connection                                        |
| <i>isc_integ_index_deactivate</i> | 335544728L | cannot deactivate index used by an Integrity Constraint                     |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                           | Number     | Message                                                                                                                     |
|--------------------------------------|------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>isc_integ_deactivate_primary</i>  | 335544729L | cannot deactivate primary index                                                                                             |
| <i>isc_unsupported_network_drive</i> | 335544732L | access to databases on file servers is not supported                                                                        |
| <i>isc_io_create_err</i>             | 335544733L | error while trying to create file                                                                                           |
| <i>isc_io_open_err</i>               | 335544734L | error while trying to open file                                                                                             |
| <i>isc_io_close_err</i>              | 335544735L | error while trying to close file                                                                                            |
| <i>isc_io_read_err</i>               | 335544736L | error while trying to read from file                                                                                        |
| <i>isc_io_write_err</i>              | 335544737L | error while trying to write to file                                                                                         |
| <i>isc_io_delete_err</i>             | 335544738L | error while trying to delete file                                                                                           |
| <i>isc_io_access_err</i>             | 335544739L | error while trying to access file                                                                                           |
| <i>isc_udf_exception</i>             | 335544740L | exception <integer> detected in blob filter or user defined function                                                        |
| <i>isc_lost_db_connection</i>        | 335544741L | connection lost to database                                                                                                 |
| <i>isc_no_write_user_priv</i>        | 335544742L | user cannot write to RDB\$USER_PRIVILEGES                                                                                   |
| <i>isc_token_too_long</i>            | 335544743L | token size exceeds limit                                                                                                    |
| <i>isc_max_att_exceeded</i>          | 335544744L | maximum user count exceeded; contact your database administrator                                                            |
| <i>isc_login_same_as_role_name</i>   | 335544745L | your login <string> is same as one of the SQL role names; ask your database administrator to set up a valid InterBase login |
| <i>isc_reftable_requires_pk</i>      | 335544746L | "REFERENCES table" without "(column)"; requires PRIMARY KEY on referenced table                                             |
| <i>isc_username_too_long</i>         | 335544747L | the username entered is too long. Maximum length is 31 bytes.                                                               |
| <i>isc_password_too_long</i>         | 335544748L | the password specified is too long. Maximum length is 8 bytes.                                                              |
| <i>isc_username_required</i>         | 335544749L | a username is required for this operation.                                                                                  |
| <i>isc_password_required</i>         | 335544750L | a password is required for this operation                                                                                   |
| <i>isc_bad_protocol</i>              | 335544751L | the network protocol specified is invalid                                                                                   |

**Table 5.5** InterBase status array error codes (*continued*)

| Error code                            | Number     | Message                                                                                                  |
|---------------------------------------|------------|----------------------------------------------------------------------------------------------------------|
| <i>isc_dup_username_found</i>         | 335544752L | a duplicate user name was found in the security database                                                 |
| <i>isc_username_not_found</i>         | 335544753L | the user name specified was not found in the security database                                           |
| <i>isc_error_adding_sec_record</i>    | 335544754L | error while attempting to add the user                                                                   |
| <i>isc_error_modifying_sec_record</i> | 335544755L | error while attempting to modify the user record                                                         |
| <i>isc_error_deleting_sec_record</i>  | 335544756L | error while attempting to delete the user record                                                         |
| <i>isc_error_updating_sec_db</i>      | 335544757L | error while updating the security database                                                               |
| <i>isc_sort_rec_size_err</i>          | 335544758L | sort record size is too big                                                                              |
| <i>isc_bad_default_value</i>          | 335544759L | cannot assign a NULL default value to a column with a NOT NULL constraint                                |
| <i>isc_invalid_clause</i>             | 335544760L | the specified user-entered string is not valid                                                           |
| <i>isc_too_many_handles</i>           | 335544761L | too many open handles to database                                                                        |
| <i>isc_optimizer_blk_exc</i>          | 335544762L | optimizer implementation limits are exceeded; for example, only 256 conjuncts (ANDs and ORs) are allowed |

# System Tables, Temporary Tables, and Views

This chapter describes the InterBase system tables and SQL system views.

**Important** Only InterBase system object names can begin with the characters “RDB\$” or “TMP\$”. No other object name in InterBase can begin with these character sequences, including tables, views, triggers, stored procedures, indexes, generators, domains, and roles.

## Overview

---

The InterBase system tables contain and track metadata. InterBase automatically creates system tables when a database is created. Each time a user creates or modifies metadata through data definition, the SQL data definition utility automatically updates the system tables.

The temporary system tables allow access to information about the database and its connections and a degree of control over transactions. By default, all users can select from permanent system tables, but only the database owner and the SYSDBA user can write to them. To gain access to temporary system tables, explicit access has to be granted to them by the database owner or the SYSDBA. These users can grant write access to others if they wish. See the [Operations Guide](#) for details about system table security.

SQL system views provide information about existing integrity constraints for a database. You must create system views yourself by creating and running an `isql` script after database definition. See [“System Views” on page 6-51](#) for the code that creates them as well as the resulting table structures.

To see system tables, use this **isql** command:

```
SHOW SYSTEM TABLES;
```

The following **isql** command lists system views along with database views:

```
SHOW VIEWS;
```

# System Tables

---

This table lists the InterBase system tables. The names of system tables and their columns start with RDB\$.

**Table 6.1** System tables

|                         |                           |
|-------------------------|---------------------------|
| RDB\$CHARACTER_SETS     | RDB\$LOG_FILES            |
| RDB\$CHECK_CONSTRAINTS  | RDB\$PAGES                |
| RDB\$COLLATIONS         | RDB\$PROCEDURE_PARAMETERS |
| RDB\$DATABASE           | RDB\$PROCEDURES           |
| RDB\$DEPENDENCIES       | RDB\$REF_CONSTRAINTS      |
| RDB\$EXCEPTIONS         | RDB\$RELATION_CONSTRAINTS |
| RDB\$FIELD_DIMENSIONS   | RDB\$RELATION_FIELDS      |
| RDB\$FIELDS             | RDB\$RELATIONS            |
| RDB\$FILES              | RDB\$ROLES                |
| RDB\$FILTERS            | RDB\$SECURITY_CLASSES     |
| RDB\$FORMATS            | RDB\$TRANSACTIONS         |
| RDB\$FUNCTION_ARGUMENTS | RDB\$TRIGGER_MESSAGES     |
| RDB\$FUNCTIONS          | RDB\$TRIGGERS             |
| RDB\$GENERATORS         | RDB\$TYPES                |
| RDB\$INDEX_SEGMENTS     | RDB\$USER_PRIVILEGES      |
| RDB\$INDICES            | RDB\$USERS                |
|                         | RDB\$VIEW_RELATIONS       |

## RDB\$CHARACTER\_SETS

RDB\$CHARACTER\_SETS describes the valid character sets available in InterBase.

**Table 6.2** RDB\$CHARACTER\_SETS

| Column name               | Datatype | Length | Description                                                                                                                                                          |
|---------------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$CHARACTER_SET_NAME   | CHAR     | 67     | Name of a character set that InterBase recognizes                                                                                                                    |
| RDB\$FORM_OF_USE          | CHAR     | 67     | Reserved for internal use. Subtype 2                                                                                                                                 |
| RDB\$NUMBER_OF_CHARACTERS | INTEGER  |        | Number of characters in a particular character set; for example, the set of Japanese characters                                                                      |
| RDB\$DEFAULT_COLLATE_NAME | CHAR     | 67     | Subtype 2: default collation sequence for the character set                                                                                                          |
| RDB\$CHARACTER_SET_ID     | SMALLINT |        | A unique identification for the character set                                                                                                                        |
| RDB\$SYSTEM_FLAG          | SMALLINT |        | Indicates whether the character set is: <ul style="list-style-type: none"> <li>• User-defined (value of 0 or NULL)</li> <li>• System-defined (value of 1)</li> </ul> |
| RDB\$DESCRIPTION          | BLOB     |        | Subtype text: Contains a user-written description of the character set                                                                                               |
| RDB\$FUNCTION_NAME        | CHAR     | 67     | Reserved for internal use; subtype 2                                                                                                                                 |
| RDB\$BYTES_PER_CHARACTER  | SMALLINT |        | Size of character in bytes                                                                                                                                           |

## RDB\$CHECK\_CONSTRAINTS

RDB\$CHECK\_CONSTRAINTS stores database integrity constraint information for CHECK constraints. In addition, the table stores information for constraints implemented with NOT NULL.

**Table 6.3** RDB\$CHECK\_CONSTRAINTS

| Column name          | Datatype | Length | Description                                                                                                                                     |
|----------------------|----------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$CONSTRAINT_NAME | CHAR     | 67     | Subtype 2: Name of a CHECK or NOT NULL constraint                                                                                               |
| RDB\$TRIGGER_NAME    | CHAR     | 67     | Subtype 2: Name of the trigger that enforces the CHECK constraint; for a NOT NULL constraint, name of the source column in RDB\$RELATION_FIELDS |

## RDB\$COLLATIONS

RDB\$COLLATIONS records the valid collating sequences available for use in InterBase.

**Table 6.4** RDB\$COLLATIONS

| Column name               | Datatype | Length | Description                                                                                                                                                                                                                                                                                          |
|---------------------------|----------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$COLLATION_NAME       | CHAR     | 67     | Name of a valid collation sequence in InterBase                                                                                                                                                                                                                                                      |
| RDB\$COLLATION_ID         | SMALLINT |        | Unique identifier for the collation sequence                                                                                                                                                                                                                                                         |
| RDB\$CHARACTER_SET_ID     | SMALLINT |        | Identifier of the underlying character set of this collation sequence <ul style="list-style-type: none"> <li>• Required before collation can proceed</li> <li>• Determines which character set is in use Corresponds to the RDB\$CHARACTER_SET_ID column in the RDB\$CHARACTER_SETS table</li> </ul> |
| RDB\$COLLATION_ATTRIBUTES | SMALLINT |        | Reserved for internal use                                                                                                                                                                                                                                                                            |

**Table 6.4** RDB\$COLLATIONS (*continued*)

| Column name        | Datatype | Length | Description                                                                                                                                                        |
|--------------------|----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$SYSTEM_FLAG   | SMALLINT |        | Indicates whether the generator is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul> |
| RDB\$DESCRIPTION   | BLOB     |        | Subtype Text: Contains a user-written description of the collation sequence                                                                                        |
| RDB\$FUNCTION_NAME | CHAR     | 67     | Reserved for internal use                                                                                                                                          |

## RDB\$DATABASE

RDB\$DATABASE defines a database.

**Table 6.5** RDB\$DATABASE

| Column name             | Datatype | Length | Description                                                                                                                                                                     |
|-------------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$DESCRIPTION        | BLOB     |        | Subtype Text: Contains a user-written description of the database; when a comment is included in a CREATE or ALTER SCHEMA DATABASE statement, <b>isql</b> writes to this column |
| RDB\$RELATION_ID        | SMALLINT |        | For internal use by InterBase                                                                                                                                                   |
| RDB\$SECURITY_CLASS     | CHAR     | 67     | Subtype 2: Security class defined in the RDB\$SECURITY_CLASSES table; the access control limits described in the named security class apply to all database usage               |
| RDB\$CHARACTER_SET_NAME | CHAR     | 67     | Subtype 2; Name of character set                                                                                                                                                |
| RDB\$PAGE_CACHE         | INTEGER  |        | Sets database page buffer cache limit. Also, tries to expand cache to that limit.                                                                                               |
| RDB\$PROCEDURE_CACHE    | INTEGER  |        |                                                                                                                                                                                 |
| RDB\$TRIGGER_CACHE      | INTEGER  |        |                                                                                                                                                                                 |
| RDB\$RELATION_CACHE     | SMALLINT |        |                                                                                                                                                                                 |

**Table 6.5** RDB\$DATABASE

| Column name           | Datatype    | Length | Description                                                                                                                                                                                               |
|-----------------------|-------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FLUSH_INTERVAL   | INTEGER     |        | Enables database flush. The interval <number> is interpreted in units of seconds.                                                                                                                         |
| RDB\$LINGER_INTERVAL  | INTEGER     |        | Allows a database to remain in memory after the last user detaches. Interval is seconds                                                                                                                   |
| RDB\$RECLAIM_INTERVAL | INTEGER     |        | Reclaim interval is in seconds. Determines how often the garbage collector thread will run to release memory from unused procedures, triggers, and internal system queries back to InterBase memory heap. |
| RDB\$SWEEP_INTERVAL   | INTEGER     |        |                                                                                                                                                                                                           |
| RDB\$GROUP_COMMIT     | CHAR(1)     |        |                                                                                                                                                                                                           |
| RDB\$PASSWORD_DIGEST  | VARCHAR(16) |        |                                                                                                                                                                                                           |

## RDB\$DEPENDENCIES

RDB\$DEPENDENCIES keeps track of the tables and columns upon which other system objects depend. These objects include views, triggers, and computed columns. InterBase uses this table to ensure that a column or table cannot be deleted if it is used by any other object.

**Table 6.6** RDB\$DEPENDENCIES

| Column name              | Datatype | Length | Description                                                                        |
|--------------------------|----------|--------|------------------------------------------------------------------------------------|
| RDB\$DEPENDENT_NAME      | CHAR     | 67     | Subtype 2; names the object this table tracks: a view, trigger, or computed column |
| RDB\$DEPENDENDED_ON_NAME | CHAR     | 67     | Subtype 2; names the table referenced by the object named above                    |
| RDB\$FIELD_NAME          | CHAR     | 67     | Subtype 2; names the column referenced by the object named above                   |

**Table 6.6** RDB\$DEPENDENCIES (*continued*)

| Column name           | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------|----------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$DEPENDENT_TYPE   | SMALLINT |        | <p>Describes the object type of the object referenced in the RDB\$DEPENDENT_NAME column; type codes (RDB\$TYPES):</p> <ul style="list-style-type: none"><li>• 0 - table</li><li>• 1 - view</li><li>• 2 - trigger</li><li>• 3 - computed_field</li><li>• 4 - validation</li><li>• 5 - procedure</li><li>• 7 - exception</li><li>• 8 - user</li><li>• 9 - field</li><li>• 10 - index</li></ul> <p>All other values are reserved for future use</p>                                                                                        |
| RDB\$DEPENDED_ON_TYPE | SMALLINT |        | <p>Describes the object type of the object referenced in the RDB\$DEPENDED_ON_NAME column; type codes (RDB\$TYPES):</p> <ul style="list-style-type: none"><li>• 0 - table</li><li>• 1 - view</li><li>• 2 - trigger</li><li>• 3 - computed_field</li><li>• 4 - validation</li><li>• 5 - procedure</li><li>• 7 - exception</li><li>• 8 - user</li><li>• 9 - field</li><li>• 10 - index</li><li>• 11 - generator</li><li>• 14 - External Functions</li><li>• 15 - Encryption</li></ul> <p>All other values are reserved for future use</p> |

**RDB\$ENCRYPTIONS**

RDB\$ENCRYPTIONS describes the characteristics of encryptions stored in the database.

**Table 6.7** RDB\$ENCRYPTIONS

| Column name                 | Datatype  | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------|-----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$ENCRYPTION_NAME        | CHAR      | 67     | A unique name for the encryption.                                                                                                                                                                                                                                                                                                                                                                                               |
| RDB\$ENCRYPTION_TYPE        | CHAR      | 16     | BASE: Defines a base encryption that has its own encryption value.<br>COPY: Copy of a BASE encryption that shares the same encryption value.<br>BACKUP: Defines an encryption used to encrypt database backup files.<br>RECOVERY: Defines an encryption that can be used to recover a password-protected encryption when the password has been lost or forgotten. This encryption cannot be used to perform database encryption |
| RDB\$ENCRYPTION_CIPHER      | CHAR      | 16     | Encryption cipher algorithm. This is either AES (Advanced Encryption Standard) or DES (Data Encryption Standard).                                                                                                                                                                                                                                                                                                               |
| RDB\$ENCRYPTION_LENGTH      | SMALLINT  |        | Encryption key length (bits) must be one of these values for AES: 128, 192 or 256. The default is 128. For DES the default is 56.                                                                                                                                                                                                                                                                                               |
| RDB\$ENCRYPTION_INIT_VECTOR | CHAR      | 6      | RANDOM: specifies that random bytes should be used with cipher block chaining (CBC) encryption mode.<br><null>: default, specifies electronic cookbook (ECB) encryption mode used.                                                                                                                                                                                                                                              |
| RDB\$ENCRYPTION_PAD         | CHAR      | 6      | RANDOM: pads value to be encrypted with random bytes.                                                                                                                                                                                                                                                                                                                                                                           |
| RDB\$ENCRYPTION_VALUE       | CHAR      | 68     | Encrypted value of the actual encryption key value.                                                                                                                                                                                                                                                                                                                                                                             |
| RDB\$ENCRYPTION_SALT        | CHAR      | 68     | Hash to verify decrypted value of actual encryption key value is correct.                                                                                                                                                                                                                                                                                                                                                       |
| RDB\$ENCRYPTION_TIMESTAMP   | TIMESTAMP |        | Timestamp when encryption key value was created or refreshed.                                                                                                                                                                                                                                                                                                                                                                   |

**Table 6.7** RDB\$ENCRYPTIONS

| Column name         | Datatype | Length | Description                                                                                                                                                   |
|---------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$SECURITY_CLASS | CHAR     | 67     | Names a security class stored in RDB\$SECURITY_CLASSES.                                                                                                       |
| RDB\$OWNER_NAME     | CHAR     | 67     | Owner of the encryption                                                                                                                                       |
| RDB\$PASSWORD2      | VARCHAR  | 68     | Password hash used to allow access to the encryption.                                                                                                         |
| RDB\$SYSTEM_FLAG    | SMALLINT |        | 0: User-defined 1: System-defined.                                                                                                                            |
| RDB\$FLAGS          | SMALLINT |        | 1: random initialization vector defined for cipher block chaining encryption mode.<br>2: random padding of plaintext<br>4: encryption is marked for deletion. |
| RDB\$DESCRIPTION    | BLOB     |        | Subtype Text: User-written description of encryption.                                                                                                         |

## RDB\$EXCEPTIONS

RDB\$EXCEPTIONS describes error conditions related to stored procedures, including user-defined exceptions.

**Table 6.8** RDB\$EXCEPTIONS

| Column name           | Datatype | Length | Description                                     |
|-----------------------|----------|--------|-------------------------------------------------|
| RDB\$EXCEPTION_NAME   | CHAR     | 67     | Subtype 2; exception name                       |
| RDB\$EXCEPTION_NUMBER | INTEGER  |        | Number for the exception                        |
| RDB\$MESSAGE          | VARCHAR  | 78     | Text of exception message                       |
| RDB\$DESCRIPTION      | BLOB     |        | Subtype Text: Text description of the exception |
| RDB\$SYSTEM_FLAG      | SMALLINT |        | Displays null                                   |

## RDB\$FIELD\_DIMENSIONS

RDB\$FIELD\_DIMENSIONS describes each dimension of an array column.

**Table 6.9** RDB\$FIELD\_DIMENSIONS

| Column name      | Datatype | Length | Description                                                                                                                        |
|------------------|----------|--------|------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FIELD_NAME  | CHAR     | 67     | Subtype 2; names the array column described by this table; the column name must exist in the RDB\$FIELD_NAME column of RDB\$FIELDS |
| RDB\$DIMENSION   | SMALLINT |        | Identifies one dimension of the ARRAY column; the first dimension is identified by the integer 0                                   |
| RDB\$LOWER_BOUND | INTEGER  |        | Indicates the lower bound of the previously specified dimension                                                                    |
| RDB\$UPPER_BOUND | INTEGER  |        | Indicates the upper bound of the previously specified dimension                                                                    |

## RDB\$FIELDS

RDB\$FIELDS defines the characteristics of a column. Each domain or column has a corresponding row in RDB\$FIELDS. Columns are added to tables by means of an entry in the RDB\$RELATION\_FIELDS table, which describes local characteristics.

For domains, RDB\$FIELDS includes domain name, null status, and default values. SQL columns are defined in RDB\$RELATION\_FIELDS. For both domains and simple columns, RDB\$RELATION\_FIELDS can contain default and null status information.

**Table 6.10** RDB\$FIELDS

| Column name            | Datatype | Length | Description                                                                                                                                                                                 |
|------------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FIELD_NAME        | CHAR     | 67     | Unique name of a domain or system-assigned name for a column, starting with SQL <sub>nnn</sub> ; the actual column names are stored in the RDB\$FIELD_SOURCE column of RDB\$RELATION_FIELDS |
| RDB\$QUERY_NAME        | CHAR     | 67     | Not used for SQL objects                                                                                                                                                                    |
| RDB\$VALIDATION_BLR    | BLOB     |        | Not used for SQL objects                                                                                                                                                                    |
| RDB\$VALIDATION_SOURCE | BLOB     |        | Not used for SQL objects                                                                                                                                                                    |
| RDB\$COMPUTED_BLR      | BLOB     |        | Subtype BLR; for computed columns, contains the BLR (Binary Language Representation) of the expression the database evaluates at the time of execution                                      |
| RDB\$COMPUTED_SOURCE   | BLOB     |        | Subtype Text: For computed columns, contains the original CHAR source expression for the column                                                                                             |

**Table 6.10** RDB\$FIELDS (*continued*)

| Column name          | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$DEFAULT_VALUE   | BLOB     |        | Stores default rule; subtype BLR                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| RDB\$DEFAULT_SOURCE  | BLOB     |        | Subtype Text; SQL description of a default value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| RDB\$FIELD_LENGTH    | SMALLINT |        | <p>Length in bytes of the field this row defines:</p> <p>For CHAR, VARCHAR, and NCHAR datatypes, this is the maximum length of the field, and InterBase uses this length when creating indexes on columns.</p> <p>For non-CHAR related datatypes, the column lengths are:</p> <ul style="list-style-type: none"> <li>• D_FLOAT - 8</li> <li>• DOUBLE - 8</li> <li>• DATE - 4</li> <li>• BLOB - 8</li> <li>• TIME - 4</li> <li>• INT64 - 8</li> <li>• SHORT - 2</li> <li>• LONG - 4</li> <li>• QUAD - 8</li> <li>• FLOAT - 4</li> <li>• TIMESTAMP - 8</li> <li>• BOOLEAN - 2</li> </ul> |
| RDB\$FIELD_PRECISION | SMALLINT |        | Stores the precision for numeric and decimal types                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| RDB\$FIELD_SCALE     | SMALLINT |        | Stores negative scale for numeric and decimal types                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

**Table 6.10** RDB\$FIELDS (*continued*)

| Column name     | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------|----------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FIELD_TYPE | SMALLINT |        | <p>Specifies the datatype of the column being defined; changing the value of this column automatically changes the datatype for all columns based on the column being defined</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>• BLOB - 261</li> <li>• BLOB_ID - 45</li> <li>• BOOLEAN - 17</li> <li>• CHAR - 14</li> <li>• CSTRING - 40</li> <li>• D_FLOAT - 11</li> <li>• DOUBLE - 27</li> <li>• FLOAT - 10</li> <li>• INT64 - 16</li> <li>• INTEGER - 8</li> <li>• QUAD - 9</li> <li>• SMALLINT - 7</li> <li>• DATE - 12 (dialect 3 DATE)</li> <li>• TIME - 13</li> <li>• TIMESTAMP - 35</li> <li>• VARCHAR - 37</li> </ul> <p>Restrictions:</p> <ul style="list-style-type: none"> <li>• The value of this column cannot be changed to or from BLOB</li> <li>• Non-numeric data causes a conversion error in a column changed from CHAR to numeric</li> </ul> <p>Changing data from CHAR to numeric and back again adversely affects index performance; for best results, delete and re-create indexes when making this type of change</p> |

**Table 6.10** RDB\$FIELDS (*continued*)

| Column name         | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|----------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FIELD_SUB_TYPE | SMALLINT |        | <p>Used to distinguish types of Blobs, CHARs, and integers</p> <p><b>1</b> If RDB\$FIELD_TYPE is 261 (Blob), predefined subtypes can be:</p> <ul style="list-style-type: none"> <li>• 0 - unspecified</li> <li>• 1 - text</li> <li>• 2 - BLR (Binary Language Representation )</li> <li>• 3 - access control list</li> <li>• 4 - reserved for future use</li> <li>• 5 - encoded description of a table's current metadata</li> <li>• 6 - description of multi-database transaction that finished irregularly</li> </ul> <p><b>2</b> If RDB\$FIELD_TYPE is 14 (CHAR), columns can be:</p> <ul style="list-style-type: none"> <li>• 0 - type is unspecified</li> <li>• 1 - fixed BINARY data</li> </ul> <p>Corresponds to the RDB\$FIELD_SUB_TYPE column in the RDB\$COLLATIONS table</p> <p><b>3</b> If RDB\$FIELD_TYPE is 7 (SMALLINT), 8 (INTEGER), or 16 (INT64), the original declaration was:</p> <ul style="list-style-type: none"> <li>• 0 or NULL - RDB\$FIELD_TYPE</li> <li>• 1 - NUMERIC</li> <li>• 2 - DECIMAL</li> </ul> |
| RDB\$MISSING_VALUE  | BLOB     |        | Not used for SQL objects                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| RDB\$MISSING_SOURCE | BLOB     |        | Not used for SQL objects                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| RDB\$DESCRIPTION    | BLOB     |        | Subtype Text: Contains a user-written description of the column being defined                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| RDB\$SYSTEM_FLAG    | SMALLINT |        | For system tables                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| RDB\$QUERY_HEADER   | BLOB     |        | Not used for SQL objects                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| RDB\$SEGMENT_LENGTH | SMALLINT |        | Used for Blob columns only; a non-binding suggestion for the length of Blob buffers                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| RDB\$EDIT_STRING    | VARCHAR  | 125    | Not used for SQL objects                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

**Table 6.10** RDB\$FIELDS (*continued*)

| Column name           | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$EXTERNAL_LENGTH  | SMALLINT |        | Length of the column as it exists in an external table; if the column is not in an external table, this value is 0                                                                                                                                                                                                                                                                                                                                                                                                               |
| RDB\$EXTERNAL_SCALE   | SMALLINT |        | Scale factor for an external column of an integer datatype; the scale factor is the power of 10 by which the integer is multiplied                                                                                                                                                                                                                                                                                                                                                                                               |
| RDB\$EXTERNAL_TYPE    | SMALLINT |        | Indicates the datatype of the column as it exists in an external table; valid values are: <ul style="list-style-type: none"> <li>• BLOB - 261</li> <li>• BLOB_ID - 45</li> <li>• BOOLEAN - 17</li> <li>• CHAR - 14</li> <li>• CSTRING - 40</li> <li>• D_FLOAT - 11</li> <li>• DOUBLE - 27</li> <li>• FLOAT - 10</li> <li>• INT64 - 16</li> <li>• INTEGER - 8</li> <li>• QUAD - 9</li> <li>• SMALLINT - 7</li> <li>• DATE - 12 (dialect 3 DATE)</li> <li>• TIME - 13</li> <li>• TIMESTAMP - 35</li> <li>• VARCHAR - 37</li> </ul> |
| RDB\$DIMENSIONS       | SMALLINT |        | For an ARRAY datatype, specifies the number of dimensions in the array; for a non-array column, the value is 0                                                                                                                                                                                                                                                                                                                                                                                                                   |
| RDB\$NULL_FLAG        | SMALLINT |        | Indicates whether a column can contain a NULL value<br>Valid values are: <ul style="list-style-type: none"> <li>• Empty: Can contain NULL values</li> <li>• 1: Cannot contain NULL values</li> </ul>                                                                                                                                                                                                                                                                                                                             |
| RDB\$CHARACTER_LENGTH | SMALLINT |        | Length in characters of the field this row defines:<br>For CHAR, VARCHAR, and NCHAR datatypes, this is the quotient of RDB\$FIELD_LENGTH divided by the number of bytes per character in the character set of the field. For other datatypes, this length value is not meaningful, and should be NULL                                                                                                                                                                                                                            |
| RDB\$COLLATION_ID     | SMALLINT |        | Unique identifier for the collation sequence                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

**Table 6.10** RDB\$FIELDS (*continued*)

| Column name                 | Datatype | Length | Description                                                                                                                                 |
|-----------------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$CHARACTER_SET_ID       | SMALLINT |        | ID indicating character set for the character or Blob columns; joins to the CHARACTER_SET_ID column of the RDB\$CHARACTER_SETS system table |
| RDB\$ENCRYPTION_ID          | SMALLINT |        | Identifies encryption ID from RDB\$ENCRYPTIONS used to encrypt this column.                                                                 |
| RDB\$DECRYPT_DEFAULT_VALUES | BLOB     |        | Subtype BLR: BLR (Binary Language Representation) for decrypt default clause.                                                               |
| RDB\$DECRYPT_DEFAULT_SOURCE | BLOB     |        | Subtype Text: SQL to define decrypt default.                                                                                                |

## RDB\$FILES

RDB\$FILES lists the secondary files and shadow files for a database.

**Table 6.11** RDB\$FILES

| Column name        | Datatype | Length | Description                                                                                                                                                                              |
|--------------------|----------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FILE_NAME     | VARCHAR  | 253    | Names either a secondary file or a shadow file for the database                                                                                                                          |
| RDB\$FILE_SEQUENCE | SMALLINT |        | <i>Either</i> the order that secondary files are to be used in the database or the order of files within a shadow set                                                                    |
| RDB\$FILE_START    | INTEGER  |        | Specifies the starting page number for a secondary file or shadow file                                                                                                                   |
| RDB\$FILE_LENGTH   | INTEGER  |        | Specifies the file length in blocks                                                                                                                                                      |
| RDB\$FILE_FLAGS    | SMALLINT |        | Reserved for system use                                                                                                                                                                  |
| RDB\$SHADOW_NUMBER | SMALLINT |        | Set number: indicates to which shadow set the file belongs; if the value of this column is 0 or missing, InterBase assumes the file being defined is a secondary file, not a shadow file |

## RDB\$FILTERS

---

RDB\$FILTERS tracks information about a Blob filter.

**Table 6.12** RDB\$FILTERS

| Column name          | Datatype | Length | Description                                                                                                                                                     |
|----------------------|----------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FUNCTION_NAME   | CHAR     | 67     | Unique name for the filter defined by this row                                                                                                                  |
| RDB\$DESCRIPTION     | BLOB     |        | Subtype Text: Contains a user-written description of the filter being defined                                                                                   |
| RDB\$MODULE_NAME     | VARCHAR  | 253    | Names the library where the filter executable is stored                                                                                                         |
| RDB\$ENTRYPOINT      | CHAR     | 31     | The entry point within the filter library for the Blob filter being defined                                                                                     |
| RDB\$INPUT_SUB_TYPE  | SMALLINT |        | The Blob subtype of the input data                                                                                                                              |
| RDB\$OUTPUT_SUB_TYPE | SMALLINT |        | The Blob subtype of the output data                                                                                                                             |
| RDB\$SYSTEM_FLAG     | SMALLINT |        | Indicates whether the filter is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul> |

---

## RDB\$FORMATS

---

RDB\$FORMATS keeps track of the format versions of the columns in a table. InterBase assigns the table a new format number at each change to a column definition. Direct metadata operations such as ALTER TABLE increment the format version; so do creating, dropping, activating, and deactivating triggers. This table allows existing application programs to access a changed table, without needing to be recompiled.

**Note** InterBase allows only 255 changes to a table's metadata. Once the limit is reached, the database must be backed up and restored before more metadata changes can be made. Only changes that affect a row's structure count toward this limit. Changing a trigger from active to inactive, for example, does not count toward the limit.

**Table 6.13** RDB\$FORMATS

| Column name      | Datatype | Length | Description                                                                                                                                 |
|------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$RELATION_ID | SMALLINT |        | Names a table that exists in RDB\$RELATIONS                                                                                                 |
| RDB\$FORMAT      | SMALLINT |        | Specifies the format number of the table; a table can have any number of different formats, depending on the number of updates to the table |
| RDB\$DESCRIPTOR  | BLOB     |        | Subtype Format: Lists each column in the table, along with its datatype, length, and scale (if applicable)                                  |

## RDB\$FUNCTION\_ARGUMENTS

RDB\$FUNCTION\_ARGUMENTS defines the attributes of a function argument.

**Table 6.14** RDB\$FUNCTION\_ARGUMENTS

| Column name            | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FUNCTION_NAME     | CHAR     | 67     | Unique name of the function with which the argument is associated; must correspond to a function name in RDB\$FUNCTIONS                                                                                                                                                                                                                                                                                                                                                                            |
| RDB\$ARGUMENT_POSITION | SMALLINT |        | Position of the argument described in the RDB\$FUNCTION_NAME column in relation to the other arguments                                                                                                                                                                                                                                                                                                                                                                                             |
| RDB\$MECHANISM         | SMALLINT |        | Specifies whether the argument is passed by value (value of 0) or by reference (value of 1)                                                                                                                                                                                                                                                                                                                                                                                                        |
| RDB\$FIELD_TYPE        | SMALLINT |        | Datatype of the argument being defined<br>Valid values are: <ul style="list-style-type: none"> <li>• BLOB - 261</li> <li>• BLOB_ID - 45</li> <li>• BOOLEAN - 17</li> <li>• CHAR - 14</li> <li>• CSTRING - 40</li> <li>• D_FLOAT - 11</li> <li>• DOUBLE - 27</li> <li>• FLOAT - 10</li> <li>• INT64 - 16</li> <li>• INTEGER - 8</li> <li>• QUAD - 9</li> <li>• SMALLINT - 7</li> <li>• DATE - 12 (dialect 3 DATE)</li> <li>• TIME - 13</li> <li>• TIMESTAMP - 35</li> <li>• VARCHAR - 37</li> </ul> |

**Table 6.14** RDB\$FUNCTION\_ARGUMENTS (*continued*)

| Column name           | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------|----------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FIELD_SCALE      | SMALLINT |        | Scale factor for an argument that has an integer datatype; the scale factor is the power of 10 by which the integer is multiplied                                                                                                                                                                                                                                                               |
| RDB\$FIELD_LENGTH     | SMALLINT |        | <p>The length of the argument defined in this row</p> <p>Valid column lengths are:</p> <ul style="list-style-type: none"> <li>• BLOB - 8</li> <li>• BOOLEAN - 2</li> <li>• D_FLOAT - 8</li> <li>• DATE - 4</li> <li>• DOUBLE - 8</li> <li>• FLOAT - 4</li> <li>• INT64 - 8</li> <li>• LONG - 4</li> <li>• QUAD - 8</li> <li>• SHORT - 2</li> <li>• TIME - 4</li> <li>• TIMESTAMP - 8</li> </ul> |
| RDB\$FIELD_SUB_TYPE   | SMALLINT |        | <p>If RDB\$FIELD_TYPE is 7 (SMALLINT), 8 (INTEGER), or 16 (INT64) the subtype can be:</p> <ul style="list-style-type: none"> <li>• 0 or NULL - RDB\$FIELD_TYPE</li> <li>• 1 - NUMERIC</li> <li>• 2 - DECIMAL</li> </ul>                                                                                                                                                                         |
| RDB\$CHARACTER_SET_ID | SMALLINT |        | Unique numeric identifier for a character set                                                                                                                                                                                                                                                                                                                                                   |
| RDB\$FIELD_PRECISION  | SMALLINT |        | The declared precision of the DECIMAL or NUMERIC function argument                                                                                                                                                                                                                                                                                                                              |

## RDB\$FUNCTIONS

RDB\$FUNCTIONS defines a user-defined function.

**Table 6.15** RDB\$FUNCTIONS

| Column name        | Datatype | Length | Description                                                                     |
|--------------------|----------|--------|---------------------------------------------------------------------------------|
| RDB\$FUNCTION_NAME | CHAR     | 67     | Unique name for a function                                                      |
| RDB\$FUNCTION_TYPE | SMALLINT |        | Reserved for future use                                                         |
| RDB\$QUERY_NAME    | CHAR     | 67     | Alternate name for the function that can be used in <b>isql</b>                 |
| RDB\$DESCRIPTION   | BLOB     |        | Subtype Text: Contains a user-written description of the function being defined |

**Table 6.15** RDB\$FUNCTIONS (*continued*)

| Column name          | Datatype | Length | Description                                                                                                                                             |
|----------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$MODULE_NAME     | VARCHAR  | 253    | Names the function library where the executable function is stored                                                                                      |
| RDB\$ENTRYPOINT      | CHAR     | 31     | Entry point within the function library for the function being defined                                                                                  |
| RDB\$RETURN_ARGUMENT | SMALLINT |        | Position of the argument returned to the calling program; this position is specified in relation to other arguments                                     |
| RDB\$SYSTEM_FLAG     | SMALLINT |        | Indicates whether the function is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value of 1)</li> </ul> |

## RDB\$GENERATORS

RDB\$GENERATORS stores information about generators, which provide the ability to generate a unique identifier for a table.

**Table 6.16** RDB\$GENERATORS

| Column name         | Datatype | Length | Description                                                                                                                                                        |
|---------------------|----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$GENERATOR_NAME | CHAR     | 67     | Name of the table to contain the unique identifier produced by the number generator                                                                                |
| RDB\$GENERATOR_ID   | SMALLINT |        | Unique system-assigned ID number for the generator                                                                                                                 |
| RDB\$SYSTEM_FLAG    | SMALLINT |        | Indicates whether the generator is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul> |

## RDB\$INDEX\_SEGMENTS

RDB\$INDEX\_SEGMENTS specifies the columns that comprise an index for a table. Modifying these rows corrupts rather than changes an index unless the RDB\$INDICES row is deleted and re-created in the same transaction.

**Table 6.17** RDB\$INDEX\_SEGMENTS

| Column name         | Datatype | Length | Description                                                                                                                                        |
|---------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$INDEX_NAME     | CHAR     | 67     | The index associated with this index segment; if the value of this column changes, the RDB\$INDEX_NAME column in RDB\$INDICES must also be changed |
| RDB\$FIELD_NAME     | CHAR     | 67     | The index segment being defined; the value of this column must match the value of the RDB\$FIELD_NAME column in RDB\$RELATION_FIELDS               |
| RDB\$FIELD_POSITION | SMALLINT |        | Position of the index segment being defined; corresponds to the sort order of the index                                                            |

## RDB\$INDICES

RDB\$INDICES defines the index structures that allow InterBase to locate rows in the database more quickly. Because InterBase provides both simple indexes (a single-key column) and multi-segment indexes (multiple-key columns), each index defined in this table must have corresponding occurrences in the RDB\$INDEX\_SEGMENTS table.

**Table 6.18** RDB\$INDICES

| Column name        | Datatype | Length | Description                                                                                                           |
|--------------------|----------|--------|-----------------------------------------------------------------------------------------------------------------------|
| RDB\$INDEX_NAME    | CHAR     | 67     | Names the index being defined; if the value of this column changes, change its value in the RDB\$INDEX_SEGMENTS table |
| RDB\$RELATION_NAME | CHAR     | 67     | Names the table associated with this index; the table must be defined in the RDB\$RELATIONS table                     |
| RDB\$INDEX_ID      | SMALLINT |        | Contains an internal identifier for the index being defined; do <i>not</i> write to this column                       |

**Table 6.18** RDB\$INDICES (*continued*)

| Column name            | Datatype         | Length | Description                                                                                                                                                                                                                                       |
|------------------------|------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$UNIQUE_FLAG       | SMALLINT         |        | Specifies whether the index allows duplicate values<br>Values: <ul style="list-style-type: none"> <li>• 0 - allows duplicate values</li> <li>• 1 - does not allow duplicate values</li> </ul> Eliminate duplicates before creating a unique index |
| RDB\$DESCRIPTION       | BLOB             |        | Subtype Text: User-written description of the index                                                                                                                                                                                               |
| RDB\$SEGMENT_COUNT     | SMALLINT         |        | Number of segments in the index; a value of 1 indicates a simple index                                                                                                                                                                            |
| RDB\$INDEX_INACTIVE    | SMALLINT         |        | Indicates whether the index is: <ul style="list-style-type: none"> <li>• Active (value of 0)</li> <li>• Inactive (value of 1)</li> </ul> This is not set for system tables.                                                                       |
| RDB\$INDEX_TYPE        | SMALLINT         |        | Contains an internal identifier for sort order, either ascending (ASC) or descending (DESC): <ul style="list-style-type: none"> <li>• ASC (value of 0)</li> <li>• DESC (value of 1)</li> </ul>                                                    |
| RDB\$FOREIGN_KEY       | CHAR             | 31     | Name of FOREIGN KEY constraint for which the index is implemented                                                                                                                                                                                 |
| RDB\$SYSTEM_FLAG       | SMALLINT         |        | Indicates whether the index is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul>                                                                                    |
| RDB\$EXPRESSION_BLR    | BLOB             |        | Subtype BLR: Contains the BLR (Binary Language Representation) for the expression, evaluated by the database at execution time; used for PC semantics                                                                                             |
| RDB\$EXPRESSION_SOURCE | BLOB             |        | Subtype Text: Contains original text source for the column; used for PC semantics                                                                                                                                                                 |
| RDB\$STATISTICS        | DOUBLE PRECISION |        | Selectivity factor for the index; the optimizer uses index selectivity, a measure of uniqueness for indexed columns, to choose an access strategy for a query                                                                                     |

## RDB\$JOURNAL\_ARCHIVES

---

RDB\$JOURNAL\_ARCHIVES stores information about the repository of database and journal archive files.

**Table 6.19** RDB\$JOURNAL\_ARCHIVES

| Column name                 | Datatype  | Length | Description                                                                                                                                                                                                                                                       |
|-----------------------------|-----------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$ARCHIVE_NAME           | VARCHAR   | 1024   | The name of the archived item.                                                                                                                                                                                                                                    |
| RDB\$ARCHIVE_TYPE           | CHAR      | 1      | The type of the archived item. 'D' indicates a database dump. 'S' indicates a secondary database file of a database dump. 'J' indicates a journal file.                                                                                                           |
| RDB\$ARCHIVE_LENGTH         | INT64     | 8      | Length of the archived item as stored in bytes.                                                                                                                                                                                                                   |
| RDB\$ARCHIVE_SEQUENCE       | INTEGER   | 4      | Sequence number of archive item.                                                                                                                                                                                                                                  |
| RDB\$ARCHIVE_TIMESTAMP      | TIMESTAMP | 8      | Timestamp when item was stored in the archive.                                                                                                                                                                                                                    |
| RDB\$DEPENDENT_ON_SEQUENCE  | INTEGER   | 4      | Sequence of archived item that this item depends on. For 'S' archive types, it would be the sequence number of the 'D' primary database dump file. For 'D' archive types, it is the sequence number of the starting journal file for recovering from the archive. |
| RDB\$DEPENDENT_ON_TIMESTAMP | TIMESTAMP | 8      | As above, but the archive timestamp for the depended on archive item.                                                                                                                                                                                             |

## RDB\$LOG\_FILES

---

RDB\$LOG\_FILES is no longer used.

## RDB\$PAGES

---

RDB\$PAGES keeps track of each page allocated to the database.

**Important** Modifying this table in any way corrupts a database.

## RDB\$PROCEDURE\_PARAMETERS

**Table 6.20** RDB\$PAGES

| Column name        | Datatype | Length | Description                                                                                                |
|--------------------|----------|--------|------------------------------------------------------------------------------------------------------------|
| RDB\$PAGE_NUMBER   | INTEGER  |        | The physically allocated page number                                                                       |
| RDB\$RELATION_ID   | SMALLINT |        | Identifier number of the table for which this page is allocated                                            |
| RDB\$PAGE_SEQUENCE | INTEGER  |        | The sequence number of this page in the table to other pages allocated for the previously identified table |
| RDB\$PAGE_TYPE     | SMALLINT |        | Describes the type of page; this information is for system use only                                        |

RDB\$PROCEDURE\_PARAMETERS stores information about each parameter for each of a database's procedures.

**Table 6.21** RDB\$PROCEDURE\_PARAMETERS

| Column name           | Datatype | Length | Description                                                                                                                                                        |
|-----------------------|----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$PARAMETER_NAME   | CHAR     | 67     | Parameter name                                                                                                                                                     |
| RDB\$PROCEDURE_NAME   | CHAR     | 67     | Name of the procedure in which the parameter is used                                                                                                               |
| RDB\$PARAMETER_NUMBER | SMALLINT |        | Parameter sequence number                                                                                                                                          |
| RDB\$PARAMETER_TYPE   | SMALLINT |        | Parameter datatype<br>Values are: <ul style="list-style-type: none"> <li>• 0 = input</li> <li>• 1 = output</li> </ul>                                              |
| RDB\$FIELD_SOURCE     | CHAR     | 31     | Global column name                                                                                                                                                 |
| RDB\$DESCRIPTION      | BLOB     |        | Subtype Text: User-written description of the parameter                                                                                                            |
| RDB\$SYSTEM_FLAG      | SMALLINT |        | Indicates whether the parameter is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul> |

## RDB\$PROCEDURES

RDB\$PROCEDURES stores information about a database's stored procedures.

**Table 6.22** RDB\$PROCEDURES

| Column name           | Datatype | Length | Description                                                                                                                                                        |
|-----------------------|----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$PROCEDURE_NAME   | CHAR     | 67     | Procedure name                                                                                                                                                     |
| RDB\$PROCEDURE_ID     | SMALLINT |        | Procedure number                                                                                                                                                   |
| RDB\$PROCEDURE_INPUTS | SMALLINT |        | Number of input parameters                                                                                                                                         |
| PROCEDURE_OUTPUTS     | SMALLINT |        | Number of output parameters                                                                                                                                        |
| RDB\$DESCRIPTION      | BLOB     |        | Subtype Text: User-written description of the procedure                                                                                                            |
| RDB\$PROCEDURE_SOURCE | BLOB     |        | Subtype Text: Source code for the procedure                                                                                                                        |
| RDB\$PROCEDURE_BLR    | BLOB     |        | Subtype BLR: BLR (Binary Language Representation) of the procedure source                                                                                          |
| RDB\$SECURITY_CLASS   | CHAR     | 67     | Security class of the procedure                                                                                                                                    |
| RDB\$OWNER_NAME       | CHAR     | 67     | User who created the procedure (the owner for SQL security purposes)                                                                                               |
| RDB\$RUNTIME          | BLOB     |        | Subtype Summary: Describes procedure metadata; used for performance enhancement                                                                                    |
| RDB\$SYSTEM_FLAG      | SMALLINT |        | Indicates whether the procedure is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul> |

## RDB\$REF\_CONSTRAINTS

RDB\$REF\_CONSTRAINTS stores referential integrity constraint information.

**Table 6.23** RDB\$REF\_CONSTRAINTS

| Column name           | Datatype | Length | Description                                                                                                                                        |
|-----------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$3CONSTRAINT_NAME | CHAR     | 67     | Name of a referential constraint                                                                                                                   |
| RDB\$CONST_NAME_UQ    | CHAR     | 67     | Name of a referenced PRIMARY KEY or UNIQUE constraint                                                                                              |
| RDB\$MATCH_OPTION     | CHAR     | 7      | Reserved for later use; currently defaults to FULL                                                                                                 |
| RDB\$UPDATE_RULE      | CHAR     | 11     | Specifies the type of action on the foreign key when the primary key is updated; values are RESTRICT, NO ACTION, CASCADE, SET NULL, or SET DEFAULT |
| RDB\$DELETE_RULE      | CHAR     | 11     | Specifies the type of action on the foreign key when the primary key is DELETED; values are RESTRICT, NO ACTION, CASCADE, SET NULL, or SET DEFAULT |

## RDB\$RELATION\_CONSTRAINTS

RDB\$RELATION\_CONSTRAINTS stores information about integrity constraints for tables.

**Table 6.24** RDB\$RELATION\_CONSTRAINTS

| Column name          | Datatype | Length | Description                                                                                                                                                                                      |
|----------------------|----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$CONSTRAINT_NAME | CHAR     | 67     | Name of a table constraint                                                                                                                                                                       |
| RDB\$CONSTRAINT_TYPE | CHAR     | 11     | Type of table constraint<br>Constraint types are: <ul style="list-style-type: none"> <li>• PRIMARY KEY</li> <li>• UNIQUE</li> <li>• FOREIGN KEY</li> <li>• PCHECK</li> <li>• NOT NULL</li> </ul> |
| RDB\$RELATION_NAME   | CHAR     | 67     | Name of the table for which the constraint is defined                                                                                                                                            |

**Table 6.24** RDB\$RELATION\_CONSTRAINTS (*continued*)

| Column name             | Datatype | Length | Description                                                               |
|-------------------------|----------|--------|---------------------------------------------------------------------------|
| RDB\$DEFERRABLE         | CHAR     | 3      | Reserved for later use; currently defaults to No                          |
| RDB\$INITIALLY_DEFERRED | CHAR     | 3      | Reserved for later use; currently defaults to No                          |
| RDB\$INDEX_NAME         | CHAR     | 67     | Name of the index used by UNIQUE, PRIMARY KEY, or FOREIGN KEY constraints |

## RDB\$RELATION\_FIELDS

For database tables, RDB\$RELATION\_FIELDS lists columns and describes column characteristics for domains.

SQL columns are defined in RDB\$RELATION\_FIELDS. The column name is correlated in the RDB\$FIELD\_SOURCE column to an underlying entry in RDB\$FIELDS that contains a system name ("SQL\$<n>"). This entry includes information such as column type and length. For both domains and simple columns, this table may contain default and nullability information.

**Table 6.25** RDB\$RELATION\_FIELDS

| Column name      | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                 |
|------------------|----------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$QUERY_NAME  | CHAR     | 31     | Alternate column name for use in <b>isql</b> ; supersedes the value in RDB\$FIELDS                                                                                                                                                                                                                                          |
| RDB\$BASE_FIELD  | CHAR     | 31     | Views only: The name of the column from RDB\$FIELDS in a table or view that is the base for a view column being defined; for the base column: <ul style="list-style-type: none"> <li>RDB\$BASE_FIELD provides the column name</li> <li>RDB\$VIEW_CONTEXT, a column in this table, provides the source table name</li> </ul> |
| RDB\$EDIT_STRING | VARCHAR  | 125    | Not used in SQL                                                                                                                                                                                                                                                                                                             |

**Table 6.25** RDB\$RELATION\_FIELDS (*continued*)

| Column name         | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FIELD_POSITION | SMALLINT |        | <p>The position of the column in relation to other columns:</p> <ul style="list-style-type: none"> <li>• <b>isql</b> obtains the ordinal position for displaying column values when printing rows from this column</li> <li>• <b>gpre</b> uses the column order for SELECT and INSERT statements</li> </ul> <p>If two or more columns in the same table have the same value for this column, those columns appear in random order</p> |
| RDB\$QUERY_HEADER   | BLOB     |        | Not used in SQL                                                                                                                                                                                                                                                                                                                                                                                                                       |
| RDB\$UPDATE_FLAG    | SMALLINT |        | Not used by InterBase; included for compatibility with other DSRI-based systems                                                                                                                                                                                                                                                                                                                                                       |
| RDB\$FIELD_ID       | SMALLINT |        | <p>Identifier for use in BLR (Binary Language Representation) to name the column</p> <ul style="list-style-type: none"> <li>• Because this identifier changes during backup and restoration of the database, try to use it in transient requests only</li> <li>• Do <i>not</i> modify this column</li> </ul>                                                                                                                          |
| RDB\$VIEW_CONTEXT   | SMALLINT |        | Alias used to qualify view columns by specifying the table location of the base column; it must have the same value as the alias used in the view BLR (Binary Language Representation) for this context stream                                                                                                                                                                                                                        |
| RDB\$DESCRIPTION    | BLOB     |        | Subtype Text: User-written description of the column being defined                                                                                                                                                                                                                                                                                                                                                                    |
| RDB\$DEFAULT_VALUE  | BLOB     |        | Subtype BLR: BLR (Binary Language Representation) for default clause                                                                                                                                                                                                                                                                                                                                                                  |
| RDB\$SYSTEM_FLAG    | SMALLINT |        | <p>Indicates whether the column is:</p> <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul>                                                                                                                                                                                                                                                                |
| RDB\$SECURITY_CLASS | CHAR     | 67     | Names a security class defined in the RDB\$SECURITY_CLASSES table; the access restrictions defined by this security class apply to all users of this column                                                                                                                                                                                                                                                                           |
| RDB\$COMPLEX_NAME   | CHAR     | 67     | Reserved for future use                                                                                                                                                                                                                                                                                                                                                                                                               |

**Table 6.25** RDB\$RELATION\_FIELDS (*continued*)

| Column name                 | Datatype | Length | Description                                                                   |
|-----------------------------|----------|--------|-------------------------------------------------------------------------------|
| RDB\$NULL_FLAG              | SMALLINT |        | Indicates whether the column may contain NULLs                                |
| RDB\$DEFAULT_SOURCE         | BLOB     |        | Subtype Text: SQL source to define defaults                                   |
| RDB\$COLLATION_ID           | SMALLINT |        | Unique identifier for the collation sequence                                  |
| RDB\$DECRYPT_DEFAULT_SOURCE | BLOB     |        | Subtype Text: SQL to define decrypt default.                                  |
| RDB\$DECRYPT_DEFAULT_VALUE  | BLOB     |        | Subtype BLR: BLR (Binary Language Representation) for decrypt default clause. |
| RDB\$ENCRYPTION_ID          | SMALLINT |        | Identifies encryption ID from RDB\$ENCRYPTIONS used to encrypt this column.   |

## RDB\$RELATIONS

RDB\$RELATIONS defines some of the characteristics of tables and views. Other characteristics, such as the columns included in the table and a description of each column, are stored in the RDB\$RELATION\_FIELDS table.

**Table 6.26** RDB\$RELATIONS

| Column name       | Datatype | Length | Description                                                                                                                          |
|-------------------|----------|--------|--------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$VIEW_BLR     | BLOB     |        | Subtype BLR: For a view, contains the BLR (Binary Language Representation) of the query InterBase evaluates at the time of execution |
| RDB\$VIEW_SOURCE  | BLOB     |        | Subtype Text: For a view, contains the original source query for the view definition                                                 |
| RDB\$_DESCRIPTION | BLOB     |        | Subtype Text: Contains a user-written description of the table being defined                                                         |
| RDB\$RELATION_ID  | SMALLINT |        | Contains the internal identification number used in BLR (Binary Language Representation) requests; do <i>not</i> modify this column  |

**Table 6.26** RDB\$RELATIONS (*continued*)

| Column name               | Datatype | Length | Description                                                                                                                                                                                                                                    |
|---------------------------|----------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$SYSTEM_FLAG          | SMALLINT |        | Indicates the contents of a table, either: <ul style="list-style-type: none"> <li>• User-data (value of 0)</li> <li>• System information (value greater than 0)</li> </ul> Do <i>not</i> set this column to 1 when creating tables             |
| RDB\$DBKEY_LENGTH         | SMALLINT |        | Length of the database key<br>Values are: <ul style="list-style-type: none"> <li>• For tables: 8</li> <li>• For views: 8 times the number of tables referenced in the view definition</li> </ul> Do <i>not</i> modify the value of this column |
| RDB\$FORMAT               | SMALLINT |        | For InterBase internal use only; do <i>not</i> modify                                                                                                                                                                                          |
| RDB\$FIELD_ID             | SMALLINT |        | The number of columns in the table; this column is maintained by InterBase: do <i>not</i> modify the value of this column                                                                                                                      |
| RDB\$RELATION_NAME        | CHAR     | 67     | The unique name of the table defined by this row                                                                                                                                                                                               |
| RDB\$SECURITY_CLASS       | CHAR     | 67     | Security class defined in the RDB\$SECURITY_CLASSES table; access controls defined in the security class apply to all uses of this table                                                                                                       |
| RDB\$EXTERNAL_FILE        | VARCHAR  | 253    | The file in which the external table is stored; if this is blank, the table does not correspond to an external file                                                                                                                            |
| RDB\$RUNTIME              | BLOB     |        | Subtype Summary: Describes table metadata; used for performance enhancement                                                                                                                                                                    |
| RDB\$EXTERNAL_DESCRIPTION | BLOB     |        | Subtype EXTERNAL_FILE_DESCRIPTION; user-written description of the external file                                                                                                                                                               |

**Table 6.26** RDB\$RELATIONS (*continued*)

| Column name                     | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$OWNER_NAME                 | CHAR     | 67     | Identifies the creator of the table or view; the creator is considered the owner for SQL security (GRANT/REVOKE) purposes                                                                                                                                                                                                                                                                                            |
| RDB\$DEFAULT_CLASS              | CHAR     | 31     | Default security class that InterBase applies to columns newly added to a table using the SQL security system                                                                                                                                                                                                                                                                                                        |
| RDB\$FLAGS                      | SMALLINT |        | 1 = SQL-defined table<br>2 = Global temporary table<br>4 = <reserved for future use><br>8 = Delete temporary rows on commit<br>16 = Preserve temporary rows on commit; rows are deleted on database detach                                                                                                                                                                                                           |
| RDB\$DATA_BLOCKING_FACTOR<br>OR | SMALLINT |        | ODS 15 creates a new column which stores a table-specific record blocking factor. It is set during GBAK restore based on the characteristics of the restored data.<br><br>If a table does not have a table-specific data blocking factor, this system column queries as NULL.                                                                                                                                        |
| RDB\$BLOB_BLOCKING_FACTOR<br>OR | BLOB     |        | ODS 15 creates a new column which stores a table-specific blob blocking factor. It is set during GBAK restore based on the characteristics of the restored blobs.<br><br>If a table does not have a table-specific blob blocking factor, this system column queries as NULL.<br><br><b>Note:</b> If a table has Blob columns and no indexes defined then the table uses the database-wide blocking factor as before. |

## RDB\$ROLES

---

RDB\$ROLES lists roles that have been defined in the database and the owner of each role.

**Table 6.27** RDB\$ROLES

| Column name     | Datatype | Length | Description                                     |
|-----------------|----------|--------|-------------------------------------------------|
| RDB\$ROLE_NAME  | CHAR     | 67     | Name of role being defined                      |
| RDB\$OWNER_NAME | CHAR     | 67     | Name of InterBase user who is creating the role |

## RDB\$SECURITY\_CLASSES

RDB\$SECURITY\_CLASSES defines access control lists and associates them with databases, tables, views, and columns in tables and views. For all SQL objects, the information in this table is duplicated in the RDB\$USER\_PRIVILEGES system table.

**Table 6.28** RDB\$SECURITY\_CLASSES

| Column name         | Datatype | Length | Description                                                                                                                                                                     |
|---------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$SECURITY_CLASS | CHAR     | 67     | Security class being defined; if the value of this column changes, change its name in the RDB\$SECURITY_CLASS column in RDB\$DATABASE, RDB\$RELATIONS, and RDB\$RELATION_FIELDS |
| RDB\$ACL            | BLOB     |        | Subtype ACL: Access control list that specifies users and the privileges granted to those users                                                                                 |
| RDB\$DESCRIPTION    | BLOB     |        | Subtype Text: User-written description of the security class being defined                                                                                                      |

## RDB\$TRANSACTIONS

---

RDB\$TRANSACTIONS keeps track of all multi-database transactions.

**Table 6.29** RDB\$TRANSACTIONS

| Column name                  | Datatype | Length | Description                                                                                                                                                                   |
|------------------------------|----------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$TRANSACTION_ID          | INTEGER  |        | Identifies the multi-database transaction being described                                                                                                                     |
| RDB\$TRANSACTION_STATE       | SMALLINT |        | Indicates the state of the transaction<br>Valid values are: <ul style="list-style-type: none"> <li>• 0 - limbo</li> <li>• 1 - committed</li> <li>• 2 - rolled back</li> </ul> |
| RDB\$TIMESTAMP               | DATE     |        | Reserved for future use                                                                                                                                                       |
| RDB\$TRANSACTION_DESCRIPTION | BLOB     |        | Subtype TRANSACTION_DESCRIPTION; describes a prepared multi-database transaction, available if the reconnect fails                                                            |

---

## RDB\$TRIGGER\_MESSAGES

---

RDB\$TRIGGER\_MESSAGES defines a trigger message and associates the message with a particular trigger.

**Table 6.30** RDB\$TRIGGER\_MESSAGES

| Column name         | Datatype | Length | Description                                                                                          |
|---------------------|----------|--------|------------------------------------------------------------------------------------------------------|
| RDB\$TRIGGER_NAME   | CHAR     | 67     | Names the trigger associated with this trigger message; the trigger name must exist in RDB\$TRIGGERS |
| RDB\$MESSAGE_NUMBER | SMALLINT |        | The message number of the trigger message being defined; the maximum number of messages is 32,767    |
| RDB\$MESSAGE        | VARCHAR  | 78     | The source for the trigger message                                                                   |

---

## RDB\$TRIGGERS

---

RDB\$TRIGGERS defines triggers.

**Table 6.31** RDB\$TRIGGERS

| Column name           | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------|----------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$TRIGGER_NAME     | CHAR     | 67     | Names the trigger being defined                                                                                                                                                                                                                                                                                                                                 |
| RDB\$RELATION_NAME    | CHAR     | 67     | Name of the table associated with the trigger being defined; this name must exist in RDB\$RELATIONS                                                                                                                                                                                                                                                             |
| RDB\$TRIGGER_SEQUENCE | SMALLINT |        | <p>Sequence number for the trigger being defined; determines when a trigger is executed in relation to others of the same type</p> <ul style="list-style-type: none"> <li>• Triggers with the same sequence number execute in alphabetic order by trigger name</li> <li>• If this number is not assigned by the user, InterBase assigns a value of 0</li> </ul> |
| RDB\$TRIGGER_TYPE     | SMALLINT |        | <p>The type of trigger being defined</p> <p>Values are:</p> <ul style="list-style-type: none"> <li>• 1 - BEFORE INSERT</li> <li>• 2 - AFTER INSERT</li> <li>• 3 - BEFORE UPDATE</li> <li>• 4 - AFTER UPDATE</li> <li>• 5 - BEFORE DELETE</li> <li>• 6 - AFTER DELETE</li> </ul>                                                                                 |
| RDB\$TRIGGER_SOURCE   | BLOB     |        | Subtype Text: Original source of the trigger definition; the <b>isql</b> SHOW TRIGGERS statement displays information from this column                                                                                                                                                                                                                          |
| RDB\$TRIGGER_BLR      | BLOB     |        | Subtype BLR: BLR (Binary Language Representation) of the trigger source                                                                                                                                                                                                                                                                                         |
| RDB\$DESCRIPTION      | BLOB     |        | Subtype Text: User-written description of the trigger being defined; when including a comment in a CREATE TRIGGER or ALTER TRIGGER statement, <b>isql</b> writes to this column                                                                                                                                                                                 |

**Table 6.31** RDB\$TRIGGERS (*continued*)

| Column name           | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                     |
|-----------------------|----------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$TRIGGER_INACTIVE | SMALLINT |        | Indicates whether the trigger being defined is: <ul style="list-style-type: none"> <li>• Active (value of 0)</li> <li>• Inactive (value of 1)</li> </ul>                                                                                                                                                                        |
| RDB\$SYSTEM_FLAG      | SMALLINT |        | Indicates whether the trigger is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul>                                                                                                                                                                |
| RDB\$FLAGS            | SMALLINT |        | 1 = SQL-defined trigger<br>2 = ignore permission checking<br>User-defined triggers require that the user executing them have underlying access permission to the objects accessed by the trigger. However, internal, system-defined triggers occasionally need to bypass those permission checks to enforce database integrity. |

## RDB\$TYPES

RDB\$TYPES records enumerated datatypes and alias names for InterBase character sets and collation orders. This capability is not available in the current release.

**Table 6.32** RDB\$TYPES

| Column name     | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                  |
|-----------------|----------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$FIELD_NAME | CHAR     | 67     | Column for which the enumerated datatype is being defined                                                                                                                                                                                                                                                                                                    |
| RDB\$TYPE       | SMALLINT |        | Identifies the internal number that represents the column specified above; type codes (same as RDB\$DEPENDENT_TYPES): <ul style="list-style-type: none"> <li>• 0 - table</li> <li>• 1 - view</li> <li>• 2 - trigger</li> <li>• 3 - computed_field</li> <li>• 4 - validation</li> <li>• 5 - procedure</li> </ul> All other values are reserved for future use |

**Table 6.32** RDB\$TYPES (*continued*)

| Column name      | Datatype | Length | Description                                                                                                                                                       |
|------------------|----------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$TYPE_NAME   | CHAR     | 67     | Text that corresponds to the internal number                                                                                                                      |
| RDB\$DESCRIPTION | BLOB     |        | Subtype Text: Contains a user-written description of the enumerated datatype being defined                                                                        |
| RDB\$SYSTEM_FLAG | SMALLINT |        | Indicates whether the datatype is: <ul style="list-style-type: none"> <li>• User-defined (value of 0)</li> <li>• System-defined (value greater than 0)</li> </ul> |

## RDB\$USER\_PRIVILEGES

RDB\$USER\_PRIVILEGES keeps track of the privileges assigned to a user through a SQL GRANT statement. There is one occurrence of this table for each user/privilege intersection.

**Table 6.33** RDB\$USER\_PRIVILEGES

| Column name        | Datatype | Length | Description                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$USER          | CHAR     | 31     | Names the user who was granted the privilege listed in the RDB\$PRIVILEGE column                                                                                                                                                                                                                                                                                                                                                                         |
| RDB\$GRANTOR       | CHAR     | 31     | Names the user who granted the privilege                                                                                                                                                                                                                                                                                                                                                                                                                 |
| RDB\$PRIVILEGE     | CHAR     | 6      | Identifies the privilege granted to the user listed in the RDB\$USER column, above. The character stored in the field corresponds to the valid values listed below.<br>Valid values are: <ul style="list-style-type: none"> <li>• ALL (A)</li> <li>• SELECT (S)</li> <li>• DELETE (D)</li> <li>• INSERT (I)</li> <li>• UPDATE (U)</li> <li>• REFERENCE (R)</li> <li>• MEMBER OF (for roles) (M)</li> <li>• DECRYPT (T)</li> <li>• ENCRYPT (E)</li> </ul> |
| RDB\$GRANT_OPTION  | SMALLINT |        | Indicates whether the privilege was granted with the WITH GRANT OPTION (value of 1) or not (value of 0)                                                                                                                                                                                                                                                                                                                                                  |
| RDB\$RELATION_NAME | CHAR     | 67     | Identifies the table or role to which the privilege applies                                                                                                                                                                                                                                                                                                                                                                                              |

**Table 6.33** RDB\$USER\_PRIVILEGES

| Column name      | Datatype | Length | Description                                                                 |
|------------------|----------|--------|-----------------------------------------------------------------------------|
| RDB\$FIELD_NAME  | CHAR     | 67     | For update privileges, identifies the column to which the privilege applies |
| RDB\$USER_TYPE   | SMALLINT |        |                                                                             |
| RDB\$OBJECT_TYPE | SMALLINT |        |                                                                             |

## RDB\$USERS

RDB\$USERS only permits users in that system table access to the database.

**Table 6.34** RDB\$USERS

| Column name                                  | Datatype     | Length                                                               | Description |
|----------------------------------------------|--------------|----------------------------------------------------------------------|-------------|
| RDB\$USER_NAME<br>(RDB\$USER_NAME)           | VARCHAR(128) | CHARACTER SET<br>UNICODE_FSS<br>Nullable                             |             |
| RDB\$SYSTEM_USER_NAME<br>(RDB\$USER_NAME)    | VARCHAR(128) | CHARACTER SET<br>UNICODE_FSS<br>Nullable                             |             |
| RDB\$GROUP_NAME<br>(RDB\$USER_NAME)          | VARCHAR(128) | CHARACTER SET<br>UNICODE_FSS<br>Nullable                             |             |
| RDB\$UID (RDB\$UID)                          | INTEGER      | Nullable                                                             |             |
| RDB\$GID (RDB\$GID)                          | INTEGER      | Nullable                                                             |             |
| RDB\$PASSWORD<br>(RDB\$PASSWORD)             | VARCHAR(32)  | CHARACTER SET<br>OCTETS Nullable                                     |             |
| RDB\$USER_ACTIVE<br>(RDB\$USER_ACTIVE)       | CHAR(2)      | Nullable                                                             |             |
| RDB\$USER_PRIVILEGE<br>(RDB\$USER_PRIVILEGE) | INTEGER      | Nullable                                                             |             |
| RDB\$DESCRIPTION<br>(RDB\$DESCRIPTION)       | BLOB         | segment 80, subtype<br>TEXT CHARACTER<br>SET UNICODE_FSS<br>Nullable |             |
| RDB\$FIRST_NAME<br>(RDB\$NAME_PART)          | VARCHAR(32)  | CHARACTER SET<br>UNICODE_FSS<br>Nullable                             |             |
| RDB\$MIDDLE_NAME<br>(RDB\$NAME_PART)         | VARCHAR(32)  | CHARACTER SET<br>UNICODE_FSS<br>Nullable                             |             |

**Table 6.34** RDB\$USERS

| Column name                                    | Datatype    | Length                                   | Description |
|------------------------------------------------|-------------|------------------------------------------|-------------|
| RDB\$LAST_NAME<br>(RDB\$NAME_PART)             | VARCHAR(32) | CHARACTER SET<br>UNICODE_FSS<br>Nullable |             |
| RDB\$DEFAULT_ROLE<br>(RDB\$USER)               | CHAR(67)    | CHARACTER SET<br>UNICODE_FSS<br>Nullable |             |
| RDB\$PASSWORD_DIGEST<br>(RDB\$PASSWORD_DIGEST) | VARCHAR(16) | Nullable                                 |             |

## RDB\$VIEW\_RELATIONS

RDB\$VIEW\_RELATIONS is not used by SQL objects.

**Table 6.35** RDB\$VIEW\_RELATIONS

| Column name        | Datatype | Length | Description                                                                                                                                                                                                                                                           |
|--------------------|----------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDB\$VIEW_NAME     | CHAR     | 67     | Name of a view: The combination of RDB\$VIEW_NAME and RDB\$VIEW_CONTEXT must be unique                                                                                                                                                                                |
| RDB\$RELATION_NAME | CHAR     | 67     | Name of a table referenced in the view definition                                                                                                                                                                                                                     |
| RDB\$VIEW_CONTEXT  | SMALLINT |        | Alias used to qualify view columns; must have the same value as the alias used in the view BLR (Binary Language Representation) for this query                                                                                                                        |
| RDB\$CONTEXT_NAME  | CHAR     | 67     | Textual version of the alias identified in RDB\$VIEW_CONTEXT<br>This variable must: <ul style="list-style-type: none"> <li>• Match the value of the RDB\$VIEW_SOURCE column for the corresponding table in RDB\$RELATIONS</li> <li>• Be unique in the view</li> </ul> |

# System Temporary Tables

The InterBase server keeps a massive collection of information about its databases, connections, transactions, and statements. This information is made available through the following system temporary tables. For more information about using these tables, see the InterBase [Operations Guide](#).

Temporary table names begin with TMP\$. InterBase offers the following system temporary tables:

**TMP\$ATTACHMENTS**      **TMP\$DATABASE**      **TMP\$POOL\_BLOCKS**  
**TMP\$POOLS**              **TMP\$PROCEDURES**      **TMP\$RELATIONS**  
**TMP\$STATEMENTS**      **TMP\$TRANSACTIONS**

## TMP\$ATTACHMENTS

The TMP\$ATTACHMENTS table contains one row for each connection to a database.

**Table 6.36** TMP\$ATTACHMENTS

| Column name        | Data type | Description                   |
|--------------------|-----------|-------------------------------|
| TMP\$ATTACHMENT_ID | INTEGER   | Connection identifier         |
| TMP\$DATABASE_ID   | INTEGER   | Database identifier           |
| TMP\$POOL_ID       | INTEGER   | Reserved                      |
| TMP\$POOL_MEMORY   | INTEGER   | Reserved                      |
| TMP\$STATEMENTS    | SMALLINT  | Number of compiled statements |
| TMP\$TRANSACTIONS  | SMALLINT  | Number of active transactions |
| TMP\$TIMESTAMP     | TIMESTAMP | Connection create timestamp   |
| TMP\$QUANTUM       | INTEGER   | Units of execution            |
| TMP\$USER          | CHAR[67]  | User name                     |
| TMP\$USER_IP_ADDR  | CHAR [31] | User IP address               |
| TMP\$USER_HOST     | CHAR [31] | User host name                |
| TMP\$USER_PROCESS  | CHAR [31] | User process ID               |
| TMP\$STATE         | CHAR [31] | CONNECTED, ACTIVE             |
| TMP\$PRIORITY      | CHAR [31] | Reserved                      |
| TMP\$DBKEY_ID      | INTEGER   | Transaction ID of dbkey       |

**Table 6.36** TMP\$ATTACHMENTS (*continued*)

| Column name              | Data type | Description                     |
|--------------------------|-----------|---------------------------------|
| TMP\$ACTIVE_SORTS        | SMALLINT  | Number of active sorts          |
| TMP\$PAGE_READS          | INTEGER   | Page reads all database files   |
| TMP\$PAGE_WRITES         | INTEGER   | Page writes all database files  |
| TMP\$PAGE_FETCHES        | INTEGER   | Page fetches all database files |
| TMP\$PAGE_MARKS          | INTEGER   | Page marks all database files   |
| TMP\$RECORD_SELECTS      | INTEGER   | Records selected by connection  |
| TMP\$RECORD_INSERTS      | INTEGER   | Records inserted by connection  |
| TMP\$RECORD_UPDATES      | INTEGER   | Records updated by connection   |
| TMP\$RECORD_DELETES      | INTEGER   | Records deleted by connection   |
| TMP\$RECORD_PURGES       | INTEGER   | Garbage collect record purges   |
| TMP\$RECORD_EXPUNGE<br>S | INTEGER   | Garbage collect record expunges |
| TMP\$RECORD_BACKOUT<br>S | INTEGER   | Garbage collect record backouts |

## TMP\$DATABASE

TMP\$DATABASE contains one row for each database you are attached to.

**Table 6.37** TMP\$DATABASE

| Column name          | Data type    | Description                           |
|----------------------|--------------|---------------------------------------|
| TMP\$DATABASE_ID     | INTEGER      | Database identifier                   |
| TMP\$DATABASE_PATH   | VARCHAR[253] | Database pathname                     |
| TMP\$ATTACHMENTS     | SMALLINT     | Number of active connections          |
| TMP\$STATEMENTS      | SMALLINT     | Number of compiled statements         |
| TMP\$STATE           | CHAR[31]     | FLUSH, SWEEP, RECLAIM                 |
| TMP\$ALLOCATED_PAGES | INTEGER      | Pages allocated to all database files |
| TMP\$POOLS           | INTEGER      | Number of memory pools                |

**Table 6.37** TMP\$DATABASE (*continued*)

| Column name                | Data type | Description                            |
|----------------------------|-----------|----------------------------------------|
| TMP\$PROCEDURES            | SMALLINT  | Number of procedures loaded            |
| TMP\$RELATIONS             | SMALLINT  | Number of relations loaded             |
| TMP\$TRIGGERS              | SMALLINT  | Number of triggers loaded              |
| TMP\$ACTIVE_THREADS        | SMALLINT  | Active threads in database             |
| TMP\$SORT_MEMORY           | INTEGER   | Sort buffer allocated memory           |
| TMP\$CURRENT_MEMORY        | INTEGER   | Current memory allocated database      |
| TMP\$MAXIMUM_MEMORY        | INTEGER   | Maximum memory ever allocated          |
| TMP\$PERMANENT_POOL_MEMORY | INTEGER   | Permanent pool memory size             |
| TMP\$CACHE_POOL_MEMORY     | INTEGER   | Buffer pool memory size                |
| TMP\$TRANSACTIONS          | SMALLINT  | Number of active transactions          |
| TMP\$TRANSACTION_COMMITS   | INTEGER   | Number of transaction commits          |
| TMP\$TRANSACTION_ROLLBACKS | INTEGER   | Number of transaction rollbacks        |
| TMP\$TRANSACTION_PREPARES  | INTEGER   | Number of transaction prepares         |
| TMP\$TRANSACTION_DEADLOCKS | INTEGER   | Number of transaction deadlocks        |
| TMP\$TRANSACTION_CONFLICTS | INTEGER   | Number of transaction update conflicts |
| TMP\$TRANSACTION_WAITS     | INTEGER   | Number of transaction wait for         |
| TMP\$NEXT_TRANSACTION      | INTEGER   | Next transaction number                |
| TMP\$OLDEST_INTERESTING    | INTEGER   | Oldest interesting transaction         |
| TMP\$OLDEST_ACTIVE         | INTEGER   | Oldest active transaction              |
| TMP\$OLDEST_SNAPSHOT       | INTEGER   | Oldest snapshot transaction            |
| TMP\$CACHE_BUFFERS         | INTEGER   | Number of cache buffers                |

**Table 6.37** TMP\$DATABASE (*continued*)

| Column name            | Data type | Description                       |
|------------------------|-----------|-----------------------------------|
| TMP\$CACHE_PRECEDENCE  | INTEGER   | Nodes in cache precedence graph   |
| TMP\$CACHE_LATCH_WAITS | INTEGER   | Buffer latch waits                |
| TMP\$CACHE_FREE_WAITS  | INTEGER   | Number of waits for a free buffer |
| TMP\$CACHE_FREE_WRITES | INTEGER   | Number of writes to free buffers  |
| TMP\$SWEEP_INTERVAL    | INTEGER   | Sweep trigger interval            |
| TMP\$SWEEP_ACTIVE      | CHAR[1]   | Y (active) N (not-active)         |
| TMP\$SWEEP_RELATION    | CHAR[67]  | Relation currently being swept    |
| TMP\$SWEEP_RECORDS     | INTEGER   | Records swept in above relation   |
| TMP\$PAGE_READS        | INTEGER   | Page reads all database files     |
| TMP\$PAGE_WRITES       | INTEGER   | Page writes all database files    |
| TMP\$PAGE_FETCHES      | INTEGER   | Page fetches all database files   |
| TMP\$PAGE_MARKS        | INTEGER   | Page marks all database files     |
| TMP\$RECORD_SELECTS    | INTEGER   | Records selected from database    |
| TMP\$RECORD_INSERTS    | INTEGER   | Records inserted into database    |
| TMP\$RECORD_UPDATES    | INTEGER   | Records updated to database       |
| TMP\$RECORD_DELETES    | INTEGER   | Records deleted from database     |
| TMP\$RECORD_PURGES     | INTEGER   | Garbage collect record purges     |
| TMP\$RECORD_EXPUNGES   | INTEGER   | Garbage collect record expunges   |
| TMP\$RECORD_BACKOUTS   | INTEGER   | Garbage collect record backouts   |

## TMP\$HEAPS

TMP\$HEAPS contains one row for each entry in the InterBase Random and Block heap.

**Table 6.38** TMP\$HEAPS

| Column name      | Data type     | Description                           |
|------------------|---------------|---------------------------------------|
| TMP\$HEAP_TYPE   | CHAR[31]      | RANDOM or BLOCK                       |
| TMP\$HEX_ADDRESS | CHAR[31]      | Memory address of a free block in hex |
| TMP\$ADDRESS     | NUMERIC(18,0) | Memory address of free block          |
| TMP\$FREE_MEMORY | INTEGER       | Amount of free memory in the block    |

## TMP\$POOL\_BLOCKS

The TMP\$POOL\_BLOCKS table contains one row for each block of memory in each pool.

**Table 6.39** TMP\$POOL\_BLOCKS

| Column name  | Data type | Description             |
|--------------|-----------|-------------------------|
| TMP\$POOL_ID | INTEGER   |                         |
| TMP\$ACC     | INTEGER   |                         |
| TMP\$ARR     | INTEGER   |                         |
| TMP\$ATT     | INTEGER   |                         |
| TMP\$BCB     | INTEGER   | Buffer control block    |
| TMP\$BDB     | INTEGER   | Buffer descriptor block |
| TMP\$BLB     | INTEGER   | Blob block              |
| TMP\$BLF     | INTEGER   |                         |
| TMP\$BMS     | INTEGER   |                         |
| TMP\$BTB     | INTEGER   |                         |
| TMP\$BTC     | INTEGER   |                         |
| TMP\$CHARSET | INTEGER   |                         |
| TMP\$CSB     | INTEGER   | Compiler scratch block  |

**Table 6.39** TMP\$POOL\_BLOCKS (*continued*)

| Column name    | Data type | Description                    |
|----------------|-----------|--------------------------------|
| TMP\$CSCONVERT | INTEGER   |                                |
| TMP\$DBB       | INTEGER   | Database block                 |
| TMP\$DCC       | INTEGER   | Data compression control block |
| TMP\$DFW       | INTEGER   | Deferred work block            |
| TMP\$DLS       | INTEGER   |                                |
| TMP\$EXT       | INTEGER   |                                |
| TMP\$FIL       | INTEGER   | File block                     |
| TMP\$FLD       | INTEGER   |                                |
| TMP\$FMT       | INTEGER   | Format block                   |
| TMP\$FRB       | INTEGER   | Free block                     |
| TMP\$FUN       | INTEGER   |                                |
| TMP\$HUNK      | INTEGER   | Hunk block                     |
| TMP\$IDB       | INTEGER   |                                |
| TMP\$IDL       | INTEGER   |                                |
| TMP\$IRB       | INTEGER   |                                |
| TMP\$IRL       | INTEGER   |                                |
| TMP\$LCK       | INTEGER   | Lock block                     |
| TMP\$LWT       | INTEGER   |                                |
| TMP\$MAP       | INTEGER   |                                |
| TMP\$MFB       | INTEGER   |                                |
| TMP\$NOD       | INTEGER   | Node block                     |
| TMP\$OPT       | INTEGER   |                                |
| TMP\$PRC       | INTEGER   |                                |
| TMP\$PRE       | INTEGER   | Precedence block               |
| TMP\$PRM       | INTEGER   |                                |
| TMP\$REC       | INTEGER   | Record block                   |
| TMP\$REL       | INTEGER   | Relation block                 |

**Table 6.39** TMP\$POOL\_BLOCKS (*continued*)

| Column name   | Data type | Description           |
|---------------|-----------|-----------------------|
| TMP\$REQ      | INTEGER   | Request block         |
| TMP\$RIV      | INTEGER   |                       |
| TMP\$RSB      | INTEGER   | Record source block   |
| TMP\$RSC      | INTEGER   |                       |
| TMP\$SAV      | INTEGER   |                       |
| TMP\$SBM      | INTEGER   | Sparse bitmap block   |
| TMP\$SCL      | INTEGER   |                       |
| TMP\$SDW      | INTEGER   |                       |
| TMP\$SMB      | INTEGER   | Sort map block        |
| TMP\$SRPB     | INTEGER   |                       |
| TMP\$STR      | INTEGER   | String block          |
| TMP\$SVC      | INTEGER   |                       |
| TMP\$SYM      | INTEGER   |                       |
| TMP\$TEXTTYPE | INTEGER   |                       |
| TMP\$TFB      | INTEGER   | Temporary field block |
| TMP\$TPC      | INTEGER   |                       |
| TMP\$TRA      | INTEGER   | Transaction block     |
| TMP\$USR      | INTEGER   |                       |
| TMP\$VCL      | INTEGER   | Vector long block     |
| TMP\$VCT      | INTEGER   |                       |
| TMP\$VCX      | INTEGER   |                       |
| TMP\$XCP      | INTEGER   |                       |

## TMP\$POOLS

The TMP\$POOLS table contains one row for each current memory pool. A pool is a collection of memory to support the allocation needs of an internal system object.

**Table 6.40** TMP\$POOLS

| Column name               | Data type | Description                   |
|---------------------------|-----------|-------------------------------|
| TMP\$POOL_ID              | INTEGER   | Pool identifier               |
| TMP\$TYPE                 | CHAR[31]  | Pool type                     |
| TMP\$POOL_MEMORY          | INTEGER   | Total memory in pool          |
| TMP\$FREE_MEMORY          | INTEGER   | Free memory in pool           |
| TMP\$EXTEND_MEMORY        | INTEGER   | Memory by which pool extended |
| TMP\$FREE_STACK_NODES     | SMALLINT  | Free linked list stack nodes  |
| TMP\$FREE_BITMAP_BUCKETS  | SMALLINT  | Free bitmap buckets           |
| TMP\$FREE_BITMAP_SEGMENTS | INTEGER   | Free bitmap segments          |

## TMP\$PROCEDURES

The TMP\$PROCEDURES table contains one row for each procedure executed since the current connection began.

**Table 6.41** TMP\$PROCEDURES

| Column name         | Data type | Description                        |
|---------------------|-----------|------------------------------------|
| TMP\$PROCEDURE_ID   | INTEGER   | Procedure identifier               |
| TMP\$DATABASE_ID    | INTEGER   | Database identifier                |
| TMP\$PROCEDURE_NAME | CHAR[67]  | Procedure name                     |
| TMP\$POOL_ID        | INTEGER   | Pool identifier                    |
| TMP\$POOL_MEMORY    | INTEGER   | Pool memory size                   |
| TMP\$CLONE          | SMALLINT  | Cloned instance number             |
| TMP\$TIMESTAMP      | TIMESTAMP | Start time of procedure            |
| TMP\$USE_COUNT      | SMALLINT  | Statements compiled with procedure |
| TMP\$QUANTUM        | INTEGER   | Units of execution                 |
| TMP\$INVOCATIONS    | INTEGER   | Number of calls to procedure       |
| TMP\$PAGE_READS     | INTEGER   | Page reads all database files      |
| TMP\$PAGE_WRITES    | INTEGER   | Page writes all database files     |
| TMP\$PAGE_FETCHES   | INTEGER   | Page fetches all database files    |
| TMP\$PAGE_MARKS     | INTEGER   | Page marks all database files      |

**Table 6.41** TMP\$PROCEDURES (*continued*)

| Column name              | Data type | Description                     |
|--------------------------|-----------|---------------------------------|
| TMP\$RECORD_SELECTS      | INTEGER   | Records selected by procedure   |
| TMP\$RECORD_INSERTS      | INTEGER   | Records inserted by procedure   |
| TMP\$RECORD_UPDATES      | INTEGER   | Records updated by procedure    |
| TMP\$RECORD_DELETES      | INTEGER   | Records deleted by procedure    |
| TMP\$RECORD_PURGES       | INTEGER   | Garbage collect record purges   |
| TMP\$RECORD_EXPUNGE<br>S | INTEGER   | Garbage collect record expunges |
| TMP\$RECORD_BACKOUT<br>S | INTEGER   | Garbage collect record backouts |

## TMP\$RELATIONS

The TMP\$RELATIONS table contains one row for each relation referenced since the current connection began.

**Table 6.42** TMP\$RELATIONS

| Column name                    | Data type | Description                             |
|--------------------------------|-----------|-----------------------------------------|
| TMP\$RELATION_ID               | SMALLINT  | Relation identifier                     |
| TMP\$DATABASE_ID               | INTEGER   | Database identifier                     |
| TMP\$RELATION_NAME             | CHAR[67]  | Relation name                           |
| TMP\$USE_COUNT                 | SMALLINT  | Statements compiled against relation    |
| TMP\$SWEEP_COUNT               | SMALLINT  | Database sweep or garbage collector     |
| TMP\$SCAN_COUNT                | INTEGER   | Sequential scans                        |
| TMP\$FORMATS                   | SMALLINT  | Number of relation formats              |
| TMP\$POINTER_PAGES             | INTEGER   | Number of relation pointer pages        |
| TMP\$DATA_PAGES                | INTEGER   | Number of relation data pages           |
| TMP\$GARBAGE_COLLECT_PAGE<br>S | INTEGER   | Number of data pages to garbage collect |
| TMP\$PAGE_READS                | INTEGER   | Page reads all database files           |
| TMP\$PAGE_WRITES               | INTEGER   | Page writes all database files          |
| TMP\$PAGE_FETCHES              | INTEGER   | Page fetches all database files         |
| TMP\$PAGE_MARKS                | INTEGER   | Page marks all database files           |

**Table 6.42** TMP\$RELATIONS (*continued*)

| Column name             | Data type | Description                         |
|-------------------------|-----------|-------------------------------------|
| TMP\$RECORD_IDX_SELECTS | INTEGER   | Records selected by index retrieval |
| TMP\$RECORD_SEQ_SELECTS | INTEGER   | Records selected by sequential scan |
| TMP\$RECORD_INSERTS     | INTEGER   | Records inserted into relation      |
| TMP\$RECORD_UPDATES     | INTEGER   | Records updated in relation         |
| TMP\$RECORD_DELETES     | INTEGER   | Records deleted from relation       |
| TMP\$RECORD_PURGES      | INTEGER   | Garbage collect record purges       |
| TMP\$RECORD_EXPUNGES    | INTEGER   | Garbage collect record expunges     |
| TMP\$RECORD_BACKOUTS    | INTEGER   | Garbage collect record backouts     |

## TMP\$STATEMENTS

The TMP\$STATEMENTS table contains one row for each statement currently executing for any current connection.

**Table 6.43** TMP\$STATEMENTS

| Column name         | Data type     | Description                          |
|---------------------|---------------|--------------------------------------|
| TMP\$STATEMENT_ID   | INTEGER       | Statement identifier                 |
| TMP\$ATTACHMENT_ID  | INTEGER       | Connection identifier                |
| TMP\$TRANSACTION_ID | INTEGER       | Transaction number                   |
| TMP\$SQL            | VARCHAR[4094] | SQL string                           |
| TMP\$POOL_ID        | INTEGER       | Pool identifier                      |
| TMP\$POOL_MEMORY    | INTEGER       | Pool memory size                     |
| TMP\$CLONE          | SMALLINT      | Cloned instance number               |
| TMP\$TIMESTAMP      | TIMESTAMP     | Start time of statement              |
| TMP\$QUANTUM        | INTEGER       | Units of execution                   |
| TMP\$INVOCATIONS    | INTEGER       | Number of calls to statement         |
| TMP\$STATE          | CHAR[31]      | ACTIVE, INACTIVE, STALLED, CANCELLED |
| TMP\$PRIORITY       | CHAR[31]      | Reserved                             |
| TMP\$PAGE_READS     | INTEGER       | Page reads all database files        |

**Table 6.43** TMP\$STATEMENTS (*continued*)

| Column name              | Data type | Description                     |
|--------------------------|-----------|---------------------------------|
| TMP\$PAGE_WRITES         | INTEGER   | Page writes all database files  |
| TMP\$PAGE_FETCHES        | INTEGER   | Page fetches all database files |
| TMP\$PAGE_MARKS          | INTEGER   | Page marks all database files   |
| TMP\$RECORD_SELECTS      | INTEGER   | Records selected by statement   |
| TMP\$RECORD_INSERTS      | INTEGER   | Records inserted by statement   |
| TMP\$RECORD_UPDATES      | INTEGER   | Records updated by statement    |
| TMP\$RECORD_DELETES      | INTEGER   | Records deleted by statement    |
| TMP\$RECORD_PURGES       | INTEGER   | Garbage collect record purges   |
| TMP\$RECORD_EXPUNGE<br>S | INTEGER   | Garbage collect record expunges |
| TMP\$RECORD_BACKOUT<br>S | INTEGER   | Garbage collect record backouts |

## TMP\$TRANSACTIONS

The TMP\$TRANSACTIONS table contains one row for each transaction that is active or in limbo.

**Table 6.44** TMP\$TRANSACTIONS

| Column name         | Data type | Description                  |
|---------------------|-----------|------------------------------|
| TMP\$TRANSACTION_ID | INTEGER   | Transaction number           |
| TMP\$ATTACHMENT_ID  | INTEGER   | Connection identifier        |
| TMP\$POOL_ID        | INTEGER   |                              |
| TMP\$POOL_MEMORY    | INTEGER   |                              |
| TMP\$TIMESTAMP      | TIMESTAMP | Start time of connection     |
| TMP\$SNAPSHOT       | INTEGER   | Snapshot transaction number  |
| TMP\$QUANTUM        | INTEGER   | Units of execution           |
| TMP\$SAVEPOINTS     | INTEGER   | savepoint number of records  |
| TMP\$READONLY       | CHAR[1]   | Transaction is read only     |
| TMP\$WRITE          | CHAR[1]   | Transaction has written data |
| TMP\$NOWAIT         | CHAR[1]   | Transaction is no wait       |

**Table 6.44** TMP\$TRANSACTIONS (*continued*)

| Column name           | Data type | Description                             |
|-----------------------|-----------|-----------------------------------------|
| TMP\$COMMIT_RETAINING | CHAR[1]   | Commit retaining performed              |
| TMP\$STATE            | CHAR[31]  | ACTIVE, LIMBO, COMMITTING, PRECOMMITTED |
| TMP\$PRIORITY         | CHAR      | Reserved                                |
| TMP\$TYPE             | CHAR[31]  | SNAPSHOT, READ_COMMITTED                |
| TMP\$PAGE_READS       | INTEGER   | Page reads all database files           |
| TMP\$PAGE_WRITES      | INTEGER   | Page writes all database files          |
| TMP\$PAGE_FETCHES     | INTEGER   | Page fetches all database files         |
| TMP\$PAGE_MARKS       | INTEGER   | Page marks all database files           |
| TMP\$RECORD_SELECTS   | INTEGER   | Records selected by transaction         |
| TMP\$RECORD_INSERTS   | INTEGER   | Records inserted by transaction         |
| TMP\$RECORD_UPDATES   | INTEGER   | Records updated by transaction          |
| TMP\$RECORD_DELETES   | INTEGER   | Records deleted by transaction          |
| TMP\$RECORD_PURGES    | INTEGER   | Garbage collect record purges           |
| TMP\$RECORD_EXPUNGES  | INTEGER   | Garbage collect record expunges         |
| TMP\$RECORD_BACKOUTS  | INTEGER   | Garbage collect record backouts         |

## TMP\$TRIGGERS

The TMP\$TRIGGERS table contains one row for each trigger executed since the current connection began.

**Table 6.45** TMP\$TRIGGERS

| Column name        | Data type | Description               |
|--------------------|-----------|---------------------------|
| TMP\$TRIGGER_ID    | INTEGER   | Trigger identifier        |
| TMP\$DATABASE_ID   | INTEGER   | Database identifier       |
| TMP\$RELATION_NAME | CHAR[67]  | Relation name for trigger |
| TMP\$TRIGGER_NAME  | CHAR[67]  | Trigger name              |

**Table 6.45** TMP\$TRIGGERS (*continued*)

| Column name            | Data type | Description                                                                                                                                                                                                                                                                                     |
|------------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TMP\$TRIGGER_TYPE      | SMALLINT  | The type of trigger being defined<br>Values are:<br>1 - BEFORE INSERT<br>2 - AFTER INSERT<br>3 - BEFORE UPDATE<br>4 - AFTER UPDATE<br>5 - BEFORE DELETE<br>6 - AFTER DELETE                                                                                                                     |
| TMP\$TRIGGER_SEQUENCE  | SMALLINT  | Sequence number for the trigger being defined; determines when a trigger is executed in relation to others of the same type<br>Triggers with the same sequence number execute in alphabetic order by trigger name<br>If this number is not assigned by the user, InterBase assigns a value of 0 |
| TMP\$TRIGGER_ORDER     | CHAR[31]  | Position of the trigger                                                                                                                                                                                                                                                                         |
| TMP\$TRIGGER_OPERATION | CHAR[31]  | UPDATE, DELETE or INSERT                                                                                                                                                                                                                                                                        |
| TMP\$POOL_ID           | INTEGER   | Pool identifier                                                                                                                                                                                                                                                                                 |
| TMP\$POOL_MEMORY       | INTEGER   | Pool memory size                                                                                                                                                                                                                                                                                |
| TMP\$CLONE             | SMALLINT  | Cloned instance number                                                                                                                                                                                                                                                                          |
| TMP\$TIMESTAMP         | TIMESTAMP | Start time of trigger                                                                                                                                                                                                                                                                           |
| TMP\$QUANTUM           | INTEGER   | Units of Execution                                                                                                                                                                                                                                                                              |
| TMP\$INVOCATIONS       | INTEGER   | Number of calls to trigger                                                                                                                                                                                                                                                                      |
| TMP\$PAGE_READS        | INTEGER   | Page reads all database file                                                                                                                                                                                                                                                                    |
| TMP\$PAGE_WRITES       | INTEGER   | Page writes all database files                                                                                                                                                                                                                                                                  |
| TMP\$PAGE_FETCHES      | INTEGER   | Page fetches all database files                                                                                                                                                                                                                                                                 |
| TMP\$PAGE_MARKS        | INTEGER   | Page marks all database files                                                                                                                                                                                                                                                                   |
| TMP\$RECORD_SELECTS    | INTEGER   | Records seelcted by trigger                                                                                                                                                                                                                                                                     |
| TMP\$RECORD_INSERTS    | INTEGER   | Records inserted by trigger                                                                                                                                                                                                                                                                     |
| TMP\$RECORD_UPDATES    | INTEGER   | Records updated by trigger                                                                                                                                                                                                                                                                      |
| TMP\$RECORD_DELETES    | INTEGER   | Records deleted by procedure                                                                                                                                                                                                                                                                    |

**Table 6.45** TMP\$TRIGGERS (*continued*)

| Column name          | Data type | Description                     |
|----------------------|-----------|---------------------------------|
| TMP\$RECORD_PURGES   | INTEGER   | Garbage collect record purges   |
| TMP\$RECORD_EXPUNGES | INTEGER   | Garbage collect record expunges |
| TMP\$RECORD_BACKOUTS | INTEGER   | Garbage collect record backout  |

## System Views

You can create a SQL script using the code provided in this section to create four views that provide information about existing integrity constraints for a database. You must create the database prior to creating these views. SQL system views are a subset of system views defined in the SQL-92 standard. Since they are defined by ANSI SQL-92, the names of the system views and their columns do not start with RDB\$.

- The CHECK\_CONSTRAINTS view

```
CREATE VIEW CHECK_CONSTRAINTS (
 CONSTRAINT_NAME,
 CHECK_CLAUSE
) AS
SELECT RDB$CONSTRAINT_NAME, RDB$TRIGGER_SOURCE
FROM RDB$CHECK_CONSTRAINTS RC, RDB$TRIGGERS RT
WHERE RT.RDB$TRIGGER_NAME = RC.RDB$TRIGGER_NAME;
```

- The CONSTRAINTS\_COLUMN\_USAGE view

```
CREATE VIEW CONSTRAINTS_COLUMN_USAGE (
 TABLE_NAME,
 COLUMN_NAME,
 CONSTRAINT_NAME
) AS
SELECT RDB$RELATION_NAME, RDB$FIELD_NAME,
RDB$CONSTRAINT_NAME
FROM RDB$RELATION_CONSTRAINTS RC, RDB$INDEX_SEGMENTS RI
WHERE RI.RDB$INDEX_NAME = RC.RDB$INDEX_NAME;
```

- The REFERENTIAL\_CONSTRAINTS view

```
CREATE VIEW REFERENTIAL_CONSTRAINTS (
 CONSTRAINT_NAME,
 UNIQUE_CONSTRAINT_NAME,
 MATCH_OPTION,
 UPDATE_RULE,
 DELETE_RULE
) AS
```

```

SELECT RDB$CONSTRAINT_NAME, RDB$CONST_NAME_UQ,
RDB$MATCH_OPTION,
 RDB$UPDATE_RULE, RDB$DELETE_RULE
FROM RDB$REF_CONSTRAINTS;

```

- The **TABLE\_CONSTRAINTS** view

```

CREATE VIEW TABLE_CONSTRAINTS (
 CONSTRAINT_NAME,
 TABLE_NAME,
 CONSTRAINT_TYPE,
 IS_DEFERRABLE,
 INITIALLY_DEFERRED
) AS
SELECT RDB$CONSTRAINT_NAME, RDB$RELATION_NAME,
 RDB$CONSTRAINT_TYPE, RDB$DEFERRABLE,
RDB$INITIALLY_DEFERRED
FROM RDB$RELATION_CONSTRAINTS;

```

## CHECK\_CONSTRAINTS

---

CHECK\_CONSTRAINTS identifies all CHECK constraints defined in the database.

**Table 6.46** CHECK\_CONSTRAINTS

| Column name     | Datatype | Length | Description                                                                                                                  |
|-----------------|----------|--------|------------------------------------------------------------------------------------------------------------------------------|
| CONSTRAINT_NAME | CHAR     | 67     | Unique name for the CHECK constraint; nullable                                                                               |
| CHECK_CLAUSE    | BLOB     |        | Subtype Text: Nullable; original source of the trigger definition, stored in the RDB\$TRIGGER_SOURCE COLUMN in RDB\$TRIGGERS |

---

## CONSTRAINTS\_COLUMN\_USAGE

---

CONSTRAINTS\_COLUMN\_USAGE identifies columns used by PRIMARY KEY and UNIQUE constraints. For FOREIGN KEY constraints, this view identifies the columns defining the constraint.

**Table 6.47** CONSTRAINTS\_COLUMN\_USAGE

| Column name     | Datatype | Length | Description                                         |
|-----------------|----------|--------|-----------------------------------------------------|
| TABLE_NAME      | CHAR     | 67     | Table for which the constraint is defined; nullable |
| COLUMN_NAME     | CHAR     | 67     | Column used in the constraint definition; nullable  |
| CONSTRAINT_NAME | CHAR     | 67     | Unique name for the constraint; nullable            |

## REFERENTIAL\_CONSTRAINTS

REFERENTIAL\_CONSTRAINTS identifies all referential constraints defined in a database.

**Table 6.48** REFERENTIAL\_CONSTRAINTS

| Column name            | Datatype | Length | Description                                                                                                  |
|------------------------|----------|--------|--------------------------------------------------------------------------------------------------------------|
| CONSTRAINT_NAME        | CHAR     | 67     | Unique name for the constraint; nullable                                                                     |
| UNIQUE_CONSTRAINT_NAME | CHAR     | 67     | Name of the UNIQUE or PRIMARY KEY constraint corresponding to the specified referenced column list; nullable |
| MATCH_OPTION           | CHAR     | 7      | Reserved for future use; always set to FULL; nullable                                                        |
| UPDATE_RULE            | CHAR     | 11     | Reserved for future use; always set to RESTRICT; nullable                                                    |
| DELETE_RULE            | CHAR     | 11     | Reserved for future use; always set to RESTRICT; nullable                                                    |

## TABLE\_CONSTRAINTS

---

TABLE\_CONSTRAINTS identifies all constraints defined in a database.

**Table 6.49** TABLE\_CONSTRAINTS

| Column name        | Datatype | Length | Description                                                               |
|--------------------|----------|--------|---------------------------------------------------------------------------|
| CONSTRAINT_NAME    | CHAR     | 67     | Unique name for the constraint; nullable                                  |
| TABLE_NAME         | CHAR     | 67     | Table for which the constraint is defined; nullable                       |
| CONSTRAINT_TYPE    | CHAR     | 11     | Possible values are UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK; nullable |
| IS_DEFERRABLE      | CHAR     | 3      | Reserved for future use; always set to No; nullable                       |
| INITIALLY_DEFERRED | CHAR     | 3      | Reserved for future use; always set to No; nullable                       |

---

# Character Sets and Collation Orders

CHAR, VARCHAR, and text Blob columns in InterBase can use many different character sets. A *character set* defines the symbols that can be entered as text in a column, and it also defines the maximum number of bytes of storage necessary to represent each symbol. In some character sets, such as ISO8859\_1, each symbol requires only a single byte of storage. In others, such as UNICODE\_FSS, each symbol requires from 1 to 3 bytes of storage.

Each character set also has an implicit *collation order* that specifies how its symbols are sorted and ordered. Some character sets also support alternative collation orders. In all cases, choice of character set limits choice of collation orders.

This chapter lists available character sets and their corresponding collation orders and describes how to specify:

- Default character set for an entire database
- Alternative character set and collation order for a particular column in a table
- Client application character set that the server should use when translating data between itself and the client
- Collation order for a value in a comparison operation
- Collation order in an ORDER BY or GROUP BY clause

# InterBase Character Sets and Collation Orders

---

The following table lists each character set that can be used in InterBase. For each character set, the minimum and maximum number of bytes used to store each character is listed, and all collation orders supported for that character set are also listed. The first collation order for a given character set is that set's default collation, the one that is used if no COLLATE clause specifies an alternative order.

**Table 7.1** Character sets and collation orders

| Character set | Char. set ID | Max. char. size | Min. char. size | Collation orders                                                                                                                                                     |
|---------------|--------------|-----------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ASCII         | 2            | 1 byte          | 1 byte          | ASCII                                                                                                                                                                |
| BIG_5         | 56           | 2 bytes         | 1 byte          | BIG_5                                                                                                                                                                |
| CYRL          | 50           | 1 byte          | 1 byte          | CYRL<br>DB_RUS<br>PDOX_CYRL                                                                                                                                          |
| DOS437        | 10           | 1 byte          | 1 byte          | DOS437<br>DB_DEU437<br>DB_ESP437<br>DB_FIN437<br>DB_FRA437<br>DB_ITA437<br>DB_NLD437<br>DB_SVE437<br>DB_UK437<br>DB_US437<br>PDOX_ASCII<br>PDOX_INTL<br>PDOX_SWEDFIN |
| DOS850        | 11           | 1 byte          | 1 byte          | DOS850<br>DB_DEU850<br>DB_ESP850<br>DB_FRA850<br>DB_FRC850<br>DB_ITA850<br>DB_NLD850<br>DB_PTB850<br>DB_SVE850<br>DB_UK850<br>DB_US850                               |
| DOS852        | 45           | 1 byte          | 1 byte          | DOS852<br>DB_CSY<br>DB_PLK<br>DB_SLO<br>PDOX_CSY<br>PDOX_HUN<br>PDOX_PLK<br>PDOX_SLO                                                                                 |

---

**Table 7.1** Character sets and collation orders (*continued*)

| Character set | Char. set ID | Max. char. size | Min. char. size | Collation orders                                                                                                                                                     |
|---------------|--------------|-----------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DOS857        | 46           | 1 byte          | 1 byte          | DOS857<br>DB_TRK                                                                                                                                                     |
| DOS860        | 13           | 1 byte          | 1 byte          | DOS860<br>DB_PTG860                                                                                                                                                  |
| DOS861        | 47           | 1 byte          | 1 byte          | DOS861<br>PDOX_ISL                                                                                                                                                   |
| DOS863        | 14           | 1 byte          | 1 byte          | DOS863<br>DB_FRC863                                                                                                                                                  |
| DOS865        | 12           | 1 byte          | 1 byte          | DOS865<br>DB_DAN865<br>DB_NOR865<br>PDOX_NORDAN4                                                                                                                     |
| EUCJ_0208     | 6            | 2 bytes         | 1 byte          | EUJC_0208                                                                                                                                                            |
| GB_2312       | 57           | 2 bytes         | 1 byte          | GB_2312                                                                                                                                                              |
| ISO8859_1     | 21           | 1 byte          | 1 byte          | ISO8859_1<br>CC_ESPLAT1<br>CC_PTBRLAT1<br>DA_DA<br>DE_DE<br>DU_NL<br>EN_UK<br>EN_US<br>ES_ES<br>FI_FI<br>FR_CA<br>FR_FR<br>IS_IS<br>IT_IT<br>NO_NO<br>PT_PT<br>SV_SV |
| ISO8859_2     | 22           | 1 byte          | 1 byte          | ISO8859_2<br>CS_CZ<br>PL_PL                                                                                                                                          |

**Table 7.1** Character sets and collation orders (*continued*)

| Character set        | Char. set ID | Max. char. size | Min. char. size | Collation orders                                                                                                                                       |
|----------------------|--------------|-----------------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| ISO8859_15           | 39           | 1 byte          | 1 byte          | ISO8859_15<br>DA_DA9<br>DE_DE9<br>DU_NL9<br>EN_UK9<br>EN_US9<br>ES_ES9<br>FI_FI9<br>FR_CA9<br>FR_FR9<br>IS_IS9<br>IT_IT9<br>NO_NO9<br>PT_PT9<br>SV_SV9 |
| KO18R                | 58           | 1 byte          | 1 byte          | RU_RU                                                                                                                                                  |
| KSC_5601             | 44           | 2 bytes         | 1 byte          | KSC_5601<br>KSC_DICTIONARY                                                                                                                             |
| NEXT                 | 19           | 1 byte          | 1 byte          | NEXT<br>NXT_DEU<br>NXT_FRA<br>NXT_ITA<br>NXT_US                                                                                                        |
| NONE                 | 0            | 1 byte          | 1 byte          | NONE                                                                                                                                                   |
| OCTETS               | 1            | 1 byte          | 1 byte          | OCTETS                                                                                                                                                 |
| SJIS_0208            | 5            | 2 bytes         | 1 byte          | SJIS_0208                                                                                                                                              |
| UNICODE_BE<br>UCS2BE | 8            | 2 bytes         | 2 bytes         | N/A at this time                                                                                                                                       |
| UNICODE_FSS          | 3            | 3 bytes         | 1 byte          | UNICODE_FSS                                                                                                                                            |
| UNICODE_LE<br>UCS2LE | 64           | 2 byte          | 2 bytes         | N/A                                                                                                                                                    |
| UTF_8                | 59           | 4 byte          | 1 bytes         | N/A at this time.                                                                                                                                      |
| WIN1250              | 51           | 1 byte          | 1 byte          | WIN1250<br>PXW_CSY<br>PXW_HUNDC<br>PXW_PLK<br>PXW_SLO                                                                                                  |
| WIN1251              | 52           | 1 byte          | 1 byte          | WIN1251<br>PXW_CYRL                                                                                                                                    |

**Table 7.1** Character sets and collation orders (*continued*)

| Character set | Char. set ID | Max. char. size | Min. char. size | Collation orders                                                             |
|---------------|--------------|-----------------|-----------------|------------------------------------------------------------------------------|
| WIN1252       | 53           | 1 byte          | 1 byte          | WIN1252<br>PXW_INTL<br>PXW_INTL850<br>PXW_NORDAN4<br>PXW_SPAN<br>PXW_SWEDFIN |
| WIN1253       | 54           | 1 byte          | 1 byte          | WIN1253<br>PXW_GREEK                                                         |
| WIN1254       | 55           | 1 byte          | 1 byte          | WIN1254<br>PXW_TURK                                                          |

## Character Set Storage Requirements

Knowing the storage requirements of a particular character set is important, because InterBase restricts the maximum amount of storage in each field in the column to 32,767 bytes for CHAR columns and 32,765 for VARCHAR columns. In the case of a single-byte character column, one character is stored in one byte, so you can define 32,767 (or 32,765 for VARCHAR) characters per single-byte column without encountering an error.

For multi-byte character sets, to determine the maximum number of characters allowed in a column definition, divide the internal byte storage limit for the datatype by the number of bytes for each character. Thus, two-byte character sets have a character limit of 16,383 per field, and three-byte character sets have a limit of 10,922 characters per field. For VARCHAR columns, the numbers are 16,382 and 10,921 respectively.

The following examples specify a CHAR datatype using the UNICODE\_FSS character set, which has a maximum size of three bytes for a single character:

```
CHAR (10922) CHARACTER SET UNICODE_FSS; /* succeeds */
CHAR (10923) CHARACTER SET UNICODE_FSS; /* fails */
```

## Support for Paradox and dBASE

Many character sets and their corresponding collations are provided to support Paradox for DOS, Paradox for Windows, dBASE for DOS, and dBASE for Windows.

### Character Sets for DOS

The following character sets correspond to MS-DOS code pages, and should be used to specify character sets for InterBase databases that are accessed by Paradox for DOS and dBASE for DOS:

**Table 7.2** Character sets corresponding to DOS code pages

| Character set | DOS code page |
|---------------|---------------|
| DOS437        | 437           |
| DOS850        | 850           |
| DOS852        | 852           |
| DOS857        | 857           |
| DOS860        | 860           |
| DOS861        | 861           |
| DOS863        | 863           |
| DOS865        | 865           |

The names of collation orders for these character sets that are specific to Paradox begin “PDOX”. For example, the DOS865 character set for DOS code page 865 supports a Paradox collation order for Norwegian and Danish called “PDOX\_NORDAN4”.

The names of collation orders for these character sets that are specific to dBASE begin “DB”. For example, the DOS437 character set for DOS code page 437 supports a dBASE collation order for Spanish called “DB\_ESP437”.

For more information about DOS code pages, and Paradox and dBASE collation orders, see the appropriate Paradox and dBASE documentation and driver books.

## Character Sets for Microsoft Windows

There are five character sets that support Windows client applications, such as Paradox for Windows. These character sets are WIN1250, WIN1251, WIN1252, WIN1253, and WIN1254.

The names of collation orders for these character sets that are specific to Paradox for Windows begin “PXW”. For example, the WIN1252 character set supports a Paradox for Windows collation order for Norwegian and Danish called “PXW\_NORDAN4”.

For more information about Windows character sets and Paradox for Windows collation orders, see the appropriate Paradox for Windows documentation and driver books.

## Additional Character Sets and Collations

Support for additional character sets and collation orders is constantly being added to InterBase. To see if additional character sets and collations are available for a newly created database, connect to the database with **isql**, then use the following set of queries to generate a list of available character sets and collations:

```

SELECT RDB$CHARACTER_SET_NAME, RDB$CHARACTER_SET_ID
FROM RDB$CHARACTER_SETS
ORDER BY RDB$CHARACTER_SET_NAME;

SELECT RDB$COLLATION_NAME, RDB$CHARACTER_SET_ID
FROM RDB$COLLATIONS
ORDER BY RDB$COLLATION_NAME;

```

## Specifying Character Sets

---

This section provides details on how to specify character sets. Specifically, it covers how to specify the following:

- The default character set for a database
- A character set for a table column
- The character set for a client attachment
- The collation order for a column
- The collation order in comparisons
- The collation order for ORDER BY and GROUP BY clauses

### Default Character Set for a Database

---

A database's default character set designation specifies the character set the server uses to tag CHAR, VARCHAR, and text Blob columns in the database when no other character set information is provided. When data is stored in such columns without additional character set information, the server uses the tag to determine how to store and transliterate that data. A default character set should always be specified for a database when it is created with CREATE DATABASE.

To specify a default character set, use the DEFAULT CHARACTER SET clause of CREATE DATABASE. For example, the following statement creates a database that uses the ISO8859\_1 character set:

```
CREATE DATABASE 'europe.ib' DEFAULT CHARACTER SET ISO8859_1;
```

#### Important

If you do not specify a character set, the character set defaults to NONE. Using character set NONE means that there is no character set assumption for columns; data is stored and retrieved just as you originally entered it. You can load any character set into a column defined with NONE, but you cannot later move that data into another column that has been defined with a different character set. In this case, no transliteration is performed between the source and destination character sets, and errors may occur during assignment.

For the complete syntax of CREATE DATABASE, see [CREATE DATABASE](#) on page 2-38.

## Character Set for a Column in a Table

---

Character sets for individual columns in a table can be specified as part of the column's CHAR or VARCHAR datatype definition. When a character set is defined at the column level, it overrides the default character set declared for the database. For example, the following **isql** statements create a database with a default character set of ISO8859\_1, then create a table where two column definitions include a different character set specification:

```
CREATE DATABASE 'europe.ib' DEFAULT CHARACTER SET ISO8859_1;

CREATE TABLE RUS_NAME (
 LNAME VARCHAR(30) NOT NULL CHARACTER SET CYRL,
 FNAME VARCHAR(20) NOT NULL CHARACTER SET CYRL,
);
```

For the complete syntax of CREATE TABLE, see [CREATE TABLE](#) on page 2-63.

## Character Set for a Client Attachment

---

When a client application, such as **isql**, connects to a database, it may have its own character set requirements. The server providing database access to the client does not know about these requirements unless the client specifies them. The client application specifies its character set requirement using the SET NAMES statement *before* it connects to the database.

SET NAMES specifies the character set the server should use when translating data from the database to the client application. Similarly, when the client sends data to the database, the server translates the data from the client's character set to the database's default character set (or the character set for an individual column if it differs from the database's default character set).

For example, the following **isql** command specifies that **isql** is using the DOS437 character set. The next command connects to the *europe* database created above, in "Specifying a Character Set for a Column in a Table":

```
SET NAMES DOS437;
CONNECT 'europe.ib' USER 'JAMES' PASSWORD 'U4EEAH';
```

For the complete syntax of SET NAMES, see [SET NAMES](#) on page 2-136. For the complete syntax of CONNECT, see [CONNECT](#) on page 2-34.

## Collation Order for a Column

---

When a CHAR or VARCHAR column is created for a table, either with CREATE TABLE or ALTER TABLE, the collation order for the column can be specified using the COLLATE clause. COLLATE is especially useful for character sets such as ISO8859\_1 or DOS437 that support many different collation orders.

For example, the following **isql** ALTER TABLE statement adds a new column to a table, and specifies both a character set and a collation order:

```
ALTER TABLE 'FR_CA_EMP'
 ADD ADDRESS VARCHAR(40) CHARACTER SET ISO8859_1 NOT NULL
 COLLATE FR_CA;
```

For the complete syntax of ALTER TABLE, see [ALTER TABLE](#) on page 2-18.

## Collation Order in Comparison

---

When CHAR or VARCHAR values are compared in a WHERE clause, it can be necessary to specify a collation order for the comparisons if the values being compared use different collation orders.

To specify the collation order to use for a value during a comparison, include a COLLATE clause after the value. For example, in the following WHERE clause fragment from an embedded application, the value to the left of the comparison operator is forced to be compared using a specific collation:

```
WHERE LNAME COLLATE FR_CA = :lname_search;
```

For the complete syntax of the WHERE clause, see [SELECT](#) on page 2-127.

## Collation Order in ORDER BY

---

When CHAR or VARCHAR columns are ordered in a SELECT statement, it can be necessary to specify a collation order for the ordering, especially if columns used for ordering use different collation orders.

To specify the collation order to use for ordering a column in the ORDER BY clause, include a COLLATE clause after the column name. For example, in the following ORDER BY clause, the collation order for two columns is specified:

```
. . .
ORDER BY LNAME COLLATE FR_CA, FNAME COLLATE FR_CA;
```

For the complete syntax of the ORDER BY clause, see [SELECT](#) on page 2-127.

## Collation Order in a GROUP BY clause

---

When CHAR or VARCHAR columns are grouped in a SELECT statement, it can be necessary to specify a collation order for the grouping, especially if columns used for grouping use different collation orders.

To specify the collation order to use for grouping columns in the GROUP BY clause, include a COLLATE clause after the column name. For example, in the following GROUP BY clause, the collation order for two columns is specified:

```
. . .
GROUP BY LNAME COLLATE FR_CA, FNAME COLLATE FR_CA;
```

For the complete syntax of the GROUP BY clause, see [SELECT](#) on page 2-127.



active set (cursors) 2-120  
 adding  
   See also inserting  
   columns 2-18  
   integrity constraints 2-18  
   secondary files 2-10  
 aggregate functions 2-5  
   AVG() 2-27  
   COUNT() 2-38  
   MAX() 2-118  
   MIN() 2-118  
   SUM() 2-142  
 ALTER DATABASE 2-10  
 ALTER DOMAIN 2-13  
 ALTER EXCEPTION 2-15  
 ALTER INDEX 2-15  
 ALTER PROCEDURE 2-16  
 ALTER TABLE 2-18  
 ALTER TRIGGER 2-24  
 applications  
   preprocessing See gpre  
 arithmetic functions See aggregate functions  
 arrays  
   See also error status array  
   viewing dimension information 6-9  
 assigning values to variables 3-3  
 assignment statements 3-3  
 averages 2-26  
 AVG() 2-26

## B

BASED ON 2-27  
 BEGIN . . . END block  
   defined 3-3, 3-4  
 BEGIN DECLARE SECTION 2-28  
 BLOB cursors  
   closing 2-32  
   declaring 2-82  
   inserting data 2-117  
   opening 2-120  
 BLOB data  
   converting subtypes 2-86  
   inserting 2-83, 2-117  
   selecting 2-83  
   updating 2-144  
 BLOB data type 7-1  
 BLOB filters  
   declaring 2-85  
   dropping 2-95  
   viewing information about 6-16

## C

CACHE option 2-36  
 cache size, changing 2-36  
 case, converting 2-145  
 CAST() 2-30  
 casting 2-30  
 CHAR data type 7-1  
 CHAR datatype  
   description 2-6  
 CHAR VARYING keyword 2-7  
 CHARACTER keyword 2-6  
 CHARACTER SET  
   default 2-40  
   domains 2-44  
   specifying 2-136  
   tables 2-67  
 character sets 7-1 to 7-9  
   additional 7-6  
   default 7-7  
   retrieving 7-6  
   specifying 7-7 to 7-8  
   table of 7-2  
 character strings, converting case 2-145  
 CHARACTER VARYING keyword 2-7  
 CHECK constraints 2-69  
   viewing information about 6-4, 6-52  
 CHECK\_CONSTRAINTS  
   system view 6-52  
 clients See SQL client applications; Windows  
   clients  
 CLOSE 2-31  
 CLOSE (BLOB) 2-32  
 code pages (MS-DOS) 7-5  
 COLLATE clause  
   domains 2-44  
   tables 2-68  
 collation orders 7-1  
   retrieving 7-6  
   specifying 2-68, 7-8  
   viewing information about 6-4  
 columns  
   adding 2-18  
   computed 2-67  
   decrypting 2-21  
   defining 2-42, 2-67  
   domain-based 2-67  
   dropping 2-18  
   encrypting 2-21  
   formatting 6-16

- index characteristics 6-20
- inheritable characteristics 2-44
- local 2-67
- specifying character sets 7-8
- viewing characteristics of 6-10, 6-16, 6-26
- comments in stored procedures and triggers 3-5
- COMMIT 2-33
- compound statements 3-3
- computed columns 2-67
- conditional statements 3-11, 3-21
- conditions, testing 3-11, 3-21
  - See also search conditions
- CONNECT 2-34
- connecting to databases 2-34
- constraints
  - See also integrity constraints
  - adding 2-18, 2-68
  - dropping 2-18
  - types 2-68
  - viewing information about 6-25, 6-53, 6-54
- CONSTRAINTS\_COLUMN\_USAGE system
  - view 6-52
- context variables 3-13 to 3-14
- conversion functions 2-6
  - UPPER() 2-145
- converting
  - case 2-145
  - datatypes 2-30
- COUNT() 2-38
- CREATE DATABASE 2-38
- CREATE DOMAIN 2-42
- CREATE ENCRYPTION 2-45
- CREATE EXCEPTION 2-46
- CREATE GENERATOR 2-47
- CREATE INDEX 2-48
- CREATE JOURNAL 2-49
- CREATE JOURNAL ARCHIVE 2-51
- CREATE PROCEDURE 2-54, 3-1
- CREATE ROLE 2-60
- CREATE SHADOW 2-61
- CREATE TABLE 2-63
- CREATE TRIGGER 2-71, 3-1
- CREATE VIEW 2-78
- creating multi-file databases 2-12
- cursors
  - active set 2-120
  - closing 2-31
  - declaring 2-81
  - opening 2-120
  - retrieving data 2-108

## D

---

data

- inserting 2-115
- retrieving 2-108
- selecting 2-127, 3-16
- sorting 7-1
- storing 7-1
- updating 2-142
- data integrity
  - adding constraints 2-18, 2-68
  - dropping constraints 2-18
- database cache buffers
  - increasing/decreasing 2-36
- database handles
  - declaring 2-134
- database objects
  - viewing relationships among 6-6
- database pages 2-40
  - viewing information about 6-22
- databases
  - altering 2-10
  - connecting to 2-34
  - creating 2-38
  - declaring scope of 2-134
  - decrypting 2-10
  - detaching 2-91
  - dropping 2-92
  - encrypting 2-10
  - multi-file 2-12
  - setting access to in SQL 2-133, 2-137, 2-141
  - shadowing 2-61, 2-99
  - viewing information about 6-54
- datatypes 2-6
  - converting 2-30
  - in table columns 2-67
  - specifying with domains 2-42
- date and time information 2-107
- DATE datatype 2-107
  - description 2-6, 2-7
- dBASE for DOS 7-5
- dBASE for Windows 7-5
- DECIMAL datatype 2-6, 2-7
- DECLARE CURSOR 2-9, 2-31, 2-81, 2-82
- DECLARE CURSOR (BLOB) 2-82
- DECLARE EXTERNAL FUNCTION 2-83
- DECLARE FILTER 2-85
- DECLARE STATEMENT 2-9, 2-31, 2-81, 2-82, 2-86
- DECLARE TABLE 2-9, 2-31, 2-68, 2-81, 2-82, 2-87
- DECLARE VARIABLE 3-5
- declaring
  - database handles 2-134
  - error status array 5-3
  - host-language variables 2-27 to 2-28, 2-101
  - local variables 3-5
  - scope of databases 2-134
  - SQL statements 2-86

- SQLCODE variable 2-29
- tables 2-87
- decrypting a database 2-10
- decrypting columns 2-21
- default character set 7-7
- default transactions 2-140
- defining
  - columns 2-42, 2-67
  - domains 2-43 to 2-44
  - integrity constraints 2-68
- DELETE 2-88, 3-13
  - WHERE clause requirement 2-89
- deleting *See* dropping
- DESCRIBE 2-90
- DISCONNECT 2-91
- domain-based columns 2-67
- domains
  - altering 2-13
  - creating 2-42
  - defining 2-43 to 2-44
  - dropping 2-93
  - inheritable characteristics 2-44
- DOUBLE PRECISION datatype 2-6
- DROP DATABASE 2-92
- DROP DOMAIN 2-93
- DROP ENCRYPTION 2-93
- DROP EXCEPTION 2-94
- DROP EXTERNAL FUNCTION 2-94
- DROP FILTER 2-95
- DROP INDEX 2-96
- DROP JOURNAL 2-97
- DROP JOURNAL ARCHIVE 2-97
- DROP PROCEDURE 2-97
- DROP ROLE 2-98
- DROP SHADOW 2-98, 2-99
- DROP TABLE 2-99
- DROP TRIGGER 2-100
- DROP VIEW 2-100
- dropping
  - columns 2-18
  - integrity constraints 2-18
  - rows 2-88
- DSQL statements
  - declaring table structures 2-87
  - executing 2-103, 2-104
  - preparing 2-121

## E

---

- either\_case switch 2-122
- ENCRYPT ON ENCRYPTION
  - grant permission to 2-113
  - revoking permission to 2-125
- encrypting a database 2-10

- encrypting columns
  - decrypting columns 2-21
- encryption keys
  - about 2-45
  - creating 2-45
  - deleting, dropping 2-93
- END DECLARE SECTION 2-101
- error codes 5-1 to 5-38
- error status array 5-3
  - declaring 5-3
  - defined 5-2
  - error codes 5-20 to 5-36
  - SQLCODE variable
    - error codes and messages 5-6 to 5-20
- error-handling routines 2-9 to ??, 5-2 to ??
  - options 5-3
  - stored procedures 3-18
  - triggers 3-18
- errors
  - run-time 5-1
  - trapping 2-145, 3-19, 5-2
  - user-defined *See* exceptions
- EVENT INIT 2-101
- EVENT WAIT 2-102
- events
  - See also* triggers
  - posting 3-15
  - registering interest in 2-101
- exact numerics 2-7
- EXCEPTION 3-6
- exceptions 2-46
  - altering 2-15
  - creating 2-46
  - defined 3-6
  - dropping 2-94
  - viewing information about 6-9
- EXECUTE 2-103
- EXECUTE IMMEDIATE 2-104
- EXECUTE PROCEDURE 2-105, 3-7
- expression-based columns *See* computed columns
- EXTERNAL FILE option 2-68
- EXTRACT() 2-107

## F

---

- FETCH 2-108
- FETCH (BLOB) 2-109
- files
  - secondary 2-10, 6-15
  - shadow 6-15
- FLOAT datatype 2-7
- FOR SELECT . . . DO 3-10
- FOREIGN KEY constraints 2-69
  - viewing information about 6-52

- formatting
  - columns 6-16
- FROM 2-131
- functions 2-5
  - aggregate 2-5
  - arguments 6-17
  - conversion 2-6, 2-145
  - numeric 2-6, 2-111
  - user-defined *See* UDFs

## G

---

- GEN\_ID() 2-110
- generators
  - creating 2-47
  - initializing 2-135
  - resetting, caution 2-136
  - returning 2-111
  - viewing information about 6-19
- gpre 2-101
  - declaring SQLCODE automatically 2-29
  - either\_case switch 2-122
  - error status array processing 5-3
  - manual switch 2-92, 2-140
- gpre directives
  - BASED ON 2-28
  - BEGIN DECLARE SECTION 2-29
  - DECLARE TABLE 2-87
  - END DECLARE SECTION 2-101
- GRANT 2-111
  - decrypt permission 2-113
  - encryption permission 2-113
- GROUP BY 2-131, 7-1, 7-7, 7-9

## H

---

- HAVING 2-131
- host-language variables
  - declaring 2-27 to 2-28, 2-101

## I

---

- I/O *See* input, output
- IF ... THEN ... ELSE 3-11
- indexes
  - activating/deactivating 2-15
  - altering 2-15
  - columns comprising 6-20
  - creating 2-48
  - dropping 2-96
  - recomputing selectivity 2-138
  - viewing structures of 6-20
- indicator variables 2-106
- initializing

- generators 2-135
- input parameters 2-55
  - defined 3-12
- input statements 2-91
- INSERT 2-115, 3-12
- INSERT CURSOR (BLOB) 2-117
- inserting
  - See also* adding
  - new rows 2-115
- INTEGER datatype 2-7
- integrity constraints
  - See also* specific type
  - adding 2-18, 2-68
  - cascading 2-22, 2-23, 2-67, 2-69, 2-70
  - dropping 2-18
  - types 2-68
  - viewing information about 6-25, 6-53, 6-54
- Interactive SQL *See* isql
- international character sets 7-1 to 7-9
  - additional 7-6
  - default 7-7
  - specifying 7-7 to 7-8
- INTO 2-131
- isc\_convert\_error 5-4
- isc\_deadlock 5-4
- isc\_integ\_fail 5-4
- isc\_lock\_conflict 5-4
- isc\_no\_dup 5-4
- isc\_not\_valid 5-4
- isc\_print\_sqlerror() 5-3
- isc\_sql\_interprete() 5-3
- isc\_status 5-3
- ISOLATION LEVEL 2-140

## J

---

- journal archiving
  - activating 2-51
- journal archiving, dropping 2-97
- journal files
  - activating 2-49
  - deleting 2-97
- journaling
  - activating 2-49
  - dropping 2-97

## K

---

- key constraints *See* FOREIGN KEY constraints;  
PRIMARY KEY constraints
- keys
  - defined 2-69
- keywords 4-1 to 4-4

## L

---

- large exact numerics 2-7
- local columns 2-67
- local variables
  - assigning values 3-3
  - declaring 3-5
- loops *See* repetitive statements
- lowercase, converting from 2-145

## M

---

- manual switch 2-92, 2-140
- MAX() 2-117
- maximum values 2-117
- messages 5-1 to 5-38
- metadata 6-1
- MIN() 2-118
- minimum values 2-118
- modifying *See* altering;updating
- MS-DOS code pages 7-5
- multi-file databases
  - creating 2-12
- multiple transactions
  - running 2-140

## N

---

- naming conventions
  - keywords and 4-1
- nested stored procedures 3-7
- NEW context variables 3-12 to 3-13
- NO RECORD\_VERSION 2-140
- NO WAIT 2-140
- nomenclature
  - stored procedures and triggers 3-2
- numbers
  - averaging 2-26
  - calculating totals 2-142
- NUMERIC datatype 2-7
- numeric function 2-6, 2-111
- numeric values *See* values

## O

---

- object delimiter 2-3
- OLD context variables 3-13
- OPEN 2-119
- OPEN (BLOB) 2-120
- ORDER BY 2-48, 2-79, 2-89, 2-130, 2-131, 2-144, 7-1, 7-7, 7-9
- output
  - error messages 5-3
- output parameters 2-55
  - defined 3-14

- output statements 2-91

## P

---

- Paradox for DOS 7-5
- Paradox for Windows 7-5, 7-6
- parameters
  - DSQL statements 2-90
  - input 2-55, 3-12
  - output 2-55, 3-14
  - stored procedures 6-23
- PLAN 2-131
- plan\_expr 2-128
- plan\_item 2-128
- plan, specifying 2-130, 2-132
- POST\_EVENT 3-15
- posting events 3-15
- PREPARE 2-121
- preprocessor *See* gpre
- primary files 2-39
- PRIMARY KEY constraints 2-47, 2-68
  - viewing information about 6-52
- privileges *See* security
- procedures *See* stored procedures

## R

---

- RDB\$CHARACTER\_SETS 6-3
- RDB\$CHECK\_CONSTRAINTS 6-4
- RDB\$COLLATIONS 6-4
- RDB\$DATABASE 6-5
- RDB\$DEPENDENCIES 6-6
- RDB\$EXCEPTIONS 6-9
- RDB\$FIELD\_DIMENSIONS 6-9
- RDB\$FIELDS 6-10
- RDB\$FILES 6-15
- RDB\$FILTERS 6-16
- RDB\$FORMATS 6-16
- RDB\$FUNCTION\_ARGUMENTS 6-17
- RDB\$FUNCTIONS 6-18
- RDB\$GENERATORS 6-19
- RDB\$INDEX\_SEGMENTS 6-20
- RDB\$INDICES 6-20
- RDB\$LOG\_FILES 6-22
- RDB\$PAGES 6-22
- RDB\$PROCEDURE\_PARAMETERS 6-23
- RDB\$PROCEDURES 6-23
- RDB\$REF\_CONSTRAINTS 6-25
- RDB\$RELATION\_CONSTRAINTS 6-25
- RDB\$RELATION\_FIELDS 6-26
- RDB\$RELATIONS 6-28
- RDB\$SECURITY\_CLASSES 6-31
- RDB\$TRANSACTIONS 6-32
- RDB\$TRIGGER\_MESSAGES 6-32
- RDB\$TRIGGERS 6-33

- RDB\$TYPES 6-34
- RDB\$USER\_PRIVILEGES 6-35, 6-36
- RDB\$VIEW\_RELATIONS 6-37
- READ COMMITTED 2-140
- read-only transactions
  - committing 2-34
- read-only views 2-80
- RECORD\_VERSION 2-140
- recursive stored procedures 3-7
- REFERENCES constraint 2-69
- referential integrity *See* integrity constraints
- REFERENTIAL\_CONSTRAINTS system view 6-53
- RELEASE argument 2-34
- RELEASE SAVEPOINT 2-123
- repetitive statements 3-10, 3-21
- repetitive tasks 3-7
- reserved words *See* keywords
- RESERVING clause 2-140
- retrieving data 2-108
- REVOKE 2-123
  - decrypt permission 2-125
  - encryption permission 2-125
- roles
  - creating 2-60
  - dropping 2-98
  - granting 2-111
  - revoking 2-123
  - system table 6-30
- ROLLBACK 2-126
- rows
  - deleting 2-88
  - inserting 2-115
  - selecting 2-108
    - stored procedures and triggers 3-16
  - sequentially accessing 2-108
  - updating 2-142
- ROWS clause 2-127
- run-time errors 5-1

## S

---

- savepoints
  - creating and using 2-127
  - releasing 2-123
  - roll back to 2-126
- search conditions (queries)
  - comparing values 3-16
  - evaluating 2-120
- secondary files 2-39
  - adding 2-10
  - viewing information about 6-15
- secondary storage devices 2-62
- security
  - access privileges 2-112
  - granting 2-111
  - revoking 2-123
  - viewing 6-35
  - viewing access control lists 6-31
- SELECT 2-127, 2-131, 3-16
  - statements 2-131
- selecting
  - data 2-127 to 2-131
  - stored procedures and triggers 3-16
- SET DATABASE 2-133, 2-137, 2-141
- SET GENERATOR 2-135
- SET NAMES 2-136, 7-8
- SET STATISTICS 2-138
- SET TRANSACTION 2-139
- shadow files
  - sets 2-62
  - viewing information about 6-15
- shadows
  - creating 2-61
  - dropping 2-99
- SMALLINT datatype 2-7
- SNAPSHOT TABLE STABILITY 2-140
- sorting
  - data 7-1
- specifying
  - collation orders 2-68, 7-8
- SQL clients
  - specifying character sets 7-8
- SQL dialects 2-2
- SQL statements 2-4
  - declaring 2-86
  - executing 2-9
- SQLCODE variable 2-9, 5-2 to 5-3
  - declaring automatically 2-29
  - error codes and messages 5-6 to 5-20
  - return values 2-9
- statements 3-3
  - See also* DSQL statements; SQL statements
  - assignment 3-3
  - compound 3-3
  - conditional 3-11, 3-21
  - executing 2-121
  - input/output 2-91
  - repetitive 3-10, 3-21
  - SELECT 2-131
  - SQLCODE and 2-9
- status array *See* error status array
- storage devices
  - secondary 2-62
- stored procedures
  - adding comments 3-5
  - altering 2-16
  - assigning values 3-3
  - creating 2-54, 3-1

- dropping 2-97
- error handling 3-18
- executing 2-105, 3-7
- indicator variables 2-106
- nested 3-7
- passing values to 3-12
- posting events 3-15
- powerful SQL extensions 3-1
- recursive 3-7
- terminating 3-19
- viewing information about 6-9, 6-23
- stored procedures and triggers 3-2
- storing data 7-1
- strings *See* character strings
- SUM() 2-142
- SUSPEND 3-17
- system tables 6-1 to 6-37
- system views 6-1, 6-51 to 6-52

## T

---

- TABLE\_CONSTRAINTS system view 6-54
- tables
  - altering 2-18
  - creating 2-63
  - declaring 2-87
  - dropping 2-99
  - inserting rows 2-115
  - viewing information about 6-25, 6-28, 6-54
- tasks, repetitive 3-7
- text 7-1
- TIME datatype 2-107
- TIMESTAMP datatype 2-107
- totals, calculating 2-142
- transaction names 2-139
- transactions
  - committing 2-33
  - default 2-140
  - multiple databases 6-32
  - read-only 2-34
  - rolling back 2-126
  - running multiple 2-104, 2-105, 2-140
  - starting 2-139
- transition features 2-3
- trapping
  - errors 2-145, 3-19, 5-2
  - warnings 2-145, 5-2
- trigger language 3-2
- triggers 3-3
  - altering 2-24
  - creating 2-71, 3-1
  - dropping 2-100
  - error handling 3-18

- message information 6-32
- NEW values 3-12 to 3-13
- OLD values 3-13
- posting events 3-15
- viewing information about 6-9, 6-33

## U

---

- UDFs 6-18
  - declaring 2-83
  - dropping 2-94
- UNION 2-131
- UNION operator 3-16
- UNIQUE constraints
  - viewing information about 6-52
- UNIQUE keys 2-69
- UPDATE 2-142, 3-12, 3-13
- updating
  - BLOB data 2-144
  - rows 2-142
- UPPER() 2-145
- uppercase, converting to 2-145
- USER name 2-66, 2-69
- user-defined errors *See* exceptions
- user-defined functions *See* UDFs
- USING clause 2-141

## V

---

- values
  - See also* NULL values
  - assigning to variables 3-3
  - averages 2-26
  - changing 3-12
  - maximum 2-117
  - minimum 2-118
  - passing to stored procedures 3-12
  - returning 3-14, 3-17
    - to SQLCODE variable 2-9
  - totals 2-142
- VARCHAR data type 7-1
- VARCHAR datatype 2-7
- variables
  - context 3-13 to 3-14
  - host-language 2-27 to 2-28, 2-101
  - indicator 2-106
  - local 3-3, 3-5
- views
  - creating 2-78
  - dropping 2-100
  - read-only 2-80
  - updatable 2-80
  - viewing characteristics of 6-28

## W

---

WAIT 2-140

warnings

*See also* errors

    trapping 2-145, 5-2

WHEN 2-46

WHEN . . . DO 3-18

WHENEVER 2-145, 5-2

WHERE 2-131

WHERE clause *See* SELECT

WHILE . . . DO 3-21

Windows applications

    character sets 7-6

Windows clients

    specifying character sets 7-8