Release Notes

InterBase 7.0
October 2002

VERSION 7

Borland®
InterBase®

Borland Software Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249
www.borland.com

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 2002 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Other product names are trademarks or registered trademarks of their respective holders.

Contents

Chapter 1
General Information

Documentation 1-1
The InterBase document set 1-1
Acrobat Reader 1-2
Contacting Borland 1-3
Migration 1-3
External tables 1-3
Migrating from older databases 1-3

Chapter 2
InterBase 7.0 features

Contents of this document 2-1
New features in InterBase 7 2-1
Issues in FrameMaker 7 2-2
Database naming 2-2
ODS11 2-2
New name for the security database 2-2
New name for Unix configuration file 2-2
New keywords 2-3
New datatype: BOOLEAN 2-3
No more SET TERM 2-3
68-byte metadata names and XSQLDA 2-4
New APIs for blobs and arrays 2-4
Client version detection 2-5
New type 4 InterClient 2-6
Upgrading clients 2-6
Legacy InterServer 2-6
SMP support 2-6
IBConfig parameter: MAX_THREADS 2-6
Expanded processor control: CPU_AFFINITY . 2-6
Increased security for external tables 2-7
New HTML reference docs 2-7
IBConsolePlus preview 2-8
Monitoring with system temporary tables 2-8
Querying system temporary tables 2-9
Refreshing the temporary tables 2-9
Listing the temporary tables 2-9
Security 2-9
Examples 2-9
Updating system temporary tables 2-10
Making global changes 2-10
System temporary table metadata 2-11
System temporary tables 2-11
TMP$ATTACHMENTS 2-11
TMP$DATABASE 2-12
TMP$POOL_BLOCKS 2-14
TMP$POOLS 2-16
TMP$PROCEDURES 2-16
TMP$RELATIONS 2-17
TMP$STATEMENTS 2-18
TMP$TRANSACTIONS 2-19
Thread-safe processing of database handles . . 2-19
Issues 2-20
License files from previous versions 2-20
Removing license certificates 2-20
The *Getting Started* book 2-20

Chapter 3
InterBase 6.5 features

Metadata security 3-1
Scripts for changing database security 3-2
Requirements and constraints 3-2
Migration issues 3-3
Performance 3-3
64 bit I/O 3-3
Asynchronous cancel 3-3
isc_dsql_free_statement() 3-3
.DSQL_close 3-4
.DSQL_drop 3-4
.DSQL_cancel 3-4
New *isc_config*, *ibconfig* file parameters 3-5
CPU_AFFINITY 3-5
SWEEP_QUANTUM 3-5
USER_QUANTUM 3-5
SWEEP_YIELD_TIME 3-5
Language enhancements 3-5
New keywords 3-5
The ROWS clause for SQL data manipulation statements 3-6
Support for ROWS clause in gpre 3-7
XML Generation 3-7
JDBC (InterClient 2.5) 3-10
Removing Deprecated JRE API calls from InterClient 3-10
New JDBC methods 3-10

Chapter

1

General Information

This chapter covers the following topics:

- Documentation
- Acrobat Reader
- Contacting Borland
- Migration

Features Please see Chapter 2 of these *Release Notes* for a detailed listing of InterBase 7 features. Chapter 3 contains a description of features added in InterBase 6.5.

Migration For instruction on migrating from InterBase 6.x, see "Migration" on page 1-3 of these *Release Notes*. If you are migrating a database from InterBase 5.6 or older to InterBase 6, 6.5, or 7, please read the "migration" appendix in the *Operations Guide*.

Installation For information about installation, please see the *install.txt* file for your platform. This file is located as follows on your InterBase CD or CD image file:

- Windows: *win32* directory
- Solaris: *solaris* directory
- Linux: *linux* directory

Documentation

This section describes how to install the InterBase document set and Acrobat Reader With Search.

The InterBase document set

The document set in PDF format requires about 13MB of space on your disk. You can install them following the instructions below. You can also install them at any later time, access them from the CD-ROM, or copy them from the CD-ROM.

The books are available for purchase in printed form from http://shop.borland.com.

The documentation set for InterBase 7.0 in PDF format does not install by default. To install the InterBase documentation on your hard drive, follow these steps:

Windows

During the install, you see a "Select Components" screen. Under the "Main Components" entry, click the red "X" next to "Online documentation" and select "Install this component" from the dropdown list. If

Linux and Solaris

The Install Menu provides an option for installing the InterBase document set. After you have performed any one of the other install options, such as InterBase server or client, the Install Menu redisplays and you can choose to install the document set.

What's in the doc set

The InterBase document set consists of the following:

Document Title	Contents
Release Notes	Contact information, new feature descriptions, known problems and workarounds
Operations Guide	Configuring and operating InterBase Server, using command-line and GUI tools
Data Definition Guide	Designing and building InterBase databases, using database utilities
Developer's Guide	Developing InterBase database applications using Borland development tools
Embedded SQL Guide	Developing InterBase database applications using embedded SQL
API Guide	Developing InterBase database applications using the InterBase Application Programming Interface
Language Reference	InterBase SQL, DSQL, isql, and stored procedure and trigger language elements, syntax and semantics
SQL Reference (SqlRef.html)	The complete SQL reference from the "SQL Statement and Function Reference" chapter of the *Language Reference*.
API Function Reference (APIFunctionRef.html)	The complete API function reference from the *API Guide*.

Acrobat Reader

You need Adobe Acrobat Reader 4 or 5 to view the *.pdf files. To take advantage of the cross-document search feature (called Full Text Search), you need the "With Search" flavor of Acrobat Reader. If you do not already have this application, the English version is available on the InterBase CD-ROM. Look in the Adobe directory under the directory for your platform. Acrobat Reader With Search is also available for free in many languages from *http://www.adobe.com/products/acrobat/readstep2.html*.

The Linux and Solaris installations provide an option for installing Acrobat Reader in the Install Menu. Windows users must navigate to \win32\Adobe on the CD-ROM and run *rp505enu.exe*.

Full Text Search The document set has Full Text Search enabled, so if you have Acrobat Reader With Search, you can use the Search button to perform cross-volume searches. (The Find button searches only a single document.) For more information about using Full Text Search, see Chapter One of the *Operations Guide*.

Links The InterBase PDF documentation set contains many hypertext links that take you to referenced points in the document with a single click. In addition, the Table of Contents and Index entries are hypertext links and therefore are clickable. Throughout the document set, clickable links appear **bold and blue**.

Contacting Borland

Mailing address

Borland Software Corporation
100 Enterprise Way
Scotts Valley, CA 95066-3249

Phone: 1-831-431-1000

Websites and email

- Borland maintains an Internet site on the world-wide web for general InterBase information. The URL of this site is: **http://www.borland.com/interbase**
- Technical information, such as white papers and FAQs, can be found at: **http://bdn.borland.com/interbase**
- For installation and presales questions, visit: **http://shop.borland.com**
- To discuss issues, with other InterBase users, visit: **http://info.borland.com/newsgroups**
- For information about how to contact InterBase representatives outside the U.S. and Canada, look at the following web page: **http://www.borland.com/bww/**
- For issues pertaining to content and presentation on our web site, send email to: **webmaster@borland.com**

Migration

Migrating databases from InterBase 6 or 6.5 to InterBase 7 is a simple process:

1. Before installing InterBase 7, back up each database with **gbak**.
2. Install InterBase 7.
3. Restore each database using the InterBase 7 **gbak**.

External tables

The only additional step required when migrating from InterBase 6.x occurs when you have external tables. For security reasons, InterBase 7 requires that one of the following two things be true for external tables:

- The tables must be in *<ib_home>/ext*.

  Or

- Each directory that contains an external database file must be listed in ibconfig using the EXTERNAL_FILE_DIRECTORY entry:

```
EXTERNAL_FILE_DIRECTORY directory1

EXTERNAL_FILE_DIRECTORY C:\data\mydata\
```

Migrating from older databases

Migrating databases from InterBase versions 5.x and older to InterBase 6 or later requires planning. Please read the Migration appendix of the InterBase *Operations Guide* for details on migrating clients and databases from InterBase 5.x to InterBase 7.

Chapter

2

InterBase 7.0 features

This chapter describes the new features available in InterBase 7 and lists minor issues.

If you are reading this document in PDF format, **bold blue** type indicates clickable links.

Contents of this document

This chapter discusses the features and issues for InterBase 7.0.If you are viewing this document in PDF format, you can click an item in the "features" or "issues" lists to jump directly to that topic.

New features in InterBase 7

This chapter includes information about the following new issues and features:

- Database naming
- ODS11
- New name for the security database
- New name for Unix configuration file
- New keywords
- New datatype: BOOLEAN
- No more SET TERM
- 68-byte metadata names and XSQLDA
- New APIs for blobs and arrays
- Client version detection
- New type 4 InterClient
- SMP support
- Increased security for external tables
- New HTML reference docs
- Monitoring with system temporary tables
- IBConsolePlus preview

Issues in FrameMaker 7

FrameMaker 7.0 has been extensively tested. We believe that only the following minor issues remain.

- License files from previous versions
- Removing license certificates
- The Getting Started book

Database naming

InterBase no longer recommends using ".gdb" as the extension for database files, since on Windows ME and Windows XP, any file that has this extension is automatically backed up by the System Restore facility. On those two platforms, using the .gdb extension for database names means that the database gets backed up every time it is touched, which can result in a significant detriment to performance. Linux and Solaris are not affected. InterBase now recommends using ".ib" as the extension for database names.

Important Our security database, formerly named *isc4.gdb* is now named *admin.ib*. For more information, see New name for the security database, below.

For the present, the InterBase example databases still have the ".gdb" name. In the future we will phase out that name and use new names.

ODS11

InterBase 7 introduces ODS11. This new On-Disk Structure is required by the presence of the new BOOLEAN datatype and 68-byte metadata names.

To upgrade your databases, back them up with an ODS10 **gbak** and then restore them with the ODS11 **gbak** that comes with the InterBase 7.

New name for the security database

In InterBase 7, InterBase's security database is named *admin.ib* on all platforms. InterBase's internal tools have all been updated to use this name. This addresses a problem that arises on some versions of Windows ME and XP where files having a *gdb* extension get backed up whenever they are touched, seriously degrading database performance.

If you wish to continue using your existing security database, you must back it up and restore it using the latest **gbak**. To change the name, specify *admin.ib* as the new name during the restore. If you have existing clients that expect to find *isc4.gdb*, you must update them to use the new name.

You can specify a name of your choice for the security database by setting the ADMIN_DB parameter in the InterBase configuration file, *ibconfig*. For example,

```
ADMIN_DB myname.ib
```

or

```
ADMIN_DB myname.gdb
```

New name for Unix configuration file

On Linux and Solaris platforms the InterBase configuration file, which was previously called *isc_config*, is now called *ibconfig*.

New keywords

InterBase 7.0 adds the following new keywords:

BOOLEAN TRUE FALSE UNKNOWN

The following keywords were added to InterBase 6.5:

ROWS TIES PERCENT

The following words are likely to be keywords in the future:

PRESERVE GLOBAL TEMPORARY

New datatype: BOOLEAN

InterBase now supports a BOOLEAN datatype, implemented to the SQL 99 standard.

Name	Size	Range
BOOLEAN	32 bits	TRUE / FALSE / UNKNOWN

Examples:

```
CREATE TABLE AWARDS_1 (isEligible BOOLEAN, name VARCHAR(20));
INSERT INTO AWARDS_1 VALUES(TRUE, 'Jim Smith');
INSERT INTO AWARDS_1 VALUES(FALSE, 'John Butler');

SELECT * FROM AWARDS_1 WHERE isEligible = TRUE;
```

ISQL and IBConsole return TRUE, FALSE, and UNKNOWN. Queries created with APIs return 1, 0, and NULL, respectively.

For ESQL and DSQL programmers, we define the following type in *ibase.h*:

```
define SQL_BOOLEAN 590
```

To use this new datatype, your database must be ODS 11, meaning that older databases must have been restored with an ODS 11 **gbak**. The dialect of the database does not matter, as long as it is ODS 11.

No more SET TERM

When you write SQL, there is no longer any need to use SET TERM to define a temporary terminator when defining stored procedures and triggers. InterBase now parses these statements correctly without the use of SET TERM. For example, the following statement works without any problem:

```
CREATE PROCEDURE foo ...
     BEGIN
        some statement;
        another statement;
        yet another;
     END;
```

Important The document set states that IBConsole and IBX still require the use of SET TERM. InterBase believes that this was corrected after the document set was written and that SET TERM is no longer required in **isql**, IBConsole, or IBX.

The old SET TERM functionality remains available in **isql**, IBConsole, and IBX, so that old scripts can still function.

68-byte metadata names and XSQLDA

Metadata names can now be 68 bytes long (67 bytes plus a null terminator). These names are available through all InterBase clients and are implemented in the new type 4 InterClient. They are being implemented in DBX and IBX and may be available by the time you read this.

The XSQLDA structure has been updated to support these long metadata names. Set the version field of this structure to SQLDA_CURRENT_VERSION to access long metadata names.

New APIs for blobs and arrays

Ten API calls that relate to blobs and arrays have been updated to support these longer metadata names. In these new APIs, the *desc* field points to an updated descriptor structure that accommodates long metadata names.

The new API calls are:

isc_array_gen_sdl2()	*isc_array_get_slice2*()
isc_array_lookup_bounds2()	*isc_array_lookup_desc2*()
isc_array_set_desc2()	*isc_array_put_slice2*()
isc_blob_default_desc2()	*isc_blob_gen_bpb2*()
isc_blob_lookup_desc2()	*isc_blob_set_desc2*()

The associated structure for arrays is ISC_ARRAY_DESC_V2:

```
typedef struct {
   short array_desc_version;
   unsigned char array_desc_dtype;
   char array_desc_scale;
   unsigned short array_desc_length;
   char array_desc_field_name [METADATALENGTH];
   char array_desc_relation_name [METADATALENGTH];
   short array_desc_dimensions;
   short array_desc_flags;
   ISC_ARRAY_BOUND array_desc_bounds [16];
} ISC_ARRAY_DESC_V2;
```

For blobs, the new structure is ISC_BLOB_DESC_V2:

```
typedef struct {
   short blob_desc_version                              /* version of descriptor */
   short blob_desc_subtype;                             /* type of Blob data */
   short blob_desc_charset;                             /* character set */
   short blob_desc_segment_size;                        /* segment size */
   unsigned char blob_desc_field_name [METADATALENGTH];   /* Blob column name */
   unsigned char blob_desc_relation_name [METADATALENGTH];   /* table name */
} ISC_BLOB_DESC_V2;
```

The associated defines are:

```
#define BLB_DESC_VERSION2 2
```

```
#define BLB_DESC_CURRENT_VERSION BLB_DESC_VERSION2

#define ARR_DESC_VERSION2 2

#define ARR_DESC_CURRENT_VERSION ARR_DESC_VERSION2
```

These new API calls and their structs are documented in the *API Guide*. See Chapter 7, Chapter 8, and the new API calls in the API Reference chapter.

Client version detection

Some clients—notably drivers, but others as well—need to query the InterBase client library for the version numbers. Three new APIs provide this capability: *isc_get_client_version*(), *isc_get_client_major_version*(), and *isc_get_client_minor_version*().

isc_get_client_version()
Syntax:

```
void isc_get_client_version (char *version)
```

isc_get_client_version() returns the version string, typically in the following format:

```
XX-dM.N.n.b
```

where

`XX` is the all-caps two-character hardware/software platform code:

WI = Windows/Intel

SO = Solaris/Sparc

LI = Linux/Intel

`d` is a distribution indicator

B = beta

T = test

V = verified

I = internal

`M` is the major version number

`N` is the minor version number

`n` is a minor-minor version number

`b` is a regular build number

You should pass a character buffer that is a least 20 characters long to this function. It does not perform the checks if the buffer is too short.

isc_get_client_major_version()
Syntax:

```
int isc_get_client_major_version ()
```

isc_get_client_major_version() returns the major version number of the client library.

isc_get_client_minor_version()
Syntax:

```
int isc_get_client_minor_version ()
```

isc_get_client_major_version() returns the minor version number of the client library.

New type 4 InterClient

InterBase 7.0 introduces InterClient 3.0, This new version of InterClient is a type 4 JDBC driver, which means that it can communicate directly with the InterBase server. InterServer is no longer needed in environments where all the clients have been upgraded to this new type 4 InterClient.

Upgrading clients

Place the new *interclient.jar* file on each client machine and ensure that it is the first instance on the CLASSPATH. Note that although the filename is the same as it was in earlier versions, the file being distributed with InterBase 7 is very different. It is a type 4 JDBC driver. Earlier versions were type 3. Be sure that you are installing the latest driver on your client machines.

Legacy InterServer

If you are not able to upgrade all of your clients immediately, install InterServer on the InterBase Server platform. InterServer will do no harm, and its presence allows a mixture of type 3 and type 4 clients to attach to the InterBase server.

The *interserver.exe* file that distributes with InterBase 7.0 has not changed from previous versions.

SMP support

InterBase now provides symmetric multiprocessor (SMP) support for both clients and servers. Previous versions of InterBase ran on SMP systems safely by allowing only a single processor at a time to execute within the InterBase components. This release exploits SMP hardware by running InterBase threads on all processors simultaneously for increased throughput and performance.

Important When you purchase a single server license, you acquire the right to use a single processor. You must purchase an additional license for each additional processor that you wish to use.

IBConfig parameter: MAX_THREADS

Setting the MAX_THREADS parameter in *ibconfig* controls the maximum number of threads that can be active at one time within the InterBase engine. The default setting is 100:

```
MAX_THREADS   100
```

The ideal setting for this number depends partly on the nature of the work being performed by your clients. If you have many clients performing very similar tasks, you may want to lower the MAX_THREADS setting to reduce contention. On the other hand, if simultaneous activity is highly diverse, setting this to a higher value may increase throughput.

Note that this setting does not affect the maximum possible threads that can be created by the InterBase server but only the number that can be active in the engine at one time. This is a Version 7.0 feature. In pre-7.0 versions the engine was single threaded, but in 7.0 the engine is SMP enabled hence multiple threads can safely run in the engine simultaneously. This configuration parameter allows the user to control/fine tune the number of active threads in the engine.

Expanded processor control: CPU_AFFINITY

On Windows multiprocessor platforms, you can specify which processors InterBase should use by adding the CPU_AFFINITY parameter to the *ibconfig* file. This setting is useful whenever the number of licensed processors is less than the number of actual processors present.

Note that when you purchase a single server license, you acquire the right to use a single processor. You must purchase one additional license for each additional processor that you wish to use.

The CPU_AFFINITY parameter populates a bit vector in which each bit represents a processor on the system on which the threads are allowed to run. The maximum number of processors is platform dependent. To specify which processors should be used, uncomment CPU_AFFINITY in the *ibconfig* file and give it a decimal value that corresponds to the binary value that results from setting a bit for each desired machine. For example, to use processors 2 and 3, set CPU_AFFINITY to 6:

```
CPU_AFFINITY6
```

The table below shows examples of how to set this value.

To use these processors	CPU_AFFINITY value	Array setting
1	1	0 0 1
2	2	0 1 0
1 and 2	3	0 1 1
3	4	1 0 0
2 and 3	6	1 1 0
1, 2, and 3	7	1 1 1

Historical note: In InterBase 6.5, this parameter was used to specify which processor the single active thread should run on. Now that SMP is available in InterBase 7, the functionality has been extended to include specifying multiple processors for multiple threads.

Increased security for external tables

Under some conditions, external tables could pose a security hazard. To counter this, InterBase has added the new requirements for external tables. External tables must meet one of the following conditions:

- The table is located in *<ib_home>/ext*. InterBase can always find external files that you place here.
- The table is not located in *<ib_home>/ext*. If you want to place external table files elsewhere, you must specify the location in the *ibconfig* configuration file by setting the EXTERNAL_FILE_DIRECTORY parameter to the location of the external file. Create a new table entry for each location.

Important For security reasons, it is extremely important that you not place files with sensitive content in the same directory as external tables.

New HTML reference docs

InterBase now includes two references in HTML form. Both of these can be accessed from the Help menu of IBConsole or directly from the *<ib_home>/HtmlRef* directory.

- **The SQL Reference** The *SqlRef.html* file replaces the older *SqlRef.hlp* file. It contains all the SQL statement information from the "SQL Statement and Function Reference" chapter of the *Language Reference*.
- **The API Function Reference** The *APIFunctionRef.html* file is an HTML version of the "API Function Reference" chapter of the *API Guide*.

IBConsolePlus preview

IBConsolePlus is a multithreaded and improved version of IBConsole currently under development at Borland. It is being offered as a public beta in association with InterBase 7.0.

IBConsolePlus can be downloaded from *https://ww5.borland.com/publicbeta/ibcplus/*. You will always find the latest version there. To download it, give *interbase* as the user name and *ibc+* as the password.

Please exercise the features in IBConsolePlus and let us know what you think of this new interface. With your feedback we will continue to enhance IBC+ and will incorporate the new console in a future release of InterBase. This is your chance to make a difference! We appreciate your help.

We have set up a public news group for your feedback:
snews://fora.borland.com/borland.private.publicbeta.ibconsoleplus

The user name for this newsgroup is *interbase* and the password is *ibc+*.

For this release of InterBase, the original IBConsole remains as the standard interface.

Monitoring with system temporary tables

The InterBase Server has always kept a lot of statistics about what was going on, but it has not been easy, or in some cases possible, to surface that information. Now, InterBase captures that information and makes it available in a set of global system temporary tables. These tables describe the runtime behavior of a database. They also provide a level of control.

Although it has always been possible to see a list of users who were currently attached to a database, you can now find out much more. For example, you can see how long each user has been connected, what application each user is running, or the total amount of data I/O used by each attachment. A glance at the temporary table metadata listed on pages 7-31 to 7-42 of the *Language Reference* will suggest the vast possibilities that are available here.

It is also possible to exercise a certain amount of control over the state of a database by performing updates to these tables. See "**Updating system temporary tables**" on page 2-10.

These system temporary tables are specific to each database attachment and are visible only to the sysdba user and the database owner. There is therefore no need for unique names and no danger of collisions by separate attachments. Each table is populated only at the point when a client queries it.

The following system temporary tables are available. Their structure is documented in the *Language Reference* on pages 7-31 to 7-42.

Table 2.1 InterBase temporary system tables

Table name	Description
TMP$ATTACHMENTS	One row for each connection to a database
TMP$DATABASE	One row for each database you are attached to
TMP$POOL_BLOCKS	One row for each block of memory in each pool
TMP$POOLS	One row for each current memory pool
TMP$PROCEDURES	One row for each procedure executed since the current connection began

Table 2.1 InterBase temporary system tables

Table name	Description
TMP$RELATIONS	One row for each relation referenced since the current connection began
TMP$STATEMENTS	One row for each statement currently executing for any current connection
TMP$TRANSACTIONS	One row for each transaction that is active or in limbo

Querying system temporary tables

Clients can query these tables using SELECT statements, just as they would query any other table. By querying these tables, a rich collection of data about server performance and user behavior is available.

You cannot create or redefine temporary tables yourself.

Tip For frequent monitoring, the best transaction control is to start the transaction as READ_COMMITTED, READ_ONLY. Then commit it with COMMIT_RETAINING. This has the least impact on the system.

Refreshing the temporary tables

To refresh the rows in the temporary tables, commit your transaction and perform the SELECT from the temporary tables again. InterBase automatically deletes the rows stored in temporary tables on a commit.

Listing the temporary tables

To display a list of these temporary tables, issue the following command in **isql**:

```
SHOW SYSTEM
```

The temporary tables are listed at the end of the system tables. To see the metadata for a particular table, issue:

```
SHOW TABLE tablename
```

Note The SHOW SYSTEM command is available only in command-line **isql**, not in IBConsole.

Security

Unlike system tables, which have a default access privilege of SELECT for PUBLIC users, the temporary tables have no default access by PUBLIC. The display and manipulation of this runtime information is restricted to SYSDBA and the database owner. These two users have the option of using the GRANT statement to allow access to other users. The statement can grant only SELECT privileges.

Examples

To illustrate the richness of the possibilities afforded by these temporary tables, here are some examples how you might query them.

Top ten SQL statements by execution

```
SELECT a.tmp$user, s.tmp$timestamp, s.tmp$sql, s.tmp$quantum
   FROM TMP$STATEMENTS s, TMP$ATTACHMENTS a
   WHERE a.TMP$ATTACHMENT_ID = s.TMP$ATTACHMENT_ID
   ORDER BY s.TMP$QUANTUM DESC ROWS 10;
```

Top ten oldest transaction snapshots

```
SELECT a.TMP$USER, t.TMP$TIMESTAMP, t.TMP$TRANSACTION_ID, t.TMP$SNAPSHOT
   FROM TMP$ATTACHMENTS a, TMP$TRANSACTIONS t
   WHERE a.TMP$ATTACHMENT_ID = t.TMP$ATTACHMENT_ID
   ORDER BY t.TMP$SNAPSHOT ROWS 10;
```

Top ten tables with the most garbage to clean up

```
SELECT TMP$RELATION_NAME, TMP$GARBAGE_COLLECT_PAGES
```

```
   FROM TMP$RELATIONS
   ORDER BY TMP$GARBAGE_COLLECT_PAGES DESC ROWS 10;
```

Top ten most executed stored procedures

```
SELECT TMP$PROCEDURE_NAME, TMP$INVOCATIONS
   FROM TMP$PROCEDURES
   ORDER BY TMP$INVOCATIONS DESC ROWS 10;
```

Is database sweep active and what's its progress?

```
SELECT TMP$SWEEP_RELATION, TMP$SWEEP_RECORDS
   FROM TMP$DATABASE
   WHERE TMP$SWEEP_ACTIVE = 'Y';
```

Pool memory allocations grouped by pool type

```
SELECT TMP$TYPE, SUM(TMP$POOL_MEMORY) TMP$TOTAL_MEMORY,
   SUM(TMP$FREE_MEMORY) TMP$TOTAL_FREE
   FROM TMP$POOLS
   GROUP BY TMP$TYPE
   ORDER BY 2 DESC;
```

Updating system temporary tables

There are cases where, having acquired information about the state of the database, you need to take appropriate action. You might, for example, detect a transaction that had unexpectedly been open for many hours, or one that was consuming resources that were needed by others. By updating the TMP$STATE column of certain temporary tables, you can perform the following updates:

- Roll back an active or limbo transaction
- Commit a limbo transaction
- Cancel an attachment's executing operation
- Shut down the current attachment
- Make an executing statement stop running

To roll back an active transaction

```
UPDATE TMP$TRANSACTIONS SET TMP$STATE = 'ROLLBACK' WHERE TMP$TRANSACTION_ID=123;
```

To roll back a limbo transaction

```
UPDATE TMP$TRANSACTIONS SET TMP$STATE = 'ROLLBACK' WHERE TMP$TRANSACTION_ID=123;
```

To commit a limbo transaction

```
UPDATE TMP$TRANSACTIONS SET TMP$STATE = 'COMMIT' WHERE TMP$TRANSACTION_ID=123;
```

To cancel the attachment's currently executing operation

```
UPDATE TMP$ATTACHMENTS SET TMP$STATE = 'CANCEL' WHERE TMP$ATTACHMENT_ID=123;
```

To shut down the current attachment

```
UPDATE TMP$ATTACHMENTS SET TMP$STATE = 'SHUTDOWN' WHERE TMP$ATTACHMENT_ID=123;
```

Shutting down an attachment detaches the user from the database and terminates the local or network attachment to the server.

To make an executing statement stop running

```
UPDATE TMP$STATEMENTS SET TMP$STATE = 'CANCEL' WHERE TMP$STATEMENT_ID=123;
```

Making global changes

The above examples operate on a single attachment or transaction. You can make more global changes. For example:

To roll back all active transactions

```
UPDATE TMP$TRANSACTIONS SET TMP$STATE = 'ROLLBACK' WHERE TMP$STATE ='ACTIVE';
```

To roll back all limbo transactions

```
UPDATE TMP$TRANSACTIONS SET TMP$STATE = 'ROLLBACK' WHERE TMP$STATE ='LIMBO';
```

To commit all limbo transactions

```
UPDATE TMP$TRANSACTIONS SET TMP$STATE = 'COMMIT' WHERE TMP$STATE ='LIMBO';
```

System temporary table metadata

The following tables provide a complete list of the system temporary tables. This information is also in the "System Tables" chapter of the *Language Reference*.

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. 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 2.2 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$USER	VARCHAR[31]	User name
TMP$USER_IP_ADDR	VARCHAR [16]	User IP address
TMP$USER_HOST_NAME	VARCHAR [16]	User host name
TMP$USER_PROCESS_ID	VARCHAR [16]	User process ID
TMP$STATE	VARCHAR [16]	CONNECTED, ACTIVE
TMP$PRIORITY	VARCHAR [16]	Reserved

Table 2.2 TMP$ATTACHMENTS (*continued*)

Column name	Data type	Description
TMP$DBKEY_ID	INTEGER	Transaction ID of dbkey
TMP$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_EXPUNGES	INTEGER	Garbage collect record expunges
TMP$RECORD_BACKOUTS	INTEGER	Garbage collect record backouts

TMP$DATABASE

TMP$DATABASE contains one row for each database you are attached to.

Table 2.3 TMP$DATABASE

Column name	Data type	Description
TMP$DATABASE_ID	INTEGER	Database identifier
TMP$DATABASE_PATH	VARCHAR[255]	Database pathname
TMP$ATTACHMENTS	SMALLINT	Number of active connections
TMP$STATEMENTS	INTEGER	Number of compiled statements
TMP$ALLOCATED_PAGES	INTEGER	Pages allocated to all database files
TMP$POOLS	INTEGER	Number of memory pools
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

Table 2.3 TMP$DATABASE (*continued*)

Column name	Data type	Description
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$TRANSACTION_WAITS	INTEGER	Number of transaction wait fors
TMP$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
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$POOL_BLOCKS

The TMP$POOL_BLOCKS table contains one row for each block of memory in each pool.

Table 2.4 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$BTB	INTEGER	
TMP$BTC	INTEGER	
TMP$CHARSET	INTEGER	
TMP$CSB	INTEGER	Compiler scratch block
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$FUN	INTEGER	
TMP$HNK	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	

Table 2.4 TMP$POOL_BLOCKS (*continued*)

Column name	Data type	Description
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
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 2.5 TMP$POOLS

Column name	Data type	Description
TMP$POOL_ID	INTEGER	Pool identifier
TMP$POOL_TYPE	VARCHAR[16]	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 2.6 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
TMP$RECORD_SELECTS	INTEGER	Records selected by procedure
TMP$RECORD_INSERTS	INTEGER	Records inserted by procedure
TMP$RECORD_UPDATES	INTEGER	Records updated by procedure

Table 2.6 TMP$PROCEDURES (*continued*)

Column name	Data type	Description
TMP$RECORD_DELETES	INTEGER	Records deleted by procedure
TMP$RECORD_PURGES	INTEGER	Garbage collect record purges
TMP$RECORD_EXPUNGES	INTEGER	Garbage collect record expunges
TMP$RECORD_BACKOUTS	INTEGER	Garbage collect record backouts

TMP$RELATIONS

The TMP$RELATIONS table contains one row for each relation referenced since the current connection began.

Table 2.7 TMP$RELATIONS

Column name	Data type	Description
TMP$RELATION_ID	INTEGER	Relation identifier
TMP$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_PAGES	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
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 2.8 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[4096]	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$STATE	VARCHAR[16]	ACTIVE, INACTIVE, STALLED, CANCELLED
TMP$PRIORITY	VARCHAR[16]	Reserved
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$RECORD_SELECTS	INTEGER	Records selected by statement
TMP$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_EXPUNGES	INTEGER	Garbage collect record expunges
TMP$RECORD_BACKOUTS	INTEGER	Garbage collect record backouts

TMP$TRANSACTIONS

The TMP$TRANSACTIONS table contains one row for each transaction that is active or in limbo.

Table 2.9 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$NOWAIT	CHAR[1]	Transaction is no wait
TMP$COMMIT_RETAINING	CHAR[1]	Commit retaining performed
TMP$STATE	VARCHAR[16]	ACTIVE, LIMBO, COMMITTING, PRECOMMITTED
TMP$TYPE	VARCHAR[16]	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

Thread-safe processing of database handles

InterBase 7.0 provides improved handling of InterBase database handles on behalf of client applications. Handle types include attachment, blob, BLR request, SQL statement, service and transaction objects. The client library manages the integrity of InterBase database handles in the face of concurrent application thread activity.

Issues

InterBase 7.0 has been extensively tested. We believe that only very minor issues remain.

License files from previous versions

Before installing, be sure to remove any existing *ib_license.dat* files from the directory you are installing to. The presence of an older license file can result in the error "The InterBase license file is missing or corrupt. Check that InterBase is installed correctly." when starting InterBase. This will be fixed for the next test.

Removing license certificates

Occasionally, if you try to remove a license certificate using the *iblicense* utility, the removal fails with an error message. If you really need to remove the license certificate, you can use a text editor to remove the appropriate line from the *iblicense.dat* file.

The *Getting Started* book

Those of you who are familiar with previous versions of InterBase will remember the *Getting Started* book. This book has been removed from the InterBase 7.0 document set. It consisted of two chapters and covered three topics: installation, licensing, and migration. The migration document is now Appendix A of the *Operations Guide*. Licensing information is in Chapter 7 of the *Operations Guide*. Platform-specific installation information is available in the *install.txt* file found in the directory for each platform. On Windows platforms, you can access install information for all platforms by running *setup.exe* and clicking the *What's New* link.

Chapter

3

InterBase 6.5 features

This chapter lists features that were introduced in InterBase 6.5 and provides an overview and summary of each. Note that this chapter contains references to ODS 10 and 10.1 InterBase 7 uses ODS 11.

Bold blue type indicates clickable links.

InterBase 6.5 features:

Metadata security

64 bit I/O

Asynchronous cancel

New isc_config, ibconfig file parameters:

CPU_AFFINITY

SWEEP_QUANTUM

USER_QUANTUM

SWEEP_YIELD_TIME

Language enhancements:

New keywords

The ROWS clause for SQL data manipulation statements

XML Generation

JDBC (InterClient 2.5)

Metadata security

This feature protects metadata from modification by unauthorized users. It adds the SQL GRANT/REVOKE security framework to InterBase's system tables. The default access privileges for PUBLIC have been changed to allow only SELECT access; this prevents ordinary users from corrupting the database by modifying the system metadata. The database owner, SYSDBA and operating system administrator (root on UNIX and Administrator on Windows) continue to have all privileges to the metadata as well as the privilege to grant access to the metadata to anyone else.

The backup/restore **gbak** utility has been modified to restore security privileges on system metadata. NOTE: **gbak** has always backed up privileges to all tables, it just never restored them to the system tables. **gbak** is also being modified to record the ODS major and minor version attributes of the database that was backed up in the backup file. Knowing that information helps the engine know what system tables need to have default privileges added when the database is restored.

Scripts for changing database security

Note The InterBase 6.5 release changes the ODS (On Disk Structure) from 10.0 to 10.1. This is a minor version update (10.0 to 10.1) to indicate that system tables now have default privileges for PUBLIC access.

When an ODS 10.0 (InterBase 6.0 engine) database is attached by InterBase 6.5, it automatically adds the default SQL privileges of SELECT-only to the system tables. InterBase 6.5 can still attach ODS 9 databases but it does *not* update them with SQL privileges; ODS 9 is supported as a bridge and, as much as, possible those databases should not be modified.

Three SQL scripts are included in *<ib_install>/examples/security*: *readmeta.sql*, *writemeta.sql* and *blindmeta.sql.* These scripts can be run against databases with ISQL to make wholesale changes to a database's metadata PUBLIC privileges.

- *Readmeta.sql* applies the default PUBLIC access privileges of the IBv6.5 engine. It can be used to return a database with customized metadata privileges back to the default.
- *Writemeta.sql* grants all metadata privileges to PUBLIC. This is the metadata access profile that existed in InterBase 6.0 and earlier.
- *Blindmeta.sql* revokes all metadata privileges from PUBLIC. This prevents any PUBLIC user from querying the system tables. This includes InterBase and third-party utilities run by PUBLIC users. For example, **gstat**, **gbak**, QLI and IBConsole would not be able to query system metadata. This script allows developers to protect their intellectual property by hiding the database design of tables, stored procedures and triggers from the general public and competitors. Blind access makes it difficult, if not impossible, for a general user to generate ad hoc queries against a database.

A user can develop a custom script to grant privileges to some individual users while denying access to others. **gbak** always preserves privileges on system tables across a database backup/restore.

A database with blind access does not prevent any user from using InterBase Data Definition Language (DDL) to define new database objects. It just prevents a user from querying or writing to the metadata directly. The project of granting privileges on a database to allow a user to define metadata objects (tables, procedures, indices etc.) is not within the scope of this project.

Requirements and constraints

Two client-side APIs, *isc_blob_lookup_desc()* and *isc_array_lookup_bounds()*, cannot execute without SELECT metadata privileges, because the APIs directly query some InterBase system tables. Thus databases that have had *blindmeta.sql* run against them are not able to execute these APIs for any users except the owner.

Preventing all access to metadata by running *blindmeta.sql* against a database prevents some utilities from correct operation when run by a general user. The special users (database owner, SYSDBA etc.) are still able to run these titillates in the normal fashion.

InterBase 6.0 and previous InterBase kits cannot access a database on behalf of a user if that user has no privileges to the system tables. The InterBase 6.5 engine had to be modified so that it could access the system tables as an agent for users who had no privilege to do so directly. Thus an InterBase 6.5 developer who runs *blindmeta.sql* on an InterBase database cannot ship that database to customers with InterBase 6.0 runtime kits and expect those users to be able to access the database. The developer would have to run *readmeta.sql* against a copy of the database and ship that to customers who have older InterBase runtimes.

Migration issues

The InterBase 6.5 and 7 engines automatically install the default SQL privileges for PUBLIC on the metadata when attaching ODS 10.0 databases. Thus if all users must be able to write to database metadata, *writemeta.sql*will have to be run against each database to restore that behavior.

Note ODS 10.1 and ODS 11 databases created anew by InterBase 6.5 and 7 have the default SQL privileges for PUBLIC installed at create time.

Pre-ODS 10 databases will have to be migrated, as usual, by database backup/restore to get the metadata protection. If the security scripts are run against pre-ODS 10 databases it is not clearly defined what the behavior will be. *Readmeta.sql* and *writemeta.sql* may function as expected but *blindmeta.sql* will prevent access to the database by normal users. In no case will a backup/restore preserve metadata privileges across a restoration. The scripts would have to be manually run against the restored database for secure metadata. It is not advised that customers run these scripts from older kits (InterBase 6.0 and older) unless they have done full life-cycle testing of such a configuration to their satisfaction.

Going forward, InterBase will always be able to figure out the required system privileges to add to new InterBase system tables in future releases. Since **gbak** has been changed to record the ODS major and minor versions of the database that was backed up, InterBase is always able to compute the incremental privileges required to bring the restored database up-to-date. It does this while preserving any customized system privileges users may have made to their databases.

Performance

64 bit I/O

InterBase 6.5 supports the creation of database files greater than 4 GB. In previous versions of InterBase to support files sizes greater than 4 GB one had to create multi-file databases. If one did not create a multi-file database and their database file exceeded the 4 GB limit file corruption could occur.

Asynchronous cancel

The ability to cancel an executing DSQL statement is an often requested feature that was included in InterBase 6.5. This feature modifies an existing DSQL API to request that a statement have its execution cancelled and interrupted.

The existing DSQL API *isc_dsql_free_statement()* has been modified to include a new option which requests that an executing statement be cancelled.

A third option of *DSQL_cancel* has been added to indicate that statement execution should be cancelled.

A successful return does not mean that statement execution was cancelled, only that the statement was marked for cancellation in the server.

The DSQL_cancel option allows for the asynchronous cancellation of an executing statement. The client that was executing the statement receives a status code of *isc_cancelled*. Once a statement has been cancelled, any subsequent execution restarts the statement, rather than resuming it.

isc_dsql_free_statement()

Frees a statement handle and all resources allocated for it, or closes a cursor associated with the statement referenced by a statement handle.

Syntax

```
ISC_STATUS isc_dsql_free_statement (ISC_STATUS *status_vector, isc_stmt_handle *stmt_handle, unsigned short
option);
```

Parameter	Type	Description
status_vector	*ISC_STATUS *	Pointer to the error status vector
stmt_handle	*isc_stmt_handle**	Pointer to a statement handle previously allocated with *isc_dsql_allocate_statement()* or *isc_dsql_alloc_statement2()*; the handle returns an error in status_vector if it is NULL
option	*unsigned short*	Either DSQL_close, DSQL_drop, or DSQL_cancel

Description *isc_dsql_free_statement()* either frees a statement handle and all resources allocated for it *(option = DSQL_drop)*, or closes a cursor associated with the statement *(option = DSQL_close)*.

Note *isc_dsql_free_statement()* does nothing if it is called with an option value other than *DSQL_drop* or *DSQL_close*.

.DSQL_close

Call *isc_dsql_free_statement()* with the *DSQL_close* option to close a cursor after it is no longer needed, that is, after fetching and processing all the rows resulting from the execution of a query. A cursor need only be closed in this manner if it was previously opened and associated with stmt_handle by *isc_dsql_set_cursor_name()*. *DSQL_close* closes a cursor, but the statement it was associated with remains available for further execution.

If you have used a cursor to perform updates or deletes on all the rows returned from the execution of a query, and you want to perform other update or delete operations on rows resulting from execution of the same statement again (possibly with different input parameters), follow these steps:

1. Close the cursor with *isc_dsql_free_statement()*.
2. Re-open it with *isc_dsql_set_cursor_name()*.
3. If desired, change the input parameters to be passed to the statement.
4. Re-execute the statement to retrieve a new select list.
5. Retrieve rows in a loop with *isc_dsql_fetch()* and process them again with *isc_dsql_execute_immediate()*.

.DSQL_drop

Statement handles allocated with *isc_dsql_allocate_statement()* must be released when no longer needed by calling *isc_dsql_free_statement()* with the *DSQL_drop* option. This option frees all resources associated with the statement handle, and closes any open cursors associated with the statement handle.

.DSQL_cancel

Statement handles allocated with *isc_dsql_allocate_statement()* must be released when no longer needed by calling *isc_dsql_free_statement*().

Chapter 6 “Working with Dynamic SQL” in the *API Guide* provides additional information on how to use API dynamic SQL (DSQL) functions to handle dynamically created SQL statements for data definition and manipulation.

New *isc_config*, *ibconfig* file parameters

Note Starting with InterBase 7.0, the configuration files for all platforms are named *ibconfig*.

CPU_AFFINITY

This parameter specifies the processors to be used by the InterBase server on a SMP machine. This parameter is only available on Windows. The default value is 1. The InterBase server uses the first CPU on the system by default. A process affinity mask is a bit vector in which each bit represents the processor on which the threads of the process are allowed to run.

Example

For processor 1:	1 (1st bit in bit vector)
For processor 2:	2 (2nd bit in bit vector)
For processors 1 & 2:	3 (11, 1st and 2nd bits in bit vector)
For processor 3:	4 (3rd bit in bit vector)
For processors 1,2 and 3:	7 (111 in bit vector)
For processors 2 and 3:	6 (110 in bit vector)

And so on.

SWEEP_QUANTUM

This parameter specifies the maximum number of records that a garbage collector thread or a sweeper thread is allowed to work before yielding control back to the worker threads. The default value is 10.

USER_QUANTUM

This parameter specifies the maximum number of records that a worker thread (thread running an user query) is allowed to work before yielding control back to other threads. The default value is 100.

SWEEP_YIELD_TIME

This parameter specifies the number of milliseconds the sweeper or the garbage collector thread sleeps. The default value is 1 millisecond. Note that this does not affect worker threads.

Notes These configuration parameters are platform independent; they are applicable on all platforms. Note that SWEEP_QUANTUM and SWEEP_YIELD_TIME affect the sweeper/garbage collector thread. Thus, ideally, when one is changed the other warrants change too. For example, if you change SWEEP_YIELD_TIME to a larger value, then it is advisable to change SWEEP_QUANTUM to a larger value also, since the sweeper thread sleeps for a larger time so it is good to do more work when its awake. This is particularly true when there are lots of back versions in the database. The default values are fine for most applications however there might be applications where in a bit of tuning is needed for optimal performance.

Specifying a value of zero or less is ignored by the server and results in using the default value for that particular parameter. However, there is no upper limit applied to these parameters. The server reports the values for each of these parameters in the *interbase.log* file on startup.

Language enhancements

New keywords

InterBase 6.5 introduces the following new keywords:

PERCENT ROWS TIES

The ROWS clause for SQL data manipulation statements

The ROWS clause is used in conjunction with the SQL ORDER BY clause to provide a ranking capability for SQL queries. The ROWS clause is not part of the SQL standard but is an InterBase SQL language extension intended to provide functionality similar to InterBase's GDML FIRST clause.

With such a ranking relation, it becomes meaningful to allow both SQL UPDATE and DELETE to take an ORDER BY clause for the purpose of restricting row modifications to a top / bottom percentage or number of rows.

The ROWS clause can also be used in isolation to restrict the number of rows returned from the result set. There is a positional, though random and unpredictable, semantic to the result set. It is used to partition a result set into subsets for retrieval purposes by applications of limited means.

The ROWS clause subsets the number of rows from the result set of an arbitrary table expression. The feature is used primarily by Web developers to parcel pieces of a larger result set from the Web server to a client browser. This type of Web application has a stateless interface with the database server and cannot gradually scroll the full result set via a cursor or download the entire result set into a cached dataset on the client. Rather the same SQL query is iteratively submitted to the database server but with a ROWS clause to specify which numbered rows from the full result set should be returned.

The ROWS clause can be used with the SQL SELECT, UPDATE and DELETE statements. It has a general syntax of ROWS <*lower_value*> [TO <*upper_value*>] [BY <*step_value*>] [PERCENT] [WITH TIES]. The <*value_expressions*> are positive numbers which specify how many rows to include in the ranking.

Note The ORDER BY clause is required when using the TIES clause.

When ROWS ... PERCENT is used, <*value_expression*> evaluates to a number less than or equal to 100. Conceptually, the COUNT of the result set is calculated, multiplied by <*value_expression*> and then divided by 100 with integer rounding per SQL conformance.

ROWS ... WITH TIES always includes rows with duplicate, equal rankings beyond the count of rows requested. Asking for the ORDER BY SCORES ROWS 3 WITH TIES with scores of 98, 96, 92, 92, 92, 87 returns 98, 96, 92, 92, 92 (5 rows and not 3) because tie scores have been requested.

The BY <step_value> allows intermediate rows from the result set to be skipped over. It can be used to break the result set into percentage bands. For example, ROWS 100 BY 5 PERCENT would reduce an arbitrary number of rows to 20 rows with each row distanced from its neighboring row by 5 percent of the total rows.

Note The ROWS clause introduces three new InterBase SQL keywords: ROWS, TIES, and PERCENT.

Syntax

```
SELECT <select_list>
FROM <table_list>
WHERE [GROUP BY clause]
[HAVING clause]
[UNION clause]
[ORDER BY clause]
[ROWS <lower_value> [TO <upper_value>] [BY <step_value>] [PERCENT] [WITH TIES]]
```

Example An informal UPDATE syntax would be as follows:

```
UPDATE <table>
SET <column_assignmnent_list>
WHERE <search_predicate>
[ORDER BY clause]
[ROWS <lower_value> [TO <upper_value>] [BY <step_value>] [PERCENT] [WITH TIES]]
```

An informal DELETE syntax has semantics similar to UPDATE:

```
DELETE FROM <table>
```

```
WHERE <search_predicate>
[ORDER BY clause]
[ROWS <lower_value> [TO <upper_value>] [BY <step_value>] [PERCENT] [WITH TIES]]
```

Examples The following two isql statements select the top 100 performing sales people. This example shows the ability for a Web developer to split the result set in half for display purposes.

```
SELECT SALESMAN, SALES_DOLLARS, SALES_REGION
FROM SALESMEN
ORDER BY SALES_DOLLARS DESC
ROWS 1 TO 50;
```

This second isql statement makes use of the TIES keyword.

```
SELECT SALESMAN, SALES_DOLLARS, SALES_REGION
FROM SALESMEN
ORDER BY SALES_DOLLARS DESC
ROWS 51 TO 100 WITH TIES;
```

The following isql statements select the best 100 performing salesmen and rewards them with a 15 percent bonus:

```
UPDATE SALESMEN
SET SALES_BONUS = 0.15 * SALES_DOLLARS
ORDER BY SALES_DOLLARS DESC
ROWS 100 WITH TIES;
```

The following isql statements selects the worst 5% performing members of sales force and deletes them:

```
DELETE FROM SALESMEN
ORDER BY SALES_DOLLARS
ROWS 5 PERCENT WITH TIES;
```

Support for ROWS clause in gpre

GPRE now supports this additional functionality of the ROWS clause. Currently this feature is only supported with the C / C++ programming language.

XML Generation

InterBase customers can now generate XML documents directly from InterBase without having to learn anything new about XML. We have added two new API calls: *isc_dsql_xml_fetch*() and *isc_dsql_xml_fetch_all*(), which clients use to create XML files.

These functions are a part of a new client side DLL called *ibxml.dll* on Windows and *ibxml.so* on Solaris and Linux.

- The structures defined for these functions are located in a new header file called *ibxml.h*.
- The new prototype definitions are included in the file *ibxml_proto.h*. This header file also internally includes *ibxml.h*. To use this feature you must to add this library to your link path, and the new header file to your compiler include files.

The new function prototypes are:

- `int isc_dsql_xml_fetch(ISC_STATUS *status, isc_stmt_handle *stmt, unsigned short da_version, XSQLDA *sqlda, IB_XMLDA *ib_xmlda);`
- `int isc_dsql_xml_fetch_all(ISC_STATUS *status, isc_stmt_handle *stmt, unsigned short da_version, XSQLDA *sqlda, IB_XMLDA *ib_xmlda);`
- `int isc_dsql_xml_buffer_fetch(ISC_STATUS *status, isc_stmt_handle *stmt, USHORT da_version, char *buffer, int buffer_size, XSQLDA *sqlda, IB_XMLDA *ib_xmlda);`

Here `status` is a pointer to the *ISC_STATUS* error status long array, and `stmt` is the pointer to the statement handle previously allocated using calls like *isc_dsql_allocate_statement*(). `da_version` indicates the version of the extended XSQLDA passed to the function. `sqlda` is a pointer to the extended SQL descriptor Area (XSQLDA). `ib_xmlda` is a pointer to an initialized XML descriptor area (IB_XMLDA) explained in the section above.

In case of the first call the user still has access to the data in the cursor using the sqlda.

The `IB_XMLDA` structure is explained below. This structure is contained in the *ibxml.h* file.

```
typedef struct ib_xmlda
  c
{
char ISC_FAR  *xmlda_file_name;     /* contains the file name which will be opened in append mode
                                       for writing the xml into. */
char ISC_FAR  *xmlda_header_tag;    /* points to the string which is printed out as the version tag*/
char ISC_FAR  *xmlda_database_tag;  /* points to the string which is printed out as the database tag in
                                       the xml file */
char ISC_FAR  *xmlda_table_tag;     /* points to the string which is printed out as the database tag in
                                       the xml file */
char ISC_FAR  *xmlda_row_tag;       /* points to the string which is printed out as the rowname tag in
                                       the xml file */
FILE          *xmlda_file_ptr;      /* points to a FILE structure which has been opened for writing
                                       the data in xml format, internal use only*/
char ISC_FAR  **xmlda_temp_buffer;  /* internal use only, used for storing the String array from
                                       fetch() */
ISC_STATUS    Xmlda_fetch_stat;     /* holds the return value from the isc_dsql_fetch() call, it indicates
                                       if we have received all the records or if we have an error */
ULONG         xmlda_flags;          /* flags explained below */
ULONG         xmlda_more_data;      /* used by the buffer call to maintain status of the last record, 0 if
                                       there is no more data 1 if there is more data has been fetched but
                                       not put out in the buffer */
ULONG         xmlda_temp_size;      /* internal use only, used to store the last records size */
USHORT        xmlda_version;        /* version of XMLDA */
USHORT        xmlda_status;         /* reserved for future use */
USHORT        xmlda_more;           /* this flag is used in conjunction with the buffered mode, if there
                                       is more XML data this is set */
USHORT        xmlda_version;        /* version of XMLDA */
USHORT        xmlda_array_size;     /* internal use only */
SLONG         xmlda_reserved;       /* reserved for future use */
              } IB_XMLDA;

#define XMLDA_ATTRIBUTE_FLAG 0x01 /* if set the data is output as attributes */
#define XMLDA_NO_NULL_DISPLAY_FLAG 0x02 /* if set the null data is not displayed */(not implememented right
for 6.5)
#define XMLDA_NO_HEADER_FLAG 0x04 /* no additional header to be displayed */
```

Before calling the functions, you must set the following elements of the structure:

1. **xmlda_file_ptr** should be assigned to a previously opened FILE pointer. The file is assumed to be open for writing. The function starts writing from the location of the write pointer. You can set or reset the write pointer to a specific location if you wish. It is recommended that you do not modify this FILE structure once it is in use by the function. You are responsible for closing the file.
2. **xmlda_version** should be currently set to 1. The function does not currently look for a value.
3. **xmlda_status** should be set to 0 the first time *isc_dsql_xml_fetch()* is called. It does not have any effect in *isc_dsql_fetch_all()*, but it is recommended that you set it to 0. It is used internally by the XML functions to keep status.

Optional variables

1. **xmlda_header_tag** points to a character string to be used as the xml header. If this is set to NULL, it prints the default header, `<?xml version="1.0">`.
2. **xmlda_database_tag** points to a character string which can be used in place of the Database tag (see example XML document below). If this is set to NULL, the XML tag defaults to "Database".
3. **xmlda_table_tag** points to a character string which can be used in place of the Tablename tag (see example XML document below). If this is set to NULL the XML tag defaults to "Tablename".
4. **xmlda_row_tag** points to a character string which can be used in place of the Row tag (see example XML document below). If this is set to NULL, the XML tag defaults to "Row".
5. **xmlda_flags** currently has two allowable values:
   - XMLDA_ATTRIBUTE_FLAG generates the XML document as attributes instead of as tags
   - XMLDA_NO_NULL_DISPLAY_FLAG (not implemented for 6.5) does not display the null data and the associated tags.
6. **xmlda_file_ptr** should be assigned to a previously opened FILE pointer. The file is assumed to be open for writing. The function starts writing from the location of the write pointer. You can set or reset the write pointer to a specific location if you wish. It is recommended that you do not modify this FILE structure once it is in use by the function. You are responsible for closing the file.

Setting the XMLDA_ATTRIBUTE_FLAG results in all the output data being generated as attributes instead of elements. This flag will only affect the actual data generated from InterBase as the user can control all the other tags by inputting the desired attribute as tags.

Setting the XMLDA_NULL_NO_DISPLAY_FLAG causes the API to skip generating rows for data that is null. The default behavior is to generate empty strings.

Blobs and arrays are not supported in this initial version.

In order to use the *isc_dsql_xml_buffer_fetch*(), you must allocate at least a 1024 character buffer, which is passed to the function in the *buffer* argument. The *buffer_size* argument reports the size of this passed buffer. The function returns the size of characters written into the buffer without the terminating null character. In case of error, it returns -1 if there is not enough memory for it to continue, or -2 if the buffer size is too small. The function does not return incomplete headers, footers, or records. The *xmlda_more_data* is set if the call should be made once again to get the complete XML buffer.

In order to make the calls work with Delphi, please use the regular POINTER type to hold space in the structure IBXMLDA for the FILE * type.

For code examples see: *xml_api_buffer.c* and *xml_api_file.c* in the *examples* directory.

JDBC (InterClient 2.5)

Removing Deprecated JRE API calls from InterClient

This feature has been implemented so that now the InterClient code is up to date with the latest shipping JREs (1.3). The only requirement for the implementation of this functionality is the existence of Java 2 on the platform. This implementation allows InterBase to be stronger now that the code has moved to the most currently maintained JRE.

New JDBC methods

Class: `DatabaseMetaData`
Methods: `supportsMixedCaseQuotedIdentifiers()`
`storesMixedCaseQuotedIdentifiers()`
`getIdentifierQuoteString()`

Description: Added quoted identifier support for InterBase 6, dialect 3 onwards for the above methods in the *DatabaseMetaData* class.

```
Class: ResultSet
Method: getBigDecimal()
```

Description: Added the *getBigDecimal*() functions without scale. The older and now deprecated functions took two possible arguments: *getBigDecimal*(*int, int scale*) or *getBigDecimal*(*string, int scale*). The new ones take *getBigDecimal*(int) and *getBigDecimal*(*String*).