Getting Started with InterBase

Disclaimer

Borland International, Inc. (henceforth, Borland) reserves the right to make changes in specifications and other information contained in this publication without prior notice. The reader should, in all cases, consult Borland to determine whether or not any such changes have been made.

The terms and conditions governing the licensing of InterBase software consist solely of those set forth in the written contracts between Borland and its customers. No representation or other affirmation of fact contained in this publication including, but not limited to, statements regarding capacity, response-time performance, suitability for use, or performance of products described herein shall be deemed to be a warranty by Borland for any purpose, or give rise to any liability by Borland whatsoever.

In no event shall Borland be liable for any incidental, indirect, special, or consequential damages whatsoever (including but not limited to lost profits) arising out of or relating to this publication or the information contained in it, even if Borland has been advised, knew, or should have known of the possibility of such damages.

The software programs described in this document are confidential information and proprietary products of Borland.

Restricted Rights Legend. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subdivision (b) (3) (ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

Corporate Headquarters: Borland International Inc., 100 Borland Way, P. O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-5300. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom.

Software Version: V3.0

Current Printing: October 1993
Documentation Version: v3.0.1

Reprint note

This documentation is a reprint of InterBase V3.0 documentation. It contains most of the information from *InterBase Previous Versions Documentation Corrections* and *InterBase Version* 3.2 *Documentation Corrections* and a new index. For information on features added since InterBase Version V3.0, consult the appropriate release notes.

Table of Contents

Preface

Who Should Read this Book

You should read this book before you read any other books in the InterBase documentation set. This book provides a:

- Conceptual overview of InterBase
- Description of each of InterBase's features
- Roadmap to InterBase's documentation set

This book assumes some database knowledge, but no knowledge about InterBase.

Using this Book

This book contains the following chapters and appendixes:

Chapter 1	Introduces InterBase, the system components, and the InterBase transaction model.
Chapter 2	Explains relational database concepts and terms and **qli**, the query and update language interpreter, and how to access the sample database.
Chapter 3	Describes defining a database.

Chapter 4	Explains developing applications using InterBase.
Chapter 5	Describes InterBase's advanced features.
Chapter 6	Describes the forms screen designer and the report writer. It also provides information on VAR and third-party products.
Chapter 7	Describes InterBase system utilities, recovery mechanisms, and system configuration options.
Appendix A	Lists supported platforms.
Appendix B	Lists product specifications.
Appendix C	Lists which InterBase features are supported in SQL and GDML.

Text Conventions

This book uses the following text conventions:

	Indicates whether the section contains information on an SQL or a GDML topic or both.
boldface	Indicates a command, option, statement, or utility. For example: • Use the **commit** command to save your changes. • Use **gdef** to extract a data definition.
italic	Indicates chapter and manual titles; identifies file-names and pathnames. Also used for emphasis, or to introduce new terms. For example: • See the introduction to SQL in the *Programmer's Guide*. • /usr/interbase/lock_header

	• Subscripts in RSE references *must* be closed by parentheses and separated by commas. • C permits only *zero-based* array subscript references.
`courier font`	Indicates what you type, example code, and system output: • $run sys$system:iscinstall • add field population_1950 long
UPPER CASE	Indicates relation names and field names: • Secure the RDB$FILES system relation. • Define a missing value of *X* for the LATITUDE_COMPASS field.

Examples

The *Example* section provides examples you can try in **qli**. These examples provide you with a basic understanding of the concepts presented in this book.

InterBase Documentation

The InterBase Version 3.0 documentation set contains the following books:

Database Operations (INT0032WW2178D) describes how to maintain InterBase databases.

Data Definition Guide (INT0032WW2178F) describes how to create and modify InterBase databases.

DDL Reference (INT0032WW2178E) describes the function and syntax for the data definition language.

DSQL Programmer's Guide (INT0032WW2179C) describes how to program with dynamic SQL (DSQL), with which you generate SQL statements for dynamically generated queries.

Forms Guide (INT0032WW2178A) describes how to create forms using the InterBase forms editor, **fred,** and how to use forms in **qli**, GDML applications.

Getting Started with InterBase (INT0032WW2179A), this book, provides an overview of InterBase components and interfaces.

Programmer's Guide (INT0032WW2178I), describes how to program with GDML, a relational data manipulation language, and with SQL, an industry standard language.

Programmer's Reference (INT0032WW2178H) describes the function and syntax for GDML and SQL.

Qli Guide (INT0032WW2178C) describes how to use **qli**, the InterBase query and update language interpreter. **Qli** allows you to read to and write from databases using interactive GDML or SQL statements.

Qli Reference (INT0032WW2178B) describes the function and syntax for the data definition, SQL, and GDML you can use in **qli.**

Sample Programs (INT0032WW2178G) contains sample programs that show the use of InterBase features.

Master Index (INT0032WW2179B) contains index entries for the entire InterBase Version 3.0 documentation set.

1

Understanding InterBase

Overview

InterBase is a relational database management system (RDBMS) designed to reduce the difficulty, expense, and risk of creating complex applications. InterBase's complete implementation of SQL supports all relational concepts and structures. To allow the creation of applications that common SQL engines cannot deliver, InterBase provides features to make the database system a powerful ally in application development.

On-Line Complex Processing

The applications at which InterBase excels are those that manage:

- Time-critical transactions whose contents and sequence are unpredictable and which must interact with other time-critical processes.
- Objects, both structured and unstructured, such as drawings, arrays, large documents, and digitized data that must be controlled and shared like traditional data.
- Data distributed in a heterogeneous multi-vendor network with minimal cost in system design, operation, and administration.

These characteristics define an On-Line Complex Processing (OLCP) application. InterBase provides a suite of features and facilities for the application designer who must deliver a full-scale OLCP application.

Time-Critical, Event-Driven Transactions

On-Line Complex Processing applications often must respond to an unpredictable variety of requests and transactions in a timely way. InterBase's features provide a uniquely responsive environment.

Transaction Management: At InterBase's core is a *multi-generational record architecture* that produces optimal throughput for multi-user, high-contention applications.

Traditional locking schemes, used by other database systems, force processes to wait in line to modify or read data that another process is using. The delays and overhead introduced by data locking are unacceptable in time-critical OLCP applications.

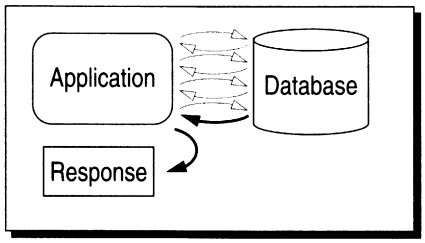
InterBase's multi-generational records provide the data consistency of record locks without their cost. Furthermore, the InterBase architecture eliminates conflicts between read and write transactions, so data analysis has no impact on the throughput of on-line transaction processing.

In addition, multi-generational records allow you to back up databases and to change the metadata while the database is on line with other applications—with no degradation in performance.

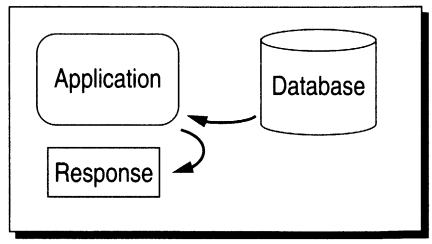
Event management: Time-critical applications frequently need to respond to unpredictable events. Rather than leave the management of these events up to the application designer, InterBase provides *event alerters.*

In brief, event alerters are a way of notifying your application immediately when a predefined data change (or other event) has occurred. Unlike traditional programmatic event poll/wait loops, InterBase event alerters use a simple, no-overhead wait mechanism to listen for events. The figures below illustrate the difference between polling and event alerters.

Polling for Database Change



InterBase Event Alerters



Because event alerters work remotely in a heterogeneous network, your program need not solve the problems of signaling remote machines of a different type.

Dynamic Integrity Control: *Trigger procedures,* or *triggers* ensure that your data maintains its integrity regardless of the unpredictable ways in which it is accessed. Triggers govern what happens to the database when data items are changed, added, or removed. For example, a trigger might forbid adding a payroll record for an employee not listed in the personnel records.

InterBase triggers provide a complete and robust environment for managing data rules. You can specify the sequence in which triggers should operate, including whether they should operate before or after the database update with which they are associated. Triggers "cascade"—that is, a trigger that updates a database causes any associated triggers to fire, which may activate additional triggers, and so on.

Recovery: InterBase provides timely recovery from system malfunctions in the following ways:

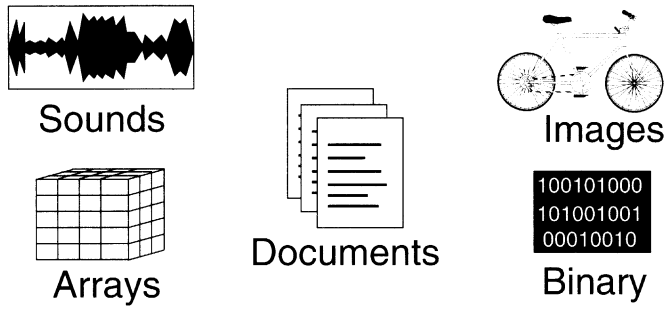
- InterBase's *multi-generational architecture* ensures that a consistent database is instantly available at system restart, regardless of the number of transactions in progress at the time of failure.

- *Journaling,* when activated, keeps a log of all transactions, to allow you to return the database to any previous uncorrupted state.

- *Shadowing,* when activated, creates a duplicate database on another hardware node, so that processing can resume instantly when the original node fails.

Object Handling Under Relational Control

Applications involving complex objects—digitized images, binary data, or lengthy text—have far outstripped the capacity of most database systems to handle the data. InterBase's advanced datatypes, illustrated in the figure below, permit you to store anything you can digitize—right in the database with other data, under full security, transaction, and recovery control.

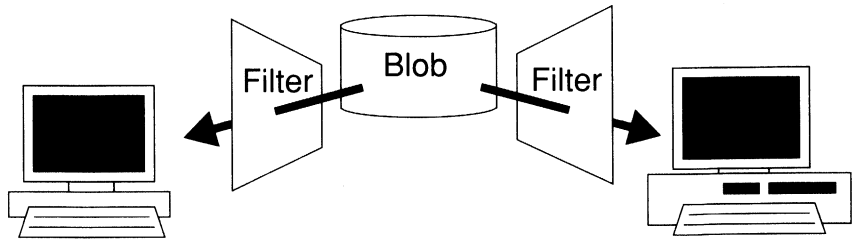


In addition, InterBase manages functions centrally, in the database itself, for common access by multiple applications without redundancy.

Arrays: Interbase's *array* datatype lets you store multi-dimensional arrays in a single field. Arrays are particularly useful in scientific and analytical applications, such as data acquisition, component testing, and time-series analysis.

Blobs: Interbase's *blob* datatype stores digital data of any type and any length. Typically blobs are used for images, digitized voice, word-processing documents, and binary executable files.

You can instruct InterBase to handle blobs more intelligently through the use of *blob filters*—code that translates the stored blob data into a another format (or subtype) more useful to a particular application or computer hardware. InterBase stores blob filters centrally, in the database to which they pertain, so that coding and maintenance is kept to a minimum.



User-Defined Functions: To perform non-standard calculations on data, you can register *user-defined functions* with the database. Once defined, these functions deliver the correct results to any workstation on the InterBase network, regardless of hardware type.

Computed Fields: InterBase's *computed fields* are fields whose value is not stored in the database, but is calculated at runtime from other record values. A computed field can be as simple as concatenating first, middle, and last names to produce a full-name field, or, with the use of user-defined functions, as complex as great-circle distance from Greenwich.

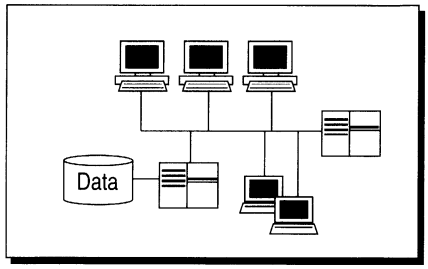
Processing Transparency in Multi-Vendor Networks

InterBase is the only RDBMS to take distributed database to its logical conclusion. Where client-server architectures allow you to distribute only the client processing, InterBase's *peer-to-peer processing* lets you distribute clients, servers and data.

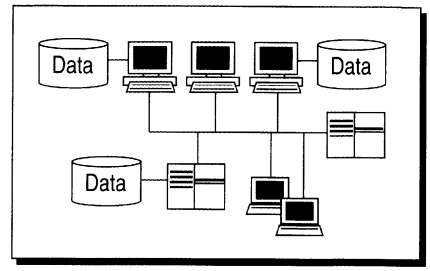
In the peer-to-peer architecture, data can reside on any or all nodes in a network, and be accessed by any other node. Because of this distribution, peer-to-peer processing is less susceptible to bottlenecks than single-server architecture, and eliminates the server as a single point of failure.

Of course, InterBase provides client-server architecture (multi-client with single-server) where application or organizational structure require. The following figure illustrates how peer-to-peer architecture is a logical extension of the client-server model.

Client-Server: Single Database Server, Multiple Clients



InterBase Peer-to-Peer: Multi-Client Multi-Server



InterBase transactions span multiple databases and nodes, allowing you to read and update records on diverse hardware platforms with complete integrity. The key to multi-node updating is InterBase's *automatic two-phase commit*.

Whenever you call for a transaction that spans more than one network node, InterBase senses the need for a two-phase commit and puts it into action. The first phase of the commit confirms that participating databases are ready to commit the transaction; the second commits it.

Applications of InterBase OLCP

Complex applications in distributed environments are difficult, costly, or even impossible to implement with conventional OLTP-style database management systems. InterBase is designed to streamline the implementation of complex distributed applications, so your application development effort can focus on supporting your business.

- A financial trading application can use event alerters to notify interested traders that a stock has changed value—regardless of the type of workstation at their desk. Data arrays capture price changes in real time; at the same time, traders can analyze data and order trades on the same database without degrading throughput.

- A commercial software developer uses InterBase's blob datatype to manage version control on software modules. Blob filters

convert centrally stored source code into the appropriate format for developers on different operating systems, then reverse the procedure when the module returns. When a module is checked in, a user-defined function calculates the changes made and stores a difference record in another blob.

- A process control application relies on InterBase's high-throughput architecture to monitor semiconductor furnace conditions in real time, and allows corrective actions to be taken from either local or remote nodes transparently.

- Other organizations use InterBase to:
  - Manage voice and data networks.
  - Analyze data acquired in real time.
  - Manage CAD data and version control.

InterBase Components

InterBase utilities serve the development phases of an OLCP application by:

- Defining the database structures which underlie your application—the application *metadata*.

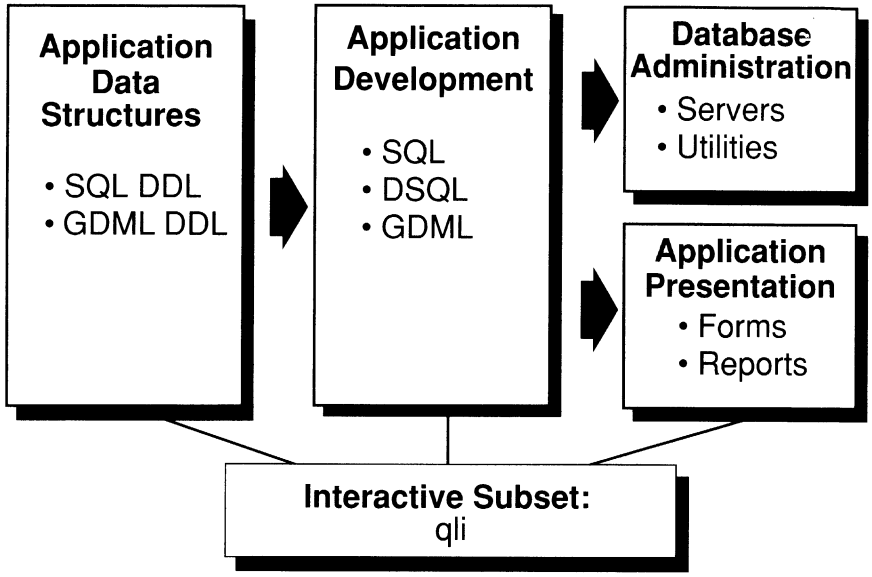
- Developing the application itself, coding in a combination of conventional third-generation languages, ANSI SQL, and InterBase's proprietary data manipulation language, GDML.

- Organizing the end-user presentation and use of the program.

- Administering databases and servers, and restoring damaged or lost data.

In addition to the application tools, InterBase supplies an interactive language interface, called **qli**, that provides access to most of InterBase's features. Using **qli**, you can explore the system, try out functionality, and prototype application components—without the impediment of the compile-run-debug cycle.

The following figure illustrates the relationship of InterBase's components to the application development process.



Data Manipulation Languages

InterBase offers you a choice of two data manipulation languages (DMLs):

- ANSI level II SQL for compliance with standards and existing programming techniques.
- GDML, a proprietary languages that accesses InterBase's advanced features not addressed by the ANSI SQL standard. Rather than extend the SQL standard with non-standard additions, InterBase provides this second language that covers both standard and advanced features.

Choosing Between SQL and GDML

Fortunately, you can combine SQL and GDML in any application. If you already know SQL, you can use it for normal data manipulation statements and still use GDML when you need special capabilities.

SQL offers you the following advantages:

- *Transportability:* Because it is a standard, SQL code can be imported from non-InterBase applications with little or no change.

- *Training:* SQL is widely known and understood, which keeps training to a minimum and makes ongoing maintenance of your application somewhat simpler.

GDML offers these advantages:

- *High-Performance Features:* Many advanced features of InterBase are not supported by SQL—or any other database language. Rather than extending the SQL standard, InterBase provides GDML to access OLCP features such as blobs, arrays, and event alerters.

- *Ease of Use:* Often, GDML syntax is easier to use than SQL. The **for**-loop construct, for instance, eliminates the need for cumbersome SQL cursors to retrieve multiple records.

- *Rich Semantics and Environmental Control:* GDML data management offers you more control over the program's environment than SQL does. For example, your organization may prefer the GDML approach to database security, or you may need the level of transaction management that GDML provides.

- *Automatic Error Handling:* Unlike SQL, every GDML statement detects and responds to errors. This avoids or detects many common programming pitfalls.

The following table shows which aspects of the InterBase RDBMS can be used from which DML. A more complete version of this table is found in Appendix C.

Feature	Database Language	
Data Definition	SQL	GDML
Data Manipulation	SQL	GDML
Interface to 3GL Programs	SQL	GDML
Security	SQL	GDML
Interactive Use (**qli**)	SQL	GDML
Reports		GDML
Blobs & Blob Filters		GDML
Arrays		GDML
Event Alerters		GDML
Triggers		GDML
User-Defined Functions		GDML
Forms		GDML

InterBase Utilities

When you create an application in InterBase, you use several different utilities, each of which is designed for a specific task:

- **Gdef** is a data definition processor which creates a database from a data definition file. **Gdef** can define relations, fields, views, security, and trigger definitions. For simple data definitions and modifications, **gdef** can be used interactively.

- **Gpre** is InterBase's program precompiler, which you use to process DML statements inside 3GL programs.

- **Fred** is a menu-driven tool for designing screen forms. Because **fred** is a database utility, you can use database fields and tables in your forms directly.

- **Qli** is InterBase's query and update interpreter. **Qli** is an ideal way to learn and test most of the system's features interactively.

- **Gbak, gfix, grst,** and **gcsu** are utilities that control backup, configuration, and restoration of databases.

Optional utilities and interfaces to other commercial software products are also available to make your application more powerful and your development process more efficient.

Where to Go From Here

The remaining chapters of this book expand on the InterBase features and concepts introduced here. To learn more about the listed InterBase features, read the chapter indicated in the following table.

To learn more about	Refer to
Application Development	Chapter 4, Developing Applications
Arrays	Chapter 5, Advanced Programming
Blobs	Chapter 3, Defining a Database
Blob Filters	Chapter 5, Advanced Programming
Computed Fields	Chapter 3, Defining a Database
Database Definition	Chapter 3, Defining a Database
Event Alerters	Chapter 5, Advanced Programming
Forms	Chapter 6, Finishing Touches
GDML DDL	Chapter 3, Defining a Database
GDML DML	Chapter 4, Developing Applications
Options & Interfaces	Chapter 6, Finishing Touches

To learn more about	Refer to
QLI	Chapter 2, Designing & Prototyping
Report Writer	Chapter 6, Finishing Touches
Servers	Chapter 7, Database Administration
SQL DDL	Chapter 3, Defining a Database
SQL DML	Chapter 4, Developing Applications
Triggers	Chapter 3, Defining a Database
Transactions	Chapter 4, Developing Applications
Two-Phase Commit	Chapter 4, Developing Applications
User-Defined Functions	Chapter 5, Advanced Programming

All features are described in detail in the InterBase documentation set.

2

Designing and Prototyping Applications

Overview

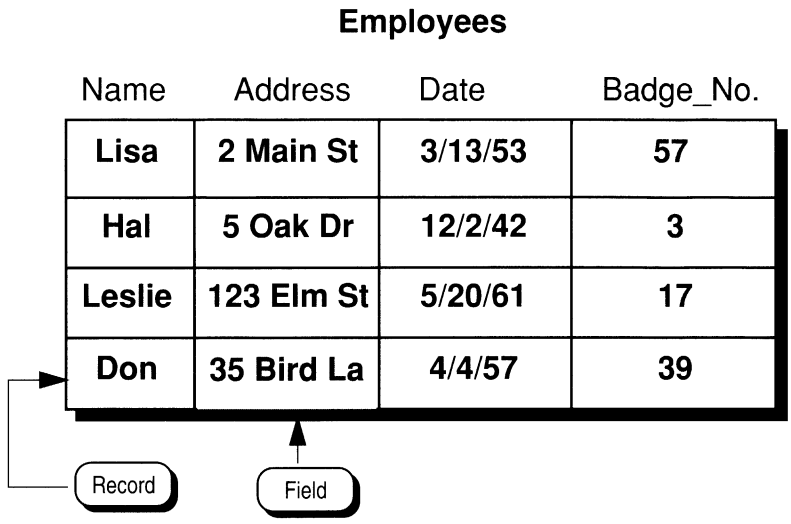
You can design and prototype applications in **qli**. **Qli** is InterBase's query and update language interpreter. You can use **qli** to:

- Display, store, and update data and prototype these operations for inclusion in application programs.
- Display or generate reports of data.
- Display, store, and update data using forms.
- Define databases, relations, fields, indexes, views, and procedures.

Before you design and prototype your database application, it is important to understand some of the basic concepts of relational database theory.

Relational Database Terms and Concepts

A *relational database* is data perceived by the user as a collection of *relations*. The diagram below shows an example of a relation called EMPLOYEES.



The EMPLOYEES relation contains *records*. A record contains *fields*, and each field in a record contains data. The data in a field can be any of the following datatypes:

- Binary, a short or longword integer with optional decimal scale
- Blob, a basic large object that holds documents, graphics, images, or any other large unstructured data
- Character, varying or fixed text
- Date, date/time
- Float, a single precision 32-bit or a double precision 64-bit datatype

- Multi-dimensional array, a subdivided database field used to store a large amount of related data elements in a structured fashion

When identifying database elements, you can use the following InterBase, SQL, or academic terms.

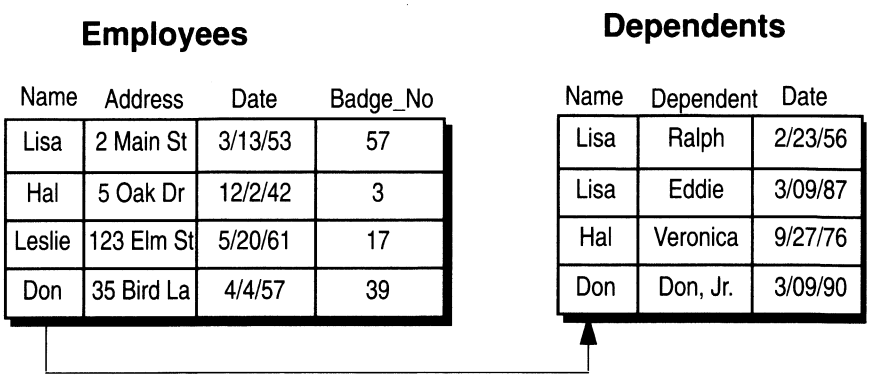
InterBase	SQL	Academic
Relation	Table	Relation
Record	Row	Tuple
Field	Column	Attribute

This book uses the InterBase terms to identify database elements.

Joining Relations

The key to a relational database is the ability to join relations through common data in fields. This is called a *join.*

The following diagram joins the NAME field from EMPLOYEES relation to the NAME field from the DEPENDENTS relation.



You can join relations in the following ways:

- One record to one record
- One record to many records
- Many records to many records

Using Qli

Qli lets you interactively:

- Access databases and InterBase features
- Prototype applications

It is also the simplest way to learn InterBase.

Qli is used in many of the examples throughout this book. The following sections discuss **qli**'s prompts and how to correct mistakes in **qli**.

Qli Prompts

Qli has two prompts, QLI> and CON>. The QLI> prompt indicates the system is ready for a new command. The CON> prompt indicates a command has not been completed and you may continue.

Note

Some of the examples presented in this book use a hyphen (-) to indicate that a command continues on the next line.

Correcting Mistakes

If you mistype a command in **qli**, type the word *edit* and **qli** makes your previously typed command available with the default operating system editor. Use the editor's commands to correct the mistake, exit from the editor, and **qli** executes the corrected command. If you do not want to execute any changes, quit from the editor.

If you want to experiment without damaging or permanently changing data in a database, you can use the **rollback** command. Rolling back data changes undoes all updates made to the database since the last commit.

Accessing the Sample Database

The examples used in this manual refer to data stored in the InterBase sample database, *atlas.gdb*.

If you plan to try the examples presented in the *Example* sections, use the *atlas.gdb* database. Before running the examples, make your own copy of the database in a local directory. This way you can make changes to the database without affecting the original database.

The following table shows how to copy the sample *atlas.gdb* database file to your current directory:

O/S	Command
Apollo Domain/IX	% cp /interbase/examples/atlas.gdb .
UNIX	% cp /usr/interbase/examples/atlas.gdb .
VMS	$ copy interbase$ivp:atlas.gdb atlas.gdb *.*

Note

If you are on a NFS mounted file system on a Sun, the UNIX copy example may not work. Instead, use the following command:

% cp /usr/interbase/examples/atlas.gdb atlas.gdb

Where to Go From Here

For more information on using **qli**, use the table below to direct you to the appropriate book in the InterBase documentation set.

To learn more about	Refer to the chapter on	In the
Accessing **qli**	Introduction to **qli**	*Qli Guide*
Using **qli**	Introduction to **qli**	*Qli Guide*
Accessing the sample database	Introduction to **qli**	*Qli Guide*
Relational concepts	Designing a database	*Data Definition Guide*

3

Defining a Database

Overview

SQL
GDML

Before you begin to develop your application, you need to understand the design of the database or databases your application will use. In some cases, you may need to define the database or databases from scratch.

You define an InterBase database by naming the primary database file and defining the appropriate database components.

You can define the following basic database components:

- The database itself
- Fields, to represent your data
- Relations, to group your data
- Indexes, to improve retrieval performance or enforce a field's uniqueness
- Views, to display a limited subset of fields or records, or to display data from a combination of relations
- Security, to prevent unauthorized access to a field, relation, or database

You can also define the following advanced database components, which are described in Chapter 5, *Advanced Features*:

- User-defined functions, to perform arithmetic calculations
- Blob filters, to convert data from one blob subtype to another
- Triggers, to enforce data integrity and define events
- Event Alerters, to notify programs of changes to the database

The components described above, which structure the database, are called *metadata*.

Rather than use multiple files, InterBase keeps all database components in a single file. This is an advantage when you are, for example, backing up, transporting, or maintaining a database.

Interfaces You Can Use

You can use either SQL or GDML to define database objects.

The following table lists the database objects that each interface can define.

Component	SQL	GDML
Database	✓	✓
Field (local)	✓	✓
Field (global)		✓
Field (computed)		✓
Relation	✓	✓
Index	✓	✓
View	✓	✓
Security	✓	✓

Component	SQL	GDML
User-Defined Functions (Chap. 5)		✓
Blob Filter (Chap. 5)		✓
Trigger (Chap. 5)		✓
Event Alerter (Chap. 5)		✓

Data Dictionary

SQL GDML

Like all relational database management systems, InterBase stores information about metadata in a *data dictionary*. The dictionary provides a central storage area for the metadata associated with a database.

InterBase creates its data dictionary by using *system relations*. Information is stored in system relations automatically when you define database components using the SQL and GDML interfaces.

You can see the system relations by using **qli show** commands. If you're an advanced programmer, you may sometimes want to access the system relations directly.

Starting the Definition Process

You start the data definition process by naming the database you want to define. You can use **qli**, **gdef**, or embedded SQL to name a new database.

To name a database using the SQL variant of **qli**, type: SQL

```
QLI> create database my_atlas.gdb
```

To name a database using the GDML variant of **qli**, type: GDML

```
QLI> define database my_atlas.gdb
```

Defining Fields

You define fields by using **qli**, **gdef**, embedded SQL, or dynamic SQL. You can define:

- *Local fields* using the SQL variant of **qli**, embedded SQL, or DSQL. You always define a local field within the context of a relation.

- *Global fields* using the GDML variant of **qli** or **gdef**. A global field is a description of a data element that is independent of the relation in which it is used. You can include a global field in any relation.

  ANSI SQL does not support the concept of a global field.

- *Computed fields* using **gdef**. A computed field is a virtual field. InterBase never stores data in such fields. Instead, it uses the formula to retrieve the requested data.

  Computed fields are not globally available, because their formula is meaningful only within the context of the relations in which they are defined.

When you define a field, you must specify its datatype. If the field is a character field, you must also specify its length.

For fields defined using GDML, you can optionally specify these other field attributes:

- *Valid if*, which specifies validation criteria for a field

- *Edit string*, which specifies a display format for a field in **qli**

- *Query header*, which specifies a column header for a **qli** display

- *Query name*, which specifies an alternate field name for use in **qli**

To define a local field called STREET using the SQL variant of **qli**, type: SQL

```
QLI> create table tourism -
CON> (street varchar(25));
```

This statement creates a local field called STREET of datatype varying that's a maximum of 25 characters long. It creates this field within the context of the TOURISM relation.

To define a global field called STREET using the GDML variant of **qli**, type: GDML

```
QLI> define field street varying [25]
```

This creates a global field called STREET of datatype varying that's a maximum of 25 characters long. This field exists independently of a relation.

Defining Relations

SQL GDML

You define relations by using **qli**, **gdef**, embedded SQL, or dynamic SQL.

For relations defined with the:

- SQL variant of **qli**, embedded SQL and DSQL, you must define the relation's fields when you define the relation.
- GDML variant of **qli** or **gdef**, you can:
  - Reference existing global fields. When you reference an existing field, you can use the **based on** clause to give it a name specific to the relation being defined.
  - Define new global fields.
  - Define computed fields.

To define a relation called PARKS using the SQL variant of **qli**, type: SQL

```
QLI> create table parks -
CON> (name varchar(20),
CON> type char(1),
CON> city varchar(20),
CON> state varchar(4));
```

GDML To define this relation using the GDML variant of **qli**, type:

```
QLI> define relation parks
CON> name varying[20],
CON> type char[1],
CON> city,
CON> state;
```

This definition assumes the CITY and STATE field have already been defined to the database.

SQL GDML

Defining Views

Views can be a:

- Vertical subset of fields from a single relation. This type of view limits the fields that are displayed.
- Horizontal subset of records from a single relation. This type of view limits the records that are displayed.
- Combined vertical and horizontal subset of records from a single relation. This type of view limits both the fields and records that are displayed.
- Subset of records from many relations. This type of view usually performs a join operation.

You define a view by using **gdef**, embedded SQL, or dynamic SQL. When you define a view, you can use a *context variable* (GDML) or *alias* (SQL) to provide name recognition and distinguish one relation from another. In the examples below, the context variable is the letter "c".

GDML Suppose you want to define a view called MAP_CITIES that limits the fields returned to the program.

To define this view using **gdef**, type:

```
GDEF> define view map_cities
CON> of c in cities
CON> c.city,
CON> c.state,
CON> c.latitude,
CON> c.longitude;
```

Now, suppose you want to define a view that joins the CITIES and STATES relations on the STATE field.

To define this view using **gdef**, type: GDML

```
GDEF> define view city_states
CON> of s in states
CON> cross c in cities over state
CON> c.city,
CON> s.state_name,
CON> c.altitude,
CON> c.latitude,
CON> c.longitude;
```

Defining Indexes

SQL GDML

Indexes improve performance for retrieval operations and can be used to enforce uniqueness for data values. As a general rule, you should define an index for:

- A relation's primary key.
- A relation's foreign keys.
- Non-key field that are accessed frequently for retrieval purposes.
- Non-key fields that are unique.

You define an index by using **gdef**, **qli**, embedded SQL, or dynamic SQL.

SQL To define a unique index for the STATE field in the STATES relation using the SQL variant of **qli**, type:

```
QLI> create unique index states_idx2
CON> on states (state)
```

GDML To define the same index using the GDML variant of **qli**, type:

```
QLI> define index states_idx1 for states
CON> unique
CON> state
```

SQL GDML

Defining Security

InterBase provides two security schemes. The SQL security scheme (**grant/revoke**) automatically limits access to relations and fields. It then lets you grant and revoke access to these fields.

The GDML security scheme lets you limit access to:

- The database itself
- Relations
- Views
- Fields in a relation or view

The GDML security scheme uses *security classes* to define classes of users and access rights. You can assign a security class to any of the database objects listed above.

Both SQL security and GDML security are enforced across all interfaces and platforms.

SQL To grant access to the STATES relation using the SQL security scheme, type the following in **qli**:

```
QLI> grant select on states to juliec,
CON> dana;
```

To secure the STATES relation from unauthorized access by using GDML security, type the following in **gdef**:

GDML

```
GDEF> modify database 'atlas.gdb';
GDEF> modify relation states
CON> security_class my_security_list;
```

This example assumes that the security_class MY_SECURITY_LIST already exists.

Where to Go From here

SQL

For more information on defining basic database components using the SQL interface, use the table below to direct you to the appropriate book in the InterBase documentation set.

To define	Refer to the chapter on	In the
Empty databases	Defining metadata with SQL	*Programmer's Guide*
	Defining metadata	*Qli Guide*
Local fields	Defining metadata with SQL	*Programmer's Guide*
	Defining metadata	*Qli Guide*
Relations	Defining metadata with SQL	*Data Definition Guide*
	Defining metadata	*Qli Guide*
Indexes	Defining metadata with SQL	*Programmer's Guide*
	Defining metadata	*Qli Guide*
Views	Defining metadata with SQL	*Programmer's Guide*

To define	Refer to the chapter on	In the
Security	Defining metadata with SQL	*Programmer's Guide*
	Defining metadata	*Qli Guide*

GDML

For more information on defining basic database components using the GDML interface, use the table below to direct you to the appropriate book in the InterBase documentation set:

To define	Refer to the chapter on	In the
Empty Databases	Creating a database	*Data Definition Guide*
	Defining metadata	*Qli Guide*
Global fields	Defining fields	*Data Definition Guide*
	Defining metadata	*Qli Guide*
Computed fields	Defining relations	*Data Definition Guide*
Relations	Defining relations	*Data Definition Guide*
	Defining metadata	*Qli Guide*
Indexes	Defining views and indexes	*Data Definition Guide*
	Defining metadata	*Qli Guide*
Views	Defining views and indexes	*Data Definition Guide*
Security	Securing Data	*Data Definition Guide*

4

Developing Applications

Overview

SQL GDML

Developing InterBase application programs involves:

- Understanding the InterBase transaction environment
- Using appropriate data manipulation language (DML) statements in your program
- Knowing how to preprocess your program

You can develop applications using either SQL, GDML, or both.

The InterBase Transaction Environment

SQL GDML

In any programming environment, a related set of changes should be made in their entirety or not at all. If your program terminates before it finishes performing the complete set of related changes to a database, you would want the database restored—or *rolled back*—to the state it was in before the set of changes started.

InterBase provides this capability through *transactions*. A transaction is a bounded set of statements that:

- Succeed or fail as a group

- Are not corrupted by changes made by other processes
- Maintain a constant image of the metadata

SQL GDML

Transaction Models

To give your application greater flexibility and help it achieve the greatest possible throughput, InterBase provides two transaction models, the:

- *Consistency model*, which is available through embedded GDML programs
- *Concurrency model*, which is the default for embedded GDML and embedded SQL programs

SQL GDML

Concurrency Model: For higher throughput in many OLCP applications, you can use InterBase's *concurrency* model. This model provides complete consistency for read-only transactions without waiting for updates or risking deadlocks. It also provides high consistency and efficient conflict resolution for read-write transactions.

The concurrency model eliminates the bottleneck of data locking by creating multi-generational records for each active transaction. Each transaction is free to read data and make changes. InterBase monitors the transactions for potential conflicts and supplies appropriate messages or resolutions.

A major advantage to the concurrency model is a transaction always reads a stable view of the database, even though changes may occur while the read takes place.

An additional advantage is the multi-generational records take up little additional space since InterBase stores difference records only. Also, InterBase deletes records when they are no longer needed.

GDML

Consistency Model: The transaction environment found in most traditional database management systems uses a *consistency* model. This model locks records or relations whenever someone reads or writes to them. The InterBase consistency model locks relations.

The consistency model is useful for special cases in which there is a high risk that simultaneous updates might damage relationships between records.

However, this model is less useful for an environment containing a mixture of update and read-only transactions. Within the consistency model, users exclude each other from individual records or entire relations. This means that read transactions must wait for update transactions to complete, even when the updates are not changing the records the reader wants.

Multiple-Database Access

SQL GDML

Another major feature of InterBase is its ability to access more than one database in a single transaction, whether the databases are on your node or elsewhere on the network.

InterBase automatically uses a *two-phase* commit when a transaction involves multiple databases:

- The first phase of the commit checks all participating databases to see if anything stands in the way of committing the transaction.

  If any database reports a failure or fails to report success during the first phase, all other subtransactions roll back.

- When all participating databases check in, InterBase issues the second phase of the commit, which causes all participating databases to commit.

The two-phase commit and two-phase rollback guarantee your data is always consistent, no matter where it resides.

Other Transaction Features

GDML

The InterBase transaction environment also gives you the ability to:

- Use multiple transactions in a single database.
- Use default transaction options.

- Override the default options.

Using Multiple Transactions: When you use multiple transactions, you can group statements into autonomous units and commit or roll them back individually. You can update related records with the certainty that all changes are made simultaneously.

Using Default Transaction Options: InterBase provides default transaction options that are useful for a variety of applications.

Overriding the Default Options: If your program requires additional control over transactions, you can override the default options. You can override these options for individual transactions and for specific relations within a transaction. You can also start a transaction explicitly.

SQL GDML

Retrieving Data

You retrieve data from an InterBase database by using either a GDML **for** loop or an SQL **select** statement.

Using these statements, you can specify:

- The fields you want to retrieve from the database
- Whether you want only unique values to be returned
- The source relations for the records
- The selection criteria for the records
- The order in which the data will be returned

SQL

Using an SQL Select Statement

The **select** statement establishes an input record stream by placing all qualifying rows in a temporary table called a *results* table.

There are two versions of the **select** statement:

- If the search condition you specify returns no more than a single record, you can use the simple version of the **select** statement.
- If you expect the search condition to return multiple rows, you must use a *cursor* with the **select** statement. A *cursor* is a device that points to rows in the results table. Using a cursor results in a more complex query, because you must open the cursor, fetch each record into corresponding host variables, and close the cursor.

The distinction between single-row and multiple-row selects applies only to embedded SQL and dynamic SQL. In **qli**, you retrieve multiple rows by using a simple **select**.

Using a GDML For Loop

GDML

Like the SQL **select** statement, the GDML **for** loop establishes an input record stream that represents the data you requested. Unlike the **select** statement, the **for** loop returns records one at a time, without using a cursor.

Within the body of the **for** loop, you can include host language statements to manipulate or to print field values, and GDML statement to access blobs or arrays.

The **for** loop stops automatically after the last record has been retrieved. If the loop encounters an error, InterBase returns a complete error message and stops processing.

To print information on rivers that are shorter than 50 miles using embedded SQL in a C program, type:

SQL

```
exec sql
   declare small_rivers cursor for
   select river, source, outflow,
      length
   from rivers where length < 50;

exec sql
   open small_rivers;
```

```
exec sql
   fetch small_rivers
      into :river, :source, :outflow
         :length;

while (!SQLCODE)
{
   printf ("%s %s %s %d\n", river,
       source, outflow, length);
exec sql
   fetch small_rivers
      into :river, :source, :outflow,
         :length;
}

if (SQLCODE != 100)
   gds_$print_status (gds_$status);

exec sql
   close small_rivers;
```

GDML To print the same information using embedded GDML in a C program, type:

```
for r in rivers
   with length < 50
   printf ("%s %s %s %d\n", r.river,
      r.source, r.outflow, r.length);
end_for;
```

SQL GDML

Storing Data

You store data in an InterBase database by using either an SQL **insert** statement or a GDML **store** statement.

Using an SQL Insert statement

SQL

The source of values for the SQL **insert** statement can be any combination of the following:

- Quoted literal expressions
- Numeric expressions
- Prompting expressions (**qli** only)
- **Qli** variables (**qli** only)
- Host variables
- Field values from a subquery

Using a GDML Store Statement

GDML

The source of values for the GDML **store** statement can be any combination of the following:

- Quoted literal expressions
- Numeric expressions
- Prompting expressions (**qli** only)
- **Qli** variables (**qli** only)
- Host variables
- Field values from other records

You can also store blob and date fields by using special routines.

To insert a new record into the RIVERS relation using embedded SQL in a C program, type: SQL

```
exec sql
   insert into rivers
```

```
        (river, source, outflow, length)
    values ('Unkety Brook', 'MA', -
        'Nashua River', 13);

if (SQLCODE)
{
printf ("SQL error, code %d\n", SQLCODE);
gds_$print_status (SQLCODE);
}
```

GDML To insert the same record using embedded GDML in a C program, type:

```
store r in rivers using
    strcpy (r.river, "Unkety Brook");
    strcpy (r.source, "MA");
    strcpy (r.outflow, "Nashua River");
    r.length = 13;
end_store;
```

SQL GDML

Modifying Data

You modify data in an InterBase database by using either an SQL **update** statement or a GDML **modify** statement.

SQL

Using the SQL Update Statement

The source of values for the SQL **update** statement can be any combination of the following:

- Quoted literal expressions
- Numeric expressions
- Qli variables (**qli** only)
- Prompting expressions (**qli** only)
- Host variables
- Field values from a subquery

Using the GDML Modify Statement

GDML

The GDML **modify** statement can be used:

- Alone, for a mass update
- Within a **for** loop
- Interactively

The source of values for the **modify** statement can be any combination of the following:

- Quoted literal expressions
- Numeric expressions
- Prompting expressions (**qli** only)
- **Qli** variables (**qli** only)
- Host variables
- Field values from other records

To modify the LENGTH field for the Jeffreys Creek river using embedded SQL, type:

SQL

```
exec sql
   update rivers set length = 13
      where name = 'Jeffreys Creek';

if (SQLCODE)
   printf ("SQLCODE = %d\n", SQLCODE);
```

Note

InterBase supports double or single quotation marks (using the apostrophe key).

GDML To modify this field using embedded GDML, type:

```
for r in rivers
   with r.name = 'Jeffreys Creek'
   modify r using
      r.length = 13;
   end_modify;
end_for;
```

SQL To selectively modify the OUTFLOW field for rivers whose sources are in New Hampshire using embedded SQL, type:

```
exec sql
   declare r cursor for select river
      from rivers where source = 'NH'
      for update of outflow;

exec sql
   open r;

exec sql
   fetch r into :river;

while (!SQLCODE)
{
printf ("New outflow for %s, ", river)'
gets (outflow);

exec sql
   update rivers set outflow = outflow
      where current of r;

if (SQLCODE)
   break;

exec sql
   fetch r into :river;
}

if (SQLCODE !=100)
   gds_$print_status (gds_$status);
}
```

```
exec sql
   close r;
```

To modify this field using embedded GDML, type: GDML

```
for r in rivers with r.source = 'NH'
   printf ('New length for
       %s: ", r.river);
   modify r using
      gets (r.length);
   end_modify;
end_for;
```

Erasing Data

SQL GDML

You erase data from an InterBase database by using either an SQL **delete** statement or a GDML **erase** statement.

Using the SQL Delete Statement

SQL

You can perform a mass delete both through the **qli** variant of SQL and through embedded SQL. You can perform a selective delete through embedded SQL.

Using the GDML Erase Statement

GDML

You can perform a mass delete both through the **qli** variant of GDML and through embedded GDML. You can also perform a selective delete through both variants of GDML.

To delete all rivers that originate in New Hampshire using embedded SQL, type: SQL

```
exec sql
   delete from rivers
       where source = 'NH';
```

GDML To do the same delete using embedded GDML, type:

```
for r in rivers with r.source = 'NH'
   erase r;
end_for;
```

SQL To selectively delete the rivers that originate in New Hampshire using embedded SQL, type:

```
exec sql
   declare r cursor for select river
      from rivers where source = 'NH';

exec sql
   open r;

exec sql
   fetch r into :river;

while (!SQLCODE)
   {
   printf ("Enter \'Y\' to delete the
      %s:", r.river);
   gets (response);
   if (!strcmp (response, "Y"))
      exec sql
         delete from rivers
         where current of r;
   if (SQLCODE)
      break;
   exec sql
      fetch r into :river;
}

if (SQLCODE != 100)
   gds_$print_status (gds_$status);

exec sql
   close r;
```

To do the same delete using embedded GDML, type: GDML

```
for r in rivers with r.source = 'NH'
   printf ("Enter \'Y\' to delete the
      %s: ", r.river);
   gets (response);
   if (!strcmp (response, "Y"))
      erase r;
end_for;
```

Preprocessing Programs

SQL GDML

Gpre is the InterBase preprocessor that translates SQL, DSQL, and GDML statements into statements the host language compiler accepts. **Gpre** does this by generating InterBase library function calls.

Gpre also translates SQL, GDML, and DSQL database variables into variables the host language compiler accepts. **Gpre** then declares these variables in host language format.

The following commands preprocess a GDML program written in C for input into the C compiler: GDML

Operating System	Sample Command
Apollo	% gpre -c -e my_program
UNIX	% gpre -c -e my_program
VMS	% gpre /c /e my_program

Where To Go From Here

SQL

For more information on developing applications using the SQL Interface, use the table below to direct you to the appropriate book in the InterBase documentation set:

To learn more about	Refer to the chapter on	In the
Transactions	The InterBase transaction environment	Programmer's Guide
	Understanding transactions	Qli Guide
Retrieving data	Retrieving data with SQL	Programmer's Guide
	Accessing data using SQL	Qli Guide
	Introductory information	DSQL Programmer's Guide
Storing data	Writing data with SQL	Programmer's Guide
	Writing data	Qli Guide
	Introductory information	DSQL Programmer's Guide
Modifying data	Writing data with SQL	Programmer's Guide
	Writing data	Qli Guide
	Introductory information	DSQL Programmer's Guide
Erasing data	Writing data with SQL	Programmer's Guide
	Writing data	Qli Guide
	Introductory information	DSQL Programmer's Guide
Preprocessing	Preprocessing your program	Programmer's Guide

For more information on developing applications using the GDML Interface, use the table below to direct you to the appropriate book in the InterBase documentation set:

GDML

To learn more about	Refer to the chapter on	In the
Transactions	The InterBase transaction environment	Programmer's Guide
	Understanding transactions	Qli Guide
Retrieving data	Retrieving data with GDML	Programmer's Guide
	Accessing data using GDML	Qli Guide
Storing data	Writing data with GDML	Programmer's Guide
	Writing data	Qli Guide
Modifying data	Writing data with GDML	Programmer's Guide
	Writing data	Qli Guide
Erasing data	Writing data with GDML	Programmer's Guide
	Writing data	Qli Guide
Preprocessing	Preprocessing your program	Programmer's Guide

5

Advanced Features

Overview

GDML

OLCP applications go beyond the capabilities of conventional relational database management systems. Requirements for such things as digitized voice, multi-dimensional arrays and instantaneous event notification make applications difficult —if not impossible—to program, particularly in a network environment. InterBase offers some unique solutions:

- Casting, to convert data from one type to another
- Subqueries, to build complex requests
- Recursive queries, to join a relation to itself
- User-defined functions, to do conversions or calculations from any program
- Blobs and blob subtypes, for storing unformatted data
- Blob filters, to convert data from one subtype to another
- Arrays, to store large amounts of related data elements in a single record field

- Triggers, to execute specific actions when a record is stored, modified, or erased
- Event Alerters, to detect and report changes in a database to users anywhere in the network

GDML

Casting

Casting converts data from one type to another. For example, you can convert floating point data to string, string to integer, date to strings. To use InterBase's casting capability, you append a casting datatype name to the database field you want to convert.

For example, to cast a date field, you qualify a field that holds time or date information by appending **.char[*n*]** to the field name.

Example

GDML

The following statement assigns today's date to a field:

```
s.statehood.char[6] == 'TODAY';
```

GDML

Subqueries

Some queries are unnatural or impossible to express as joins. For example, in the atlas database, you cannot get a list of states with a smaller than average area by joining states to itself. You can easily get that list with a *subquery*. A subquery is a query within another query. The outer query is evaluated first. The subquery operates on the outer query's results.

You can use subqueries to build complex requests. For example, nested **for** loops in GDML can produce outer joins or combine relations from different databases.

Example

The following example uses a nested **for** loop to list all states, and then the baseball teams for states that have them: GDML

```
QLI> for s in states sorted by s.state
CON> begin
CON>   print s.state_name
CON>     for b in baseball_teams over
CON>     state sorted by b.team_name
CON>   print b.team_name, b.city,
CON>   b.home_stadium
CON> end
```

Recursive Queries GDML

A *recursive query* joins a relation to itself to establish a hierarchy. Using the GDML **request** options, you can specify the level of a request, or *instantiation*, so you can execute a parts explosion or bill-of-materials. Increasing or decreasing the level moves you up or down the instantiation tree, thus moving up or down the bill-of-materials.

Note

If you use SQL statements in a GDML program, you cannot use them with a request instantiation. SQL does not support request levels.

User-Defined Functions GDML

A *user-defined function* (UDF) is an executable routine you define and add to InterBase to do conversions or calculations from any program.

For example, you can create a function that changes a temperature value from Fahrenheit to Celsius. Or you can create a function that calculates compound interest.

You can use UDFs to:

- Select records

- Create computed fields
- Validate data

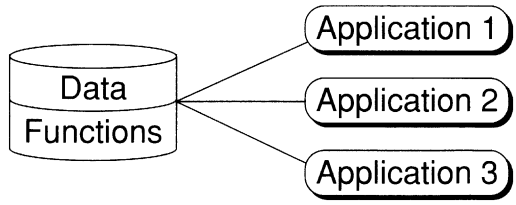
You can also use user-defined functions in triggers.

GDML Advantages of User-Defined Functions

InterBase's UDFs are managed by the database. In other database systems, each application needs its own copy of the function modules, possibly coded differently for differing operating environments. With InterBase, all UDFs are accessible to all applications. This eliminates the need for multiple copies of the same function module. Additionally, you can write the code for the platform on which the data resides.

You have the security of knowing that all programs in your application are using the same function and that when your business requirements change, you can upgrade the function in one place.

The figure below shows applications sharing the UDFs you define for your database.



GDML Defining User-Defined Functions

Creating a user-defined function involves:

- Defining the function's location and parameters to the database by using **gdef**.
- Writing the function, creating the function library, and accessing the function.

Example

An example of a function definition is presented below. To define a function called ABS by using **gdef**, type: GDML

```
GDEF> define function ABS
CON> module_name 'FUNCLIB'
CON> entry_point 'FN_ABS'
CON> double by value,
CON> double by value return_value;
```

Programming with User-Defined Functions GDML

To program with user-defined functions:

1. Write the function and compile it into object code.
2. Define the function's location and parameters to the database with **gdef**.
3. Create the function library and make it available to InterBase at run time.

Blobs GDML

The *basic large object* or blob datatype is available with InterBase and other DSRI-compatible software. A blob looks like a stream or sequential file, but behaves much like a field in a relation. They are stored as a whole and are accessed in discrete chunks called *segments*.

Blobs are best suited for the storage of:

- Unformatted data, such as:
  - Text
  - Images
  - Digitized data
  - CAD drawings

- Any other entity that does not lend itself to storage as longwords or strings.

GDML Blob Subtypes

When you define a blob, you can specify a subtype that describes the blob data. There are two categories of subtypes you can use:

- Predefined subtypes that InterBase uses internally
- Subtypes that you define as needed

You use blob subtypes with blob filters in storing and retrieving blob data.

GDML Advantages of Blobs

The blob datatype provides unstructured data with all of the advantages of a database management system, including:

- Full transaction control
- Maintenance by the same utilities as more structured data
- Manipulation with high-level user interfaces

With blobs, you can keep unstructured entities right in your database. You do not have to store pointers to non-database files, nor do you have to make sure the database and its attendant files remain synchronized.

GDML Programming with Blobs

GDML supports several ways of accessing blob data. You can use:

- **For** loops for reading blobs.
- Statements similar to those used for file processing for both reading and writing blobs.
- A library of routines supporting blob and/or file interchange and stream-like processing.

- Call interface routines.
- Blob filters.

Blob Filters

GDML

A *blob filter* is a program that converts data stored in blob fields from one blob subtype to another subtype. A blob filter is stored with the database, so it becomes part of a database rather than a piece of each application's code.

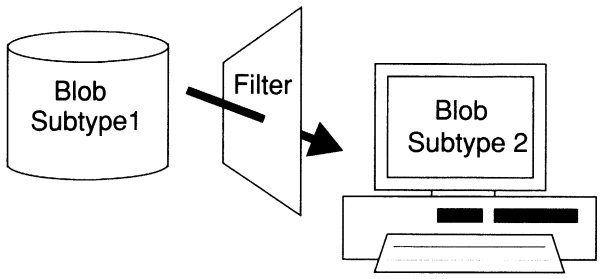
For example, in a database of employee information, you might have a blob field that stores each performance review. To keep this information confidential you might store the review in encrypted form. You can then create blob filters to encrypt the review upon storage, and to decrypt upon retrieval.

Advantages of Blob Filters

GDML

Blob filters reduce application code since all filters are available to all applications. Additionally, they are kept in a central location (the database) so they are associated with a database rather than an application.

The figure below shows how a blob filter can convert data to a format appropriate to a particular platform.



GDML Defining Blob Filters

A blob filter is a program that converts data stored in blob fields from one blob subtype to another subtype.

For example, suppose you have a blob field containing text stored in the **nroff** markup language, and you want that text displayed in its formatted form. You can create a blob filter that automatically formats the marked up text whenever the blob field is retrieved.

Creating a blob filter involves:

- Defining the filter's location and parameters to the database by using **gdef**.
- Writing the filter, creating a filter library, and accessing the filter.

An example of defining a blob filter is presented below.

Example

GDML To define a blob filter called NROFF, type:

```
GDEF> define filter nroff_test_filter
CON> input_type -1
CON> output_type 1
CON> module_name 'FILTERLIB'
CON> entry_point 'NROFF_FILTER';
```

GDML Programming with Blob Filters

To program with blob filters:

1. Write the filter program in a host language and compile it.
2. Define the filter's location and parameters with **gdef**.
3. Build a shared filter library and make it available to InterBase at run time.
4. Write a program that requests blob filtering.

Arrays

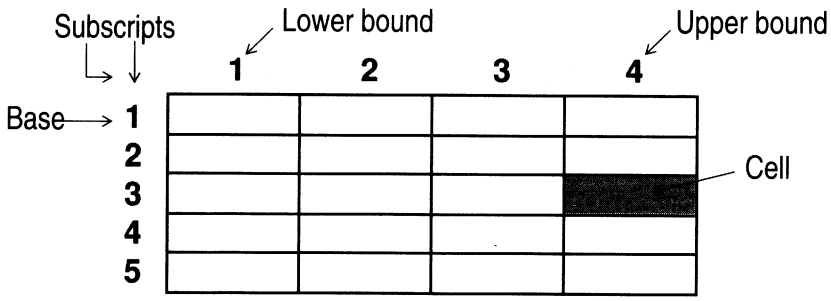
GDML

An *array* is a multi-dimensional data structure that holds data elements of the same type in subdivisions called cells.

You use an array when:

- The data elements naturally form an ordered set (or sequence)
- You want to control the set as a single field
- You want to be able to identify and access each element of the set individually

A two-dimensional array named *sample* is shown below.



Textbook Approach: Suppose you have an application that accepts fifteen input readings from ten machines five times a second. Standard relational theory produces a normalized record like the one shown below:

SQL
GDML

Time	Machine	Reading	Value
10:10:05.1	623	Temp1	135
10:10:05.1	624	Temp2	136
10:10:05.1	623	Temp3	102
⇓	⇓	⇓	⇓

This application stores 750 forty or fifty byte records per second. After 24 hours, you have 64,800,000 records, totalling over 3 billion bytes of data.

SQL GDML **Standard Work-Around:** An alternative is to store all the data for a single reading of a single machine in one record:

Time	Machine	Temp1	Temp2	Temp3 ⇒
10:10:05.1	623	135	102	157
10:10:05.1	624	136	98	175

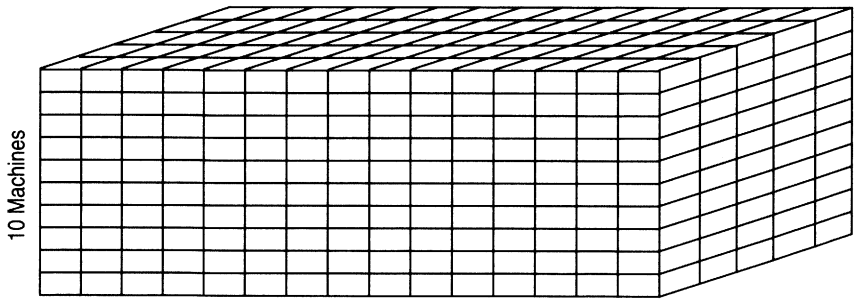
This application reduces the amount of data by storing ten 72 byte records per second or 4,320,000 records per day. However, each reading goes in a different field, causing long insert or store statements.

Additionally, analyzing the data is complicated because most programming languages are designed to manipulate arrays rather than records of numbers.

GDML **Array Solution:** The best solution for this applications is to use an array. In this example an array would capture one record per second. Each record consists of a time field and a three-dimensional array:

- One dimension represents the five readings per second within a reading
- A second dimension represents the machines
- A third dimension represents the fifteen readings

Not only does using an array compress 750 records into one, it also makes it easier to store and analyze data with a 3GL. For example, if you have a record whose values are out of bounds according to your criteria, the programming language interfaces allow you to retrieve that specific value.



Advantages of Arrays

GDML

With InterBase arrays you do not have to store array data in multiple two-dimensional records. Since InterBase arrays are multi-dimensional, you can store arrays as a whole in a single field. Accessing and retrieving data in arrays is then fast and simple.

Programming with Arrays

GDML

To access an array, you can refer to it from:

- A GDML record selection expression.
- Host language statements within a **for**, **modify**, or **store** loop.

Triggers

GDML

A *trigger* is a piece of code that executes a specific action when a record in a relation is stored, modified, or erased. Because triggers can access other relations, they can provide both referential integrity and application integrity.

You define triggers by using **gdef**.

You can use triggers to provide data integrity, post events, and maintain an audit log. Triggers are automatically executed, regardless of the interface through which you store, modify, or erase the associated

records. Therefore, you can be assured that any integrity you intend to protect will not be bypassed by any user or program.

Example

GDML To define a trigger that lets you store a new TOURISM record only if the record contains a valid state, type:

```
GDEF> define trigger store_tourism
CON> for cities
CON> pre store 0:
CON> begin
CON> if not any s in states with
CON> s.state = new.state
CON> abort 1;
CON> end;
CON> message 1:"State name is invalid."
CON> end_trigger;
```

GDML Event Alerters

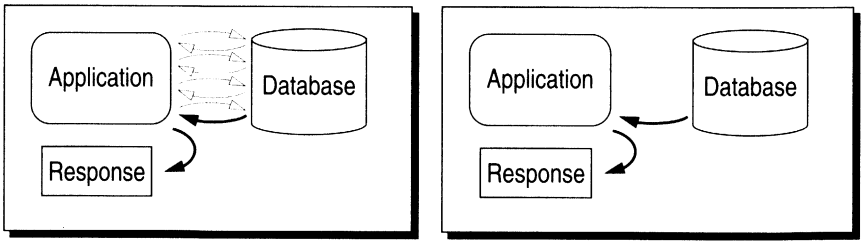
An *event alerter* is a mechanism for detecting and reporting changes in a database. They are designed to aid application developers in getting the right information to the right person when it's needed. Essentially, an *event alerter* is a signal the database sends to notify interested applications when a specific event has taken place.

Event alerters are useful anywhere you want a quick response to new information without wasting computer or network resources by polling see if a change has occurred. Once notified of an event, your program can respond to the information and initiate a task that the database cannot perform, such as running or controlling an external process.

An *event* can be any type of database insertion, deletion, or modification. For example, you can define an event that notifies all interested programs when an account is overdrawn.

You could use events in stock trading to notify traders when the price of a certain stock changes more than a specified amount. Or you could use events for process control, causing an alarm to sound when temperature fluctuates above or below a specific setting.

Polling for Database Change *InterBase Event Alerters*



There are two types of notification with events. An interested process can receive:

- Synchronous notification, so the process sleeps until an event occurs.
- Asynchronous notification, so the process continues running.

Advantages of Event Alerters

GDML

Event alerters:

- Reduce the overhead and network traffic that polling can cause
- Provide more real-time information than periodic polling
- Provide timely notification of significant changes to interested users in a heterogenous environment
- Do not miss intermittent events
- Reduce application code maintenance, because the same events are usable by multiple applications

Defining Event Alerters

GDML

Creating an event alerter mechanism involves:

- Defining an event in the database. You define an event by creating a trigger with **gdef**.
- Creating a program to wait for an event.

Example

GDML To define an event that notifies interested programs when an account is overdrawn, type:

```
GDEF> define trigger store_debit
CON> for accounts
CON> post store 0:
CON> if total d.amount of d in debits >
CON> total c.amount of c in credits
CON> then post overdrawn;
CON> end_trigger;
```

GDML Programming with Event Alerters

To program event alerters:

- Define a special trigger that posts an event notification.
- Create a program that registers interest in an event and requests either synchronous or asynchronous notification.

Where to Go From Here

GDML

For more information on advanced features, use the table below to direct you to the appropriate book in the InterBase documentation set.

To learn more about	Refer to the chapter on	In the
Arrays	Using Arrays	*Programmer's Guide*
Blobs	Using blob fields	*Programmer's Guide*
Blob filters	Using blob filters	*Programmer's Guide*
Casting	Retrieving data with GDML	*Programmer's Guide*
Event alerters	Programming with events	*Programmer's Guide*
Recursive Queries	Accessing data in Qli	*Qli Guide*
	Retrieving data with GDML	*Programmer's Guide*
Subqueries	Accessing data in Qli	*Qli Guide*
	Retrieving data with GDML	*Programmer's Guide*
User-defined functions	Creating user-defined	*Data Definition Guide*

6

Finishing Touches

Overview

GDML

A successful database application depends on the effective collection and retrieval of data. To simply these processes, InterBase provides forms for collecting and displaying data, and reports for formatting the output of database queries.

Forms

GDML

InterBase *forms* are screen images you use for collecting and displaying data. You use forms with:

- An interactive editor, **fred**, that provides menu support for building forms
- GDML statements for incorporating and manipulating forms and menus in GDML applications
- **Qli** statements for manipulating predefined forms and for using default forms

The following form shows the SKI_AREAS relation:

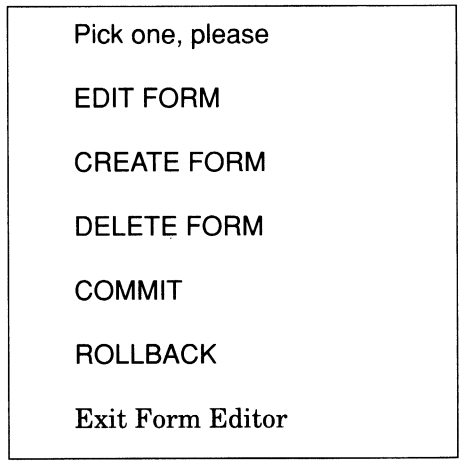
NAME	Birchwood Acres
TYPE	N
CITY	Groton
STATE	MA
<ENTER> or <R15> to continue, <R1> to stop	

The InterBase forms editor, **fred**, provides an interactive way to define forms.

Fred:

- Uses menus that let you generate a new form or revise an existing form.
- Automatically generates forms from a single menu choice.
- Edits forms to your application's requirements, letting you add fields to a form, move field labels and data input areas at will, and change the appearance of labels and input areas.
- Generates forms that reference multiple relations by choosing fields from another relation to include in the form.
- Stores forms in the same database as the relations it references and automatically stores the form in the database (or discards it if you want).

When you invoke **fred** it displays a menu listing the top level options:



- EDIT FORM lets you edit an existing form. You can add or delete fields, reformat the form, and even save it as a new form.
- CREATE FORM lets you create a new form based on an existing relation.
- DELETE FORM pops up a menu listing all of the forms in the current database. Select the name of the form you want to delete using the cursor keys, and press Enter. The form is deleted from the database.
- COMMIT writes all operations since the last commit or rollback to the database.
- ROLLBACK lets you undo changes to the database if you have not yet committed them.

GDML

Report Writer

InterBase's report writer allows you to format the output of database queries into a paper or screen report.

To create a report in qli, you

- Issue a **report** command
- Provide a record selection expression
- Provide a print list

Example

GDML

The following simple report displays all records in the SKI_AREAS relation:

```
QLI> report ski_areas sorted by state
CON>                  print city, state, name
CON> end_report

 CITY                 STATE   NAME
================      =====   ==============
Carlisle              MA      Great Farm
Groton                MA      Birchwood Acres
Waterville Valley     NH      Waterville Valley
New Ipswich           NH      Windblown
Mt. Washington        NH      Bretton Woods
Dixville Notch        NH      Wilderness
Stowe                 VT      Epson Hills
Stowe                 VT      Mt. Mansfield
Stowe                 VT      Trapp Family Lodge
```

The report writer has numerous options you can use to create complex reports. For example, a few of the elements you can control are:

- The length and width of the page
- A header at the beginning of the report

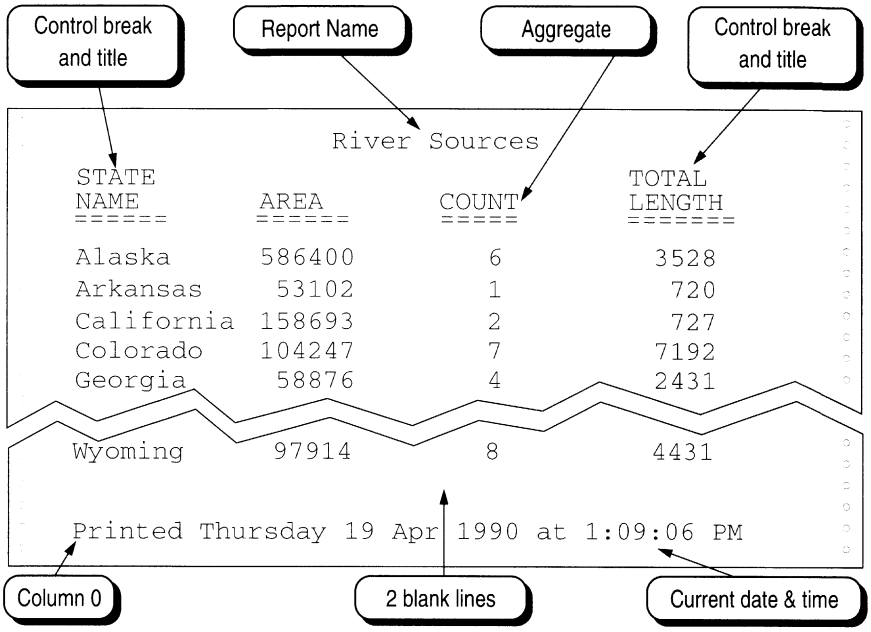
- A header and footer on every page
- Control groups (for example, cities by state)
- Aggregate values for control groups

Example

The following example shows a complex report. It uses many of the **report** command's options, including a report header, aggregate values, and page footers:

GDML

```
QLI>report states cross rivers -
CON>with state = source -
CON>sorted by state_name
CON>set report_name = "River Sources"
CON>at top of state_name print state_name
CON>at bottom of state_name print area,
CON>count,total length ("Total"/"Length")
CON>at bottom of page print skip 2,
CON>col 0,
CON>"Printed " | format "now" using
CON>w(9)bddbmmby(4)" at "tt:tt:ttbpp
CON>end_report
```



Options and Interfaces

Your application can be customized further through the use of various optional modules and interfaces that work with InterBase. The options are:

- Pictor
- Contessa
- Third-party Interfaces

Pictor

InterBase's point-and-click query tool, Pictor, allows workstation users to navigate around a database without the use of a DML. Using screen buttons and menus, users can select, print, change, and add data to existing databases.

Pictor is particularly useful in making off-the-cuff queries against unfamiliar databases. The database structure is visible on-screen, making it easy to determine what data is of interest. Pop-up menus, subwindows, and dialog boxes guide the user through the process of creating complex queries without requiring knowledge of any database language.

Infrequent users of InterBase will value the efficiency that Pictor gives.

Contessa

Contessa for InterBase is a full-featured application development environment that allows novice users as well as seasoned programmers to create sophisticated applications quickly and easily.

The Contessa environment provides a set of tools with which, by pointing and clicking with the mouse, graphical objects (lines, buttons, text fields, etc.) may be added to a graphical blackboard until a working interface is generated. Application objects can have executable "scripts" associated with them.

Contessa's tools are ideal for a broad range of applications such as:

- Rapid prototyping
- Data monitoring
- Customer services
- Financial modeling
- Diagnostics
- Process control

Third-Party Interfaces

InterBase works with a variety of commercial software products, including:

- Fourth-generation languages

- Application development environments
- Graphical display packages
- Statistical packages
- Industry-specific software

For full details on the list of InterBase software partners, contact an InterBase sales representative.

Where to Go From Here

For more information on the features discussed in this chapter, use the table below to direct you to the appropriate book in the InterBase documentation set.

To learn more about	Refer to the
Forms	*Forms Guide*
	Qli Guide
Fred	*Forms Guide*
	Qli Guide
Writing Reports	*Qli Guide*

7

Database Administration

Overview

InterBase provides several system utilities that make database administration and operations easy. Unlike other relational databases, InterBase does not require a traditional database administrator. Database administration and operations can be done by any individual who is familiar with InterBase.

In addition to supplying database administration utilities, InterBase also provides recovery mechanisms to help you recover from human disasters and system failures.

System Utilities

InterBase provides the following system utilities for performing database administration and operations:

- **Gbak**, the backup and restore utility
- **Gfix**, the database maintenance utility
- **Gcsu**, the central server management utility
- **Gltj**, the journal server utility

- **Gcon**, the console program used in communicating with the journal server
- **Grst**, the journal file restoration utility

Recovery Tools

InterBase provides recovery mechanisms that help you recover from a human disaster or a system failure. The recovery mechanisms are:

- Automatic recovery
- After-image journaling
- Disk shadowing

Routine system failures are generally handled without human intervention by the automatic recovery feature. More severe failures or user errors call for journaling or shadowing.

Automatic Recovery

InterBase's multi-generational record architecture ensures that the database is always available in an internally consistent state—even after a system crash. If the node supporting the database is brought up again after a power failure, for instance, the database can be opened immediately and processing resumed.

The database effectively returns to the state following the last committed transaction. Transactions left incomplete at the time of the crash are disregarded, and their associated generational records are marked for deletion.

Because it is a natural product of InterBase's architecture, automatic recovery is always available.

After-Image Journaling

After-image journaling is particularly useful when a program or user corrupts the database. Journaling allows you to return the database to any previous intact state. For example, if a user accidentally deleted all

customer records and committed the transaction, journaling would allow you to return the database to the state just before the accident. Journaling also allows for the recovery of data in the event the original database file becomes unreadable.

Once you enable journaling, changes to the database are automatically recorded in a journal file. InterBase supports journaling on Apollo, UNIX, and VMS systems.

As a recovery mechanism, journaling offers the following advantages:

- Provides recovery from user or media errors.
- Allows you to select the precise previous state to which the database should be returned.

Journaling is an optional part of InterBase. It must be activated before it protects your database.

Disk Shadowing

In the event that the disk or cpu serving a database becomes unusable, you may recover from a *disk shadow.* A disk shadow is a physical copy of a database stored in the same format as a database. Once enabled, a disk shadow maintains a duplicate, in-sync copy of the database it is shadowing, on another network node. Disk shadowing is supported on Apollo, UNIX and VMS systems.

As a recovery mechanism, disk shadowing offers the following advantages:

- Minimal impact on performance
- Quick recovery
- Predictable disk usage, identical to that of the original database

Disk shadowing is optional. It must be activated before it protects your database.

Where to Go From Here

For more information on InterBase utilities and recovery mechanisms, use the table below to direct you to the appropriate book in the InterBase documentation set.

To learn more about	Refer to the chapter on	In the
Using **gbak**	Backup and Recovery	*Database Operations*
Gbak syntax	Reference information	*Database Operations*
Using **gfix**	Database Maintenance	*Database Operations*
Gfix syntax	Reference information	*Database Operations*
Automatic recovery	Database Maintenance	*Database Operations*
Journaling	Journaling	*Database Operations*
Disk shadowing	Disk Shadowing	*Database Operations*

A

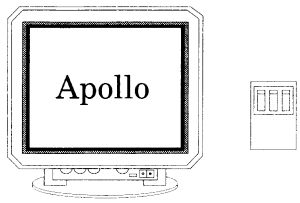
Supported Systems

InterBase supports the following platforms:

- DEC
- HP/Apollo
- HP
- Sun
- IBM
- Motorola
- Silicon Graphics
- Data General
- SCO

The following tables list the machines, operating system, communication protocols, and languages for each of the supported platforms.

HP/Apollo Platforms



Platforms

- DN3XXX
- DN4XXX
- DN10000
- HP 9000/400

Operating systems

- SR10.2+

Communication protocols

- Apollo MBX
- TCP/IP

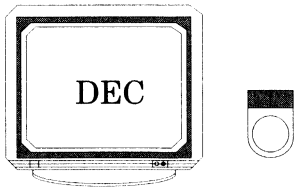
Languages DN3XXX,DN4XXX, HP 9000/400

- Ada
- C
- FORTRAN
- Pascal
- C++

Languages DN10000

- C
- Pascal
- FORTRAN
- Ada

DEC Platforms



Platforms

- DECstation
- VAXstation

Operating systems

- VAX/VMS 5.0+
- Ultrix 4.2
- VAX/ULTRIX 3.1
- VAX/VMS V5.3-5

Communication protocols

- DECnet
- TCP/IP

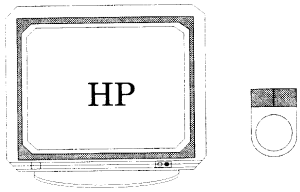
Languages (VMS)

- Ada
- Basic
- C
- COBOL
- FORTRAN
- Pascal
- PL/1

Languages (Ultrix)

- C

HP Platforms



Platforms

- 9000 Series 300
- 9000 series 400
- 9000 Series 600
- 9000 Series 800
- 9000/700

Operating Systems

- HP-UX 7.3
- HP-UX 8.0.5

Communication protocols

- TCP/IP

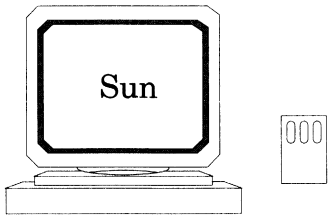
Languages 9000/300, 400

- C
- C++
- FORTRAN

Languages 9000/600, 800, 700

- C
- FORTRAN

Sun Platforms



Platforms

- Sun-3
- Sun-4
- SPARCstation
- Sun MPs

Operating Systems

- SunOS 4.1
- SunOS 4.1.1

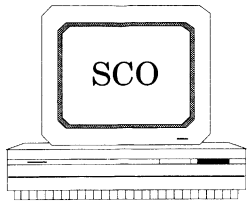
Communication protocols

- TCP/IP

Languages

- Ada
- C
- C++
- FORTRAN

Santa Cruz Operation Platforms



Platforms

- 386compatible
- 486compatible

Operating Systems

- SCO UNIX 3.2.2

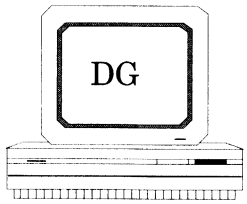
Communication protocols

- TCP/IP

Languages

- C

Data General AViiON Platforms



Platforms

- Data General AViiON

Operating Systems

- DG-UX 5.4

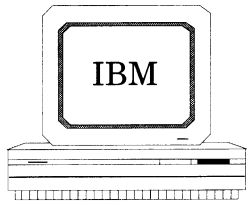
Communication protocols

- TCP/IP

Languages

- C
- C++
- FORTRAN

IBM Platforms



Platforms

- 386 Compatible
- RS/6000

Operating Systems

- SCO UNIX 3.2.2
- AIX 3.1

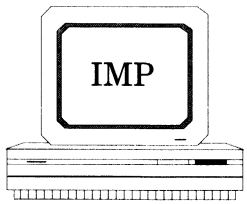
Communication protocols

- TCP/IP

Languages

- C

Motorola Platforms



Platforms

- Motorola IMP
- Motorola Delta

Operating Systems

- UNIPLUS+ V3.1

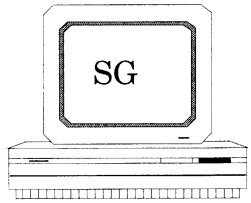
Communication protocols

- TCP/IP

Languages

- C

Silicon Graphics Platforms



Platforms

- Silicon Graphics

Operating Systems

- IRIX 3.3

Communication protocols

- TCP/IP

Languages

- C
- FORTRAN

B

Specifications

General

Relational DBMS
ANSI Level II SQL
Multi-user
Transaction-based
Forms interface
Active data dictionary
Views and virtual fields
Automatic data validation and triggers
On-line restructuring
Multiple databases per transaction
Distributed read-write access
DSRI/OSRI compatibility

Operations

Relational restrict, project, join, union
Global and scalar aggregates
Compare, string search, and existential operators
Nested and recursive requests
Storage and retrieval of very large objects
Event alerters notify distributed applications of database changes

Concurrency Control

Multi-user, concurrent update

Atomic transactions
Two compatible levels of concurrency:
    Serializable
    High concurrency
Automatic two-phase commit and rollback
Multiple database transactions
Multiple parallel transactions per process

Availability

High multi-user throughput
No performance degradation during:
    Online backup
    Online metadata changes
    Rollback recovery
Instant system availability after hardware crash

Database Recovery

Automatic cooperative rollback
After-image journaling
Validate and repair utility
"Careful write" with precedence list
Shadowing

Datatypes

Short and long integer with decimal scale
Single and double precision floating
Fixed, null-terminated, and varying strings
Maximum string length 32Kb
Dates (100 AD to 5941 AD)
Blobs (arbitrarily large, semi-structured objects)
Multi-dimensional arrays
Null or missing values

Integrity

Unique key
Dictionary-based data validation criteria
Triggers enforce integrity and business rules
    Cannot be bypassed

May cause other triggers to activate (cascade)
Multiple triggers per database action controlled independently
User-specified firing sequence
Pre- and post-action specification
Consistent behavior under transaction control

Security

Secure fields, relations, views, and databases
By security class, user, group, or view

Dictionary

Active data dictionary
Metadata updates concurrent with other transactions
Dictionary manages:
- User-defined functions
- Trigger procedures
- Blob filters (translation code)

Interfaces

Host language preprocessor
Interactive application language
Forms
Icon-based interactive query and update (Pictor: add-on module)
Architecture-independent, message-based low-level interface for VAR and third-party system development

Host Language Preprocessor

C, FORTRAN, Pascal, COBOL, BASIC, PL/I, and Ada
ANSI Level II SQL and DEC Rdb compatibility
Intermixed SQL, GDML, and lower level calls (if needed)

Interactive 4GL

SQL and proprietary language (GDML)
Query and update
On-line hierarchical help
Automatic and programmable prompts
Nested command procedures
Automatic and programmable forms

Forms Interface

Forms editor
Embedded DML forms language
Forms in 4GL

Restructuring

On line with active transactions
Add field, relation, index, trigger
Drop field, relation, index, trigger
Change field length, datatype
Change trigger actions
Add or drop fields in relation context
Add or drop keys from index

Remote Database Access

Transparent to host program
Architecture-independent protocol
Network access between different machine architectures
Client/server, multi-threaded, multi-server (peer-to-peer)
Independent of communication system

Utilities

Data definition utility
Host language preprocessor
Interactive application language
Forms editor
Administration and operations utililities

Optimizations

B-tree indexes (single and compound key)
Arbitrary key positions in record
Index selection
Index combination (bitmap techniques)
Join order selection
Fast load index creation

Data Compression

Data run-length encoded
Prefix and tail compression in index

Specifications

Access method: ~250 KB memory
Complete system, including utilities & examples typically 4–6 MB disk
Record size (excl. blob): 65,000 bytes
Field size (excl. blob): 32,000 bytes
Fields per record: 16,000
Index key: 255 bytes
Blob size: no limit
Records per relation: no limit
Indexes per relation: 64

Documentation

Database Operations
Data Definition Guide
DDL Reference
DSQL Programmer's Guide
Forms Guide
Getting Started with InterBase
Programmer's Guide
Programmer's Reference
Qli Guide
Qli Reference
Sample Programs
Master Index
Platform-specific installation instructions and information
Pictor User's Guide (optional)
Access Method Reference Manual (optional)

Support and Training

One year of update support is included in the purchase price for InterBase. Technical support staff is available via hot-line telephone as an option.

Interbase Software provides courses at training centers on the east and west coasts, and on-site at a customer's location. Contact Interbase for more information.

C

Language Features

This table indicates which InterBase features are supported by ANSI Level II SQL, and which by GDML.

Feature	Database Language	
Defining		
Databases	SQL	GDML
Relations	SQL	GDML
Local Fields	SQL	GDML
Global Fields		GDML
Computed Fields		GDML
Views	SQL	GDML
Indexes	SQL	GDML
Triggers		GDML
Transactions		
Multi-Generational Records	SQL	GDML
Consistency Mode		GDML
Concurrency Mode	SQL	GDML

Feature	Database Language	
Data Access		
Retrieving Records	SQL	GDML
Modifying Records	SQL	GDML
Erasing Records	SQL	GDML
Adding Records	SQL	GDML
Advanced Features		
Casting		GDML
Subqueries		GDML
Recursive Queries		GDML
User-Defined Functions		GDML
Events		GDML
Blob & Blob Filters		GDML
Security	SQL	GDML
Arrays		GDML
Finishing Touches		
Forms		GDML
Report Writer		GDML
Interface to 3GL Programs	SQL	GDML

D

Interbase Offices

Headquarters:

Borland International Inc.
1800 Green Hills Road
P. O. BOX 660001
Scotts Valley, CA 95067-0001
Phone: 408-438-8400

Northeast Office

209 Burlington Road
Bedford, MA 01730
Phone: 617-275-3222
FAX: 617-271-0221

Mid-Atlantic

1420 Spring Hill Road
Suites 460-470
McLean, VA 22102
Phone: 703 -448-2300
Fax: 703-448-2308

Southwest

20101 Hamilton Avenue
Torrance, CA 90502
Phone: 310-538-7458

Northwest

1611 116th Avenue NE
Suite 22
Bellevue, WA 98004
Phone: 206-646-4975
FAX: 206-646-3020

Index

T

U

V