Database objects and concepts

A database consists of a variety of database objects, such as tables, views, domains, stored procedures, and triggers. Database objects contain all the information about the structure of the database and the data. Because they encapsulate information about the data, database objects are referred to as metadata.

The following sections provide an overview of the InterBase database objects and concepts:

Tables

Relational databases store all their data in tables. A table is a data structure consisting of an unordered set of horizontal rows, each containing the same number of vertical columns. The intersection of an individual row and column is a field that contains a specific piece of information. Much of the power of relational databases comes from defining the relations among the tables.

InterBase stores information about metadata in special tables, called system tables. System tables have predefined columns that store information about the type of metadata in that table. All system tables begin with "RDB$". An example of a system table is RDB$RELATIONS, which stores information about each table in the database.

System tables have the same structure as user-defined tables and are stored in the same database as the user-defined tables. Because the metadata, user-defined tables, and data are all stored in the same database file, each database is a complete unit and can be easily transported between machines.

System tables can be modified like any other database tables. Unless you understand all the interrelationships between the system tables, however, modifying them directly may adversely affect other system tables and disrupt your database.

For a complete discussion of tables, see the Data Definition Guide.
For a complete description of the InterBase system tables and views, see the Language Reference.

Columns

Creating a table mainly involves defining the columns in the table. The main attributes of a column include:

- The name of the column
- Data type of the column or the domain on which it is based
- Whether or not the column is allowed to be NULL
- Optional referential integrity constraints

For a complete discussion on columns, see the Data Definition Guide.

Data types

Data is stored in a predefined format called a data type. Data types can be classified into four categories: numeric, character, date, and BLOB. Numeric data types handle everything from integers to double-precision floating point values. Character data types hold strings of text. Date data types are used for storing date and time values.

While numeric, character, and date are standard data types, the BLOB data type deserves special mention.

BLOB data types

InterBase supports a binary large object (BLOB) data type, that can hold data of unlimited size. The BLOB is an extension of the standard relational model, which ordinarily provides only for data types of fixed width.
The BLOB data type is analogous to a flat file because BLOB data can be stored in any format (for example, binary or ASCII). A BLOB, however, is not a separate file. BLOB data is stored in the database with all other data. Because BLOB columns often contain large, variable amounts of data, BLOB columns are stored and retrieved in segments.

Conversion of BLOB data to other data types in InterBase is not directly supported, but on some platforms, BLOB filters can translate BLOB data from one BLOB format to another.

For a complete discussion of data types, see the Programmer's Guide.

Domains

In addition to explicitly stating the data type of columns, InterBase allows global column definitions, or domains, upon which column definitions can be based. A domain specifies a data type, and a set of column attributes and constraints. Subsequent table definitions can use the domain to define columns.

For a complete discussion on domains, see the Data Definition Guide.

Referential integrity constraints

InterBase allows you to define referential integrity rules for a column, called referential integrity constraints. Integrity constraints govern column-to-table and table-to-table relationships and validate data entries. They are implemented through primary keys, foreign keys, and check constraints. Basically, a primary key is a column (or group of columns) that uniquely identifies a row in a table. A foreign key is a column whose value must match a value of a column in another table. A check constraint limits data entry to a specific range or set of values.

For example, an EMPLOYEE table could be defined to have a foreign key column named DEPT_NO that is defined to match the department number column in a DEPARTMENT table. This would ensure that each employee in the EMPLOYEE table is assigned to an existing department in the DEPARTMENT table.

For more information see the Data Definition Guide.

Indexes

Indexes are mechanisms for improving the speed of data retrieval. An index identifies columns that can be used to retrieve and sort rows efficiently in the table. It provides a means to scan only a specific subset of the rows in a table, improving the speed of data access.

InterBase automatically defines unique indexes for a table's PRIMARY KEY and FOREIGN KEY constraints.

For a complete discussion of indexes, see the Data Definition Guide.

Views

A view is a virtual table that is not physically stored in the database, but appears exactly like a "real" table. A view can contain data from one or more tables or other views and is used to store often-used queries or query sets in a database.

Views can also provide a limited means of security, because they can provide users access to a subset of available data while hiding other related and sensitive data.

For a complete discussion of views, see the Data Definition Guide.

Stored procedures

A stored procedure is a self-contained program written in InterBase procedure and trigger language, an extension of SQL. Stored procedures are part of a database's metadata. Stored procedures can receive input parameters from and return values to applications and can be executed explicitly from applications, or substituted for a table name in a SELECT statement.

Stored procedures provide:

- Modular design: stored procedures can be shared by applications that access the same database, eliminating duplicate code, and reducing the size of applications.
- Streamlined maintenance: when a procedure is updated, the changes are automatically reflected in all applications that use it without the need to recompile and relink them. They are compiled and optimized only once for each client.
- Improved performance: especially for remote client access. Stored procedures are executed by the server, not the client, which reduces network traffic.
- Secure access to data: you can design your stored procedures to access privileged data in ways you specify.

For a complete discussion of stored procedures, see the Data Definition Guide.

Triggers

A trigger is a self-contained routine associated with a table or view that automatically performs an action when a row in the table or view is inserted, updated, or deleted.

Triggers can provide:

- Automatic enforcement of data restrictions to ensure that users enter only valid values into columns.
- Reduced application maintenance, because changes to a trigger are automatically reflected in all applications that use the associated table without the need to recompile and relink them.
- Automatic logging of changes to tables. An application can keep a running log of changes with a trigger that fires whenever a table is modified.

When a database operation invokes a trigger, it has immediate access to data being stored, modified, or erased. The trigger may also access data in other tables. Using the available data, you can design the trigger to:

- Cancel an operation, possibly with an error message.
- Set values in the accessed record.
- Insert, update, or delete rows in other tables.

For a complete discussion of triggers, see the Data Definition Guide.

Generators

A generator is a mechanism that creates a unique, sequential number that is automatically inserted into a column by the database when SQL data manipulation operations such as INSERT or UPDATE occur. Generators are typically used to produce unique values that can be inserted into a column that is used as a PRIMARY KEY. Any number of generators can be defined for a database, as long as each generator has a unique name.

For a complete discussion of generators, see the Data Definition Guide.

Security

SQL security is controlled at the table level with access privileges, a list of operations that a user is allowed to perform on a given table or view. The GRANT statement assigns access privileges for a table or view to specified users or procedures. The REVOKE statement removes previously granted access privileges.

For a complete discussion of security, see the Operations Guide, Chapter 6: “Database Security.”

Database design

Design framework

The following steps provide a framework for designing a database:

1. Collect and analyze the real-world objects that you want to model in your database. Organize the objects into entities and attributes and make a list.
2. Map the entities and attributes to InterBase tables and columns.
3. Determine an attribute that will uniquely identify each object.
4. Develop a set of rules that govern how each table is accessed, populated, and modified.
5. Establish relationships between the objects (tables and columns).

Collecting and analyzing objects

Before designing the database objects—the tables and columns—you need to organize and analyze the real-world data on a conceptual level. There are four primary goals:

- Identify the major functions and activities of your organization. For example: hiring employees, shipping products, ordering parts, processing paychecks, and so on.
- Identify the objects of those functions and activities. Building a business operation or transaction into a sequence of events will help you identify all of the entities and relationships the operation entails. For example, when you look at a process like "hiring employees," you can immediately identify entities such as the JOB, the EMPLOYEE, and the DEPARTMENT.
- Identify the characteristics of those objects. For example, the EMPLOYEE entity might include such information as EMPLOYEE_ID, FIRST_NAME, LAST_NAME, JOB, SALARY, and so on.
- Identify certain relationships between the objects. For example, how do the EMPLOYEE, JOB, and DEPARTMENT entities relate to each other? The employee has one job title and belongs to one department, while a single department has many employees and jobs. Simple graphical flow charts help to identify the relationships.

Mapping entities and attributes

Based on the requirements that you collect, identify the objects that need to be in the database—the entities and attributes. An entity is a type of person, object, or thing that needs to be described in the database. It may be an object with a physical existence, like a person, a car, or an employee, or it may be an object with a conceptual existence, like a company, a job, or a project. Each entity has properties, called attributes, that describe it. For example, suppose you are designing a database that must contain information about each employee in the company, departmental-level information, information about current projects, and information about customers and sales.

The example below shows how to create a list of entities and attributes that organizes the required data.

Entities	Attributes
EMPLOYEE	Employee Number Last Name First Name Department Number Job Code Phone Extension Salary
DEPARTMENT	Department Number Department Name Department Head Name Dept Head Employee Num Budget Location Phone Number
PROJECT	Project ID Project Name Project Description Team Leader Product
CUSTOMER	Customer Number Customer Name Contact Name Phone Number Address
SALES	PO Number Customer Number Sales Rep Order Date Ship Date Order Status

By listing the entities and associated attributes this way, you can begin to eliminate redundant entries. Do the entities in your list work as tables? Should some columns be moved from one group to another? Does the same attribute appear in several entities? Each attribute should appear only once, and you need to determine which entity is the primary owner of the attribute. For example, DEPARTMENT HEAD NAME should be eliminated because employee names (FIRST NAME and LAST NAME) already exist in the EMPLOYEE entity. DEPARTMENT HEAD EMPLOYEE NUM can then be used to access all of the employee-specific information by referencing EMPLOYEE NUMBER in the EMPLOYEE entity.

Designing tables

In a relational database, the database object that represents a single entity is a table, which is a two-dimensional matrix of rows and columns. Each column in a table represents an attribute. Each row in the table represents a specific instance of the entity. After you identify the entities and attributes, create the data model, which serves as a logical design framework for creating your InterBase database. The data model maps entities and attributes to InterBase tables and columns, and is a detailed description of the database—the tables, the columns, the properties of the columns, and the relationships between tables and columns.

The example below shows how the EMPLOYEE entity from the entities/attributes list has been converted to a table.

EMP_NO	LAST_NAME	FIRST_NAME	DEPT	JOB_CODE	PHONE_EXT	SALARY
24	Smith	John	100	Eng	4968	64000
48	Carter	Catherine	900	Sales	4967	72500
36	Smith	Jane	600	Admin	4800	37500

Each row in the EMPLOYEE table represents a single employee. EMP_NO, LAST_NAME, FIRST_NAME, DEPT_NO, JOB_CODE, PHONE_EXT, and SALARY are the columns that represent employee attributes. When the table is populated with data, rows are added to the table, and a value is stored at the intersection of each row and column, called a field. In the EMPLOYEE table, "Smith" is a data value that resides in a single field of an employee record.

Determining unique attributes

One of the tasks of database design is to provide a way to uniquely identify each occurrence or instance of an entity so that the system can retrieve any single row in a table. The values specified in the table's primary key distinguish the rows from each other. A PRIMARY KEY or UNIQUE constraint ensures that values entered into the column or set of columns are unique in each row. If you try to insert a value in a PRIMARY KEY or UNIQUE column that already exists in another row of the same column, InterBase prevents the operation and returns an error.

For example, in the EMPLOYEE table, EMP_NO is a unique attribute that can be used to identify each employee in the database, so it is the primary key. When you choose a value as a primary key, determine whether it is inherently unique. For example, no two social security numbers or driver's license numbers are ever the same. Conversely, you should not choose a name column as a unique identifier due to the probability of duplicate values. If no single column has this property of being inherently unique, then define the primary key as a composite of two or more columns which, when taken together, are unique.

A unique key is different from a primary key in that a unique key is not the primary identifier for the row, and is not typically referenced by a foreign key in another table. The main purpose of a unique key is to force a unique value to be entered into the column. You can have only one primary key defined for a table, but any number of unique keys.

Developing a set of rules

When designing a table, you need to develop a set of rules for each table and column that establishes and enforces data integrity. These rules include:

- Selecting a data type.
- Choosing international character sets.
- Specifying domains.
- Setting default values and NULL status.
- Defining constraints.

Selecting a data type

Once you have chosen a given attribute as a column in the table, you can choose a data type for the attribute. The data type defines the set of valid data that the column can contain. The data type also determines which operations can be performed on the data, and defines the disk space requirements for each data item.

The general categories of SQL data types include:

- Character data types.
- Whole number (integer) data types.
- Fixed and floating decimal data types.
- A DATE data type to represent date and time.
- A BLOB data type to represent unstructured binary data, such as graphics and digitized voice.

For a complete discussion of data types, see Data Defintion Guide.

Choosing international character sets

When you create the database, you can specify a default character set. A default character set determines:

- What characters can be used in CHAR, VARCHAR, and BLOB text columns.
- The default collation order that is used in sorting a column.

The collation order determines the order in which values are sorted. The COLLATE clause of CREATE TABLE allows users to specify a particular collation order for columns defined as CHAR and VARCHAR text data types. You must choose a collation order that is supported for the column's given character set. The collation order set at the column level overrides a collation order set at the domain level.

Choosing a default character set is primarily intended for users who are interested in providing a database for international use. For example, the following statement creates a database that uses the ISO8859_1 character set, typically used to support European languages:

```
CREATE DATABASE "employee.gdb"
DEFAULT CHARACTER SET ISO8859_1;
```

You can override the database default character set by creating a different character set for a column when specifying the data type. The data type specification for a CHAR, VARCHAR, or BLOB text column definition can include a CHARACTER SET clause to specify a particular character set for a column. If you do not specify a character set, the column assumes the default database character set. If the database default character set is subsequently changed, all columns defined after the change have the new character set, but existing columns are not affected.

If you do not specify a default character set at the time the database is created, the character set defaults to NONE. This means that there is no character set assumption for the columns; data is stored and retrieved just as it was originally entered. You can load any character set into a column defined with NONE, but you cannot load that same data into another column that has been defined with a different character set. No transliteration will be performed between the source and the destination character sets.

Specifying domains

When several tables in the database contain columns with the same definitions and data types, you can create domain definitions and store them in the database. Users who create tables can then reference the domain definition to define column attributes locally.

For a complete discussion on domains, and instructions on how to define domain-based columns, see the Data Definition Guide.

Setting default values and NULL status

You can set an optional default value that is automatically entered into a column when you do not specify an explicit value. Defaults can save data entry time and prevent data entry errors. For example, a possible default for a DATE column could be today's date, or in a Y/N flag column for saving changes, "Y" could be the default. Column-level defaults override defaults set at the domain level.

Assign a NULL status to insert a NULL in the column if the user does not enter a value. Assign NOT NULL to force the user to enter a value, or to define a default value for the column. NOT NULL must be defined for PRIMARY KEY and UNIQUE key columns.

Defining constraints

Integrity constraints are rules that govern column-to-table and table-to-table relationships, and validate data entries. They span all transactions that access the database and are automatically maintained by the system. Integrity constraints can be applied to an entire table or to an individual column. A PRIMARY KEY or UNIQUE constraint guarantees that no two values in a column or set of columns will ever be the same.
Data values that uniquely identify rows (a primary key) in one table can also appear in other tables. A foreign key is a column or set of columns in one table that contain values that match a primary key in another table.

CHECK constraints

Along with preventing the duplication of values using UNIQUE and PRIMARY KEY constraints, you can specify another type of data entry validation. A CHECK constraint places a condition or requirement on the data values in a column at the time the data is entered. The CHECK constraint enforces a search condition that must be true in order to insert into or update the table or column.

Establishing relationships between objects

The relationship between tables and columns in the database must be defined in the design. For example, how are employees and departments related? An employee can have only one department (a one-to-one relationship), but a department has many employees (a one-to-many relationship). How are projects and employees related? An employee can be working on more than one project, and a project can include several employees (a many-to-many relationship). Each of these different types of relationships has to be modeled in the database.

The relational model represents one-to-many relationships with primary key/foreign key pairings. Refer to the following two tables. A project can include many employees, so to avoid duplication of employee data, the PROJECT table can reference employee information with a foreign key. TEAM_LEADER is a foreign key referencing the primary key, EMP_NO, in the EMPLOYEE table.

PROJ_ID	TEAM_LEADER	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	44	Automap	blob data	hardware
VBASE	47	Video database	blob data	software
HWRII	24	Translator upgrade	blob data	software

EMP_NO	LAST_NAME	FIRST_NAME	DEPT	JOB_CODE	PHONE_EXT	SALARY
24	Smith	John	100	Eng	4968	64000
48	Carter	Catherine	900	Sales	4967	72500
36	Smith	Jane	600	Admin	4800	37500

Enforcing referential integrity

The primary reason for defining foreign keys is to ensure that the integrity of the data is maintained when more than one table references the same data—rows in one table must always have corresponding rows in the referencing table. InterBase enforces referential integrity in the following ways:

- Before a foreign key can be added, the unique or primary keys that the foreign key references must already be defined.
- If information is changed in one place, it must be changed in every other place that it appears. For example, to change a value in the EMP_NO column of the EMPLOYEE table (the primary key), that value must also be updated in the TEAM_LEADER column of the PROJECT table (the foreign key).
- If a user could delete a row containing a primary key in one table, the meaning of any rows in another table that contain that value as a foreign key would be lost. To preserve referential integrity, InterBase prevents anyone from deleting values from a primary key that have matching foreign keys in another referencing table. The user must first delete all foreign key rows before deleting the primary key to which they refer.
- InterBase also prevents users from adding a value in a column defined as a foreign key that does not reference an existing primary key value. For example, to change a value in the TEAM_LEADER column of the PROJECT table, that value must first be updated in the EMP_NO column of the EMPLOYEE table.

How to normalize a database

After your tables, columns, and keys are defined, look at the design as a whole and analyze it using normalization guidelines in order to find logical errors. As mentioned in the overview, normalization involves breaking down larger tables into smaller ones in order to group data together that is naturally related.

Note: A detailed explanation of the normal forms are out of the scope of this document. There are many excellent books on the subject on the market.

When a database is designed using proper normalization methods, data related to other data does not need to be stored in more than one place—if the relationship is properly specified. The advantages of storing the data in one place are:

- The data is easier to update or delete.
- When each data item is stored in one location and accessed by reference, the possibility for error due to the existence of duplicates is reduced.
- Because the data is stored only once, the possibility for introducing inconsistent data is reduced.

In general, the normalization process includes:

- Eliminating repeating groups.
- Removing partially-dependent columns.
- Removing transitively-dependent columns.

When to break the rules

You should try to correct any normalization violations, or else make a conscious decision to ignore them in the interest of ease of use or performance. Just be sure that you understand the design trade-offs that you are making, and document your reasons. It may take several iterations to reach a design that is a desirable compromise between purity and reality, but this is the heart of the design process.

For example, suppose you always want data about dependents every time you look up an employee, so you decide to include DEP1_NAME, DEP1_BIRTHDATE, and so on for DEP1 through DEP30, in the EMPLOYEE table. Generally speaking, that is terrible design, but the requirements of your application are more important than the abstract purity of your design. In this case, if you wanted to compute the average age of a given employee's dependents, you would have to explicitly add field values together, rather than asking for a simple average. If you wanted to find all employees with a dependent named "Jennifer," you would have to test 30 fields for each employee instead of one. If those are not operations that you intend to perform, then go ahead and break the rules. If the efficiency attracts you less than the simplicity, you might consider defining a view that combines records from employees with records from a separate DEPENDENTS table.

While you are normalizing your data, remember that InterBase offers direct support for array columns, so if your data includes, for example, hourly temperatures for twenty cities for a year, you could define a table with a character column that contains the city name, and a 24 by 366 matrix to hold all of the temperature data for one city for one year. This would result in a table containing 20 rows (one for each city) and two columns, one NAME column and one TEMP_ARRAY column. A normalized version of that record might have 366 rows per city, each of which would hold a city name, a Julian date, and 24 columns to hold the hourly temperatures.

Eliminating repeating groups

When a field in a given row contains more than one value for each occurrence of the primary key, then that group of data items is called a repeating group. This is a violation of the first normal form, which does not allow multi-valued attributes.

Refer to the DEPARTMENT table. For any occurrence of a given primary key, if a column can have more than one value, then this set of values is a repeating group. Therefore, the first row, where DEPT_NO = "100," contains a repeating group in the DEPT_LOCATIONS column.

DEPT_NO	DEPARTMENT	HEAD_DEPT	BUDGET	DEPT_LOCATIONS
100	Sales	000	1000000	Monterey, Santa Cruz, Salinas
600	Engineering	120	1100000	San Francisco
900	Finance	000	400000	Monterey

In the next example, even if you change the attribute to represent only one location, for every occurrence of the primary key "100," all of the columns contain repeating information except for DEPT_LOCATION, so this is still a repeating group.

DEPT_NO	DEPARTMENT	HEAD_DEPT	BUDGET	DEPT_LOCATION
100	Sales	000	1000000	Monterey
100	Sales	000	1000000	Santa Cruz
600	Engineering	120	1100000	San Francisco
100	Sales	000	1000000	Salinas

To normalize this table, we could eliminate the DEPT_LOCATION attribute from the DEPARTMENT table, and create another table called DEPT_LOCATIONS. We could then create a primary key that is a combination of DEPT_NO and DEPT_LOCATION. Now a distinct row exists for each location of the department, and we have eliminated the repeating groups.

DEPT_NO	DEPT_LOCATION
100	Monterey
100	Santa Cruz
600	San Francisco
100	Salinas

Removing partially-dependent columns

Another important step in the normalization process is to remove any non-key columns that are dependent on only part of a composite key. Such columns are said to have a partial key dependency. Non-key columns provide information about the subject, but do not uniquely define it.

For example, suppose you wanted to locate an employee by project, and you created the PROJECT table with a composite primary key of EMP_NO and PROJ_ID.

EMP_NO	PROJ_ID	LAST_NAME	PROJ_NAME	PROJ_DESC	PRODUCT
44	DGPII	Smith	Automap	blob data	hardware
47	VBASE	Jenner	Video database	blob data	software
24	HWRII	Stevens	Translator upgrade	blob data	software

The problem with this table is that PROJ_NAME, PROJ_DESC, and PRODUCT are attributes of PROJ_ID, but not EMP_NO, and are therefore only partially dependent on the EMP_NO/PROJ_ID primary key. This is also true for LAST_NAME because it is an attribute of EMP_NO, but does not relate to PROJ_ID. To normalize this table, we would remove the EMP_NO and LAST_NAME columns from the PROJECT table, and create another table called EMPLOYEE_PROJECT that has EMP_NO and PROJ_ID as a composite primary key. Now a

unique row exists for every project that an employee is assigned to.

Removing transitively-dependent columns

The third step in the normalization process is to remove any non-key columns that depend upon other non-key columns. Each non-key column must be a fact about the primary key column. For example, suppose we added TEAM_LEADER_ID and PHONE_EXT to the PROJECT table, and made PROJ_ID the primary key. PHONE_EXT is a fact about TEAM_LEADER_ID, a non-key column, not about PROJ_ID, the primary key column.

PROJ_ID	TEAM_LEADER_ID	PHONE_EXT	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	44	4929	Automap	blob data	hardware
VBASE	47	4967	Video database	blob data	software
HWRII	24	4668	Translator upgrade	blob data	software

To normalize this table, we would remove PHONE_EXT, change TEAM_LEADER_ID to TEAM_LEADER, and make TEAM_LEADER a foreign key referencing EMP_NO in the EMPLOYEE table.

PROJ_ID	TEAM_LEADER	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	44	Automap	blob data	hardware
VBASE	47	Video database	blob data	software
HWRII	24	Translator upgrade	blob data	software

EMP_NO	LAST_NAME	FIRST_NAME	DEPT_NO	JOB_CODE	PHONE_EXT	SALARY
24	Smith	John	100	Eng	4968	64000
48	Carter	Catherine	900	Sales	4967	72500
36	Smith	Jane	600	Admin	4800	37500

How to choose indexes

Once you have your design, you need to consider what indexes are necessary. The basic trade-off with indexes is that more distinct indexes make retrieval by specific criteria faster, but updating and storage slower. One optimization is to avoid creating several indexes on the same column. For example, if you sometimes retrieve employees based on name, department, badge number, or department name, you should define one index for each of these columns. If a query includes more than one column value to retrieve, InterBase will use more than one index to qualify records. In contrast, defining indexes for every permutation of those three columns will actually slow both retrieval and update operations.

When you are testing your design to find the optimum combination of indexes, remember that the size of the tables affects the retrieval performance significantly. If you expect to have tables with 10,000 to 100,000 records each, do not run tests with only 10 to 100 records.

Another factor that affects index and data retrieval times is page size. By increasing the page size, you can store more records on each page, thus reducing the number of pages used by indexes. If any of your indexes are more than 4 levels deep, you should consider increasing the page size. If indexes on volatile data (data that is regularly deleted and restored, or data that has index key values that change frequently) are less than 3 levels deep, you should consider reducing your page size. In general, you should use a page size larger than your largest record, although InterBase's data compression will generally shrink records that contain lots of string data, or lots of numeric values that are 0 or NULL. If your records have those characteristics, you can probably store records on pages which are 20% smaller than the full record size. On the other hand, if your records are not compressible,

you should add 5% to the actual record size when comparing it to the page size.

InterBase-specific design suggestions

Increasing cache size

When InterBase reads a page from the database onto disk, it stores that page in its cache, which is a set of buffers that are reserved for holding database pages. Ordinarily, the default cache size of 255 buffers is adequate (though this default may be different on different platforms). If your application includes joins of 5 or more tables, InterBase automatically increases the size of the cache. If your application is well localized, that is, it uses the same small part of the database repeatedly, you may want to consider increasing the cache size so that you never have to release one page from cache to make room for another.

Creating a multi-file, distributed database

If you feel that your application performance is limited by disk bandwidth, you might consider creating a multi-file database and distributing it across several disks. Multi-file databases were designed to avoid limiting databases to the size of a disk on systems that do not support multi-disk files.

Working with databases

This help file contains text on several topics related to working with databases:

- How to create databases
- How to alter databases
- How to drop databases
- How to extract metadata from existing databases

How to create databases

To create a database and its components, InterBase uses an implementation of SQL which conforms to the ANSI SQL-89 entry-level standard and follows SQL-92 and SQL3 beta specifications for advanced features.

Building a database involves defining the data. For this purpose InterBase provides a set of statements called the Data Definition Language (DDL).

An InterBase database is a single file comprising all the metadata and data in the database.

To create a new database for the InterBase Server, use Windows ISQL. Create a database in ISQL with an interactive command or with the CREATE DATABASE statement in an ISQL script file.

Prerequisites

Before creating the database, you should know:

- Where to create the database. Users who create databases need to know only the logical names of the available devices in order to allocate database storage. Only the system administrator needs to be concerned about physical storage (disks, disk partitions, operating system files).
- The tables that the database will contain.
- The record size of each table, which affects what database page size you choose. A record that is too large to fit on a single page requires more than one page fetch to read or write to it, so access could be faster if you increase the page size.
- How large you expect the database to grow. The number of records also affects the page size because the number of pages affects the depth of the index tree. Larger page size means fewer total pages. InterBase operates more efficiently with a shallow index tree.
- The number of users that will be accessing the database.

Although you can create, alter, and drop a database interactively, it is preferable to use a data definition filedata_def_file because it provides a record of the structure of the database. It is easier to modify a source file than it is to start over by retyping interactive SQL statements.

Using a data definition file

A data definition file may include statements to create, alter, or drop a database, or any other SQL statements. To issue SQL statements through a data definition file, create an ISQL script and process it using Windows ISQL.

Using CREATE DATABASE

CREATE DATABASE establishes a new database and populates its system tables, or metadata, which are the tables that describe the internal structure of the database. CREATE DATABASE must occur before creating database tables, views, and indexes.

CREATE DATABASE optionally allows you to do the following:

- Specify a user name and a password
- Change the default page size of the new database
- Specify a default character set for the database
- Add secondary files to expand the database

CREATE DATABASE must be the first statement in the data definition file. You cannot create a database directly from the ISQL command line.

In DSQL, CREATE DATABASE can only be executed with EXECUTE IMMEDIATE. The database handle and transaction name, if present, must be initialized to zero prior to use.

The syntax for CREATE DATABASE is:

```
CREATE {DATABASE | SCHEMA} "<filespec>"
[USER "username" [PASSWORD "password"]]
[PAGE_SIZE [=] int]
[LENGTH [=] int [PAGE[S]]]
[DEFAULT CHARACTER SET charset]
[<secondary_file>];
<secondary_file> = FILE "<filespec>" [<fileinfo>] [<secondary_file>]
<fileinfo> = LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int
[<fileinfo>]
```

The complete syntax of CREATE DATABASE is in Language Reference.

Creating a single-file database

Although there are many optional parameters, CREATE DATABASE requires only one parameter, "<filespec>", which is the new database file specification. The file specification contains the device name, path name, and database name.

By default, a database is created as a single file, called the primary file. The following example creates a single-file database, named employee.gdb, in the current directory.

```
CREATE DATABASE "employee.gdb";
```

Specifying file size for a single-file database

You can optionally specify a file length, in pages, for the primary file. For example, the following statement creates a database that is stored in one 10,000-page long file:

```
CREATE DATABASE "employee.gdb" LENGTH 10000;
```

If the database grows larger than the specified file length, InterBase extends the primary file beyond the LENGTH limit until the disk space runs out. To avoid this, you can store a database in more than one file, called a secondary file.

Note: Use LENGTH for the primary file only if defining a secondary file in the same statement.

Creating a multi-file database

A multi-file database consists of a primary file and one or more secondary files. You can create one or more secondary files to be used for overflow purposes only; you cannot specify what information goes into each file because InterBase handles this automatically. Each secondary file is typically assigned to a different disk than that of the main database. When the primary file fills up, InterBase allocates one of the secondary files that was created. When that secondary file fills up, another secondary file is allocated, and so on, until all of the secondary file allocations run out.

Whenever possible, the database should be created locally; create the database on the same machine where you are running ISQL. If the database is created locally, secondary file names can include a full file specification, including both host or node names, and a directory path to the location of the database file. If the database is created on a remote server, secondary file specifications cannot include a node name, as all secondary files must reside on the same node.

Specifying file size of a secondary file

Unlike primary files, when you define a secondary file, you must declare either a file length in pages, or a starting page number. The LENGTH parameter specifies a database file size in pages.

If you choose to describe page ranges in terms of length, list the files in the order in which they should be filled. The following example creates a database that is stored in four 10,000-page files, Starting with page 10,001, the files are filled in the order employee.gdb, employee.gd1, employee.gd2, and employee.gd3.

```
CREATE DATABASE "employee.gdb"
  FILE "employee.gd1" STARTING AT PAGE 10001
  LENGTH 10000 PAGES
  FILE "employee.gd2"
  LENGTH 10000 PAGES
  FILE "employee.gd3"
  LENGTH 10000 PAGES;
```

Note: Because file-naming conventions are platform-specific, for the sake of simplicity, none of the examples provided include the device and path name portions of the file specification.

When the last secondary file fills up, InterBase automatically extends the file beyond the LENGTH limit until its disk space runs out. You can either specify secondary files when the database is defined, or add them later, as they become necessary, using ALTER DATABASE. Defining secondary files when a database is created immediately reserves disk space for the database.

Specifying the starting page number of a secondary file

If you do not declare a length for a secondary file, then you must specify a starting page number. STARTING AT PAGE specifies the beginning page number for a secondary file.

The primary file specification in a multi-file database does not need to include a length, but secondary file specifications must then include a starting page number. You can specify a combination of length and starting page numbers for secondary files.

InterBase overrides a secondary file length that is inconsistent with the starting page number. In the next example, the primary file is 10,000 pages long, but the first secondary file starts at page 5,000:

```
CREATE DATABASE "employee.gdb" LENGTH 10000
  FILE "employee.gd1" STARTING AT PAGE 5000
  LENGTH 10000 PAGES
  FILE "employee.gd2"
  LENGTH 10000 PAGES
  FILE "employee.gd3";
```

InterBase generates a primary file that is 10,000 pages long, starting the first secondary file at page 10,001.

Specifying user name and password

For access to InterBase databases on a server, a valid user name and password is required. This is validated against the security database, ISC4.GDB. Every InterBase server comes with SYSDBA user and password “masterkey”. SYSDBA must authorize all other users on a server. For servers that require secure databases, it is strongly recommended that the database administrator change the “masterkey” password for SYSDBA as soon as possible and assign authorized users to the database. Passwords are restricted to 8 characters in length.

The following statement creates a database with a user name and password:

```
CREATE DATABASE "employee.gdb" USER "SALES" PASSWORD "mycode";
```

If you are using Windows ISQL or Server Manager, specify the user name and password in the appropriate dialog box.

If you are using command-line ISQL, a user name and password can be specifed on the command line.

Specifying database page size

You can optionally override the default page size of 1024 bytes for database pages by specifying a different PAGE_SIZE. PAGE_SIZE can be 1024, 2048, 4096, or 8192. The next statement creates a single-file database with a page size of 2048 bytes:

```
CREATE DATABASE "employee.gdb" PAGE_SIZE 2048;
```

When to increase page size

Increasing page size can improve performance for several reasons:

- Indexes work faster because the depth of the index is kept to a minimum.
- Keeping large rows on a single page is more efficient. (A row that is too large to fit on a single page requires more than one page fetch to read or write to it.)
- BLOB data is stored and retrieved more efficiently when it fits on a single page. If an application typically stores large BLOB columns (between 1K and 2K), a page size of 2048 bytes is preferable to the default (1024).

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

Changing page size for an existing database

To change a page size of an existing database, follow these steps:

1. Back up the database.
2. Restore the database using the PAGE_SIZE option to specify a new page size.

Specifying the default character set

DEFAULT CHARACTER SET allows you to optionally set the default character set for the database. The character set determines:

- What characters can be used in CHAR, VARCHAR, and BLOB text columns.
- The default collation order that is used in sorting a column.

Choosing a default character set is useful for all databases, even those where international use is not an issue. Choice of character set determines if transliteration among character sets is possible. For example, the following statement creates a database that uses the ISO8859_1 character set, typically used in Europe to support European languages:

```
CREATE DATABASE "employee.gdb"
DEFAULT CHARACTER SET "ISO8859_1";
```

Using CHARACTER SET NONE

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

For example:

```
CREATE TABLE MYDATA (PART_NUMBER CHARACTER(30) CHARACTER SET NONE);
SET NAMES LATIN1;
INSERT INTO MYDATA (PART_NUMBER) VALUES ("à");
SET NAMES DOS437;
SELECT * FROM MYDATA;
```

The data ("à") is returned just as it was entered, without the à being transliterated from the input character (LATIN1) to the output character (DOS437). If the column had been set to anything other than NONE, the transliteration would have occurred.

How to alter databases

A database can be altered by its creator, the database administrator (SYSDBA), or another user designated by the SYSDBA.
ALTER DATABASE requires exclusive access to the database.
The syntax for ALTER DATABASE is:

```
ALTER {DATABASE | SCHEMA}
ADD <add_clause>;
<add_clause> =
FILE "<filespec>" <fileinfo> [<add_clause>]
<fileinfo> = LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int
[<fileinfo>]
```

The complete syntax of ALTER DATABASE is in the Data Definition Guide.

Example

Use ALTER DATABASE to add one or more secondary files to an existing database. Secondary files are useful for controlling the growth and location of a database. They permit database files to be spread across storage devices, but must remain on the same node as the primary database file.
You must specify a range of pages for each file either by providing the number of pages in each file, or by providing the starting page number for the file. The following statement adds two secondary files to the currently connected database:

```
ALTER DATABASE
  ADD FILE "employee.gd1"
  STARTING AT PAGE 10001
   LENGTH 10000
  ADD FILE "employee.gd2"
   LENGTH 10000;
```

How to drop databases

DROP DATABASE is the command that deletes the database currently connected to, including any associated shadow and log files. Dropping a database deletes any data it contains. A database can be dropped by its creator, the database administrator (SYSDBA), or another user designated by the SYSDBA.

The following statement deletes the current database:

```
DROP DATABASE;
```

How to extract metadata from existing databases

ISQL enables you to extract data definition statements from a database and store them in an output file. All keywords and objects are extracted into the file in uppercase.

The output file enables users to:

- Examine the current state of a database's system tables before planning alterations. This is especially useful when the database has changed significantly since its creation.
- Create a database with schema definitions that are identical to the extracted database.
- Make changes to the database, or create a new database source file with a text editor.

InterBase Documentation Roadmap

InterBase books online

Book	Description
API Guide	Guide and reference for the InterBase client API library.
Data Definition Guide	Designing and creating databases and metadata.
Language Reference	Reference for InterBase SQL statements, errors, and system tables.
Operations Guide	Information on installing and using the InterBase server.
Programmer's Guide	Methods of programming embedded SQL applications using the gpre preprocessor.
InterBase Tutorial	Step by step tutorial in InterBase SQL and database definition. **New!**

You can access the full InterBase 5 documentation set online. All five books are available as PDF documents in a subdirectory of your InterBase installation location.

The InterBase 5 books online are fully searchable if you have **Adobe Acrobat Reader with Search**. The files also have numerous hypertext links within them, indicated by **bold green text**. Finally, the files open Acrobat Reader with a navigation window displayed by default. See the InterBase 5 Release Notes for more details about using Acrobat Reader effectively.

If **Adobe Acrobat Reader with Search** is not installed on your system, you can install it from the InterBase 5 CD-ROM, or you can download the Acrobat Reader with Search software from http://www.adobe.com/prodindex/acrobat/readstep.html

InterBase introduction

Introductory topics in using InterBase to create databases are available in Windows Help format.

Topic	Description
Introduction	Introduction to concepts and definitions of database objects.
Database design	Methods of designing database structure before you start creating metadata.
Database creation	Creating and deleting databases.
Glossary	InterBase glossary of terms.

InterBase tools

You can use Windows Help to view documentation for using the graphical InterBase tools on Windows 95 or Windows NT.

Tool	Description

Windows ISQL	Interactive query and database definition tool.
Server Manager	Database and server configuration and maintenance environment.
ComDiag	Network communication diagnostics tool.
ODBC Driver	INTERSOLV DataDirect ODBC driver.

InterBase version compatibility

When the InterBase version 5.x client in this kit connects to a version 5 InterBase server, all InterBase 5 features are available. When this client runs against the InterBase 4.2.x server or any other pre-version 5 server, the client must not reference features any of the version 5 features, including:

- SQL Roles (CREATE ROLE, GRANT ROLE TO, and GRANT TO ROLE)
- Cascading declarative referential integrity (ON UPDATE and ON DELETE)
- References to isc_lock_print, iblockpr, or gstat command-line utilities
- User configuration API functions isc_create_user(), isc_modify_user(), and isc_delete_user().

In addition, you cannot back up a version 5 database and restore it as a 4.2.x database. Only backups created from 4.2.x databases can be restored as 4.2.x databases.

Sorry, I cannot find either the PDF file or the Adobe Acrobat Reader with Search software

You must have **Adobe Acrobat Reader with Search** installed on your PC in order for the shortcut buttons above to view the online books.

If Acrobat Reader is not installed on your system, you can install it from the InterBase 5.0 CD-ROM, or you can download the Acrobat Reader software from http://www.adobe.com/prodindex/acrobat/readstep.html