Using InterBase From Delphi

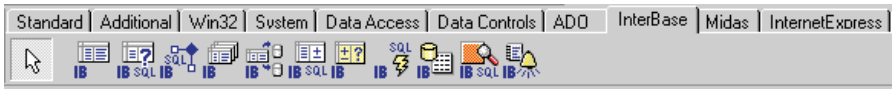
by Nigel Cohen

This article explains the use of InterBase from within Delphi, in a BDE-independent environment. It is intended for all Delphi programmers who currently use the BDE, and for those want to make better use of InterBase in a way that will work within both the Windows and (using Kylix) Linux environments.

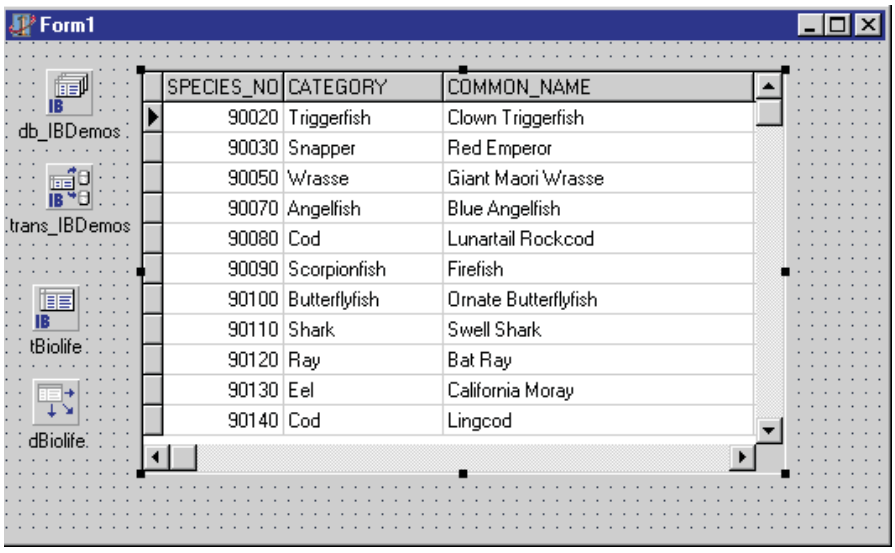
Why Switch From The BDE?

Borland's entire focus at present seems to be providing programmers using Delphi, C++ and Java with a consistent tool to produce native executables that run under both Windows and Linux from the same (or at least very similar) source code.

There is a clear belief in some quarters that Linux is due to become a mainstream operating system for desktop computers. Even if this proves optimistic, it is a popular server operating system, encompassing both network and webserver use. The Borland team have focused their considerable skills on providing a Delphi tool that will compile applications for either Windows and Linux with no changes, or limited changes, to the source. This is vastly more complex than it sounds, but by all accounts it is well on the way to being achieved. The project name for the Linux initiative at Borland, as I'm sure you all know by now, is Kylix.



➤ *Figure1: Interbase Express components.*



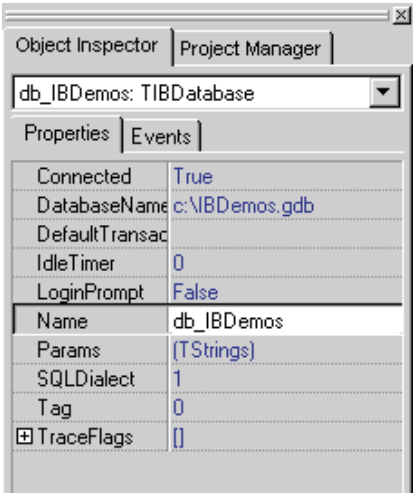
➤ *Figure 2: Delphi form.*

The key advantage for Delphi programmers is that we will be able to service demand from customers who use either Windows or Linux, or both, for a lot less additional programming time than would be the case for many other options (such as C++).

One constraint is that the Borland Database Engine (BDE) does not translate to Linux. So to take advantage of the exciting Windows and Linux opportunities, we must rewrite our code without the BDE. And if we're going to do that, it makes sense to move from a file-sharing type of database engine (which many would now consider outmoded) to a server-based database engine... such as InterBase. This gives advantages of power and scalability, as well as being very feature-rich.

The principles below should look familiar to anyone who uses the BDE to store, retrieve and manipulate data through a database. So the switch should present far fewer technical problems than might otherwise be imagined. If this is the case, then the value of switching to the approach of *direct* access to the database (InterBase in this article) is to open up the entire Linux world, for little additional effort.

➤ *Figure 3: TIBDatabase component.*



How Do I Get Started?

InterBase provides components within Delphi collectively called the InterBase Express components (IBX for short). Four components are crucial: `TIBDatabase`, `TIBTransaction`, `TIBTable` and `TIBQuery`. With these components, you can drive InterBase from Delphi.

If you need to install InterBase, I cover that in a later section, entitled, handily enough, *Installing InterBase*. Otherwise, drop `TIBDatabase`, `TIBTransaction` and `TIBTable` components onto a new form or data module. Also, drop onto the form a `TDataSource` component from the `DataAccess` page of

the palette, and a TDBGrid from the DataControls page (Figure 2).

TIBDatabase Component

The TIBDatabase component deals with connecting your Delphi application to a physical database file. In concept, the program you write makes a single connection to an InterBase database at some point. All the security aspects are dealt with at this point (such as the logon process, etc). You can then repetitively select the connected component, avoiding the time-consuming need to re-connect to the database with each access.

To connect to the database, enter the Windows pathname of the database file in the DatabaseName property. For databases on other machines, use the format

```
DataServer:c:\IBDemos.GDB
```

where DataServer is the name of the machine on which the database file resides, instead of the instinctive j:\IBDemos.GDB format (Figure 3).

To set the login properties, double click the TIBDatabase component. If you want to force your users to login to InterBase when connecting, uncheck the LoginPrompt checkbox. If you want to control access yourself, you can fill in the InterBase user name and password in the dialog, designing your own more relevant login system. The Settings edit box is filled in automatically when you complete the other edit boxes (Figure 4).

To check everything is filled in correctly, set the Connected property to True. If you get an error message, it probably means that either the pathname is incorrectly typed or the user name and password do not tally with those InterBase knows about.

TIBTransaction Component

This component gives you the (rather incredible) ability to set a 'start' point (using the StartTransaction instruction), allow the user to enter any number of entries to the database, and then to roll-back all transactions to the start point if something goes wrong (using the Rollback command), or pass the 'pending' entries through to the database (using the Commit instruction).

Whilst this is useful in advanced programs, for simpler applications it can almost be ignored once it is filled in properly, because it takes care of itself if there is no intention to do anything clever with it. You should note, however, that InterBase *will not work* without using TIBTransaction.

The thing you must do is select your DefaultDatabase (and change the name if you want). You can then ignore this component until you are ready to introduce your program to transaction handling.

TIBTable Component

The TIBTable component works almost in an identical way to TTable. Fill in the name of the database you want, then select the desired table name. You should then be able to connect to the table.

Because TIBTable is descended from TDataSet, you can then use the TIBTable component almost exactly like a TTable. You can hook it up to a DataSource, for example, which can then be hooked up to any 'normal' database component, or you can double click it to define your data fields, lookup fields and computed fields.

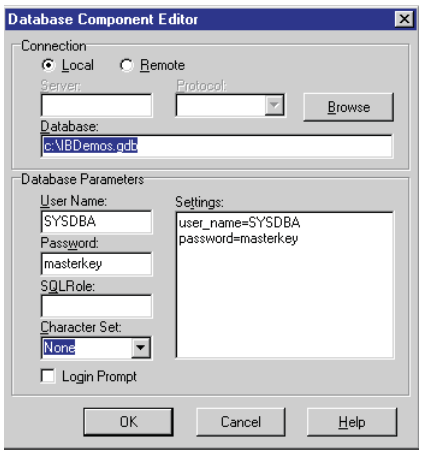
Will This Change Under Kylix?

In Kylix, the IBExpress component set is exchanged for a new DBExpress component set, which will work with both Windows and Linux. From the very preliminary demonstration I have seen, it looks like there will be few changes required to the code I present in this article to cope with the forthcoming releases of Delphi. This is certainly the message that I have heard broadcast clearly by Borland staffers. So the techniques I describe should fit in nicely to the new Kylix environment.

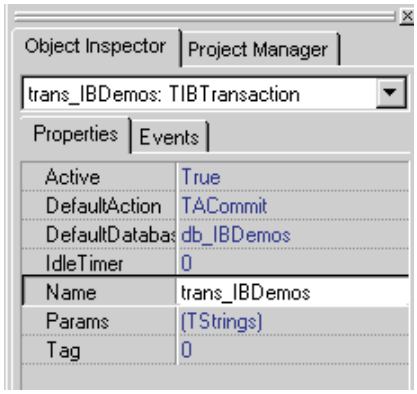
Installing InterBase

InterBase is available free from many sources. It is now shipped with Delphi itself, or it can be

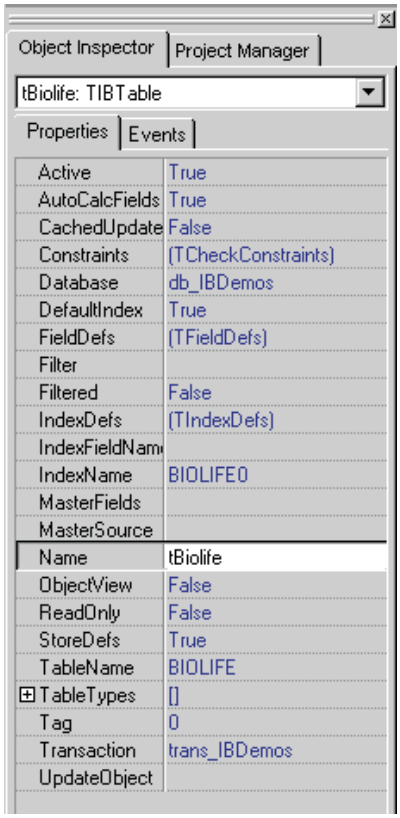
➤ *Figure 4: TIBDatabase Editor component.*



➤ *Figure 5: TIBTransaction component.*



➤ *Figure 6: TIBTable component.*



downloaded free from the Borland website (www.borland.com) or the IBPhoenix site (www.ibphoenix.com). The latest version (which is the Open Source version) is InterBase 6.01 super server. My advice is to avoid the rather buggy version 6.0 if you can.

Once downloaded, simply double click on the installation program and follow the instructions.

If you are running Windows NT, you need Service Pack 5 (or higher) installed. The Service Pack is accessible from the Microsoft website (www.microsoft.com). We were very anxious about upgrading our nice stable system from Service Pack 4, so we proceeded with some trepidation. The unexpectedly good news is that the upgrade was almost as simple as any upgrade we have ever encountered with a Microsoft product. Clicking the *download* instruction on the Microsoft website carried out, as we later discovered, an automated interrogation of our system, which then identified and downloaded just the files we needed and went on to implement the changes, with no further input from us.

A 'one click' upgrade to Service Pack 5 sounds too good to be true, but it worked for us, and is perhaps why Borland could accept the requirement that Service Pack 5 be installed as a condition to using the latest InterBase 6 system.

Putting InterBase To Work

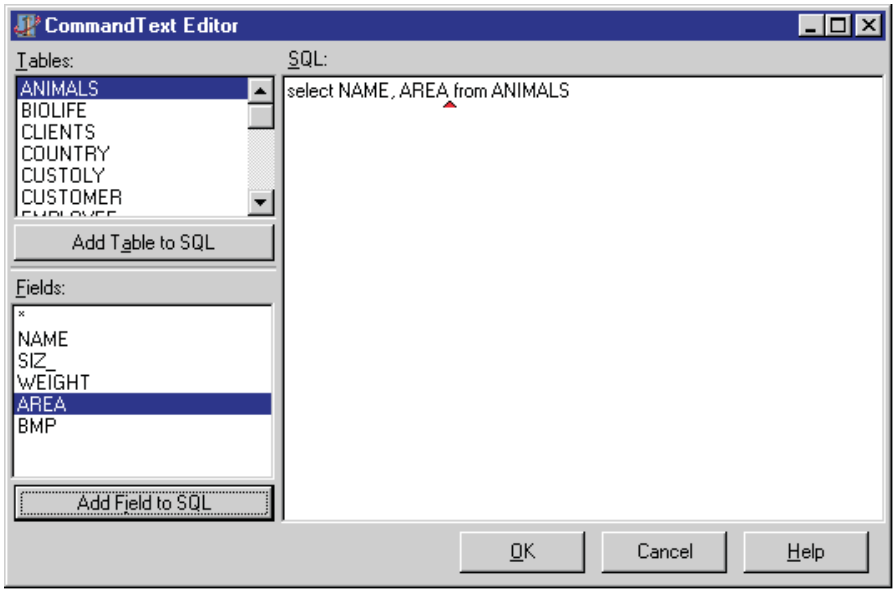
The programming language you send through to a database is called Structured Query Language (SQL for short, usually pronounced as 'sequel' *[though opinion is divided! Ed]*). The mechanism for sending through individual SQL commands is a `TIBQuery` component. Once the `TIBQuery` has been set up, you can fill it with your SQL instruction, send it to InterBase to run, and then use it to review, display and walk through the result set as if it were (generally) a `TTable`.

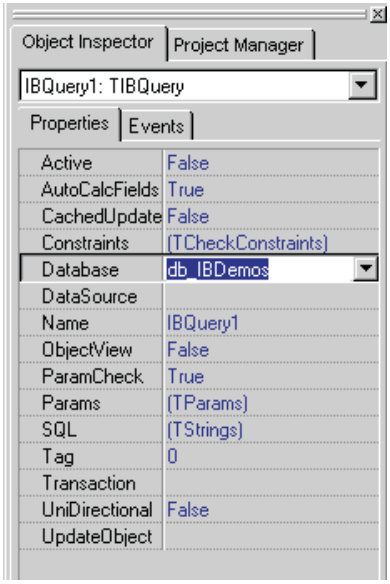
What follows is a brief explanation of how to use the `TIBQuery` component, before we move on to the interesting part explaining the nuts and bolts of how to use SQL to manipulate an InterBase database.

➤ *Listing 1*

```
SalesQuery := TIBQuery.create(self);
  // ******************************************
  // This line will probably become
  // SalesQuery := TDBQuery.create(self);
  // under the Kylix version, with no other code change
  // ******************************************
SalesQuery.SQL.clear;
SalesQuery.SQL.Add(' Select NAME, AREA ');
SalesQuery.SQL.Add(' from ANIMALS ');
SalesQuery.open;
```

➤ *Figure 8: Command Text Editor.*





➤ *Figure 7: TIBQuery.*

TIBQuery

Dropping a `TIBQuery` onto the form shows a familiar property list: see Figure 7.

Fill in the `Database name`, then click the ellipsis (...) in the `SQL` property to display the Command Text Editor (see Figure 8).

You can enter your SQL in the `SQL` edit area and click `OK`. Or, you can select a table and click the `Add Table to SQL` button, then click to select the fields you want and, for each one, click the `Add Field to SQL` and let the editor do the hard work for you. Either way, click the `OK` button to save your SQL. Try clicking the `Active` property of your `TIBQuery` to run the query. Don't forget to hook up the `Dataset` component to the `TIBQuery` to see the information displayed in the `DBGrid` component.

If the query will not become active, it is either because your SQL is faulty (such as misspelling the table name), or because your database component cannot be made active (maybe you incorrectly specifed the location of the database, or perhaps used an invalid user name or password in the `TIBDatabase` login scripts).

An important feature of the `TIBQuery` component is the ability to set the properties at runtime. You can create the query, set the

database, set the SQL and run it with the code in Listing 1.

SQL

SQL is fairly standard between database servers. The majority adhere to a single standard, and InterBase is well specified in this respect. So the SQL commands I will discuss next may well work with other database servers too, such as Oracle or Microsoft SQL server.

Therefore, exploiting the ability to generate SQL within a Delphi program and pass it through to InterBase with a `TIBQuery` will stand you in good stead if you want to move to another database server in the future. Migrating databases is never easy, of course, but at least running the SQL should be one less problem to have to deal with. Within InterBase there are several categories of operation you may need to know about.

The basic SQL instructions include those that change data by inserting, updating or deleting a record (a 'row' in SQL-speak). You can retrieve data in a variety of ways. You can obtain full records, or just a restricted set of fields within a record (for example, just the first and last name of a person, without having to sift through the date of birth, title, address, etc). You can specify a filter, where only records meeting a desired criteria are returned (such as all companies within a specific postcode). You can also retrieve data as groups (such as sales totals, average costs or the total number of units sold, grouped by customer name).

There is a whole suite of SQL commands that deal with *metadata*. This is the data that relates to the database itself, rather than the information it contains. Metadata includes, for example, the names of the tables, the names and datatypes of columns within each table, and the indexes. Needless to say, great care needs to be taken when using the metadata SQL commands, because this is where you issue instructions to delete entire tables. This is very much something to avoid in a live database if you want to avoid your status slipping from 'Most Popular Person' within the organisation!

The final suite of commands invokes the functions and procedures that you may have programmed directly into InterBase. Within InterBase, you can set up a number of functions that can be called from outside. These include stored procedures, triggers and views, which I will discuss in more detail more later.

Basic SQL

The four most basic SQL commands are `Select`, `Update`, `Insert` and `Delete`. These are explained below. Once the basic commands are set, each can be modified and enhanced in consistent ways: I'll run through the enhancements in the following sections.

When working with SQL, I often find that understanding the problem correctly gets you 80% of the way to understanding the solution. So, to bring this article to life, the examples below come from our own website selling eBooks. Go to the website, www.ebooks.uk.net, to see the problems and solutions I will discuss in action.

The Terminology

Many people use database terminology in different ways. For the avoidance of doubt, I will use the terms in the way they are used in InterBase, as follows:

```
Database (IBDemos.gdb)
|
|—Table (Animals)
|— Table (Person)
|—-|
    |—Column (ref)
    |—Column (first name)
    |—Column (last name)
```

A record is a single item within a table (such as the information for an individual person).

A column is a category within a table (such as `Ref` or `FirstName`).

A field is a single item within a record (such as the `Ref` or first name of an individual person).

Select Statement

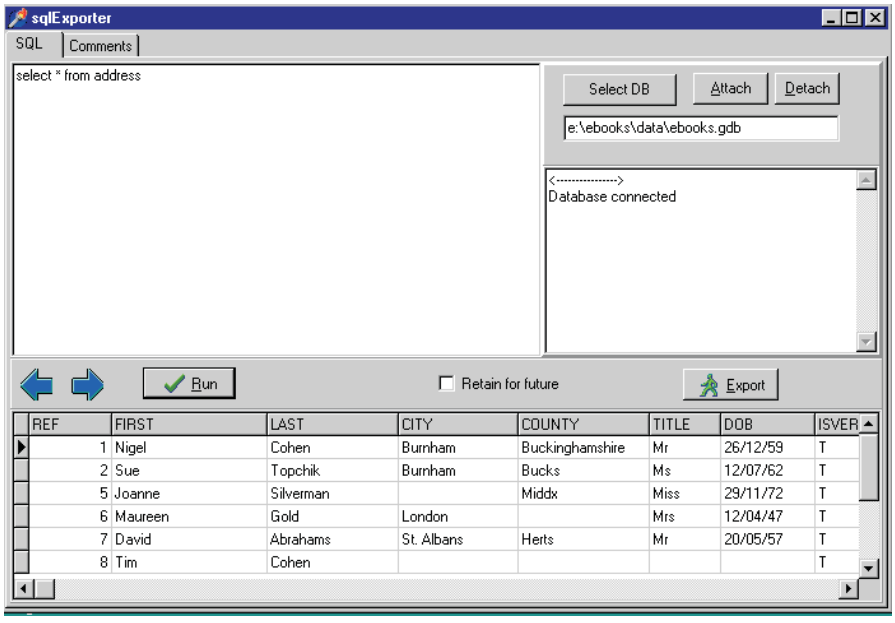
To retrieve all the records from a database, use the command:

```
select * from address ;
```

Where * is the wildcard character and `address` is the table name.

The results of the query are illustrated in Figure 9. To set up this simple database interrogator, we have a `TMemo` into which the SQL commands can be typed, a `TEdit` box to allow us to enter the location of the database we want to interrogate, and a `TDBGrid` for the results. The `TDBGrid` points to a `TIBQuery` via a `TDataSet` component. When the user clicks on the `Run` button, the code executed is something like:

➤ *Figure 9: SqlEditor.*



```
IBQuery.SQL.text := memo1.text;
IBQuery.open;
```

To retrieve specific fields, separate the list of field names with a comma (,), for example:

```
select Ref, First, Last
  from Address ;
```

Note that commands are terminated with the semi-colon (;). The instructions above are not case-sensitive. `Ref`, `First` and `Last` are fields within the table `Address`.

Insert Statement

To insert a record to a table, use the command:

```
insert into Address
  (Ref, First, Last) ;
  values
  (2 , 'Nigel','Cohen')
```

Note that a number has no quotation marks and text has single quotation marks.

Update Statement

To update a record, you usually need to define which record you want to update. I'll explain exactly how this is achieved a little later on. If you don't specify which record(s) to update, the command below will update *every* record in the table:

```
update Address
  set First='', Last='' ;
```

Delete Statement

To delete a record, you almost always need to define exactly which record you want to delete. The *rather dangerous* command below is used to delete every record in the table:

```
delete from Address ;
```

```
update Visitors
set NewVisitor = 'True'
where (VisitedDate > '12/1/2000' and VisitorStatus = 'Visitor')
  or (VisitedDate>'11/15/2000' and VisitorStatus = 'Programmer') ;
```

➤ *Above: Listing 2*

➤ *Below: Listing 3*

```
select    Booktitle, Wordcount, Price
from      Books
where     BookSubject='Poetry' and BookPurpose='Entertainment'
order by  Date desc, BookTitle,
```

Enhancements To The Basic SQL Commands

SQL would not be very useful without the ability to restrict the records returned to those we actually require. Within the eBooks website, for example, we track visitors to the site. If we change the layout, it is important for us to see the effect on behaviour before and after the change. So the first modification is the `where` clause to restrict the records selected, updated, etc, to those where certain conditions are specified:

```
select PageName
  from Visitors
  where VisitedDate >
  '12/1/2000' and
  VisitorStatus = 'Visitor';
```

`PageName` is a field name in the table `Visitors` of type `string`. `VisitorDate` and `VisitorStatus` are also fields in the same table.

In this example, there are two conditions that must be met if the item `PageName` is to be reported. You can equally use the `or` condition if that will accomplish what is desired. If you need to mix the `or` and the `and` conditions, then you should use brackets around the related conditions to ensure that the query works as intended, as shown in Listing 2.

To make a condition the opposite of, use the word `not` before the condition, such as:

```
delete from Visitors
  where not
  ( VisitedDate > '12/1/2000' )
  and
  VisitorStatus like 'Visit%'
```

The brackets above are not really necessary, and have been used to clarify what the `not` negates.

This example illustrates another technique. Within a SQL command, the * string matching wildcard character is not allowed by InterBase. Its equivalent is the `%` character. And instead of stating

```
VisitorStatus = 'Visit%'
```

you use the command `like`, otherwise, the `%` wildcard works just as you would expect the * wildcard to work: you are free to use it at the end of the statement, at the beginning of a string, in the middle, at the end, or indeed any combination you might want.

Often you will want to control in which order the result set appears. This feature is useful if, for example, you want to run through the results in a controlled order, using the `Order by` statement: see the code in Listing 3.

This query returns its result set of book titles in descending date order (that is, most recent publication date first), if that is what you want to display. If two books happen to have been published on exactly the same date, they will be displayed in ascending alphabetic order. `Order by` assumes the instruction `asc` (for ascending), unless you explicitly state `desc` (for descending). Note that both the `where` statement and the `order by` statement used fields in the record that were not included within the selected fields. This is entirely valid.

```
select sum(Price) as Sales, avg(Price), count(SalesRef) as SalesCount
  from   BookSales
  where  AuthorRef = 50 and SalesDate > '11/1/2000' and SalesDate < '1/12/2000'
```

➤ *Above: Listing 4*

➤ *Below: Listing 5*

```
select AuthorName, sum(Price) as Sales, avg(Price), count(SalesRef) as SalesCount
  from     BookSales
  where    SalesDate > '11/1/2000' and SalesDate < '1/12/2000'
  group by  AuthorName
```

```
select    BookTitle, AuthorName, sum(Price) as Sales, avg(Price), count(SalesRef)
            as SalesCount
from      BookSales
where     SalesDate > '11/1/2000' and SalesDate < '1/12/2000'
group by  BookTitle, AuthorName
```

➤ *Above: Listing 6*

➤ *Below: Listing 7*

```
create table RatingTheBook
(
Ref           integer        NOT NULL PRIMARY KEY,
ReviwersRef   integer        NOT NULL,
BookRef       integer        NOT NULL,
RatingDate    date,
RatingAward   numeric(3,1)   default 0 ,
Comment       varchar(5000)  default null,
RatersAge     integer        default 0,
IsActive      char(1)        default 'T'
);
```

Grouping SQL

The next category of SQL is used mainly for reporting purposes.

InterBase has a number of arithmetic functions that can be used, including `SUM()`, `AVG()`, and `COUNT()`, to sum numbers, average them, or count the number of items within the group, respectively. To report on the sale of books by author, for example, you can use the SQL command shown in Listing 4.

In the example, the phrase `as Sales`, which is optional, tells InterBase to call the column `sum(Price)` by the name `Sales` when talking to the `TIBQuery` component. The query reports, for the author with reference 50, their book sales, the average price of books sold and the number of sales transactions in the month of November.

Whilst this is useful, the functionality is hugely enhanced by the facility to group items together using the `group by` statement. This instructs InterBase to carry out its selection routines on each category or grouping. In the eBooks website, this is particularly valuable where we want to report on statistics by author, rather than simply listing overall totals: see Listing 5.

In this example, the sales, average price and number of sales are reported separately for each author. This is great for management, who may want to devote more promotional space on the site to the authors whose books the customers prefer to read. One common error when using the `group by` command is to try using a grouping that conflicts with the selected items. If you are selecting book titles within the `select` statement in Listing 5, for example, InterBase would complain at the instruction to group the result set by `Author`, since it would not know how to collate the book title. To see the statistics by book title and by author, the solution would be to select both author and book title, and to group by both: see Listing 6.

The above solution does not quite achieve what we desire, in that it provides a break at each new book title, but it does not provide a sub-total for the author each time an author changes. Two possible solutions are to run two separate queries, each providing the different breaks, or to use a stored procedure, which I'll discuss shortly.

Metadata SQL

You need to update the metadata far less frequently than the data itself. If you've never met metadata SQL before, you may want to look at the SQL files on this month's companion disk. To get started on working with metadata in InterBase, you may want to use some or all of the commands given in Listing 7.

This statement creates a table called `RatingTheBook`. It has a `Ref` which is its primary key and it has various default values. The InterBase documentation provides help on the datatypes available. For those using InterBase 6.0 and above, the datatypes are fairly intuitive. If you use an earlier version of InterBase, you would be wise to restrict your use of datatypes to those shown, noting that the Delphi `float` or `double` is replaced by the `numeric(3,1)` format, where the first number is the total length of the field (in characters), and the second number is the number of decimal places required.

Note the mandatory use of commas to terminate fields, no comma for the last statement and a semi-colon to terminate the statement, as shown in Listing 8.

This statement changes an existing table (`Address`), by adding some columns and deleting one (`VerifiedComments`, added in an earlier `Create` statement). Note the commas and semi-colons.

Advanced SQL

There are two constraints so far that cause difficulty, but for which there is a solution.

So far, we have selected data from one table alone. To combine data from two tables, you can use the syntax shown in Listing 9.

In this example, we want to report the name of the author, whose name is in the `Address` table, when we report on the book title in the `Books` table. The database was set up with `Books` including a column called `AuthorRef` to refer back to the `Address` table. The conditions required to join the tables together are defined within the `where` statement. Note that each table is given a letter within the `from` statement: this can be used to distinguish fields in two tables

```
alter table Address
add IsVerified         char(1)          default 'F',
add VerifiedDate       date,
add Comments           varchar(2000)    default null,
drop VerifiedComment;
```

➤ *Above: Listing 8*

➤ *Below: Listing 9*

```
select    (a.first || ' ' || a.last ) as Name, b.BookTitle
from      Address a,  Books b
where     a.Ref  =  b.AuthorRef
order by  a.last, a.first
```

which have a common name (such as `ref`).

This is fine when we want to report only on records that match in both tables. But this may be inadequate. In the eBooks site, we report on customers who have ordered books. The report is misleading, though, if we do not also see the customers who have *not* ordered books along with the others. So the statement uses an alternative method of joining with the `join` statement, see Listing 10.

In this example, the `left outer join` means: 'select everything in the table on the left (the `Address` table), and only items within the joined table, `OrderDetails`, where the `OrderDetails` field `Ref` equals the `Address` field `Ref`'.

If we had used `inner join`, InterBase would have reported only the records in `Address` which match the records in `OrderDetails` (omitting, therefore, anyone who has not ordered anything). Similarly, if we had used `right outer join`, InterBase would have reported only `Address` records for which there was an order, but all records within `OrderDetails`, even if there were no matching `Address` record. This `right outer join` report might be interesting to, say, the programmer responsible for ensuring no orders are taken where the buyer has not filled in their address.

The joins can be extended indefinitely if information is required for more than two tables, as shown in Listing 11.

Final Refinements

InterBase has two more features that give almost unlimited power to the programmer.

With `select` statements, there are often occasions where you will want to pull out a result set and then perform a second `select` statement on that result set. For example, if you pulled out a list of summed sales by author, you may want to report it in descending order of summed sales. In the first select statement, the `order by` statement will not be able to work on the results of the summing. So there is a `View` facility that can be programmed into InterBase which allows you to define a `Select` statement. The name of the `View` can then be used within a second `Select` statement as if it is a table, with the result set of the first `View` available to be manipulated by the second select statement.

The icing on the cake for InterBase is stored procedures. A second problem with the `select` statement is its inability to do work on a record as you are going through selecting or summing it. You may want, for example, to update a record based on a calculation to be performed at the time of the sum. Or you may want to produce totals conditionally, such as adding a sale to the `Debit` total if it is plus, and to the `Credit` total if it is minus. The stored procedure allows you to define a `select` statement and, on a record by record basis, do anything you want to, before moving on to the next record. The syntax and use of stored procedures is explained in my *Delphi To InterBase In 15 Minutes* guide which is included on this month's companion disk.

There is no doubt that InterBase is a powerful tool. Borland say they are fully committed to it as a key tool within the Borland/Inprise product range. There is a lot of reticence by programmers to use this powerful and free tool for fear of its complexity. I hope this article has convinced you that the reputation of InterBase as being difficult to implement is ill founded.

Nigel Cohen develops systems using Delphi and InterBase, such as the ebooks.uk.net website featured in this article.

➤ *Above: Listing 10*

```
select a.Ref, a.First, a.Last, d.OrderDate, d.Bookref
from Address a left outer join OrderDetails d
on d.PersonRef = a.Ref
where OrderDate > '12/1/2000'
order by Last, OrderDate
```

➤ *Below: Listing 11*

```
select a.Ref, a.First, a.Last, d.OrderDate, d.Bookref, b.BookTitle
from Address a left outer join OrderDetails d
on d.PersonRef = a.Ref
left outer join books b
on b.ref = d.bookref
where OrderDate > '12/1/2000'
order by Last, OrderDate
```

➤ *Figure 10: Shameless plug: the eBooks website, from which a number of the examples in this article come.*