JOIN

In practice it seldom occurs that all relevant information can be found in a single database table. It is much more often the case that the data required is distributed across several tables and linked by relations. Indeed, information in a normalized database should be spread across multiple tables!

In a fully normalized database, the vast majority of tables have a primary key consisting of one or two columns only. If a referential integrity relationship exists, these primary key columns are replicated in other tables to ensure consistency in the data. These are the columns that allow you to establish logical links between these tables. When queries are performed, tables are commonly joined on these columns.

There is actually no restriction by design to the number of tables that may be joined. However the task of joining tables is exponential in relation to the number of tables in the join. The largest practical number of tables in a join is about 16, but experiment with your application and a realistic volume of data to find the most complex join that has an acceptable performance.

When you establish a join, Firebird/InterBase® looks for matching values in the designated columns of each table. It does not care if a value appears once on one side of the join and multiple times on the other side, as is often the case.

In this instance, Firebird/InterBase® joins each matching row in TableB to the single matching row in TableA, thereby creating what is known as a virtual row. Each TableB row can logically be linked to a single unambiguous row in TableA.

Firebird/InterBase® also provides options for establishing a relationship where a value can appear on one side of the join instead of both. This is known as an OUTER JOIN.

The following statement selects from both TableA and TableB tables:

```
SELECT column_list
FROM TableA, TableB;
```

When you select from two or more tables, these tables are normally joined on a common column. For example, you might join TableA and TableB tables on the column that is common to each of them, the TableA_ID.

Theoretically it is not necessary to specify a join column. If you do not specify one, Firebird/InterBase® performs a Cartesian product between the two tables, joining each row in one table to each row in the other. So, for example, if the first table had 100 rows, and the second had 20, the result set would have 2000 rows. Such a join normally makes no sense because the row information in one table is not logically related to the row information in the other table, except where column and field values are shared between the tables.

Firebird/InterBase® does not prevent you from establishing a meaningless join. You can issue an SQL statement that joins, for example, Orders.PaymentMethod with Customer.Country, and Firebird/InterBase® processes the statement! But the result set is always empty because there are no matching values in either column.

JOIN syntax

Firebird/InterBase® currently supports two methods to link two or more tables via a common column:

- the traditional SQL syntax, and
- the SQL '92 syntax.

The traditional SQL syntax integrates the link in the WHERE clause:

```
SELECT <ColumnList>
     FROM Table1 Synonym1 , Table2 Synonym2
     WHERE Synonym1.JoinColumn = Synonym2.JoinColumn
     AND <Other_WHERE_Conditions> ;
```

The following example illustrates this syntax:

```
SELECT C.Name, C.Country, O.OrderID, O.SaleDate, O.TotalInvoice
     FROM Customer C, Orders O
     WHERE C.CustomerID = O.CustomerID
     AND C.Country != 'U.S.A.'
     ORDER BY C.Name, O.OrderID;
```

As opposed to traditional SQL syntax, the SQL 92 syntax detaches the link from the WHERE clause and relocates it in the FROM clause, i.e. that area, in which the tables to be used are defined:

```
SELECT <ColumnList>
     FROM Table1 Alias1 JOIN Table2 Alias2
               ON Alias1.Column = Alias2.Column
     WHERE <Where_Conditions> ;
```

Example

```
SELECT C.Name, C.Country, O.OrderID, O.SaleDate, O.TotalInvoice
     FROM Customer C JOIN Orders O
         ON   C.CustomerID = O.CustomerID )
     WHERE C.Country != 'U.S.A.'
   ORDERBY C.Name, O.OrderID;
```

Either syntax can be used at any time; they are virtually interchangeable. The difference is that the SQL 92 syntax permits OUTER JOINs, whereas the traditional syntax does not.

Specifying columns and rows

When two or more tables are joined, rows can be included from either table in the result. It is also possible to specify WHERE conditions to limit the rows in either table that are considered for the join.

For example, the following statement asks for customers in Florida who placed orders in 1994 with a total invoice of more than $5,000 for the order:

```
SELECT C.Name, C.City, O.SaleDate, O.TotalInvoice
FROM Customer C JOIN Orders O
ON C.CustomerID = O.CustomerID
```

```
WHERE C.State_Province = 'FL'
AND O.SaleDate BETWEEN '1/1/94' AND '12/31/94'
AND O.TotalInvoice > 5000;
```

Please refer to Joining more than two tables for further information.

back to top of page

INNER JOIN

When you join two tables, the result set includes only those rows where the joining value appears in both tables.

Syntax

```
TableA JOIN TableB
```

The join applies to the table written to the left of the command.

For example, the following query joins Stock to LineItem to find out many orders included each stock item:

```
SELECT S.StockID, COUNT( L.OrderID )
FROM Stock S JOIN Lineitem L
ON S.StockID = L.StockID
GROUP BY S.StockID
```

From a theoretical standpoint, this is known as an INNER JOIN, but the INNER keyword is optional. What if you also want to include those stock items that have not yet been ordered, so that the result set shows all stock items. These items do not appear in the LineItem table at all. The solution lies in performing an OUTER JOIN. An outer join includes every column in one table and a subset of columns in the other table.

back to top of page

OUTER JOIN

When you join two tables, the result set includes only those rows where the joining value appears in both tables.

There are three types of outer joins:

SQL92 syntax permits outer joins, whereas the traditional syntax does not.

Types of outer joins

- LEFT OUTER JOIN, which includes all rows from the table on the left side of the join expression.
- RIGHT OUTER JOIN, which includes all rows from the table on the right side of the join expression.
- FULL OUTER JOIN, which includes all rows from both tables.

Syntax

```
TableA LEFT OUTER JOIN TableB
```

The join applies to the table written to the left of the command.

```
TableA RIGHT OUTER JOIN TableB
```

The join applies to the table written to the right of the command.

When your tables are linked in a referential relationship on a foreign key column, only the LEFT OUTER JOIN usually makes sense. For example, every order includes a customer from the Customer table. If you join Customer to Orders with a RIGHT OUTER JOIN, the result is the same as if you had performed an INNER JOIN.

The following query modifies the preceding example to include all stock items, even the one that have not yet been ordered:

```
SELECT S.StockID, COUNT( L.OrderID )
FROM Stock S LEFT OUTER JOIN Lineitem L
ON S.StockID = L.StockID
GROUP BY S.StockID
```

Adding selection criteria

If two tables are joined using an outer join, and there are also selection criteria in the table where the inclusion operator is placed, it would appear as first glance that you are asking two conflicting questions.

Consider the following query, which asks for the value of all orders placed by customers located in California, including those customers who might not have placed an order.

```
SELECT C.Name, SUM( O.TotalInvoice )
FROM Customer C LEFT OUTER JOIN Orders O
ON C.CustomerID = O.CustomerID
WHERE C.State_Province = 'CA'
GROUP BY C.Name;
```

On the one hand, the LEFT OUTER JOIN is asking Firebird/InterBase® to include all customers in the result set, whether or not that customer has also placed any orders. On the other hand, the query is also asking Firebird/InterBase® to limit the query to only those customers located in California.

Firebird/InterBase® resolves this apparent conflict by always processing the WHERE clause before processing any outer joins. The Customer table is first limited to those customers in California, and this intermediate result is then joined to the Orders table to which of the California customers have

placed orders.

back to top of page

CROSS JOIN

CROSS JOIN? was introduced in Firebird 2.0. Logically, this syntax pattern:

```
A CROSS JOIN B
```

is equivalent to either of the following:

```
A INNER JOIN B ON 1 = 1
```

or, simply:

```
FROM A, B
```

Joining more than two tables

The SQL92 join syntax provides for joins that reference more than two tables. The trick is to establish the join with the first pair of tables, then join this product with the third table, and so on.For example, the following query finds customers and the order details, where the order included a specific stock item:

```
SELECT C.Name, O.SaleDate, L.Quantity
FROM Customer C JOIN Orders O
ON ( C.CustomerID = O.CustomerID )
JOIN LineItem L
ON ( O.OrderID = L.OrderID )
WHERE L.StockID = '5313';
```

This syntax can be extended to any number of tables. You can even create a circular join. For example, the following statement asks for customers who have ordered products that were made by vendors in the same state as the customer. This query requires a series of joins from Customer to Orders to LineItem to Stock to Vendors, and another join from the Customer state to the Vendor's state.

```
SELECT DISTINCT C.Name, V.VendorName, C.State_Province
FROM Customer C JOIN Orders O
ON ( C.CustomerID = O.CustomerID )
JOIN LineItem L
ON ( O.OrderID = L.OrderID )
JOIN Stock S
ON ( L.StockID = S.StockID )
```

```
JOIN Vendors V
ON ( S.VendorID = V.VendorID )
AND ( C.State_Province = V.State_Province );
```

Note an important limitation in this SELECT statement: tables are added to the JOIN expression one at a time. You cannot reference columns from a table until the table has been joined to the expression. For example, the condition linking the Customer and Vendor tables on their State columns cannot be specified until the Vendor table has been added to the expression and correctly joined.

back to top of page

Self joins / reflexive joins

A self-join, also known as a reflexive join, is a join in which a table is joined to itself. It compares rows of data within a single table. For example, we could add another column to the employee table in the sample employee database that would contain the employee's manager number. Since managers are also stored in the employee table, we could create a self-join on the employee table to determine the name of each employee's manager.

```
SELECT e1.full_name AS Employee, e2.full_name AS Manager
FROM employee e1 JOIN employee e2
ON e1.mng_id = e2.emp_no;
```

Named columns join

Two new JOIN types were introduced in Firebird 2.1: the NAMED COLUMNS join and its close relative, the NATURAL join.

```
<named columns join> ::=
  <table reference> <join type> JOIN <table reference>
    USING ( <column list> )
```

- All columns specified in <column list> should exist in the tables at both sides.
- An equi-join (<left table>.<column> = <right table>.<column>) is automatically created for all columns (ANDed).
- The USING columns can be accessed without qualifiers - in this case, the result is equivalent to COALESCE(<left table>.<column>, <right table>.<column>).
- In "SELECT *", USING columns are expanded once, using the above rule.

Example

```
select * from employee
  join department
  using (dept_no);
```

Natural join

```
<natural join> ::=
  <table reference> NATURAL <join type> JOIN <table primary>
```

1. A “named columns join” is automatically created with all columns common to the left and right tables.
2. If there is no common column, a CROSS JOIN is created.

Example

```
select * from employee_project
  natural join employee
  natural join project;
```