Lab 05: Joining Tables

Objective(s):

To learn the use of Joins in SELECT statement.

Joins

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how SQL should use data from one table to select the rows in another table.

A join condition defines the way two tables are related in a query by:

- Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table.
- Specifying a logical operator (for example, = or <>,) to be used in comparing values from the columns.

There are various forms of the JOIN clause. These will include:

- INNER JOIN
- OUTER JOIN (both LEFT and RIGHT)
- FULL OUTER JOIN
- CROSS JOIN

A JOIN does just what it sounds like—it puts the information from two tables together into one result set. We can think of a result set as being a “virtual” table. It has both columns and rows, and the columns have data types. How exactly does a JOIN put the information from two tables into a single result set? Well, that depends on how you tell it to put the data together—that’s why there are four different kinds of JOINs. The thing that all JOINs have in common is that they match one record up with one or more other records to make a record that is a superset created by the combined columns of both records.

For example, let's take a record from a table we'll call Films:

FilmID	FilmName	YearMade
1	My Fair Lady	1964

Now let's follow that up with a record from a table called Actors:

FilmID	FirstName	LastName
1	Rex	Harrison

With a JOIN, we could create one record from two records found in totally separate tables:

FilmID	FilmName	YearMade	FirstName	LastName
1	My Fair Lady	1964	Rex	Harrison

1) INNER JOINs

INNER JOINs are far and away the most common kind of JOIN. They match records together based on one or more common fields, as do most JOINs, but an INNER JOIN returns only the records where there are matches for whatever field(s) you have said are to be used for the JOIN. In our previous examples, every record has been included in the result set at least once, but this situation is rarely the case in the real world.

Let's modify our tables and see what we would get with an INNER JOIN. Here's our Films table:

FilmID	FilmName	YearMade
1	My Fair Lady	1964
2	Unforgiven	1992

And our Actors table:

FilmID	FirstName	LastName
1	Rex	Harrison
1	Audery	Hepburn
2	Clint	Eastwood
5	Humphery	Bogart

Using an INNER JOIN, our result set would look like this:

FilmID	FilmName	YearMade	FirstName	LastName
1	My Fair Lady	1964	Rex	Harrison
1	My Fair Lady	1964	Audery	Hepburn
2	Unforgiven	1992	Clint	Eastwood

Syntax

SELECT <*select list*>FROM <*first_table*><*join_type*><*second_table*>[ON <*join_condition*>]

<join type > can be INNER JOIN, LEFT OUTER, RIGHT OUTER, FULL JOIN and CROSS JOIN

2) OUTER JOIN

The LEFT OUTER returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2). While The RIGHT OUTER returns all the rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

3) FULL OUTER JOIN

The FULL OUTER JOIN keyword return rows when there is a match in one of the tables

4) CROSS JOIN

This returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table

Example

Fire up the Management Studio and take a test drive of INNER JOINs using the following code against Northwind:

```
SELECT *
FROM Products
INNER JOIN Suppliers
ON Products.SupplierID = Suppliers.SupplierID
```

The results of this query, you should get something in the order of 77 rows back. There are several things worth noting about the results:

a) The SupplierID column appears twice, but there’s nothing to say which one is from which table.

b) All columns were returned from both tables.

c) The first columns listed were from the first table listed.

Exercise

Using EMP , DEPT and SALGRADE tables, solve the following queries.

1. Create a unique listing of all jobs that are in department 30. Include the location of department in the output.

2. Write a query to display the employee name, department name and loc of all employees who earn a commission.

3. Create a query that displays the name, job, department name, salary, and grade for all employees.

4. Create a query to display the name and hire date of any employee hired after employee ADAMS.

5. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates.