Upon completion we should be able to do the following:

1. Write SELECT statements to access data from more then one table using equality and nonequality joins.
2. View data the generally does not meet a join condition using outer, inner, left, and right joins.
3. Join a table to itself.
4. Use joins in the dialects of Oracle, MySQL and MSSQL,

SQL Joins

The join concept allows us to truly utilize the power of a Relational Database. We are able to use the Primary Key and Foreign Key relationship as presented in 1970 by E. F. Codd in a landmark paper “A Relational Model of Data for Data for Large Shared Data Banks”.

This concept allows us to obtain data from one or more tables when the combined data fits into the criteria desired.

Types of Joins

There are several types of joins but most of them fall into two categories:

1) Equijoins (also called SIMPLE or INNER joins)

2) Non-equijoins

There are additional join methods such as:

1) Outer joins

2) Self joins

What is a Join

When data from more than one table is required, a Where condition is used. Rows in one table can be joined to rows in another table according to common values existing in corresponding columns, this usually is the primary key and foreign key columns.

The table are linked with the WHERE clause of the SQL statement and we have to follow one rule.

The number of WHERE clauses is (n-1) with n being the number of tables. Three tables being joined means two WHERE clauses.

Guidelines

1) When writing a SELECT statement that joins two tables, precede the column names with the table name for clarity and to enhance database access.

2) If the same column name appears in more than one table, the column name MUST be prefixed with the table name.

3) Any valid statement or function is allowed in the WHERE clause.

Sample Join

Example of an Equijoin of tables EMP and DEPT.

```
Select emp.empno, emp.name, emp.dept, dept.deptno,
dept.loc
from emp, dept
Where emp.deptno = dept.deptno;
```

We could also use table aliases:

```
Select e.empno, e.name, e.dept, d.deptno, d.loc
from emp e, dept d
Where e.deptno = d.deptno;
```

Cartesian Product

When a join condition is invalid or omitted, the results is a Cartesian product in which all combinations of rows will be displayed. All rows in the first table are joined to all rows in the second table. Example:

If you join two table, one with 14 rows and one with 4 rows you will receive 56 rows in the result set. For this reason we should always include a VALID join condition in a WHERE clause.

Guideline for Table Aliases

1) Table aliases can be up to 30 character in length, but usually the shorter the better.

2) If a table alias is used for a particular table name in the FROM clause, then that table alias MUST be precede the table name through out the SELECT statement.

3) Table aliases should be meaningful.

4) The table alias is valid only for the current SELECT statement.

5) If you join more than two tables you will need to use brackets.

Non-Equijoins

A non-equijoin is the joining of tables where the columns of one table do not correspond directly to the columns of another. Example:

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
Where e.sal between s.lowsal AND s.hisal;
```

We are checking the salary column of EMP and comparing it to a range of salaries (lowsal and hisal) in the SALGRADE table.

More non-euijoins

From the previous slide note the following:

1) There would be no duplication of records returned since each employee has only one value in the SAL column that has to be inside the ranges of the SALGRADE table.

2) No employee can receive less than the lowest value in losal or more then the highest value in hisal.

Outer Joins

This is obtained if a row does not satisfy a join condition. There are two types of outer joins.

1) right outer join – meaning if the table on the RIGHT side of the where condition has rows that were not matched, those rows will also be returned.

2) left outer join – meaning if the table on the LEFT side of the where condition has rows that were not matched, those rows will also be returned.

Self Joins

This is the joining a table to itself. We do this by creating an alias for each instance of the table reference.

Example:

```
SELECT worker.ename, manager.ename
FROM emp worker, emp manager
WHERE worker.mgr = manager.empno;
```

Oracle INNER join

```
SELECT customers.customer_id, orders.order_id,
orders.order_date
FROM customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id
ORDER BY customers.customer_id;
```

Oracle LEFT Outer join

```
SELECT customers.customer_id, orders.order_id,
orders.order_date
FROM customers
LEFT OUTER JOIN orders ON customers.customer_id =
orders.customer_id
ORDER BY customers.customer_id;
```

Oracle RIGHT Outer join

```
SELECT customers.customer_id, orders.order_id,
orders.order_date
FROM customers
RIGHT OUTER JOIN orders ON customers.customer_id =
orders.customer_id
ORDER BY customers.customer_id;
```

Oracle FULL Outer join

```
SELECT customers.customer_id, orders.order_id,
orders.order_date
FROM customers
FULL OUTER JOIN orders ON customers.customer_id =
orders.customer_id
ORDER BY customers.customer_id;
```

Oracle SELF join

```
SELECT e1.last_name||' works for '||e2.last_name
"Employees and Their Managers"
 FROM employees e1, employees e2
WHERE e1.manager_id = e2.employee_id;
```

3 table join

It is very similar in all three dialects.

```
Select cus.CompanyName, o.OrderDate, emp.LastName
From ((Employees emp
inner join orders o on  emp.EmployeeID = o.EmployeeID)
inner join customers cus on  cus.CustomerID =
o.CustomerID);
```

MySQL INNER join

```
SELECT Orders.OrderID, Customers.CustomerName,
Orders.OrderDate
FROM Orders
INNER JOIN Customers ON
Orders.CustomerID=Customers.CustomerID;
```

MySQL LEFT join

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
ORDER BY Customers.CustomerName;
```

MySQL RIGHT join

```
SELECT Orders.OrderID, Employees.LastName,
Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID =
Employees.EmployeeID
ORDER BY Orders.OrderID;
```

MySQL FULL OUTTER join

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON
Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

MySQL SELF join

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

MSSQL INNER join

```
SELECT
    CONCAT(m.lastname, ', ', m.firstname) AS 'Manager',
    CONCAT(e.lastname, ', ', e.firstname) AS 'Direct report'
FROM
    employees e
        INNER JOIN
    employees m ON m.employeeNumber = e.reportsto
ORDER BY manager;
```

MSSQL LEFT join

```
SELECT
    CONCAT(m.lastname, ', ', m.firstname) AS 'Manager',
    CONCAT(e.lastname, ', ', e.firstname) AS 'Direct report'
FROM
    employees e
        LEFT  JOIN
    employees m ON m.employeeNumber = e.reportsto
ORDER BY manager;
```

MSSQL RIGHT join

```
SELECT
    CONCAT(m.lastname, ', ', m.firstname) AS 'Manager',
    CONCAT(e.lastname, ', ', e.firstname) AS 'Direct report'
FROM
    employees e
     RIGHT JOIN
    employees m ON m.employeeNumber = e.reportsto
ORDER BY manager;
```

MSSQL FULL join

```
SELECT
    CONCAT(m.lastname, ', ', m.firstname) AS 'Manager',
    CONCAT(e.lastname, ', ', e.firstname) AS 'Direct report'
FROM
    employees e
    FULL JOIN
    employees m ON m.employeeNumber = e.reportsto
ORDER BY manager;
```

MSSQL SELF join

```
SELECT
    CONCAT(m.lastname, ', ', m.firstname) AS 'Manager',
    CONCAT(e.lastname, ', ', e.firstname) AS 'Direct report'
FROM
    employees e
        SELF JOIN
    employees m ON m.employeeNumber = e.reportsto
ORDER BY manager;
```