JOINS:

To determine an employee’s department name, you compare the value in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an equijoin—that is, values in the DEPARTMENT_ID column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements.
Note: Equijoins are also called simple joins or inner joins.

Use a decision matrix for simplifying writing joins. For example, if you want to display the name and department number of all the employees who are in the same department as Goyal, you can start by making the following decision tree:

What? **Columns to Display**	Where? **Originating Table**	How? **Condition**

last_name
department_name
	Employees
	Departments
		last_name=’Goyal’
		employees.department_id = departments.department_id

The first column gives the column list in the SELECT statement, the second column gives the tables for the FROM clause, and the third column gives the condition for the WHERE clause.

Retrieving Records with Equijoins

```
SELECT employees.employee_id, employees.last_name,
     employees.department_id, departments.department_id,
     departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

The SELECT clause specifies the column names to retrieve:
employee last name, employee number, and department number, which are columns in the EMPLOYEES table department number, department name, and location ID, which are columns in the DEPARTMENTS table.

The FROM clause specifies the two tables that the database must access:
EMPLOYEES table
DEPARTMENTS table

The WHERE clause specifies how the tables are to be joined:
EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID

Because the DEPARTMENT_ID column is common to both tables, it must be prefixed by the table name to avoid ambiguity.

Using the AND Operator

In addition to the join, you may have criteria for your WHERE clause to restrict the rows under consideration for one or more tables in the join. For example, to display employee Matos' department number and department name, you need an additional condition in the WHERE clause.

```
SELECT last_name, employees.department_id,
        department_name
  FROM   employees, departments
  WHERE  employees.department_id = departments.department_id
  AND    last_name = 'Matos';
```

Qualifying Ambiguous Column Names

You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT_ID column could be from either the DEPARTMENTS table or the EMPLOYEES table. It is necessary to add the table prefix to execute your query.
If there are no common column names between the two tables, there is no need to qualify the columns. However, using the table prefix improves performance, because you tell the Oracle Server exactly where to find the columns.
The requirement to qualify ambiguous column names is also applicable to columns that may be ambiguous in other clauses, such as the SELECT clause or the ORDER BY clause.

Table Aliases

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. You can use table aliases instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, therefore using less memory.
Notice how table aliases are identified in the FROM clause in the example. The table name is specified in full, followed by a space and then the table alias. The EMPLOYEES table has been given an alias of e, and the DEPARTMENTS table has an alias of d.

Guidelines:

- Table aliases can be up to 30 characters in length, but shorter is better.
- If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid only for the current SELECT statement.

```
SELECT e.employee_id, e.last_name, e.department_id,
      d.department_id, d.location_id
FROM   employees e , departments d
WHERE  e.department_id = d.department_id;
```

Additional Search Conditions
Sometimes you may need to join more than two tables. For example, to display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

```
  SELECT e.last_name, d.department_name, l.city
  FROM   employees e, departments d, locations l
  WHERE  e.department_id = d.department_id
  AND    d.location_id = l.location_id;
```

Non-Equijoins
A non-equijoin is a join condition containing something other than an equality operator. The relationship between the EMPLOYEES table and the JOB_GRADES table has an example of a non-equijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST_SALARY and HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equals (=).

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary  BETWEEN j.lowest_sal AND j.highest_sal;
```

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:

- None of the rows in the job grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits provided by the job grade table. That is, no employee earns less than the lowest value contained in the LOWEST_SAL column or more than the highest value contained in the HIGHEST_SAL column.

Note: Other conditions, such as <= and >= can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using BETWEEN. Table aliases have been specified in the example for performance reasons, not because of possible ambiguity.

BETWEEN … AND … is actually translated by the Oracle server to a pair of AND conditions (a >= lower limit) and (a <= higher limit) and IN (…) is translated by the Oracle server to a set of OR conditions (a = value1 OR a = value2 OR a = value3). So using BETWEEN … AND … , IN(…) has no performance benefits; the benefit is logical simplicity.

Self-Join: Joining a Table to Itself

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self join. For example, to find the name of Whalen's manager, you need to:

- Find Whalen in the EMPLOYEES table by looking at the LAST_NAME column.
- Find the manager number for Whalen by looking at the MANAGER_ID column. Whalen's manager number is 101.
- Find the name of the manager with EMPLOYEE_ID 101 by looking at the LAST_NAME column. Kochhar's employee number is 101, so Kochhar is Whalen's manager.

In this process, you look in the table twice. The first time you look in the table to find Whalen in the LAST_NAME column and MANAGER_ID value of 101. The second time you look in the EMPLOYEE_ID column to find 101 and the LAST_NAME column to find Kochhar.

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```

This example joins the EMPLOYEES table to itself. To simulate two tables in the FROM clause, there are two aliases, namely w and m, for the same table, EMPLOYEES.
In this example, the "WHERE" clause contains the join that means "where a worker's manager number matches the employee number for the manager."

(The column heading in the result of the query on the slide seems meaningless. A meaningful column alias should have been used instead.)

There are only 19 rows in the output, but there are 20 rows in the EMPLOYEES table. This occurs because employee King, who is the president, does not have a manager.

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row will not appear in the query result. For example, in the equijoin condition of EMPLOYEES and DEPARTMENTS tables, employee Grant does not appear because there is no department ID recorded for her in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

```
  SELECT e.last_name, e.department_id, d.department_name
  FROM   employees e, departments d
  WHERE  e.department_id = d.department_id;
```

Using Outer Joins to Return Records with No Direct Match
The missing rows can be returned if an outer join operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the “side” of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more rows from the nondeficient table can be joined.
In the syntax:

table1.column =   is the condition that joins (or relates) the tables together.

table2.column (+)   is the outer join symbol, which can be placed on either side of the WHERE clause condition, but not on both sides. (Place the outer join symbol following the name of the column in the table without the matching rows.)

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id ;
```

This example displays employee last names, department ID’s and department names. The Contracting department does not have any employees. The empty value is shown in the output shown.

Outer Join Restrictions

- The outer join operator can appear on only one side of the expression—the side that has information missing. It returns those rows from one table that have no direct match in the other table.
- A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator.

Oracle	**SQL: 1999**
Equi-Join	Natural/Inner Join
Outer-Join	Left Outer Join
Self-Join	Join ON
Non-Equi-Join	Join USING
Cartesian Product	Cross Join

Cross Joins

The CROSS JOIN clause produces the cross product of two tables which is the same as a Cartesian product between the two tables.

```
SELECT last_name, department_name
FROM employees
CROSS JOIN departments ;
```

Corresponding Oracle syntax:

```
SELECT last_name, department_name
FROM employees, departments;
```

Natural Joins

- Based on all columns in the two tables that have the same name.
- Selects rows from the two tables that have equal values in all matched columns.
- Returns an error if the columns having the same names contain different data types.

```
SELECT department_id, department_name,
location_id, city
FROM departments
NATURAL JOIN locations ;
```

Corresponding Oracle syntax:

```
SELECT d.department_id, d.department_name, d.location_id, l.city
FROM departments d, locations l
WHERE d.location_id = l.location_id;
```

JOIN USING:

Natural joins use all columns with matching names and data types to join the tables. The USING clause can be used to specify only those columns that should be used for an equijoin. The columns referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement.

For example, this statement is valid:

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE location_id = 1400;
```

This statement is invalid because the LOCATION_ID is qualified in the WHERE clause:

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE d.location_id = 1400;
```

```
ORA-25154: column part of USING clause cannot have qualifier
```

The same restriction applies to NATURAL joins also. Therefore columns that have the same name in both tables have to be used without any qualifiers.

QUESTION: If you want to show information from two tables which have no columns with matching names or data types, how would you do that?

ANSWER: Join with another table which has matching columns for each of the other tables.

EXAMPLE:

Give last names and countries where each employee was based.

```
Select last_name, country_id
FROM employees e JOIN departments d
ON e.department_id=d.department_id
JOIN locations l
ON d.location_id=l.location_id
```

(Notice that the departments table is used for a join, although no data from that table is shown in the results from the query.)

Inner and Outer Joins

The join of two tables returning only matched rows is an inner join.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)
```

Left/right outer join: A join between two tables that returns the results of the inner join as well as unmatched rows in the left (or right) tables is a left (or right) outer join.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id)
```

The above query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no match in the DEPARTMENTS table.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id)
```

The Oracle proprietary syntax is as follows for the left outer join:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id (+) = e.department_id;
```

Full outer join: A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It also retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON (e.department_id = d.department_id) ;
```

This can be done with Oracle syntax, but you don't have to know how to do this:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id (+) = d.department_id
UNION
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id (+)
```

You can apply additional conditions in the WHERE clause. The example shown performs a join on the EMPLOYEES and DEPARTMENTS tables, and, in addition, displays only employees with a manager ID equal to 149.

```
SELECT e.employee_id, e.last_name, e.department_id, d.department_id,
d.location_id
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)
AND e.manager_id = 149 ;
```