**JOINS**

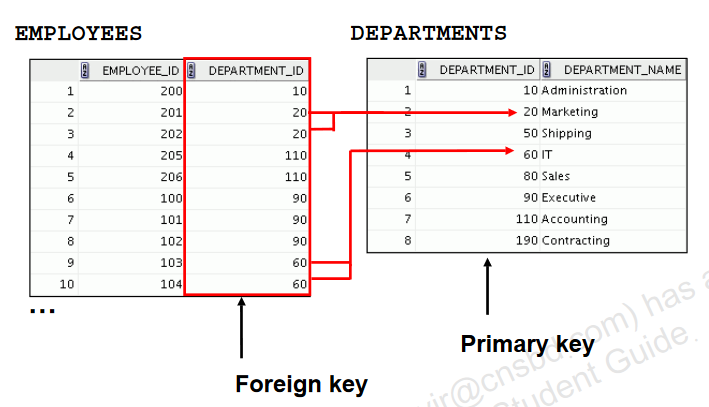
What is JOINS?

Joins are two types

1. Physical Join
2. Logical Join

* Physical Join:

Physical join means implementing relationship between two tables.

****

This is also called as referential integrity constrains

* Logical Join:

Logical Join are two types

* To get combined data
* To get combination of data

What is combined data?

* By suing set operators
  + Same Data type of data
  + Same number of columns from each table

What is combination of data?

* Different data type of data
* Different number of columns select for each tables.

combination of data are

* Natural Join
* Outer Joins
* Cross Joins
* Self Join
* Non equijoins

**Natural Join**

* The NATURAL JOIN clause is based on all the columns in the two tables that have the same name.
* It selects rows from the two tables that have equal values in all matched columns.
* If the columns having the same names have different data types, an error is returned.
* **USING clause**
* **ON clause**

|  |  |  |
| --- | --- | --- |
| **Natural Join** | **Using Clause** | **On clause** |
| * **Same name** * **Same Data-type** * **Same values** * **Match all the column between 2 tables** | * **Same name** * **Different Data-type** * **Match only one column between 2 tables** * **Using clause cannot have identifier** | * **Filter the where clause** |

|  |
| --- |
| Capture.PNG  SELECT department\_id,department\_name,  Location\_id,city  FROM departments  Natural JOIN locations; |

SELECT department\_id, department\_name, location\_id, city

From departments

NATURAL JOIN locations

WHERE department\_id IN(10,30);

|  |
| --- |
| Capture.PNG  SELECT first\_name, department\_name  FROM employees NATURAL JOIN departments; |

|  |
| --- |
| Capture.PNG  SELECT last\_name, job\_title, max\_salary  FROM employees NATURAL JOIN jobs; |

|  |
| --- |
| SELECT location\_id, street\_address, city,  state\_province,country\_name  FROM locations NATURAL JOIN countries; |

**Creating Joins with the USING Clause**

* If several columns have the same names but the data types do not match, use the USING clause to specify the columns for the equijoin.
* Use the USING clause to match only one column when more than one column matches.
* The NATURAL JOIN and USING clauses are mutually exclusive.

SELECT employee\_id,last\_name,

Location\_id,department\_id

FROM employees JOIN departments

USING(department\_id);

**Using Table Aliases with the USING Clause**

* Do not qualify a column that is used in the USING clause.
* If the same column is used elsewhere in the SQL statement, do not alias it.

|  |
| --- |
| Capture.PNG  SELECT l.city,d.department\_name  FROM locations l JOIN departments d  USING(location\_id)  WHERE location\_id=1400;  Capture.PNG |

|  |
| --- |
| Capture.PNG  SELECT e.employee\_id, l.city, d.department\_name  FROM employees e JOIN departments d  USING (department\_id)  JOIN locations l USING (location\_id);  Capture.PNG |

|  |
| --- |
| * **In the following statement,manager\_id is present in both the employeesand departmentstable; if manager\_idis not prefixed with a table alias, it gives a “column ambiguously defined” error.**   **Capture.PNG**  SELECT first\_name,d.department\_name,d.manager\_id  FROM employees e JOIN departments d  USING(department\_id)  WHERE department\_id=50;  Capture.PNG |

|  |
| --- |
| * **The HR department needs a report of only those employees with corresponding departments. Write a query to display the last name, department number, and department name for these employees.**     SELECT last\_name, department\_id, department\_name  FROM employees JOIN departments  USING(department\_id); |

**Creating Joins with the ON Clause**

* The join condition for the natural join is basically an equijoin of all columns with the same name.
* Use the ON clause to specify arbitrary conditions or specify columns to join.
* The join condition is separated from other search conditions.
* The ONclause makes code easy to understand.

|  |
| --- |
| Capture.PNG  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);  Capture.PNG  **SAME RESULT**  SELECT e.employee\_id, e.last\_name, e.department\_id,  d.department\_id, d.location\_id  FROM employees e JOIN departments d  ON (e.manager\_id=d.manager\_id  AND e.department\_id = d.department\_id); |

|  |
| --- |
| Capture.PNG  SELECT employee\_id, city,department\_name  FROM employees e  JOIN departments d  ON d.department\_id = e.department\_id  JOIN locations l ON d.location\_id = l.location\_id;  Capture.PNG |

|  |
| --- |
| * **Ther HR department needs a report of employees in Toronto. Display the last name, job, department number, and the department name for all employees who work in Toronto.**     SELECT e.last\_name, e.job\_id, e.department\_id, d.department\_name  FROM employees e JOIN departments d  ON (e.department\_id = d.department\_id)  JOIN locations l  ON (d.location\_id = l.location\_id)  WHERE LOWER(l.city) = 'toronto'; |

|  |
| --- |
| * The HR department wants to determine the names of all the employees who were hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.     SELECT e.last\_name, e.hire\_date  FROM employees e JOIN employees davies  ON (davies.last\_name = 'Davies')  WHERE davies.hire\_date < e.hire\_date; |

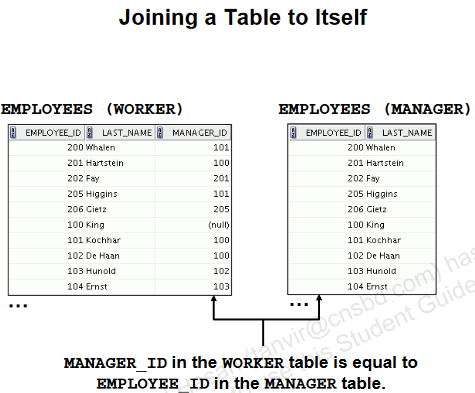
|  |
| --- |
| * The Hr department needs to find the names and hire dates of all the employees who were hired before their managers, along with their managers’ names and hire dates.     SELECT w.last\_name, w.hire\_date, m.last\_name, m.hire\_date  FROM employees w JOIN employees m  ON (w.manager\_id = m.employee\_id)  WHERE w.hire\_date < m.hire\_date; |

**Applying Additional Conditions to a Join**

|  |
| --- |
| **Capture.PNG**  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 ;  Capture.PNG  **OR**  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 ; |

**Self-Joins Using the ON Clause**

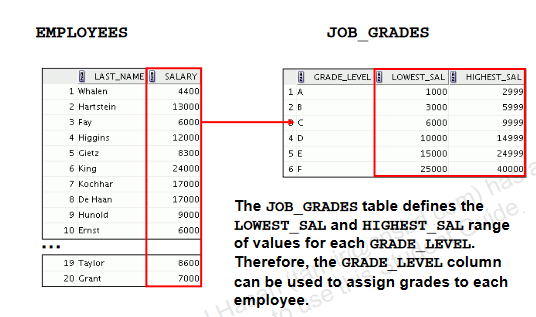
* **A self join is a join in which a table is joined with itself.**
* **To join a table itself means that each row of the table is combined with itself and with every other row of the table.**
* **The table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition.**
* **The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.**
* **To perform a self join, Oracle Database combines and returns rows of the table that satisfy the join condition.**



|  |
| --- |
| SELECT worker.last\_name emp, manager.last\_name mgr  FROM employees worker JOIN employees manager  ON (worker.manager\_id = manager.employee\_id);  Capture.PNG |

|  |
| --- |
| * Create a report to display employees’ last name and employee number along with their manager’s last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively.     SELECT w.last\_name "Employee", w.employee\_id "EMP#" ,  m.last\_name "Manager", m.employee\_id "MNG#"  FROM employees w JOIN employees m  ON (w.manager\_id = m.employee\_id); |

**NOnequijoins**



**Retrieving Records with Nonequijoins**

SELECT e.last\_name, e.salary, j.grade\_level

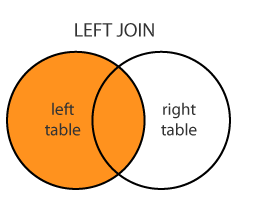
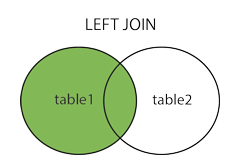
FROM employees e JOIN job\_grades j ON e.salary

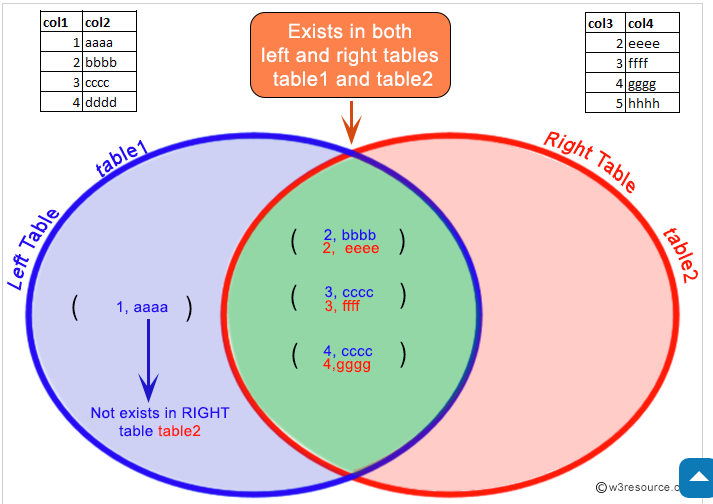
BETWEEN j.lowest\_sal AND j.highest\_sal;

**OUTRE JOINS**

**LEFT OUTER JOINS**

**LEFT OUTER JOINS:** The LEFT JOIN keyword returns all rows from the left table (table1),with the matching rows in the right table (table2).  The result is NULL in the right side when there is no match.





|  |
| --- |
| Capture.PNG  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); |

|  |
| --- |
| * Display all employees including King, who has no manager. Order the results by the employee number     SELECT w.last\_name "Employee", w.employee\_id "EMP#",  m.last\_name "Manager", m.employee\_id "Mgr#"  FROM employees w  LEFT OUTER JOIN employees m  ON (w.manager\_id = m.employee\_id)  ORDER BY 2; |

**RIGHT OUTER JOIN**

Returns all rows from the right table, even if there are no matches in the left table.

|  |
| --- |
| SELECT e.last\_name, d.department\_id, d.department\_name  FROM employees e RIGHT OUTER JOIN departments d  ON (e.department\_id = d.department\_id) ; |

**FULL OUTER JOIN**

|  |
| --- |
| SELECT e.last\_name, d.department\_id, d.department\_name  FROM employees e FULL OUTER JOIN departments d  ON (e.department\_id = d.department\_id) |

**Cartesian Products**

* A Cartesian product is formed when:
  + A join condition is omitted
  + A join condition is invalid
  + All rows in the first table are joined to all rows in the second table
* Always include a valid join condition if you want to avoid a Cartesian product.

|  |
| --- |
|  |

**Creating Cross Joins**

* The CROSS JOINclause produces the cross-product of two tables.
* This is also called a Cartesian product between the two tables.

|  |
| --- |
| SELECT last\_name, department\_name  FROM employees  CROSS JOIN departments ; |

|  |
| --- |
| * Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label.     SELECT e.department\_id department, e.last\_name employee,  c.last\_name colleague  FROM employees e JOIN employees c  ON (e.department\_id = c.department\_id)  WHERE e.employee\_id <> c.employee\_id  ORDER BY e.department\_id, e.last\_name, c.last\_name; |