Defining a Column Alias

- Renames a column heading
- Is useful with calculations
- Immediately follows column name;

  AS keyword between column name and alias

Using Column Aliases

```
SQL> SELECT ename AS name, sal AS salary
     FROM   emp;
```

```
NAME                 SALARY
------------- ----------
...
```

```
SQL> SELECT ename Name,
     sal*12 AS AnnualSalary
     FROM   emp;
```

```
Name           AnnualSalary
-------------

-------------
...
```

Using the LIKE Operator

- Use the LIKE operator to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers.
  - % denotes zero or many characters.
  - _ denotes one character.

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE 'S%';
```

Using the LIKE Operator

- You can combine pattern-matching characters.

```
SQL> SELECT   ename
  2  FROM emp
  3  WHERE    ename LIKE '_A%';
```

```
ENAME
----------
MARTIN
JAMES
WARD
```

Using the IS NULL Operator

- Test for null values with the IS NULL operator.

```
SQL> SELECT  ename, mgr
  2  FROM    emp
  3  WHERE   mgr IS NULL;
```

```
ENAME            MGR
---------- ----------
KING
```

Logical Operators

Operator	Meaning
AND	Returns TRUE if *both* component conditions are TRUE
OR	Returns TRUE if *either* component condition is TRUE
NOT	Returns TRUE if the following condition is FALSE

Using the AND Operator

AND requires both conditions to be TRUE.

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  AND    job='CLERK';
```

```
    EMPNO ENAME      JOB              SAL
--------- ---------- --------- ---------
     7876 ADAMS      CLERK          1100
     7934 MILLER     CLERK          1300
```

Using the OR Operator

OR requires either condition to be TRUE.

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  OR     job='CLERK';
```

```
    EMPNO ENAME      JOB             SAL
--------- ---------- --------- ---------
     7839 KING       PRESIDENT      5000
     7698 BLAKE      MANAGER        2850
     7782 CLARK      MANAGER        2450
     7566 JONES      MANAGER        2975
     7654 MARTIN     SALESMAN       1250
     ...
     7900 JAMES      CLERK           950
     ...
14 rows selected.
```

Using the NOT Operator

```
SQL> SELECT ename, job
  2  FROM   emp
  3  WHERE  job NOT IN ('CLERK','MANAGER','ANALYST');
```

```
ENAME      JOB
---------- ---------
KING       PRESIDENT
MARTIN     SALESMAN
ALLEN      SALESMAN
TURNER     SALESMAN
WARD       SALESMAN
```

Sorting Data

- Sort rows with the ORDER BY clause
  - ASC: ascending order, default
  - DESC: descending order
- The ORDER BY clause comes last in the SELECT statement.

```
SQL> SELECT    ename, job, deptno, hiredate
  2  FROM      emp
  3  ORDER BY hiredate;
```

```
ENAME      JOB           DEPTNO HIREDATE
---------- --------- ---------- ---------
SMITH      CLERK             20 17-DEC-80
ALLEN      SALESMAN          30 20-FEB-81
...
14 rows selected.
```

Sorting in Descending Order

```
SQL> SELECT    ename, job, deptno, hiredate
  2  FROM      emp
  3  ORDER BY hiredate DESC;
```

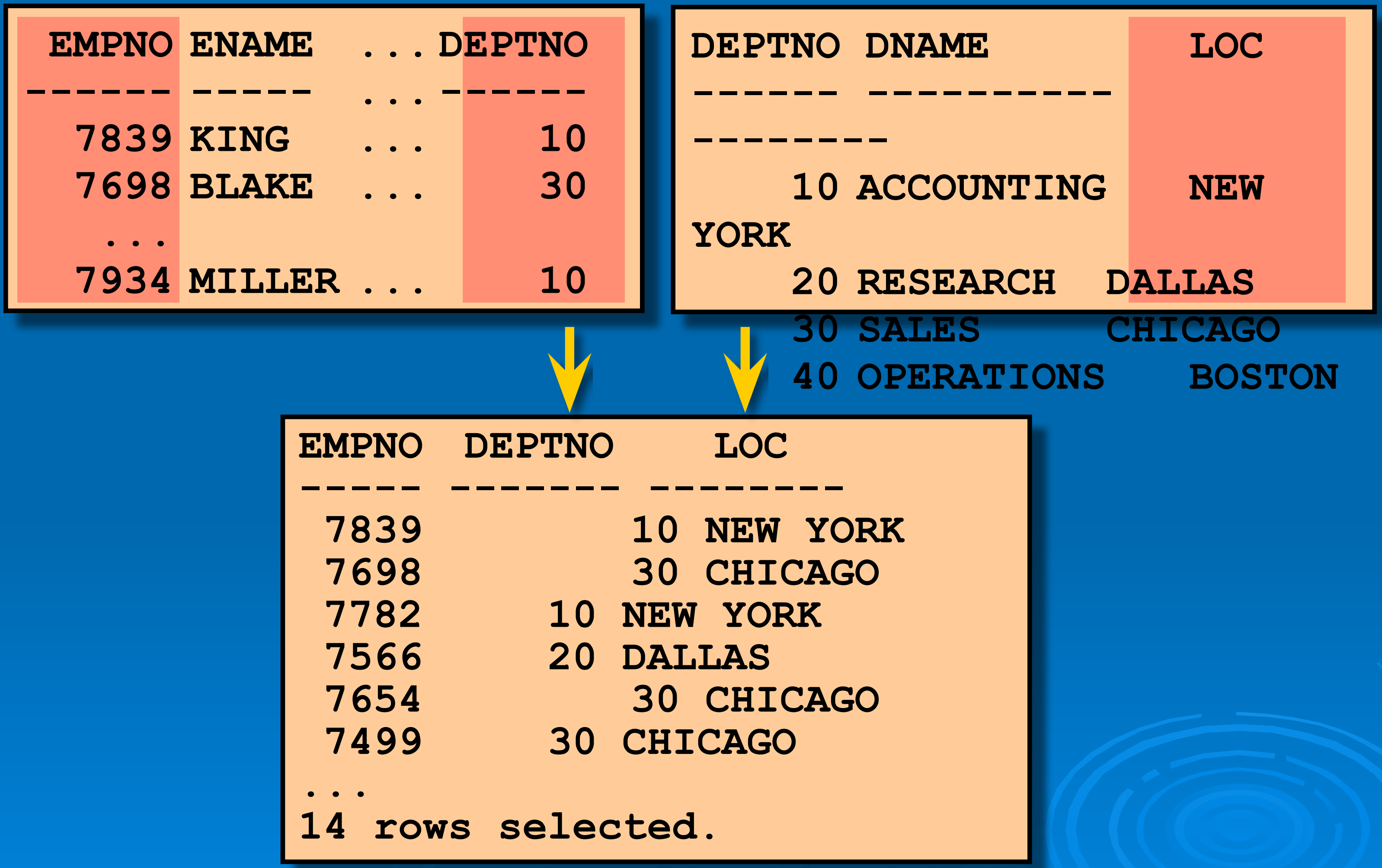
```
ENAME      JOB          DEPTNO HIREDATE
---------- --------- --------- ---------
ADAMS      CLERK            20 12-JAN-83
SCOTT      ANALYST          20 09-DEC-82
MILLER     CLERK            10 23-JAN-82
JAMES      CLERK            30 03-DEC-81
FORD       ANALYST          20 03-DEC-81
KING       PRESIDENT        10 17-NOV-81
MARTIN     SALESMAN         30 28-SEP-81
...
14 rows selected.
```

Sorting by Column Alias

```
SQL> SELECT    empno, ename, sal*12 annsal
  2  FROM      emp
  3  ORDER BY annsal;
```

```
     EMPNO ENAME          ANNSAL
---------- ---------- ----------
      7369 SMITH            9600
      7900 JAMES           11400
      7876 ADAMS           13200
      7654 MARTIN          15000
      7521 WARD            15000
      7934 MILLER          15600
      7844 TURNER          18000
...
14 rows selected.
```

Obtaining Data from Multiple Tables



EMP

```
  EMPNO ENAME    ...DEPTNO
------- -----    ...------
   7839 KING     ...    10
   7698 BLAKE    ...    30
    ...
   7934 MILLER   ...    10
```

DEPT

```
DEPTNO  DNAME          LOC
------ ----------
--------
      10 ACCOUNTING     NEW
YORK
      20 RESEARCH   DALLAS
      30 SALES      CHICAGO
      40 OPERATIONS     BOSTON
```

```
EMPNO  DEPTNO    LOC
----- ------- ---------
 7839        10 NEW YORK
 7698        30 CHICAGO
 7782     10 NEW YORK
 7566     20 DALLAS
 7654        30 CHICAGO
 7499     30 CHICAGO
...
14 rows selected.
```

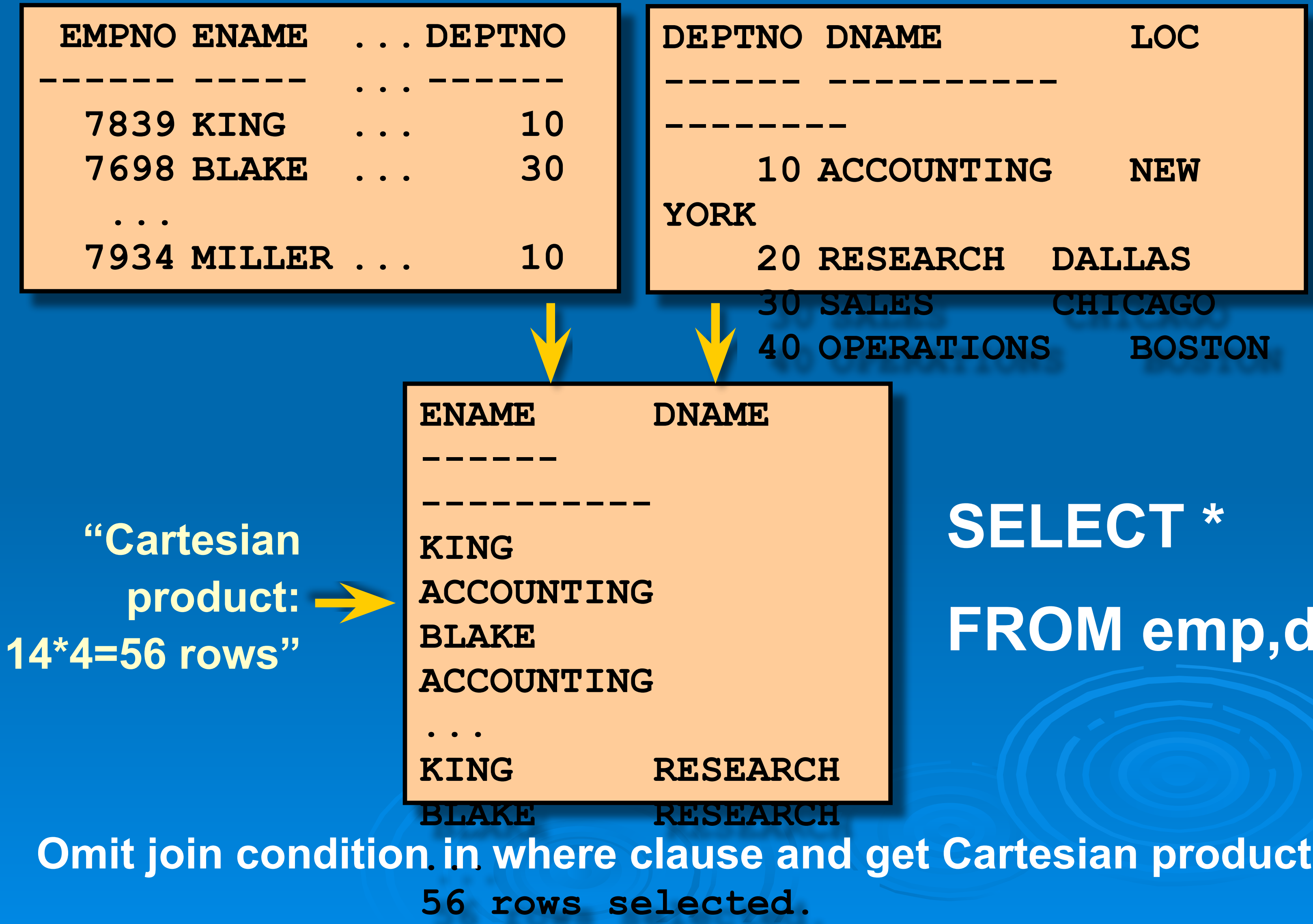
What Is a Join?

- Use a join to query data from more than one table.

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column1 = table2.column2;
```

  - Write the join condition in the WHERE clause.
  - Prefix the column name with the table name when the same column name appears in more than one table.

Generating a Cartesian Product



EMP (14 rows)

```
 EMPNO ENAME    ...DEPTNO
------ -----    ...------
  7839 KING     ...    10
  7698 BLAKE    ...    30
 ...
  7934 MILLER ...      10
```

DEPT (4 rows)

```
DEPTNO DNAME          LOC
------ ----------
---------
    10 ACCOUNTING     NEW
YORK
    20 RESEARCH   DALLAS
    30 SALES          CHICAGO
    40 OPERATIONS     BOSTON
```

"Cartesian product: 14*4=56 rows"

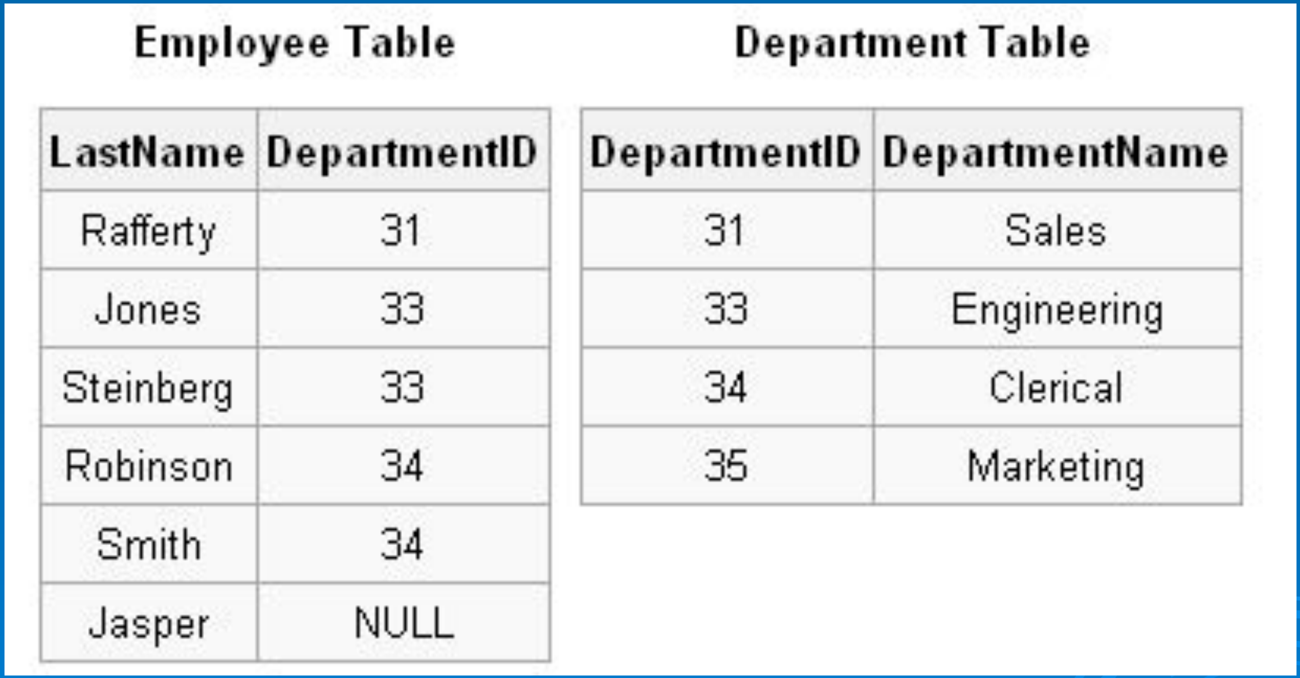
```
ENAME      DNAME
------
----------
KING
ACCOUNTING
BLAKE
ACCOUNTING
...
KING       RESEARCH
BLAKE      RESEARCH
...
56 rows selected.
```

SELECT *

FROM emp,dept;

Omit join condition in where clause and get Cartesian product

Sample Tables



Employee Table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
Jasper	NULL

Department Table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

INNER JOIN

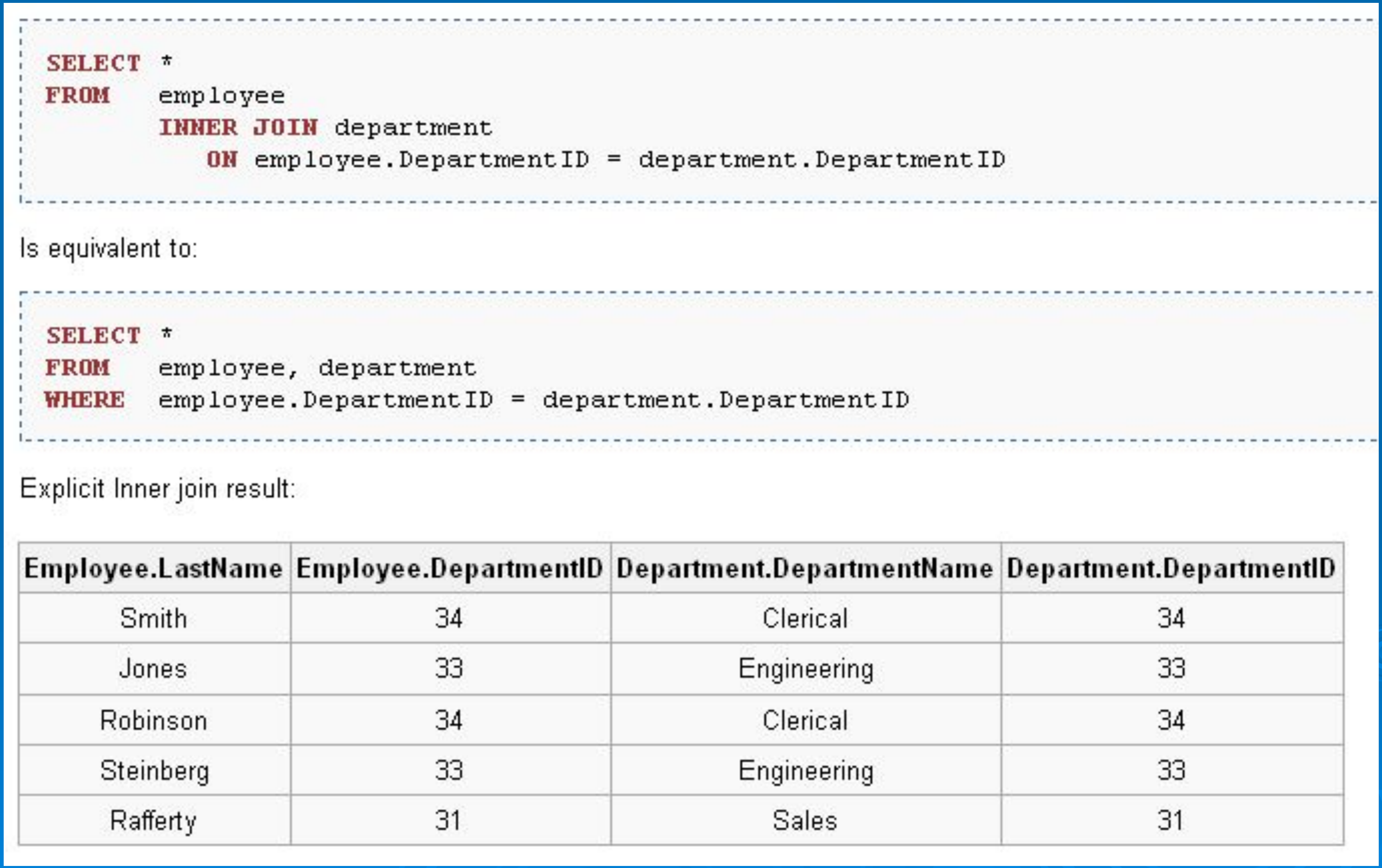
- Combines records from two tables whenever there are matching values in a common field.

Syntax

- FROM *table1* INNER JOIN *table2* ON *table1.field1 compopr table2.field2*

- *table1* and *table2* are names of two tables
- *compopr* is the comparision operator
- *field1* and *field2* are names of join fields

INNER JOIN EXAMPLE



```
SELECT *
FROM   employee
       INNER JOIN department
          ON employee.DepartmentID = department.DepartmentID
```

Is equivalent to:

```
SELECT *
FROM   employee, department
WHERE  employee.DepartmentID = department.DepartmentID
```

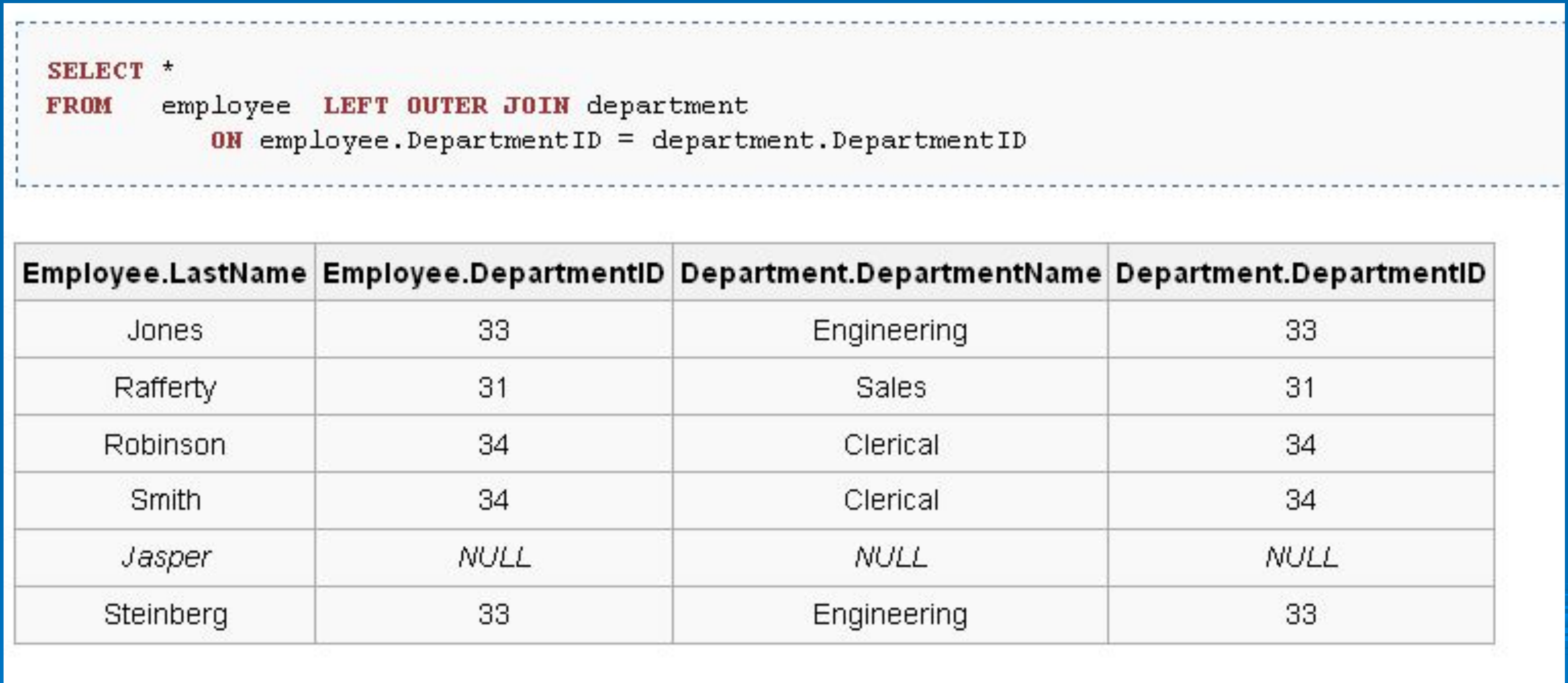
Explicit Inner join result:

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31

LEFT OUTER JOIN

- In outer join all records from left side table in LEFT JOIN operation are added to the **resulting relation**, even if there are no matching values in the joined field from the table on the right.
- Records from the table on the right are combined with those from the table on the left only when there are matching values in the joined fields. When a left-side record has no match, a row of **Null** values is joined on the right side.

Outer Join (Left)



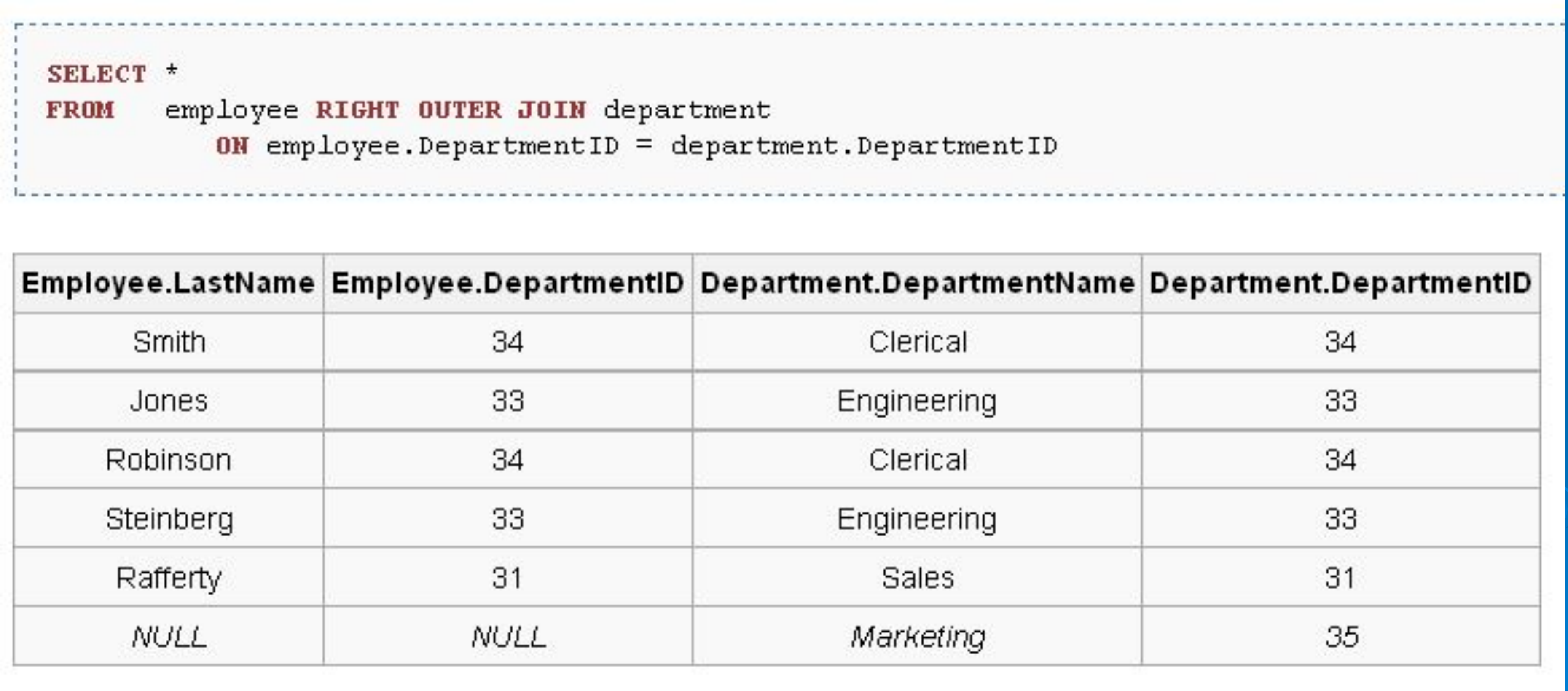
```
SELECT *
FROM   employee  LEFT OUTER JOIN department
          ON employee.DepartmentID = department.DepartmentID
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Jasper	*NULL*	*NULL*	*NULL*
Steinberg	33	Engineering	33

RIGHT OUTER JOIN

- In outer join all records from right side table in RIGHT JOIN operation are added to the **resulting relation**, even if there are no matching values in the joined field from the table on the left.
- Records from the table on the left are combined with those from the table on the right only when there are matching values in the joined fields. When a right-side record has no match, a row of **Null** values is joined on the left side.

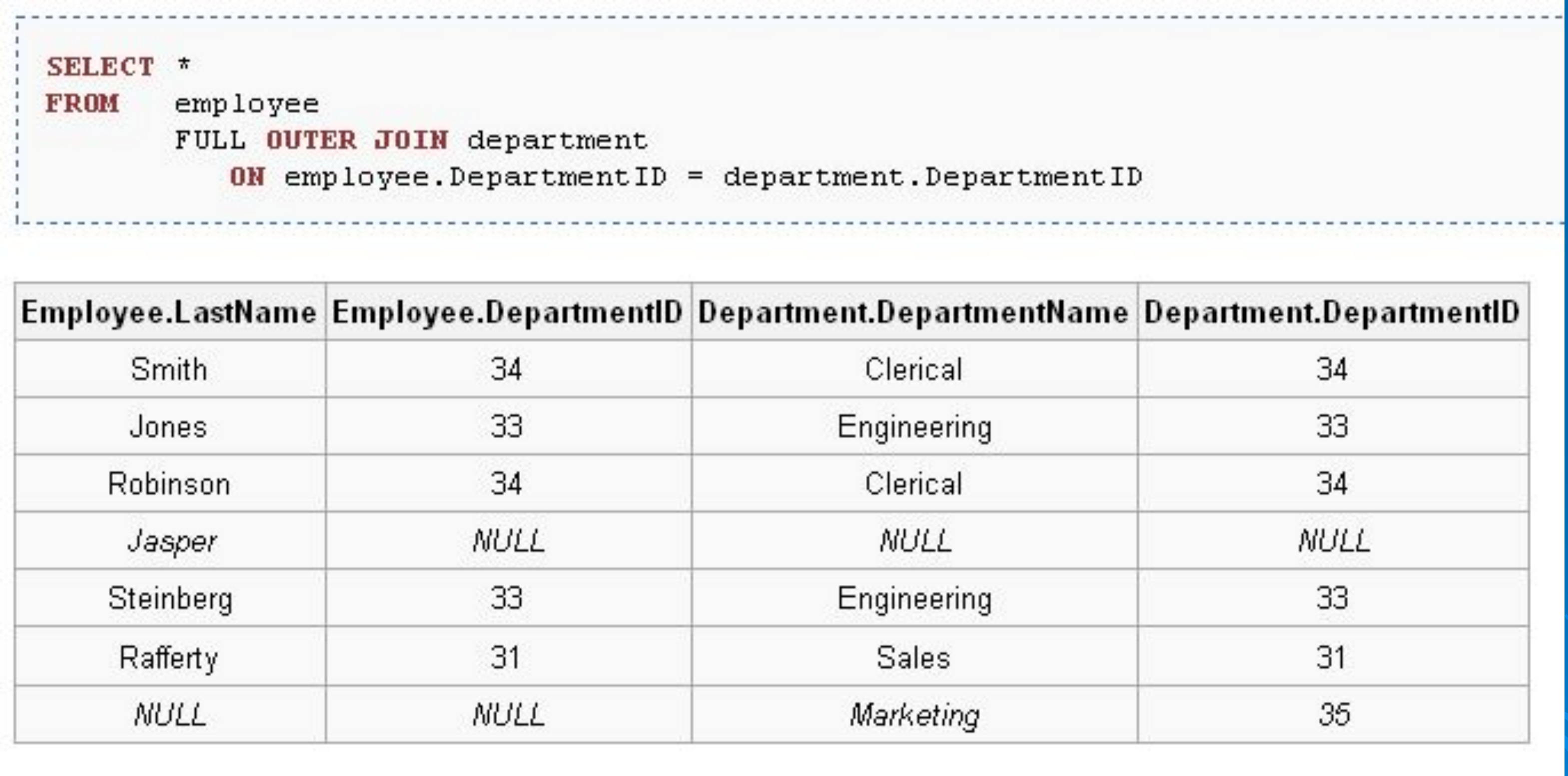
Outer Join (Right)



```
SELECT *
FROM   employee RIGHT OUTER JOIN department
          ON employee.DepartmentID = department.DepartmentID
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	*NULL*	*Marketing*	*35*

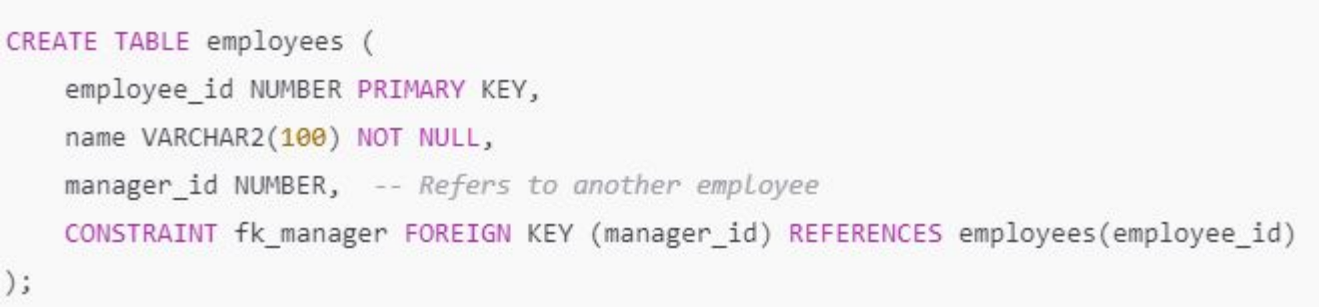
Full Outer Join



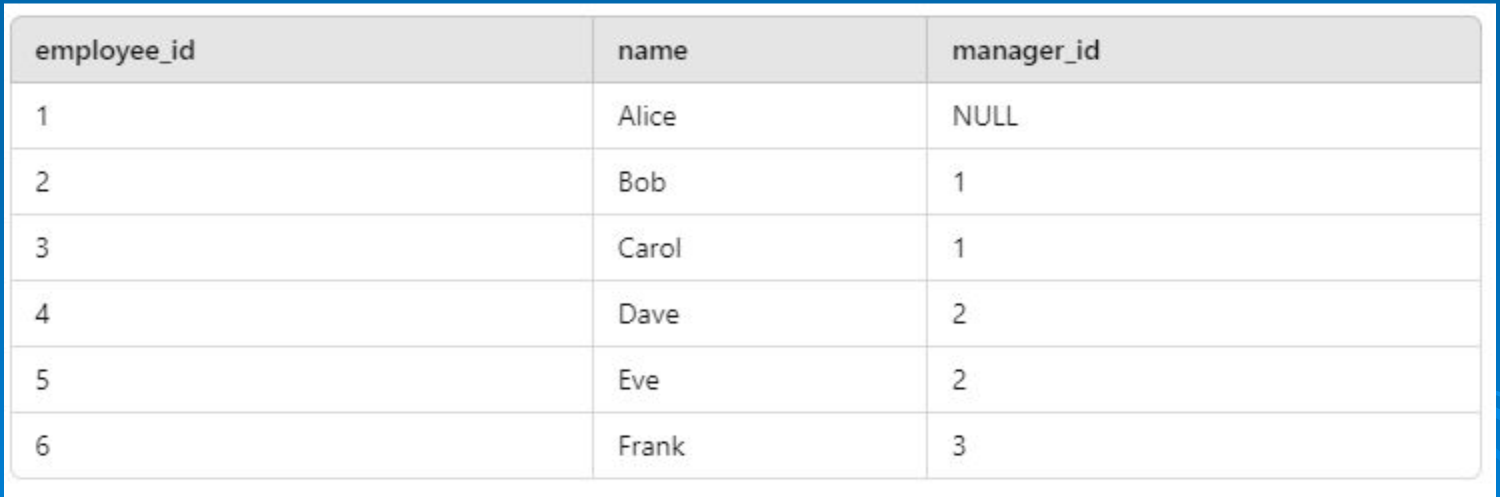
```
SELECT *
FROM   employee
       FULL OUTER JOIN department
          ON employee.DepartmentID = department.DepartmentID
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Jasper	*NULL*	*NULL*	*NULL*
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	*NULL*	*Marketing*	*35*

Self Join

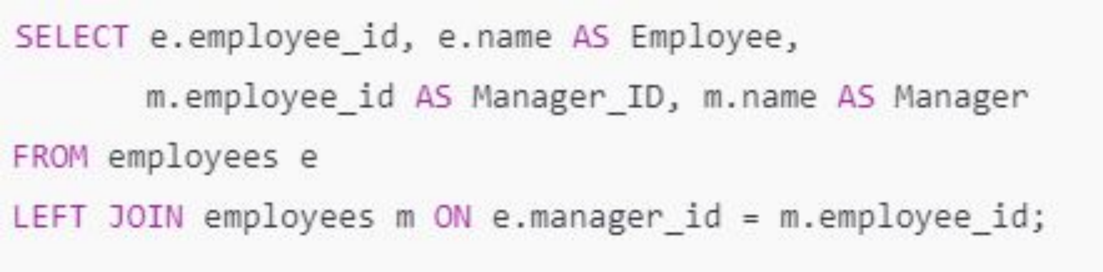


```
CREATE TABLE employees (
    employee_id NUMBER PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    manager_id NUMBER,  -- Refers to another employee
    CONSTRAINT fk_manager FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
);
```

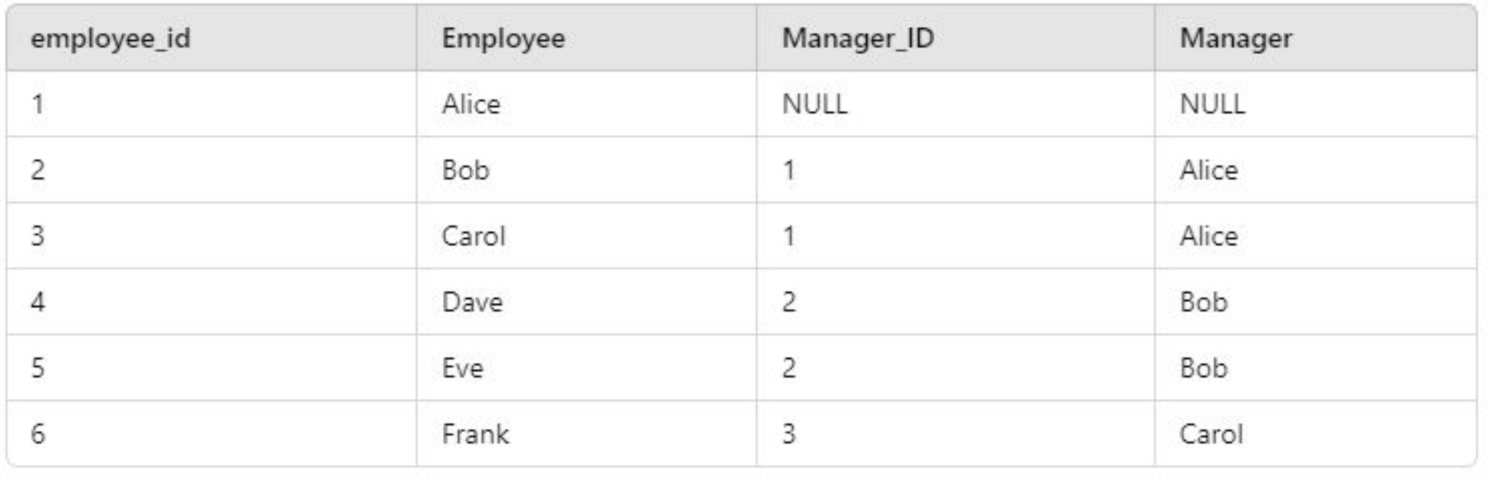


employee_id	name	manager_id
1	Alice	NULL
2	Bob	1
3	Carol	1
4	Dave	2
5	Eve	2
6	Frank	3

Self Join

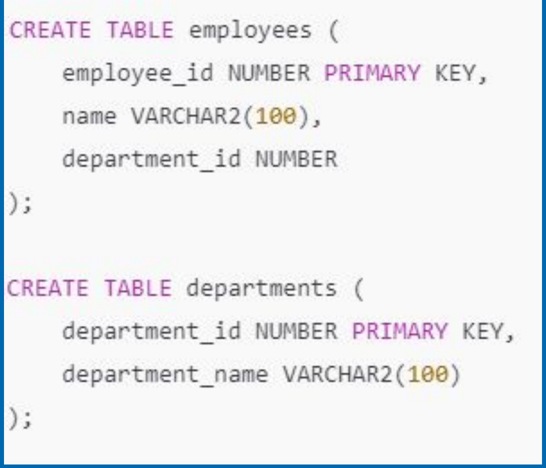


```sql
SELECT e.employee_id, e.name AS Employee,
       m.employee_id AS Manager_ID, m.name AS Manager
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.employee_id;
```



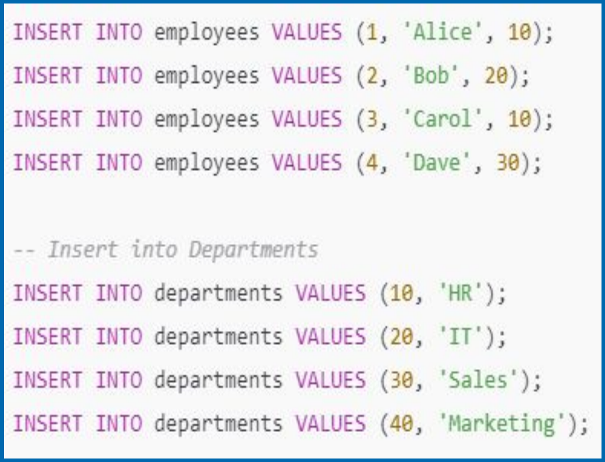
employee_id	Employee	Manager_ID	Manager
1	Alice	NULL	NULL
2	Bob	1	Alice
3	Carol	1	Alice
4	Dave	2	Bob
5	Eve	2	Bob
6	Frank	3	Carol

Natural Join



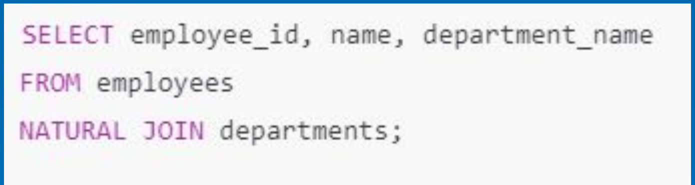
```
CREATE TABLE employees (
    employee_id NUMBER PRIMARY KEY,
    name VARCHAR2(100),
    department_id NUMBER
);

CREATE TABLE departments (
    department_id NUMBER PRIMARY KEY,
    department_name VARCHAR2(100)
);
```



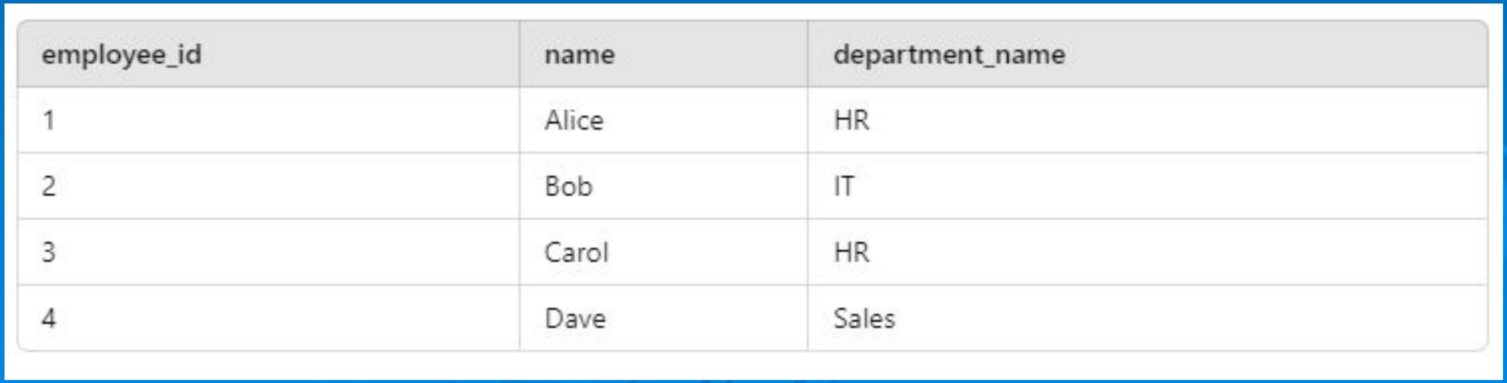
```
INSERT INTO employees VALUES (1, 'Alice', 10);
INSERT INTO employees VALUES (2, 'Bob', 20);
INSERT INTO employees VALUES (3, 'Carol', 10);
INSERT INTO employees VALUES (4, 'Dave', 30);

-- Insert into Departments
INSERT INTO departments VALUES (10, 'HR');
INSERT INTO departments VALUES (20, 'IT');
INSERT INTO departments VALUES (30, 'Sales');
INSERT INTO departments VALUES (40, 'Marketing');
```



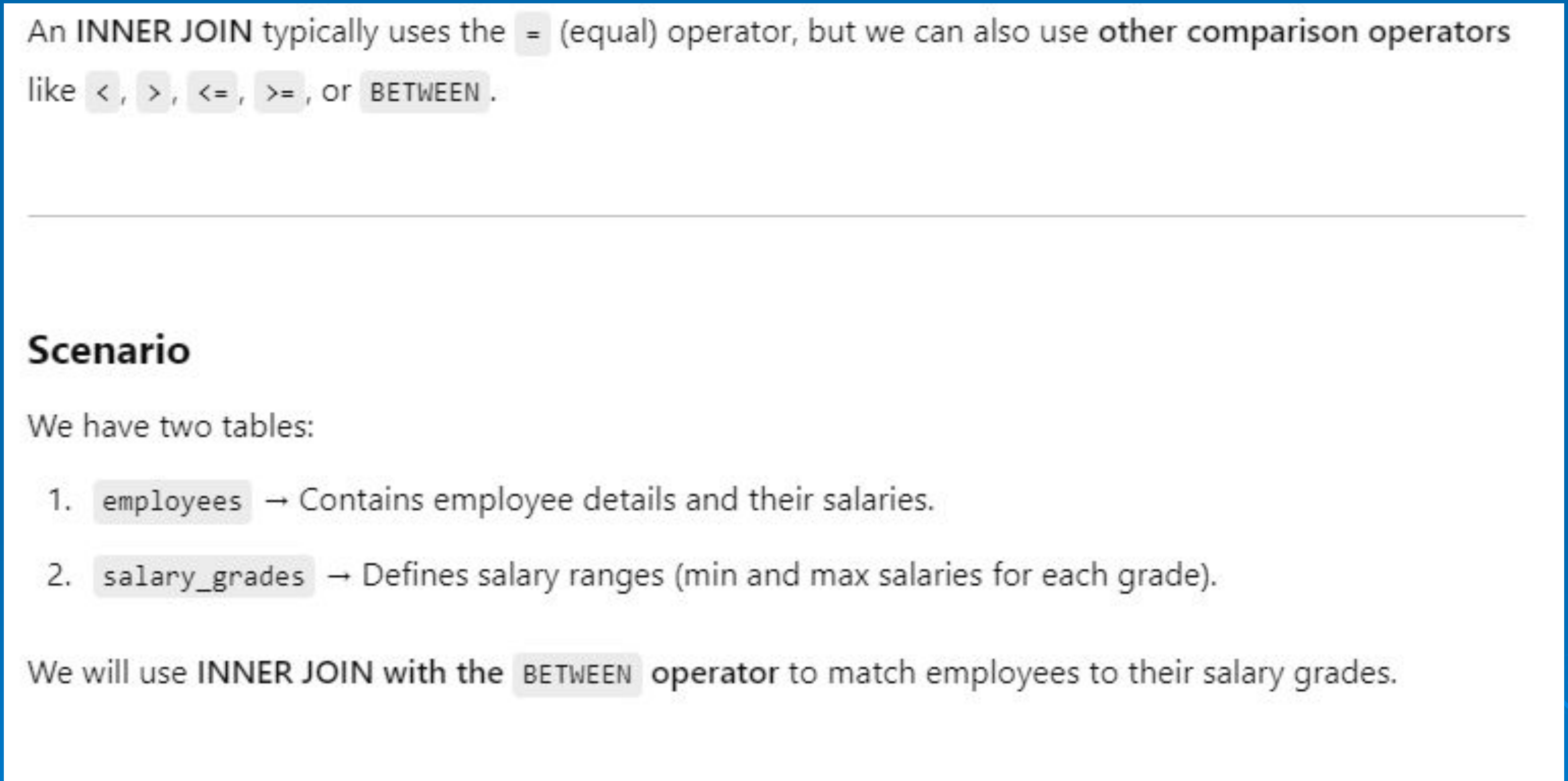
```
SELECT employee_id, name, department_name
FROM employees
NATURAL JOIN departments;
```

- Both tables have department_id (Common column).
- Oracle automatically joins them using department_id in a NATURAL JOIN.
- Only matching records are included (like an INNER JOIN).



employee_id	name	department_name
1	Alice	HR
2	Bob	IT
3	Carol	HR
4	Dave	Sales

INNER JOIN Without Equal (=) Operator



An **INNER JOIN** typically uses the `=` (equal) operator, but we can also use **other comparison operators** like `<`, `>`, `<=`, `>=`, or `BETWEEN`.

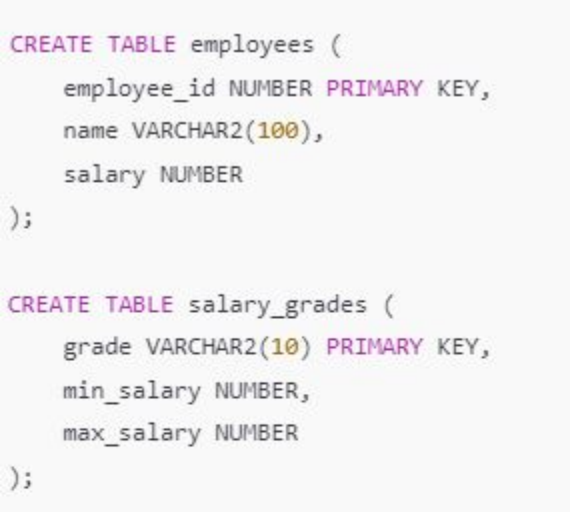
Scenario

We have two tables:

1. `employees` → Contains employee details and their salaries.
2. `salary_grades` → Defines salary ranges (min and max salaries for each grade).

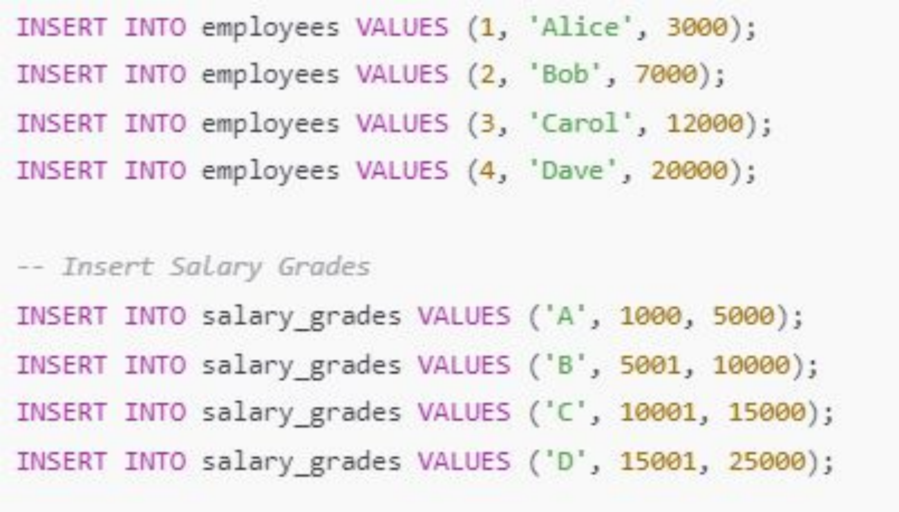
We will use **INNER JOIN with the** `BETWEEN` **operator** to match employees to their salary grades.

INNER JOIN Without Equal (=) Operator



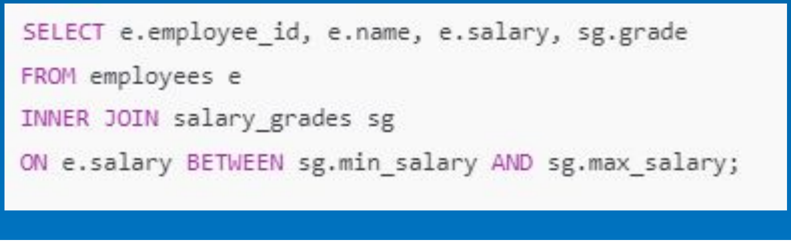
```
CREATE TABLE employees (
    employee_id NUMBER PRIMARY KEY,
    name VARCHAR2(100),
    salary NUMBER
);

CREATE TABLE salary_grades (
    grade VARCHAR2(10) PRIMARY KEY,
    min_salary NUMBER,
    max_salary NUMBER
);
```

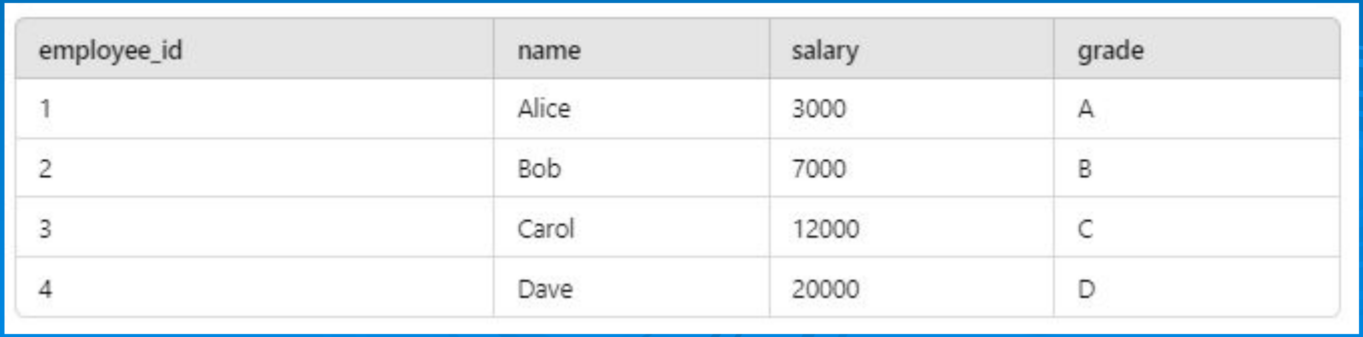


```
INSERT INTO employees VALUES (1, 'Alice', 3000);
INSERT INTO employees VALUES (2, 'Bob', 7000);
INSERT INTO employees VALUES (3, 'Carol', 12000);
INSERT INTO employees VALUES (4, 'Dave', 20000);

-- Insert Salary Grades
INSERT INTO salary_grades VALUES ('A', 1000, 5000);
INSERT INTO salary_grades VALUES ('B', 5001, 10000);
INSERT INTO salary_grades VALUES ('C', 10001, 15000);
INSERT INTO salary_grades VALUES ('D', 15001, 25000);
```



```
SELECT e.employee_id, e.name, e.salary, sg.grade
FROM employees e
INNER JOIN salary_grades sg
ON e.salary BETWEEN sg.min_salary AND sg.max_salary;
```



employee_id	name	salary	grade
1	Alice	3000	A
2	Bob	7000	B
3	Carol	12000	C
4	Dave	20000	D

Aggregations

- SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- Also, COUNT(*) counts the number of tuples.

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

Using AVG and SUM Functions

- You can use AVG and SUM for numeric data.

```
SQL> SELECT   AVG(sal), MAX(sal),
  2      MIN(sal), SUM(sal)
  3 FROM    emp
  4 WHERE   job LIKE 'SALES%';
```

```
AVG(SAL)  MAX(SAL)  MIN(SAL)  SUM(SAL)
-------- --------- --------- ---------
    1400      1600      1250      5600
```

Using MIN and MAX Functions

- You can use MIN and MAX for any datatype.

```
SQL> SELECT   MIN(hiredate), MAX(hiredate)
  2  FROM emp;
```

```
MIN(HIRED MAX(HIRED
--------- ---------
17-DEC-80 12-JAN-83
```

Using the COUNT Function

- COUNT(*) returns the number of rows in a table.

```
SQL> SELECT   COUNT(*)
  2  FROM emp
  3  WHERE    deptno = 30;
```

```
 COUNT(*)
---------
        6
```

Using the COUNT Function

- COUNT(*expr*) returns the number of nonnull rows.

```
SQL> SELECT   COUNT(comm)
  2  FROM emp
  3  WHERE    deptno = 30;
```

```
COUNT(COMM)
-----------
          4
```

Group Functions and Null Values

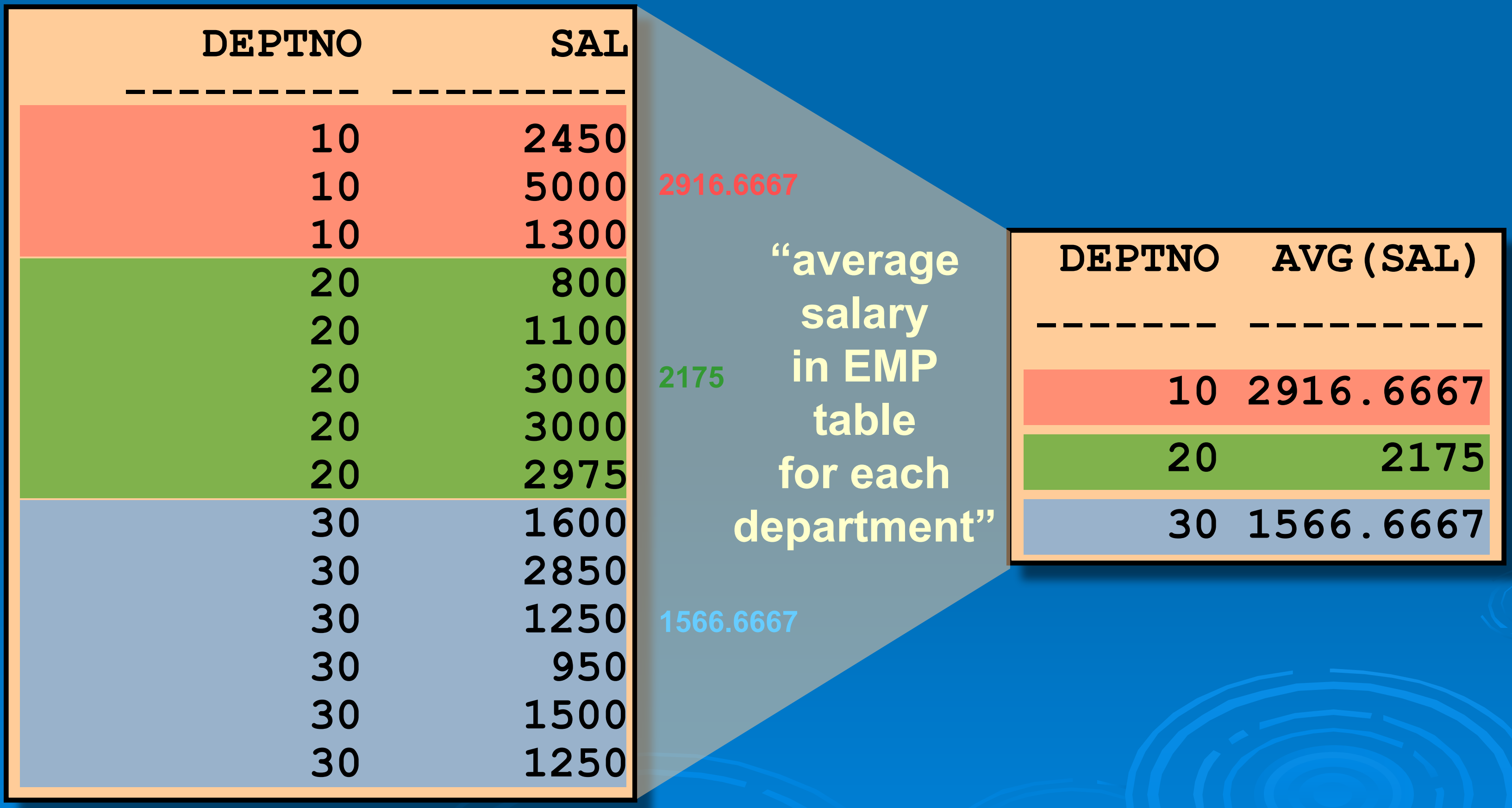
- Group functions ignore null values in the column.

```
SQL> SELECT AVG(comm)
  2  FROM   emp;
```

```
 AVG(COMM)
---------
      550
```

Creating Groups of Data

EMP



```
    DEPTNO        SAL
---------- ----------
        10       2450
        10       5000
        10       1300
        20        800
        20       1100
        20       3000
        20       3000
        20       2975
        30       1600
        30       2850
        30       1250
        30        950
        30       1500
        30       1250
```

"average salary in EMP table for each department"

```
  DEPTNO   AVG(SAL)
-------- ----------
      10  2916.6667
      20       2175
      30  1566.6667
```

Creating Groups of Data: GROUP BY Clause

```
SELECT column, group_function(column)
FROM       table
[WHERE condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

- Divide rows in a table into smaller groups by using the GROUP BY clause.

Using the GROUP BY Clause

- All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SQL> SELECT    deptno, AVG(sal)
  2  FROM      emp
  3  GROUP BY deptno;
```

```
   DEPTNO   AVG(SAL)
--------- ----------
       10 2916.6667
       20       2175
       30 1566.6667
```

Using the GROUP BY Clause

- The GROUP BY column does not have to be in the SELECT list.

```
SQL> SELECT    AVG(sal)
  2  FROM      emp
  3  GROUP BY deptno;
```

```
 AVG(SAL)
---------
2916.6667
     2175
1566.6667
```