Advanced SQL

Joins and Subqueries

Reading Data from Multiple Tables

- Subqueries (Implemented as Nested Queries)
  - Retrieves data from one table and the results could be used for further filtration of data.
- Joins (Combing multiple tables on-the-go)
  - Most common Types of Joins are
    - Equijoin / Inner Join
    - Natural joins
    - Self join
    - Non-equijoin
    - Outer join
    - Cross Join

Processing Multiple Tables Using Joins

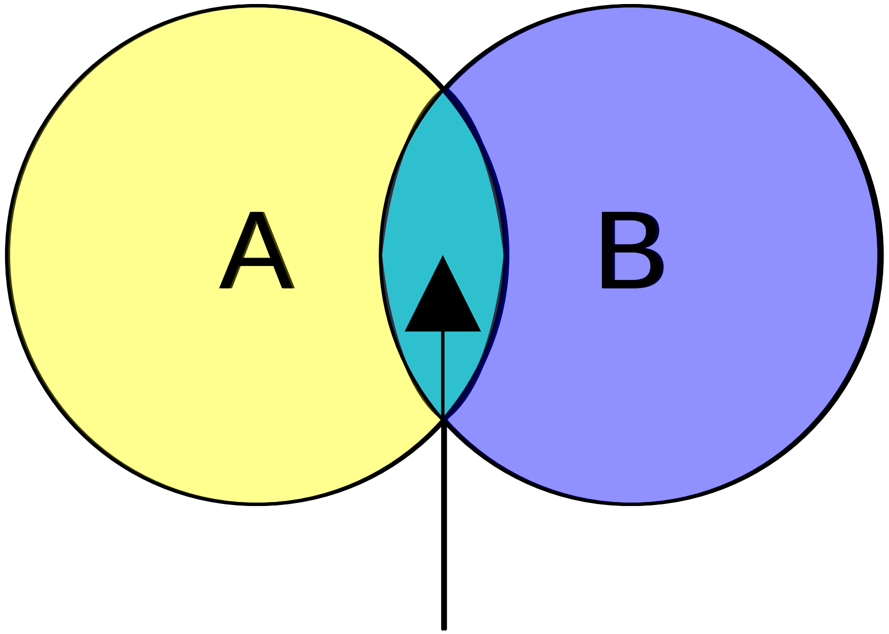
- Join - Most frequently used operation - brings together data from multiple tables into one resultant table

- Join can be achieved in two ways

  - Implicitly by referring in a WHERE clause to the matching of common columns over which the tables are joined

  - Explicitly by JOIN.....ON commands in FROM clause

SQL Joins: Defining Join Types: **`INNER JOIN`**



INTERSECTION

SQL Joins Defining Join Types: `INNER JOIN`

- An **`INNER JOIN`** is also an *equijoin*, or equality join between equals.
- An **`INNER JOIN`** matches on one or a set of columns values from one table:
  - When one table is involved, an **`INNER JOIN`** creates an intersection between two copies of a single table (typically done with two different column names).
  - When two or more tables are involved, an **`INNER JOIN`** creates an intersection between the tables based on designated column names.

Defining Join Types: `INNER JOIN`

- Create an `INNER JOIN` by placing a position specific set of tables in the `FROM` clause followed by an `ON` or `USING` clause.
- Equality statements are between one or more columns in two copies of one table or two tables:
- When the columns share the same name and data type,
  - use the `USING` clause.
- When the columns have different names but the same data type,
  - use the `ON` clause.
- If only the word `JOIN` is used, an `INNER JOIN` is assumed by the SQL parser.

Defining Join Types: INNER JOIN

-
  ```
  SELECT  a.column1, b.column2
       FROM   table1 a, table2 b
        WHERE  a.columnpk =
  b.columnfk;
  ```

-
  ```
   SELECT   a.column1, b.column2
        FROM   table1 a [INNER] JOIN table2 b
          ON     a.columnpk = b.columnfk;
  ```

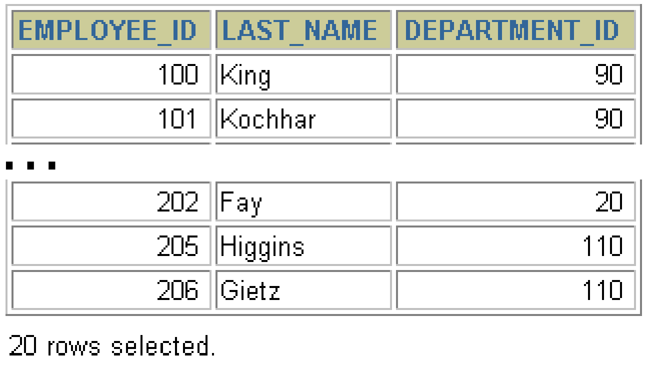
-
  ```
   SELECT   a.column1, b.column2
        FROM     table1 a [INNER] JOIN table2
  b
          USING (same_column_name);
  ```

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
- To avoid a Cartesian product, always include a valid join condition.

Generating a Cartesian Product

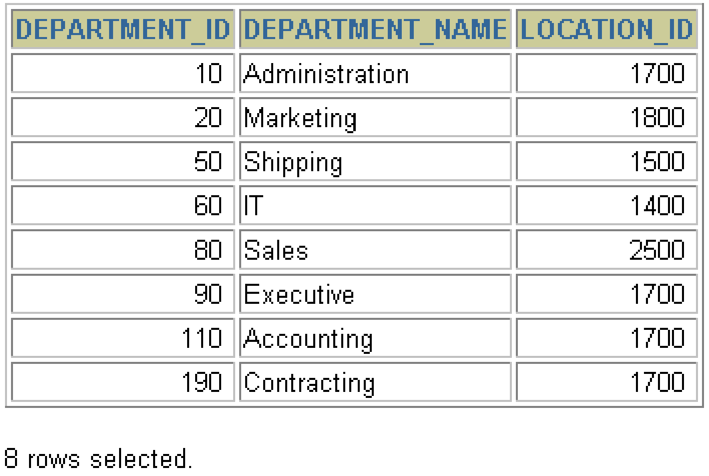
EMPLOYEES (20 rows)



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
…		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected.

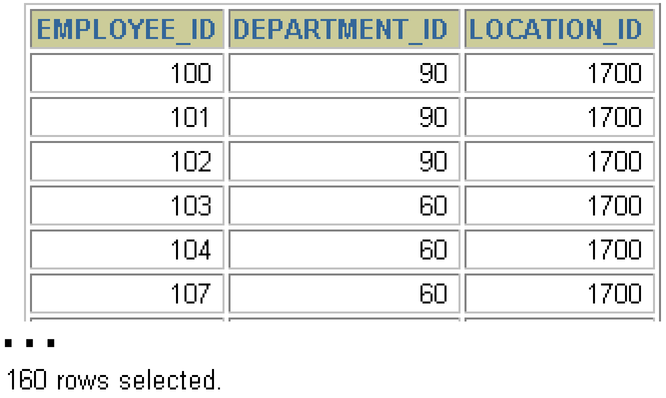
DEPARTMENTS (8 rows)



DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

Cartesian product:
20 x 8 = 160 rows



EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700

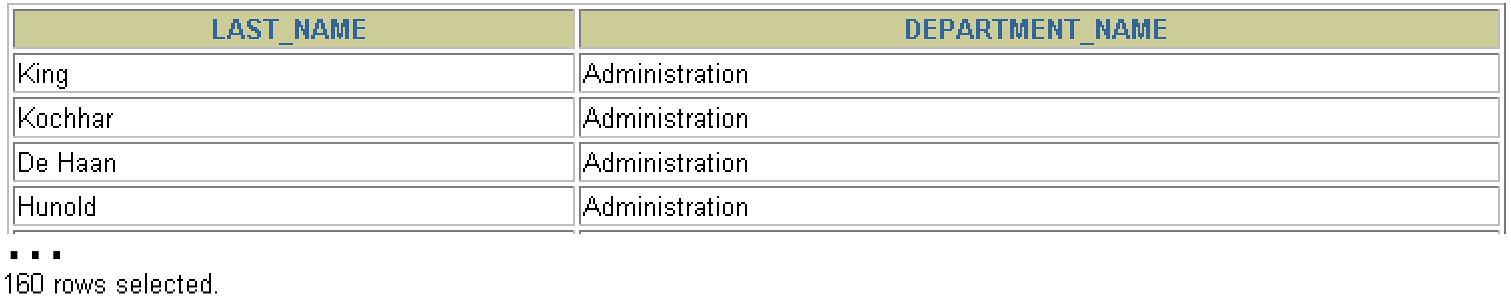
…

160 rows selected.

Creating Cross Joins

- The `CROSS JOIN` clause 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 ;
```



LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration

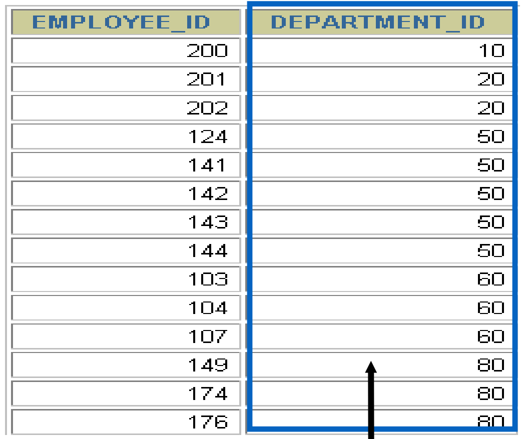
...

160 rows selected.

Retrieving Record with Equijoin

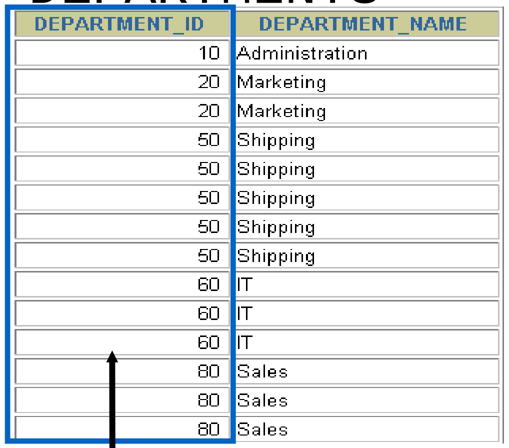
Employees ∞ Department

EMPLOYEES



EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

DEPARTMENTS



DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales

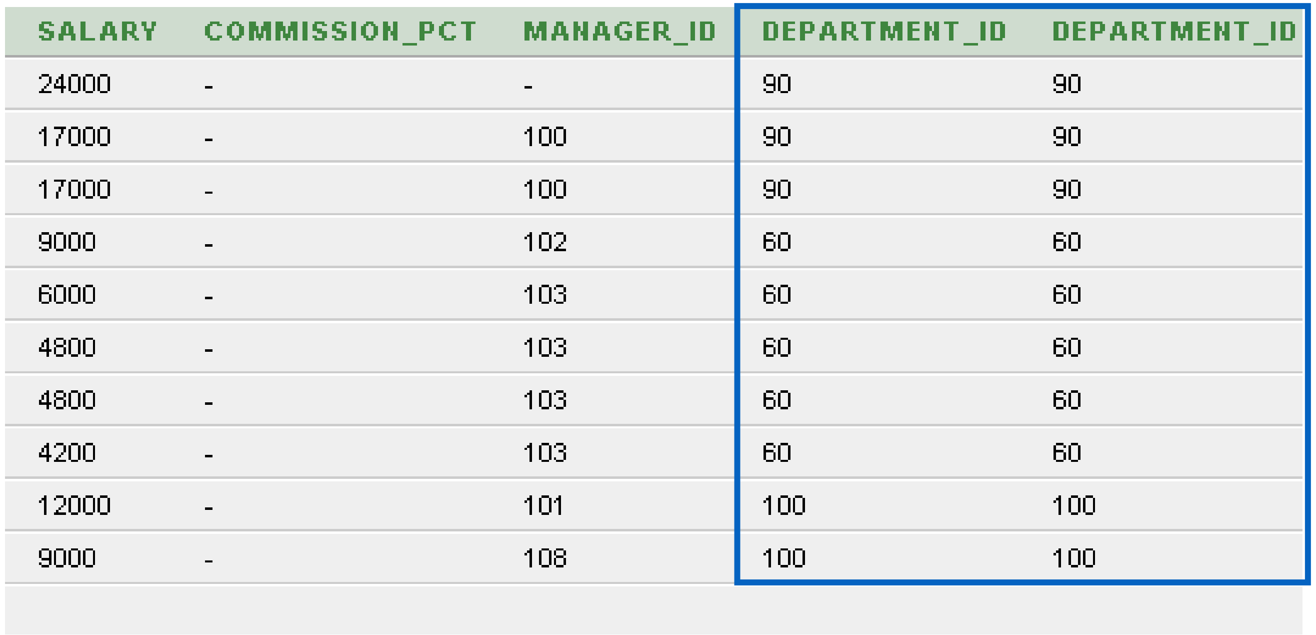
Foreign key

Primary key

Using Equijoin

Write SQL statement to do this: Employees ∞ Department

```
Select  *
From  employees ,departments
Where employees.department_id = departments.department_id
```



SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	DEPARTMENT_ID
24000	-	-	90	90
17000	-	100	90	90
17000	-	100	90	90
9000	-	102	60	60
6000	-	103	60	60
4800	-	103	60	60
4800	-	103	60	60
4200	-	103	60	60
12000	-	101	100	100
9000	-	108	100	100

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

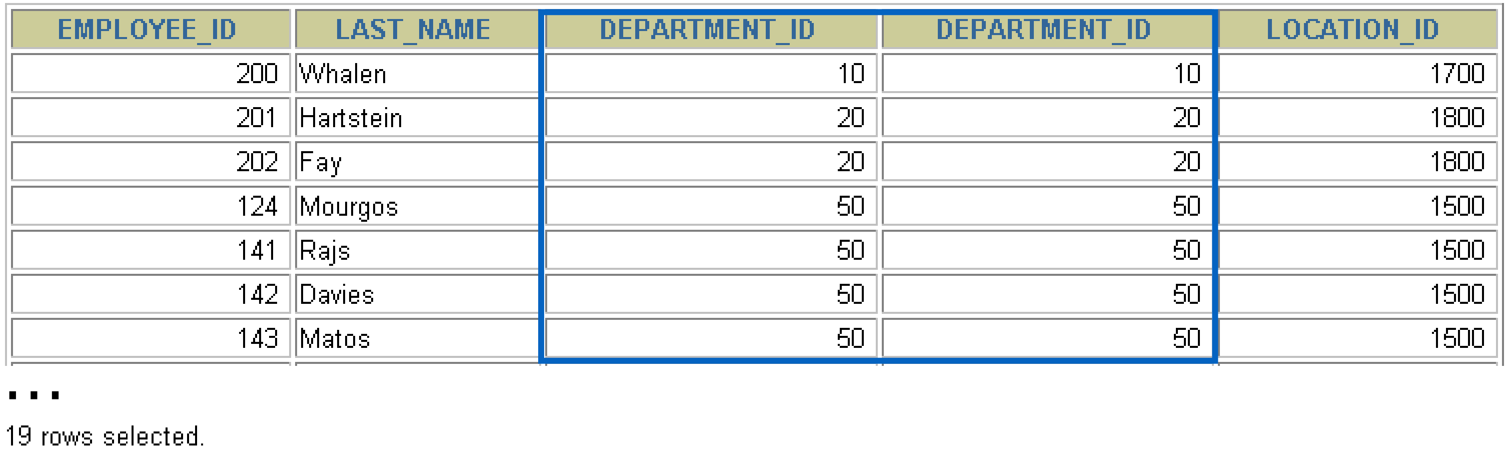
Using Table Aliases

- Use table aliases to simplify queries.
- Use table aliases to improve performance.

```
SELECT e.employee_id, e.last_name,
       d.location_id, department_id
FROM   employees e INNER JOIN departments d
USING (department_id) ;
```

Retrieving Records with the ON Clause

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



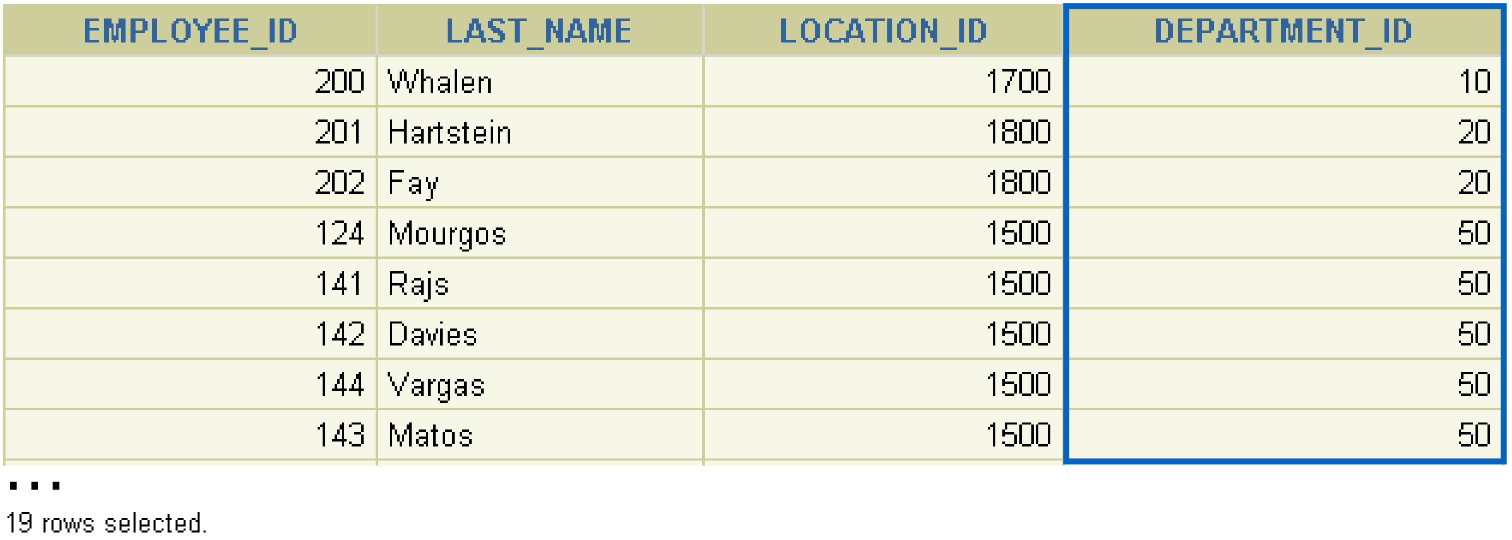
EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

19 rows selected.

Retrieving Records with the USING Clause

```
SELECT employees.employee_id, employees.last_name,
       departments.location_id, department_id
FROM   employees INNER JOIN departments
USING (department_id) ;
```



EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
124	Mourgos	1500	50
141	Rajs	1500	50
142	Davies	1500	50
144	Vargas	1500	50
143	Matos	1500	50

...

19 rows selected.

```
SELECT s.sid, s.name, r.bid
FROM Sailors s INNER JOIN Reserves r
ON s.sid = r.sid
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/96
95	103	11/12/96

What is the result of above query???

Joins Example

- Show all customers and order date who have placed an order

```
• SELECT CUSTOMER_NAME , ORDER_DATE
        FROM CUSTOMER, ORDER
                    WHERE CUSTOMER.CUSTOMER_ID = ORDER.CUSTOMER_ID
```

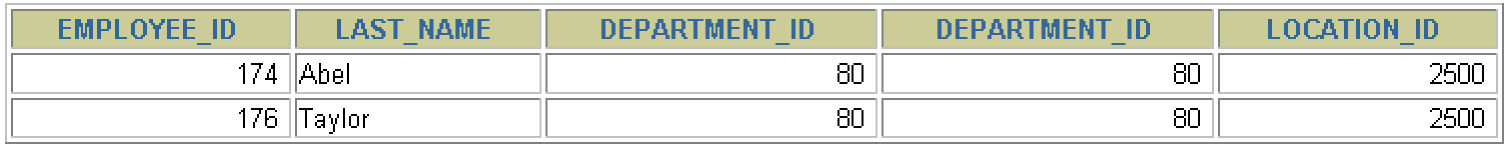
```
SELECT CUSTOMER_NAME , ORDER_DATE
            FROM CUSTOMER INNER JOIN ORDER
            ON       CUSTOMER.CUSTOMER_ID =
ORDER.CUSTOMER_ID
```

```
SELECT CUSTOMER_NAME , ORDER_DATE
   FROM CUSTOMER INNER JOIN ORDER
            USING CUSTOMER_ID
```

Applying Additional Conditions to a Join

- Show employee id , last name, dept id and location id who have a manager ID 149.

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



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

Joins Example

- Show the students’ name and marks who failed in course CSC271

-
```
SELECT    S.std_name,    R.marks
          FROM    Student S    INNER JOIN    Result R
          ON        S.std_id = R.std_id
          AND       R.marks<50       AND         course_id =
‘CSC271’
```

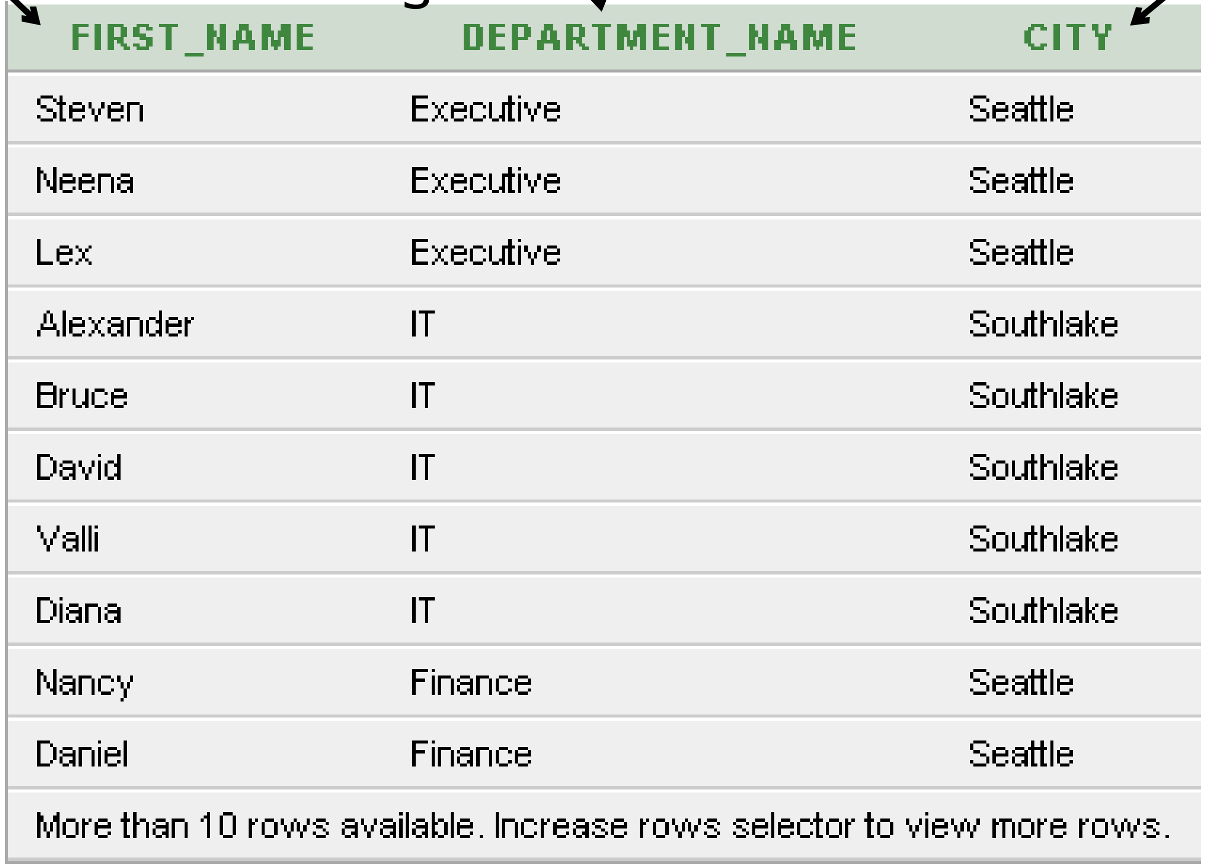
-
```
SELECT    S.std_name,    R.marks
          FROM    Student S    INNER JOIN    Result R
          USING    std_id
          AND       R.marks<50       AND         course_id =
‘CSC271’
```

Joining More than two table

Employees

Departments

Locations



FIRST_NAME	DEPARTMENT_NAME	CITY
Steven	Executive	Seattle
Neena	Executive	Seattle
Lex	Executive	Seattle
Alexander	IT	Southlake
Bruce	IT	Southlake
David	IT	Southlake
Valli	IT	Southlake
Diana	IT	Southlake
Nancy	Finance	Seattle
Daniel	Finance	Seattle

More than 10 rows available. Increase rows selector to view more rows.

Joining More than two table

```
select first_name,department_name,city
from employees E,departments D,locations L
where E.department_id=D.department_id
      and D.location_id=L.location_id
```

```
select first_name,department_name,city
from employees
JOIN departments
ON(employees.department_id=departments.department_id)
JOIN locations
ON(departments.location_id=locations.location_id)
```

```
select first_name,department_name,city
from employees JOIN departments using(department_id)
               JOIN locations using(location_id)
```

SQL Joins Defining Join Types: Non-equijoin

- A *non-equijoin* is an indirect match:
  - Occurs when one column value is found in the range between two other column values
  - Uses the **`BETWEEN`** operator.
  - Also occurs when one column value is found by matching against a criterion using an inequality operator.

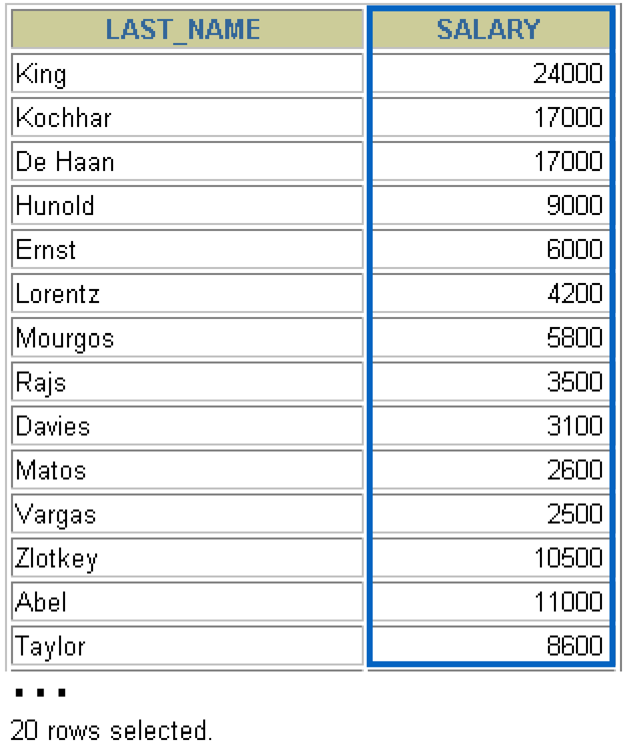
SQL Joins Defining Join Types: Non-equijoin

- Example:

```
SELECT    a.column1, b.column2
FROM      table1 a, table2 b
WHERE     a.columnpk >= b.columnfk;

SELECT    a.column1, b.column2
FROM      table1 a, table2 b
WHERE     a.cola BETWEEN  b.colx AND b.coly;
```

Non-Equijoins

EMPLOYEES

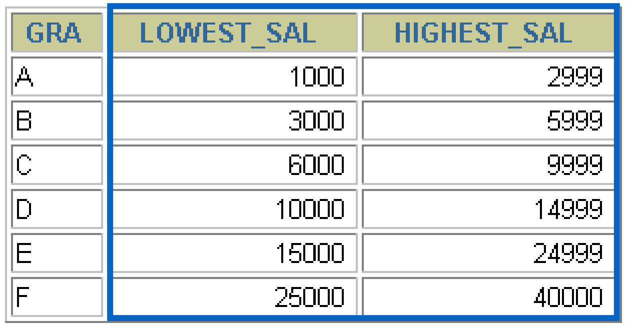


LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...

20 rows selected.

JOB_GRADES

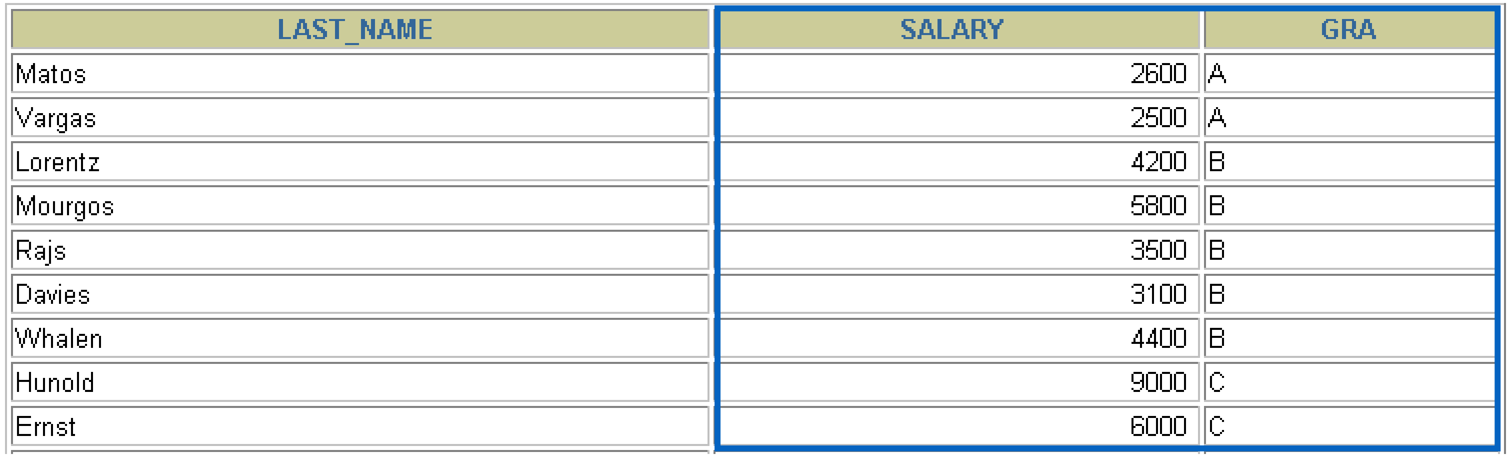


GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB_GRADES table.

Retrieving Records with Non-Equijoins

```
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;
```



LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

20 rows selected.

SQL Joins

Defining Join Types: Natural Join

- We have already learned that an EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables and an equal sign (=) is used as comparison operator in the where clause to refer equality.

- The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only.

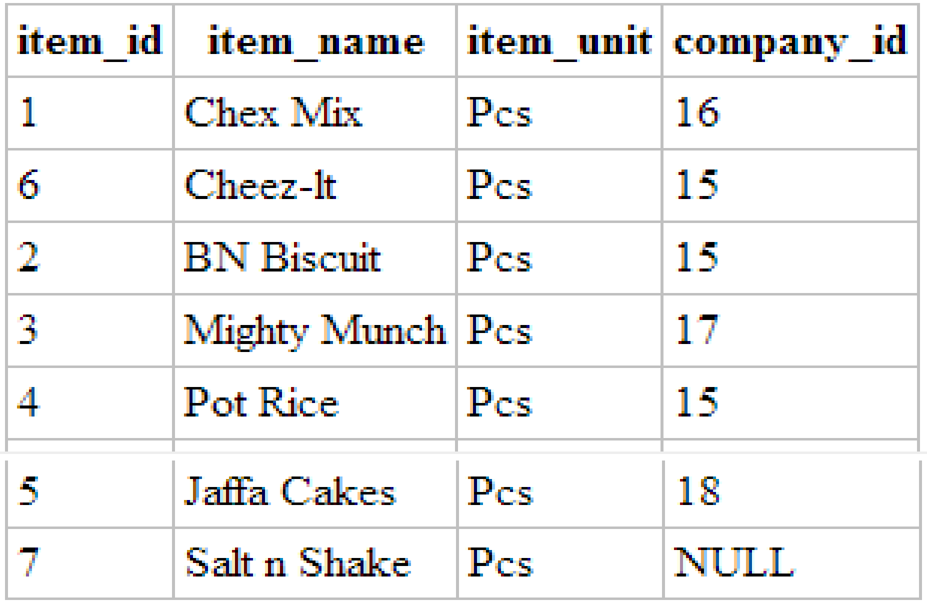
Natural Join : Guidelines

- The associated tables have one or more pairs of identically named columns.
- The columns must be the same data type.
- No need to use ON clause in a natural join.

```
SELECT   a.column1, b.column2
 FROM table1 a NATURAL JOIN table2 b;
```

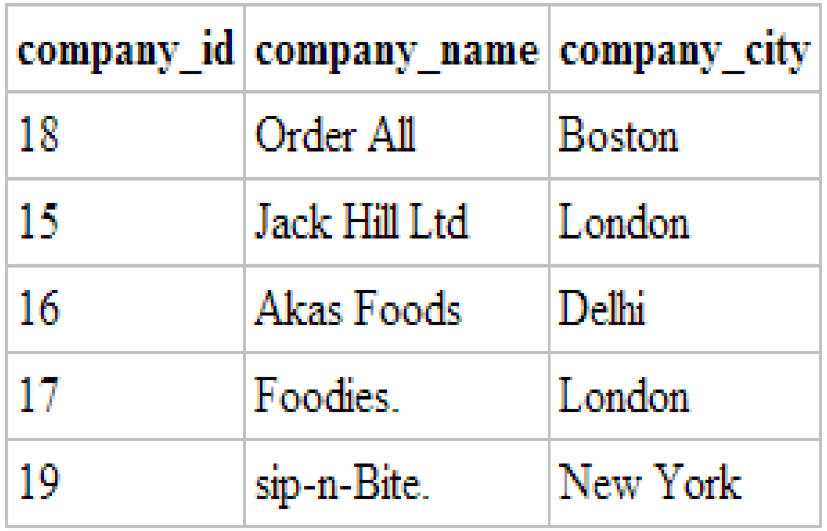
NATURAL JOIN - EXAMPLE

Food



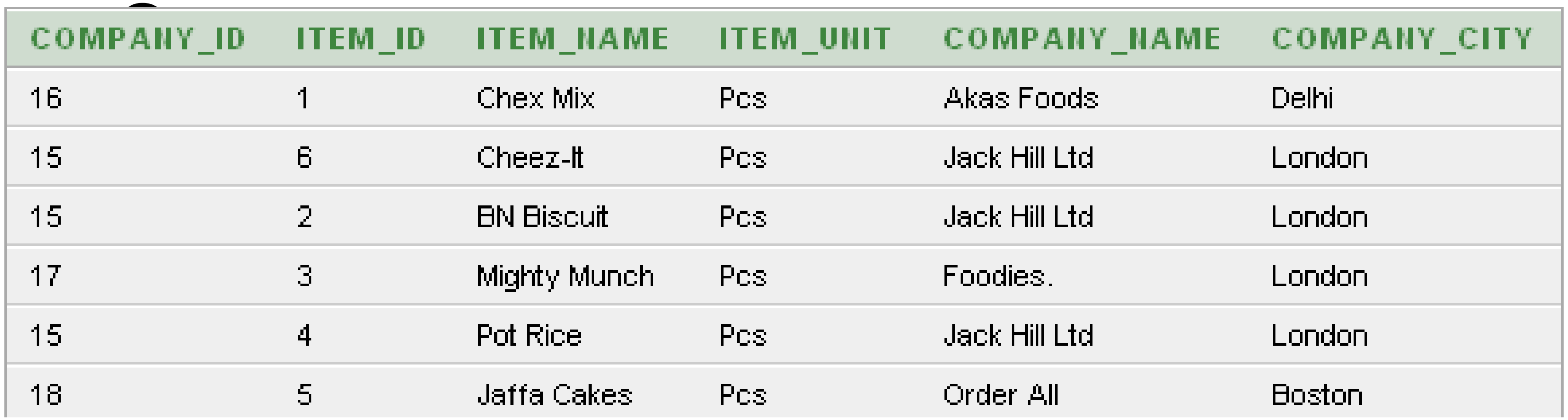
item_id	item_name	item_unit	company_id
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	NULL

COMPANY

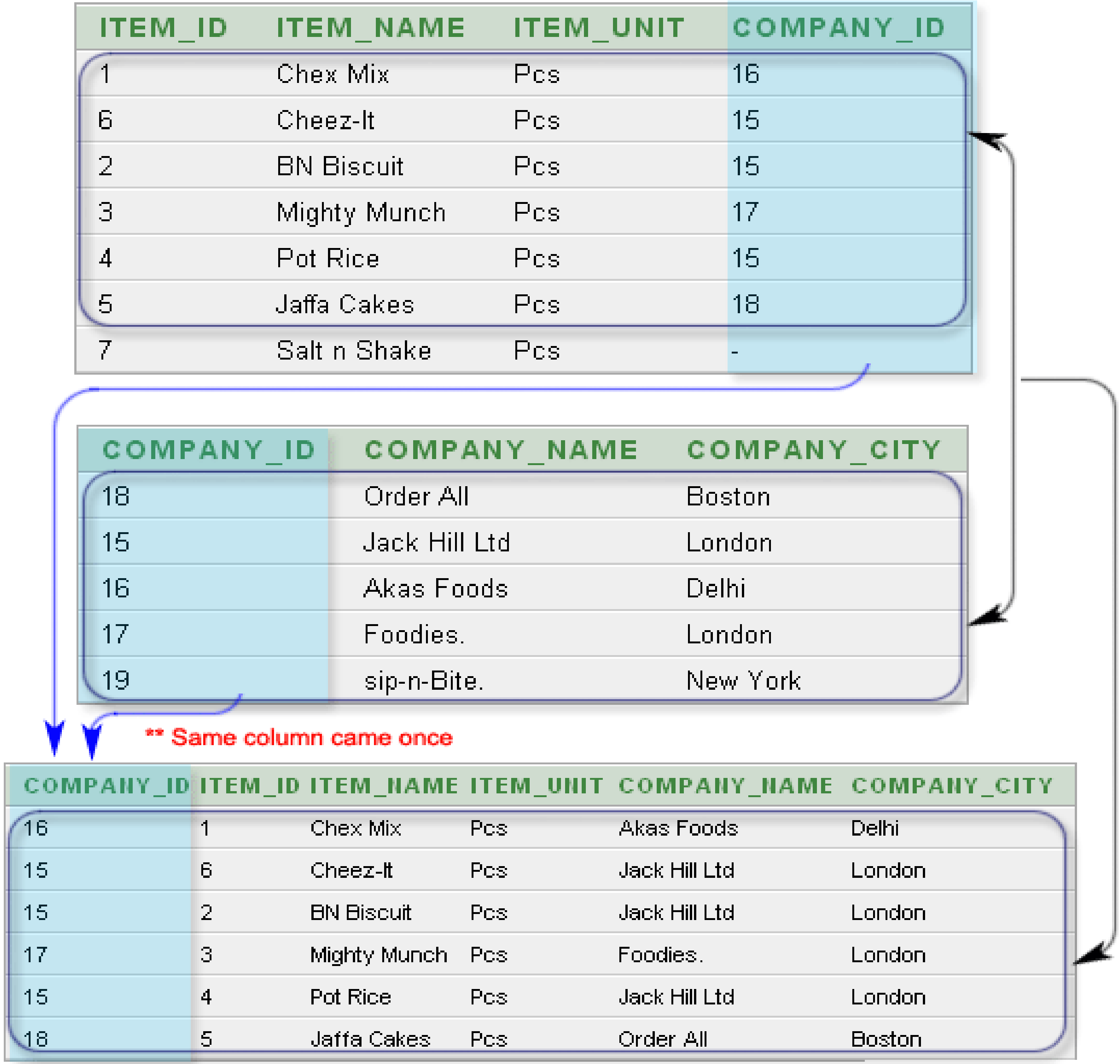


company_id	company_name	company_city
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

- Select * from Food NATURAL JOIN



COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston



ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

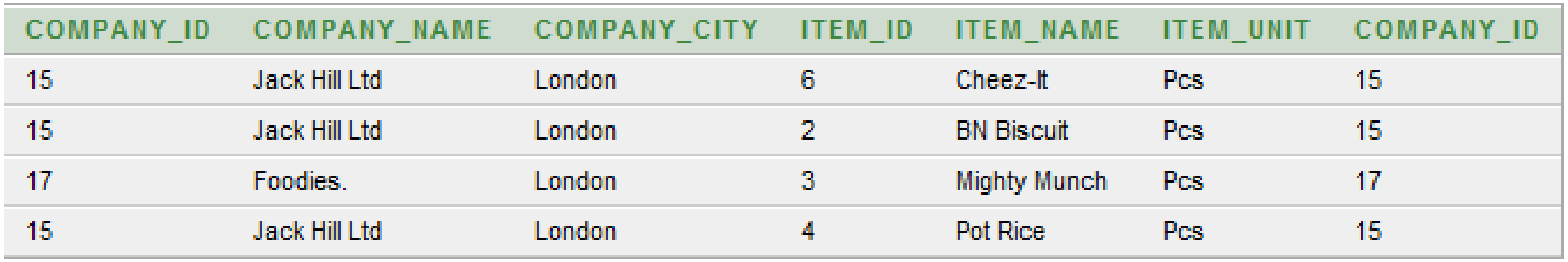
COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

** Same column came once

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

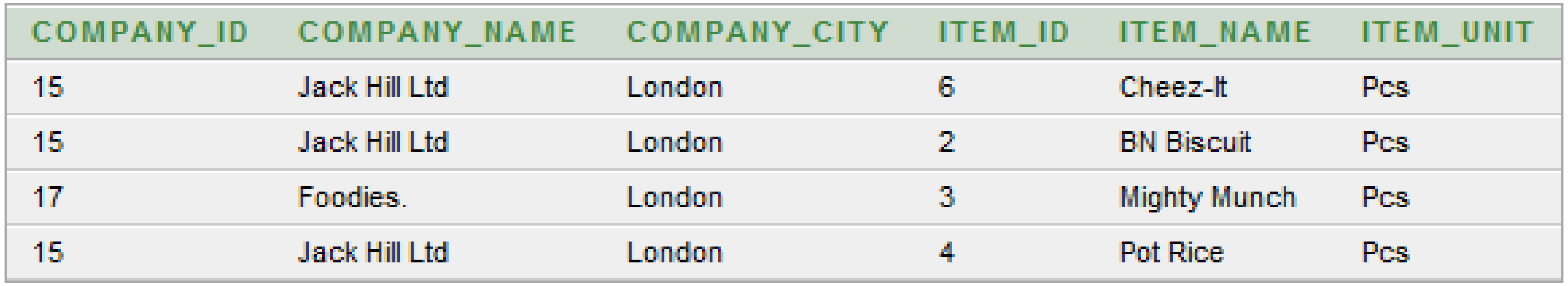
Difference btw INNER JOIN & NATURAL JOIN

- SELECT * FROM company INNER JOIN food ON company.company_id = food.company_id



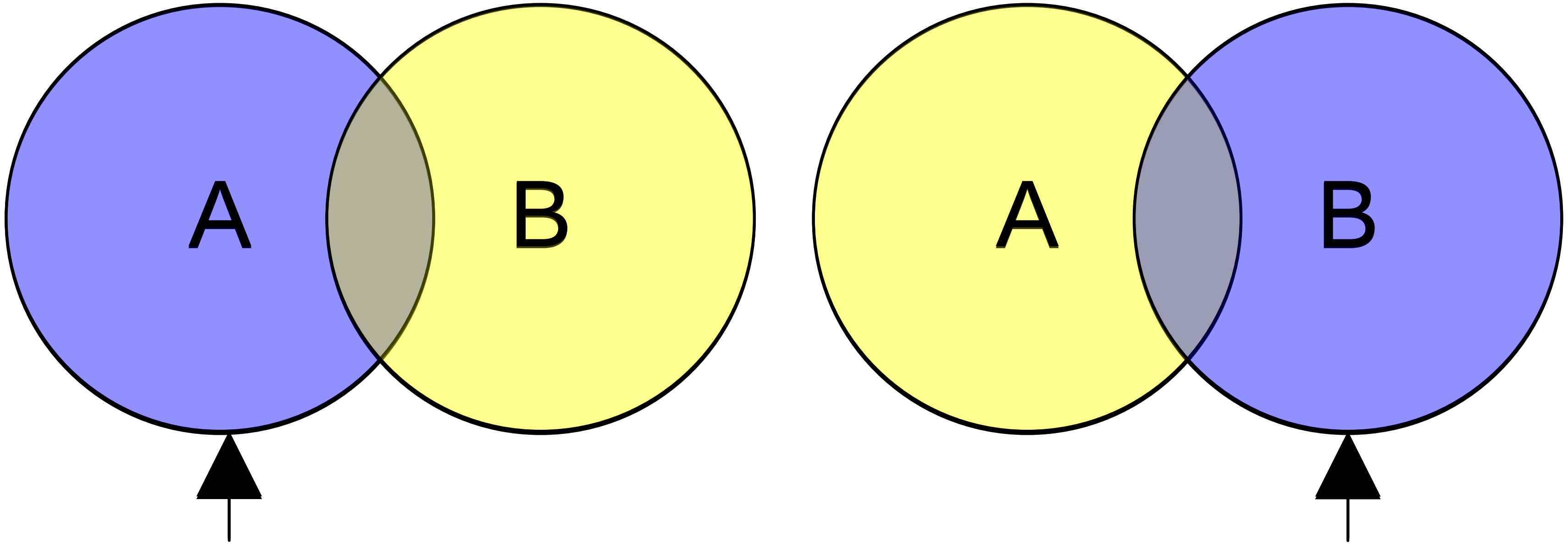
COMPANY_ID	COMPANY_NAME	COMPANY_CITY	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
15	Jack Hill Ltd	London	6	Cheez-It	Pcs	15
15	Jack Hill Ltd	London	2	BN Biscuit	Pcs	15
17	Foodies.	London	3	Mighty Munch	Pcs	17
15	Jack Hill Ltd	London	4	Pot Rice	Pcs	15

- Select * from company NATURAL JOIN food



COMPANY_ID	COMPANY_NAME	COMPANY_CITY	ITEM_ID	ITEM_NAME	ITEM_UNIT
15	Jack Hill Ltd	London	6	Cheez-It	Pcs
15	Jack Hill Ltd	London	2	BN Biscuit	Pcs
17	Foodies.	London	3	Mighty Munch	Pcs
15	Jack Hill Ltd	London	4	Pot Rice	Pcs

SQL Joins Defining Join Types: Outer Join

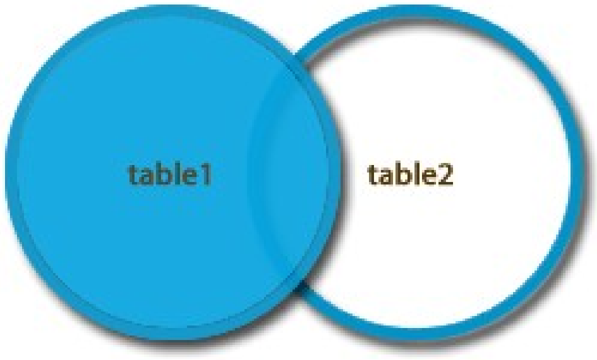


LEFT JOIN RIGHT JOIN

SQL Joins Outer Join

- ANSI Syntax:
  - These are defined by **`LEFT JOIN`** and **`RIGHT JOIN`** operators.
  - Both **`LEFT [OUTER] JOIN`** and **`RIGHT [OUTER] JOIN`** are synonymous with **`LEFT JOIN`** and **`RIGHT JOIN`** respectively, the **`OUTER`** is assumed when left out.
  - The **`LEFT [OUTER] JOIN`** returns all matched rows, plus all unmatched rows from the table on the left of the join clause(use nulls in fields of non-matching tuples)
  - The **`RIGHT [OUTER] JOIN`** returns all matched rows, plus all unmatched rows from the table on the right of the join clause.

Left Outer Join



- ANSI SQL Example:

```
SELECT    a.column1, b.column2
FROM      table1 a LEFT [OUTER] JOIN table2
 b
ON        a.columnpk = b.columnfk;
```

- Oracle Example (left join):

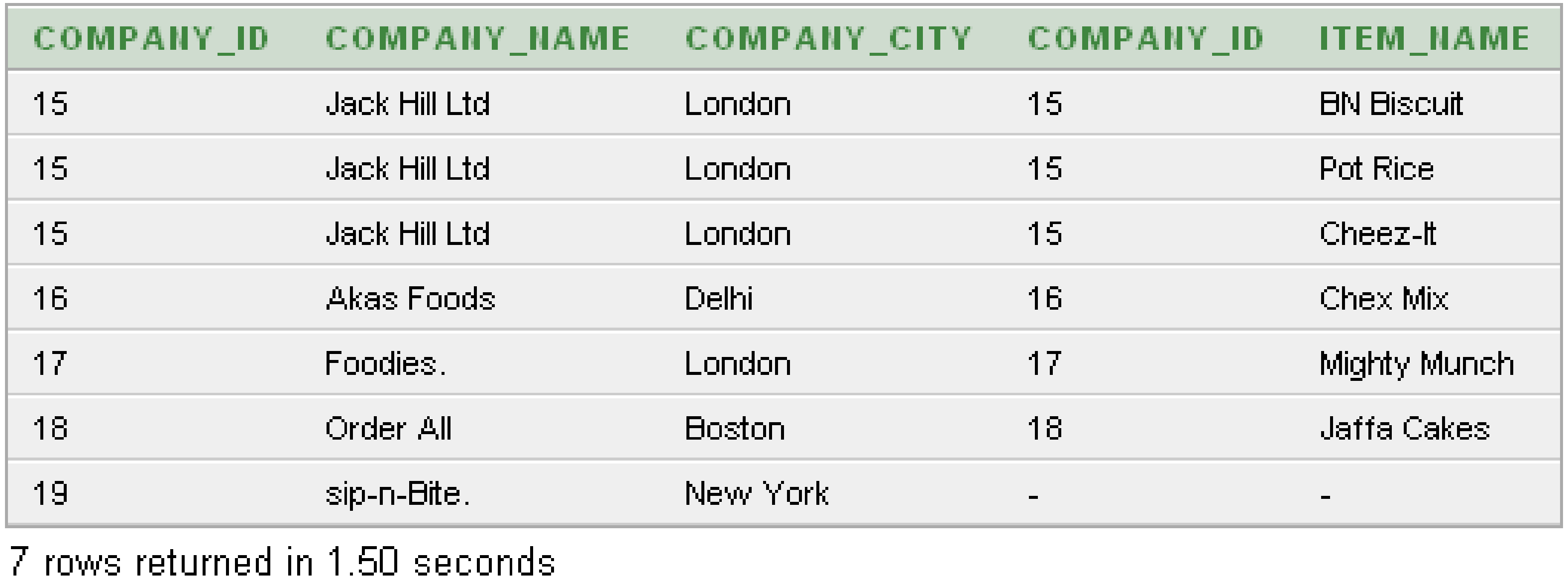
```
SELECT    a.column1, b.column2
FROM      table1 a, table2 b
WHERE     a.columnpk = b.columnfk(+);
```

LEFT OUTER JOIN

- SELECT c.company_id,c.company_name, c.company_city, f.company_id, f.item_name

  FROM company c LEFT OUTER JOIN food f

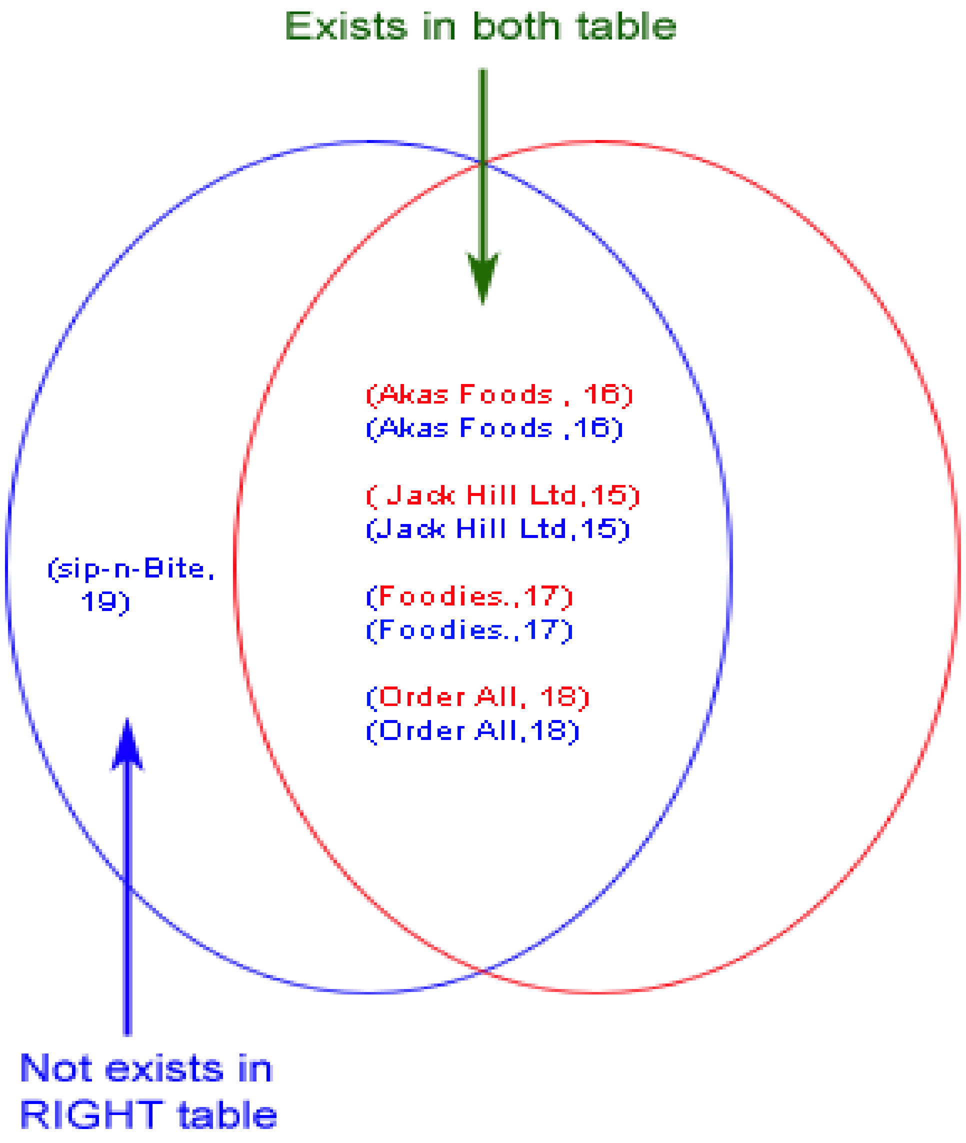
  ON c.company_id = f.company_id;



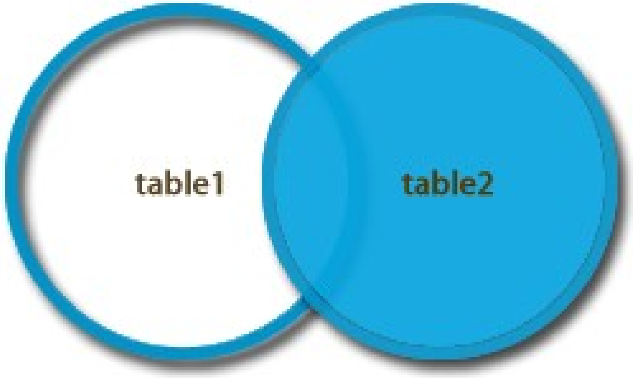
COMPANY_ID	COMPANY_NAME	COMPANY_CITY	COMPANY_ID	ITEM_NAME
15	Jack Hill Ltd	London	15	BN Biscuit
15	Jack Hill Ltd	London	15	Pot Rice
15	Jack Hill Ltd	London	15	Cheez-It
16	Akas Foods	Delhi	16	Chex Mix
17	Foodies.	London	17	Mighty Munch
18	Order All	Boston	18	Jaffa Cakes
19	sip-n-Bite.	New York	-	-

7 rows returned in 1.50 seconds

LEFT OUTER JOIN



Right Outer Join



- ANSI SQL Example:

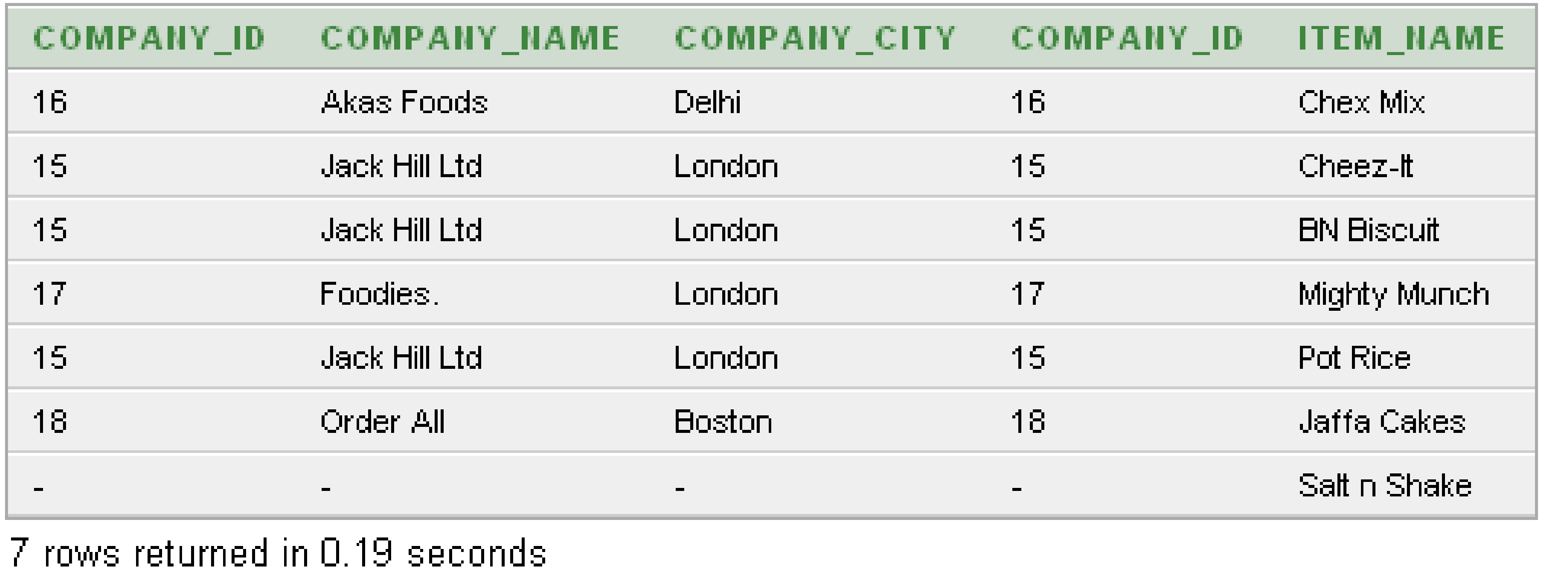
```
SELECT    a.column1, b.column2
FROM      table1 a RIGHT [OUTER] JOIN table2 b
ON        a.columnpk = b.columnfk;
```

- Oracle Example (left join):

```
SELECT    a.column1, b.column2
FROM      table1 a, table2 b
ON        a.columnpk(+) = b.columnfk;
```

RIGHT OUTER JOIN

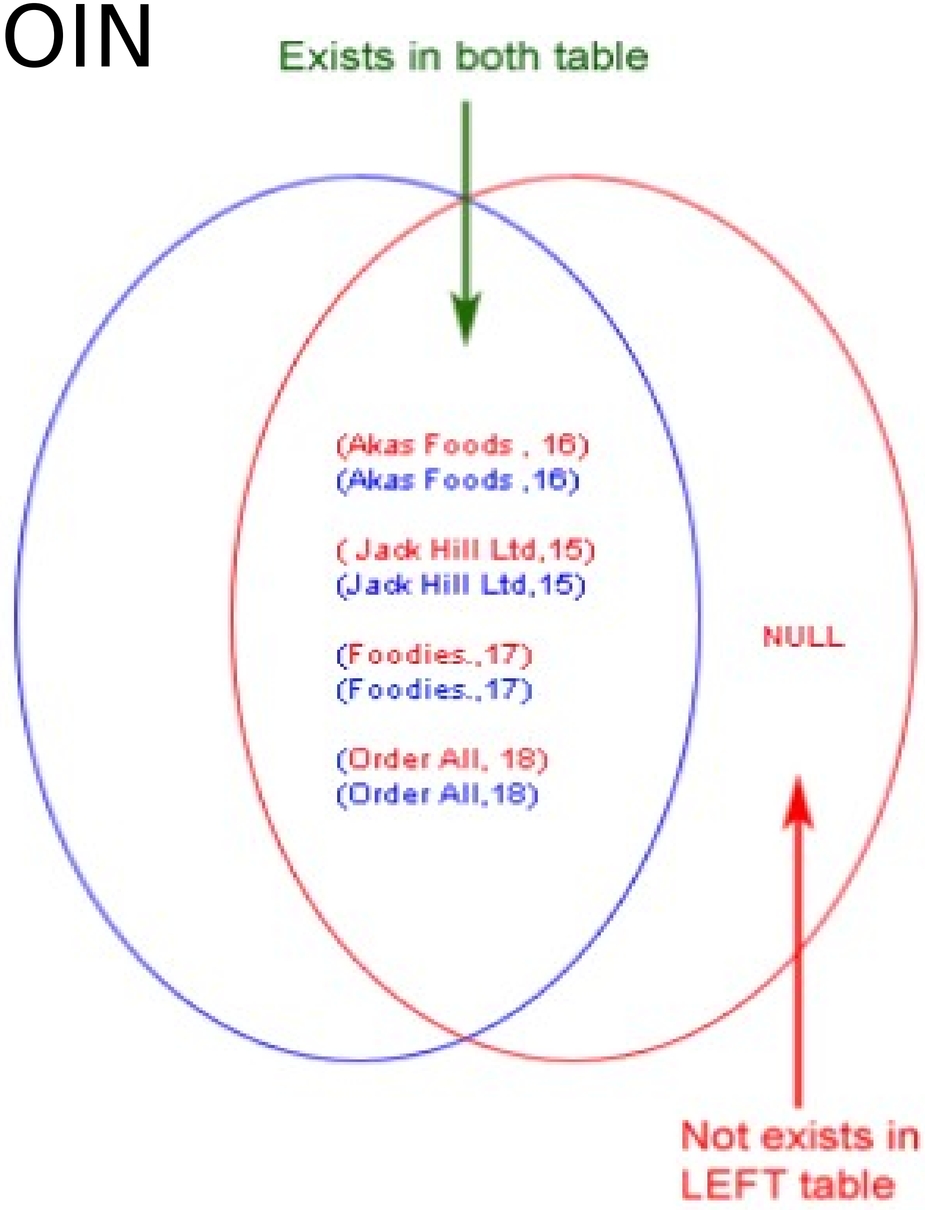
- SELECT c.company_id,c.company_name, c.company_city, f.company_id, f.item_name
  FROM company c
  RIGHT OUTER JOIN food f
  ON
  c



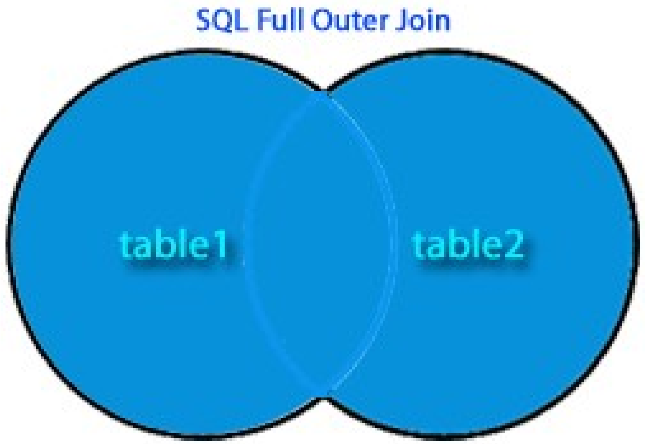
COMPANY_ID	COMPANY_NAME	COMPANY_CITY	COMPANY_ID	ITEM_NAME
16	Akas Foods	Delhi	16	Chex Mix
15	Jack Hill Ltd	London	15	Cheez-It
15	Jack Hill Ltd	London	15	BN Biscuit
17	Foodies.	London	17	Mighty Munch
15	Jack Hill Ltd	London	15	Pot Rice
18	Order All	Boston	18	Jaffa Cakes
-	-	-	-	Salt n Shake

7 rows returned in 0.19 seconds

RIGHT OUTER JOIN



Full Outer Join



- A match that includes all matches between two tables plus all non-matches whether on the left or right side of a join.

- SQL Example:

```
SELECT    a.column1, b.column2
FROM      table1 a FULL OUTER JOIN table2 b
ON        a.columnpk = b.columnfk;
```

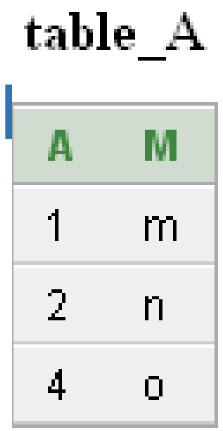
- Oracle syntax: The **UNION** operator to mimic the behavior.

Full Outer Join - Example

- SELECT * FROM

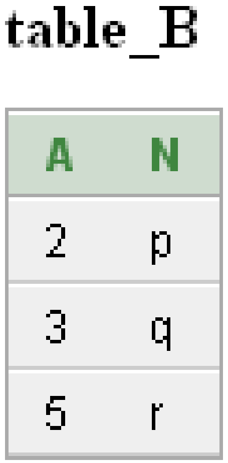
  table_A FULL OUTER JOIN table_B

  ON table_B.A;



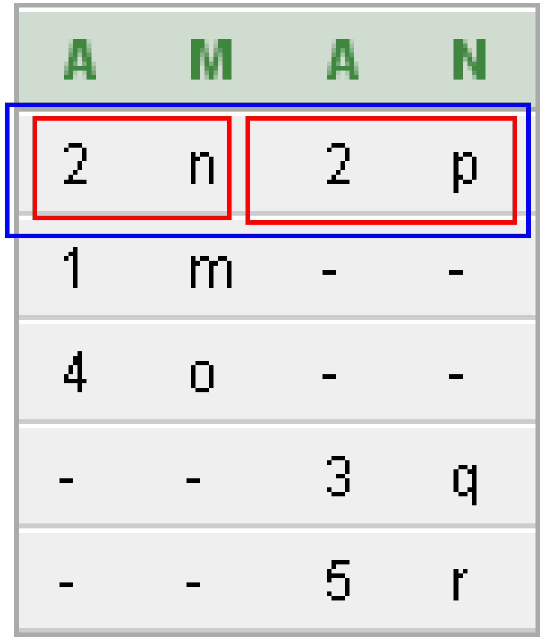
table_A

A	M
1	m
2	n
4	o



table_B

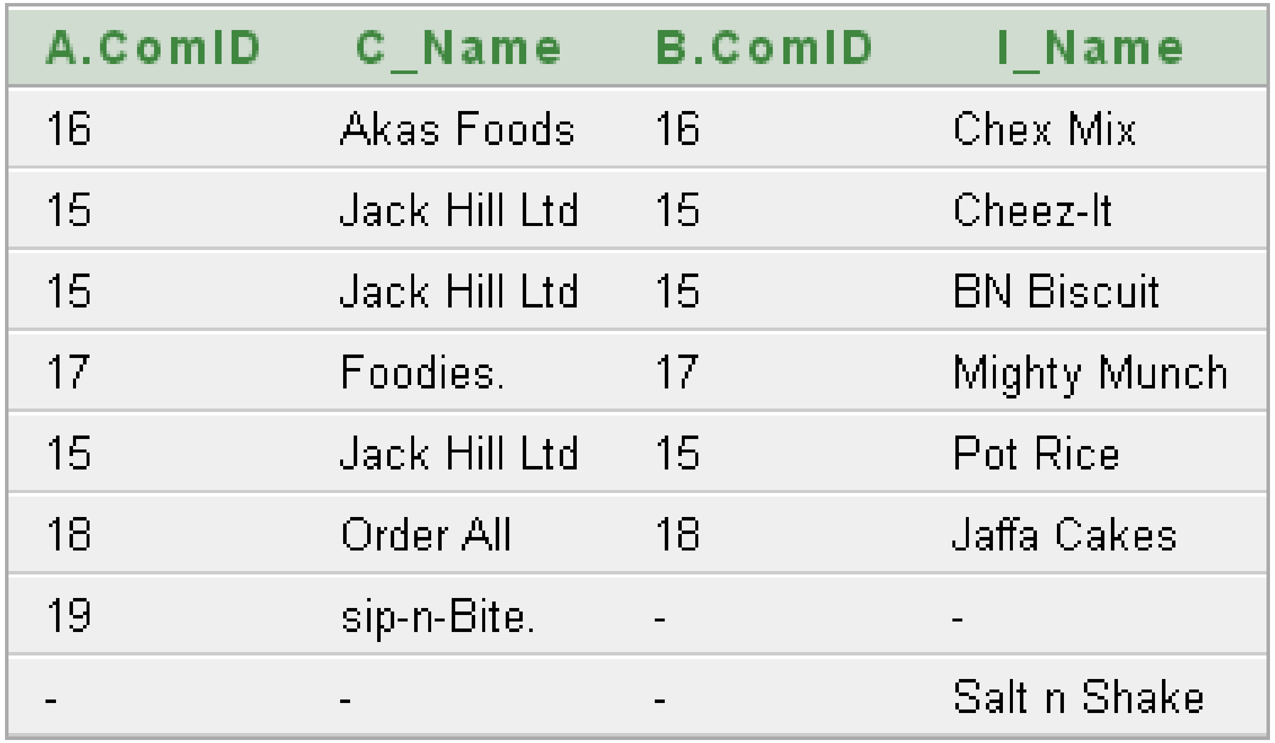
A	N
2	p
3	q
5	r



A	M	A	N
2	n	2	p
1	m	-	-
4	o	-	-
-	-	3	q
-	-	5	r

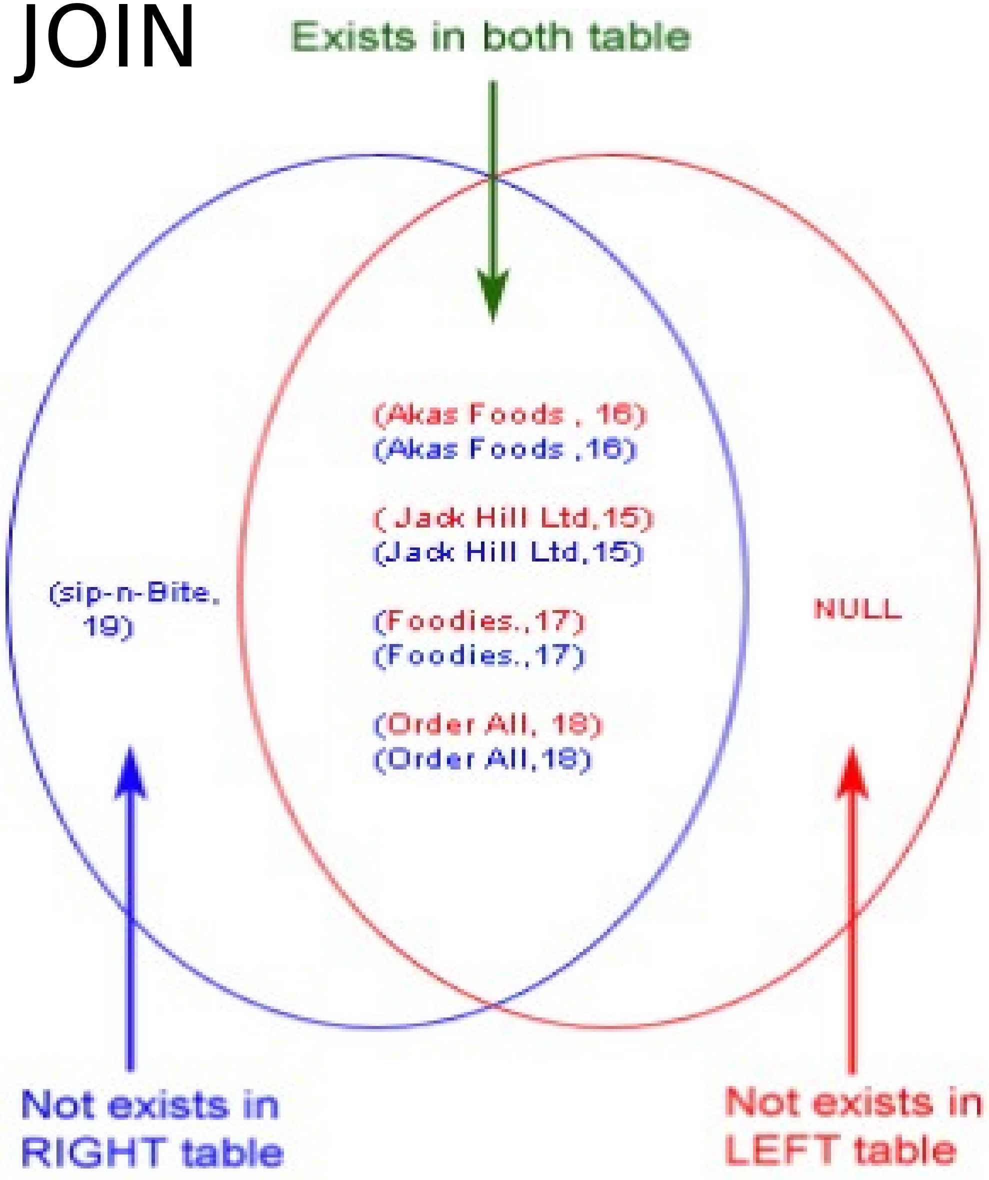
Full OUTER JOIN

```
SELECT
 a.company_id AS "a.ComID",  a.company_name AS "C_Name",  b.company_id AS "b.ComID",   b.item_name AS "I_Name"
FROM   company a  FULL OUTER JOIN foods b
ON a.company_id = b.company_id;
```



A.ComID	C_Name	B.ComID	I_Name
16	Akas Foods	16	Chex Mix
15	Jack Hill Ltd	15	Cheez-It
15	Jack Hill Ltd	15	BN Biscuit
17	Foodies.	17	Mighty Munch
15	Jack Hill Ltd	15	Pot Rice
18	Order All	18	Jaffa Cakes
19	sip-n-Bite.	-	-
-	-	-	Salt n Shake

Full OUTER JOIN



Full Outer Join

- The combination of LEFT OUTER JOIN and RIGHT OUTER JOIN and combined by, using UNION clause

```
SELECT   a.column1, b.column2
FROM     table1 a LEFT [OUTER] JOIN table2 b
ON    a.columnpk = b.columnfk
UNION
SELECT   a.column1, b.column2
FROM     table1 a RIGHT [OUTER] JOIN table2 b
ON    a.columnpk = b.columnfk;
```

Full Outer Join – oracle example

```
SELECT    a.column1, b.column2
FROM      table1 a, table2 b
WHERE     a.columnpk(+) = b.columnfk
UNION
SELECT    a.column1, b.column2
FROM      table1 a, table2 b
WHERE     a.columnpk = b.columnfk(+);
```

Outer join

- e.g. List the customer name, ID number, and order number for all customers listed in the CUSTOMER table. Include customer information even if there is no order available for that customer

  -
    ```
    SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME,
    ORDER_ID
              FROM CUSTOMER_T LEFT OUTER JOIN ORDER_T
              ON CUSTOMER_T.CUSTOMER_ID =
                                        ORDER_T.CUSTOMER_ID
    ```

  - The syntax LEFT OUTER JOIN was selected because the CUSTOMER_T table was named first, and it is the table from which we wish all rows returned (regardless of whether there is a matching order in the ORDER_T table)

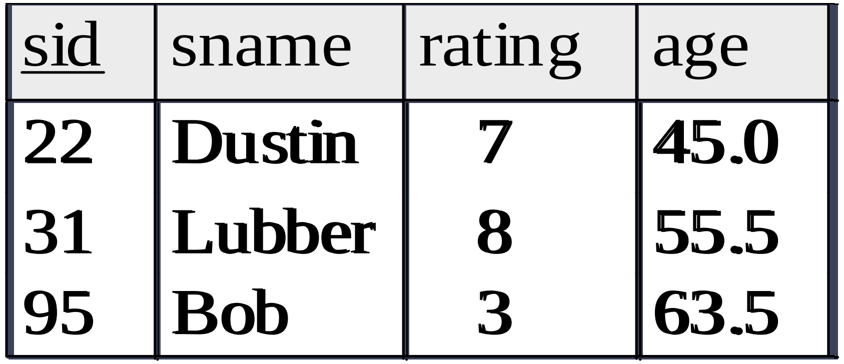
Outer join

- e.g. List the customer name, ID number, and order number for all orders listed in the ORDER table. Include order number even if there is no customer name and identification number available

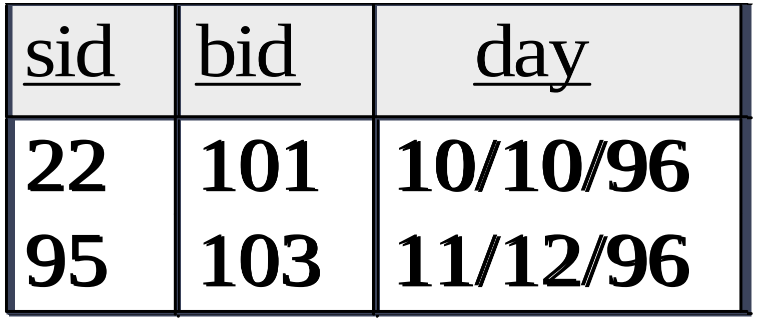
```
SELECT CUSTOMER_T.CUSTOMER_ID, CUSTOMER_NAME,
ORDER_ID
          FROM CUSTOMER_T RIGHT OUTER JOIN ORDER_T
          ON CUSTOMER_T.CUSTOMER_ID =
                                         ORDER_T.CUSTOMER_ID
```

LEFT OUTER JOIN

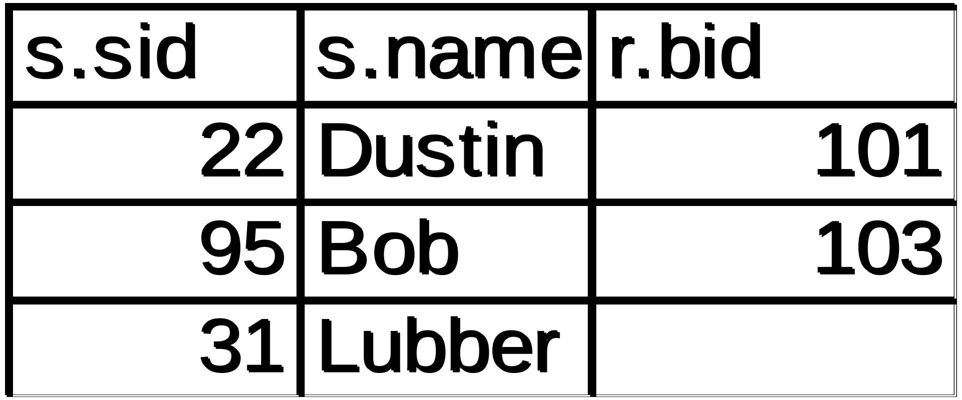
```
SELECT s.sid, s.name, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
```



sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5



sid	bid	day
22	101	10/10/96
95	103	11/12/96

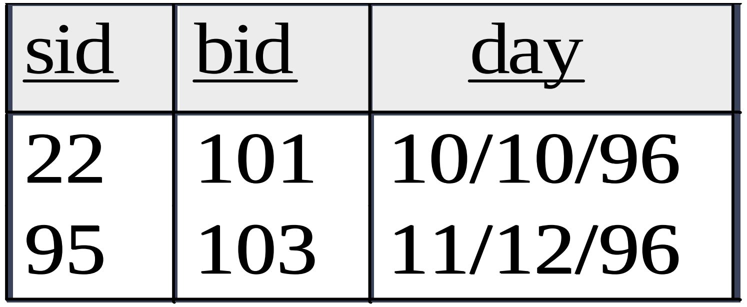


s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103
31	Lubber	

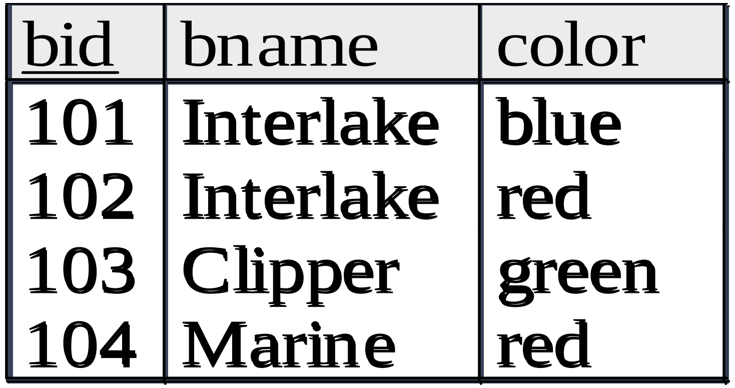
Returns all sailors & information on whether they have reserved boats

RIGHT OUTER JOIN

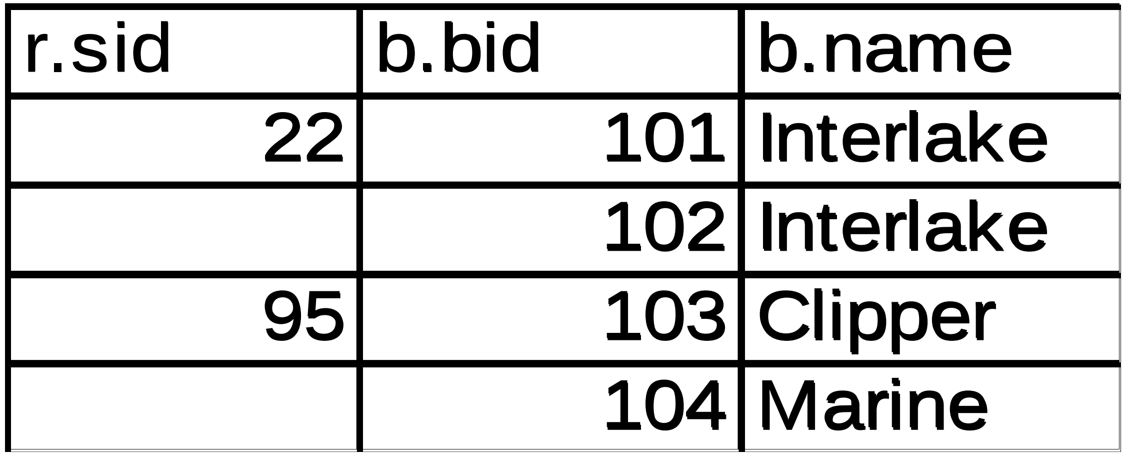
SELECT r.sid, b.bid, b.name
FROM Reserves r RIGHT OUTER JOIN Boats b
ON r.bid = b.bid



sid	bid	day
22	101	10/10/96
95	103	11/12/96



bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

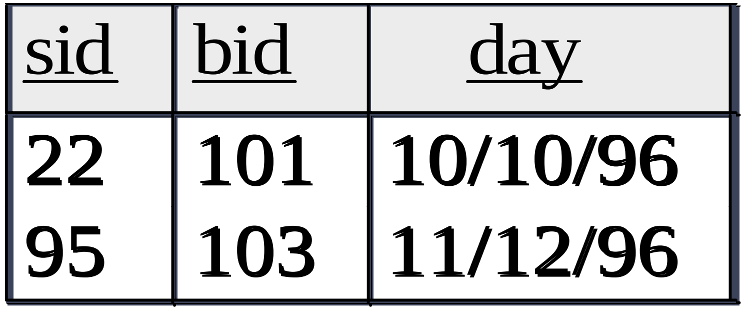


r.sid	b.bid	b.name
22	101	Interlake
	102	Interlake
95	103	Clipper
	104	Marine

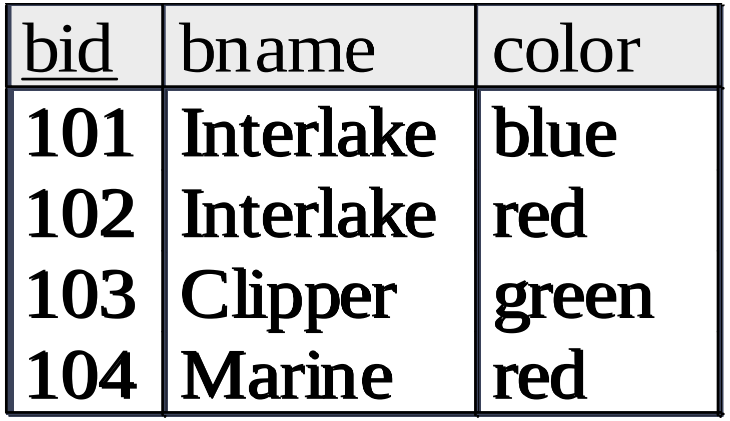
Returns all boats & information on which ones are reserved.

FULL OUTER JOIN

```
SELECT r.sid, b.bid, b.name
FROM Reserves r FULL OUTER JOIN Boats b
ON r.bid = b.bid
```



sid	bid	day
22	101	10/10/96
95	103	11/12/96



bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

r.sid	b.bid	b.name
22	101	Interlake
	102	Interlake
95	103	Clipper
	104	Marine

Returns all boats & all information on reservations

SQL Joins

Defining Join Types: Self Join

- A SELF JOIN is another type of join in sql which is used to join a table to itself,
  - specially when the table has a FOREIGN KEY which references its own PRIMARY KEY.
- A recursive join internally within a single table based on a primary and foreign key residing in each row of data in a table.
- You must use table name aliases to create a ***SELF JOIN***.
- Self joins typically use two separate column names.

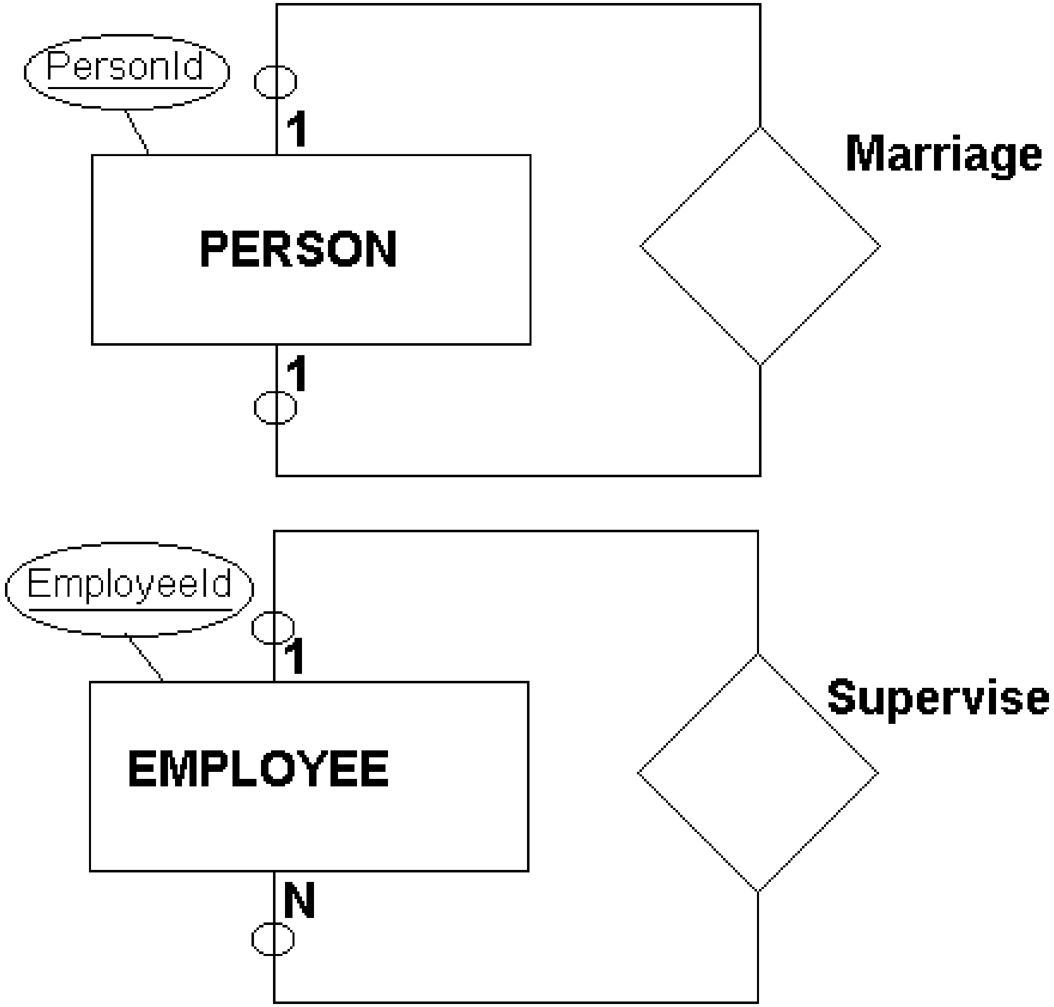
SQL Joins

Defining Join Types: Self Join

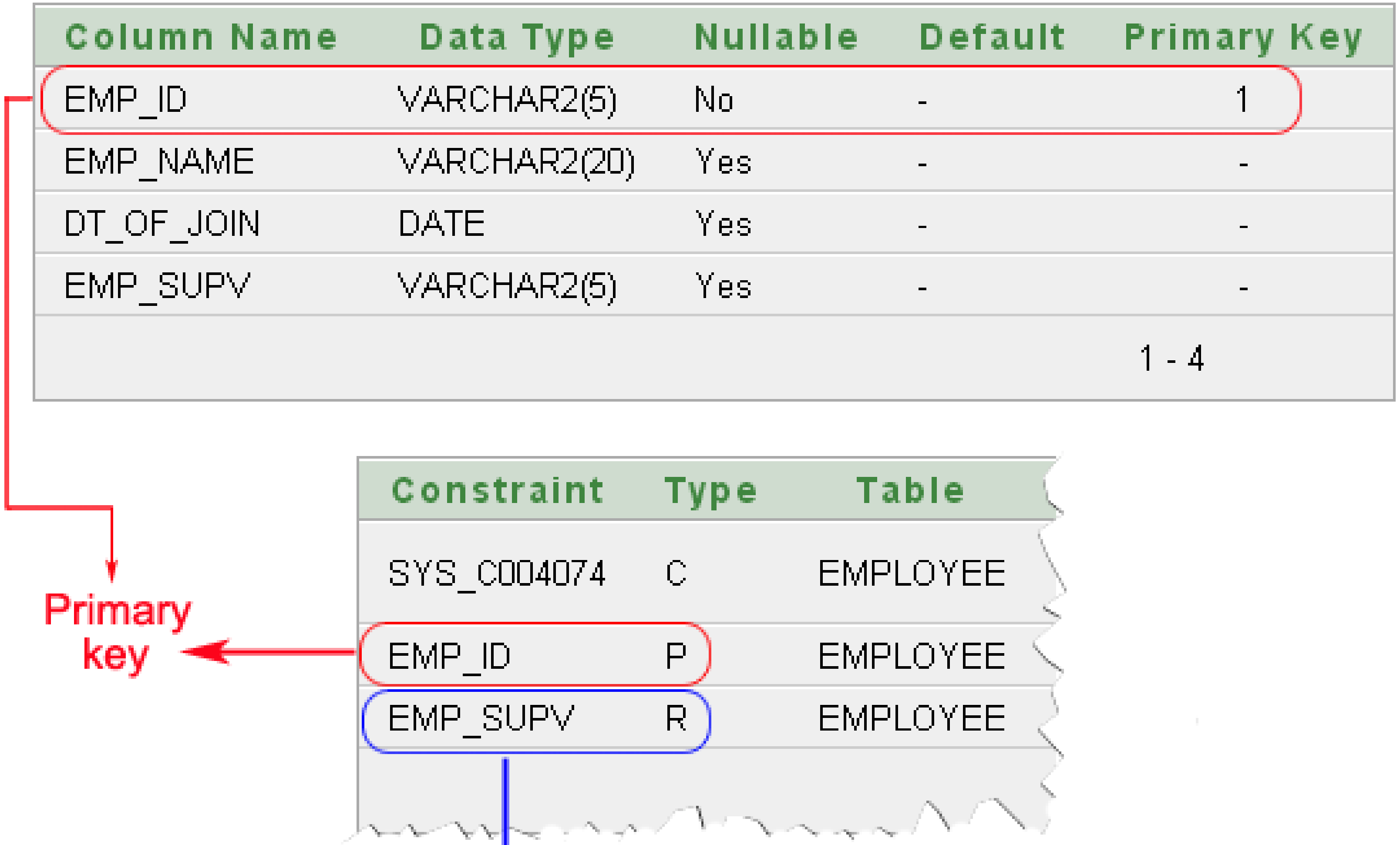
- Example:

```
SELECT   a.column1, b.column2
FROM     table1 a [INNER] JOIN table1 b
ON       a.columnpk = b.columnfk;
--------------------------------------------------------
------------
SELECT   a.column1, b.column2
FROM     table1 a, table1 b
WHERE    a.columnpk = b.columnfk;
```

Self Join - Unary Relationship In Database



The structure of the table



Column Name	Data Type	Nullable	Default	Primary Key
EMP_ID	VARCHAR2(5)	No	-	1
EMP_NAME	VARCHAR2(20)	Yes	-	-
DT_OF_JOIN	DATE	Yes	-	-
EMP_SUPV	VARCHAR2(5)	Yes	-	-
				1 - 4

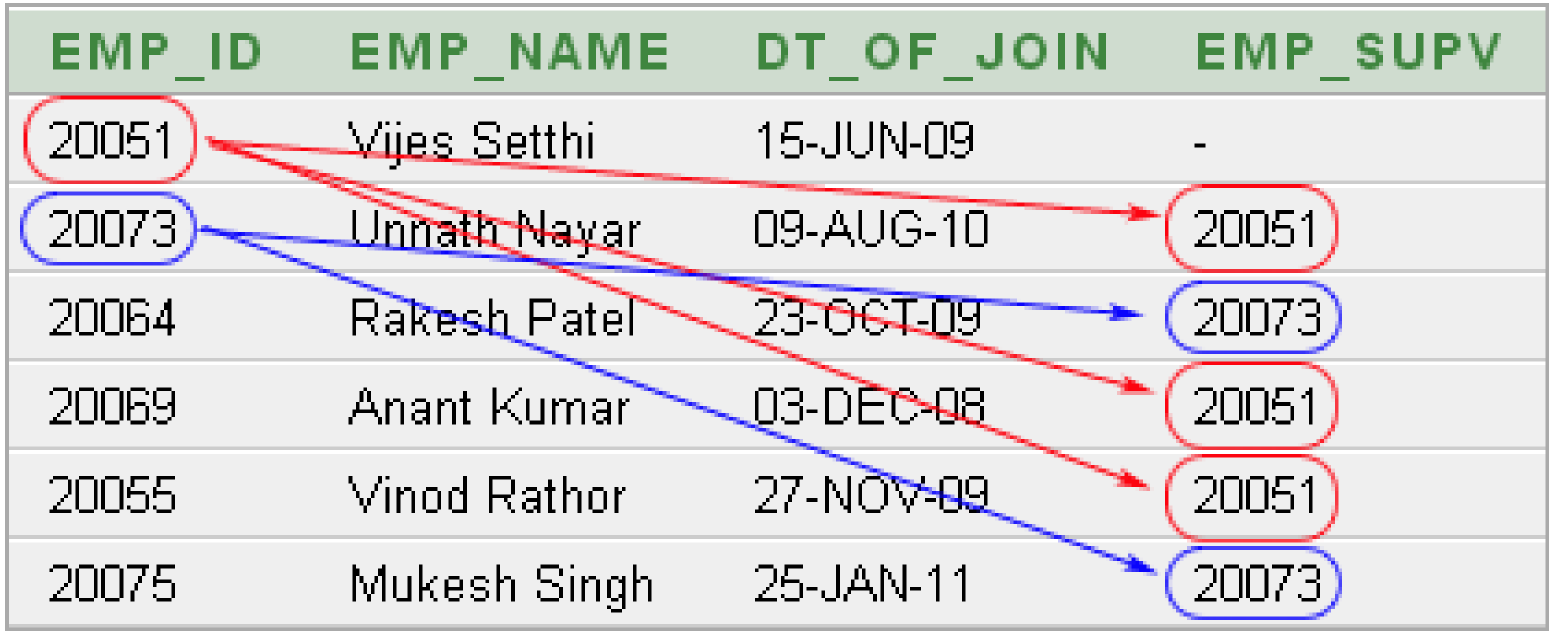
Constraint	Type	Table
SYS_C004074	C	EMPLOYEE
EMP_ID	P	EMPLOYEE
EMP_SUPV	R	EMPLOYEE

Primary key

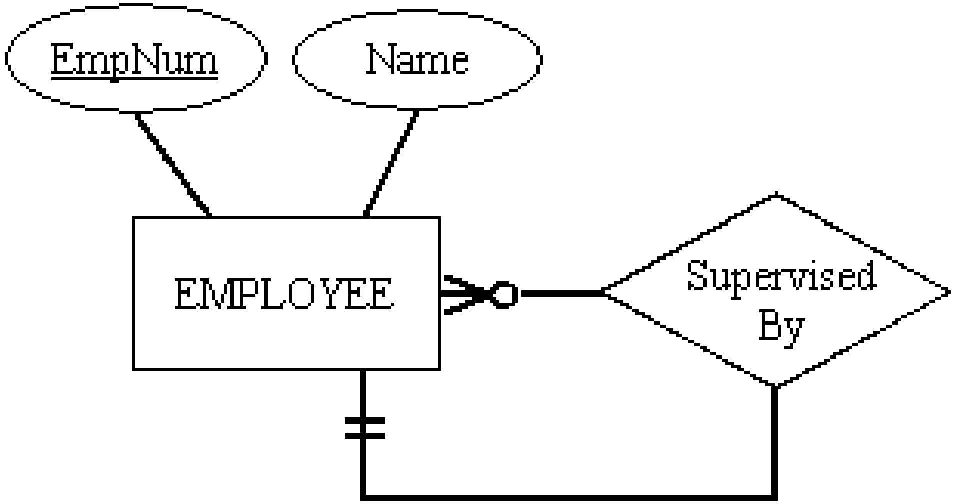
Foreign key

Referencing EMP_ID of this table

Unary relationship to employee

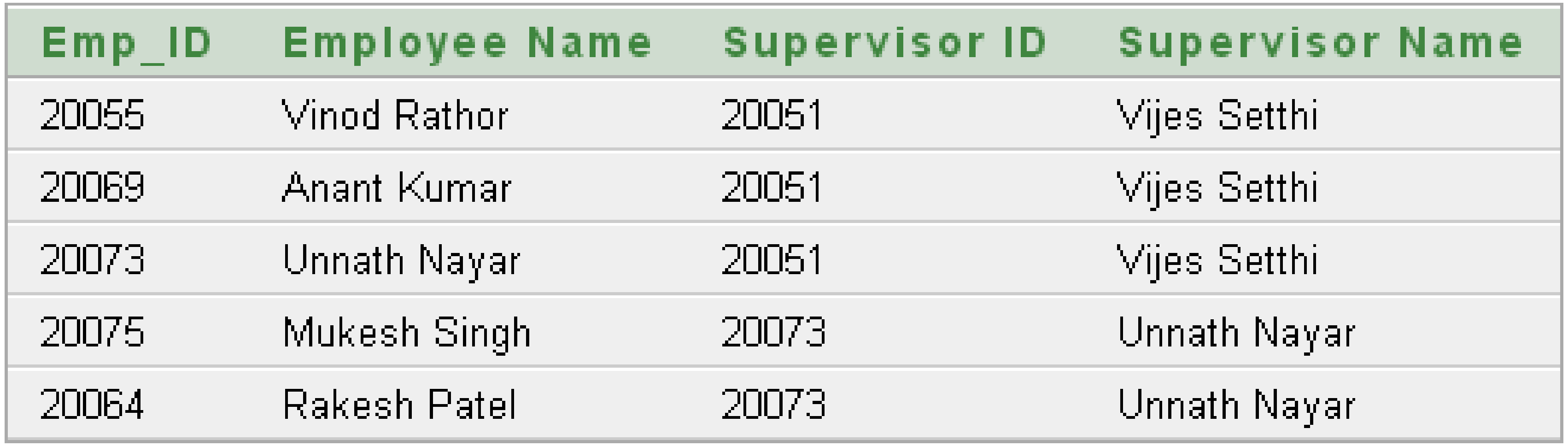


EMP_ID	EMP_NAME	DT_OF_JOIN	EMP_SUPV
20051	Vijes Setthi	15-JUN-09	-
20073	Unnath Nayar	09-AUG-10	20051
20064	Rakesh Patel	23-OCT-09	20073
20069	Anant Kumar	03-DEC-08	20051
20055	Vinod Rathor	27-NOV-09	20051
20075	Mukesh Singh	25-JAN-11	20073



Self Join - Example

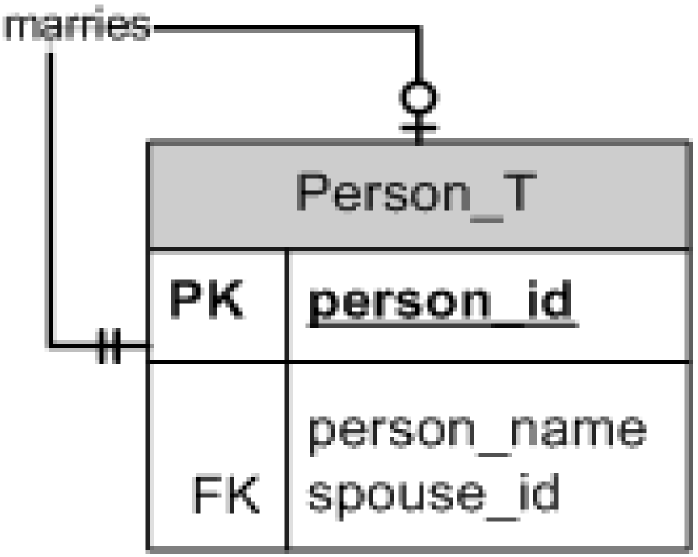
```
• SELECT      a.emp_id AS "Emp_ID",
                 a.emp_name AS "Employee Name",
                b.emp_id AS "Supervisor ID",
                  b.emp_name AS "Supervisor Name"
         FROM employee a, employee b
                         WHERE a.emp_id = b. emp_supv;
```



Emp_ID	Employee Name	Supervisor ID	Supervisor Name
20055	Vinod Rathor	20051	Vijes Setthi
20069	Anant Kumar	20051	Vijes Setthi
20073	Unnath Nayar	20051	Vijes Setthi
20075	Mukesh Singh	20073	Unnath Nayar
20064	Rakesh Patel	20073	Unnath Nayar

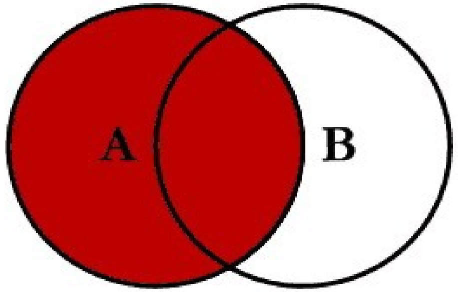
Self Join - Example

- Display the persons’ name along with their spouse name.

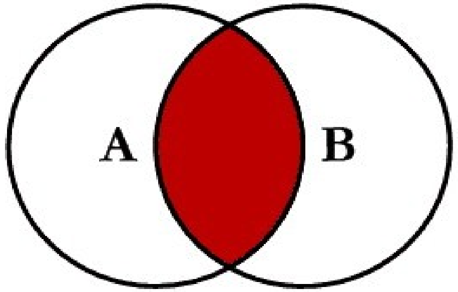


```
SELECT   p.person_name as “Person
Name”,
              s.person_name as “Spouse
  Name”
FROM      Person p, Person s
WHERE  p.person_id = s.spouse_id
```

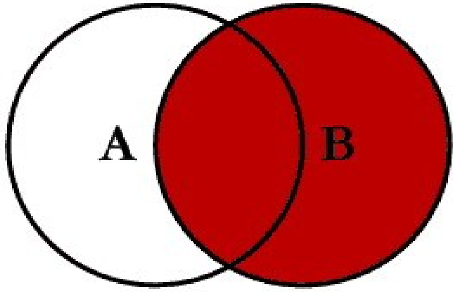
SQL JOINS



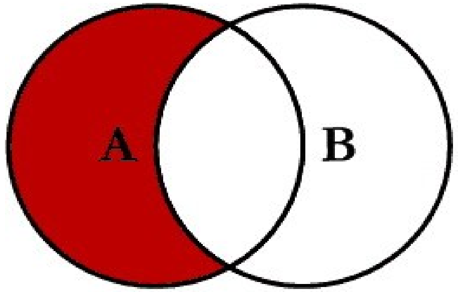
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



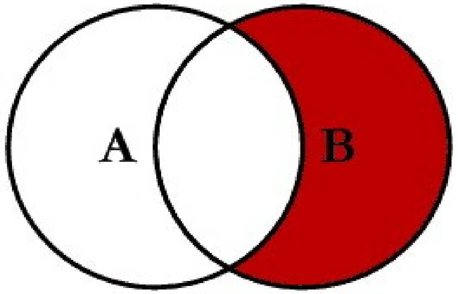
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



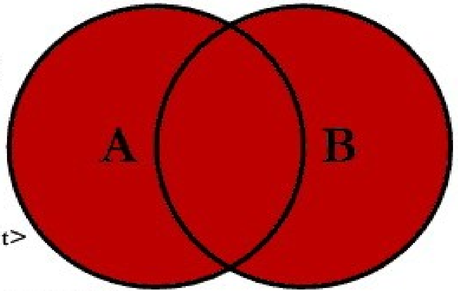
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



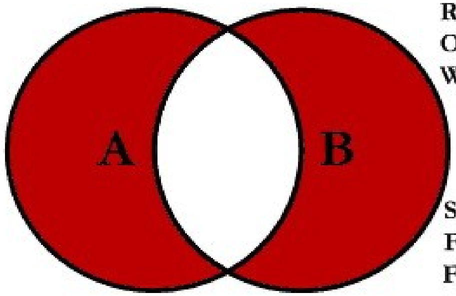
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Introduction

- Querying one table already done & practiced!
- Real power of relational database
  - Storage of data in multiple tables
  - Necessitates creating queries to use multiple tables
- Two Basic approaches for processing multiple tables
  - Sub-queries
  - Join

Processing Multiple Tables Using Sub-queries

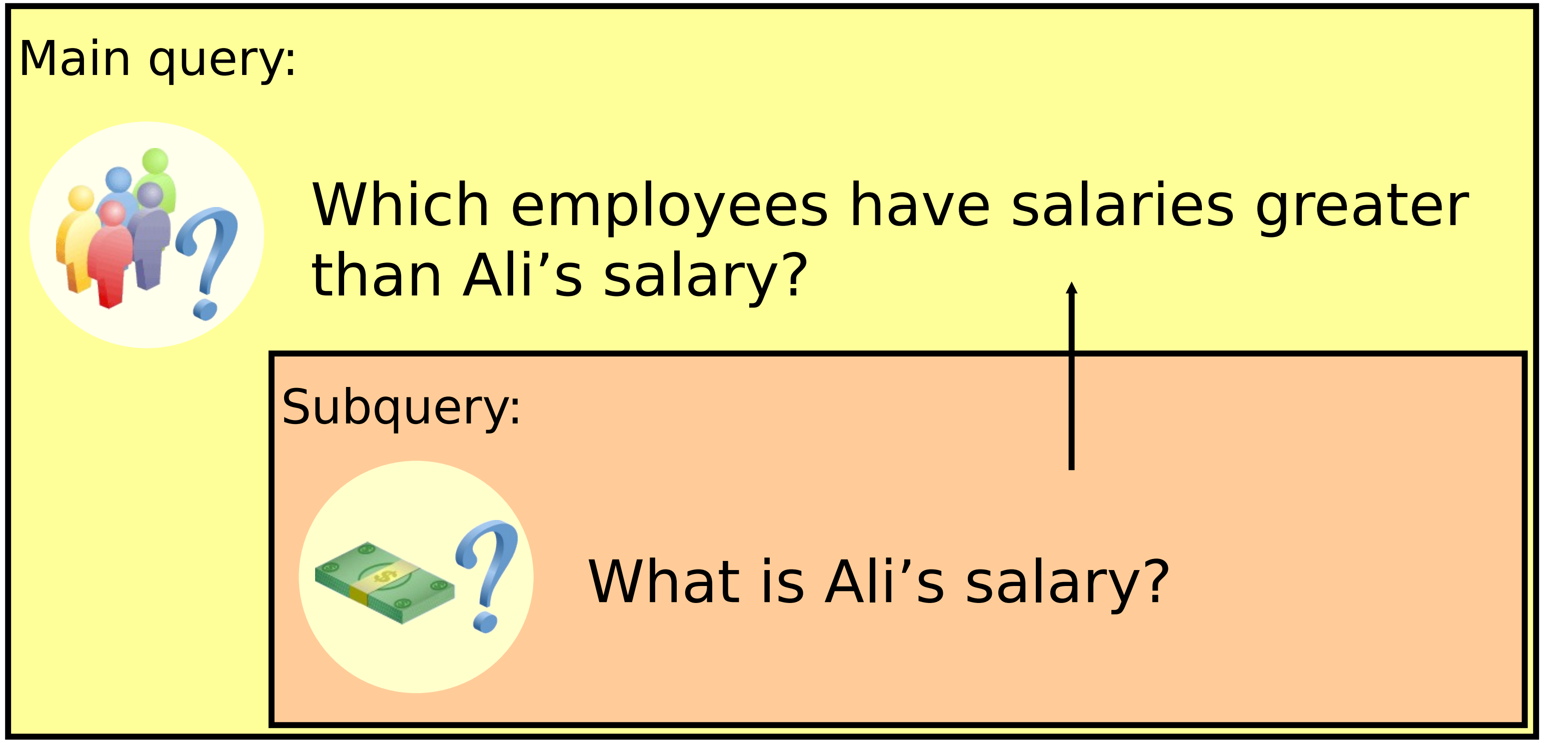
- A *subquery* is a query within a query.
- Subqueries enable you to write queries that select data rows for criteria that are actually developed while the query is executing at *run time*.
- Subquery – placing an inner query (SELECT statement) inside an outer query
  - Inner query provides a set of one or more values for outer query

Processing Multiple Tables Using Sub-queries

- One of the two basic approaches to process multiple tables
  - Different people will have different preferences about which technique to use
  - Joining is useful when data from several tables are to be retrieved and displayed
  - Subquery when data from tables in outer query are to be displayed only

Using a Subquery to Solve a Problem

- Who has a salary greater than Ali’s?



Main query:

Which employees have salaries greater than Ali’s salary?

Subquery:

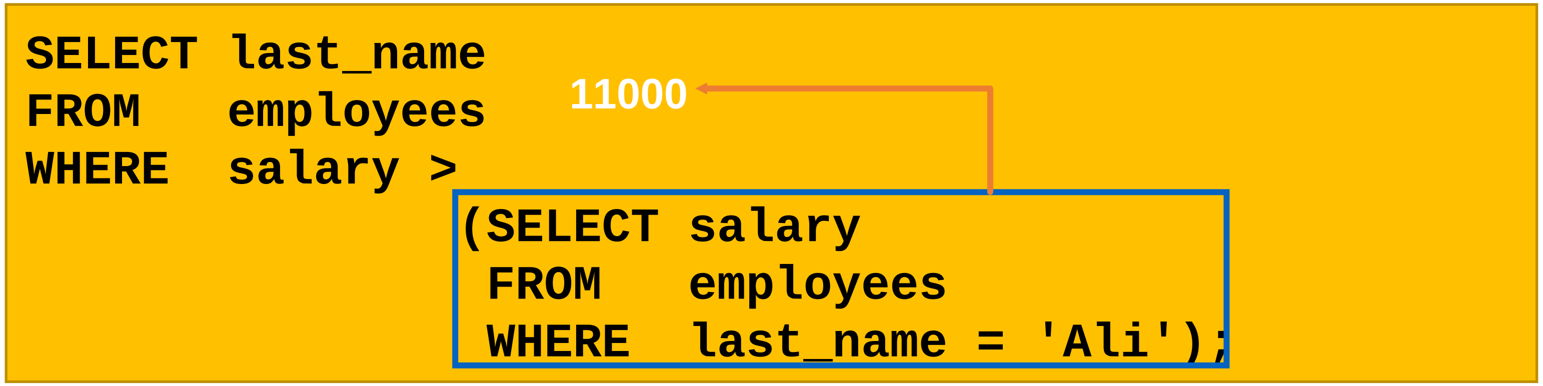
What is Ali’s salary?

Subquery Syntax

```
SELECT   select_list
FROM     table
WHERE    expr operator
                    (SELECT       select_list
                     FROM         table);
```

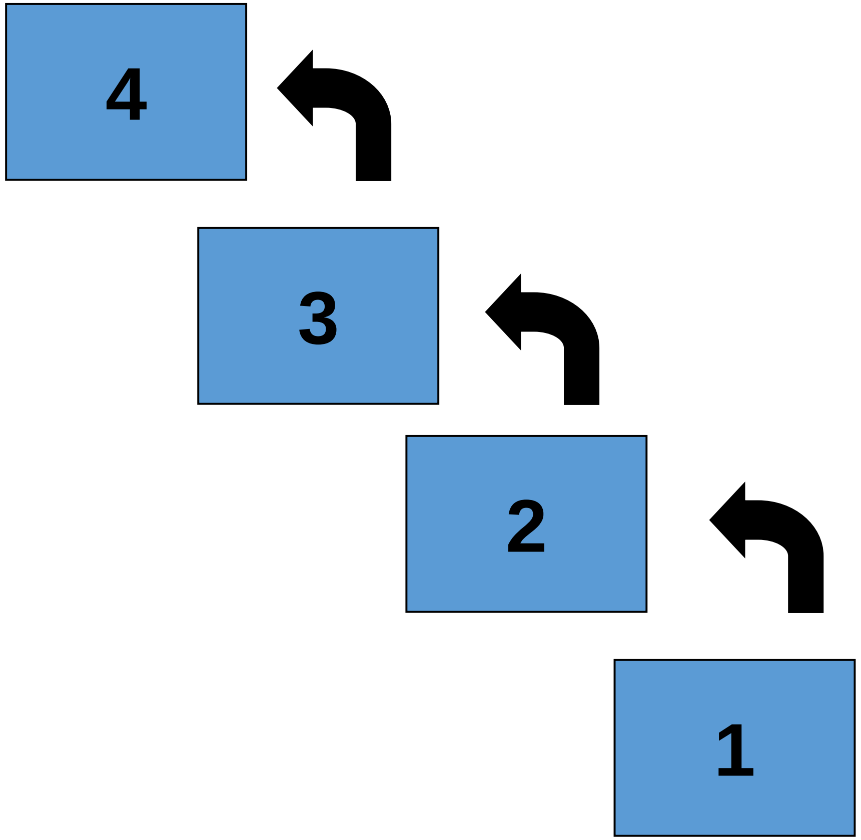
- The subquery (inner query) executes once before the main query (outer query).
- The result of the subquery is used by the main query.

Using a Sub-query



```
SELECT last_name
FROM   employees
WHERE  salary >
               (SELECT salary
                FROM   employees
                WHERE  last_name = 'Ali');
```

The basic concept is to pass a single value or many values from the subquery to the next query and so on.



When reading or writing SQL subqueries, you should start from the bottom upwards, working out which data is to be passed to the next query up.

Subquery Types

- There are three basic types of subqueries.

1. Subqueries that operate on lists by use of the IN operator or with a comparison operator.
    - These subqueries can return a group of values, but the values must be from a single column of a table.

SUBQUERY TYPES

2. Subqueries that use an unmodified comparison operator (=, <, >, <>)
   - these subqueries must return only a single, *scalar* value.
3. Subqueries that use the EXISTS operator to test the *existence* of data rows satisfying specified criteria.

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The `ORDER BY` clause in the subquery is not needed.
  - Subqueries cannot manipulate their results internally.
- Use single-row operators with single-row subqueries, and use multiple-row operators with
  multiple-row subqueries.

Sub-Queries Example

```
• SELECT CUSTOMER_NAME  FROM CUSTOMER_T, ORDER_T
   WHERE CUSTOMER_T.CUSTOMER_ID = ORDER_T.CUSTOMER_ID
                 AND      ORDER_ID = 1008;
```

```
○ SELECT CUSTOMER_NAME  FROM CUSTOMER_T
     WHERE  CUSTOMER_ID =
          (SELECT CUSTOMER_ID  FROM ORDER_T
                 WHERE ORDER_ID = 1008);
```

SUBQUERIES AND THE IN Operator

- Subqueries that are introduced with the keyword **IN** take the general form:
  - WHERE expression [NOT] IN (subquery)
- The only difference in the use of the IN operator with subqueries is that the list does not consist of *hard-coded* values.

SUBQUERIES AND COMPARISON OPERATORS

- The general form of the WHERE clause with a comparison operator is similar to that used thus far in the text.
- Note that the subquery is again enclosed by parentheses.

```
WHERE <expression> <comparison_operator>
   (subquery)
```

SUBQUERIES AND COMPARISON OPERATORS

- The most important point to remember when using a subquery with a comparison operator is that the subquery can only return a single or *scalar* value.
- This is also termed a *scalar subquery* because a single column of a single row is returned by the subquery.

To identify the students who have failed in course CSC273

```
Select student_id
From marks
Where course_id = ‘CSC273’
And grade < 40;
```

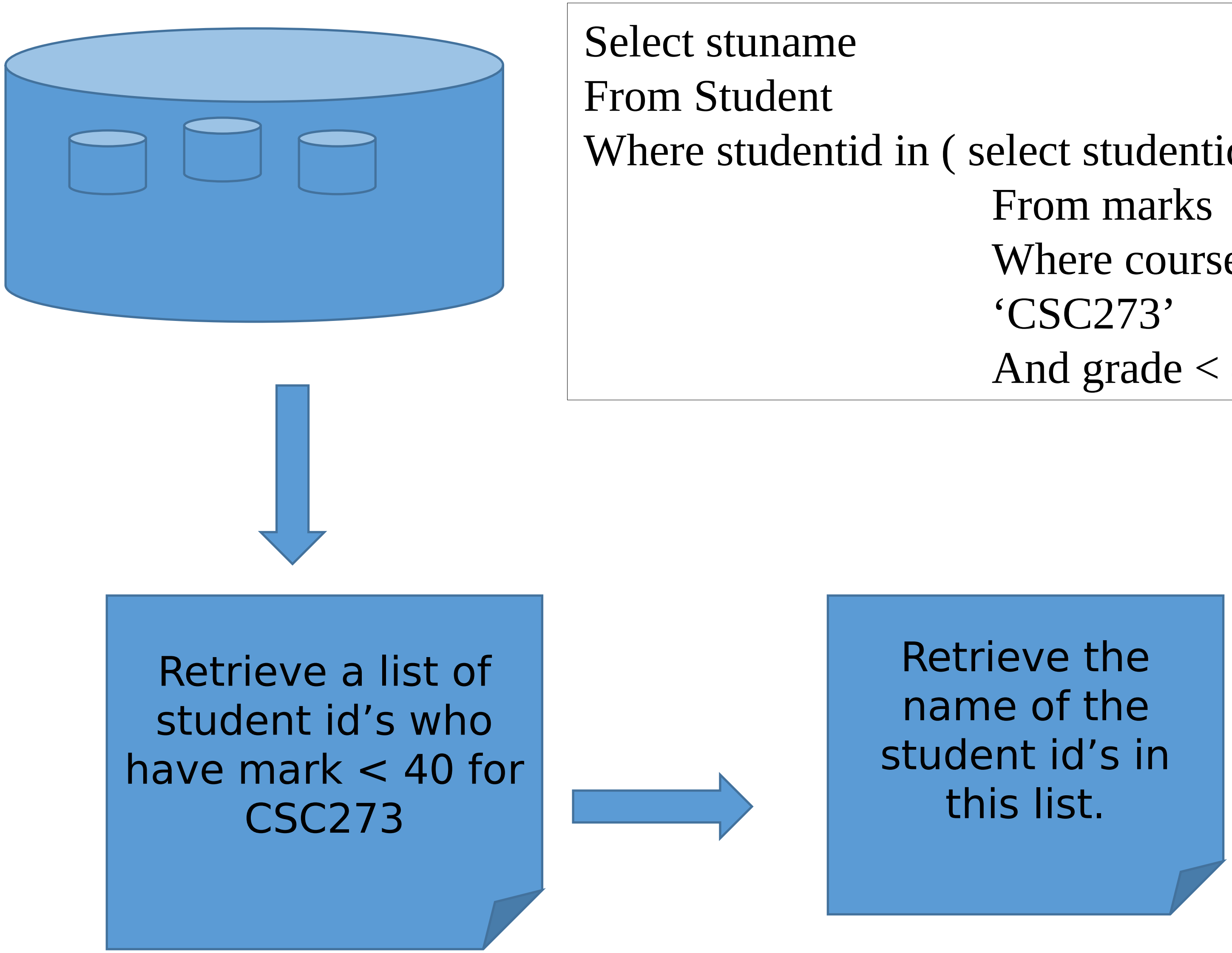
If we want to retrieve a name based on a student id

```
Select stu_name
From student
Where student_id = 9292145;
```

```
Select stu_name
From Student
Where student_id in ( select student_id
                      From marks
                      Where course_id = ‘CSC273’
                      And grade < 40);
```

Why use IN?

```
Select stuname
From Student
Where studentid in ( select studentid
                     From marks
                     Where courseid =
                     ‘CSC273’
                     And grade < 40);
```



Retrieve a list of student id’s who have mark < 40 for CSC273

Retrieve the name of the student id’s in this list.

Subquery Example

- Show all customers who have placed an order

Many programmers simply use IN even if equal sign (=) would also work

The IN operator will test to see if the CUSTOMER_ID value of a row is included in the list returned from the subquery

```
SELECT CUSTOMER_NAME    FROM CUSTOMER_T
WHERE CUSTOMER_ID IN
            (SELECT DISTINCT CUSTOMER_ID FROM
  ORDER_T);
```

Subquery is embedded in parentheses. In this case it returns a list that will be used in the WHERE clause of the outer query

SUBQUERIES AND COMPARISON OPERATORS

- If we substitute this query as a subquery in another SELECT statement, then that SELECT statement will fail.
- This is demonstrated in the next SELECT statement. Here the SQL code will fail because the subquery uses the greater than (>) comparison operator and the subquery returns multiple values.

```
SELECT emp_ssn
FROM employee
  WHERE emp_salary >
    (SELECT emp_salary
     FROM employee
       WHERE emp_salary > 40000);
```

Aggregate Functions and Comparison Operators

- The aggregate functions (AVG, SUM, MAX, MIN, and COUNT) always return a *scalar* result table.
- Thus, a subquery with an aggregate function as the object of a comparison operator will always execute provided you have formulated the query properly.

Aggregate Functions and Comparison Operators

```
SELECT emp_last_name "Last Name",
    emp_first_name "First Name",
    emp_salary "Salary"
FROM employee
WHERE emp_salary >
       (SELECT AVG(emp_salary)
        FROM employee);
```

```
Last Name    First Name   Salary
--------------- --------------- ----------
Bordoloi       Bijoy           $55,000
Joyner         Suzanne          $43,000
Zhu            Waiman          $43,000
Joshi          Dinesh          $38,000
```

Exercise

1. *Write a query that will list the names of who is older than the average student.*

TIP the sub-query needs to select the average age of students this should be used then as a filter.

```
SELECT stu_name
        FROM student
                WHERE age >
                        (SELECT avg(age) FROM student);
```

This will return 25 students of the 74 who are enrolled as being older than the average age.

Comparison Operators Modified with the ALL or ANY Keywords

- The ALL and ANY keywords can modify a comparison operator to allow an outer query to accept multiple values from a subquery.
- The general form of the WHERE clause for this type of query is shown here.

  WHERE <expression>
  <comparison_operator> [ALL | ANY]
  (subquery)

- Subqueries that use these keywords may also include GROUP BY and HAVING clauses.

The ALL Keyword

- The ALL keyword modifies the greater than comparison operator to mean greater than all values.

```
SELECT emp_ssn
FROM employee
  WHERE emp_salary >
   (SELECT emp_salary
    FROM employee
      WHERE emp_salary >
  40000);
```

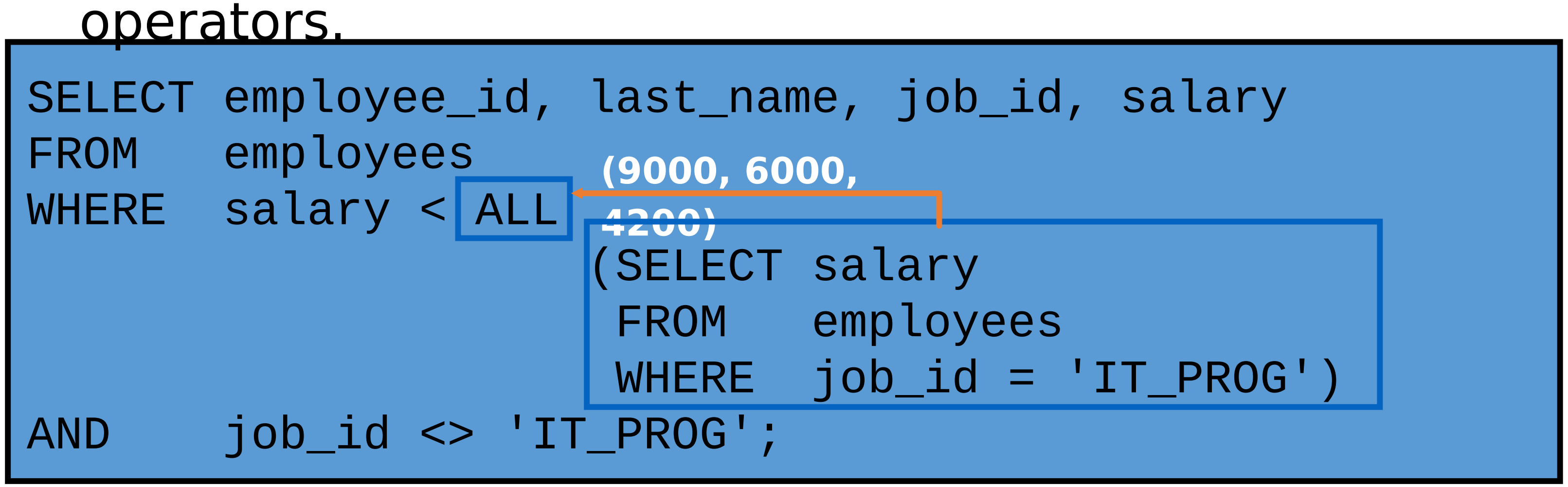
```
SELECT emp_ssn
FROM employee
WHERE emp_salary > ALL
    (SELECT emp_salary
       FROM employee
   WHERE emp_salary >
  40000);
```

Using the **ALL** Operator in Multiple-Row Subqueries

The slide example displays employees whose salary is less than the salary of all employees with a job ID of IT_PROG and whose job is not IT_PROG.

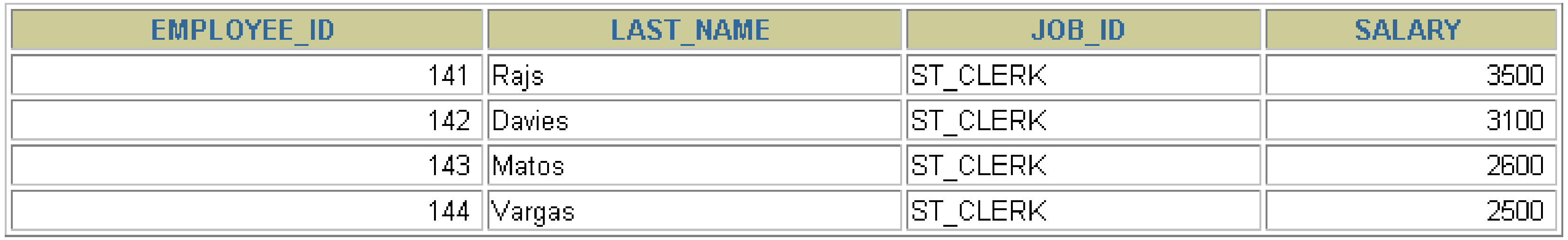
>ALL means more than the maximum, and <ALL means less than the minimum.

The NOT operator can be used with IN, ANY, and ALL operators.



```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                   (SELECT salary
                    FROM   employees
                    WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

(9000, 6000, 4200)



EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

Using the ANY Operator in Multiple-Row Subqueries

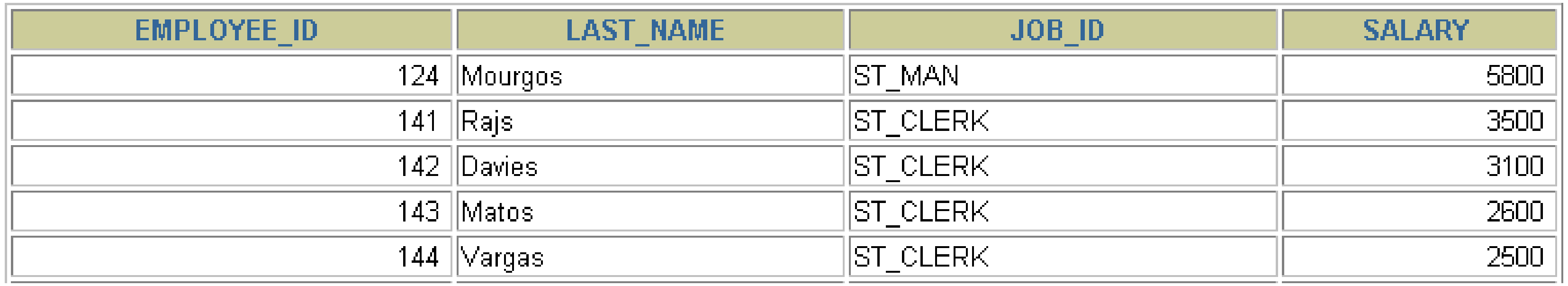
The slide example displays employees who are not IT programmers and whose salary is less than that of any IT programmer.

The maximum salary that a programmer earns is $9,000.

<ANY means less than the maximum. >ANY means more than the minimum.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                    (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

(9000, 6000, 4200)



EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

An "= ANY" (Equal Any) Example

- The "= ANY" operator is exactly equivalent to the IN operator.
- For example, to find the names of employees that have male dependents, you can use either IN or "= ANY" – both of the queries shown below will produce an identical result table.

```
SELECT emp_last_name "Last Name", emp_first_name "First Name"
FROM employee
WHERE emp_ssn IN
    (SELECT dep_emp_ssn
     FROM dependent
     WHERE dep_gender = 'M');

SELECT emp_last_name "Last Name", emp_first_name "First Name"
FROM employee
WHERE emp_ssn = ANY
    (SELECT dep_emp_ssn
     FROM dependent
     WHERE dep_gender = 'M');
```

A "!= ANY" (Not Equal Any) Example

- The "= ANY" is identical to the IN operator.
- However, the "!= ANY" (not equal any) is **not** equivalent to the NOT IN operator.
- If a subquery of employee salaries produces an intermediate result table with the salaries
  - $38,000, $43,000, and $55,000,
- then the WHERE clause shown here means
  - "NOT $38,000" AND "NOT $43,000" AND "NOT $55,000".

```
WHERE NOT IN (38000, 43000, 55000);
```

- However, the "!= ANY" comparison operator and keyword combination shown in this next WHERE clause means
  - "NOT $38,000" OR "NOT $43,000" OR "NOT $55,000".

MULTIPLE LEVELS OF NESTING

- Subqueries may themselves contain subqueries.
- When the WHERE clause of a subquery has as its object another subquery, these are termed *nested subqueries*.
- Consider the problem of producing a listing of employees that worked more than 10 hours on the project named *Order Entry*.

  - employee,
  - assignment,
  - project

emp_ssn	last_name	first_name

emp_ssn	pro_no	work_hours

pro_no	pro_name	

Example

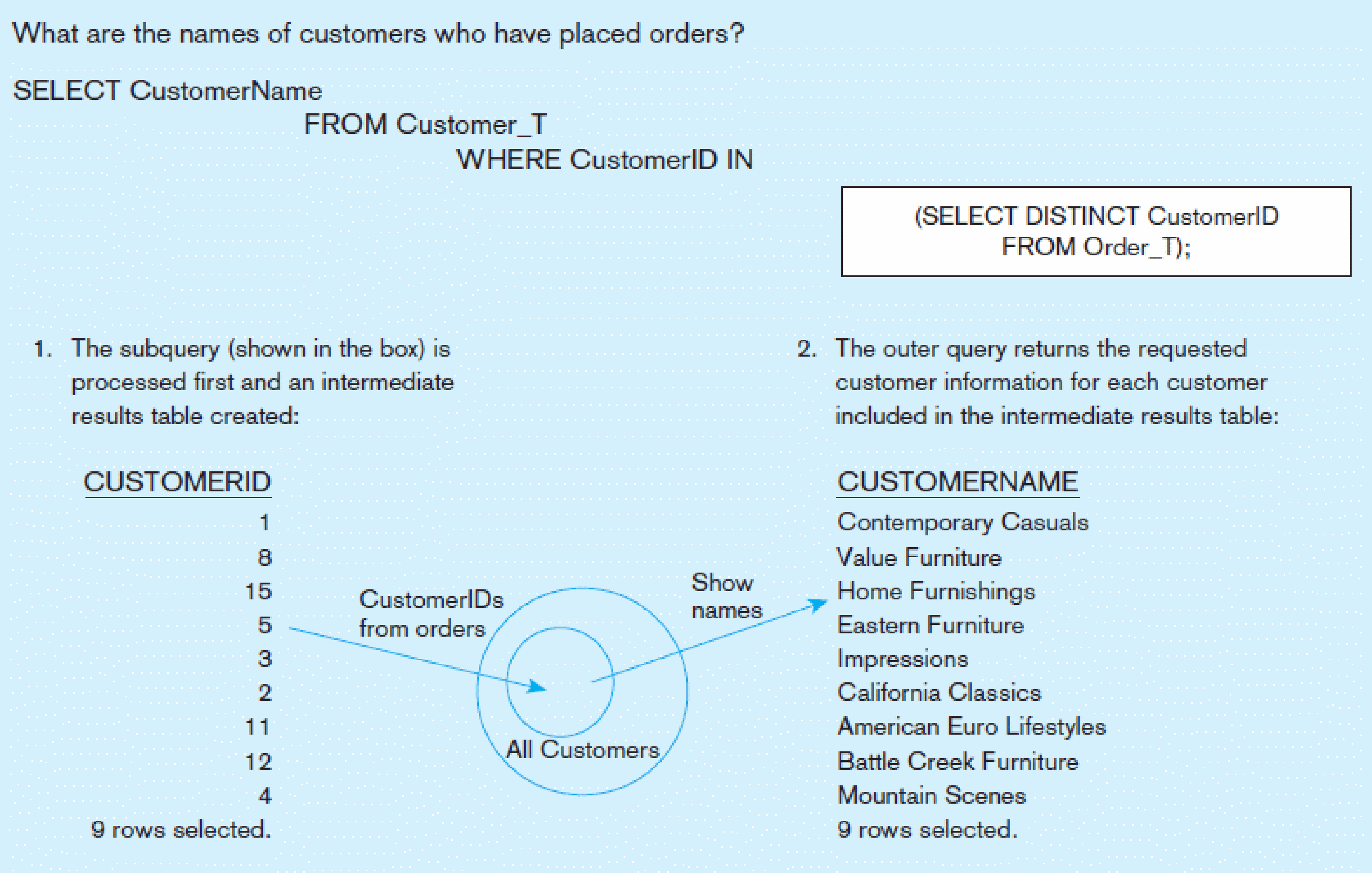
```
SELECT emp_last_name "Last Name",  emp_first_name "First Name"
  FROM employee  WHERE emp_ssn IN
          (SELECT work_emp_ssn
                 FROM assignment
           WHERE work_hours > 10 AND work_pro_number IN
           (SELECT pro_number
                FROM project
           WHERE pro_name = 'Order Entry') );
```

```
Last Name    First Name
--------------- ---------------
Bock            Douglas
Prescott        Sherri
```

Correlated vs. Non-correlated Subqueries

- Subqueries can be:
  - Noncorrelated–executed once for the entire outer query
  - Correlated–executed once for each row returned by the outer query
- **Non-correlated** subqueries:
  - Do not depend on data from the outer query
  - Execute once for the entire outer query
- **Correlated** subqueries:
  - Make use of data from the outer query
  - Execute once for each row of the outer query
  - Usually use the EXISTS operator

Processing a noncorrelated subquery



What are the names of customers who have placed orders?

```
SELECT CustomerName
            FROM Customer_T
                  WHERE CustomerID IN
                                          (SELECT DISTINCT CustomerID
                                           FROM Order_T);
```

1. The subquery (shown in the box) is processed first and an intermediate results table created:

```
CUSTOMERID
         1
         8
        15
         5
         3
         2
        11
        12
         4
9 rows selected.
```

2. The outer query returns the requested customer information for each customer included in the intermediate results table:

```
CUSTOMERNAME
Contemporary Casuals
Value Furniture
Home Furnishings
Eastern Furniture
Impressions
California Classics
American Euro Lifestyles
Battle Creek Furniture
Mountain Scenes
9 rows selected.
```

A noncorrelated subquery processes completely before the outer query begins

Correlated Subquery Example

- Show all orders that include furniture finished in natural ash

The EXISTS operator will return a TRUE value if the subquery resulted in a non-empty set, otherwise it returns a FALSE

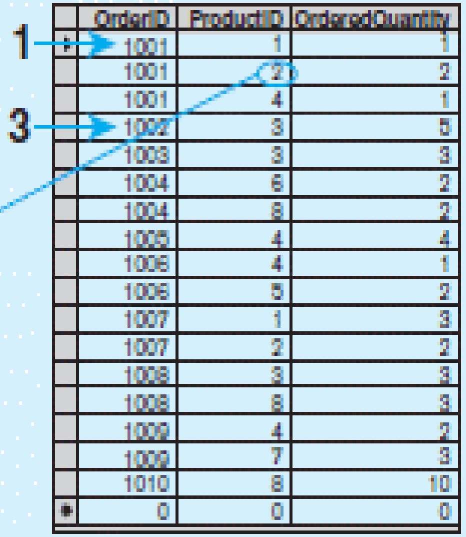
```
SELECT DISTINCT ORDER_ID FROM ORDER_LINE_T
WHERE  EXISTS

   (SELECT * FROM PRODUCT_T
   WHERE PRODUCT_ID = ORDER_LINE_T.PRODUCT_ID

   AND PRODUCT_FINISH = ‘Natural ash’);
```

The subquery is testing for a value that comes from the outer query

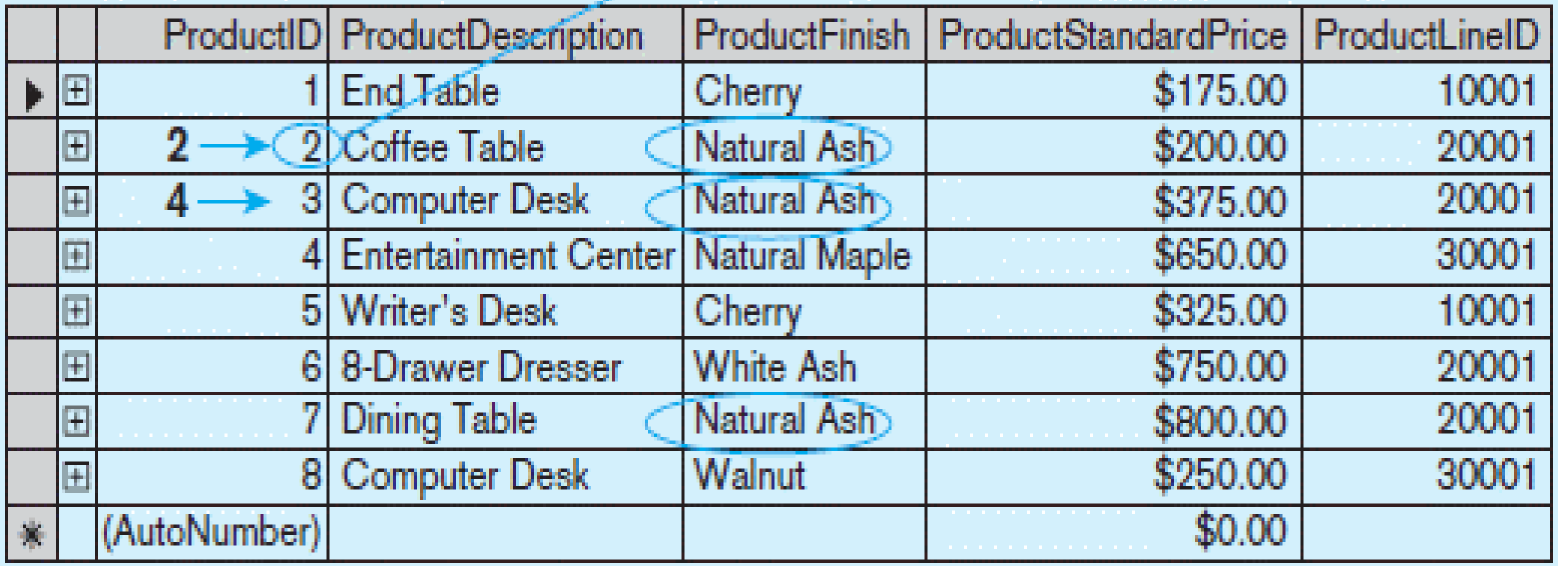
What are the order IDs for all orders that have included furniture finished in natural ash?

```
SELECT DISTINCT OrderID FROM OrderLine_T
WHERE EXISTS
          (SELECT *
            FROM Product _T
                      WHERE ProductID = OrderLine_T.ProductID
                            AND Productfinish = 'Natural Ash');
```

Subquery refers to outer-query data, so executes once for each row of outer query

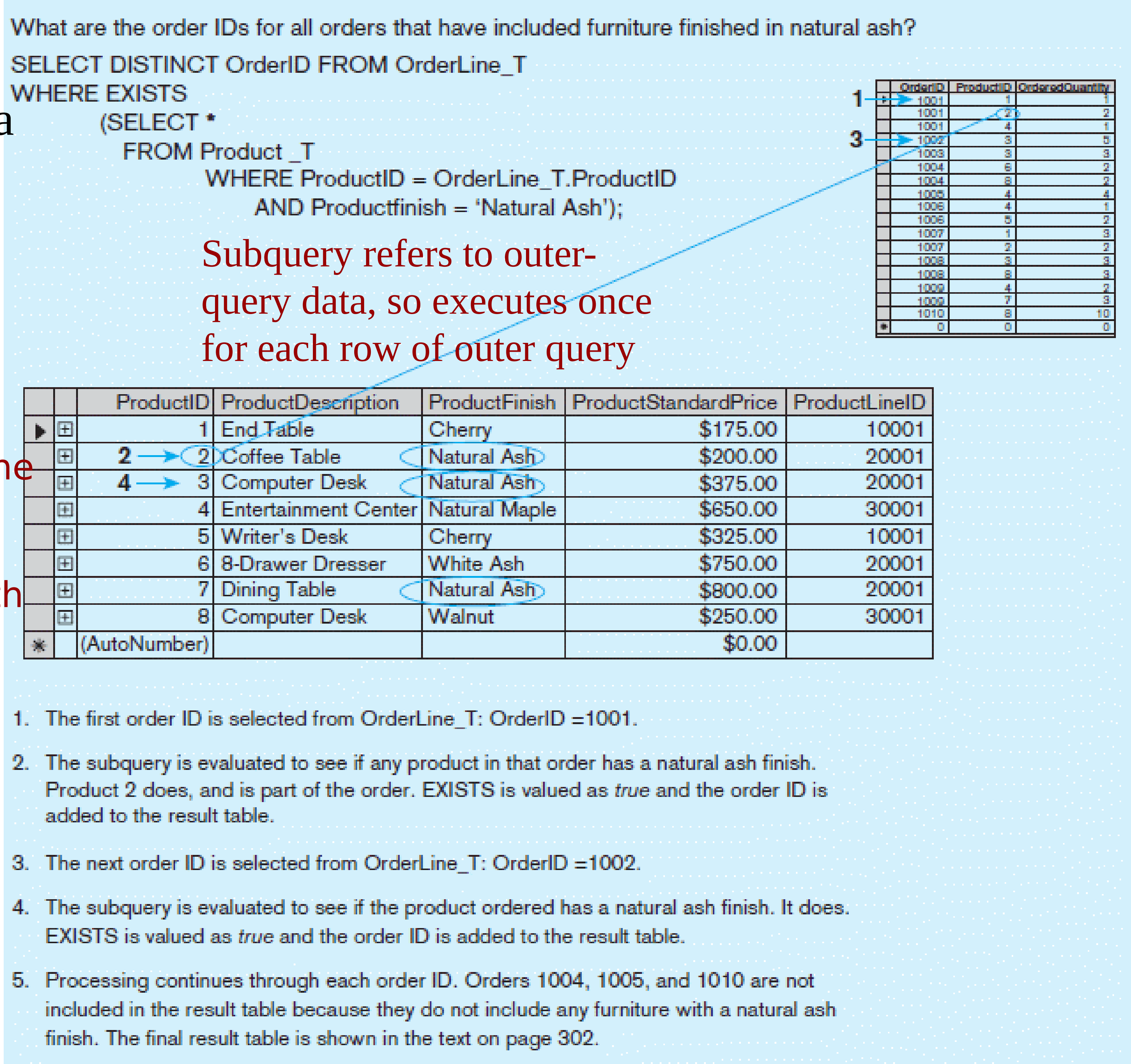


OrderID	ProductID	OrderedQuantity
1001	1	1
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1007	1	3
1007	2	2
1008	3	3
1008	8	3
1009	4	2
1009	7	3
1010	8	10
0	0	0



ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	End Table	Cherry	$175.00	10001
2	Coffee Table	Natural Ash	$200.00	20001
3	Computer Desk	Natural Ash	$375.00	20001
4	Entertainment Center	Natural Maple	$650.00	30001
5	Writer's Desk	Cherry	$325.00	10001
6	8-Drawer Dresser	White Ash	$750.00	20001
7	Dining Table	Natural Ash	$800.00	20001
8	Computer Desk	Walnut	$250.00	30001
(AutoNumber)			$0.00	

Processing a correlated subquery



What are the order IDs for all orders that have included furniture finished in natural ash?

```
SELECT DISTINCT OrderID FROM OrderLine_T
WHERE EXISTS
        (SELECT *
          FROM Product _T
                WHERE ProductID = OrderLine_T.ProductID
                      AND Productfinish = 'Natural Ash');
```

	OrderID	ProductID	OrderedQuantity
	1001	1	1
	1001	2	2
	1001	4	1
	1002	3	5
	1003	3	3
	1004	6	2
	1004	8	2
	1005	4	4
	1006	4	1
	1006	5	2
	1007	1	3
	1007	2	2
	1008	3	3
	1008	8	3
	1009	4	2
	1009	7	3
	1010	8	10
*	0	0	0

Subquery refers to outer-query data, so executes once for each row of outer query

		ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
▶	⊞	1	End Table	Cherry	$175.00	10001
	⊞	2	Coffee Table	Natural Ash	$200.00	20001
	⊞	3	Computer Desk	Natural Ash	$375.00	20001
	⊞	4	Entertainment Center	Natural Maple	$650.00	30001
	⊞	5	Writer's Desk	Cherry	$325.00	10001
	⊞	6	8-Drawer Dresser	White Ash	$750.00	20001
	⊞	7	Dining Table	Natural Ash	$800.00	20001
	⊞	8	Computer Desk	Walnut	$250.00	30001
*		(AutoNumber)			$0.00	

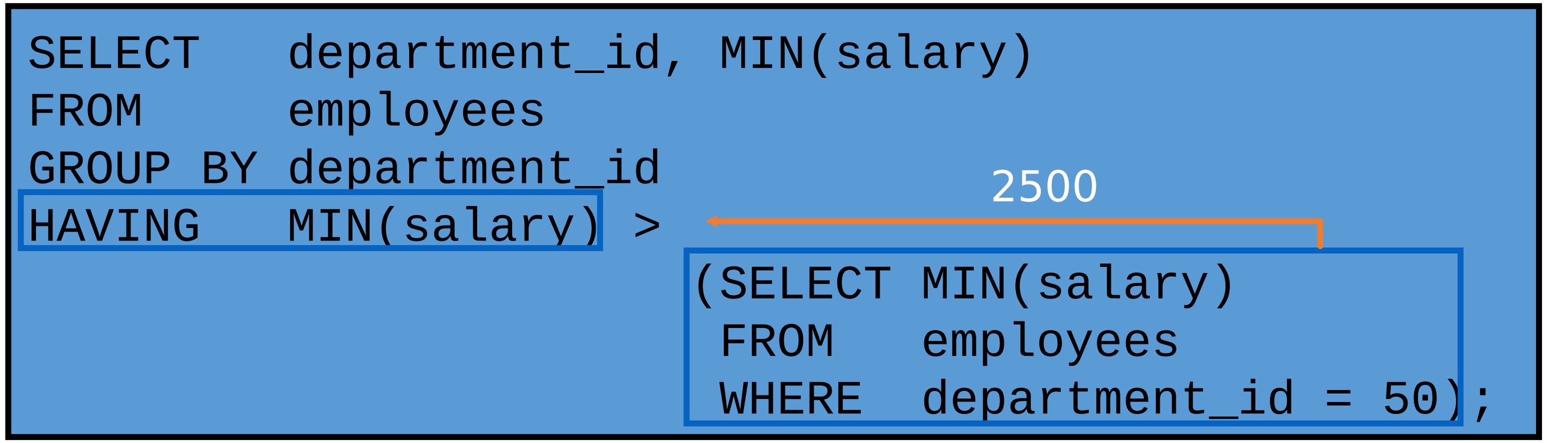
1. The first order ID is selected from OrderLine_T: OrderID =1001.
2. The subquery is evaluated to see if any product in that order has a natural ash finish. Product 2 does, and is part of the order. EXISTS is valued as *true* and the order ID is added to the result table.
3. The next order ID is selected from OrderLine_T: OrderID =1002.
4. The subquery is evaluated to see if the product ordered has a natural ash finish. It does. EXISTS is valued as *true* and the order ID is added to the result table.
5. Processing continues through each order ID. Orders 1004, 1005, and 1010 are not included in the result table because they do not include any furniture with a natural ash finish. The final result table is shown in the text on page 302.

Note: only the orders that involve products with Natural Ash will be included in the final results

The **HAVING** Clause with Subqueries

- Display all the departments that have a minimum salary greater than that of department 50.

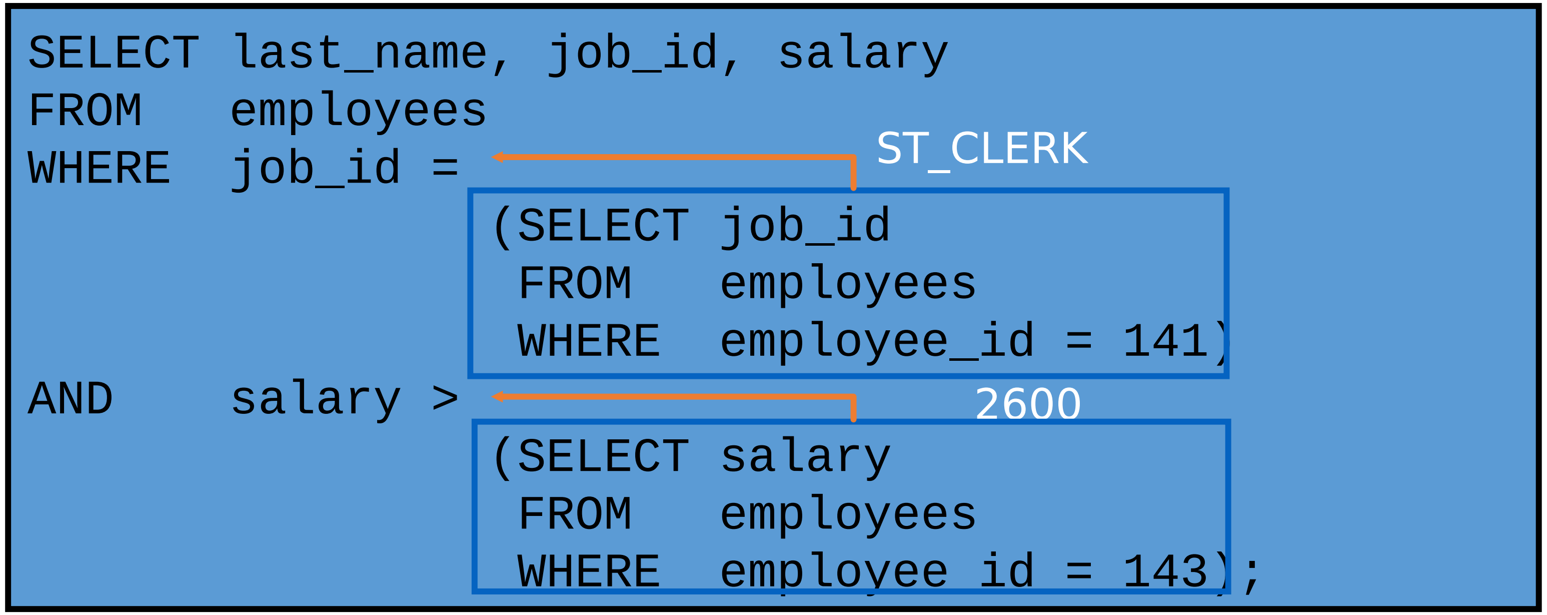
emp_id	dept_id	salary
1001	40	5000
1002	30	4500
1003	50	2500
1004	50	4000
1005	30	3700
1006	40	3500



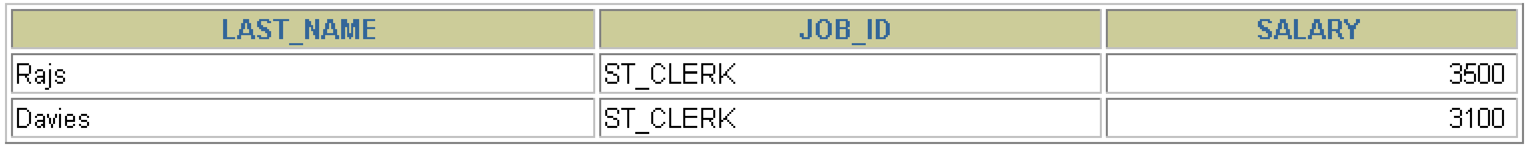
```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                       (SELECT MIN(salary)
                        FROM   employees
                        WHERE  department_id = 50);
```

Exercise: Executing Single-Row Subqueries

display employees whose job ID is the same as that of employee 141 and whose salary is greater than that of employee 143.



```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141)
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  employee_id = 143);
```



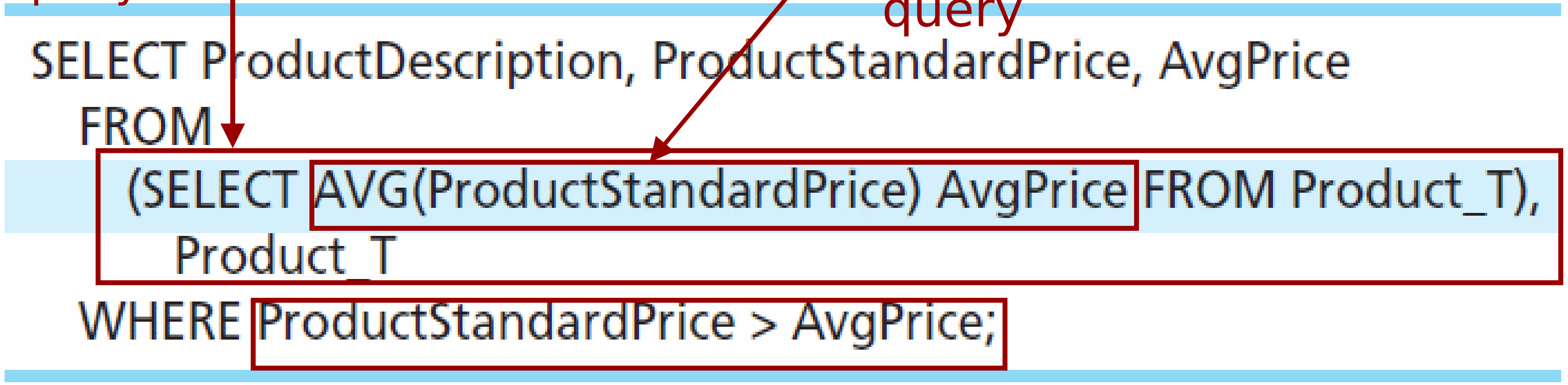
LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

Subquery – Derived Table Example

- Show all products whose standard price is higher than the average price

Subquery forms the derived table used in the FROM clause of the outer query

One column of the subquery is an aggregate function that has an alias name. That alias can then be referred to in the outer query



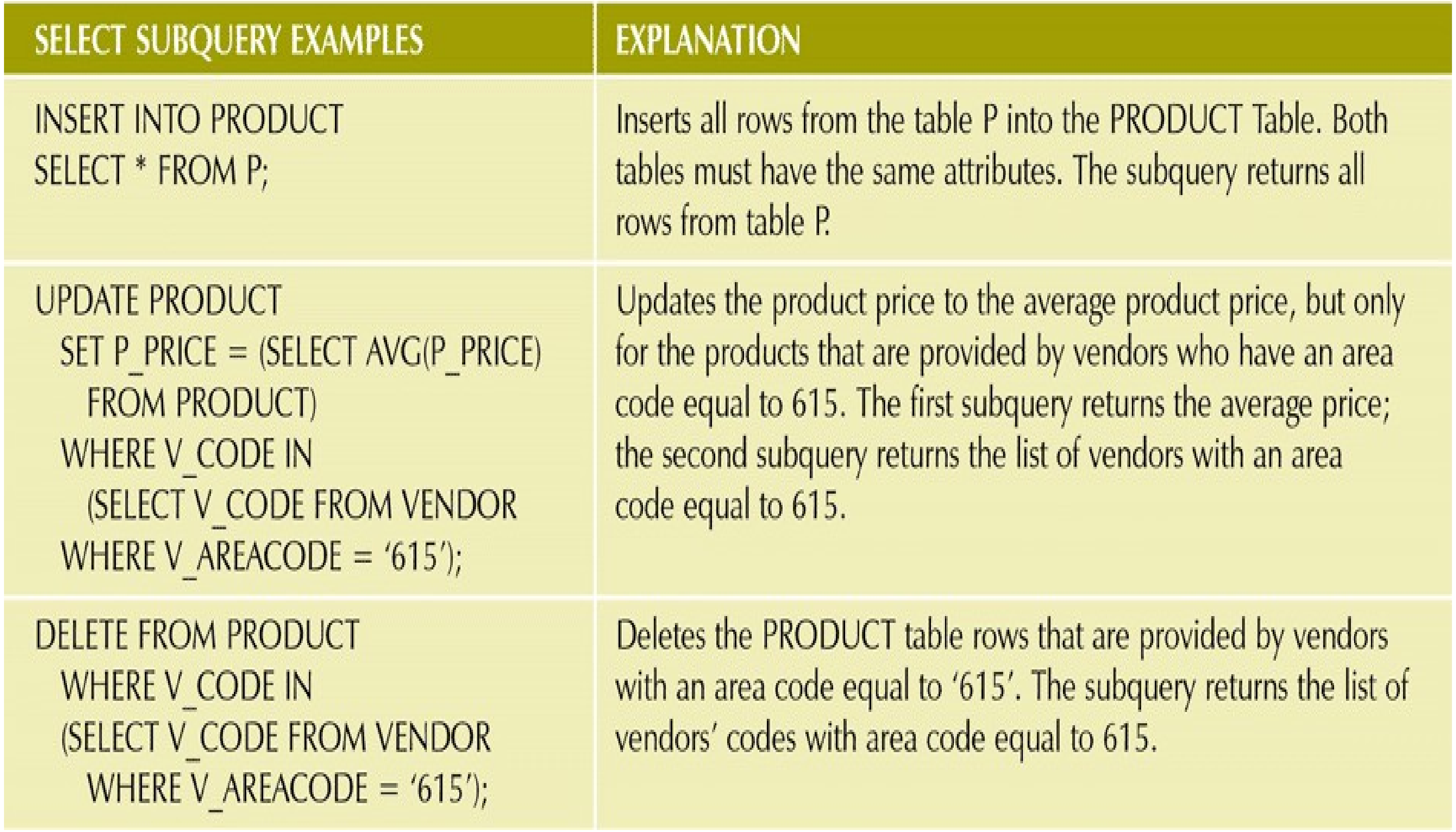
```
SELECT ProductDescription, ProductStandardPrice, AvgPrice
  FROM
    (SELECT AVG(ProductStandardPrice) AvgPrice FROM Product_T),
      Product_T
  WHERE ProductStandardPrice > AvgPrice;
```

The WHERE clause normally cannot include aggregate functions, but because the aggregate is performed in the subquery its result can be used in the outer query's WHERE clause.

Derived table is required when we want to display information from subquery e.g here we want to show both the standard price and the average standard price

SELECT Sub-query Examples

TABLE 7.2 SELECT SUBQUERY EXAMPLES



SELECT SUBQUERY EXAMPLES	EXPLANATION

INSERT INTO PRODUCT
SELECT * FROM P;
	Inserts all rows from the table P into the PRODUCT Table. Both tables must have the same attributes. The subquery returns all rows from table P.

UPDATE PRODUCT
SET P_PRICE = (SELECT AVG(P_PRICE)
FROM PRODUCT)
WHERE V_CODE IN
(SELECT V_CODE FROM VENDOR
WHERE V_AREACODE = '615');
	Updates the product price to the average product price, but only for the products that are provided by vendors who have an area code equal to 615. The first subquery returns the average price; the second subquery returns the list of vendors with an area code equal to 615.

DELETE FROM PRODUCT
WHERE V_CODE IN
(SELECT V_CODE FROM VENDOR
WHERE V_AREACODE = '615');
	Deletes the PRODUCT table rows that are provided by vendors with an area code equal to '615'. The subquery returns the list of vendors' codes with area code equal to 615.