Displaying Data from Multiple Tables

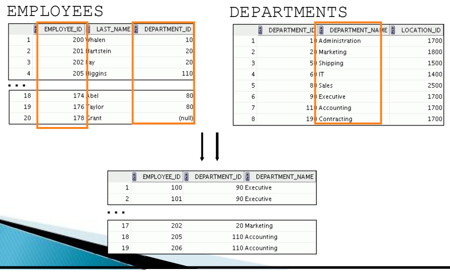
Objectives

- After completing this lesson, you should be able to do the following:
  - Write `SELECT` statements to access data from more than one table using equijoins and nonequijoins
  - Join a table to itself by using a self-join
  - View data that generally does not meet a join condition by using outer joins
  - Generate a Cartesian product of all rows from two or more tables

Tables

- **EMPLOYEES**(EMPLOYEE_ID , FIRST_NAME , LAST_NAME , EMAIL , PHONE_NUMBER , HIRE_DATE , JOB_ID , SALARY , COMMISSION_PCT , MANAGER_ID , DEPARTMENT_ID)
- **DEPARTMENTS**(DEPARTMENT_ID , DEPARTMENT_NAME , MANAGER_ID,LOCATION_ID)
- **LOCATIONS**(LOCATION_ID,STREET_ADDRESS,POSTAL_CODE, CITY,STATE_PROVINCE,COUNTRY_ID)

- A *join* is used to view information from multiple tables.
- You can *join* tables together to view information from more than one table.



EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
. . .			
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	100	90	Executive
2	101	90	Executive
. . .			
17	202	20	Marketing
18	205	110	Accounting
19	206	110	Accounting

Types of Joins

- Joins that are compliant with the SQL:1999 standard include the following:
  - Cross joins
  - Natural joins
  - `USING` clause
  - Full (or two-sided) outer joins
  - Arbitrary join conditions for outer joins

Joining Tables Using SQL:1999 Syntax

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

```
SELECT   table1.column, table2.column
FROM     table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

table1.column denotes the table and column from which data is retrieved

NATURAL JOIN joins two tables based on the same column name

JOIN *table* USING *column_name* performs an equijoin based on the column name

JOIN *table* ON *table1.column_name* performs an equijoin based on the condition in the ON clause, = *table2.column_name*

LEFT/RIGHT/FULL OUTER is used to perform outer joins

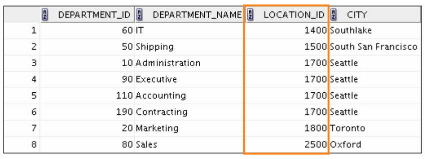
CROSS JOIN returns a Cartesian product from the two tables

Creating Natural Joins

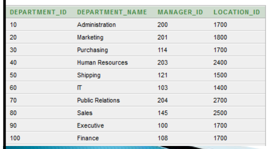
- You can join tables automatically based on columns in the two tables that have matching data types and names. You do this by using the keywords NATURAL JOIN.
  - It selects rows from the two tables that have equal values in all matched columns.
  - If the columns having the same names have different data types, an error is returned.

Retrieving Records with Natural Joins

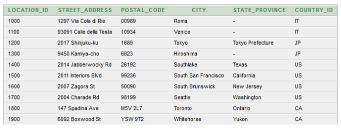
```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;  J1
```



	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

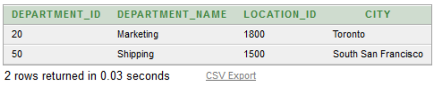


LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1000	1297 Via Cola di Rie	00989	Roma	-	IT
1100	93091 Calle della Testa	10934	Venice	-	IT
1200	2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
1300	9450 Kamiya-cho	6823	Hiroshima	-	JP
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1600	2007 Zagora St	50090	South Brunswick	New Jersey	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
1900	6092 Boxwood St	YSW 9T2	Whitehorse	Yukon	CA

Natural by using a WHERE clause

- limits the rows of output to those with a department ID equal to 20 or 50:

```
SELECT  department_id, department_name,
            location_id, city
   FROM     departments
   NATURAL  JOIN locations
   WHERE    department_id IN (20, 50); J2
```



DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
20	Marketing	1800	Toronto
50	Shipping	1500	South San Francisco

2 rows returned in 0.03 seconds   CSV Export

Creating Joins with the USING Clause

- Natural joins use all columns with matching names and data types to join the tables.
- If several columns have the same names but the data types do not match, natural join can be applied by using the USING clause to specify the columns that should be used for an equijoin.
- Use the USING clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.
- The NATURAL JOIN and USING clauses are mutually exclusive.

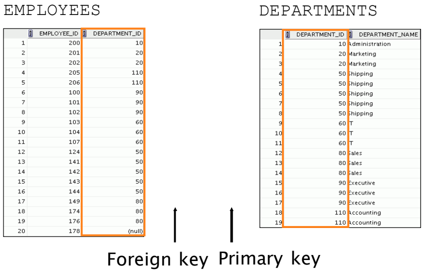
```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING (location_id)
WHERE  location_id = 1400; J3
```

- The following 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; J4
```

- **Note**: The same restriction also applies to NATURAL joins. Therefore, columns that have the same name in both tables must be used without any qualifiers.

Joining Column Names



EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60
11	107	60
12	124	50
13	141	50
14	142	50
15	143	50
16	144	50
17	149	80
18	174	80
19	176	80
20	178	(null)

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	20	Marketing
4	50	Shipping
5	50	Shipping
6	50	Shipping
7	50	Shipping
8	50	Shipping
9	60	IT
10	60	IT
11	60	IT
12	80	Sales
13	80	Sales
14	80	Sales
15	90	Executive
16	90	Executive
17	90	Executive
18	110	Accounting
19	110	Accounting

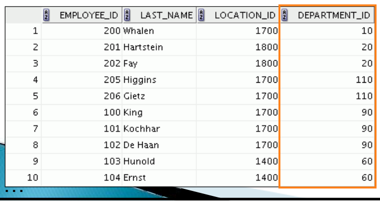
Foreign key Primary key

- The relationship between the EMPLOYEES and DEPARTMENTS tables is an *equijoin;*
- **Note:** Equijoins are also called *simple joins* or *inner joins.*

Retrieving Records with the USING Clause

- Ex. joins the DEPARTMENT_ID column in the EMPLOYEES and DEPARTMENTS tables, and thus shows the location where an employee works.

```
SELECT employees.employee_id, employees.last_name,
       departments.location_id, department_id
FROM   employees JOIN departments
USING (department_id) ; J5
```



	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	205	Higgins	1700	110
5	206	Gietz	1700	110
6	100	King	1700	90
7	101	Kochhar	1700	90
8	102	De Haan	1700	90
9	103	Hunold	1400	60
10	104	Ernst	1400	60

...

Qualifying Ambiguous Column Names

- You need to qualify the names of the columns with the table name to avoid ambiguity.
- Without the table prefixes, the DEPARTMENT_ID column in the SELECT list could be from either the DEPARTMENTS table or the EMPLOYEES table.
- It is necessary to add the table prefix to execute your query

```
SELECT employees.employee_id, employees.last_name,
       departments.department_id, departments.location_id
FROM   employees JOIN departments
ON     employees.department_id = departments.department_id; J6
```

- 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.

- **Note**: When joining with the USING clause, you cannot qualify a column that is used in the USING clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it.

Using 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.
- How table aliases are identified in the FROM clause
- The table name is specified in full, followed by a space and then the table alias.

```
SELECT e.employee_id, e.last_name,
       d.location_id, department_id
FROM   employees e JOIN departments d
USING (department_id) ; J7
```

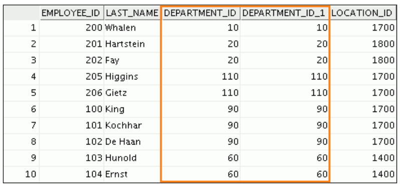
- **Guidelines**
  - Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
  - If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
  - Table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

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 `ON` clause makes code easy to understand.

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 JOIN departments d
ON     (e.department_id = d.department_id); J8
```



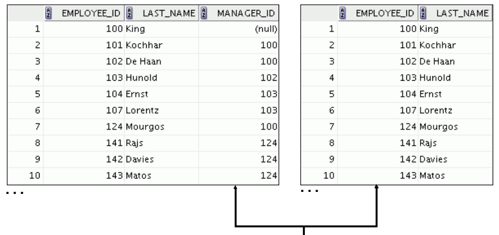
	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	205	Higgins	110	110	1700
5	206	Gietz	110	110	1700
6	100	King	90	90	1700
7	101	Kochhar	90	90	1700
8	102	De Haan	90	90	1700
9	103	Hunold	60	60	1400
10	104	Ernst	60	60	1400

…

- Note: You can also use the ON clause to join columns that have different names.

Self-Joins Using the ON Clause

- Sometimes you need to join a table to itself.
  Ex. To find the name of each employee's manager



EMPLOYEES (WORKER)

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

…

EMPLOYEES (MANAGER)

	EMPLOYEE_ID	LAST_NAME
1	100	King
2	101	Kochhar
3	102	De Haan
4	103	Hunold
5	104	Ernst
6	107	Lorentz
7	124	Mourgos
8	141	Rajs
9	142	Davies
10	143	Matos

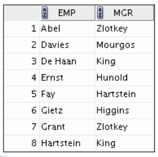
…

`MANAGER_ID` in the `WORKER` table is equal to `EMPLOYEE_ID` in the `MANAGER` table.

Self-Joins Using the ON Clause

- The `ON` clause can also be used to join columns that have different names, within the same table or in a different table.

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON     (e.manager_id = m.employee_id); J9
```



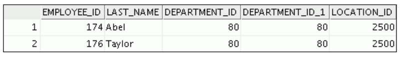
	EMP	MGR
1	Abel	Zlotkey
2	Davies	Mourgos
3	De Haan	King
4	Ernst	Hunold
5	Fay	Hartstein
6	Gietz	Higgins
7	Grant	Zlotkey
8	Hartstein	King

…

Applying Additional Conditions to a Join

- You can apply additional conditions to the join.
- Ex. Performs a join on the `EMPLOYEES` and `DEPARTMENTS` tables and, in addition, displays only employees who have a manager ID of 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 ;   J10
```



	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

- To add additional conditions to the `ON` clause, you can add `AND` clauses.

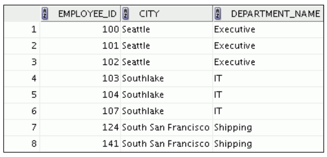
- Alternatively, you can use a `WHERE` clause to apply additional conditions:

```
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)
WHERE  e.manager_id = 149;      J11
```

Creating Three-Way Joins with the ON Clause

- A three-way join is a join of three tables.
- In SQL:1999-compliant syntax, joins are performed from left to right. So, the first join to be performed is EMPLOYEES JOIN DEPARTMENTS.
- The first join condition can reference columns in EMPLOYEES and DEPARTMENTS but cannot reference columns in LOCATIONS.
- The second join condition can reference columns from all three tables.

```
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;    J12
```

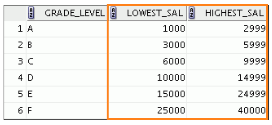


	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping

Nonequijoins

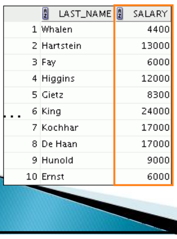
- A nonequijoin is a join condition containing something other than an equality operator.
- The relationship between the EMPLOYEES table and the JOB_GRADES table is an example of a nonequijoin
- The relationship is obtained using an operator other than equality (=).

JOB_GRADES



	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

EMPLOYEES



	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000

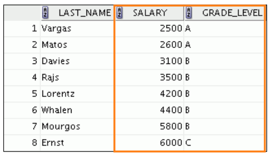
...

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

Retrieving Records with Nonequijoins

Ex. creates a nonequijoin to evaluate an employee's job title. The salary must be *between* any pair of the low and high salary ranges.

```
SELECT e.last_name, e.salary, j.job_title
FROM   employees e JOIN jobs j
ON     e.salary
       BETWEEN j.min_salary AND j.max_salary;  J13
```

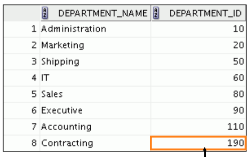


	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C

Outer Joins

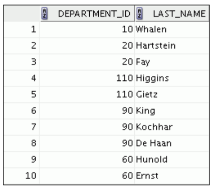
- If a row does not satisfy a join condition, the row does not appear in the query result.

DEPARTMENTS



	DEPARTMENT_NAME	DEPARTMENT_ID
1	Administration	10
2	Marketing	20
3	Shipping	50
4	IT	60
5	Sales	80
6	Executive	90
7	Accounting	110
8	Contracting	190

EMPLOYEES



	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst

...

There are no employees in department 190.

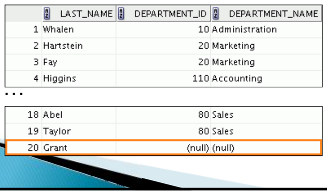
INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an inner join.
- A join between two tables that returns the results of the inner join as well as unmatched rows from the left (or right) tables is called a left (or right) 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.
  - There are three types of outer joins:
    - LEFT OUTER
    - RIGHT OUTER
    - FULL OUTER

LEFT OUTER JOIN

- Query retrieves all rows in the EMPLOYEES table, which is the table on the left even if there is no match in the DEPARTMENTS table.

```
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) ;  J14
```

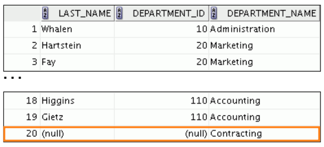


	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting
...			
18	Abel	80	Sales
19	Taylor	80	Sales
20	Grant	(null)	(null)

RIGHT OUTER JOIN

- Query retrieves all rows in the DEPARTMENTS table, which is the table on the right even if there is no match in the EMPLOYEES 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) ;  J15
```

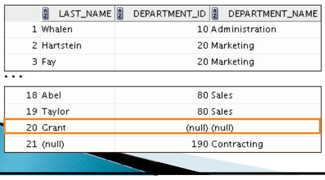


	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
...			
18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	(null)	Contracting

FULL OUTER JOIN

- 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, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;  J16
```



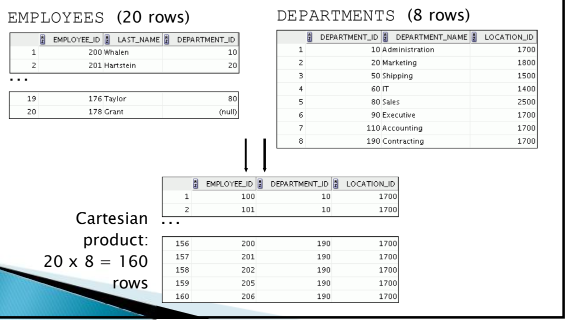
	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
...			
18	Abel	80	Sales
19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

Cartesian Products

- When a join condition is invalid or omitted completely, the result is a *Cartesian product*, in which all combinations of rows are displayed.
- All rows in the first table are joined to all rows in the second table.
- A Cartesian product tends to generate a large number of rows, and the result is rarely useful.
- You should always include a valid join condition unless you have a specific need to combine all rows from all tables.

Generating a Cartesian Product

- shows employee last name and department name from the EMPLOYEES and DEPARTMENTS tables.
- Because no join condition has been specified



EMPLOYEES (20 rows)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
...			
19	176	Taylor	80
20	178	Grant	(null)

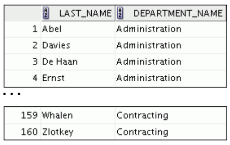
DEPARTMENTS (8 rows)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	100	10	1700
2	101	10	1700
...			
156	200	190	1700
157	201	190	1700
158	202	190	1700
159	205	190	1700
160	206	190	1700

Cartesian product: 20 x 8 = 160 rows

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;  J17
```



	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
...		
159	Whalen	Contracting
160	Zlotkey	Contracting