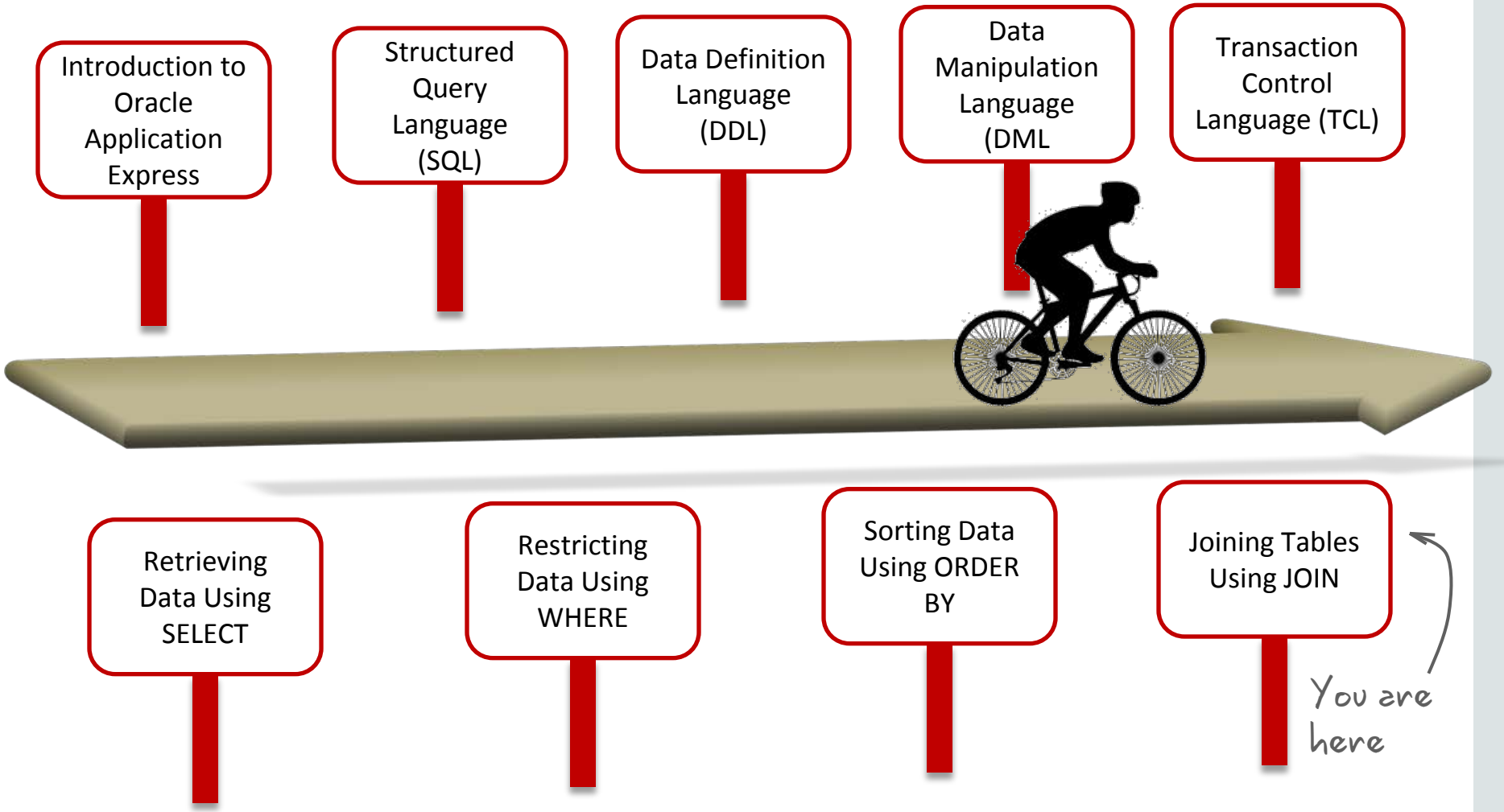

ORACLE
ACADEMY

Database Foundations

6-9
Joining Tables Using JOIN

ORACLE ACADEMY

Roadmap



Objectives

This lesson covers the following objectives:

- Write SELECT statements to access data from more than one table using equijoins and non-equijoins
- Use a self-join to join a table to itself
- Use OUTER joins view data that generally does not meet a join condition
- Generate a Cartesian product (cross join) of all rows from two or more tables

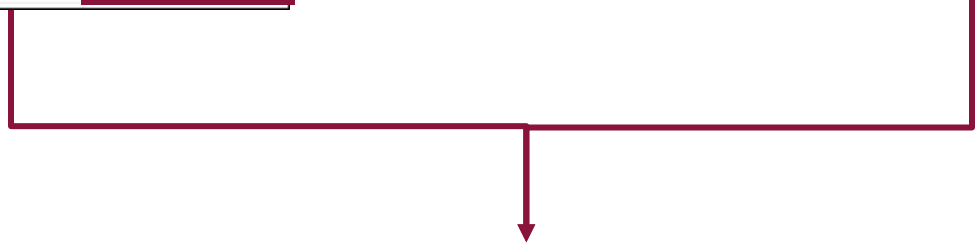
Obtaining Data from Multiple Tables

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
40	Human Resources	2400



EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
114	30	Purchasing

Types of Joins

Joins that are compliant with the SQL:1999 standard:

- Natural join with the NATURAL JOIN clause
- Join with the USING Clause
- Join with the ON Clause
- OUTER joins:
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN
- CROSS JOIN

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];
```

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to increase the speed of parsing the statement.
- Instead of full table name prefixes, use table aliases. (Table alias gives a table a shorter name, keeps SQL code smaller, uses less memory)
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

Creating Natural Joins

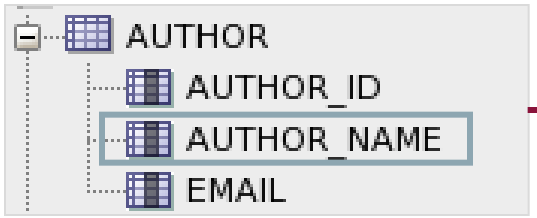
- The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name and the same data type.
- It selects rows from the two tables that have equal values in all matched columns.
- If columns with the same names have different data types, an error is returned.

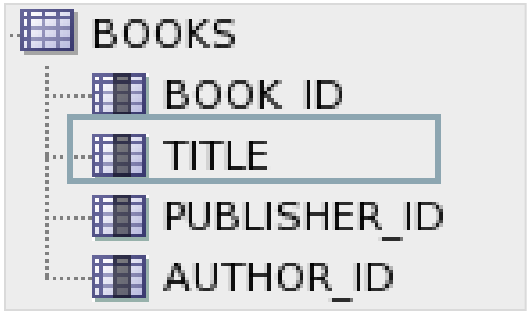
Retrieving Records with Natural Joins

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

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
30	Purchasing	1700	Seattle
90	Executive	1700	Seattle
100	Finance	1700	Seattle
110	Accounting	1700	Seattle
120	Treasury	1700	Seattle

Case Scenario: Natural Join





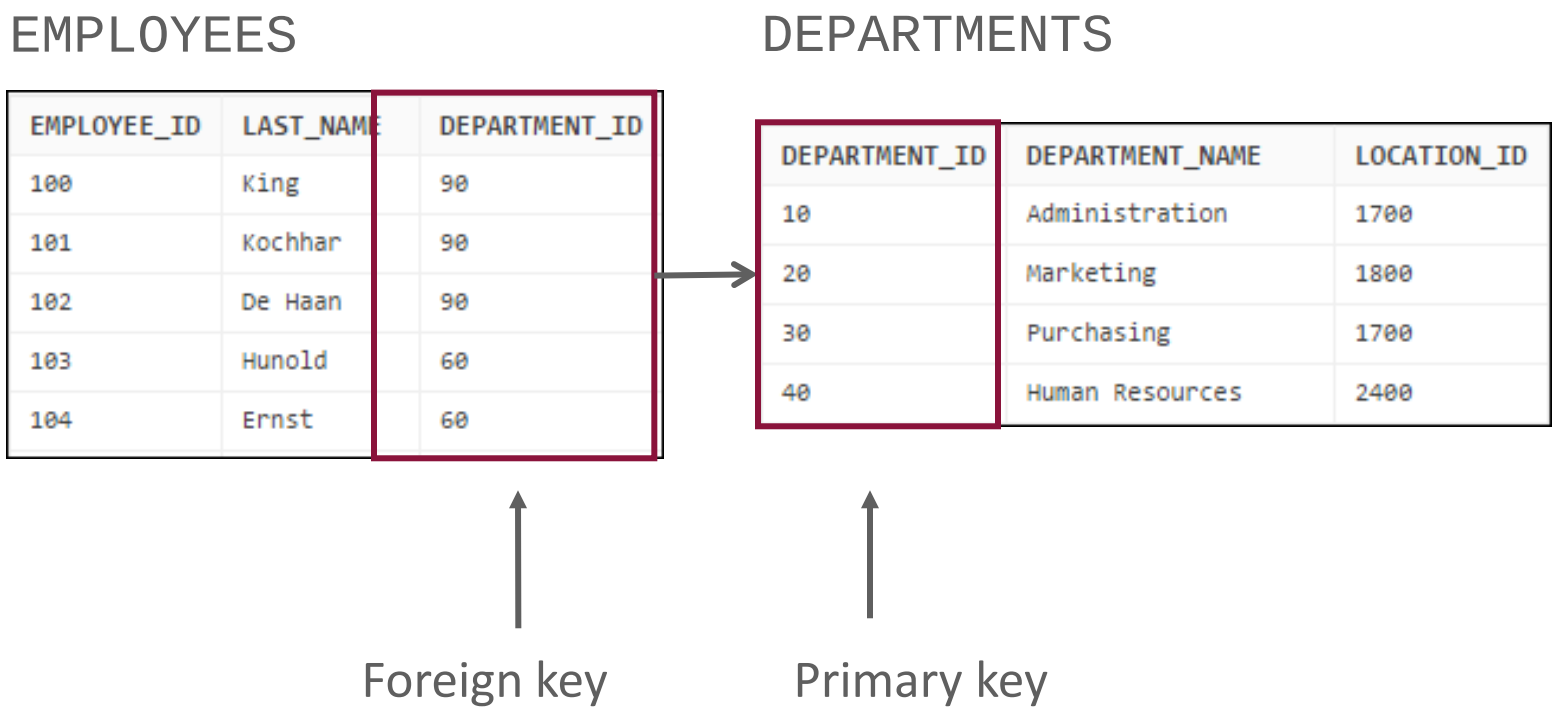

```
SELECT TITLE,AUTHOR_NAME
FROM AUTHOR NATURAL JOIN BOOKS;

TITLE
Florentine Tragedy
A Vision
Citizen of the World
The Complete Poetical Works of Oliver Goldsmith
Androcles and the Lion
An Unsocial Socialist
A Thing of Beauty is a Joy Forever
Beyond the Pale
The Clicking of Cuthbert
Bride of Frankenstein
Shelley Poetry and Prose
War and Peace

 12 rows selected
```

Data retrieved using a natural join

Creating Joins with the USING Clause

- If more than one column has the same name in two tables, use the `USING` clause to specify the single column for the `JOIN` instead of a `NATURAL JOIN`.
- The `USING` clause can also be used to match columns that have the same name but different data types.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

Joining Column Names



Retrieving Records with the USING Clause

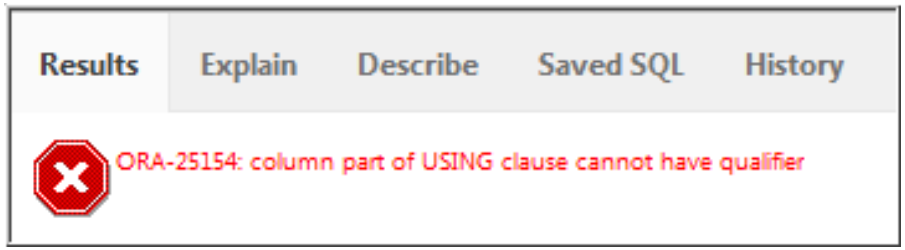
```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
114	Raphaely	1700	30
115	Khoo	1700	30
116	Baida	1700	30
117	Tobias	1700	30
118	Himuro	1700	30

Using Table Aliases with the USING Clause

- Do not use a table name or alias in the USING clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

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



Creating Joins with the ON Clause

- A `NATURAL JOIN` creates an equijoin of all columns with the same name and data type.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.

Creating Joins with the ON Clause

- The ON clause makes code easy to understand.
- A USING clause creates an equijoin between two tables using one column with the same name, regardless of the data type.
- An ON clause creates an equijoin between two tables using one column from each table, regardless of the name or data type.

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);
```

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
114	Raphaely	30	30	1700
115	Khoo	30	30	1700
116	Baida	30	30	1700
117	Tobias	30	30	1700
118	Himuro	30	30	1700

...

Creating Three-Way Joins with the ON Clause

```
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;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
100	Seattle	Executive
101	Seattle	Executive
102	Seattle	Executive
103	Southlake	IT
104	Southlake	IT
105	Southlake	IT
106	Southlake	IT
107	Southlake	IT

...

Case Scenario: ON Clause

Retrieving data from three tables

```
SELECT TITLE,TRANSACTION_ID, AUTHOR_NAME
FROM AUTHOR f
JOIN BOOKS b
ON f.AUTHOR_ID = b.AUTHOR_ID
JOIN BOOK_TRANSACTION t
ON b.BOOK_ID = t.BOOK_ID
;
```

Successful retrieval of data by using the ON clause

TITLE	TRANSACTION_ID	AUTHOR_NAME
War and Peace	OD0001	Leo Tolstoy
The Clicking of Cuthbert	OD0002	P. G. Wodehouse
An Unsocial Socialist	OD0003	George Bernard Shaw

Applying Additional Conditions to a Join

Use the AND clause or the 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)
AND    e.manager_id = 149 ;
```

Or

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

Joining a Table to Itself

EMPLOYEES (WORKER)

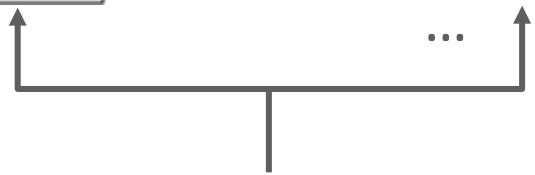
EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...



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

Self-Joins Using the ON Clause

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON    (worker.manager_id = manager.employee_id);
```

EMP	MGR
Smith	Cambrault
Ozer	Cambrault
Kumar	Cambrault
Fox	Cambrault
Bloom	Cambrault
Bates	Cambrault
Hunold	De Haan
Vishney	Errazuriz

...

Nonequijoins

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

...

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

The JOB_GRADES table defines the LOWEST_SAL and HIGHEST_SAL range of values for each GRADE_LEVEL. Therefore, the GRADE_LEVEL column can be used to assign grades to each employee.

Retrieving Records with Nonequijoins

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

	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
9	Fay	6000	C
10	Grant	7000	C

...

Returning Records with No Direct Match Using OUTER Joins

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

There are no employees in department 190.

Equijoin with 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
...		
18	80	Abel
19	80	Taylor

Employee "Grant" has not been assigned a department ID.

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 the unmatched rows from the left (or right) table 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.

LEFT OUTER JOIN

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
De Haan	90	Executive
Hunold	60	IT
Ernst	60	IT
Austin	60	IT
Pataballa	60	IT
Lorentz	60	IT

...

Grant	50	Shipping

RIGHT OUTER JOIN

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Tobias	30	Purchasing
Colmenares	30	Purchasing
Baida	30	Purchasing
Raphaely	30	Purchasing

...

-	230	IT Helpdesk
-	240	Government Sales
-	250	Retail Sales
-	260	Recruiting
-	270	Payroll

FULL OUTER JOIN

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
De Haan	90	Executive
Hunold	60	IT
Ernst	60	IT
Austin	60	IT

...

Grant	-	-

-	190	Contracting

Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted.
  - A join condition is invalid.
- Always include a valid join condition if you want to avoid a Cartesian product.

Generating a Cartesian Product

EMPLOYEES (20 rows)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110

...

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

Cartesian product:
20 x 8 = 160 rows

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700

...

21	200	10	1800
22	201	20	1800

...

159	176	80	1700
160	178	(null)	1700

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 ;
```

Summary

In this lesson, you should have learned how to:

- Write SELECT statements to access data from more than one table using equijoins and non-equijoins
- Use a self-join to join a table to itself
- Use OUTER joins view data that generally does not meet a join condition
- Generate a Cartesian product (cross join) of all rows from two or more tables


ORACLE
ACADEMY