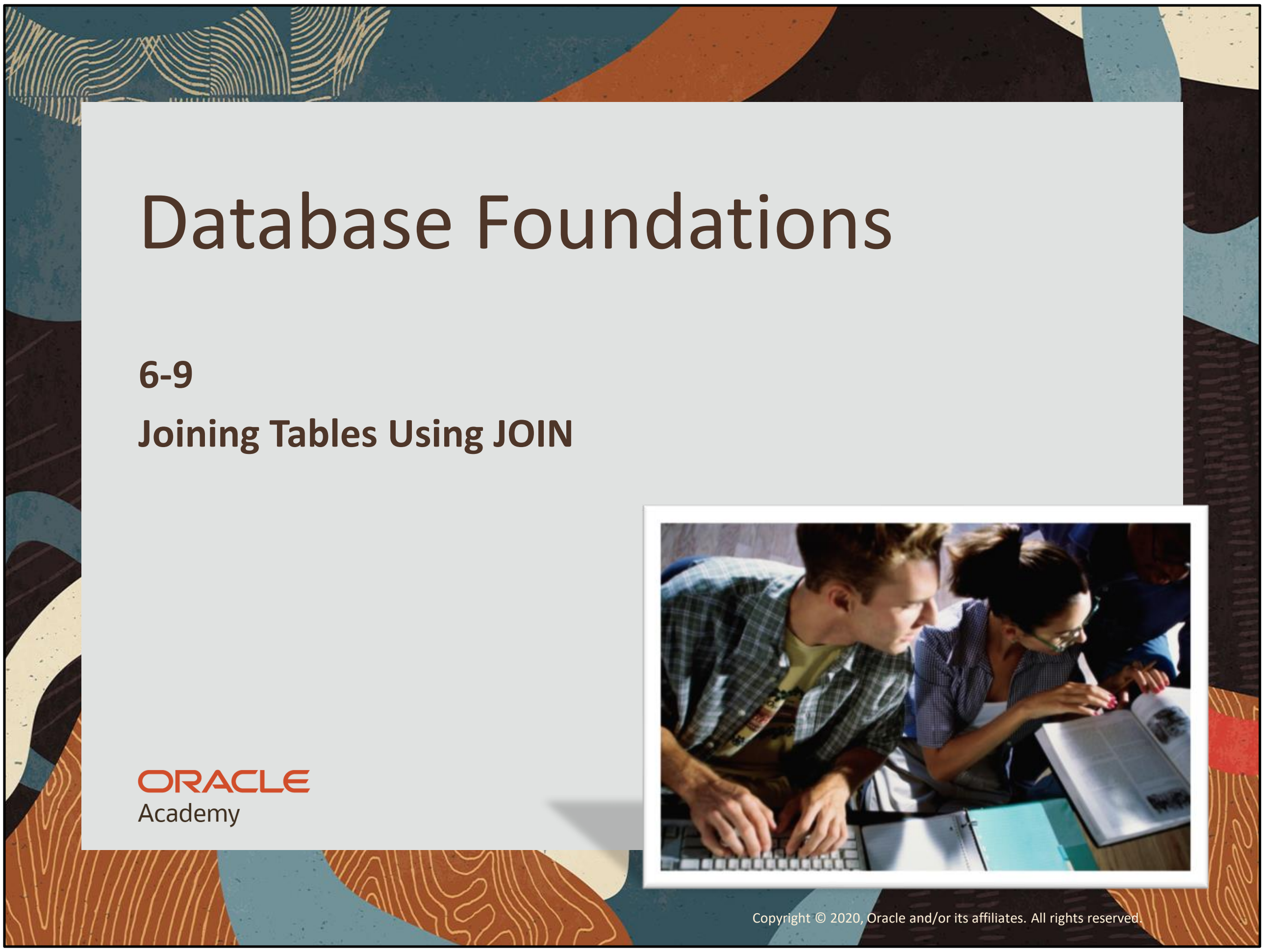
ORACLE

Academy

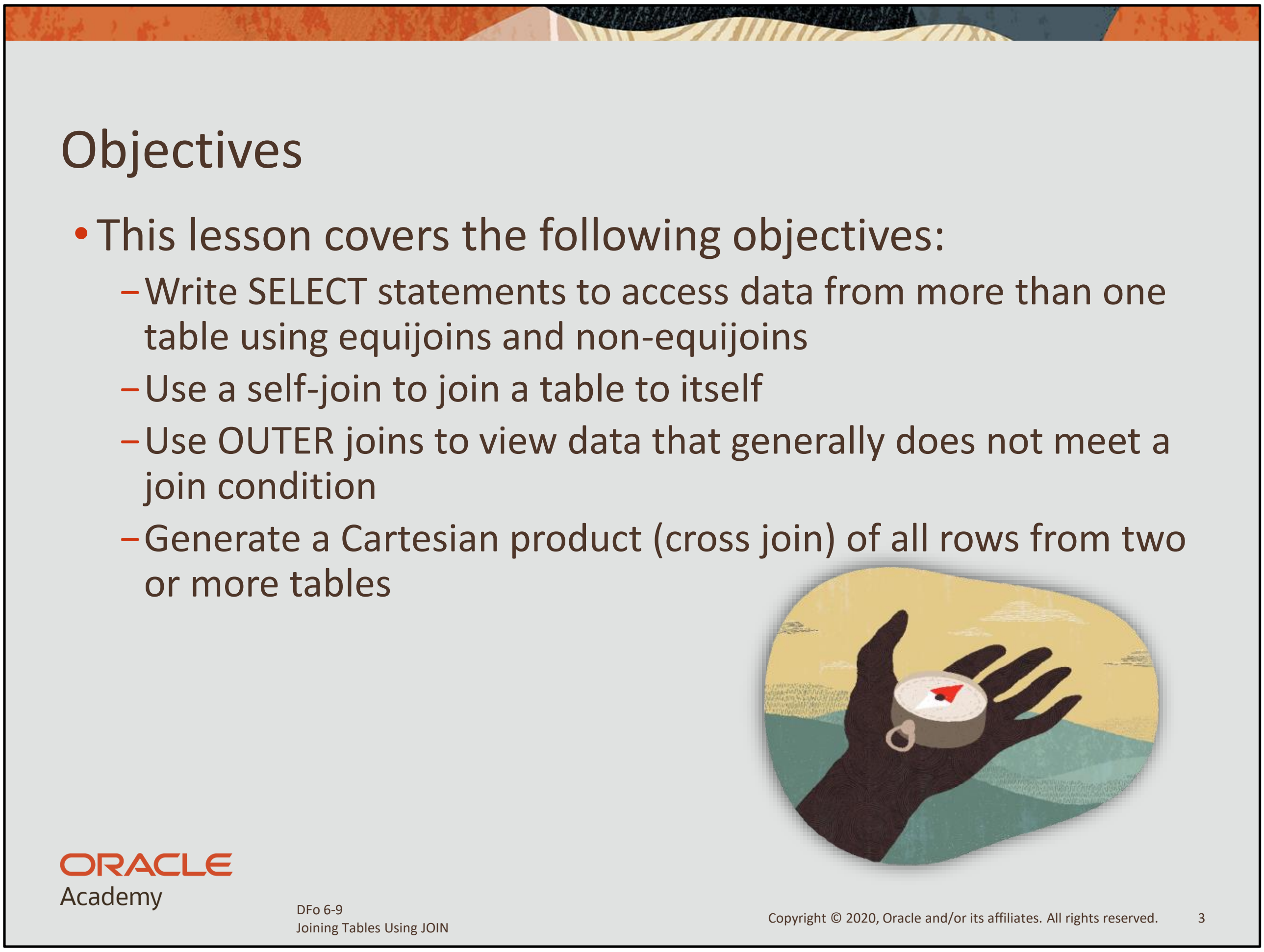


Database Foundations

6-9

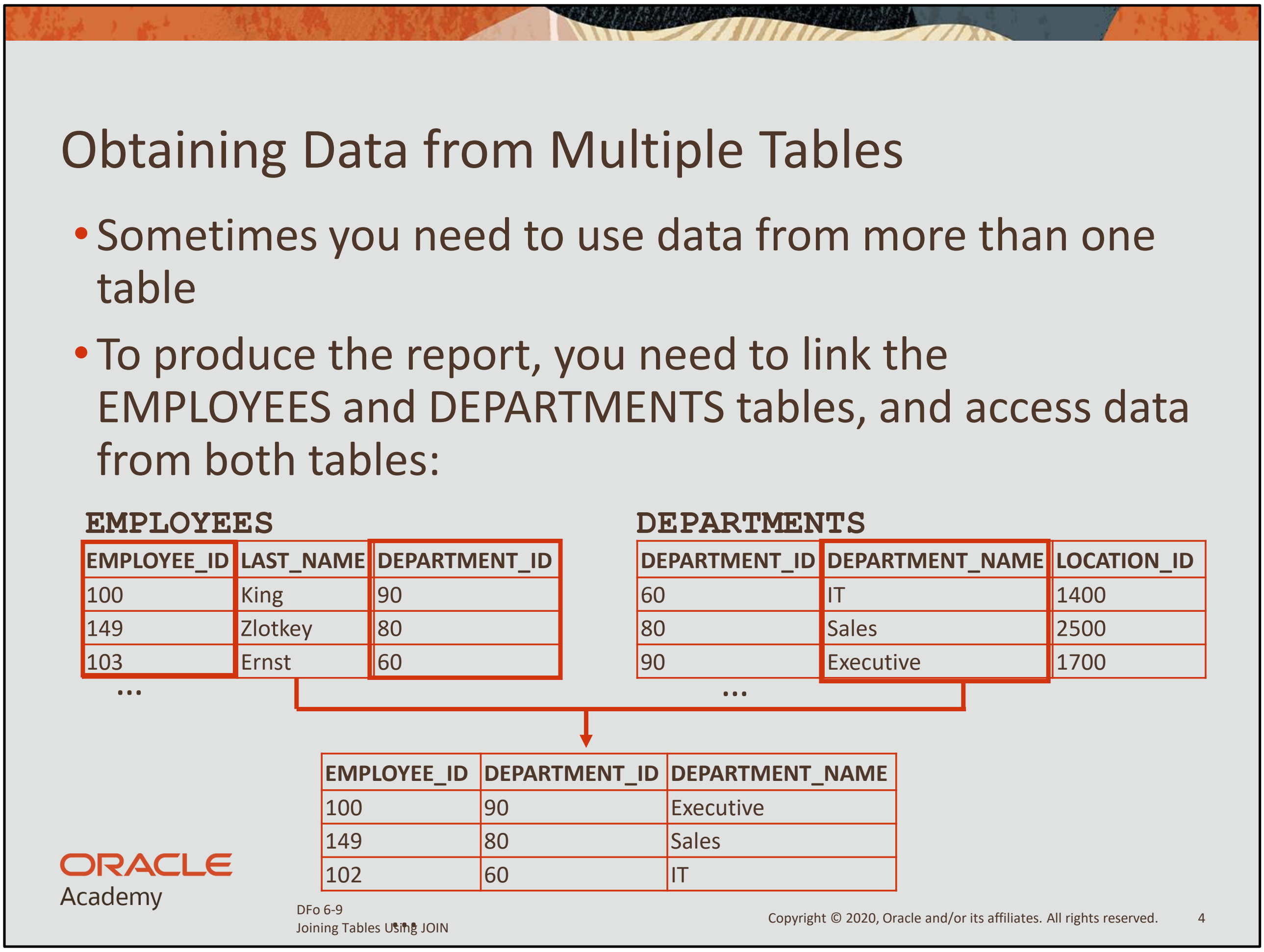
Joining Tables Using JOIN

ORACLE
Academy



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 to 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

- Sometimes you need to use data from more than one table
- To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables, and access data from both tables:

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
149	Zlotkey	80
103	Ernst	60

…

DEPARTMENTS

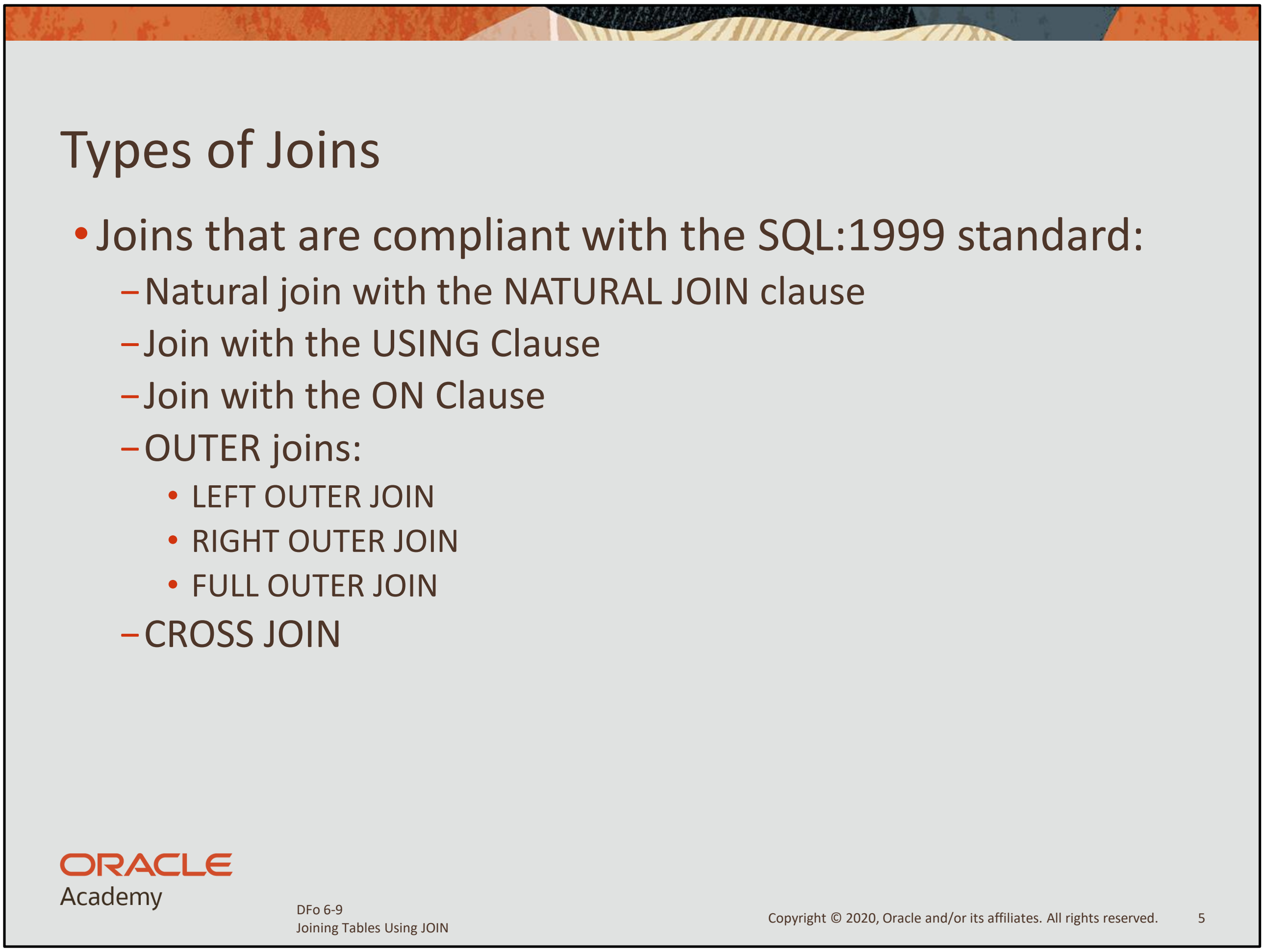
DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
60	IT	1400
80	Sales	2500
90	Executive	1700

…

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
100	90	Executive
149	80	Sales
102	60	IT

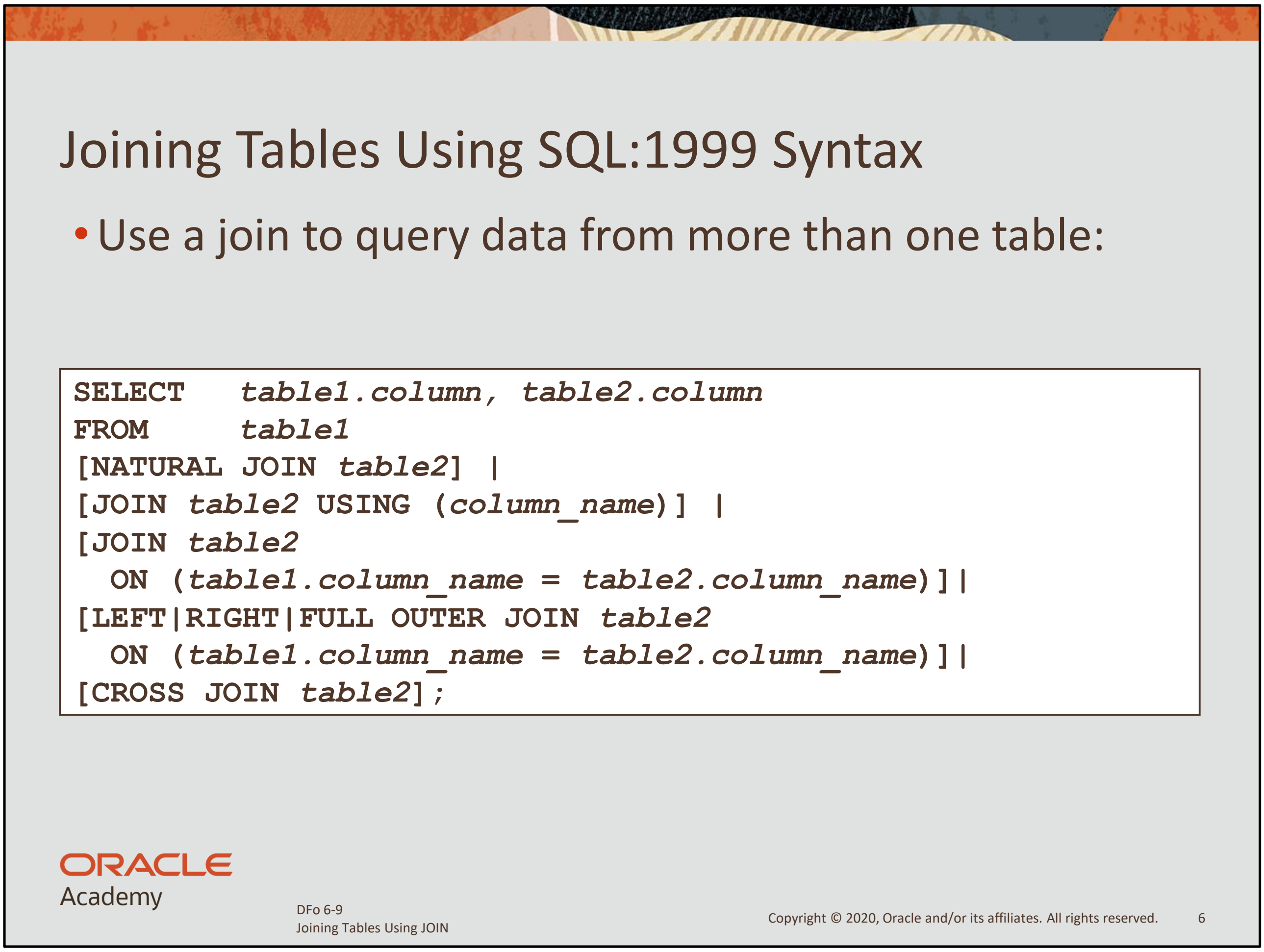
…

ORACLE
Academy



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

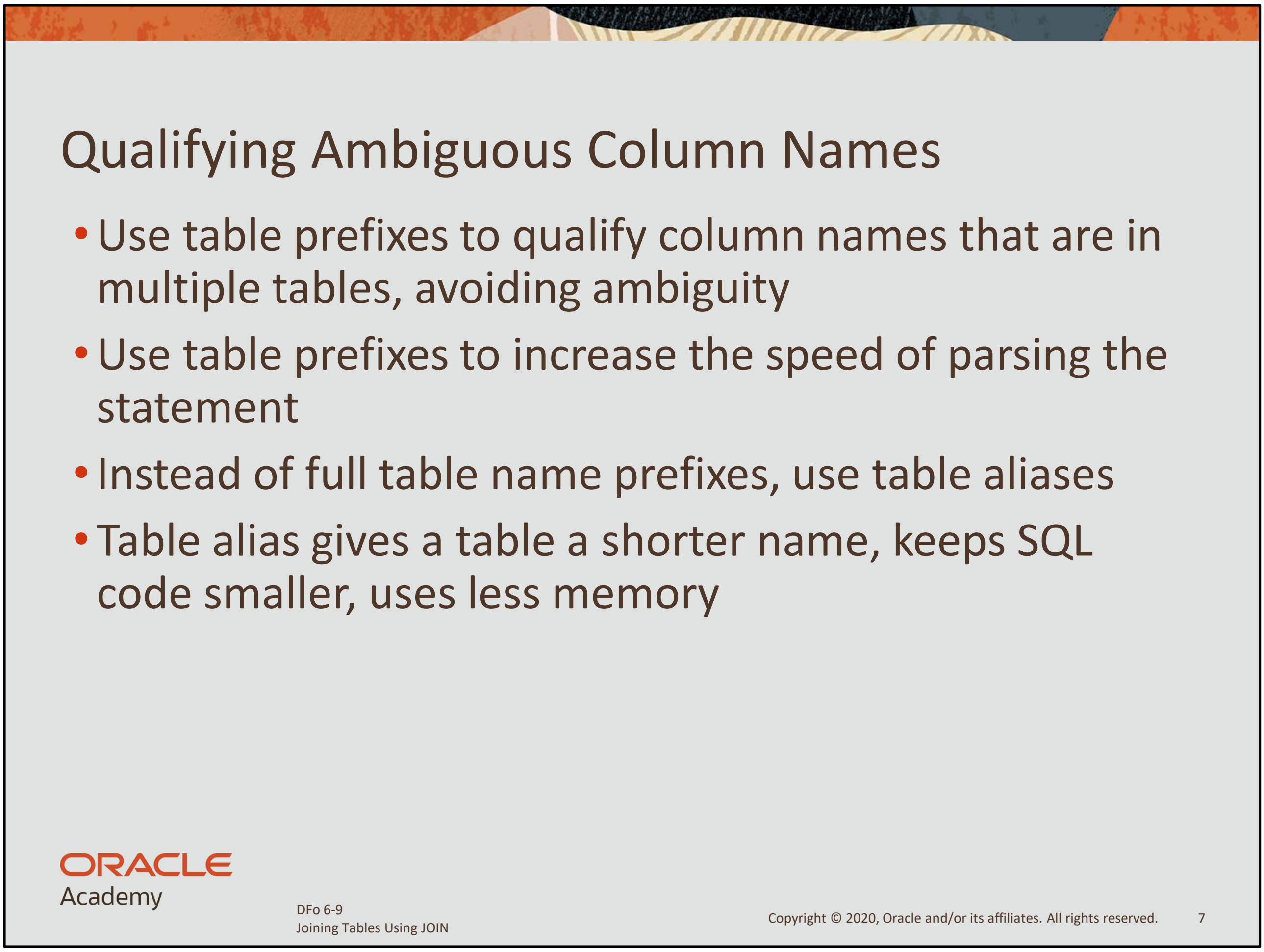
- Use a join to query data from more than one table:

```
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];
```

ORACLE
Academy

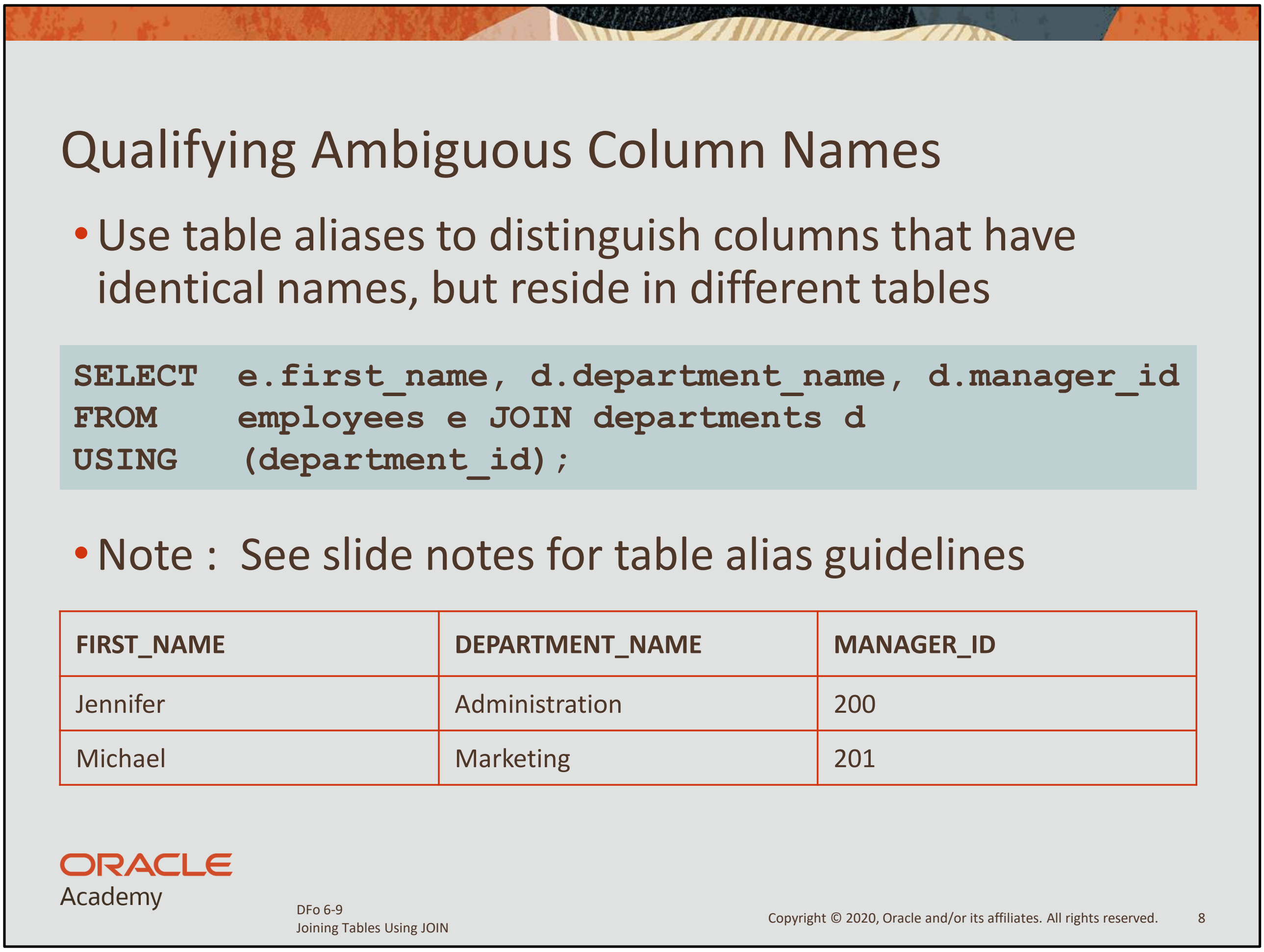
In the syntax:

- table1.column denotes the table and the column from which data is retrieved.
- NATURAL JOIN joins two tables based on the same column name.
- JOIN table2 USING column_name performs an equijoin based on the column name.
- JOIN table2 ON table1.column_name = table2.column_name performs an equijoin based on the condition in the ON clause.
- LEFT/RIGHT/FULL OUTER is used to perform OUTER joins.
- CROSS JOIN returns a Cartesian product from the two tables.



Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables, avoiding ambiguity
- 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



Qualifying Ambiguous Column Names

- Use table aliases to distinguish columns that have identical names, but reside in different tables

```
SELECT  e.first_name, d.department_name, d.manager_id
FROM    employees e JOIN departments d
USING   (department_id);
```

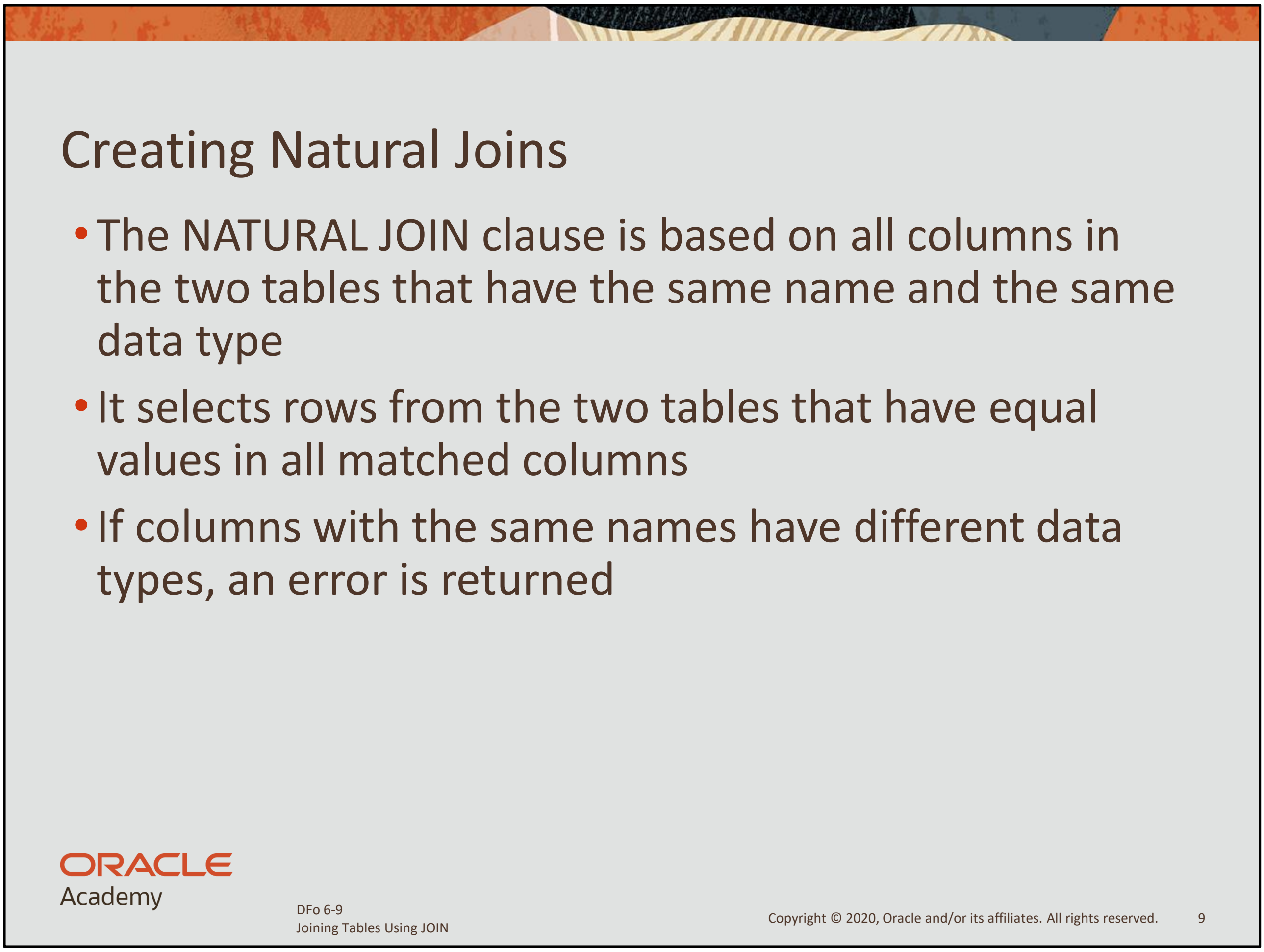
- Note : See slide notes for table alias guidelines

FIRST_NAME	DEPARTMENT_NAME	MANAGER_ID
Jennifer	Administration	200
Michael	Marketing	201

ORACLE
Academy

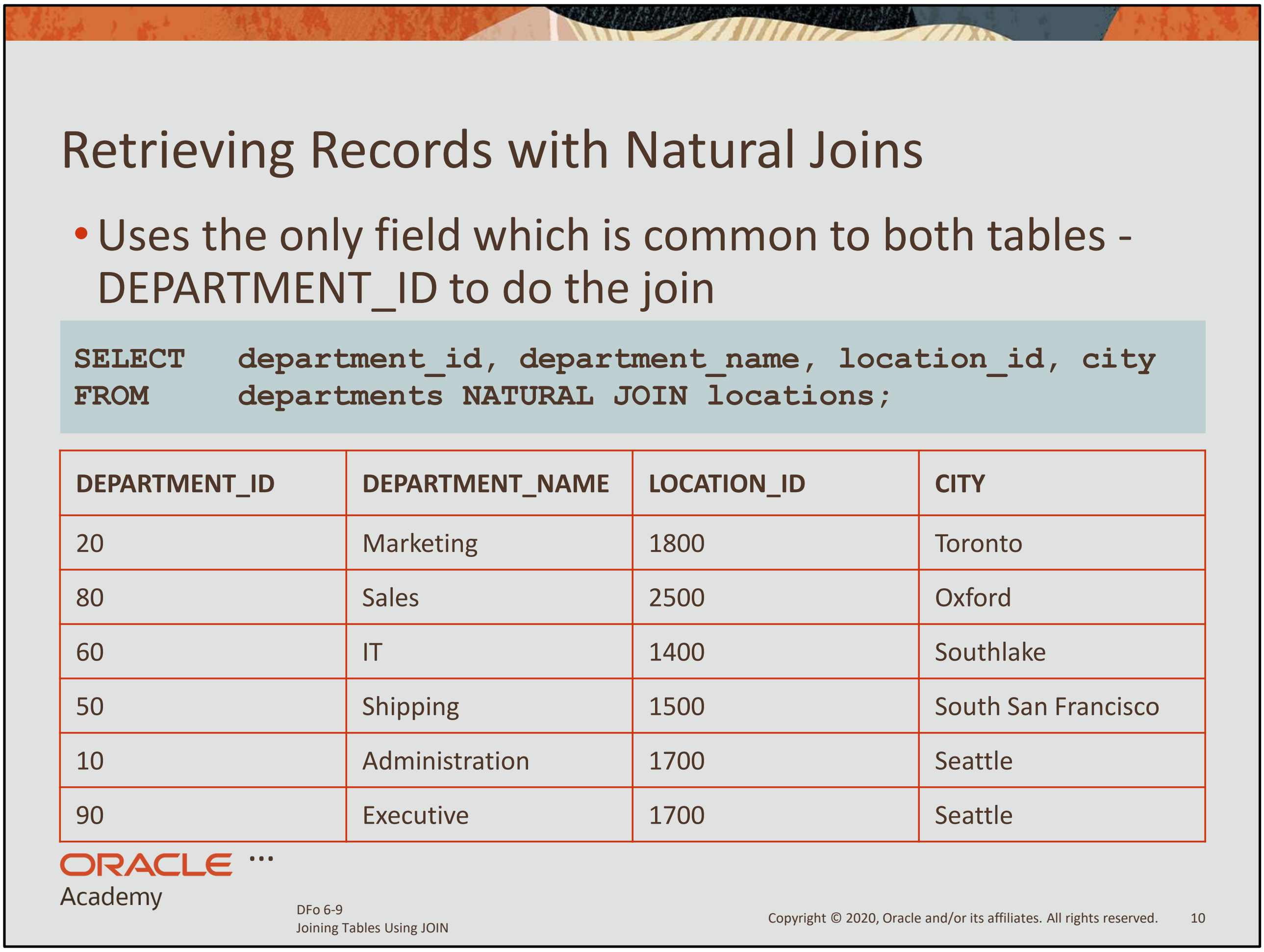
Guidelines

- The table name is specified in full, followed by a space, and then the table alias. For example, the EMPLOYEES table can be given an alias of e and the DEPARTMENTS table an alias of d.
- Table aliases can be up to 30 characters long, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the FROM clause, that table alias must be substituted for the table name throughout the SELECT statement (but not in USING clause – this will be discussed later.
- Table aliases should be meaningful.
- The table alias is valid for only the current SELECT statement.



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

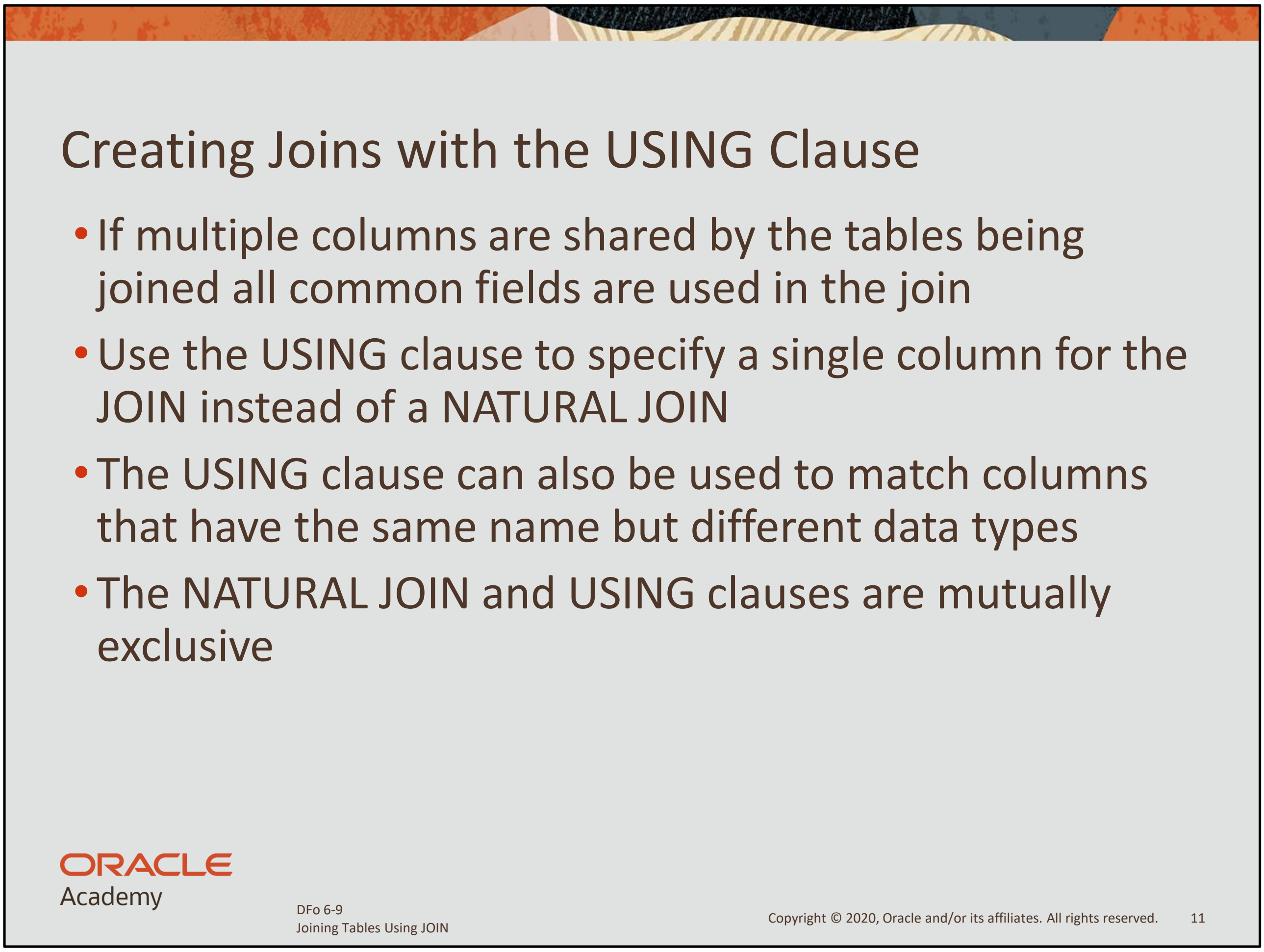


Retrieving Records with Natural Joins

- Uses the only field which is common to both tables - DEPARTMENT_ID to do the join

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

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
20	Marketing	1800	Toronto
80	Sales	2500	Oxford
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle



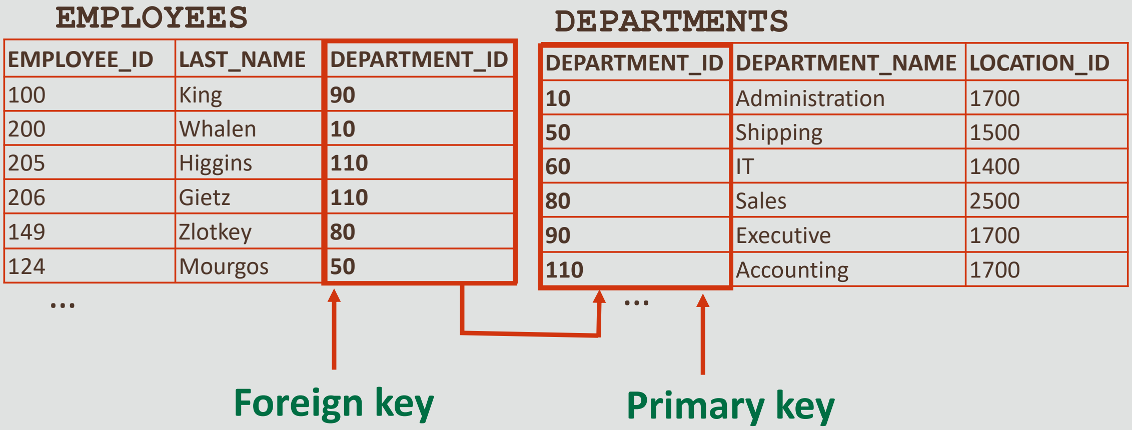
Creating Joins with the USING Clause

- If multiple columns are shared by the tables being joined all common fields are used in the join
- Use the USING clause to specify a 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

ORACLE
Academy

Joining Column Names

- Values in the DEPARTMENT_ID column in both the tables must be equal. This is called an equijoin (also called simple or inner join)



EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	**90**
200	Whalen	**10**
205	Higgins	**110**
206	Gietz	**110**
149	Zlotkey	**80**
124	Mourgos	**50**

…

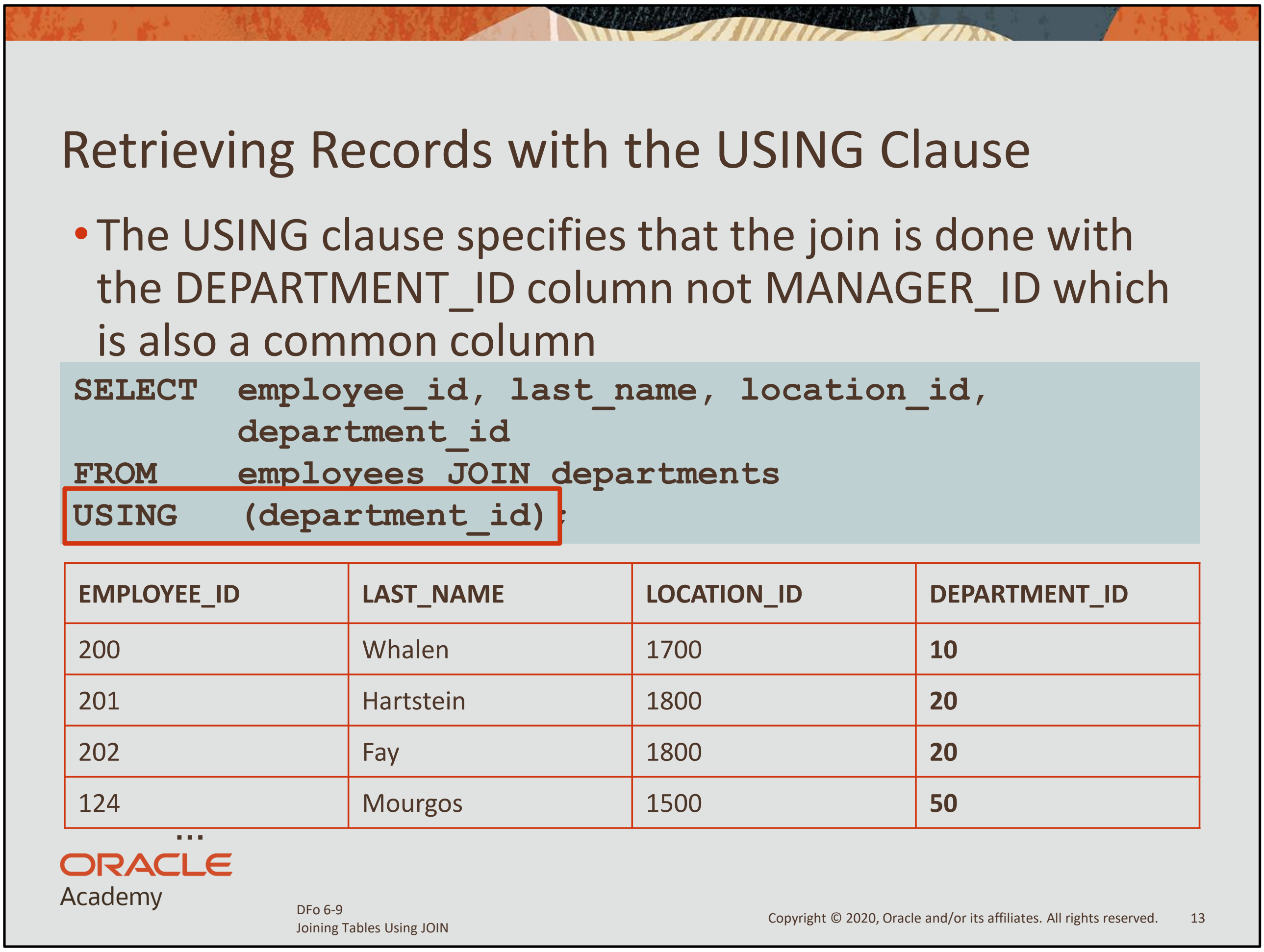
DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700

…

Foreign key

Primary key



Retrieving Records with the USING Clause

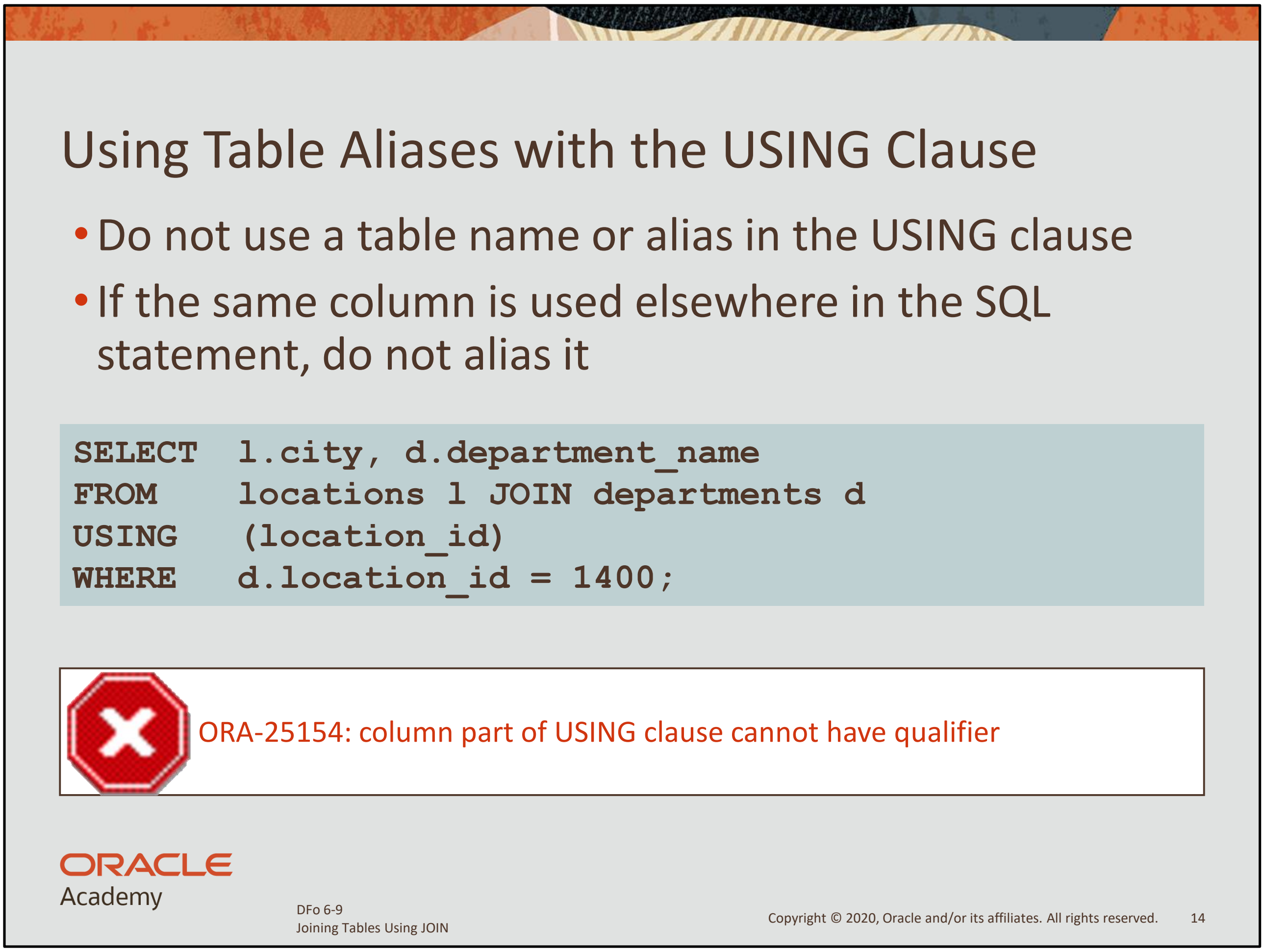
- The USING clause specifies that the join is done with the DEPARTMENT_ID column not MANAGER_ID which is also a common column

```
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**
124	Mourgos	1500	**50**

…

ORACLE
Academy



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;
```

☒ ORA-25154: column part of USING clause cannot have qualifier

ORACLE
Academy

For example, the following statement is valid:

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

The columns that are common in both the tables, but that are not used in the USING clause, must be prefixed with a table alias; otherwise, the "column ambiguously defined" error is returned.

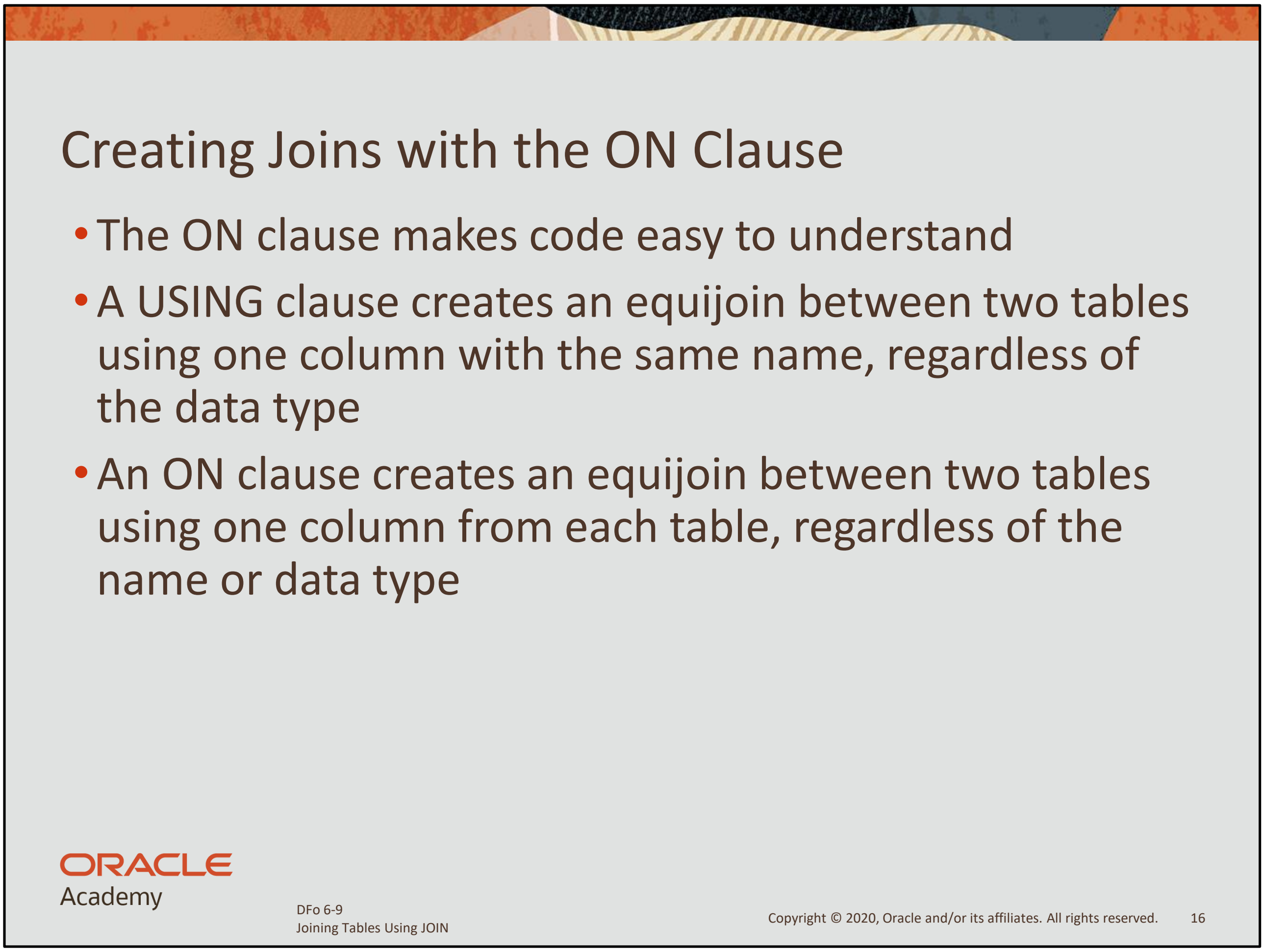
In the following statement, manager_id is in the EMPLOYEES and DEPARTMENTS tables; if manager_id is not prefixed with a table alias, a "column ambiguously defined" error is returned.

The following statement is valid:

```
SELECT first_name, d.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
WHERE  department_id = 50;
```

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

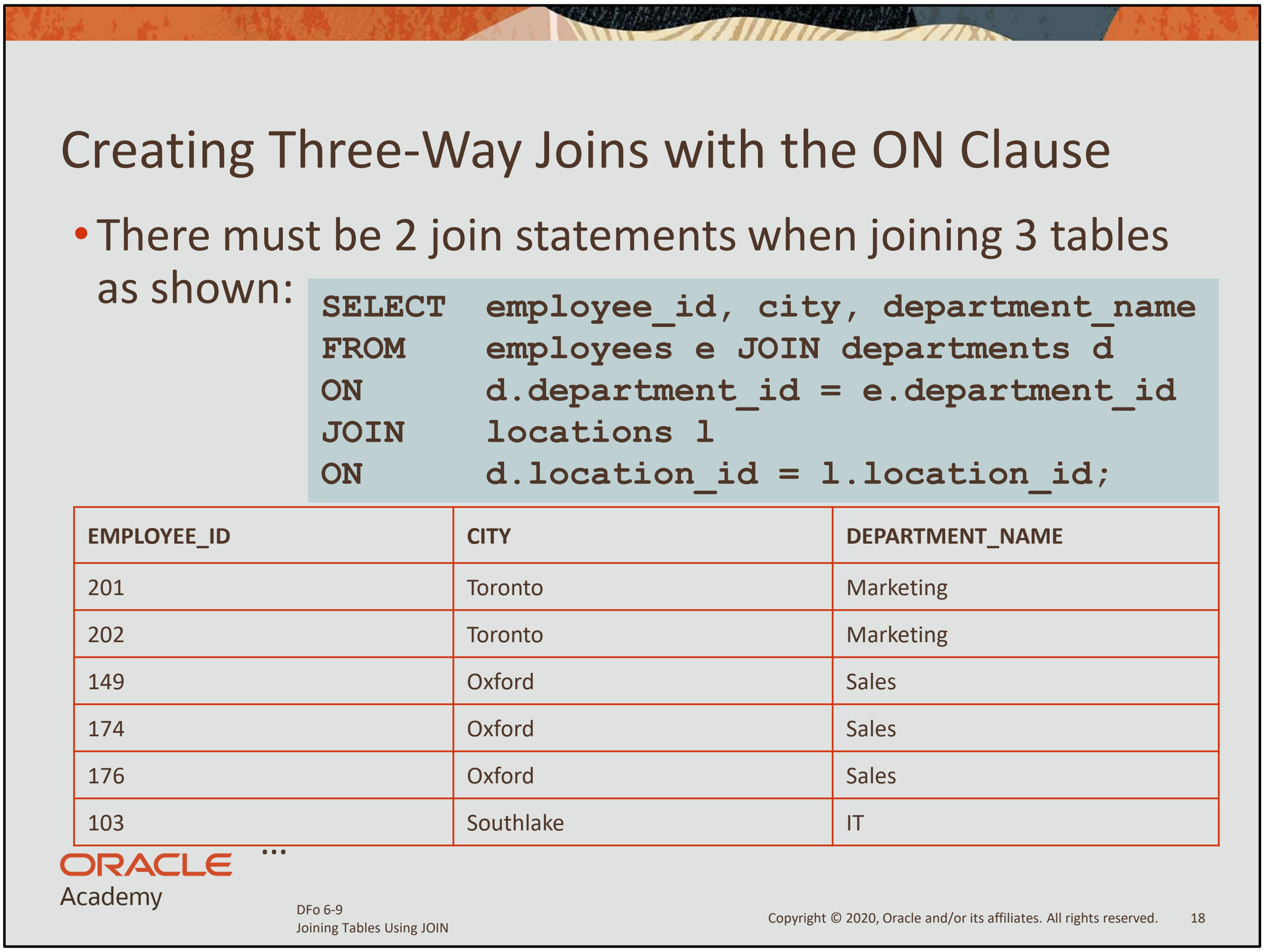
Retrieving Records with the ON Clause

- You can also use the ON clause to join columns that have different names or data types

```
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
124	Mourgos	**50**	**50**	1500
141	Rajs	**50**	**50**	1500
142	Davies	**50**	**50**	1500

ORACLE
Academy



Creating Three-Way Joins with the ON Clause

- There must be 2 join statements when joining 3 tables as shown:

```
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
201	Toronto	Marketing
202	Toronto	Marketing
149	Oxford	Sales
174	Oxford	Sales
176	Oxford	Sales
103	Southlake	IT

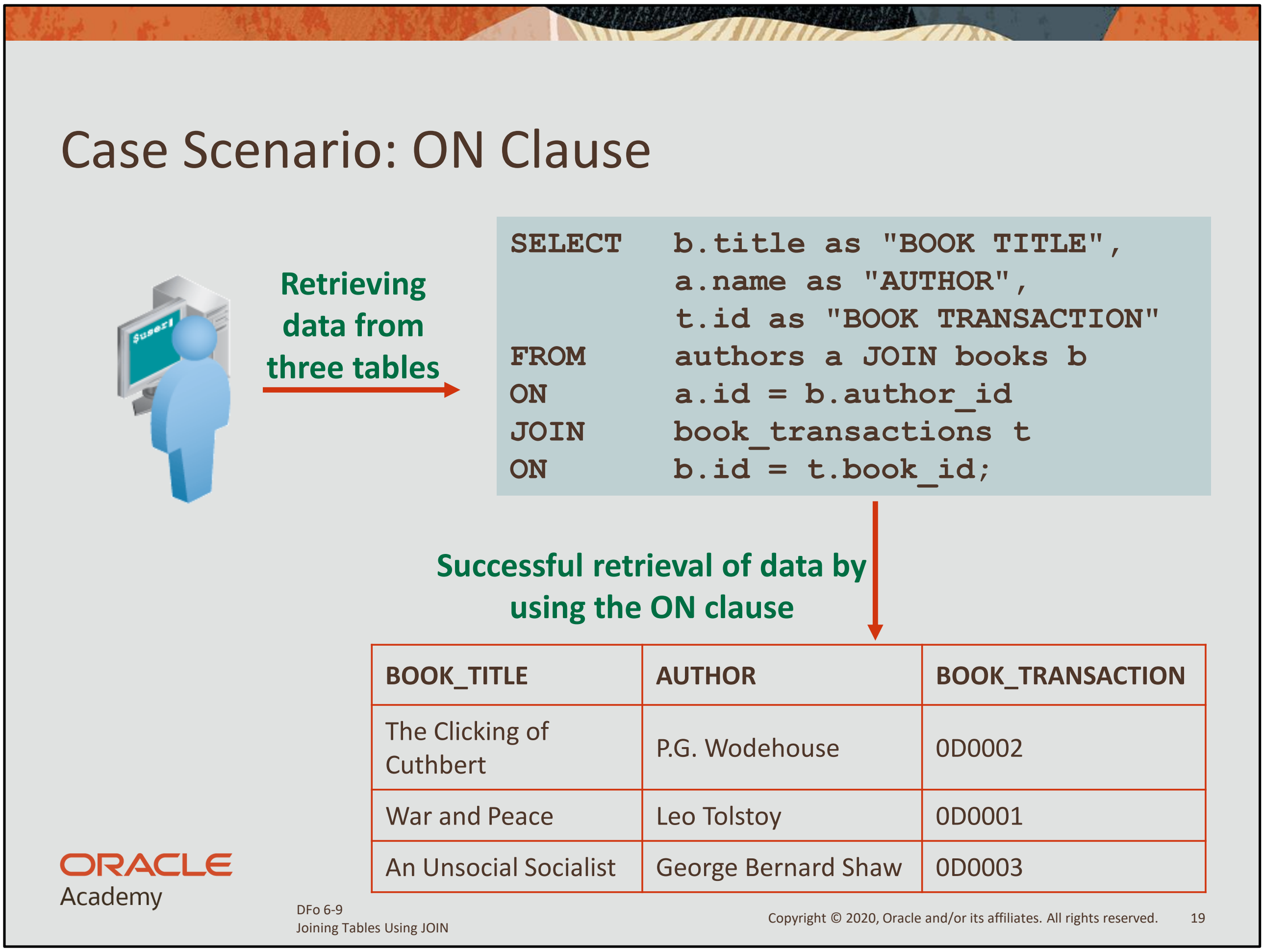
…

ORACLE
Academy

A three-way join is a join of three tables. The optimizer decides the execution of the join as well as the order. Here, the first join to be performed is EMPLOYEES JOIN DEPARTMENTS. The first join condition can reference columns in EMPLOYEES and DEPARTMENTS, but it cannot reference columns in LOCATIONS. The second join condition can reference columns from all three tables.

The code example in the slide can also be accomplished with the USING clause:

```
SELECT e.employee_id, l.city, d.department_name
FROM employees e
JOIN departments d
USING (department_id)
JOIN locations l
USING (location_id);
```



Case Scenario: ON Clause

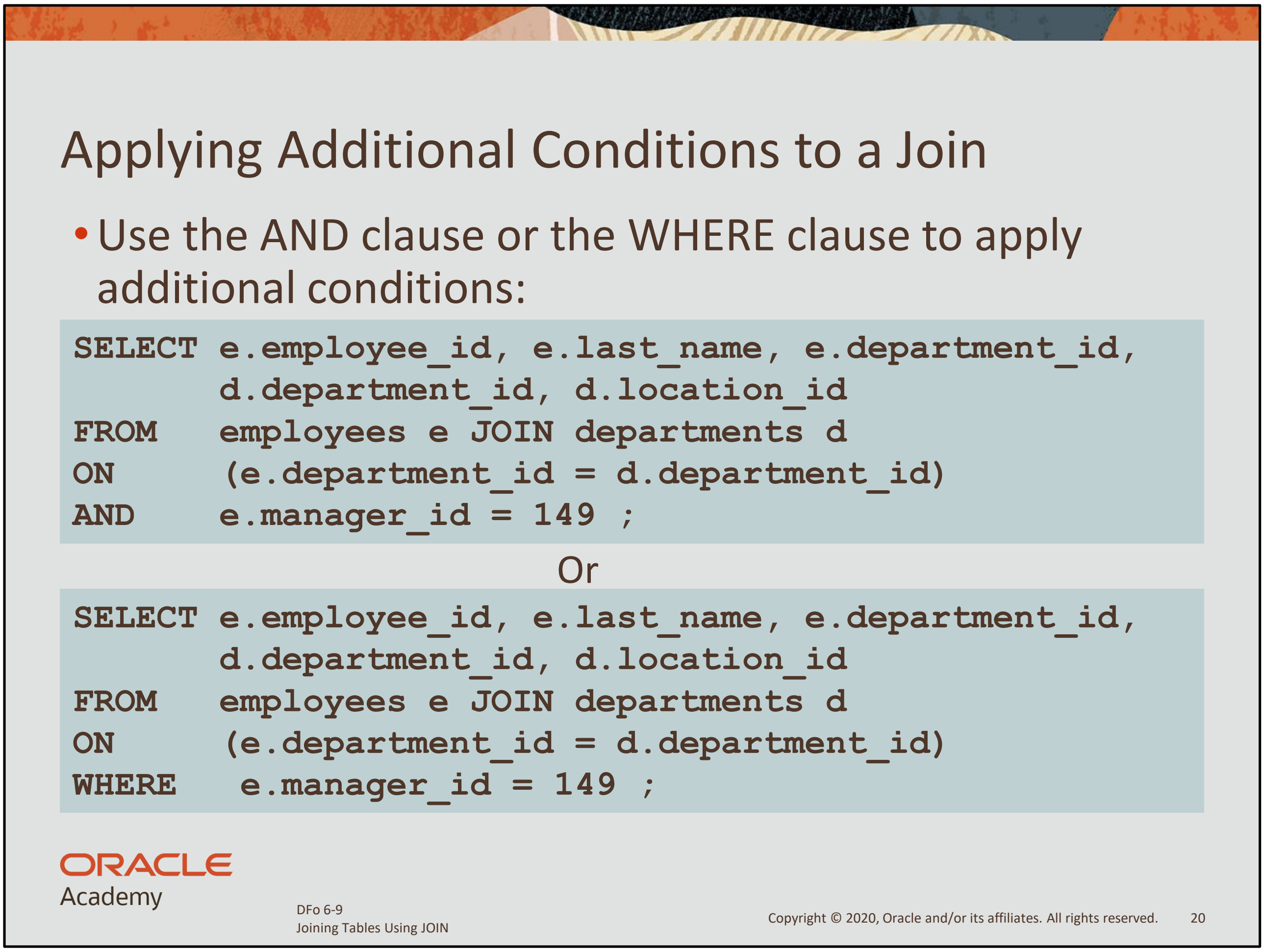
Retrieving data from three tables

```
SELECT   b.title as "BOOK TITLE",
         a.name as "AUTHOR",
         t.id as "BOOK TRANSACTION"
FROM     authors a JOIN books b
ON       a.id = b.author_id
JOIN     book_transactions t
ON       b.id = t.book_id;
```

Successful retrieval of data by using the ON clause

BOOK_TITLE	AUTHOR	BOOK_TRANSACTION
The Clicking of Cuthbert	P.G. Wodehouse	0D0002
War and Peace	Leo Tolstoy	0D0001
An Unsocial Socialist	George Bernard Shaw	0D0003

ORACLE
Academy



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 ;
```

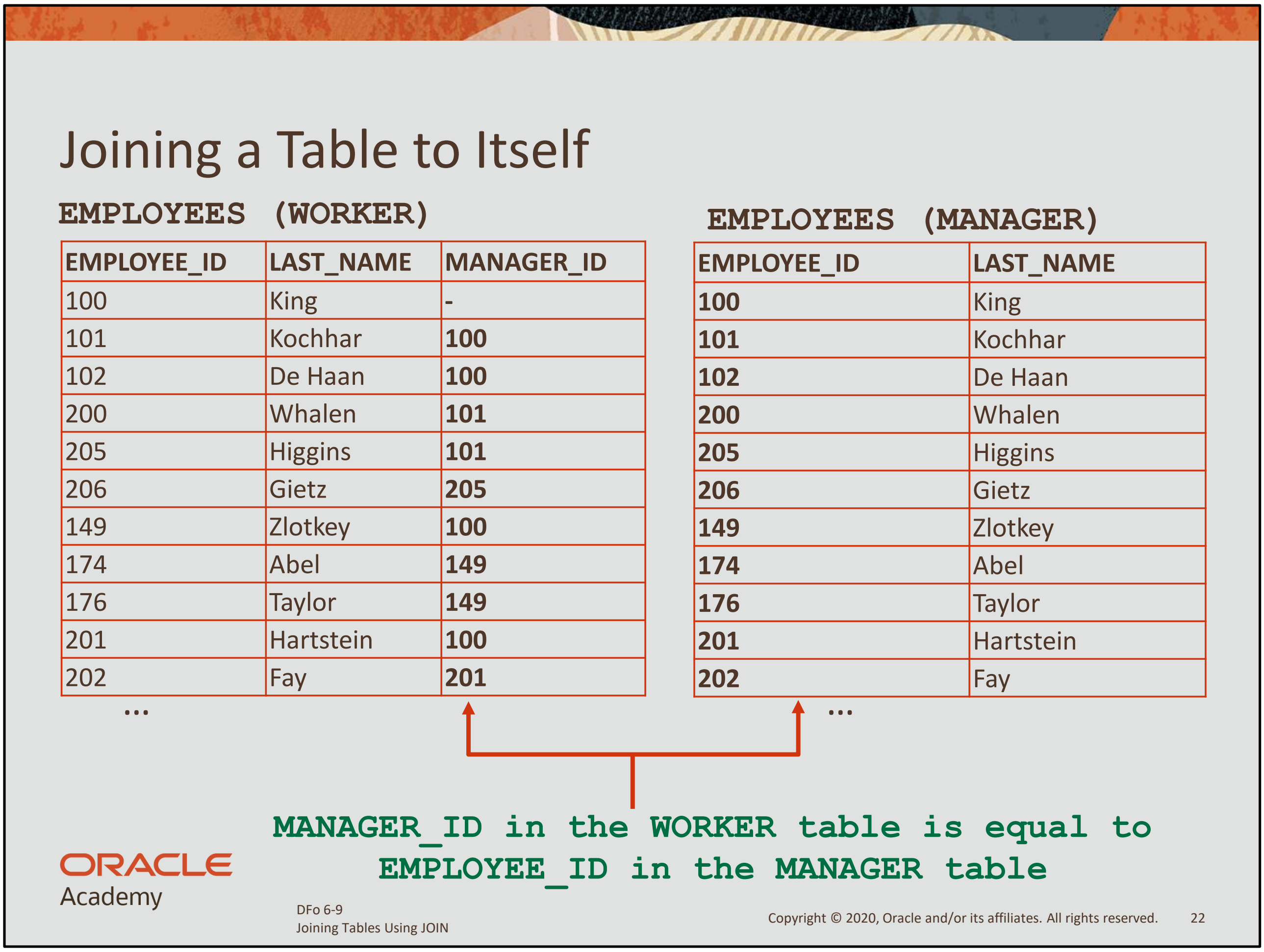
Or

```
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 ;
```

ORACLE
Academy

Project Exercise 1

- DFo_6_9_1_Project
  - Oracle Baseball League Store Database
  - Write SELECT Statements Using Data From Multiple Tables Using Equijoins and Non-Equijoins
  - - Natural Joins, USING and ON Clause, 3-way Joins



Joining a Table to Itself

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	-
101	Kochhar	**100**
102	De Haan	**100**
200	Whalen	**101**
205	Higgins	**101**
206	Gietz	**205**
149	Zlotkey	**100**
174	Abel	**149**
176	Taylor	**149**
201	Hartstein	**100**
202	Fay	**201**

…

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
200	Whalen
205	Higgins
206	Gietz
149	Zlotkey
174	Abel
176	Taylor
201	Hartstein
202	Fay

…

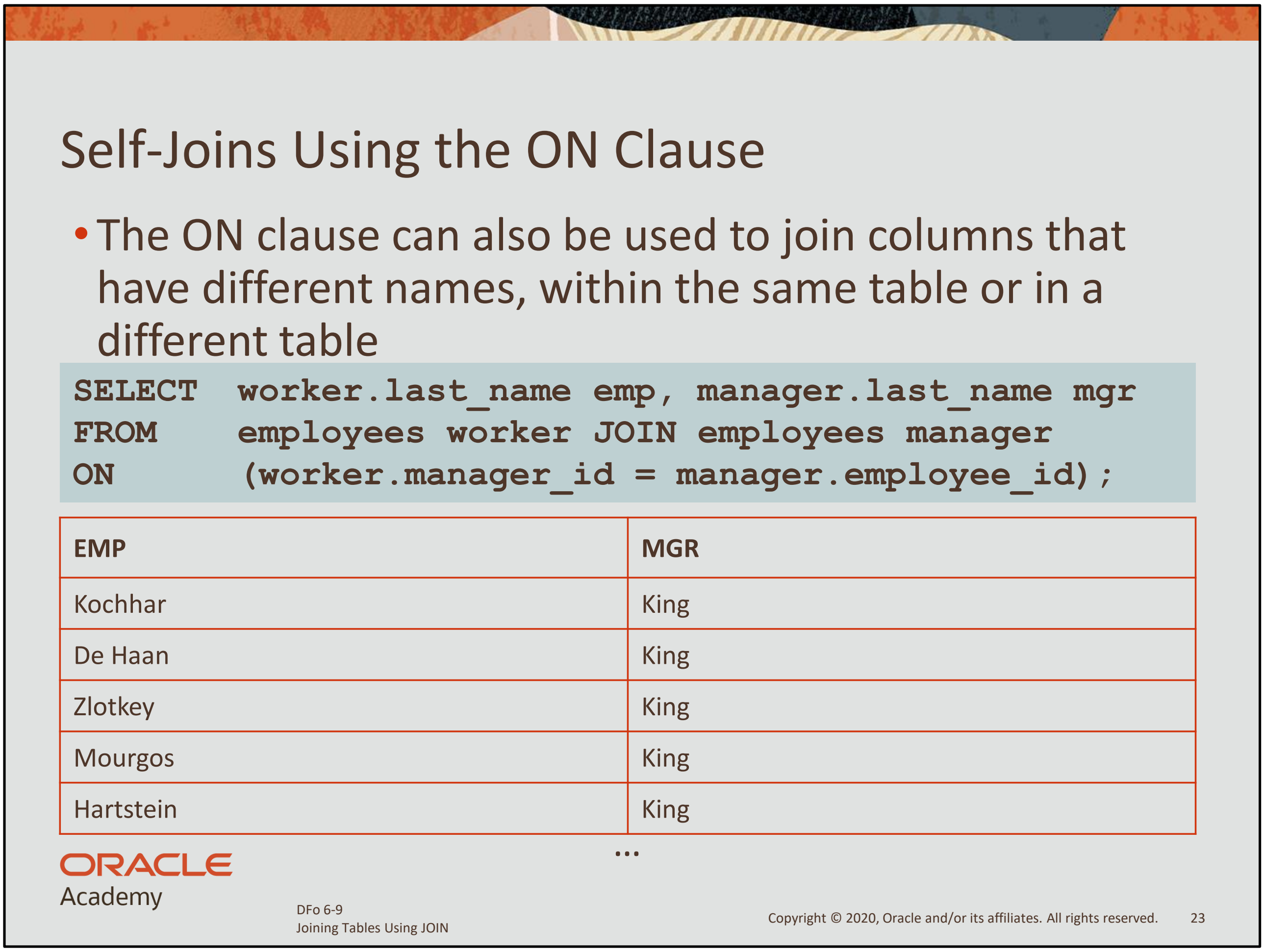
MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table

ORACLE
Academy

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to either join the EMPLOYEES table to itself or perform a self-join. For example, to find the name of Kochar’s manager:

- Find Kochar in the EMPLOYEES table by looking at the LAST_NAME column .
- Find the manager number for Kochar by looking at the MANAGER_ID column. Kochar’s manager number is 100.
- Find the name of the manager with EMPLOYEE_ID 100 by looking at the LAST_NAME column. King’s employee number is 100, so King is Kochar’s manager.

In this process, you look in the table twice. The first time you look in the table to find Kochar in the LAST_NAME column and the MANAGER_ID value of 100. The second time you look in the EMPLOYEE_ID column to find 100 and the LAST_NAME column to find King.



Self-Joins Using the ON Clause

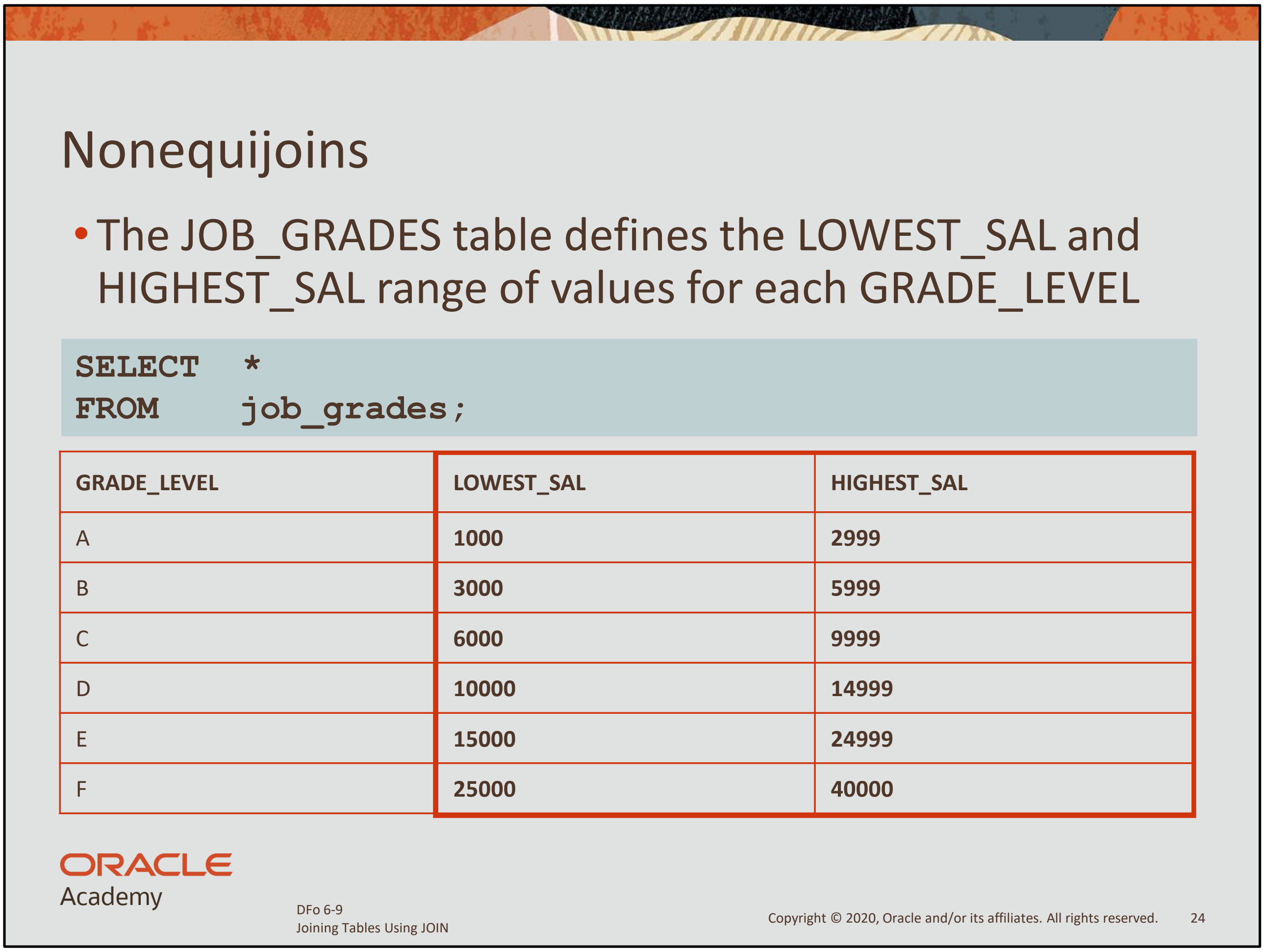
- The ON clause can also be used to join columns that have different names, within the same table or in a different table

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

EMP	MGR
Kochhar	King
De Haan	King
Zlotkey	King
Mourgos	King
Hartstein	King

…

ORACLE
Academy



Nonequijoins

- The JOB_GRADES table defines the LOWEST_SAL and HIGHEST_SAL range of values for each GRADE_LEVEL

```
SELECT   *
FROM     job_grades;
```

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

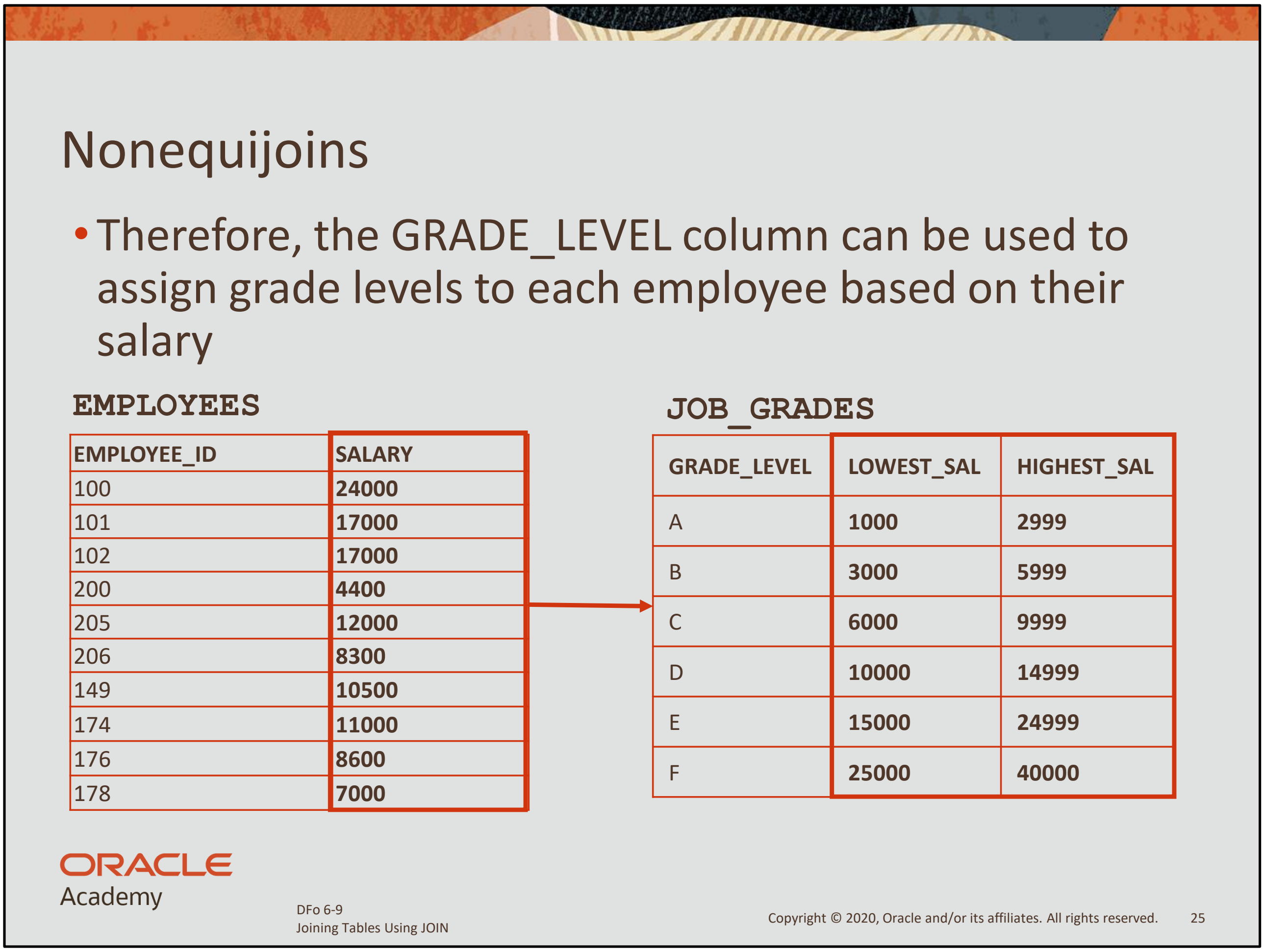
Academy

DFo 6-9
Joining Tables Using JOIN

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 SALARY column in the EMPLOYEES table ranges between the values in the LOWEST_SAL and HIGHEST_SAL columns of the JOB_GRADES table. Therefore, employees can be graded based on their salaries. The relationship is obtained by using an operator other than the equality (=) operator.



Nonequijoins

- Therefore, the GRADE_LEVEL column can be used to assign grade levels to each employee based on their salary

EMPLOYEES

EMPLOYEE_ID	SALARY
100	24000
101	17000
102	17000
200	4400
205	12000
206	8300
149	10500
174	11000
176	8600
178	7000

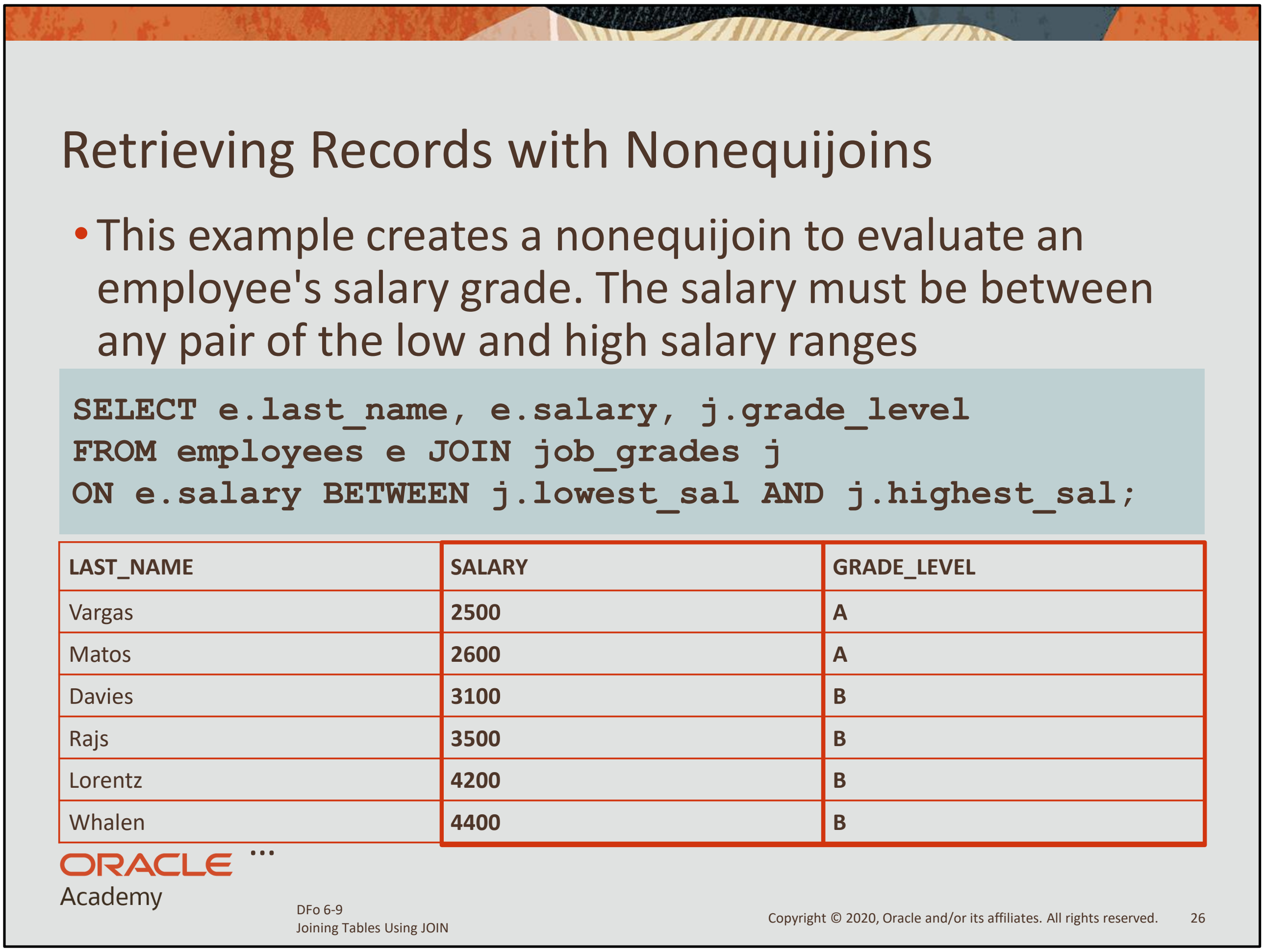
JOB_GRADES

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

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 SALARY column in the EMPLOYEES table ranges between the values in the LOWEST_SAL and HIGHEST_SAL columns of the JOB_GRADES table. Therefore, employees can be graded based on their salaries. The relationship is obtained by using an operator other than the equality (=) operator.



Retrieving Records with Nonequijoins

- This example creates a nonequijoin to evaluate an employee's salary grade. The salary must be between any pair of the low and high salary ranges

```
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
Vargas	2500	A
Matos	2600	A
Davies	3100	B
Rajs	3500	B
Lorentz	4200	B
Whalen	4400	B

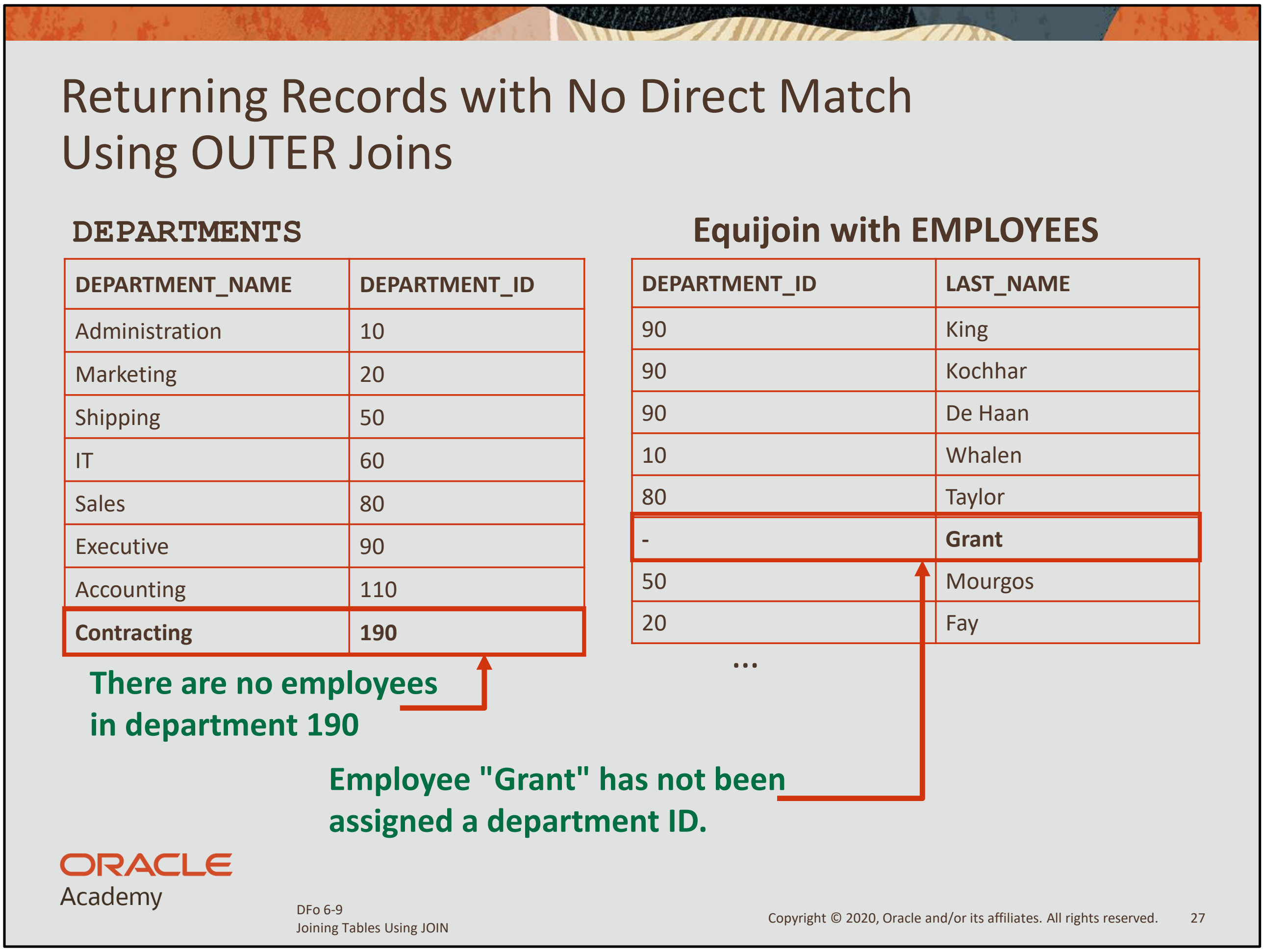
…

ORACLE
Academy

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list for the following reasons:

- None of the rows in the JOB_GRADES table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits provided by the job grade table. That is, no employee earns less than the lowest value contained in the LOWEST_SAL column or more than the highest value contained in the HIGHEST_SAL column.

Table aliases are specified in the slide example for performance reasons, not because of possible ambiguity.



Returning Records with No Direct Match Using OUTER Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	**190**

There are no employees in department 190

Equijoin with EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
10	Whalen
80	Taylor
-	**Grant**
50	Mourgos
20	Fay

…

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

DFo 6-9
Joining Tables Using JOIN

27

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

In the slide example, a simple equijoin condition is used on the EMPLOYEES and DEPARTMENTS tables to return the result on the right.

```
SELECT  employees.department_id, department_name, last_name
from employees, departments
where employees.department_id = departments.department_id;
```

The result set does not contain:

- Department ID 190, because there are no employees with that department ID recorded in the EMPLOYEES table
- The employee with the last name of Grant, because this employee has not been assigned a department ID

To return the department record that does not have any employees, or employees that do not have an assigned department, you can use an OUTER join.

INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an INNER join. (NATURAL JOIN, USING, ON clauses)
- 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 left and right OUTER join is a full OUTER join

LEFT OUTER JOIN

- Here we want to see all employee (left table) records even if they are not assigned to a department

```
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
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Vargas	50	Shipping
Matos	50	Shipping
Higgins	110	Accounting
Grant	-	-

…

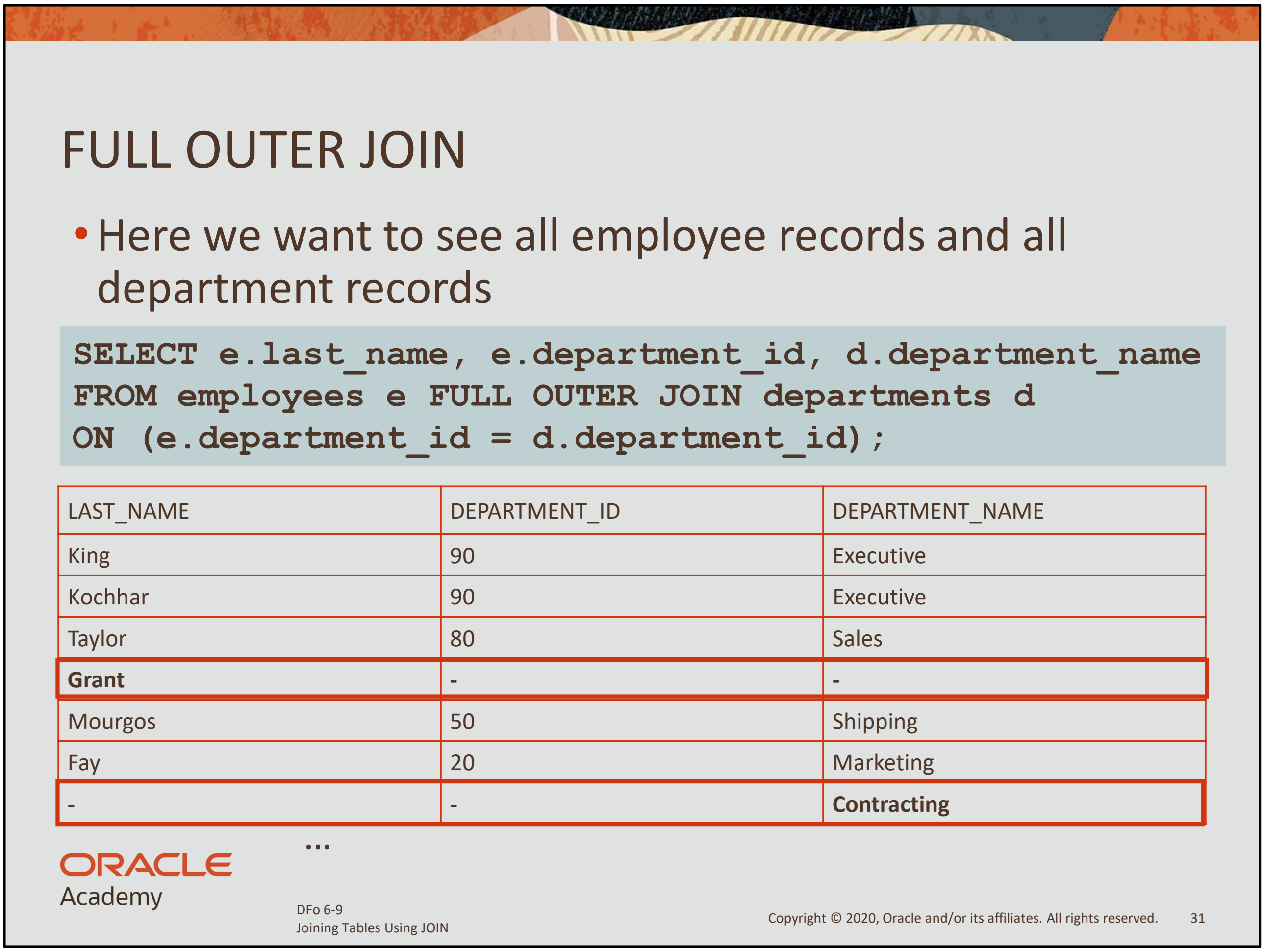
RIGHT OUTER JOIN

- Here we want to see all department (right table) records even if they have no employees in them

```
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);
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
-	-	**Contracting**

…



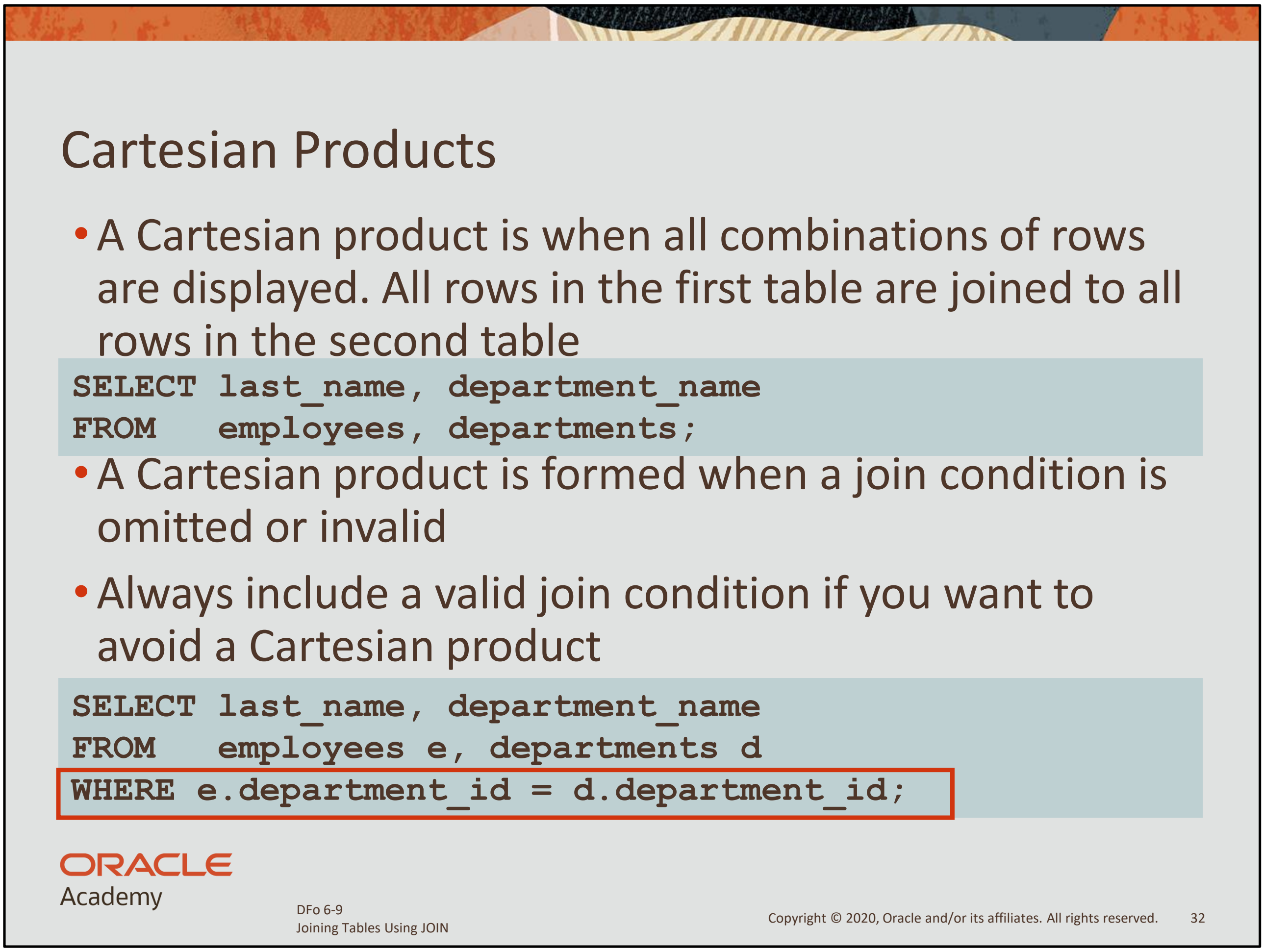
FULL OUTER JOIN

- Here we want to see all employee records and all department records

```
SELECT e.last_name, e.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
Taylor	80	Sales
Grant	-	-
Mourgos	50	Shipping
Fay	20	Marketing
-	-	**Contracting**

…



Cartesian Products

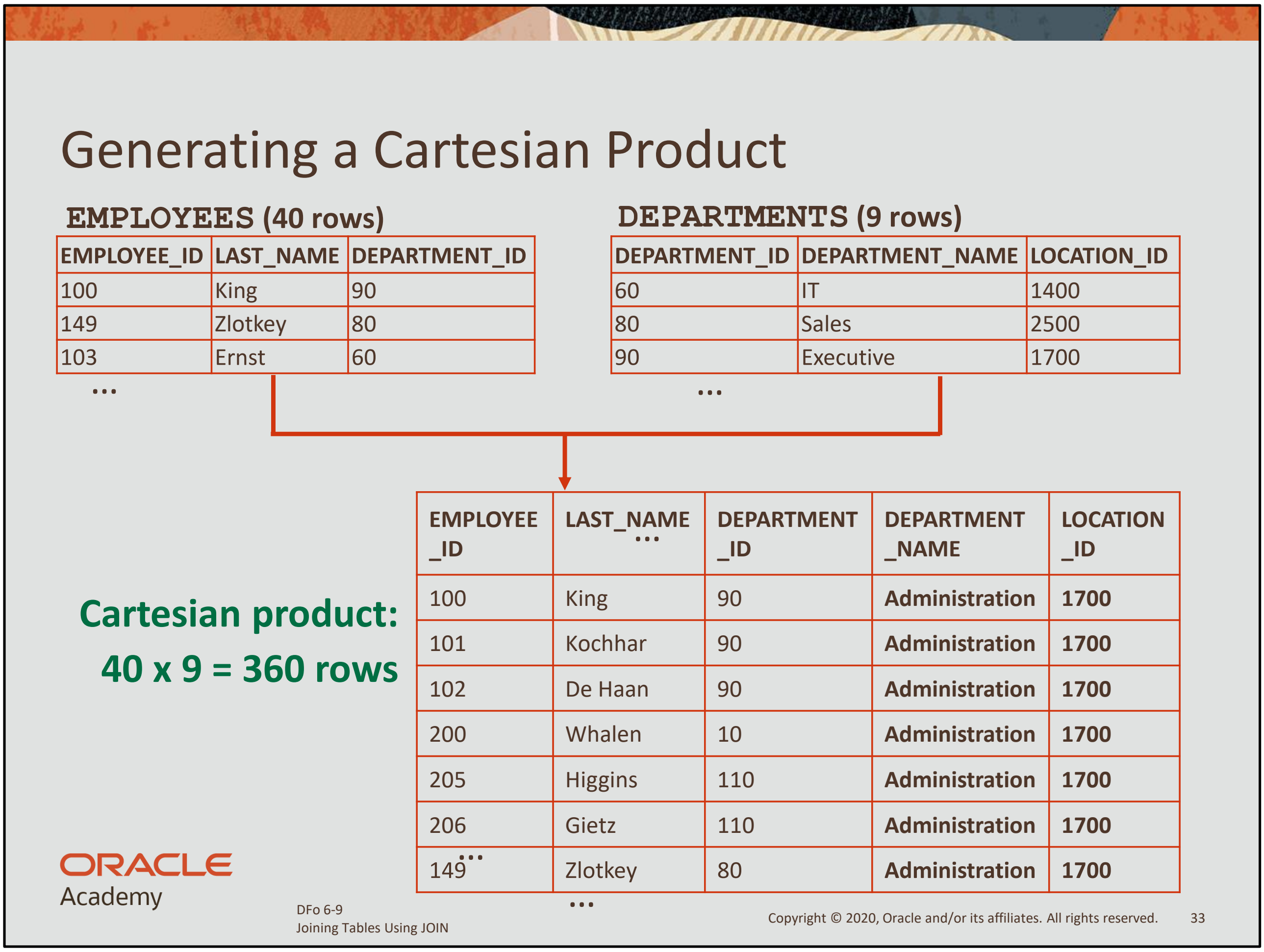
- A Cartesian product is when all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table

```
SELECT last_name, department_name
FROM   employees, departments;
```

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

```
SELECT last_name, department_name
FROM   employees e, departments d
WHERE e.department_id = d.department_id;
```

A Cartesian product tends to generate a large number of rows, and the result is rarely useful except for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.



Generating a Cartesian Product

EMPLOYEES (40 rows)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
149	Zlotkey	80
103	Ernst	60

...

DEPARTMENTS (9 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
60	IT	1400
80	Sales	2500
90	Executive	1700

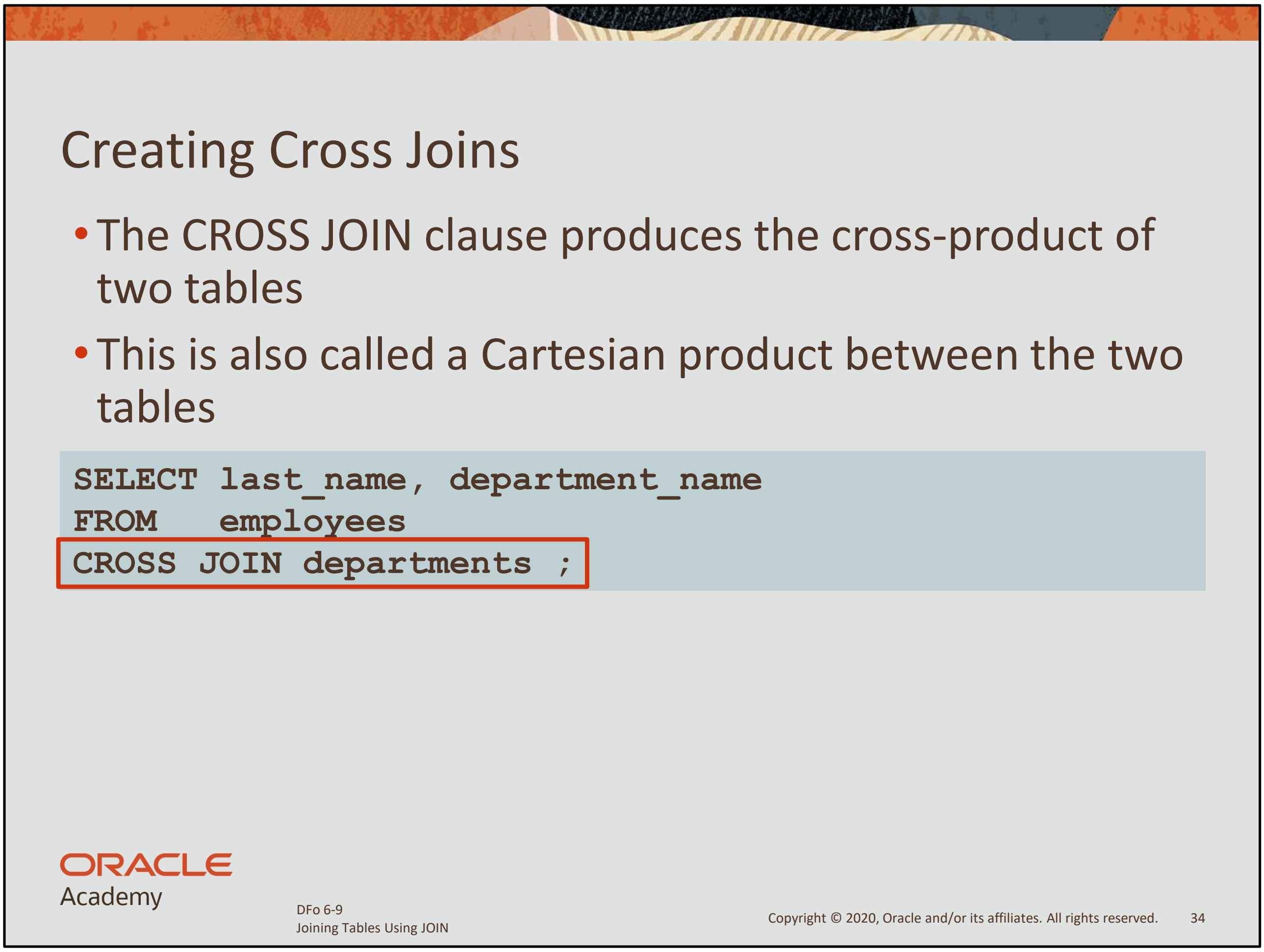
...

Cartesian product:

40 x 9 = 360 rows

EMPLOYEE_ID	LAST_NAME ...	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
100	King	90	**Administration**	**1700**
101	Kochhar	90	**Administration**	**1700**
102	De Haan	90	**Administration**	**1700**
200	Whalen	10	**Administration**	**1700**
205	Higgins	110	**Administration**	**1700**
206	Gietz	110	**Administration**	**1700**
149...	Zlotkey	80	**Administration**	**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 ;
```

ORACLE
Academy

The CROSS JOIN technique can be usefully applied to some situations. For example, to return total labor cost by office by month, even if month X has no labor cost, you can do a cross join of Offices with a table of all Months.

It is a good practice to explicitly state CROSS JOIN in your SELECT when you intend to create a Cartesian product. Therefore, it is very clear that you intend for this to happen and it is not the result of missing joins.

Project Exercise 2

- DFo_6_9_2_Project
  - Oracle Baseball League Store Database
  - Write SELECT Statements Using Data From Multiple Tables Using Equijoins and Non-Equijoins
  - - Self Joins, OUTER JOINs, Cartesian Products

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 to 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

ORACLE

Academy