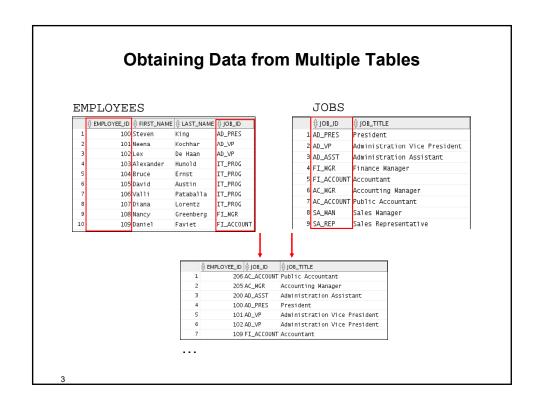
Displaying Data from Multiple Tables Using Joins

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



Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- 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 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];
```

5

Creating Natural Joins

- The NATURAL JOIN clause is based on all the columns that have the same name in two tables.
- 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.

```
SELECT * FROM table1 NATURAL JOIN table2;
```

Retrieving Records with Natural Joins

SELECT employee_id, first_name, job_id, job_title from employees NATURAL JOIN jobs;

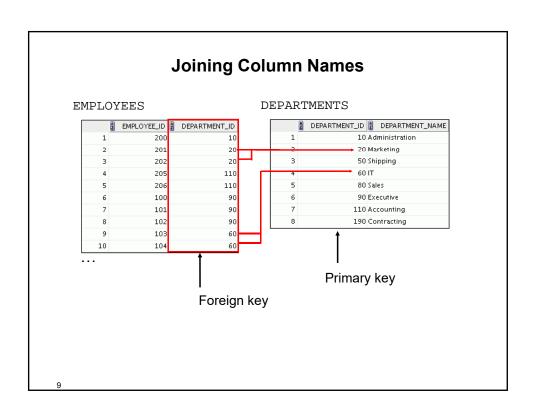
		∮ FIRST_NAME	∮ JOB_ID	∯ JOB_TITLE
1	100	Steven	AD_PRES	President
2	101	Neena	AD_VP	Administration Vice President
3	102	Lex	AD_VP	Administration Vice President
4	103	Alexander	IT_PR0G	Programmer
5	104	Bruce	IT_PROG	Programmer
6	105	David	IT_PROG	Programmer
7	106	Valli	IT_PROG	Programmer
8	107	Diana	IT_PROG	Programmer
9	108	Nancy	FI_MGR	Finance Manager
10	109	Dani e1	FI_ACCOUNT	Accountant
11	110	John	FI_ACCOUNT	Accountant

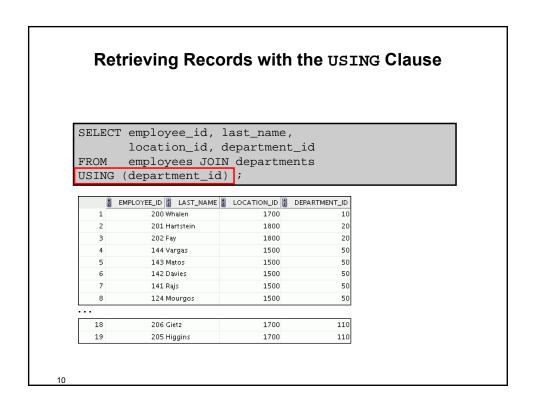
...

7

Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, use the USING clause to specify the columns for the equijoin.
- Use the USING clause to match only one column when more than one column matches.





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 of 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.

11

Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the NATURAL join or a join with a USING clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT 1.city, d.department_name
FROM locations 1 JOIN departments d
USING (location_id)
WHERE d.location_id = 1400;
```

ORA-25154: column part of USING clause cannot have qualifier 25154. 00000 - "column part of USING clause cannot have qualifier" "Cause: Columns that are used for a named-join (either a NATURAL join or a join with a USING clause) cannot have an explicit qualifier. *Action: Remove the qualifier. Error at Line: 4 Column: 6

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.

13

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_1	location_id
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400

• • •

Creating Three-Way Joins

```
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;
```

	A	EMPLOYEE_ID	A	CITY	A	DEPARTMENT_NAME
1		100	Sea	ittle	Exe	cutive
2		101	Sea	ittle	Ехе	ecutive
3		102	Sea	ittle	Exe	ecutive
4		103	Soi	uthlake	ΙT	
5		104	Soi	uthlake	ΙT	
6		107	Soi	uthlake	ΙT	
7		124	Soi	uth San Francisco	Shi	pping
8		141	Soi	uth San Francisco	Shi	pping
9		142	Soi	uth San Francisco	Shi	pping

• • •

15

Applying Additional Conditions to a Join

Use the ${\tt AND}$ clause or the ${\tt WHERE}$ 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)

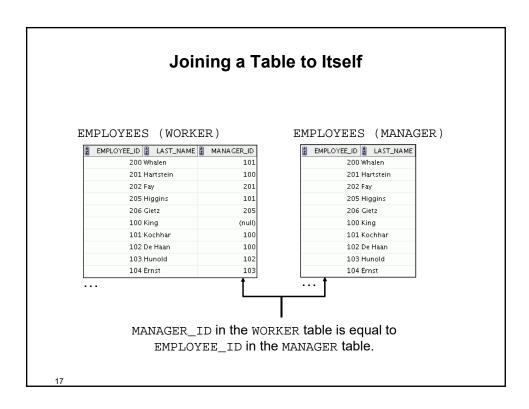
AND e.manager_id = 149;
```

Or

```
SELECT e.employee_id, e.last_name, e.department_id, d.department_id, d.location_id

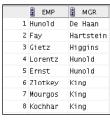
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)

WHERE e.manager_id = 149;
```

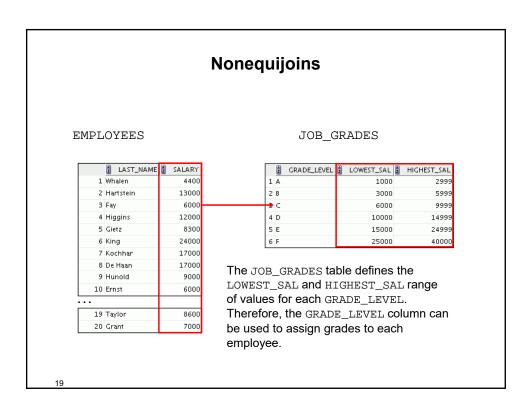


Self-Joins Using the ON Clause

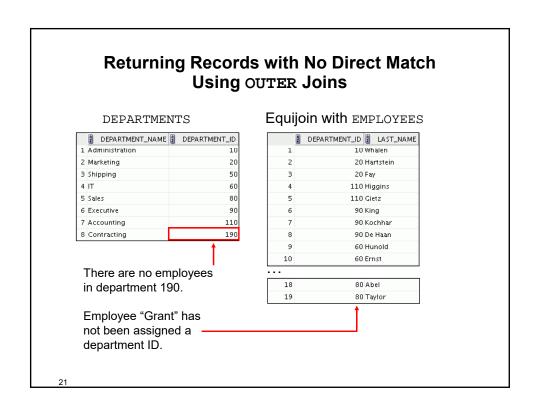
SELECT worker.last_name emp, manager.last_name mgr
FROM employees worker JOIN employees manager
ON (worker.manager_id = manager.employee_id);

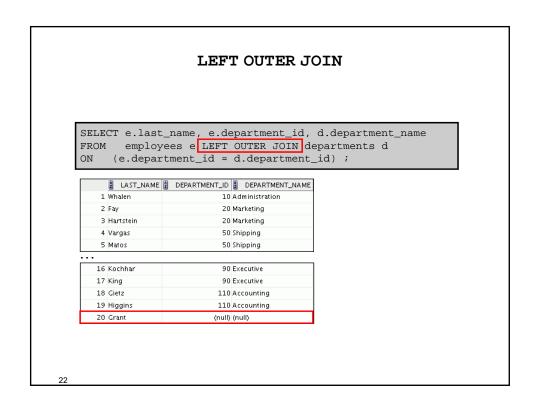


. . .



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 2600 A 2 Matos 3 Davies 3100 B 3500 B 4 Rajs 4200 B 5 Lorentz 6 Whalen 4400 B 7 Mourgos 5800 B 8 Ernst 6000 C 9 Fay 6000 C 10 Grant 7000 C





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_ij);

	LAST_NAME	2 DEPARTMENT_ID 2 DEPARTMENT_NAME
1	Whalen	10 Administration
2	Hartstein	20 Marketing
3	Fay	20 Marketing
4	Davies	50 Shipping
5	Vargas	50 Shipping
6	Rajs	50 Shipping
- 7	Mourgos	50 Shipping
8	Matos	50 Shipping

18 Higgins	110 Accounting
19 Gietz	110 Accounting
20 (null)	190 Contracting

23

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);

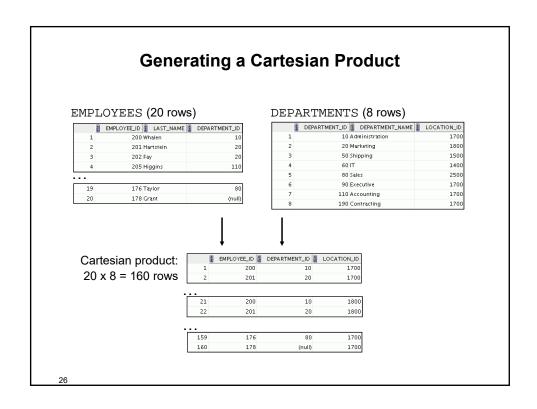
	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Huno1d	60	IT

· . . .

15	Grant	(null)	(null)
16	Wha1en	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

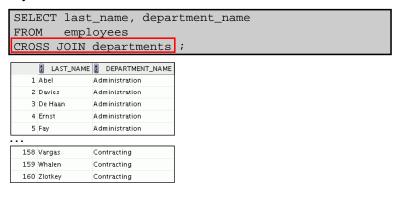
Cartesian Products

- Cartesian product is a join of every row of one table to every row of another table.
- A Cartesian product generates a large number of rows and the result is rarely useful.



Creating Cross Joins

- A CROSS JOIN is a JOIN operation that produces the Cartesian product of two tables.
- To create a Cartesian product, specify the CROSS JOIN in your SELECT statement.



Summary

In this lesson, you should have learned how to:

- 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