DISPLAYING DATA FROM MULTIPLE TABLES

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 in a `WHERE` clause.**

A Cartesian product is generated if a join condition is omitted. The example in the slide displays employee last name and department name from the `EMPLOYEES` and `DEPARTMENTS` tables. Because no `WHERE` clause has been specified, all rows (20 rows) from the `EMPLOYEES` table are joined with all rows (8 rows) in the `DEPARTMENTS` table, thereby generating 160 rows in the output.

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

LAST_NAME	DEPT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
...	...
Gietz	Contracting

160 rows selected.

Types of Joins

Oracle Proprietary Joins (8*i* and prior):

- Equijoin
- Nonequijoin
- Outer join
- Self join

SQL: 1999 Compliant Joins:

- Cross joins
- Natural joins
- Using clause
- Full or two sided outer joins
- Arbitrary join conditions for outer joins

Joining Tables Using Oracle Syntax

Use a join to query data from more than one table.

```
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column1 = table2.column2;
```

- **Write the join condition in the `WHERE` clause.**
- **Prefix the column name with the table name when the same column name appears in more than one table.**

Guidelines

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join *n* tables together, you need a minimum of *n*-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.

For more information, see *Oracle9i SQL Reference,* "SELECT."

What Is an Equijoin?

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
205	110
206	110

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
	Accounting
110	Accounting

19 rows selected.

Retrieving Records with Equijoins

```
SELECT employees.employee_id, employees.last_name,
       employees.department_id, departments.department_id,
       departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.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
205	Higgins	110	110	
206	Gietz	110	110	1700

19 rows selected.

Additional Search Conditions Using the AND Operator

In addition to the join, you may have criteria for your WHERE clause to restrict the rows under consideration for one or more tables in the join. For example, to display employee Matos' department number and department name, you need an additional condition in the WHERE clause.

```
SELECT last_name, employees.department_id,
       department_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id
AND    last_name = 'Matos';
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Matos	50	Shipping

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

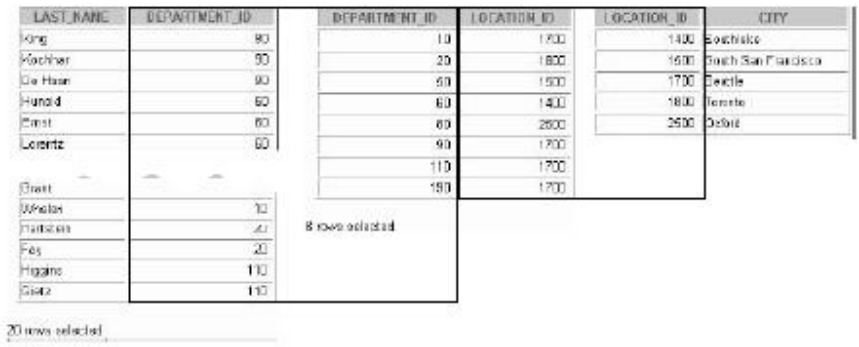
Using Table Aliases

- Simplify queries by using table aliases
- Improve performance by using table prefixes

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

Joining More than Two Tables

EMPLOYEES DEPARTMENTS LOCATIONS



To join *n* tables together, you need a minimum of *n*-1 join conditions. For example, to join three tables, a minimum of two joins is required.

```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

LAST_NAME	DEPARTMENT_NAME	CITY
Hunold	IT	Southlake
Ernst	IT	Southlake
Lorentz	IT	Southlake
Mourgos	Shipping	South San Francisco
Rajs	Shipping	South San Francisco

Nonequijoins

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
...	...
Higgins	12000
Gietz	8300

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

6 rows selected.

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

A nonequijoin is a join condition containing something other than an equality operator.

The relationship between the EMPLOYEES table and the JOB_GRADES table has an example of a nonequijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST_SALARY and HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equals (=).

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  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
...	...	...
Kochhar	17000	E
De Haan	17000	E

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

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

- None of the rows in the job grade 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.

Note: Other conditions, such as <= and >= can be used, but `BETWEEN` is the simplest. Remember to specify the low value first and the high value last when using `BETWEEN`.

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

Outer Joins

DEPARTMENTS

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

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
10	W...en
20	Hartstein
20	Fay
110	Higgins
110	Gietz

20 rows selected.

There are no employees in department 190.

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row will not appear in the query result. For example, in the equijoin condition of EMPLOYEES and DEPARTMENTS tables, employee Grant does not appear because there is no department ID recorded for her in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
,.iggins	110	Accounting
Gietz	110	Accounting

19 rows selected.

Outer Joins Syntax

- You use an outer join to also see rows that do not meet the join condition.
- The outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column(+) = table2.column;
```

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column = table2.column(+);
```

Using Outer Joins to Return Records with No Direct Match

The missing rows can be returned if an outer join operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the side of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more rows from the nondeficient table can be joined.

In the syntax:

table1.column =	is the condition that joins (or relates) the tables together.
table2.column (+)	is the outer join symbol, which can be placed on either side of the WHERE clause condition, but not on both sides. (Place the outer join symbol following the name of the column in the table without the matching rows.)

Using Outer Joins

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE 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
Higgins	110	[illegible]
Gietz	110	Accounting
		Contracting

20 rows selected.

Outer Join Restrictions

- The outer join operator can appear on only *one* side of the expression: the side that has information missing. It returns those rows from one table that have no direct match in the other table.
- A condition involving an outer join cannot use the `IN` operator or be linked to another condition by the `OR` operator.

Self Joins

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
101	Kochhar	100
102	De Haan	100
124	Mourgos	100
149	Zlotkey	100
201	Hartstein	100
200	Whalen	101
206	Gietz	205

19 rows selected.

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
100	King
100	King
100	King
100	King
101	Kochhar
205	Higgins

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

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

- Find Whalen in the `EMPLOYEES` table by looking at the `LAST_NAME` column.
- Find the manager number for Whalen by looking at the `MANAGER_ID` column. Whalen's manager number is 101.
- Find the name of the manager with `EMPLOYEE_ID` 101 by looking at the `LAST_NAME` column. Kochhar's employee number is 101, so Kochhar is Whalen's manager.

In this process, you look in the table twice. The first time you look in the table to find Whalen in the `LAST_NAME` column and `MANAGER_ID` value of 101. The second time you look in the `EMPLOYEE_ID` column to find 101 and the `LAST_NAME` column to find Kochhar.

Joining a Table to Itself

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id;
```

W.LAST_NAME||'WORKSFOR'||M.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
[illegible]
Ray works for Hartstein
Gietz works for Higgins

19 rows selected.

Joining Tables Using SQL: 1999 Syntax

Use a join to query data from more than one table.

```
SELECT   table1.column, table2.column
FROM     table1
[CROSS JOIN table2] |
[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)];
```

Defining Joins

Using the SQL: 1999 syntax, you can obtain the same results as what was shown in the prior pages.

In the syntax:

table1.column	Denotes the table and column from which data is retrieved
CROSS JOIN	Returns a Cartesian product from the two tables
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* = *table2.column_name*	Performs an equijoin based on the condition in the ON clause
LEFT/RIGHT/FULL OUTER	

Creating Cross Joins

- The `CROSS JOIN` clause produces the cross-product of two tables.
- This is the same as a Cartesian product between the two tables.

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

LAST_NAME	DEPARTMENT_NAME
Fay	Contracting
Higgins	Contracting
Gietz	Contracting

160 rows selected.

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.
- 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, then an error is returned.

Note: The join can happen only on columns having the same names and data types in both the tables. If the columns have the same name, but different data types, then the `NATURAL JOIN` syntax causes an error.

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
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

Equijoins

The natural join can also be written as an equijoin:

```
SELECT department_id, department_name,
       departments.location_id, city
FROM   departments, locations
WHERE  departments.location_id = locations.location_id;
```

Natural Joins with a `WHERE` Clause

Additional restrictions on a natural join are implemented by using a `WHERE` clause. The example below 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);
```

Creating Joins with the `USING` Clause

- **If several columns have the same names but the data types do not match, the `NATURAL JOIN` clause can be modified with the `USING` clause to specify the columns that should be used for an equijoin.**
  Note: 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.**

Natural joins use all columns with matching names and data types to join the tables. The `USING` clause can be used to specify only those columns that should be used for an equijoin. The columns referenced in the `USING` clause should not have a qualifier (table name or alias) anywhere in the SQL statement.

For example, this statement is valid:

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

This 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;
ORA-25154: column part of USING clause cannot have qualifier
```

The same restriction applies to `NATURAL` joins also. Therefore columns that have the same name in both tables have to be used without any qualifiers.

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

EMPLOYEE_ID	LAST_NAME	LOCATION_ID
200	Whalen	1700
201	Hartstein	1800
202	Fay	1800
124	Mourgos	1500
141	Rajs	1500
142	Davies	1500
205	Higgins	1700
206	Gietz	1700

The example shown joins the DEPARTMENT_ID column in the EMPLOYEES and DEPARTMENTS tables, and thus shows the location where an employee works.

This can also be written as an equijoin:

```
SELECT employee_id, last_name,
       employees.department_id, location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- To specify arbitrary conditions or specify columns to join, the ON clause is used.
- Separates the join condition from other search conditions.
- The ON clause makes code easy to understand.

```
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
205	Higgins	110	110	1700
206	Gietz	110	110	1700

19 rows selected

The ON clause can also be used as follows to join columns that have different names:

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

EMP	MGR
Kochhar	King
De Haan	King
Mourgos	King
Gietz	Higgins

19 rows selected.

The preceding example is a self join of the EMPLOYEE table to itself, based on the EMPLOYEE_ID and MANAGER_ID columns.

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
107	Southlake	IT
124	South San Francisco	Shipping
206	Seattle	

19 rows selected

Three-Way Joins

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.

This can also be written as a three-way equijoin:

```
SELECT employee_id, city, department_name
FROM   employees, departments, locations
WHERE  employees.department_id = departments.department_id
AND    departments.location_id = locations.location_id;
```

INNER versus OUTER Joins

- In SQL: 1999, the join of two tables returning only matched rows is an inner join.
- A join between two tables that returns the results of the inner join as well as unmatched rows left (or right) tables is 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.

Joins: Comparing SQL: 1999 to Oracle Syntax

Oracle	SQL: 1999
Equijoin	Natural or Inner Join
Outerjoin	Left Outer Join
Selfjoin	Join ON
Nonequijoin	Join USING
Cartesian Product	Cross 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
...	...	...
Ernst	60	IT
Grant		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting

20 rows selected.

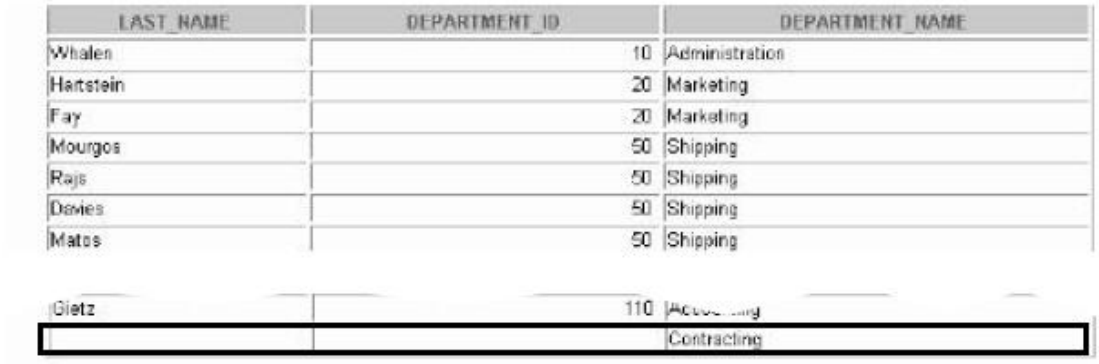
This query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no match in the DEPARTMENTS table.

This query was completed in earlier releases as follows:

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

RIGHT OUTER JOIN

```
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
Matos	50	Shipping
Gietz	110	Accounting
		Contracting

20 rows selected.

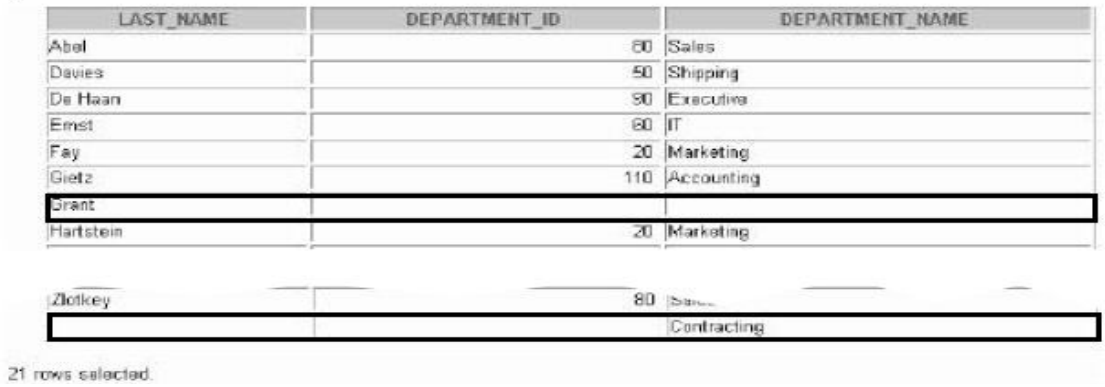
This query retrieves all rows in the DEPARTMENTS table, which is the right table even if there is no match in the EMPLOYEES table.

This query was completed in earlier releases as follows:

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

FULL OUTER JOIN

```
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
Abel	80	Sales
Davies	50	Shipping
De Haan	90	Executive
Ernst	60	IT
Fay	20	Marketing
Gietz	110	Accounting
Grant		
Hartstein	20	Marketing
Zlotkey	80	Sales
		Contracting

21 rows selected.

This 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.

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;
```

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

You can apply additional conditions in the `WHERE` clause. The example shown performs a join on the `EMPLOYEES` and `DEPARTMENTS` tables, and, in addition, displays only employees with a manager ID equal to 149.

ASSIGNMENTS

1. Write a query to display the last name, department number, and department name for all employees.

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
Vargas	50	Shipping
Hunold	60	IT
...ggins	110	Ac...
Gietz	110	Accounting

19 rows selected.

2. Create a unique listing of all jobs that are in department 30. Include the location of department 90 in the output.

JOB_ID	LOCATION_ID
SA_MAN	2500
SA_REP	2500

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

LAST_NAME	DEPARTMENT_NAME	LOCATION_ID	CITY
Zlotkey	Sales	2500	Oxford
Abel	Sales	2500	Oxford
Taylor	Sales	2500	Oxford

4. Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names. Place your SQL statement in a text file named `lab4_4.sql`.

LAST_NAME	DEPARTMENT_NAME
Whalen	Administration
Hartstein	Marketing
Fay	Marketing
Rajs	Shipping
Davies	Shipping
Matos	Shipping
Vargas	Shipping
Taylor	Sales
Kochhar	Executive
De Haan	Executive

10 rows selected.

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
Hartstein	MK_MAN	20	Marketing
Fay	MK_REP	20	Marketing

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns `Employee`, `Emp#`, `Manager`, and `Mgr#`, respectively. Place your SQL statement in a text file named `lab4_6.sql`.

Employee	EMP#	Manager	Mgr#
Kochhar	101	King	100
De Haan	102	King	100
Mourgos	124	King	100
Zlotkey	149	King	100
Abel	174	Zlotkey	149
Taylor	176	Zlotkey	149
Grant	178	Zlotkey	149
Fay	202	Hartstein	201
Gietz	206	Higgins	205

19 rows selected.

7. Modify `lab4_6.sql` to display all employees including King, who has no manager. Order the results by the employee number.
Place your SQL statement in a text file named `lab4_7.sql`. Run the query in `lab4_7.sql`.

Employee	EMP#	Manager	Mgr#
King	100		
Kochhar	101	King	100
De Haan	102	King	100
Hunold	103	De Haan	102
Ernst	104	Hunold	103
Lorentz	107	Hunold	103
Mourgos	124	King	100
Higgins	205	Kochhar	101
Gietz	206	Higgins	205

20 rows selected.

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label.

DEPARTMENT	EMPLOYEE	COLLEAGUE
20	Fay	Hartstein
20	Hartstein	Fay
50	Davies	Matos
50	Davies	Mourgos
50	Davies	Rajs
50	Davies	Vargas
50	Matos	Davies
50	Matos	Mourgos
50	Matos	Rajs
50	Matos	Vargas
110	Gietz	Higgins
110	Higgins	Gietz

42 rows selected.

9. Show the structure of the JOB_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees.

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRA
Matos	ST_CLERK	Shipping	2600	A
Vargas	ST_CLERK	Shipping	2500	A
Lorentz	IT_PROG	IT	4200	B
Mourgos	ST_MAN	Shipping	5800	B
Rajs	ST_CLERK	Shipping	3500	B
Davies	ST_CLERK	Shipping	3100	B
Whalen	AD_ASST	Administration	4400	B

De Haan	AD_VP	Executive	17000	E

19 rows selected.

10. Create a query to display the name and hire date of any employee hired after employee Davies.

LAST_NAME	HIRE_DATE
Lorentz	07-FEB-99
Mourgos	16-NOV-99
Matos	15-MAR-98
Vargas	09-JUL-98
Zlotkey	29-JAN-00
Taylor	24-MAR-98
Grant	24-MAY-99
Fay	17-AUG-97

8 rows selected.

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns `Employee`, `Emp Hired`, `Manager`, and `Mgr Hired`, respectively.

Employee	Emp Hired	Manager	Mgr Hired
Whalen	17-SEP-87	Kochhar	21-SEP-89
Hunold	03-JAN-90	De Haan	13-JAN-93
Rajs	17-OCT-95	Mourgos	16-NOV-99
Davies	29-JAN-97	Mourgos	16-NOV-99
Matos	15-MAR-98	Mourgos	16-NOV-99
Vargas	09-JUL-98	Mourgos	16-NOV-99
Abel	11-MAY-96	Zlotkey	29-JAN-00
Taylor	24-MAR-98	Zlotkey	29-JAN-00
Grant	24-MAY-99	Zlotkey	29-JAN-00

9 rows selected.