

D

Commonly Used SQL Commands

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this appendix, you should be able to:

- Execute a basic `SELECT` statement
- Create, alter, and drop a table using data definition language (DDL) statements
- Insert, update, and delete rows from one or more tables using data manipulation language (DML) statements
- Commit, roll back, and create savepoints by using transaction control statements
- Perform join operations on one or more tables

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to obtain data from one or more tables using the `SELECT` statement, how to use DDL statements to alter the structure of data objects, how to manipulate data in the existing schema objects by using DML statements, how to manage the changes made by DML statements, and how to use joins to display data from multiple tables using SQL:1999 join syntax.

Basic SELECT Statement

- Use the `SELECT` statement to:
 - Identify the columns to be displayed
 - Retrieve data from one or more tables, object tables, views, object views, or materialized views
- A `SELECT` statement is also known as a query because it queries a database.
- Syntax:

```
SELECT { * | [DISTINCT] column | expression [alias] , ... }  
FROM    table;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In its simplest form, a `SELECT` statement must include the following:

- A `SELECT` clause, which specifies the columns to be displayed
- A `FROM` clause, which identifies the table containing the columns that are listed in the `SELECT` clause

In the syntax:

<code>SELECT</code>	Is a list of one or more columns
<code>*</code>	Selects all columns
<code>DISTINCT</code>	Suppresses duplicates
<code>column / expression</code>	Selects the named column or the expression
<code>alias</code>	Gives different headings to the selected columns
<code>FROM table</code>	Specifies the table containing the columns

Note: Throughout this course, the words *keyword*, *clause*, and *statement* are used as follows:

- A *keyword* refers to an individual SQL element—for example, `SELECT` and `FROM` are keywords.
- A *clause* is a part of a SQL statement (for example, `SELECT employee_id, last_name`).
- A *statement* is a combination of two or more clauses (for example, `SELECT * FROM employees`).

SELECT Statement

- Select all columns:

```
SELECT *  
FROM job_history;
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA_REP	80
9	176	01-JAN-07	31-DEC-07	SA_MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

- Select specific columns:

```
SELECT manager_id, job_id  
FROM employees;
```

	MANAGER_ID	JOB_ID
1	(null)	AD_PRES
2	100	AD_VP
3	100	AD_VP
4	102	IT_PROG
5	103	IT_PROG
6	103	IT_PROG
7	100	ST_MAN
8	124	ST_CLERK
9	124	ST_CLERK
10	124	ST_CLERK

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can display all columns of data in a table by following the `SELECT` keyword with an asterisk (*) or by listing all the column names after the `SELECT` keyword. The first example in the slide displays all the rows from the `job_history` table. Specific columns of the table can be displayed by specifying the column names, separated by commas. The second example in the slide displays the `manager_id` and `job_id` columns from the `employees` table.

In the `SELECT` clause, specify the columns in the order in which you want them to appear in the output. For example, the following SQL statement displays the `location_id` column before displaying the `department_id` column:

```
SELECT location_id, department_id FROM departments;
```

Note: You can enter your SQL statement in a SQL Worksheet and click the Run Statement icon or press F9 to execute a statement in SQL Developer. The output displayed on the Results tabbed page appears as shown in the slide.

WHERE Clause

- Use the optional WHERE clause to:
 - Filter rows in a query
 - Produce a subset of rows
- Syntax:

```
SELECT * FROM table
[WHERE condition];
```

- Example:

```
SELECT location_id from departments
WHERE department_name = 'Marketing';
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The WHERE clause specifies a condition to filter rows, producing a subset of the rows in the table. A condition specifies a combination of one or more expressions and logical (Boolean) operators. It returns a value of TRUE, FALSE, or NULL. The example in the slide retrieves the location_id of the marketing department.

The WHERE clause can also be used to update or delete data from the database.

Example:

```
UPDATE departments
SET department_name = 'Administration'
WHERE department_id = 20;
and
DELETE from departments
WHERE department_id =20;
```

ORDER BY Clause

- Use the optional ORDER BY clause to specify the row order.
- Syntax:

```
SELECT * FROM table
[WHERE condition]
[ORDER BY {<column>|<position> } [ASC|DESC] [, ...] ];
```

- Example:

```
SELECT last_name, department_id, salary
FROM employees
ORDER BY department_id ASC, salary DESC;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The ORDER BY clause specifies the order in which the rows should be displayed. The rows can be sorted in ascending or descending fashion. By default, the rows are displayed in ascending order.

The example in the slide retrieves rows from the `employees` table ordered first by ascending order of `department_id`, and then by descending order of `salary`.

GROUP BY Clause

- Use the optional GROUP BY clause to group columns that have matching values into subsets.
- Each group has no two rows having the same value for the grouping column or columns.
- Syntax:

```
SELECT <column1, column2, ... column_n>  
FROM   table  
[WHERE condition]  
[GROUP BY <column> [, ...] ]  
[ORDER BY <column> [, ...] ] ;
```

- Example:

```
SELECT department_id, MIN(salary), MAX (salary)  
FROM employees  
GROUP BY department_id ;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The GROUP BY clause is used to group selected rows based on the value of `expr(s)` for each row. The clause groups rows but does not guarantee order of the result set. To order the groupings, use the ORDER BY clause.

Any SELECT list elements that are not included in aggregation functions must be included in the GROUP BY list of elements. This includes both columns and expressions. The database returns a single row of summary information for each group.

The example in the slide returns the minimum and maximum salaries for each department in the employees table.

Data Definition Language

- DDL statements are used to define, structurally change, and drop schema objects.
- The commonly used DDL statements are:
 - CREATE TABLE, ALTER TABLE, and DROP TABLE
 - GRANT, REVOKE
 - TRUNCATE

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

DDL statements enable you to alter the attributes of an object without altering the applications that access the object. You can also use DDL statements to alter the structure of objects while database users are performing work in the database. These statements are most frequently used to:

- Create, alter, and drop schema objects and other database structures, including the database itself and database users
- Delete all the data in schema objects without removing the structure of these objects
- Grant and revoke privileges and roles

Oracle Database implicitly commits the current transaction before and after every DDL statement.

CREATE TABLE Statement

- Use the CREATE TABLE statement to create a table in the database.
- Syntax:

```
CREATE TABLE tablename (  
  {column-definition | Table-level constraint}  
  [ , {column-definition | Table-level constraint} ] * )
```

- Example:

```
CREATE TABLE teach_dept (  
  department_id NUMBER(3) PRIMARY KEY,  
  department_name VARCHAR2(10));
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use the CREATE TABLE statement to create a table in the database. To create a table, you must have the CREATE TABLE privilege and a storage area in which to create objects.

The table owner and the database owner automatically gain the following privileges on the table after it is created:

- INSERT
- SELECT
- REFERENCES
- ALTER
- UPDATE

The table owner and the database owner can grant the preceding privileges to other users.

ALTER TABLE Statement

- Use the ALTER TABLE statement to modify the definition of an existing table in the database.
- Example1:

```
ALTER TABLE teach_dept  
ADD location_id NUMBER NOT NULL;
```

- Example 2:

```
ALTER TABLE teach_dept  
MODIFY department_name VARCHAR2(30) NOT NULL;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The ALTER TABLE statement allows you to make changes to an existing table.

You can:

- Add a column to a table
- Add a constraint to a table
- Modify an existing column definition
- Drop a column from a table
- Drop an existing constraint from a table
- Increase the width of the VARCHAR and CHAR columns
- Change a table to have read-only status

Example 1 in the slide adds a new column called `location_id` to the `teach_dept` table.

Example 2 updates the existing `department_name` column from `VARCHAR2(10)` to `VARCHAR2(30)`, and adds a `NOT NULL` constraint to it.

DROP TABLE Statement

- The DROP TABLE statement removes the table and all its data from the database.
- Example:

```
DROP TABLE teach_dept;
```

- DROP TABLE with the PURGE clause drops the table and releases the space that is associated with it.

```
DROP TABLE teach_dept PURGE;
```

- The CASCADE CONSTRAINTS clause drops all referential integrity constraints from the table.

```
DROP TABLE teach_dept CASCADE CONSTRAINTS;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DROP TABLE statement allows you to remove a table and its contents from the database, and pushes it to the recycle bin. Dropping a table invalidates dependent objects and removes object privileges on the table.

Use the PURGE clause along with the DROP TABLE statement to release back to the tablespace the space allocated for the table. You cannot roll back a DROP TABLE statement with the PURGE clause, nor can you recover the table if you have dropped it with the PURGE clause.

The CASCADE CONSTRAINTS clause allows you to drop the reference to the primary key and unique keys in the dropped table.

GRANT Statement

- The GRANT statement assigns the privilege to perform the following operations:
 - Insert or delete data.
 - Create a foreign key reference to the named table or to a subset of columns from a table.
 - Select data, a view, or a subset of columns from a table.
 - Create a trigger on a table.
 - Execute a specified function or procedure.
- Example:

```
GRANT SELECT any table to PUBLIC;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can use the GRANT statement to:

- Assign privileges to a specific user or role, or to all users, to perform actions on database objects
- Grant a role to a user, to PUBLIC, or to another role

Before you issue a GRANT statement, check that the `derby.database.sql` authorization property is set to `True`. This property enables SQL Authorization mode. You can grant privileges on an object if you are the owner of the database.

You can grant privileges to all users by using the `PUBLIC` keyword. When `PUBLIC` is specified, the privileges or roles affect all current and future users.

Privilege Types

Assign the following privileges by using the GRANT statement:

- ALL PRIVILEGES
- DELETE
- INSERT
- REFERENCES
- SELECT
- UPDATE

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Database provides a variety of privilege types to grant privileges to a user or role:

- Use the ALL PRIVILEGES privilege type to grant all privileges to the user or role for the specified table.
- Use the DELETE privilege type to grant permission to delete rows from the specified table.
- Use the INSERT privilege type to grant permission to insert rows into the specified table.
- Use the REFERENCES privilege type to grant permission to create a foreign key reference to the specified table.
- Use the SELECT privilege type to grant permission to perform SELECT statements on a table or view.
- Use the UPDATE privilege type to grant permission to use the UPDATE statement on the specified table.

REVOKE Statement

- Use the REVOKE statement to remove privileges from a user to perform actions on database objects.
- Revoke a *system privilege* from a user:

```
REVOKE DROP ANY TABLE  
FROM hr;
```

- Revoke a *role* from a user:

```
REVOKE dw_manager  
FROM sh;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The REVOKE statement removes privileges from a specific user (or users) or role to perform actions on database objects. It performs the following operations:

- Revokes a role from a user, from PUBLIC, or from another role
- Revokes privileges for an object if you are the owner of the object or the database owner

Note: To revoke a role or system privilege, you must have been granted the privilege with the ADMIN OPTION.

TRUNCATE TABLE Statement

- Use the TRUNCATE TABLE statement to remove all the rows from a table.
- Example:

```
TRUNCATE TABLE employees_demo;
```

- By default, Oracle Database performs the following tasks:
 - De-allocates space used by the removed rows
 - Sets the NEXT storage parameter to the size of the last extent removed from the segment by the truncation process

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The TRUNCATE TABLE statement deletes all the rows from a specific table. Removing rows with the TRUNCATE TABLE statement can be more efficient than dropping and re-creating a table.

Dropping and re-creating a table:

- Invalidates the dependent objects of the table
- Requires you to re-grant object privileges
- Requires you to re-create indexes, integrity constraints, and triggers.
- Re-specify its storage parameters

The TRUNCATE TABLE statement spares you from these efforts.

Note: You cannot roll back a TRUNCATE TABLE statement.

Data Manipulation Language

- DML statements query or manipulate data in the existing schema objects.
- A DML statement is executed when:
 - New rows are added to a table by using the `INSERT` statement
 - Existing rows in a table are modified using the `UPDATE` statement
 - Existing rows are deleted from a table by using the `DELETE` statement
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Data Manipulation Language (DML) statements enable you to query or change the contents of an existing schema object. These statements are most frequently used to:

- Add new rows of data to a table or view by specifying a list of column values or using a subquery to select and manipulate existing data
- Change column values in the existing rows of a table or view
- Remove rows from tables or views

A collection of DML statements that forms a logical unit of work is called a transaction. Unlike DDL statements, DML statements do not implicitly commit the current transaction.

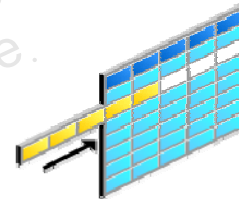
INSERT Statement

- Use the INSERT statement to add new rows to a table.
- Syntax:

```
INSERT INTO  table [(column [, column...])]  
VALUES      (value [, value...]);
```

- Example:

```
INSERT INTO  departments  
VALUES      (200, 'Development', 104, 1400);  
1 rows inserted.
```



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The INSERT statement adds rows to a table. Make sure to insert a new row containing values for each column and to list the values in the default order of the columns in the table. Optionally, you can also list the columns in the INSERT statement.

Example:

```
INSERT INTO job_history (employee_id, start_date, end_date,  
                        job_id)  
VALUES (120, '25-JUL-06', '12-FEB_08', 'AC_ACCOUNT');
```

The syntax discussed in the slide allows you to insert a single row at a time. The VALUES keyword assigns the values of expressions to the corresponding columns in the column list.

UPDATE Statement Syntax

- Use the UPDATE statement to modify the existing rows in a table.
- Update more than one row at a time (if required).

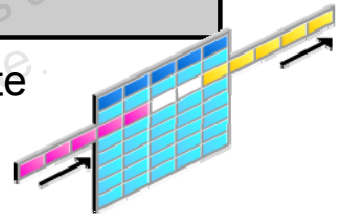
```
UPDATE    table
SET       column = value [, column = value, ...]
[WHERE    condition];
```

- Example:

```
UPDATE    copy_emp
SET
```

22 rows updated

- Specify SET *column_name* = NULL to update a column value to NULL.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The UPDATE statement modifies the existing values in a table. Confirm the update operation by querying the table to display the updated rows. You can modify a specific row or rows by specifying the WHERE clause.

Example:

```
UPDATE employees
SET    salary = 17500
WHERE  employee_id = 102;
```

In general, use the primary key column in the WHERE clause to identify the row to update. For example, to update a specific row in the employees table, use employee_id to identify the row instead of employee_name, because more than one employee may have the same name.

Note: Typically, the condition keyword is composed of column names, expressions, constants, subqueries, and comparison operators.

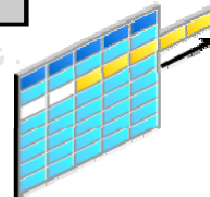
DELETE Statement

- Use the DELETE statement to delete the existing rows from a table.
- Syntax:

```
DELETE [FROM] table
[WHERE condition];
```

- Write the DELETE statement using the WHERE clause to delete specific rows from a table.

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 rows deleted
```



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DELETE statement removes existing rows from a table. You must use the WHERE clause to delete a specific row or rows from a table based on the condition. The condition identifies the rows to be deleted. It may contain column names, expressions, constants, subqueries, and comparison operators.

The first example in the slide deletes the finance department from the departments table. You can confirm the delete operation by using the SELECT statement to query the table.

```
SELECT *
FROM departments
WHERE department_name = 'Finance';
```

If you omit the WHERE clause, all rows in the table are deleted. Example:

```
DELETE FROM copy_emp;
```

The preceding example deletes all the rows from the copy_emp table.

Transaction Control Statements

- Transaction control statements are used to manage the changes made by DML statements.
- The DML statements are grouped into transactions.
- Transaction control statements include:
 - COMMIT
 - ROLLBACK
 - SAVEPOINT

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A transaction is a sequence of SQL statements that Oracle Database treats as a single unit. Transaction control statements are used in a database to manage the changes made by DML statements and to group these statements into transactions.

Each transaction is assigned a unique `transaction_id` and it groups SQL statements so that they are either all committed, which means they are applied to the database, or all rolled back, which means they are undone from the database.

COMMIT Statement

- Use the COMMIT statement to:
 - Permanently save the changes made to the database during the current transaction
 - Erase all savepoints in the transaction
 - Release transaction locks
- Example:

```
INSERT INTO departments
VALUES      (201, 'Engineering', 106, 1400);
COMMIT;
```

```
1 rows inserted.
committed.
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The COMMIT statement ends the current transaction by making all the pending data changes permanent. It releases all row and table locks, and erases any savepoints that you may have marked since the last commit or rollback. The changes made using the COMMIT statement are visible to all users.

Oracle recommends that you explicitly end every transaction in your application programs with a COMMIT or ROLLBACK statement, including the last transaction, before disconnecting from Oracle Database. If you do not explicitly commit the transaction and the program terminates abnormally, the last uncommitted transaction is automatically rolled back.

Note: Oracle Database issues an implicit COMMIT before and after any data definition language (DDL) statement.

ROLLBACK Statement

- Use the ROLLBACK statement to undo changes made to the database during the current transaction.
- Use the TO SAVEPOINT clause to undo a part of the transaction after the savepoint.
- Example:

```
UPDATE      employees
SET         salary = 7000
WHERE      last_name = 'Ernst';
SAVEPOINT   Ernst_sal;

UPDATE      employees
SET         salary = 12000
WHERE      last_name = 'Mourgos';

ROLLBACK TO SAVEPOINT Ernst_sal;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The ROLLBACK statement undoes work done in the current transaction. To roll back the current transaction, no privileges are necessary.

Using ROLLBACK with the TO SAVEPOINT clause performs the following operations:

- Rolls back only the portion of the transaction after the savepoint
- Erases all savepoints created after that savepoint. The named savepoint is retained, so you can roll back to the same savepoint multiple times.

Using ROLLBACK without the TO SAVEPOINT clause performs the following operations:

- Ends the transaction
- Undoes all the changes in the current transaction
- Erases all savepoints in the transaction

SAVEPOINT Statement

- Use the `SAVEPOINT` statement to name and mark the current point in the processing of a transaction.
- Specify a name to each savepoint.
- Use distinct savepoint names within a transaction to avoid overriding.
- Syntax:

```
SAVEPOINT savepoint;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The `SAVEPOINT` statement identifies a point in a transaction to which you can later roll back. You must specify a distinct name for each savepoint. If you create a second savepoint with the same identifier as an earlier savepoint, the earlier savepoint is erased.

After a savepoint has been created, you can either continue processing, commit your work, roll back the entire transaction, or roll back to the savepoint.

A simple rollback or commit erases all savepoints. When you roll back to a savepoint, any savepoints marked after that savepoint are erased. The savepoint to which you have rolled back is retained.

When savepoint names are reused within a transaction, the Oracle Database moves (overrides) the savepoint from its old position to the current point in the transaction.

Joins

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.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When data from more than one table in the database is required, a *join* condition is used. Rows in one table can be joined to rows in another table according to common values that exist in the corresponding columns (usually primary and foreign key columns).

To display data from two or more related tables, write a simple join condition in the **WHERE** clause.

In the syntax:

<code>table1.column</code>	Denotes the table and column from which data is retrieved
<code>table1.column1 =</code>	Is the condition that joins (or relates) the tables together
<code>table2.column2</code>	

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.

Types of Joins

- Natural join
- Equijoin
- Nonequijoin
- Outer join
- Self-join
- Cross join

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To join tables, you can use Oracle's join syntax.

Note: Before the Oracle 9i release, the join syntax was proprietary. The SQL:1999-compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax.

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use table aliases instead of full table name prefixes.
- Table aliases give a table a shorter name.
 - This keeps SQL code smaller and uses less memory.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When joining two or more tables, 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. Therefore, it is necessary to add the table prefix to execute your query. If there are no common column names between the two tables, there is no need to qualify the columns. However, using a table prefix improves performance, because you tell the Oracle server exactly where to find the columns.

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. Therefore, 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. Table aliases help to keep SQL code smaller, thereby using less memory.

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`.

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, that table alias must be substituted for the table name throughout the `SELECT` statement.
- Table aliases should be meaningful.
- A table alias is valid only for the current `SELECT` statement.

Natural Join

- The NATURAL JOIN clause is based on all the columns in the two tables that have the same name.
- It selects rows from tables that have the same names and data values of columns.
- Example:

```
SELECT country_id, location_id, country_name, city
FROM countries NATURAL JOIN locations;
```

	COUNTRY_ID	LOCATION_ID	COUNTRY_NAME	CITY
1	US	1400	United States of America	Southlake
2	US	1500	United States of America	South San Francisco
3	US	1700	United States of America	Seattle
4	CA	1800	Canada	Toronto
5	UK	2500	United Kingdom	Oxford

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can join tables automatically based on the columns in the two tables that have matching data types and names. You do this by using the NATURAL JOIN keywords.

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

In the example in the slide, the COUNTRIES table is joined to the LOCATIONS table by the COUNTRY_ID column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

Equijoins

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

...

DEPARTMENTS

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

Foreign key

Primary key

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

An **equijoin** is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. To determine an employee's department name, you compare the values in the `DEPARTMENT_ID` column in the `EMPLOYEES` table with the `DEPARTMENT_ID` values in the `DEPARTMENTS` table. The relationship between the `EMPLOYEES` and `DEPARTMENTS` tables is an *equijoin*; that is, values in the `DEPARTMENT_ID` column in both tables must be equal. Often, this type of join involves primary and foreign key complements.

Note: Equijoins are also called *simple joins*.

Retrieving Records with Equijoins

```
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	144 Vargas	50	50	1500
5	143 Matos	50	50	1500
6	142 Davies	50	50	1500
7	141 Rajs	50	50	1500
8	124 Mourgos	50	50	1500
9	103 Hunold	60	60	1400
10	104 Ernst	60	60	1400
11	107 Lorentz	60	60	1400

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the example in the slide:

- **The SELECT clause specifies the column names to retrieve:**
 - Employee last name, employee ID, and department ID, which are columns in the EMPLOYEES table
 - Department ID and location ID, which are columns in the DEPARTMENTS table
- **The FROM clause specifies the two tables that the database must access:**
 - EMPLOYEES table
 - DEPARTMENTS table
- **The WHERE clause specifies how the tables are to be joined:**

`e.department_id = d.department_id`

Because the DEPARTMENT_ID column is common to both tables, it must be prefixed with the table alias to avoid ambiguity. Other columns that are not present in both the tables need not be qualified by a table alias, but it is recommended for better performance.

Note: When you use the Execute Statement icon to run the query, SQL Developer suffixes a “_1” to differentiate between the two DEPARTMENT_IDS.

Additional Search Conditions Using the AND and WHERE Operators

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
AND d.department_id IN (20, 50);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
WHERE d.department_id IN (20, 50);
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In addition to the join, you may have criteria for your `WHERE` clause to restrict the rows in consideration for one or more tables in the join. The example in the slide performs a join on the `DEPARTMENTS` and `LOCATIONS` tables and, in addition, displays only those departments with ID equal to 20 or 50. To add additional conditions to the `ON` clause, you can add `AND` clauses. Alternatively, you can use a `WHERE` clause to apply additional conditions.

Both queries produce the same output.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping

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	Fay	6000	C

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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 that are 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 the `BETWEEN` condition. The Oracle server translates the `BETWEEN` condition to a pair of `AND` conditions. Therefore, using `BETWEEN` has no performance benefits, but should be used only for logical simplicity.

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

Retrieving Records by Using the USING Clause

- You can use the USING clause to match only one column when more than one column matches.
- You cannot specify this clause with a NATURAL join.
- Do not qualify the column name with a table name or table alias.
- Example:

```
SELECT country_id, country_name, location_id, city
FROM   countries JOIN locations
      USING (country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	US	United States of America	1400	Southlake
2	US	United States of America	1500	South San Francisco
3	US	United States of America	1700	Seattle
4	CA	Canada	1800	Toronto
5	UK	United Kingdom	2500	Oxford

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, the COUNTRY_ID columns in the COUNTRIES and LOCATIONS tables are joined and thus the LOCATION_ID of the location where an employee works is shown.

Retrieving Records by Using 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 ON clause makes code easy to understand.

```
SELECT e.employee_id, e.last_name, j.department_id,  
FROM   employees e JOIN job_history j  
ON     (e.employee_id = j.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	101	Kochhar	110
2	101	Kochhar	110
3	102	De Haan	60
4	176	Taylor	80
5	176	Taylor	80
6	200	Whalen	90
7	200	Whalen	90
8	201	Hartstein	20

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use the ON clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the WHERE clause.

In this example, the EMPLOYEE_ID columns in the EMPLOYEES and JOB_HISTORY tables are joined using the ON clause. Wherever an employee ID in the EMPLOYEES table equals an employee ID in the JOB_HISTORY table, the row is returned. The table alias is necessary to qualify the matching column names.

You can also use the ON clause to join columns that have different names. The parentheses around the joined columns, as in the example in the slide, (e.employee_id = j.employee_id), is optional. So, even ON e.employee_id = j.employee_id will work.

Note: When you use the Execute Statement icon to run the query, SQL Developer suffixes a '_1' to differentiate between the two employee_ids.

Left Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from the left table is called a LEFT OUTER JOIN.
- Example:

```
SELECT c.country_id, c.country_name, l.location_id, l.city
FROM   countries c LEFT OUTER JOIN locations l
ON     (c.country_id = l.country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	CA	Canada	1800	Toronto
2	DE	Germany	(null)	(null)
3	UK	United Kingdom	2500	Oxford
4	US	United States of America	1400	Southlake
5	US	United States of America	1500	South San Francisco
6	US	United States of America	1700	Seattle

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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

Right Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from the right table is called a RIGHT OUTER JOIN.
- Example:

```
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
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping

...

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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

Full Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from both tables is called a FULL OUTER JOIN.
- Example:

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

	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	DEPARTMENT_NAME
1	King	(null)	(null)	(null)
2	Kochhar	90	100	Executive
3	De Haan	90	100	Executive
4	Hunold	(null)	(null)	(null)

...

19	Higgins	(null)	(null)	(null)
20	Gietz	110	205	Accounting
21	(null)	190	(null)	Contracting
22	(null)	10	200	Administration

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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

Self-Join: Example

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker JOIN employees manager
ON     worker.manager_id = manager.employee_id
ORDER BY worker.last_name;
```

	WORKER.LAST_NAME 'WORKS FOR' MANAGER.LAST_NAME
1	Abel works for Zlotkey
2	Davies works for Mourgoss
3	De Haan works for King
4	Ernst works for Hunold
5	Fay works for Hartstein
6	Gietz works for Higgins
7	Grant works for Zlotkey
8	Hartstein works for King
9	Higgins works for Kochhar

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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. The example in the slide joins the `EMPLOYEES` table to itself. To simulate two tables in the `FROM` clause, there are two aliases, namely `worker` and `manager`, for the same table, `EMPLOYEES`.

In this example, the `WHERE` clause contains the join that means "where a worker's manager ID matches the employee ID for the manager."

Cross Join

- A CROSS JOIN is a JOIN operation that produces the Cartesian product of two tables.
- Example:

```
SELECT department_name, city  
FROM department CROSS JOIN location;
```

	DEPARTMENT_NAME	CITY
1	Administration	Oxford
2	Administration	Seattle
3	Administration	South San Francisco
4	Administration	Southlake
5	Administration	Toronto
6	Marketing	Oxford
7	Marketing	Seattle
8	Marketing	South San Francisco
9	Marketing	Southlake
10	Marketing	Toronto

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The CROSS JOIN syntax specifies the cross product. It is also known as a Cartesian product. A cross join produces the cross product of two relations, and is essentially the same as the comma-delimited Oracle Database notation.

You do not specify any WHERE condition between the two tables in the CROSS JOIN.

Summary

In this appendix, you should have learned how to use:

- The `SELECT` statement to retrieve rows from one or more tables
- DDL statements to alter the structure of objects
- DML statements to manipulate data in the existing schema objects
- Transaction control statements to manage the changes made by DML statements
- Joins to display data from multiple tables

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There are many commonly used commands and statements in SQL. It includes the DDL statements, DML statements, transaction control statements, and joins.

YASINEB MAZOUZ (z.m.malak@hotmail.com) has a non-transferable
license to use this Student Guide.