11

Creating Views

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Describe a view**
- **Create, alter the definition of, and drop a view**
- **Retrieve data through a view**
- **Insert, update, and delete data through a view**
- **Create and use an inline view**
- **Perform top-*n* analysis**

ORACLE

Lesson Aim

In this lesson, you learn to create and use views. You also learn to query the relevant data dictionary object to retrieve information about views. Finally, you learn to create and use inline views, and perform top-*n* analysis using inline views.

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence	Generates primary key values
Index	Improves the performance of some queries
Synonym	Alternative name for an object

ORACLE

What Is a View?

EMPLOYEES Table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
							6000
							4200
							5800
							3500
							31
							2600
							250
149	Zlotkey			10500		SA_MAN	10500
174	Abel			11000		SA_REP	11000
176	Taylor			8600		SA_REP	860
					FEB-96	MK_MAN	13000
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	600
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	830

20 rows selected.

ORACLE

What Is a View?

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.

Why Use Views?

- To restrict data access
- To make complex queries easy
- To provide data independence
- To present different views of the same data

ORACLE

Advantages of Views

- Views restrict access to the data because the view can display selective columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of users access to data according to their particular criteria.

For more information, see *Oracle9i SQL Reference,* "`CREATE VIEW`."

Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

ORACLE

Simple Views versus Complex Views

There are two classifications for views: simple and complex. The basic difference is related to the DML (`INSERT`, `UPDATE`, and `DELETE`) operations.

- A simple view is one that:
  - Derives data from only one table
  - Contains no functions or groups of data
  - Can perform DML operations through the view
- A complex view is one that:
  - Derives data from many tables
  - Contains functions or groups of data
  - Does not always allow DML operations through the view

Creating a View

- **You embed a subquery within the `CREATE VIEW` statement.**

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- **The subquery can contain complex `SELECT` syntax.**

ORACLE

Creating a View

You can create a view by embedding a subquery within the `CREATE VIEW` statement.

In the syntax:

`OR REPLACE`	re-creates the view if it already exists
`FORCE`	creates the view regardless of whether or not the base tables exist
`NOFORCE`	creates the view only if the base tables exist (This is the default.)
`view`	is the name of the view
`alias`	specifies names for the expressions selected by the view's query (The number of aliases must match the number of expressions selected by the view.)
`subquery`	is a complete `SELECT` statement (You can use aliases for the columns in the `SELECT` list.)
`WITH CHECK OPTION`	specifies that only rows accessible to the view can be inserted or updated
`constraint`	is the name assigned to the `CHECK OPTION` constraint
`WITH READ ONLY`	ensures that no DML operations can be performed on this view

Creating a View

- **Create a view, EMPVU80, that contains details of employees in department 80.**

```
CREATE VIEW  empvu80
AS SELECT  employee_id, last_name, salary
   FROM    employees
   WHERE   department_id = 80;
View created.
```

- **Describe the structure of the view by using the *i*SQL*Plus DESCRIBE command.**

```
DESCRIBE empvu80
```

ORACLE

Creating a View (continued)

The example in the slide creates a view that contains the employee number, last name, and salary for each employee in department 80.

You can display the structure of the view by using the *i*SQL*Plus `DESCRIBE` command.

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)

Guidelines for creating a view:

- The subquery that defines a view can contain complex `SELECT` syntax, including joins, groups, and subqueries.
- The subquery that defines the view cannot contain an `ORDER BY` clause. The `ORDER BY` clause is specified when you retrieve data from the view.
- If you do not specify a constraint name for a view created with the `WITH CHECK OPTION`, the system assigns a default name in the format `SYS_C`*n*.
- You can use the `OR REPLACE` option to change the definition of the view without dropping and re-creating it or regranting object privileges previously granted on it.

Creating a View

- **Create a view by using column aliases in the subquery.**

```
CREATE VIEW  salvu50
AS SELECT  employee_id ID_NUMBER, last_name NAME,
           salary*12 ANN_SALARY
   FROM    employees
   WHERE   department_id = 50;
View created.
```

- **Select the columns from this view by the given alias names.**

ORACLE

Creating a View (continued)

You can control the column names by including column aliases within the subquery.

The example in the slide creates a view containing the employee number (EMPLOYEE_ID) with the alias ID_NUMBER, name (LAST_NAME) with the alias NAME, and annual salary (SALARY) with the alias ANN_SALARY for every employee in department 50.

As an alternative, you can use an alias after the CREATE statement and prior to the SELECT subquery. The number of aliases listed must match the number of expressions selected in the subquery.

```
CREATE VIEW   salvu50 (ID_NUMBER, NAME, ANN_SALARY)
  AS SELECT  employee_id, last_name, salary*12
  FROM    employees
  WHERE   department_id = 50;
View created.
```

Retrieving Data from a View

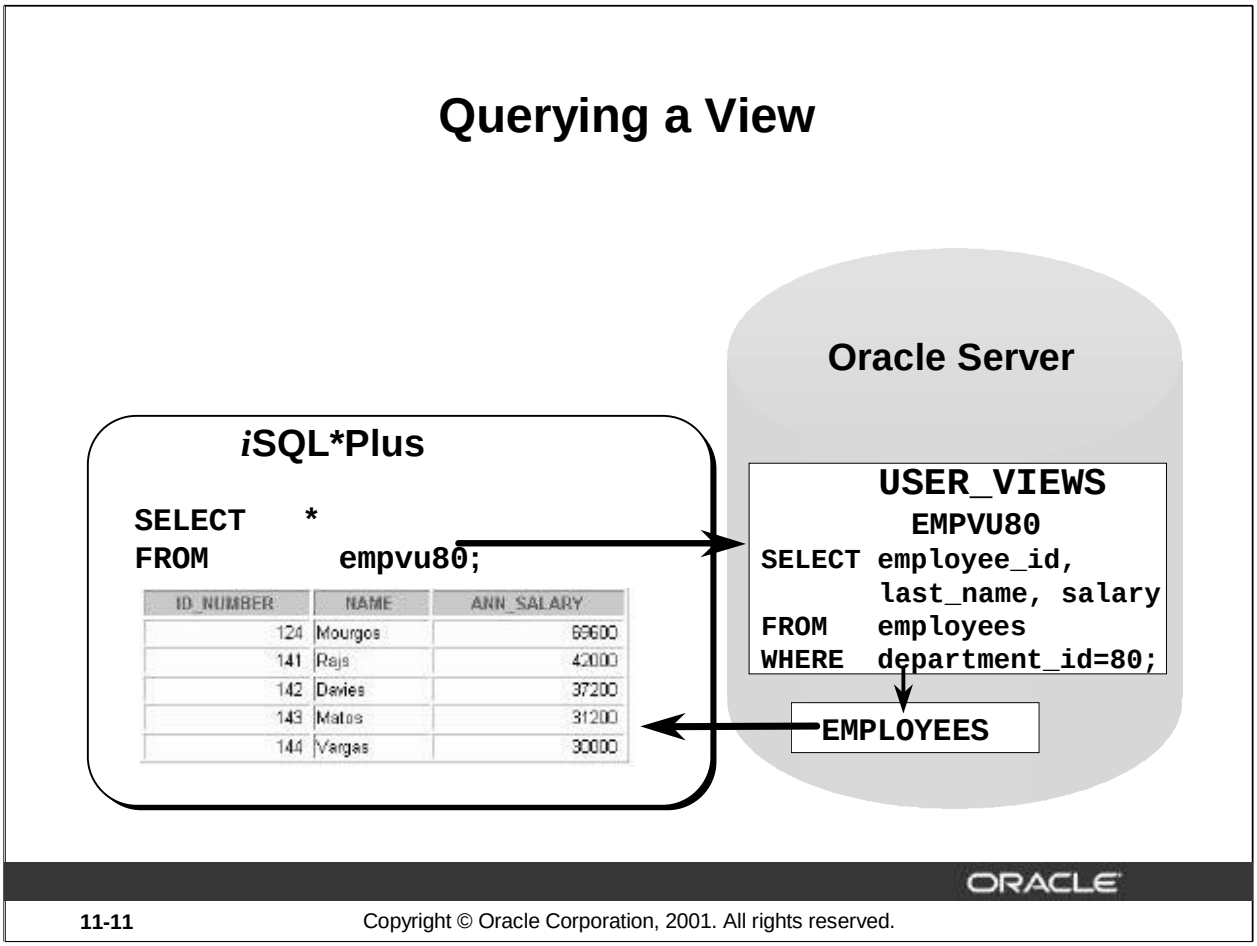
```
SELECT *
FROM  salvu50;
```

ID_NUMBER	NAME	ANN_SALARY
124	Mourgos	69600
141	Rajs	42000
142	Davies	37200
143	Matos	31200
144	Vargas	30000

ORACLE

Retrieving Data from a View

You can retrieve data from a view as you would from any table. You can display either the contents of the entire view or just specific rows and columns.



Views in the Data Dictionary

Once your view has been created, you can query the data dictionary view called `USER_VIEWS` to see the name of the view and the view definition. The text of the `SELECT` statement that constitutes your view is stored in a `LONG` column.

Data Access Using Views

When you access data using a view, the Oracle Server performs the following operations:

1. It retrieves the view definition from the data dictionary table `USER_VIEWS`.
2. It checks access privileges for the view base table.
3. It converts the view query into an equivalent operation on the underlying base table or tables. In other words, data is retrieved from, or an update is made to, the base tables.

Modifying a View

- **Modify the `EMPVU80` view by using `CREATE OR REPLACE VIEW` clause. Add an alias for each column name.**

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' ' || last_name,
           salary, department_id
   FROM    employees
   WHERE   department_id = 80;
View created.
```

- **Column aliases in the `CREATE VIEW` clause are listed in the same order as the columns in the subquery.**

ORACLE

Modifying a View

With the `OR REPLACE` option, a view can be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, re-creating, and regranting object privileges.

Note: When assigning column aliases in the `CREATE VIEW` clause, remember that the aliases are listed in the same order as the columns in the subquery.

Creating a Complex View

Create a complex view that contains group functions to display values from two tables.

```
CREATE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT    d.department_name, MIN(e.salary),
             MAX(e.salary),AVG(e.salary)
   FROM      employees e, departments d
   WHERE     e.department_id = d.department_id
   GROUP BY  d.department_name;
View created.
```

ORACLE

Creating a Complex View

The example in the slide creates a complex view of department names, minimum salaries, maximum salaries, and average salaries by department. Note that alternative names have been specified for the view. This is a requirement if any column of the view is derived from a function or an expression.

You can view the structure of the view by using the *i*SQL*Plus `DESCRIBE` command. Display the contents of the view by issuing a `SELECT` statement.

```
SELECT  *
FROM    dept_sum_vu;
```

NAME	MINSAL	MAXSAL	AVGSAL
Accounting	8300	12000	10150
Administration	4400	4400	4400
Executive	17000	24000	19333.3333
IT	4200	9000	6400
Marketing	6000	13000	9500
Sales	8600	11000	10033.3333
Shipping	2500	5800	3500

7 rows selected.

Rules for Performing DML Operations on a View

- **You can perform DML operations on simple views.**
- **You cannot remove a row if the view contains the following:**
  - **Group functions**
  - **A `GROUP BY` clause**
  - **The `DISTINCT` keyword**
  - **The pseudocolumn `ROWNUM` keyword**

ORACLE

Performing DML Operations on a View

You can perform DML operations on data through a view if those operations follow certain rules.

You can remove a row from a view unless it contains any of the following:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword

Rules for Performing DML Operations on a View

You cannot modify data in a view if it contains:

- **Group functions**
- **A `GROUP BY` clause**
- **The `DISTINCT` keyword**
- **The pseudocolumn `ROWNUM` keyword**
- **Columns defined by expressions**

ORACLE

Performing DML Operations on a View (continued)

You can modify data through a view unless it contains any of the conditions mentioned in the previous slide or columns defined by expressions: for example, `SALARY * 12`.

Rules for Performing DML Operations on a View

You cannot add data through a view if the view includes:

- **Group functions**
- **A `GROUP BY` clause**
- **The `DISTINCT` keyword**
- **The pseudocolumn `ROWNUM` keyword**
- **Columns defined by expressions**
- **`NOT NULL` columns in the base tables that are not selected by the view**

ORACLE

Performing DML Operations on a View (continued)

You can add data through a view unless it contains any of the items listed in the slide or there are NOT NULL columns, without default values, in the base table that are not selected by the view. All required values must be present in the view. Remember that you are adding values directly into the underlying table through the view.

For more information, see *0racle9i SQL Reference*, "`CREATE VIEW`."

Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay within the domain of the view by using the WITH CHECK OPTION clause.

```
CREATE OR REPLACE VIEW empvu20
AS SELECT   *
   FROM     employees
   WHERE    department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck;
View created.
```

- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

ORACLE

Using the WITH CHECK OPTION Clause

It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.

The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows which the view cannot select, and therefore it allows integrity constraints and data validation checks to be enforced on data being inserted or updated.

If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, with the constraint name if that has been specified.

```
UPDATE empvu20
  SET    department_id = 10
  WHERE employee_id = 201;
UPDATE empvu20
       *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Note: No rows are updated because if the department number were to change to 10, the view would no longer be able to see that employee. Therefore, with the WITH CHECK OPTION clause, the view can see only employees in department 20 and does not allow the department number for those employees to be changed through the view.

Denying DML Operations

- **You can ensure that no DML operations occur by adding the `WITH READ ONLY` option to your view definition.**
- **Any attempt to perform a DML on any row in the view results in an Oracle server error.**

ORACLE

Denying DML Operations

You can ensure that no DML operations occur on your view by creating it with the `WITH READ ONLY` option. The example in the slide modifies the `EMPVU10` view to prevent any DML operations on the view.

Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
    (employee_number, employee_name, job_title)
AS SELECT   employee_id, last_name, job_id
   FROM     employees
   WHERE    department_id = 10
   WITH READ ONLY;
View created.
```

ORACLE

Denying DML Operations

Any attempts to remove a row from a view with a read-only constraint results in an error.

```
DELETE FROM empvu10
  WHERE employee_number = 200;
DELETE FROM empvu10
        *
ERROR at line 1:
ORA-01752: cannot delete from view without exactly one key-
preserved table
```

Any attempts to insert a row or modify a row using the view with a read-only constraint results in the following Oracle Server error:

```
01733: virtual column not allowed here.
```

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
View dropped.
```

ORACLE

Removing a View

You use the `DROP VIEW` statement to remove a view. The statement removes the view definition from the database. Dropping views has no effect on the tables on which the view was based. Views or other applications based on deleted views become invalid. Only the creator or a user with the `DROP ANY VIEW` privilege can remove a view.

In the syntax:

view is the name of the view

Inline Views

- **An inline view is a subquery with an alias (or correlation name) that you can use within a SQL statement.**
- **A named subquery in the `FROM` clause of the main query is an example of an inline view.**
- **An inline view is not a schema object.**

ORACLE

Inline Views

An inline view is created by placing a subquery in the `FROM` clause and giving that subquery an alias. The subquery defines a data source that can be referenced in the main query. In the following example, the inline view `b` returns the details of all department numbers and the maximum salary for each department from the `EMPLOYEES` table. The `WHERE a.department_id = b.department_id AND a.salary < b.maxsal` clause of the main query displays employee names, salaries, department numbers, and maximum salaries for all the employees who earn less than the maximum salary in their department.

```
SELECT  a.last_name, a.salary, a.department_id, b.maxsal
FROM    employees a, (SELECT   department_id, max(salary) maxsal
                      FROM     employees
                      GROUP BY department_id) b
WHERE   a.department_id = b.department_id
AND     a.salary < b.maxsal;
```

LAST_NAME	SALARY	DEPARTMENT_ID	MAXSAL
Fay	6000	20	13000
Rajs	3500	50	5800
Davies	3100	50	5800
Gietz	8300	110	10300

12 rows selected.

Top-*n* Analysis

- **Top-*n* queries ask for the *n* largest or smallest values of a column. For example:**
  - **What are the ten best selling products?**
  - **What are the ten worst selling products ?**
- **Both largest values and smallest values sets are considered top-*n* queries.**

ORACLE

Top-n Analysis

Top-*n* queries are useful in scenarios where the need is to display only the *n* top-most or the *n* bottommost records from a table based on a condition. This result set can be used for further analysis. For example using top-*n* analysis you can perform the following types of queries:

- The top three earners in the company
- The four most recent recruits in the company
- The top two sales representatives who have sold the maximum number of products
- The top three products that have had the maximum sales in the last six months

Performing Top-*n* Analysis

The high-level structure of a top-*n* analysis query is:

```
SELECT [column_list], ROWNUM
FROM   (SELECT [column_list]
        FROM table
        ORDER  BY Top-N_column)
WHERE  ROWNUM <=  N;
```

ORACLE

Performing Top-*n* Analysis

Top-*n* queries use a consistent nested query structure with the elements described below:

- A subquery or an inline view to generate the sorted list of data. The subquery or the inline view includes the `ORDER BY` clause to ensure that the ranking is in the desired order. For results retrieving the largest values, a `DESC` parameter is needed.
- An outer query to limit the number of rows in the final result set. The outer query includes the following components:
  - The `ROWNUM` pseudocolumn, which assigns a sequential value starting with 1 to each of the rows returned from the subquery.
  - A `WHERE` clause, which specifies the *n* rows to be returned. The outer `WHERE` clause must use a < or <= operator.

Example of Top-*n* Analysis

To display the top three earner names and salaries from the EMPLOYEES table.

```
SELECT ROWNUM as RANK, last_name, salary
FROM  (SELECT last_name,salary FROM employees
       ORDER BY salary DESC)
WHERE ROWNUM <= 3;
```

RANK	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000

ORACLE

Example of Top-*n* Analysis

The example in the slide illustrates how to display the names and salaries of the top three earners from the EMPLOYEES table. The subquery returns the details of all employee names and salaries from the EMPLOYEES table, sorted in the descending order of the salaries. The WHERE ROWNUM < 3 clause of the main query ensures that only the first three records from this result set are displayed.

Here is another example of top-*n* analysis that uses an inline view. The example below uses the inline view E to display the four most senior employees in the company.

```
SELECT ROWNUM as SENIOR,E.last_name, E.hire_date
FROM  (SELECT last_name,hire_date FROM employees
       ORDER BY hire_date)E
WHERE rownum <= 4;
```

SENIOR	LAST_NAME	HIRE_DATE
1	King	17-JUN-87
2	Whalen	17-SEP-87
3	Kochhar	21-SEP-89
4	Hunold	03-JAN-90

Summary

In this lesson you should have learned that a view is derived from data in other tables or other views and provides the following advantages:

- **Restricts database access**
- **Simplifies queries**
- **Provides data independence**
- **Provides multiple views of the same data**
- **Can be dropped without removing the underlying data**

ORACLE

What Is a View?

A view is based on a table or another view and acts as a window through which data on tables can be viewed or changed. A view does not contain data. The definition of the view is stored in the data dictionary. You can see the definition of the view in the `USER_VIEWS` data dictionary table.

Advantages of Views

- Restrict database access
- Simplify queries
- Provide data independence
- Provide multiple views of the same data
- Can be removed without affecting the underlying data

View Options

- Can be a simple view, based on one table
- Can be a complex view based on more than one table or can contain groups of functions
- Can replace other views with the same name
- Can contain a check constraint
- Can be read-only

Practice 11 Overview

This practice covers the following topics:

- **Creating a simple view**
- **Creating a complex view**
- **Creating a view with a check constraint**
- **Attempting to modify data in the view**
- **Displaying view definitions**
- **Removing views**

ORACLE

Practice 11 Overview

In this practice, you create simple and complex views and attempt to perform DML statements on the views.

Practice 11

1. Create a view called EMPLOYEES_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.
2. Display the contents of the EMPLOYEES_VU view.

EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60
206	Gietz	110

20 rows selected.

3. Select the view name and text from the USER_VIEWS data dictionary view.

   Note: Another view already exists. The EMP_DETAILS_VIEW was created as part of your schema.

   Note: To see more contents of a LONG column, use the *i*SQL*Plus command SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

VIEW_NAME	TEXT
EMPLOYEES_VU	SELECT employee_id, last_name employee, department_id FROM employees
EMP_DETAILS_VIEW	SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.country_id, e.first_name, e.last_name, e.salary, e.commission_pct, d.department_name, j.job_title, l.city, l.state_province, c.country_name, r.region_name FROM employees e, departments d, jobs j, locations l, countries c, regions r WHERE e.department_id = d.department_id AND d.location_id = l.location_id AND l.country_id = c.country_id AND c.region_id = r.region_id AND j.job_id = e.job_id WITH READ ONLY

4. Using your EMPLOYEES_VU view, enter a query to display all employee names and department numbers.

EMPLOYEE	DEPARTMENT_ID
King	90
Kochhar	90
Gietz	110

20 rows selected.

Practice 11 (continued)

5. Create a view named `DEPT50` that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns `EMPNO`, `EMPLOYEE`, and `DEPTNO`. Do not allow an employee to be reassigned to another department through the view.
6. Display the structure and contents of the `DEPT50` view.

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(6)
EMPLOYEE	NOT NULL	VARCHAR2(25)
DEPTNO		NUMBER(4)

EMPNO	EMPLOYEE	DEPTNO
124	Mourgos	50
141	Rajs	50
142	Davies	50
143	Matos	50
144	Vargas	50

7. Attempt to reassign Matos to department 80.

If you have time, complete the following exercise:

8. Create a view called `SALARY_VU` based on the employee last names, department names, salaries, and salary grades for all employees. Use the `EMPLOYEES`, `DEPARTMENTS`, and `JOB_GRADES` tables. Label the columns `Employee`, `Department`, `Salary`, and `Grade`, respectively.

12

Other Database Objects

ORACLE

15

Using SET Operators

ORACLE

Objectives

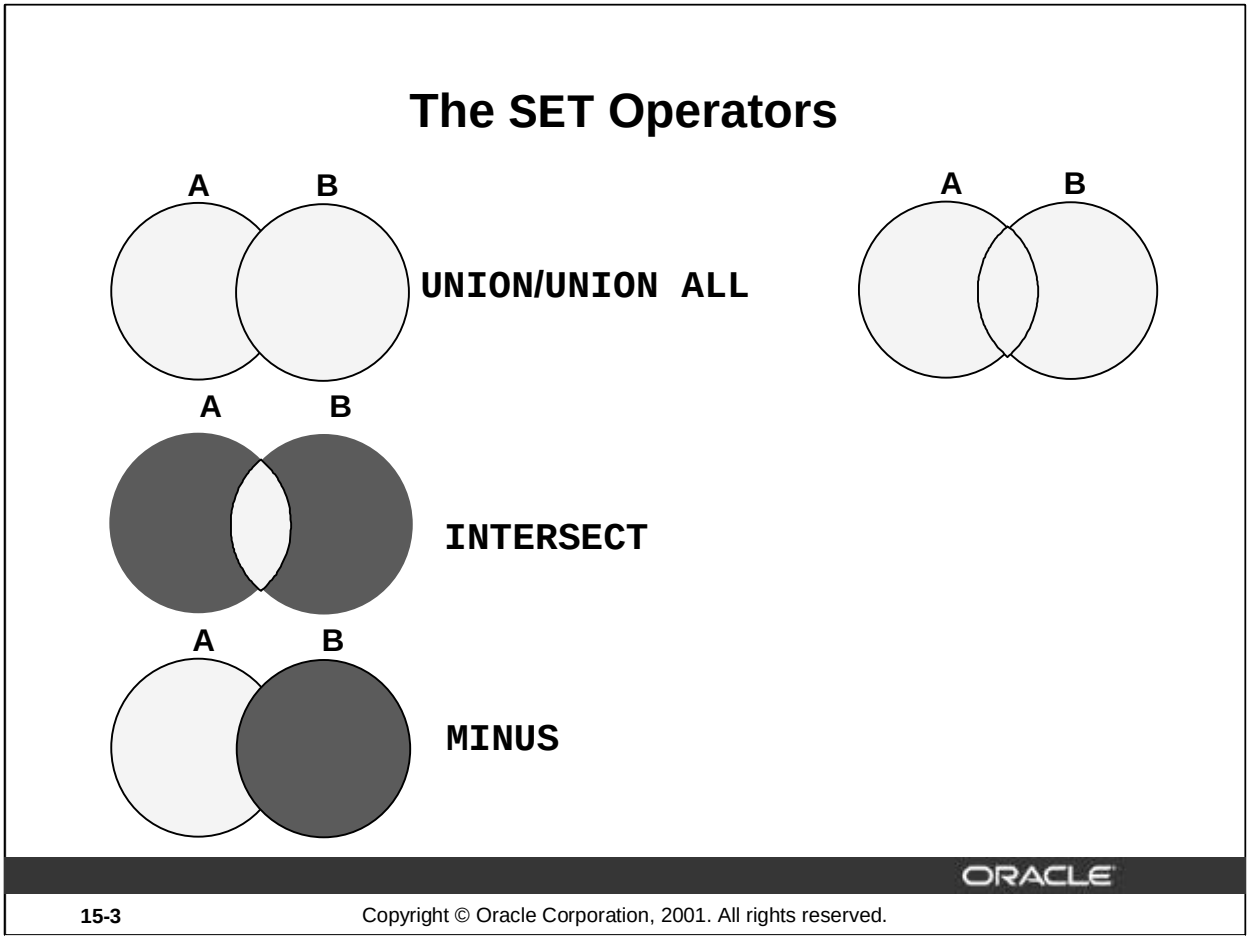
After completing this lesson, you should be able to do the following:

- **Describe SET operators**
- **Use a SET operator to combine multiple queries into a single query**
- **Control the order of rows returned**

ORACLE

Lesson Aim

In this lesson, you learn how to write queries by using SET operators.



The SET Operators

The SET operators combine the results of two or more component queries into one result. Queries containing SET operators are called *compound queries*.

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
INTERSECT	All distinct rows selected by both queries
MINUS	All distinct rows that are selected by the first SELECT statement and that are not selected in the second SELECT statement

All SET operators have equal precedence. If a SQL statement contains multiple SET operators, the Oracle server evaluates them from left (top) to right (bottom) if no parentheses explicitly specify another order. You should use parentheses to specify the order of evaluation explicitly in queries that use the INTERSECT operator with other SET operators.

Note: In the slide, the light color (grey) in the diagram represents the query result.

Tables Used in This Lesson

The tables used in this lesson are:

- **EMPLOYEES: Provides details regarding all current employees**
- **JOB_HISTORY: When an employee switches jobs, the details of the start date and end date of the former job, the job identification number and department are recorded in this table**

ORACLE

Tables Used in This Lesson

Two tables are used in this lesson. They are the EMPLOYEES table and the JOB_HISTORY table.

The EMPLOYEES table stores the employee details. For the human resource records, this table stores a unique identification number and email address for each employee. The details of the employee's job identification number, salary, and manager are also stored. Some of the employees earn a commission in addition to their salary; this information is tracked too. The company organizes the roles of employees into jobs. Some of the employees have been with the company for a long time and have switched to different jobs. This is monitored using the JOB_HISTORY table. When an employee switches jobs, the details of the start date and end date of the former job, the job identification number and department are recorded in the JOB_HISTORY table.

The structure and the data from the EMPLOYEES and the JOB_HISTORY tables are shown on the next page.

There have been instances in the company of people who have held the same position more than once during their tenure with the company. For example, consider the employee Taylor, who joined the company on 24-MAR-1998. Taylor held the job title SA_REP for the period 24-MAR-98 to 31-DEC-98 and the job title SA_MAN for the period 01-JAN-99 to 31-DEC-99. Taylor moved back into the job title of SA_REP, which is his current job title.

Similarly consider the employee Whalen, who joined the company on 17-SEP-1987. Whalen held the job title AD_ASST for the period 17-SEP-87 to 17-JUN-93 and the job title AC_ACCOUNT for the period 01-JUL-94 to 31-DEC-98. Taylor moved back into the job title of AD_ASST, which is his current job title.

Tables Used in This Lesson (continued)

```
DESC employees
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

```
SELECT employee_id, last_name, job_id, hire_date, department_id
FROM employees;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	DEPARTMENT_ID
100	King	AD_PRES	17-JUN-87	90
101	Kochhar	AD_VP	21-SEP-89	90
102	De Haan	AD_VP	13-JAN-93	90
103	Hunold	IT_PROG	03-JAN-90	60
104	Ernst	IT_PROG	21-MAY-91	60
107	Lorentz	IT_PROG	07-FEB-99	60
124	Mourgos	ST_MAN	16-NOV-99	50
141	Rajs	ST_CLERK	17-OCT-95	50
142	Davies	ST_CLERK	29-JAN-97	50
143	Matos	ST_CLERK	15-MAR-98	50
144	Vargas	ST_CLERK	09-JUL-98	50
149	Zlotkey	SA_MAN	29-JAN-00	80
174	Abel	SA_REP	11-MAY-96	80
176	Taylor	SA_REP	24-MAR-98	80
178	Grant	SA_REP	24-MAY-99	
200	Whalen	AD_ASST	17-SEP-87	10
201	Hartstein	MK_MAN	17-FEB-96	20
202	Fay	MK_REP	17-AUG-97	20
205	Higgins	AC_MGR	07-JUN-94	110
206	Gietz	AC_ACCOUNT	07-JUN-94	110

20 rows selected.

Tables Used in This Lesson (continued)

```
DESC job_history
```

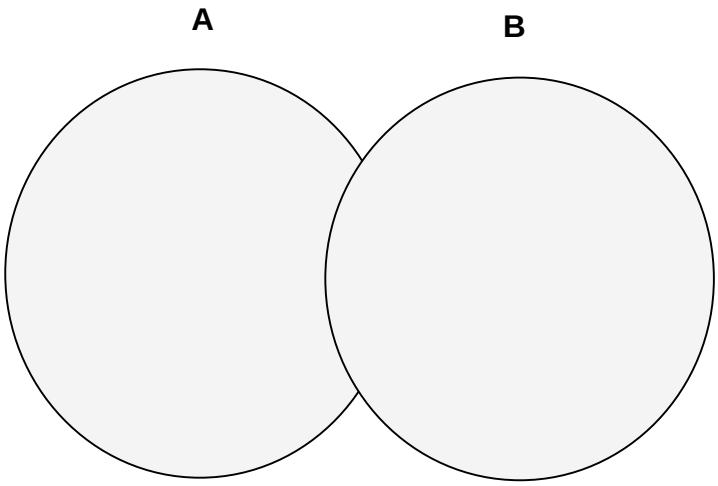
Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

```
SELECT * FROM job_history;
```

EMPLOYEE_ID	START_DAT	END_DATE	JOB_ID	DEPARTMENT_ID
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

10 rows selected.

The UNION SET Operator



The UNION operator returns results from both queries after eliminating duplications.

ORACLE

The UNION SET Operator

The UNION operator returns all rows selected by either query. Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.

Guidelines

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
- The IN operator has a higher precedence than the UNION operator.
- By default, the output is sorted in ascending order of the first column of the SELECT clause.

Using the UNION Operator

Display the current and previous job details of all employees. Display each employee only once.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AC_ACCOUNT
101	AD_VP
178	SA_REP
200	AC_ACCOUNT
200	AD_ASST

28 rows selected.

ORACLE

Using the UNION SET Operator

The UNION operator eliminates any duplicate records. If there are records that occur both in the EMPLOYEES and the JOB_HISTORY tables and are identical, the records will be displayed only once. Observe in the output shown on the slide that the record for the employee with the EMPLOYEE_ID 200 appears twice as the JOB_ID is different in each row.

Consider the following example:

```
SELECT  employee_id, job_id, department_id
FROM    employees
UNION
SELECT  employee_id, job_id, department_id
FROM    job_history;
```

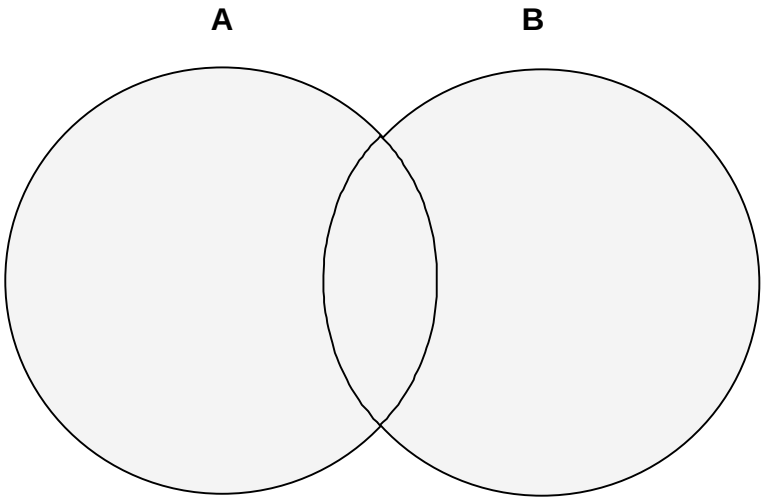
EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
100	AD_PRES	90
101	AC_ACCOUNT	110
101	AC_MGR	110
101	AD_VP	90
200	AC_ACCOUNT	90
200	AD_ASST	10
200	AD_ASST	90
201	MK_MAN	20

29 rows selected.

Using the UNION SET Operator (continued)

In the preceding output, employee 200 appears three times. Why? Notice the DEPARTMENT_ID values for employee 200. One row has a DEPARTMENT_ID of 90, another 10, and the third 90. Because of these unique combinations of job IDs and department IDs, each row for employee 200 is unique and therefore not considered a duplicate. Observe that the output is sorted in ascending order of the first column of the SELECT clause, EMPLOYEE_ID in this case.

The UNION ALL Operator



The UNION ALL operator returns results from both queries including all duplications.

ORACLE

The UNION ALL Operator

Use the UNION ALL operator to return all rows from multiple queries.

Guidelines

- Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.
- The DISTINCT keyword cannot be used.

Note: With the exception of the above, the guidelines for UNION and UNION ALL are the same.

Using the UNION ALL Operator

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY  employee_id;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
100	AD_PRES	90
174	SA_REP	80
176	SA_REP	80
176	SA_MAN	80
176	SA_REP	80
205	AC_MGR	110
206	AC_ACCOUNT	110

30 rows selected.

ORACLE

The UNION ALL Operator (continued)

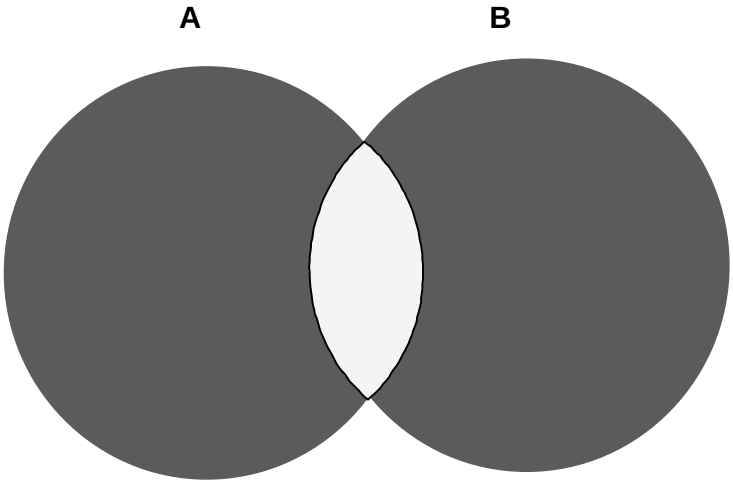
In the example, 30 rows are selected. The combination of the two tables totals to 30 rows. The UNION ALL operator does not eliminate duplicate records. The duplicate records are highlighted in the output shown in the slide. UNION returns all distinct rows selected by either query. UNION ALL returns all rows selected by either query, including all duplicates. Consider the query on the slide, now written with the UNION clause:

```
SELECT   employee_id, job_id,department_id
FROM     employees
UNION
SELECT   employee_id, job_id,department_id
FROM      job_history
ORDER BY employee_id;
```

The preceding query returns 29 rows. This is because it eliminates the following row (as it is a duplicate):

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
176	SA_REP	80

The INTERSECT Operator



The INTERSECT operator returns results that are common to both queries.

ORACLE

The INTERSECT Operator

Use the INTERSECT operator to return all rows common to multiple queries.

Guidelines

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- Reversing the order of the intersected tables does not alter the result.
- INTERSECT does not ignore NULL values.

Using the INTERSECT Operator

Display the employee IDs and job IDs of employees who are currently in a job title that they have held once before during their tenure with the company

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
176	SA_REP
200	AD_ASST

ORACLE

The INTERSECT Operator (continued)

In the example in this slide, the query returns only the records that have the same values in the selected columns in both tables.

What will be the results if you add the DEPARTMENT_ID column to the SELECT statement from the EMPLOYEES table and add the DEPARTMENT_ID column to the SELECT statement from the JOB_HISTORY table and run this query? The results may be different because of the introduction of another column whose values may or may not be duplicates.

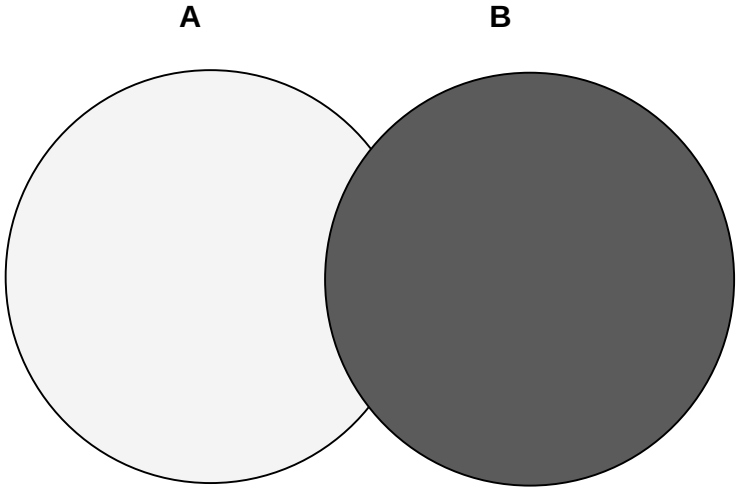
Example

```
SELECT employee_id, job_id, department_id
FROM   employees
INTERSECT
SELECT employee_id, job_id, department_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
176	SA_REP	80

Employee 200 is no longer part of the results because the EMPLOYEES.DEPARTMENT_ID value is different from the JOB_HISTORY.DEPARTMENT_ID value.

The MINUS Operator



The MINUS operator returns rows from the first query that are not present in the second query.

ORACLE

The MINUS Operator

Use the MINUS operator to return rows returned by the first query that are not present in the second query (the first SELECT statement MINUS the second SELECT statement).

Guidelines

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- All of the columns in the WHERE clause must be in the SELECT clause for the MINUS operator to work.

The MINUS Operator

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id
FROM   employees
MINUS
SELECT employee_id
FROM   job_history;
```

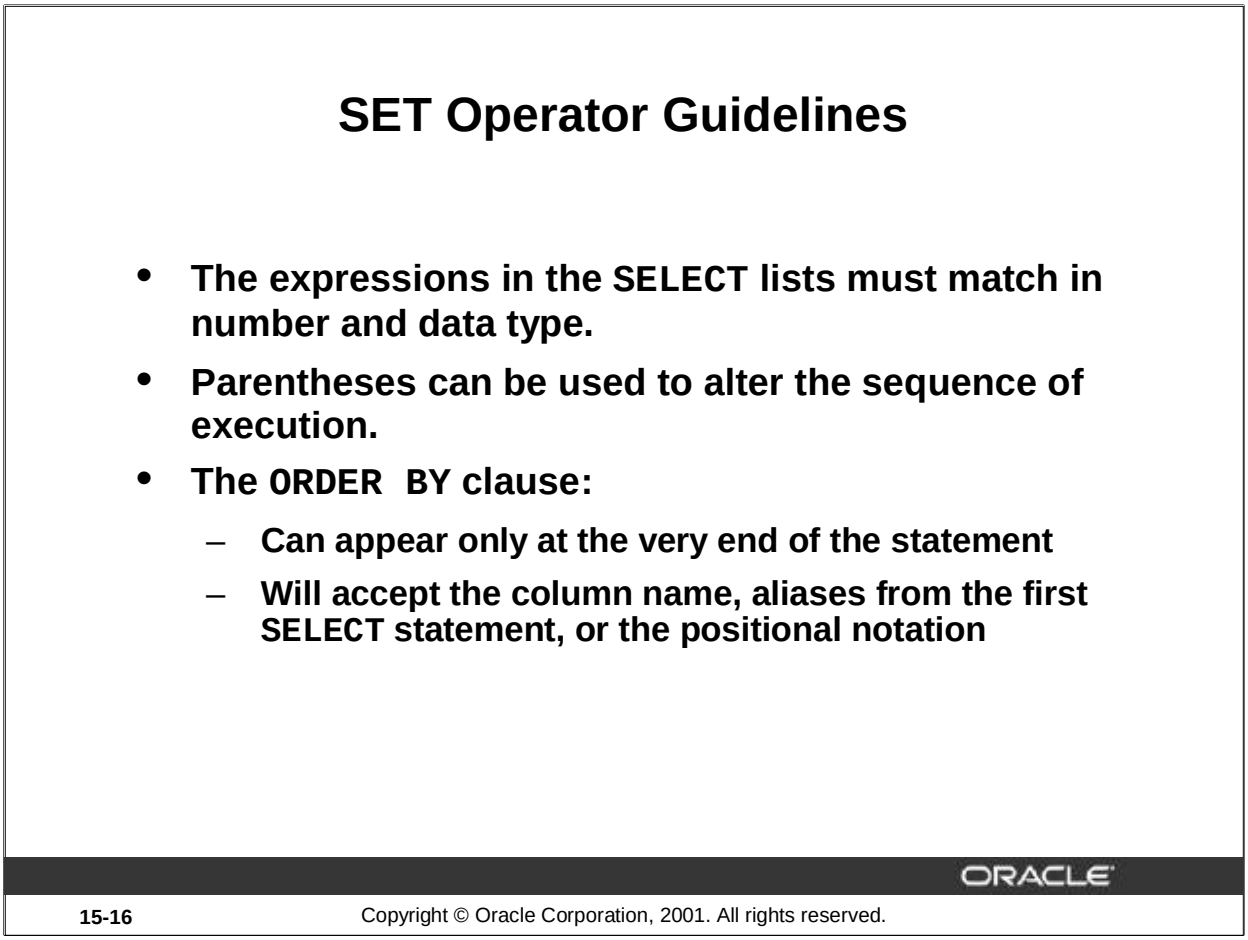
EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AD_VP
201	MK_MAN
202	MK_REP
205	AC_MGR
206	AC_ACCOUNT

18 rows selected.

ORACLE

The MINUS Operator (continued)

In the example in the slide, the employee IDs in the JOB_HISTORY table are subtracted from those in the EMPLOYEES table. The results set displays the employees remaining after the subtraction; they are represented by rows that exist in the EMPLOYEES table but do not exist in the JOB_HISTORY table. These are the records of the employees who have not changed their jobs even once.

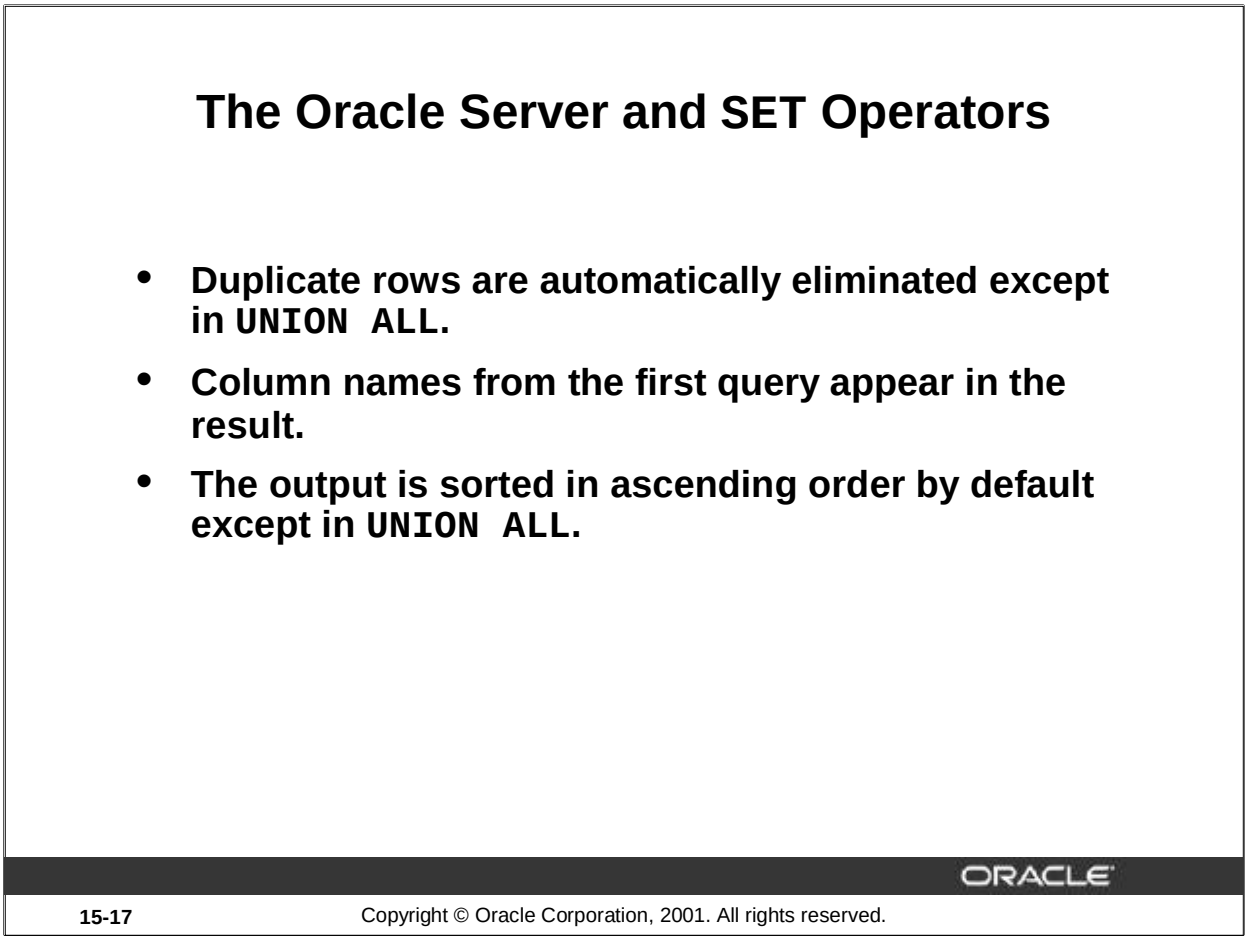


SET Operator Guidelines

- The expressions in the select lists of the queries must match in number and datatype. Queries that use `UNION`, `UNION ALL`, `INTERSECT`, and `MINUS` `SET` operators in their `WHERE` clause must have the same number and type of columns in their `SELECT` list. For example:

```
SELECT employee_id, department_id
FROM   employees
WHERE  (employee_id, department_id)
       IN (SELECT employee_id, department_id
           FROM   employees
           UNION
           SELECT  employee_id, department_id
           FROM   job_history);
```

- The `ORDER BY` clause:
  - Can appear only at the very end of the statement
  - Will accept the column name, an alias, or the positional notation
- The column name or alias, if used in an `ORDER BY` clause, must be from the first `SELECT` list.
- `SET` operators can be used in subqueries.



The Oracle Server and SET Operators

When a query uses SET operators, the Oracle Server eliminates duplicate rows automatically except in the case of the `UNION ALL` operator. The column names in the output are decided by the column list in the first `SELECT` statement. By default, the output is sorted in ascending order of the first column of the `SELECT` clause.

The corresponding expressions in the select lists of the component queries of a compound query must match in number and datatype. If component queries select character data, the data type of the return values are determined as follows:

- If both queries select values of datatype `CHAR`, the returned values have datatype `CHAR`.
- If either or both of the queries select values of datatype `VARCHAR2`, the returned values have datatype `VARCHAR2`.

Matching the SELECT Statements

Using the UNION operator, display the department ID, location, and hire date for all employees.

```
SELECT department_id, TO_NUMBER(null) location, hire_date
FROM   employees
UNION
SELECT department_id, location_id,  TO_DATE(null)
FROM   departments;
```

DEPARTMENT_ID	LOCATION	HIRE_DATE
10	1700	
10		17-SEP-87
20	1800	
20		17-FEB-96
		-94
190	1700	
		24-MAY-99

27 rows selected.

ORACLE

Matching the SELECT Statements

As the expressions in the select lists of the queries must match in number , you can use dummy columns and the data type conversion functions to comply with this rule. In the slide, the name `location` is given as the dummy column heading. The `TO_NUMBER` function is used in the first query to match the `NUMBER` data type of the `LOCATION_ID` column retrieved by the second query. Similarly, the `TO_DATE` function in the second query is used to match the `DATE` datatype of the `HIRE_DATE` column retrieved by the second query.

Matching the SELECT Statement

Using the UNION operator, display the employee ID, job ID, and salary of all employees.

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID	SALARY
100	AD_PRES	24000
101	AC_ACCOUNT	0
101	AC_MGR	0
205	AC_MGR	12008
206	AC_ACCOUNT	8300

30 rows selected.

ORACLE

Matching the SELECT Statement: Example

The EMPLOYEES and JOB_HISTORY tables have several columns in common; for example, EMPLOYEE_ID, JOB_ID and DEPARTMENT_ID. But what if you want the query to display the EMPLOYEE_ID, JOB_ID, and SALARY using the UNION operator, knowing that the salary exists only in the, EMPLOYEES table?

The code example in the slide matches the EMPLOYEE_ID and the JOB_ID columns in the EMPLOYEES and in the JOB_HISTORY tables. A literal value of 0 is added to the JOB_HISTORY SELECT statement to match the numeric SALARY column in the EMPLOYEES SELECT statement.

In the preceding results, each row in the output that corresponds to a record from the JOB_HISTORY table contains a 0 in the SALARY column.

Controlling the Order of Rows

Produce an English sentence using two UNION operators.

```
COLUMN a_dummy NOPRINT
SELECT 'sing' AS "My dream", 3 a_dummy
FROM dual
UNION
SELECT 'I''d like to teach', 1
FROM dual
UNION
SELECT 'the world to', 2
FROM dual
ORDER BY 2;
```

My dream
I'd like to teach
the world to
sing

ORACLE

Controlling the Order of Rows

By default, the output is sorted in ascending order on the first column. You can use the `ORDER BY` clause to change this.

Using `ORDER BY` to Order Rows

The `ORDER BY` clause can be used only once in a compound query. If used, the `ORDER BY` clause must be placed at the end of the query. The `ORDER BY` clause accepts the column name, an alias, or the positional notation. Without the `ORDER BY` clause, the code example in the slide produces the following output in the alphabetical order of the first column:

My dream
I'd like to teach
sing
the world to

Note: Consider a compound query where the `UNION SET` operator is used more than once. In this case, the `ORDER BY` clause can use only positions rather than explicit expressions.

Summary

In this lesson, you should have learned the following:

- **`UNION` returns all distinct rows.**
- **`UNION ALL` returns all rows, including duplicates.**
- **`INTERSECT` returns all rows shared by both queries.**
- **`MINUS` returns all distinct rows selected by the first query but not by the second.**
- **`ORDER BY` can appear only at the very end of the statement.**

ORACLE

Summary

- The `UNION` operator returns all rows selected by either query. Use the `UNION` operator to return all rows from multiple tables and eliminate any duplicate rows.
- Use the `UNION ALL` operator to return all rows from multiple queries. Unlike with the `UNION` operator, duplicate rows are not eliminated and the output is not sorted by default.
- Use the `INTERSECT` operator to return all rows common to multiple queries.
- Use the `MINUS` operator to return rows returned by the first query that are not present in the second query.
- Remember to use the `ORDER BY` clause only at the very end of the compound statement.
- Make sure that the corresponding expressions in the `SELECT` lists match in number and data type.

Practice 15 Overview

This practice covers the following topics:

- **Writing queries using the SET operators**
- **Discovering alternative join methods**

ORACLE

Practice 15 Overview

In this practice, you write queries using the SET operators.

Practice 15

1. List the department IDs for departments that do not contain the job ID ST_CLERK, using SET operators.

DEPARTMENT_ID
10
20
60
80
90
110
190

7 rows selected.

2. Display the country ID and the name of the countries that have no departments located in them, using SET operators.

CO	COUNTRY_NAME
DE	Germany

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID, using SET operators.

JOB_ID	DEPARTMENT_ID
AD_ASST	10
ST_CLERK	50
ST_MAN	50
MK_MAN	20
MK_REP	20

4. List the employee IDs and job IDs of those employees who are currently in the job title that they have held once before during their tenure with the company.

EMPLOYEE_ID	JOB_ID
176	SA_REP
200	AD_ASST

Practice 15 (Continued)

5. Write a compound query that lists the following:
   - Last names and department ID of all the employees from the `EMPLOYEES` table, irrespective of the fact whether they belong to any department or not
   - Department ID and department name of all the departments from the `DEPARTMENTS` table, irrespective of the fact whether they have employees working in them or not.

LAST_NAME	DEPARTMENT_ID	TO_CHAR(NULL)
Abel	80	
Davies	50	
De Haan	90	
Ernst	60	
Fay	20	
Gietz	110	
Grant		
Hartstein	20	
Higgins	110	
Hunold	60	
King	90	
Kochhar	90	
Lorentz	60	
Matos	50	
Mourgos	50	
Rajs	50	
Taylor	80	
Vargas	50	
Whalen	10	
Zlotkey	80	
	10	Administration
	20	Marketing
	50	Shipping
	60	IT
	80	Sales
	90	Executive
	110	Accounting
	190	Contracting

28 rows selected.

13

Controlling User Access

ORACLE

Objectives

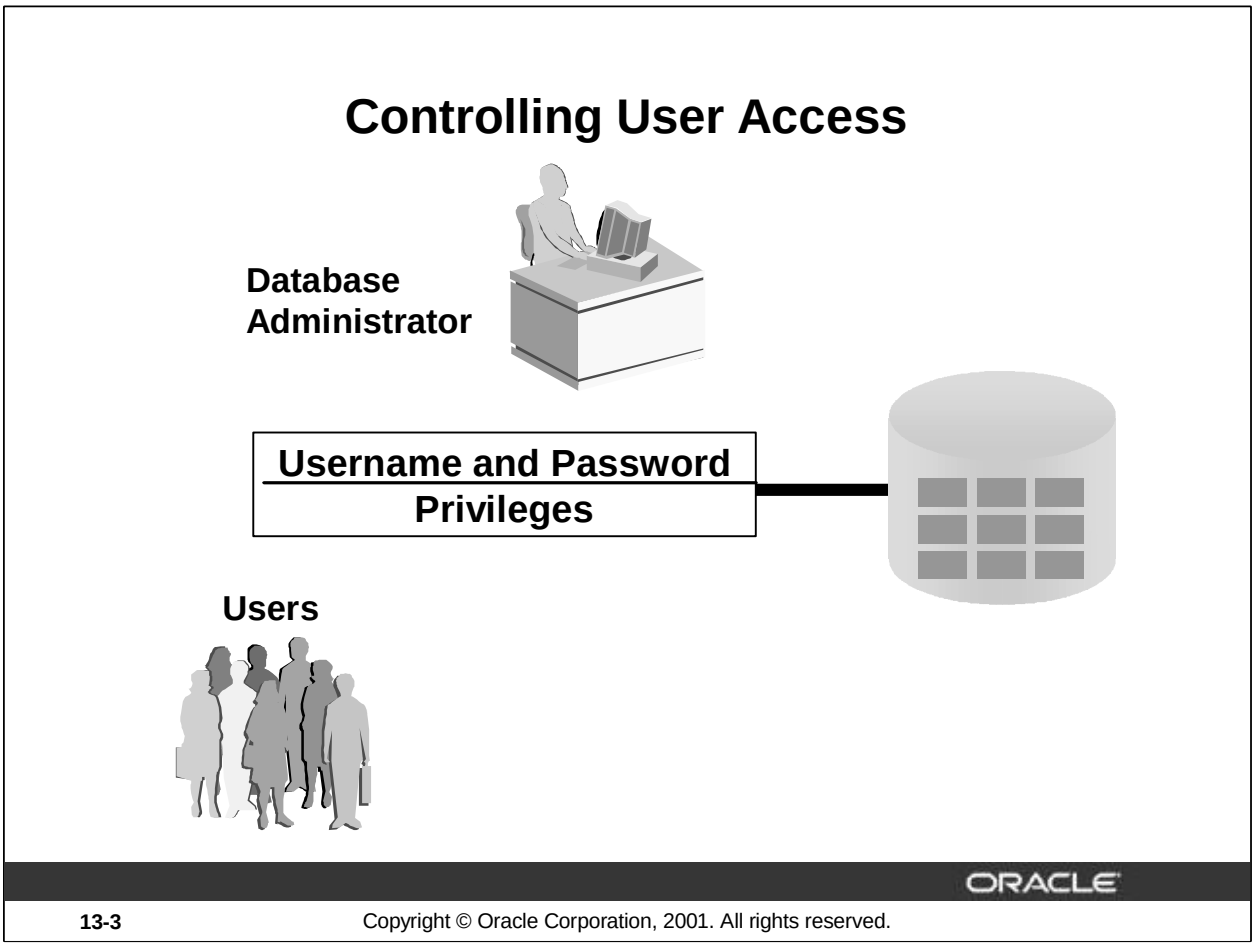
After completing this lesson, you should be able to do the following:

- **Create users**
- **Create roles to ease setup and maintenance of the security model**
- **Use the `GRANT` and `REVOKE` statements to grant and revoke object privileges**
- **Create and access database links**

ORACLE

Lesson Aim

In this lesson, you learn how to control database access to specific objects and add new users with different levels of access privileges.



Controlling User Access

In a multiple-user environment, you want to maintain security of the database access and use. With Oracle server database security, you can do the following:

- Control database access
- Give access to specific objects in the database
- Confirm given and received privileges with the Oracle data dictionary
- Create synonyms for database objects

Database security can be classified into two categories: system security and data security. System security covers access and use of the database at the system level, such as the username and password, the disk space allocated to users, and the system operations that users can perform. Database security covers access and use of the database objects and the actions that those users can have on the objects.

Privileges

- **Database security:**
  - **System security**
  - **Data security**
- **System privileges: Gaining access to the database**
- **Object privileges: Manipulating the content of the database objects**
- **Schemas: Collections of objects, such as tables, views, and sequences**

ORACLE

Privileges

Privileges are the right to execute particular SQL statements. The database administrator (DBA) is a high-level user with the ability to grant users access to the database and its objects. The users require system privileges to gain access to the database and object privileges to manipulate the content of the objects in the database. Users can also be given the privilege to grant additional privileges to other users or to roles, which are named groups of related privileges.

Schemas

A schema is a collection of objects, such as tables, views, and sequences. The schema is owned by a database user and has the same name as that user.

For more information, see *Oracle9i Application Developer's Guide - Fundamentals,* "Establishing a Security Policy," and *Oracle9i Concepts*, "Database Security."

System Privileges

- More than 100 privileges are available.
- The database administrator has high-level system privileges for tasks such as:
  - Creating new users
  - Removing users
  - Removing tables
  - Backing up tables

ORACLE

System Privileges

More than 100 distinct system privileges are available for users and roles. System privileges typically are provided by the database administrator.

Typical DBA Privileges

System Privilege	Operations Authorized
CREATE USER	Grantee can create other Oracle users (a privilege required for a DBA role).
DROP USER	Grantee can drop another user.
DROP ANY TABLE	Grantee can drop a table in any schema.
BACKUP ANY TABLE	Grantee can back up any table in any schema with the export utility.
SELECT ANY TABLE	Grantee can query tables, views, or snapshots in any schema.
CREATE ANY TABLE	Grantee can create tables in any schema.

Creating Users

The DBA creates users by using the CREATE USER statement.

```
CREATE USER user
IDENTIFIED BY   password;
```

```
CREATE USER  scott
IDENTIFIED BY   tiger;
User created.
```

ORACLE

Creating a User

The DBA creates the user by executing the CREATE USER statement. The user does not have any privileges at this point. The DBA can then grant privileges to that user. These privileges determine what the user can do at the database level.

The slide gives the abridged syntax for creating a user.

In the syntax:

user	is the name of the user to be created
password	specifies that the user must log in with this password

For more information, see *Oracle9i SQL Reference*, "GRANT" and "CREATE USER."

User System Privileges

- **Once a user is created, the DBA can grant specific system privileges to a user.**

```
GRANT privilege [, privilege...]
TO user [, user| role, PUBLIC...];
```

- **An application developer, for example, may have the following system privileges:**
  - **CREATE SESSION**
  - **CREATE TABLE**
  - **CREATE SEQUENCE**
  - **CREATE VIEW**
  - **CREATE PROCEDURE**

ORACLE

Typical User Privileges

Now that the DBA has created a user, the DBA can assign privileges to that user.

System Privilege	Operations Authorized
CREATE SESSION	Connect to the database
CREATE TABLE	Create tables in the user's schema
CREATE SEQUENCE	Create a sequence in the user's schema
CREATE VIEW	Create a view in the user's schema
CREATE PROCEDURE	Create a stored procedure, function, or package in the user's schema

In the syntax:

privilege	is the system privilege to be granted
user |role|PUBLIC	is the name of the user, the name of the role, or PUBLIC designates that every user is granted the privilege

Note: Current system privileges can be found in the dictionary view SESSION_PRIVS.

Granting System Privileges

The DBA can grant a user specific system privileges.

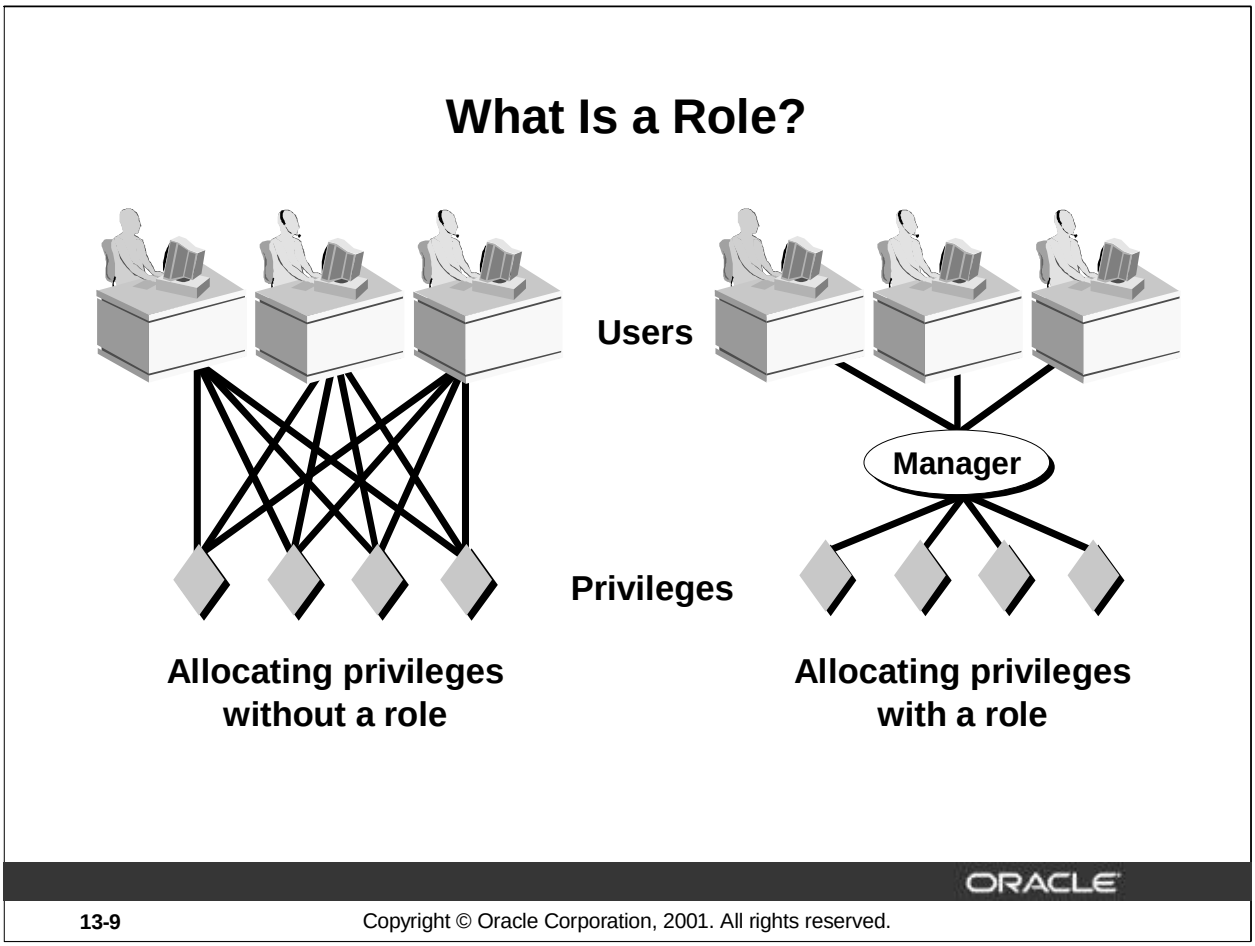
```
GRANT  create session, create table,
       create sequence, create view
TO     scott;
Grant succeeded.
```

ORACLE

Granting System Privileges

The DBA uses the GRANT statement to allocate system privileges to the user. Once the user has been granted the privileges, the user can immediately use those privileges.

In the example in the slide, user Scott has been assigned the privileges to create sessions, tables, sequences, and views.



What Is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges.

A user can have access to several roles, and several users can be assigned the same role. Roles are typically created for a database application.

Creating and Assigning a Role

First, the DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax

```
CREATE   ROLE  role;
```

In the syntax:

role is the name of the role to be created

Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.

Creating and Granting Privileges to a Role

- Create a role

```
CREATE ROLE manager;
Role created.
```

- Grant privileges to a role

```
GRANT create table, create view
TO manager;
Grant succeeded.
```

- Grant a role to users

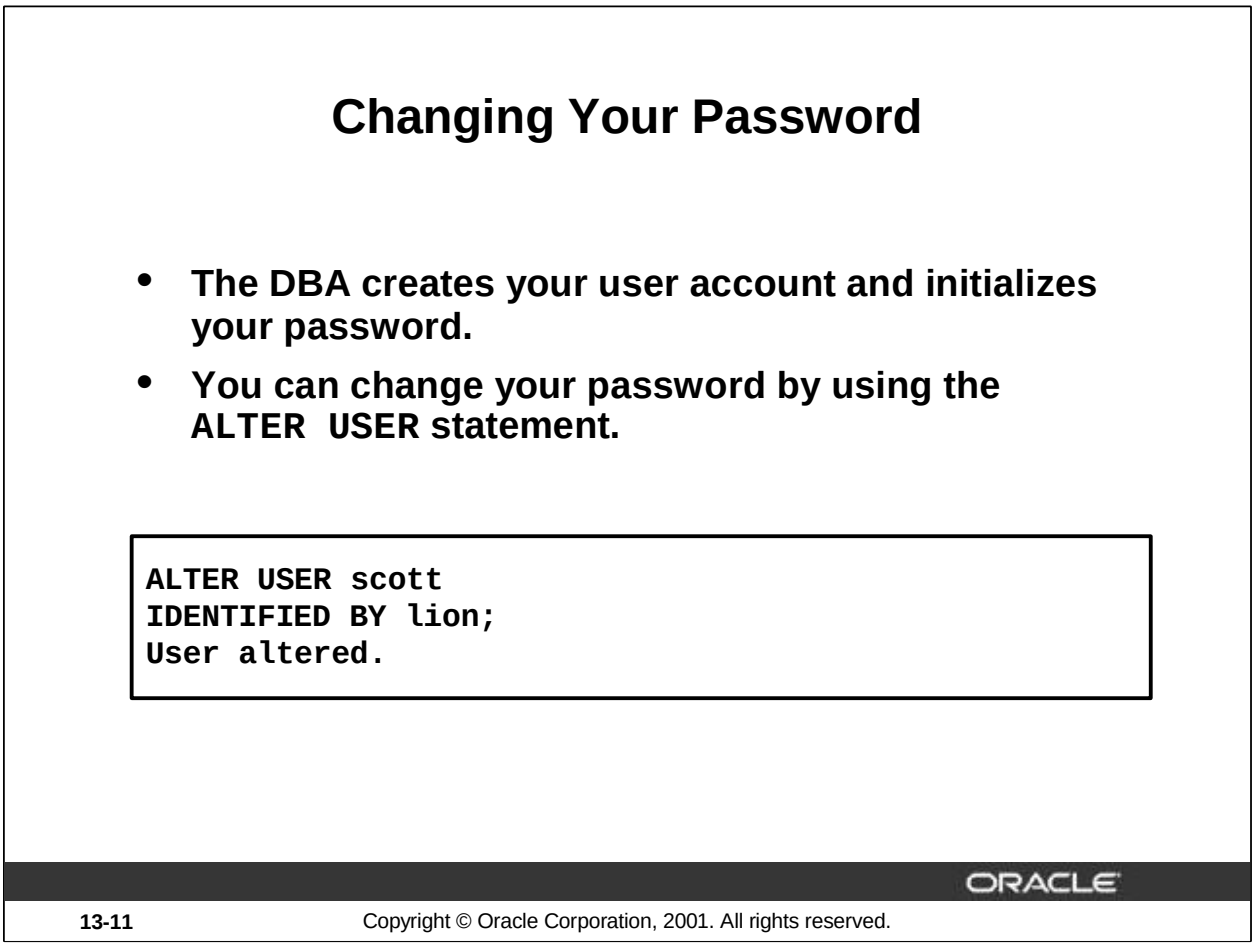
```
GRANT manager TO DEHAAN, KOCHHAR;
Grant succeeded.
```

ORACLE

Creating a Role

The example in the slide creates a manager role and then allows managers to create tables and views. It then grants DeHaan and Kochhar the role of managers. Now DeHaan and Kochhar can create tables and views.

If users have multiple roles granted to them, they receive all of the privileges assoicated with all of the roles.



Changing Your Password

The DBA creates an account and initializes a password for every user. You can change your password by using the ALTER USER statement.

Syntax

```
ALTER USER user IDENTIFIED BY password;
```

In the syntax:

user	is the name of the user
password	specifies the new password

Although this statement can be used to change your password, there are many other options. You must have the ALTER USER privilege to change any other option.

For more information, see *Oracle9i SQL Reference*, "ALTER USER."

Object Privileges

Object Privilege	Table	View	Sequence	Procedure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√	√		
SELECT	√	√	√	
UPDATE	√	√		

ORACLE

Object Privileges

An `object privilege` is a privilege or right to perform a particular action on a specific table, view, sequence, or procedure. Each object has a particular set of grantable privileges. The table in the slide lists the privileges for various objects. Note that the only privileges that apply to a sequence are `SELECT` and `ALTER`. `UPDATE`, `REFERENCES`, and `INSERT` can be restricted by specifying a subset of updatable columns. A `SELECT` privilege can be restricted by creating a view with a subset of columns and granting the `SELECT` privilege only on the view. A privilege granted on a synonym is converted to a privilege on the base table referenced by the synonym.

Object Privileges

- **Object privileges vary from object to object.**
- **An owner has all the privileges on the object.**
- **An owner can give specific privileges on that owner's object.**

```
GRANT       object_priv [(columns)]
ON          object
TO          {user|role|PUBLIC}
[WITH GRANT OPTION];
```

ORACLE

Granting Object Privileges

Different object privileges are available for different types of schema objects. A user automatically has all object privileges for schema objects contained in the user's schema. A user can grant any object privilege on any schema object that the user owns to any other user or role. If the grant includes `WITH GRANT OPTION`, then the grantee can further grant the object privilege to other users; otherwise, the grantee can use the privilege but cannot grant it to other users.

In the syntax:

object_priv	is an object privilege to be granted
`ALL`	specifies all object privileges
columns	specifies the column from a table or view on which privileges are granted
`ON` *object*	is the object on which the privileges are granted
`TO`	identifies to whom the privilege is granted
`PUBLIC`	grants object privileges to all users
`WITH GRANT OPTION`	allows the grantee to grant the object privileges to other users and roles

Granting Object Privileges

- **Grant query privileges on the EMPLOYEES table.**

```
GRANT  select
ON     employees
TO     sue, rich;
Grant succeeded.
```

- **Grant privileges to update specific columns to users and roles.**

```
GRANT  update (department_name, location_id)
ON     departments
TO     scott, manager;
Grant succeeded.
```

ORACLE

Guidelines

- To grant privileges on an object, the object must be in your own schema, or you must have been granted the object privileges WITH GRANT OPTION.
- An object owner can grant any object privilege on the object to any other user or role of the database.
- The owner of an object automatically acquires all object privileges on that object.

The first example in the slide grants users Sue and Rich the privilege to query your EMPLOYEES table. The second example grants UPDATE privileges on specific columns in the DEPARTMENTS table to Scott and to the manager role.

If Sue or Rich now want to SELECT data from the employees table, the syntax they must use is:

```
SELECT  *
FROM    scott.employees;
```

Alternatively, they can create a synonym for the table and SELECT from the synonym:

```
CREATE SYNONYM emp FOR scott.employees;
SELECT * FROM emp;
```

Note: DBAs generally allocate system privileges; any user who owns an object can grant object privileges.

Using the WITH GRANT OPTION and PUBLIC Keywords

- Give a user authority to pass along privileges.

```
GRANT  select, insert
ON     departments
TO     scott
WITH   GRANT OPTION;
Grant succeeded.
```

- Allow all users on the system to query data from Alice's DEPARTMENTS table.

```
GRANT  select
ON     alice.departments
TO     PUBLIC;
Grant succeeded.
```

ORACLE

The WITH GRANT OPTION Keyword

A privilege that is granted with the WITH GRANT OPTION clause can be passed on to other users and roles by the grantee. Object privileges granted with the WITH GRANT OPTION clause are revoked when the grantor's privilege is revoked.

The example in the slide gives user Scott access to your DEPARTMENTS table with the privileges to query the table and add rows to the table. The example also allows Scott to give others these privileges.

The PUBLIC Keyword

An owner of a table can grant access to all users by using the PUBLIC keyword.

The second example allows all users on the system to query data from Alice's DEPARTMENTS table.

Confirming Privileges Granted

Data Dictionary View	Description
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
USER_ROLE_PRIVS	Roles accessible by the user
USER_TAB_PRIVS_MADE	Object privileges granted on the user's objects
USER_TAB_PRIVS_RECD	Object privileges granted to the user
USER_COL_PRIVS_MADE	Object privileges granted on the columns of the user's objects
USER_COL_PRIVS_RECD	Object privileges granted to the user on specific columns
USER_SYS_PRIVS	Lists system privileges granted to the user

ORACLE

Confirming Granted Privileges

If you attempt to perform an unauthorized operation (for example, deleting a row from a table for which you do not have the `DELETE` privilege) the Oracle Server does not permit the operation to take place.

If you receive the Oracle Server error message `table or view does not exist`, you have done either of the following:

- Named a table or view that does not exist
- Attempted to perform an operation on a table or view for which you do not have the appropriate privilege

You can access the data dictionary to view the privileges that you have. The chart in the slide describes various data dictionary views.

How to Revoke Object Privileges

- **You use the REVOKE statement to revoke privileges granted to other users.**
- **Privileges granted to others through the WITH GRANT OPTION clause are also revoked.**

```
REVOKE {privilege [, privilege...]|ALL}
ON     object
FROM   {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

ORACLE

Revoking Object Privileges

Remove privileges granted to other users by using the REVOKE statement. When you use the REVOKE statement, the privileges that you specify are revoked from the users you name and from any other users to whom those privileges were granted through the WITH GRANT OPTION clause.

In the syntax:

CASCADE CONSTRAINTS	is required to remove any referential integrity constraints made to the object by means of the REFERENCES privilege

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

Revoking Object Privileges

As user Alice, revoke the SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

```
REVOKE  select, insert
ON      departments
FROM    scott;
Revoke succeeded.
```

ORACLE

Revoking Object Privileges (continued)

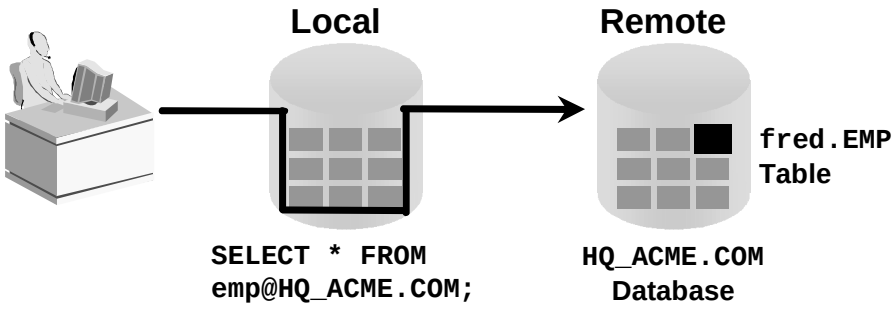
The example in the slide revokes SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

Note: If a user is granted a privilege with the WITH GRANT OPTION clause, that user can also grant the privilege with the WITH GRANT OPTION clause, so that a long chain of grantees is possible, but no circular grants are permitted. If the owner revokes a privilege from a user who granted the privilege to other users, the revoking cascades to all privileges granted.

For example, if user A grants SELECT privilege on a table to user B including the WITH GRANT OPTION clause, user B can grant to user C the SELECT privilege with the WITH GRANT OPTION clause as well, and user C can then grant to user D the SELECT privilege. If user A revokes privilege from user B, then the privileges granted to users C and D are also revoked.

Database Links

A database link connection allows local users to access data on a remote database.



ORACLE

Database Links

A database link is a pointer that defines a one-way communication path from an Oracle database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry.

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, they must define a link that is stored in the data dictionary of database B.

A database link connection gives local users access to data on a remote database. For this connection to occur, each database in the distributed system must have a unique global database name. The global database name uniquely identifies a database server in a distributed system.

The great advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object's owner. In other words, a local user can access a remote database without having to be a user on the remote database.

The example shows a user `SCOTT` accessing the `EMP` table on the remote database with the global name `HQ.ACME.COM`.

Note: Typically, the DBA is responsible for creating the database link. The dictionary view `USER_DB_LINKS` contains information on links to which a user has access.

Database Links

- **Create the database link.**

```
CREATE PUBLIC DATABASE LINK hq.acme.com
USING 'sales';
Database link created.
```

- **Write SQL statements that use the database link.**

```
SELECT *
FROM fred.emp@HQ.ACME.COM;
```

ORACLE

Using Database Links

The example in the slide creates a database link. The `USING` clause identifies the service name of a remote database.

Once the database link is created, you can write SQL statements against the data in the remote site. If a synonym is set up, you can write SQL statements using the synonym.

For example:

```
CREATE PUBLIC SYNONYM HQ_EMP FOR emp@HQ.ACME.COM;
```

Then write a SQL statement that uses the synonym:

```
SELECT * FROM HQ_EMP;
```

You cannot grant privileges on remote objects.

Summary

In this lesson you should have learned about DCL statements that control access to the database and database objects.

Statement	Action
`CREATE USER`	Creates a user (usually performed by a DBA)
`GRANT`	Gives other users privileges to access the your objects
`CREATE ROLE`	Creates a collection of privileges (usually performed by a DBA)
`ALTER USER`	Changes a user's password
`REVOKE`	Removes privileges on an object from users

ORACLE

Summary

DBAs establish initial database security for users by assigning privileges to the users.

- The DBA creates users who must have a password. The DBA is also responsible for establishing the initial system privileges for a user.
- Once the user has created an object, the user can pass along any of the available object privileges to other users or to all users by using the `GRANT` statement.
- A DBA can create roles by using the `CREATE ROLE` statement to pass along a collection of system or object privileges to multiple users. Roles make granting and revoking privileges easier to maintain.
- Users can change their password by using the `ALTER USER` statement.
- You can remove privileges from users by using the `REVOKE` statement.
- With data dictionary views, users can view the privileges granted to them and those that are granted on their objects.
- With database links, you can access data on remote databases. Privileges cannot be granted on remote objects.

Practice 13 Overview

This practice covers the following topics:

- **Granting other users privileges to your table**
- **Modifying another user's table through the privileges granted to you**
- **Creating a synonym**
- **Querying the data dictionary views related to privileges**

ORACLE

Practice 13 Overview

Team up with other students for this exercise about controlling access to database objects.

Practice 13

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

   _____

2. What privilege should a user be given to create tables?

   _____

3. If you create a table, who can pass along privileges to other users on your table?

   _____

4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?

   _____

5. What command do you use to change your password?

   _____

6. Grant another user access to your `DEPARTMENTS` table. Have the user grant you query access to his or her `DEPARTMENTS` table.
7. Query all the rows in your `DEPARTMENTS` table.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

8. Add a new row to your `DEPARTMENTS` table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.
9. Create a synonym for the other team's `DEPARTMENTS` table.

Practice 13 (continued)

10. Query all the rows in the other team's DEPARTMENTS table by using your synonym.

Team 1 SELECT statement results:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
510	Human Resources		
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

9 rows selected.

Team 2 SELECT statement results:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
500	Education		
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

9 rows selected.

Practice 13 (continued)

11. Query the USER_TABLES data dictionary to see information about the tables that you own.

TABLE_NAME
COUNTRIES
DEPARTMENTS
EMPLOYEES
JOBS
JOB_GRADES
JOB_HISTORY
LOCATIONS
REGIONS

8 rows selected.

12. Query the ALL_TABLES data dictionary view to see information about all the tables that you can access. Exclude tables that are you own.

    Note: Your list may not exactly match the list shown below.

TABLE_NAME	OWNER
DEPARTMENTS	***owner***

13. Revoke the SELECT privilege on your table from the other team.