11

Creating Views

Schedule:	Timing	Topic
	20 minutes	Lecture
	20 minutes	Practice
	40 minutes	Total

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

Lesson Aim

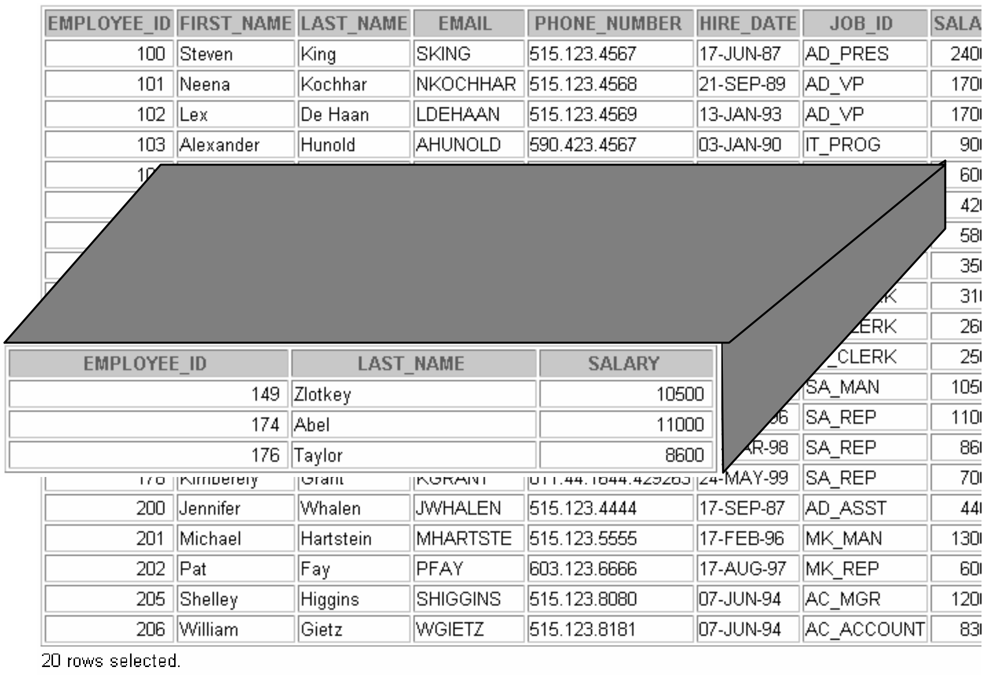
In this lesson, you learn how to create and use views. You also learn to query the relevant data dictionary object to retrieve information about views. Finally, you learn to create and use inline views, and perform Top-N analysis using inline views.

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

What is a View?

EMPLOYEES Table:



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
10							60
							42
							58
							35
						K	31
						ERK	26
						_CLERK	25
						SA_MAN	105
					6	SA_REP	110
					R-98	SA_REP	86
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	83

20 rows selected.

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

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.

Instructor Note

Demo: `11_easyvu.sql`

Purpose: The view shown on the slide is created as follows:

```
CREATE OR REPLACE VIEW simple_vu
AS SELECT employee_id, last_name, salary
   FROM   employees;
```

Why Use Views?

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

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

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.

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 on 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.**

Creating a View (continued)

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

The example on 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.

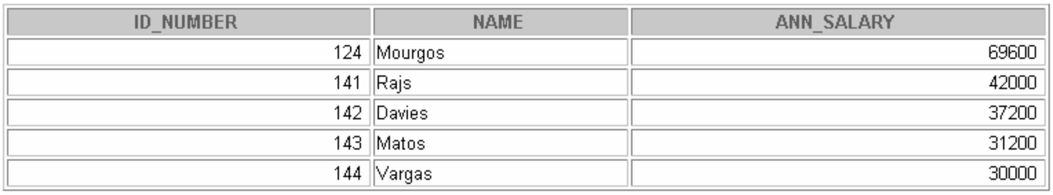
```
CREATE VIEW   salvu50 (ID_NUMBER, NAME, ANN_SALARY)
  AS SELECT  employee_id, last_name, salary*12
     FROM    employees
     WHERE   department_id = 50;
View created.
```

Instructor Note

Let students know about materialized views or snapshots. The terms *snapshot* and *materialized view* are synonymous. Both refer to a table that contains the results of a query of one or more tables, each of which may be located on the same or on a remote database. The tables in the query are called master tables or detail tables. The databases containing the master tables are called the master databases. For more information regarding materialized views refer to: *Oracle9i SQL Reference,* "CREATE MATERIALIZED VIEW / SNAPSHOT."

Retrieving Data from a View

```
SELECT *
FROM  salvu50;
```

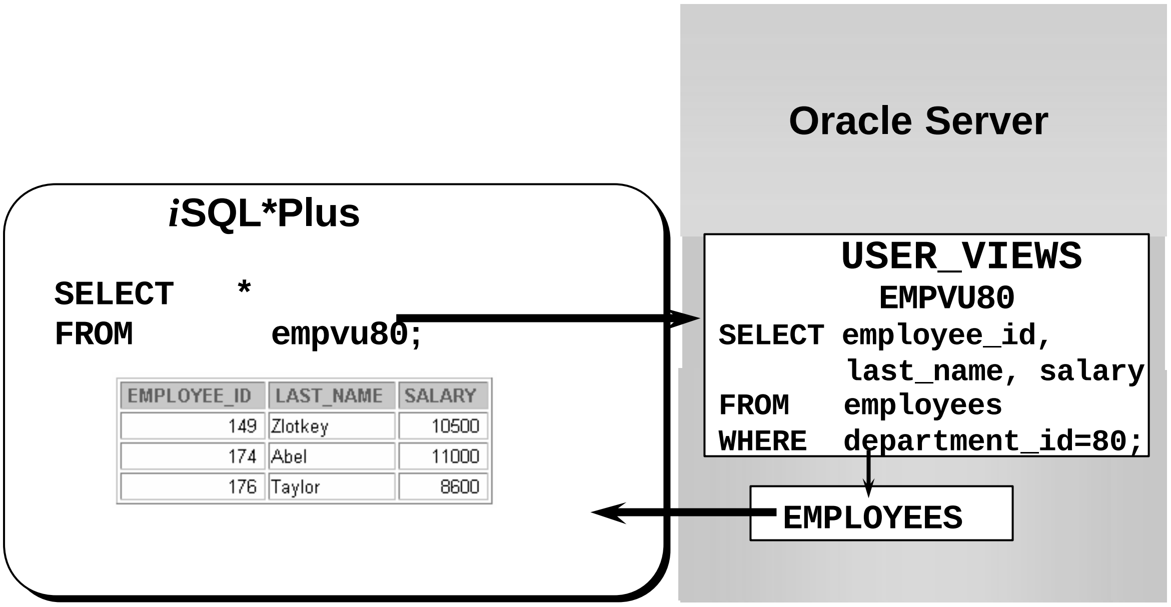


ID_NUMBER	NAME	ANN_SALARY
124	Mourgos	69600
141	Rajs	42000
142	Davies	37200
143	Matos	31200
144	Vargas	30000

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.

Querying a View



```
SELECT   *
FROM     empvu80;
```

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

```
SELECT employee_id,
       last_name, salary
FROM   employees
WHERE  department_id=80;
```

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.

Instructor Note

The view text is stored in a column of `LONG` data type. You may need to set `ARRAYSIZE` to a smaller value or increase the value of `LONG` to view the text.

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.

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.

Instructor Note

The `OR REPLACE` option started with Oracle7. With earlier versions of Oracle, if the view needed to be changed, it had to be dropped and re-created.

Demo: `11_emp.sql`

Purpose: To illustrate creating a view using aliases

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.
```

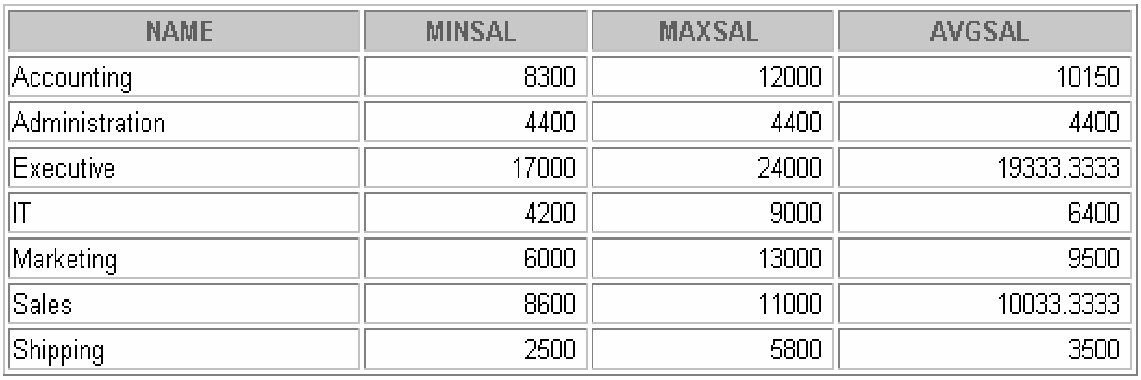
ORACLE

Creating a Complex View

The example on 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**

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

Instructor Note

For each row returned by a query, the `ROWNUM` pseudocolumn returns a number indicating the order in which Oracle server selects the row from a table or set of joined rows. The first row selected has a `ROWNUM` of 1, the second has 2, and so on.

Rules for Performing DML Operations on a View

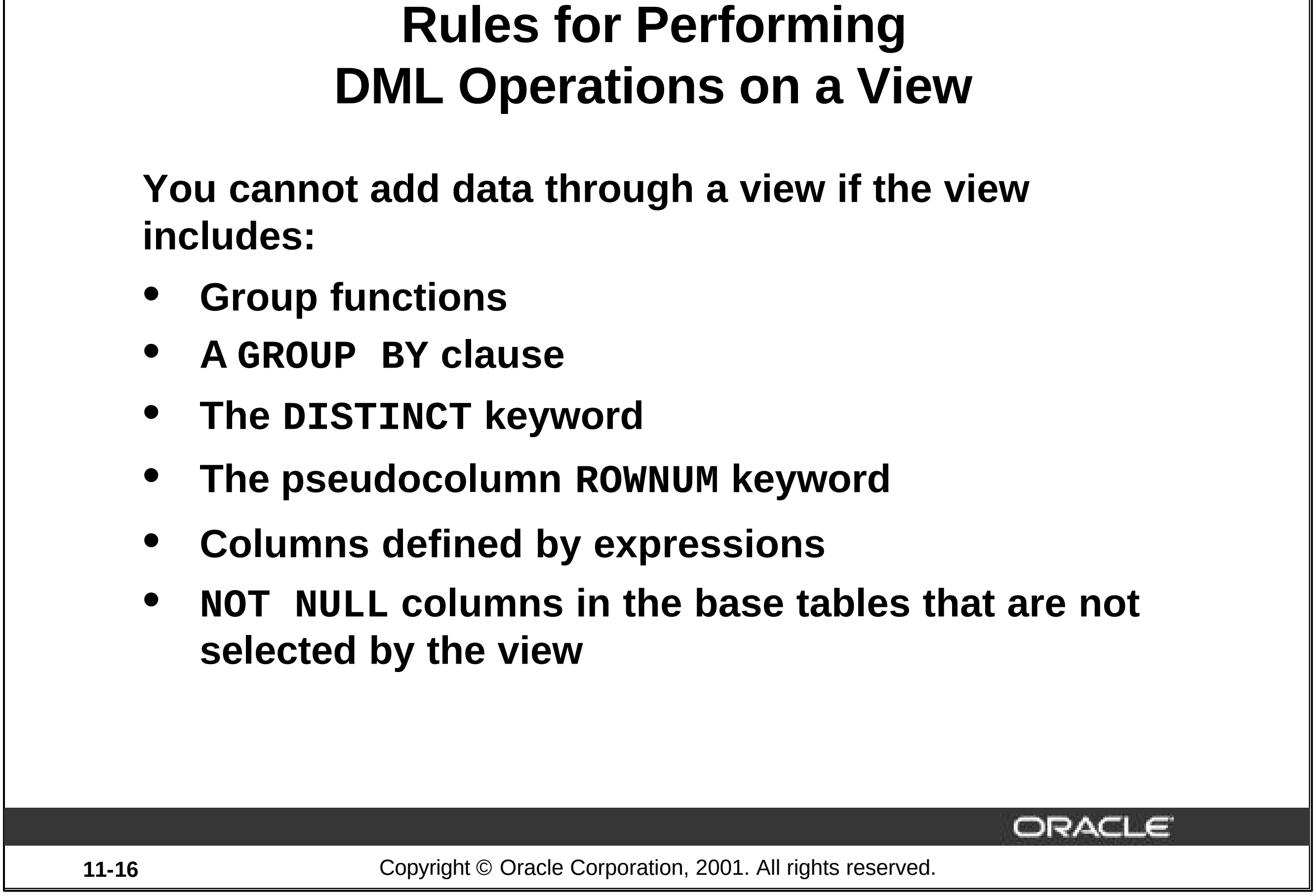
You cannot modify data in a view if it contains:

- **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**

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`."

Instructor Note

With Oracle7.3 and later, you can modify views that involve joins with some restrictions. The restrictions for DML operations described in the slide also apply to join views. Any `UPDATE`, `INSERT`, or `DELETE` statement on a join view can modify only one underlying base table. If at least one column in the subquery join has a unique index, then it may be possible to modify one base table in a join view. You can query `USER_UPDATABLE_COLUMNS` to see whether the columns in a join view can be updated.

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.

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 on the slide modifies the `EMPVU10` view to prevent any DML operations on the view.

Instructor Note (for pages 11-17)

If the user does not supply a constraint name, the system assigns a name in the form `SYS_C`*`n`*, where *`n`* is an integer that makes the constraint name unique within the system.

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.
```

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 attempt to insert a row or modify a row using the view with a read-only constraint results in 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.
```

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

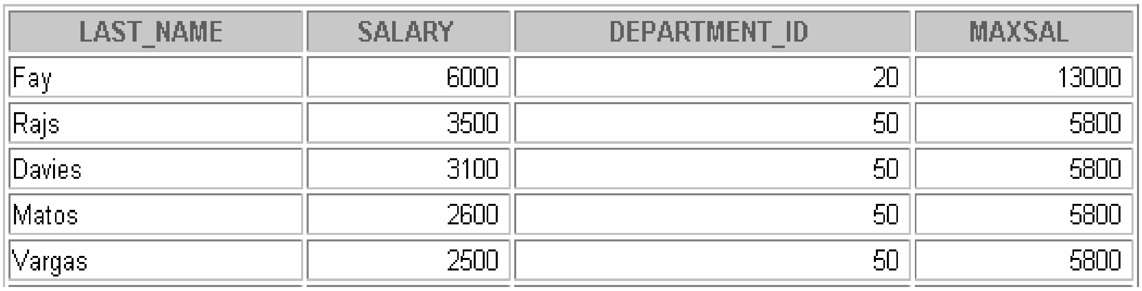
Inline Views

- **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.**

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
Matos	2600	50	5800
Vargas	2500	50	5800

...

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* bottom-most 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

Instructor Note

The capability to include the `ORDER BY` clause in a subquery makes Top-N analysis possible.

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:

1 2 3

```
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

1 2 3

Example of "Top-N" Analysis

The example on 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 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**
- **An inline view is a subquery with an alias name.**
- **Top-N analysis can be done using subqueries and outer queries.**

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

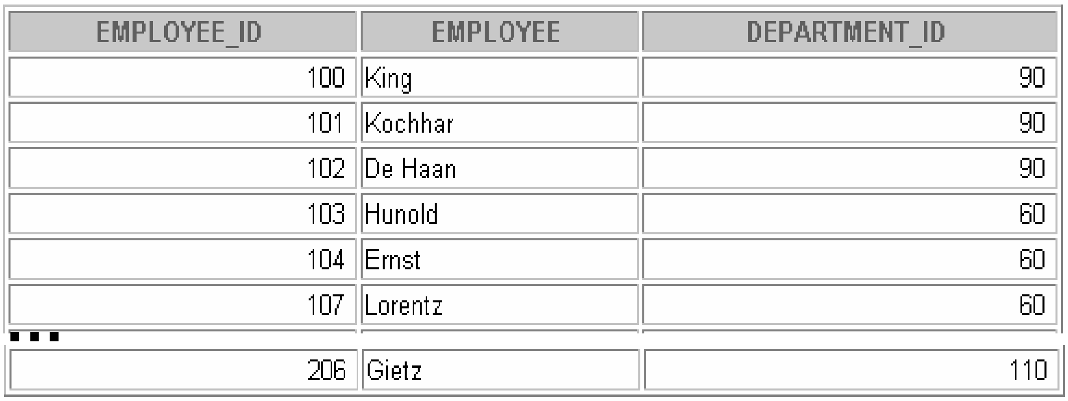
Practice 11 Overview

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.



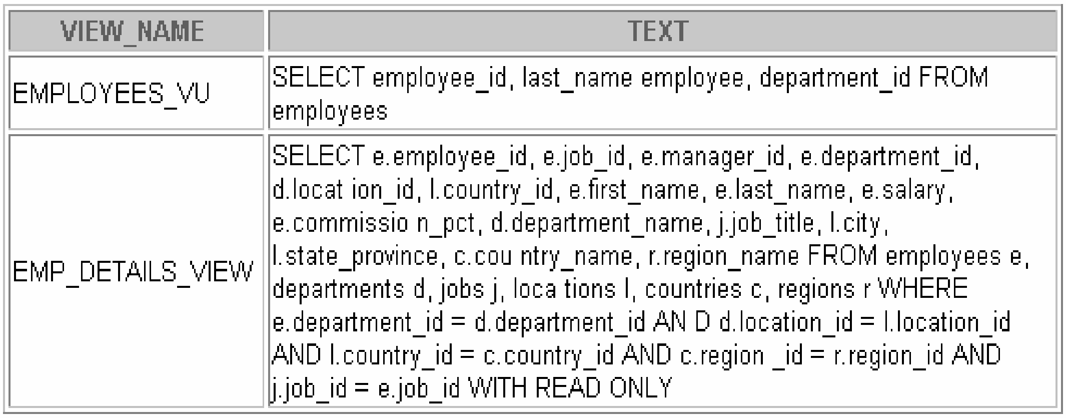
EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60
...		
206	Gietz	110

20 rows selected.

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

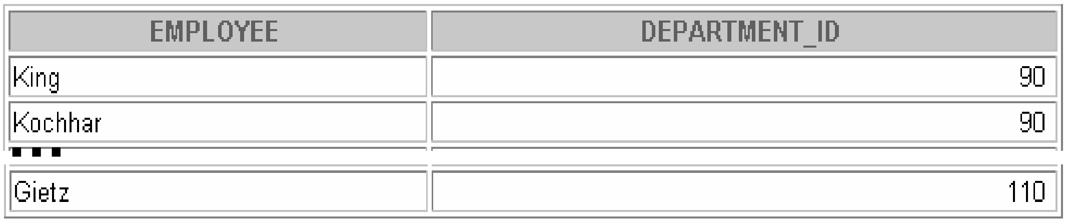
   Note: Another view already exists. The `EMP_DETAILS_VIEW` was created as part of your schema.

   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.locat ion_id, l.country_id, e.first_name, e.last_name, e.salary, e.commissio n_pct, d.department_name, j.job_title, l.city, l.state_province, c.cou ntry_name, r.region_name FROM employees e, departments d, jobs j, loca tions l, countries c, regions r WHERE e.department_id = d.department_id AN D 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

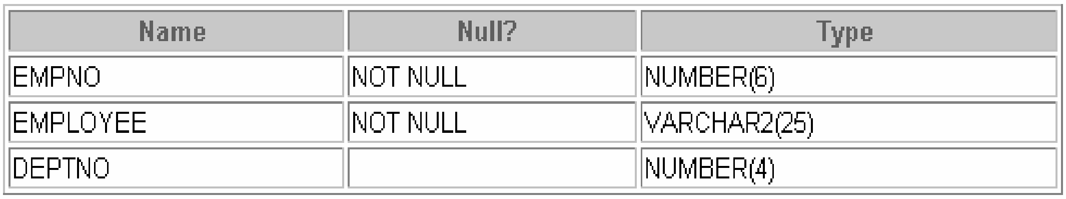
4. Using your `EMPLOYEES_VU` view, enter a query to display all employee names and department numbers.



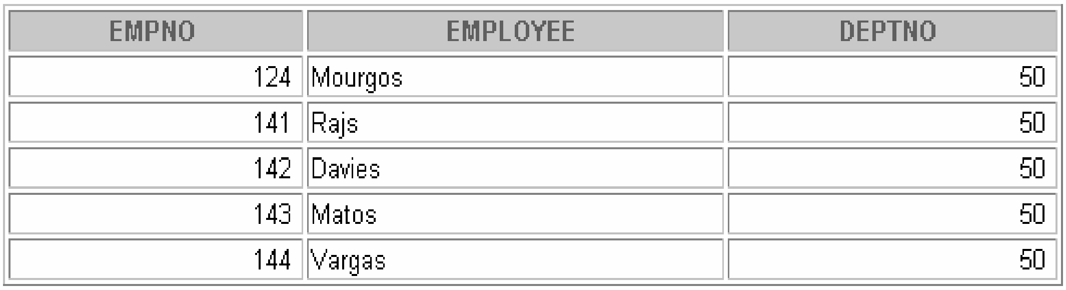
EMPLOYEE	DEPARTMENT_ID
King	90
Kochhar	90
...	
Gietz	110

20 rows selected.

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.