Database Programming with SQL

15-1
Creating Views
Objectives

This lesson covers the following objectives:

• List three uses for views from the standpoint of a database administrator

• Explain, from a business perspective, why it is important to be able to create and use logical subsets of data derived from one or more tables

• Create a view with and without column aliases in the subquery using a single base table
Objectives

This lesson covers the following objectives:

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

• Retrieve data from a view
Purpose

• Take a minute to look back at what you've learned so far as an Oracle Academy student.

• How easy would it be to explain what you know to someone who hasn't taken this class?

• You should pat yourself on the back!

• The level of knowledge you have acquired is understood by only a select few.
Purpose

• Now, imagine yourself as the Database Administrator of a business.

• What do you do when a manager asks you to make it possible for him to be able to retrieve and input data using the company's database?

• "Don't make it too complicated. I just want to be able to prepare reports about all our operations."
Purpose

• Should these employees have access to all of the company's data?
• How will they execute commands that require join conditions?
• Is it wise to allow data input from anyone?
• These are questions that you, as DBA, need to know how to answer.
• In this section, you will learn how to create "views" -- virtual representations of tables customized to meet specific user requirements.
View

- A view, like a table, is a database object.
- However, views are not "real" tables.
- They are logical representations of existing tables or of another view.
- Views contain no data of their own.
- They function as a window through which data from tables can be viewed or changed.
View

- The tables on which a view is based are called "base" tables.
- The view is a query stored as a SELECT statement in the data dictionary.

```
CREATE VIEW view_employees
AS SELECT employee_id emp_id, first_name, last_name, email
FROM employees
WHERE employee_id BETWEEN 100 and 124;
```

```
SELECT *
FROM view_employees;
```

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Steven</td>
<td>King</td>
<td>SKING</td>
</tr>
<tr>
<td>101</td>
<td>Neena</td>
<td>Kochhar</td>
<td>NKOCHHAR</td>
</tr>
<tr>
<td>102</td>
<td>Lex</td>
<td>De Haan</td>
<td>LDEHAAN</td>
</tr>
<tr>
<td>103</td>
<td>Alexander</td>
<td>Hunold</td>
<td>AHUNOLD</td>
</tr>
<tr>
<td>104</td>
<td>Bruce</td>
<td>Ernst</td>
<td>BERNST</td>
</tr>
<tr>
<td>107</td>
<td>Diana</td>
<td>Lorentz</td>
<td>DLORENTZ</td>
</tr>
<tr>
<td>124</td>
<td>Kevin</td>
<td>Mourgos</td>
<td>KMOURGOS</td>
</tr>
</tbody>
</table>
Why Use Views?

• Views restrict access to base table data because the view can display selective columns from the table.

• Views can be used to reduce the complexity of executing queries based on more complicated SELECT statements.

• For example, the creator of the view can construct join statements that retrieve data from multiple tables.

• The user of the view neither sees the underlying code nor how to create it.

• The user, through the view, interacts with the database using simple queries.
Why Use Views?

• Views can be used to retrieve data from several tables, providing data independence for users.

• Users can view the same data in different ways.

• Views provide groups of users with access to data according to their particular permissions or criteria.
Creating a View

• To create a view, embed a subquery within the CREATE VIEW statement.

• The syntax of a view statement is as follows:

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR REPLACE</td>
<td>Re-creates the view if it already exists.</td>
</tr>
<tr>
<td>FORCE</td>
<td>Creates the view whether or not the base tables exist.</td>
</tr>
<tr>
<td>NOFORCE</td>
<td>Creates the view only if the base table exists (default).</td>
</tr>
<tr>
<td>view_name</td>
<td>Specifies the name of the view.</td>
</tr>
<tr>
<td>alias</td>
<td>Specifies a name for each expression selected by the view's query.</td>
</tr>
<tr>
<td>subquery</td>
<td>Is a complete SELECT statement. You can use aliases for the columns in the SELECT list. The subquery can contain complex SELECT syntax.</td>
</tr>
</tbody>
</table>
## Creating a View

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WITH CHECK OPTION</td>
<td>Specifies that rows remain accessible to the view after insert or update operations.</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>Is the name assigned to the CHECK OPTION constraint.</td>
</tr>
<tr>
<td>WITH READ ONLY</td>
<td>Ensures that no DML operations can be performed on this view.</td>
</tr>
</tbody>
</table>
Creating a View

• Example:

```sql
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, region_id, country_name, capitol
    FROM wf_countries
WHERE location LIKE '%Europe';
```

```sql
SELECT * FROM view_euro_countries
ORDER BY country_name;
```

<table>
<thead>
<tr>
<th>COUNTRY_ID</th>
<th>REGION_ID</th>
<th>COUNTRY_NAME</th>
<th>CAPITOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>155</td>
<td>Bailiwick of Guernsey</td>
<td>Saint Peter Port</td>
</tr>
<tr>
<td>203</td>
<td>155</td>
<td>Bailiwick of Jersey</td>
<td>Saint Helier</td>
</tr>
<tr>
<td>387</td>
<td>39</td>
<td>Bosnia and Herzegovina</td>
<td>Sarajevo</td>
</tr>
<tr>
<td>420</td>
<td>151</td>
<td>Czech Republic</td>
<td>Prague</td>
</tr>
<tr>
<td>298</td>
<td>154</td>
<td>Faroe Islands</td>
<td>Torshavn</td>
</tr>
<tr>
<td>49</td>
<td>155</td>
<td>Federal Republic of Germany</td>
<td>Berlin</td>
</tr>
<tr>
<td>33</td>
<td>155</td>
<td>French Republic</td>
<td>Paris</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Guidelines for Creating a View

• The subquery that defines the view can contain complex SELECT syntax.

• The subquery that defines the view should not contain an ORDER BY clause. The ORDER BY clause is specified when you retrieve data from the view.

• You can use the OR REPLACE option to change the definition of the view without having to drop it or re-grant object privileges previously granted on it.

• Aliases can be used for the column names in the subquery.
## CREATE VIEW Features

- Two classifications of views are used: simple and complex.
- The table summarizes the features of each view.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Simple Views</th>
<th>Complex Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tables used to derive data</td>
<td>One</td>
<td>One or more</td>
</tr>
<tr>
<td>Can contain functions</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can contain groups of data</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can perform DML operations (INSERT, UPDATE, DELETE) through a view</td>
<td>Yes</td>
<td>Not always</td>
</tr>
</tbody>
</table>
Simple View

• The view shown below is an example of a simple view.

• The subquery derives data from only one table and it does not contain a join function or any group functions.

• Because it is a simple view, INSERT, UPDATE, DELETE, and MERGE operations affecting the base table could possibly be performed through the view.

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, country_name, capitol
    FROM wf_countries
WHERE location LIKE '%Europe';
```
Simple View

- Column names in the SELECT statement can have aliases as shown below.

- Note that aliases can also be listed after the CREATE VIEW statement and before the SELECT subquery.

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id AS "ID", country_name AS "Country",
        capitol AS "Capitol City"
FROM wf_countries
WHERE location LIKE '%Europe';
```

```
CREATE OR REPLACE VIEW view_euro_countries("ID", "Country",
        "Capitol City")
AS SELECT country_id, country_name, capitol
FROM wf_countries
WHERE location LIKE '%Europe';
```
Simple View

• It is possible to create a view whether or not the base tables exist.

• Adding the word FORCE to the CREATE VIEW statement creates the view.

• As a DBA, this option could be useful during the development of a database, especially if you are waiting for the necessary privileges to the referenced object to be granted shortly.

• The FORCE option will create the view despite it being invalid.

• The NOFORCE option is the default when creating a view.
Complex View

- Complex views are views that can contain group functions and joins.
- The following example creates a view that derives data from two tables.

```
CREATE OR REPLACE VIEW view_euro_countries
("ID", "Country", "Capitol City", "Region")
AS SELECT c.country_id, c.country_name, c.capitol, r.region_name
    FROM wf_countries c JOIN wf_world_regions r
    USING (region_id)
    WHERE location LIKE '%Europe';

SELECT *
FROM view_euro_countries;
```
Complex View

<table>
<thead>
<tr>
<th>ID</th>
<th>Country</th>
<th>Capitol City</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>355</td>
<td>Republic of Albania</td>
<td>Tirana</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>385</td>
<td>Republic of Croatia</td>
<td>Zagreb</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>359</td>
<td>Republic of Bulgaria</td>
<td>Sofia</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>378</td>
<td>Republic of San Marino</td>
<td>San Marino</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>34</td>
<td>Kingdom of Spain</td>
<td>Madrid</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>38</td>
<td>The Holy See (State of the Vatican City)</td>
<td>Vatican City</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>30</td>
<td>Hellenic Republic</td>
<td>Athens</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>386</td>
<td>Republic of Slovenia</td>
<td>Ljubljana</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>381</td>
<td>Republic of Montenegro</td>
<td>Cetinje</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>351</td>
<td>Portuguese Republic</td>
<td>Lisbon</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>376</td>
<td>Principality of Andorra</td>
<td>Andorra la Vella</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>350</td>
<td>Gibraltar</td>
<td>Gibraltar</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>387</td>
<td>Bosnia and Herzegovina</td>
<td>Sarajevo</td>
<td>Southern Europe</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Complex View

- Group functions can also be added to complex-view statements.

```sql
CREATE OR REPLACE VIEW view_high_pop
    ("Region ID", "Highest population")
AS SELECT region_id, MAX(population)
    FROM wf_countries
    GROUP BY region_id;

SELECT * FROM view_high_pop;
```

<table>
<thead>
<tr>
<th>Region ID</th>
<th>Highest population</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>1095351995</td>
</tr>
<tr>
<td>151</td>
<td>142893540</td>
</tr>
<tr>
<td>30</td>
<td>1313973713</td>
</tr>
<tr>
<td>13</td>
<td>107449525</td>
</tr>
<tr>
<td>11</td>
<td>131859731</td>
</tr>
<tr>
<td>29</td>
<td>11382820</td>
</tr>
<tr>
<td>155</td>
<td>82422299</td>
</tr>
<tr>
<td>21</td>
<td>298444215</td>
</tr>
<tr>
<td>14</td>
<td>74777981</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Modifying a View

• To modify an existing view without having to drop then re-create it, use the OR REPLACE option in the CREATE VIEW statement.

• The old view is replaced by the new version.

• For example:

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, region_id, country_name, capitol
    FROM wf_countries
WHERE location LIKE '%Europe';
```
Terminology

Key terms used in this lesson included:

• Alias
• Complex review
• CREATE VIEW
• FORCE
• NOFORCE
• REPLACE
Terminology

Key terms used in this lesson included:

• Simple view
• Subquery
• View
• VIEW_NAME
Summary

In this lesson, you should have learned how to:

• List three uses for views from the standpoint of a database administrator

• Explain, from a business perspective, why it is important to be able to create and use logical subsets of data derived from one or more tables

• Create a view with and without column aliases in the subquery using a single base table
Summary

In this lesson, you should have learned how to:

• Create a complex view that contains group functions to display values from two tables
• Retrieve data from a view