



AROR UNIVERSITY  
OF ART, ARCHITECTURE,  
DESIGN & HERITAGE,  
SUKKUR, SINDH

**Department of Artificial Intelligence & Multimedia Gaming**  
**CSC-207: Database Systems**

**Lab # 11: To Work with SQL Views and Indexes**

**Objectives**

1. Introduction to Views in SQL.
2. Creating Views in SQL.
3. Modifying Views in SQL.
4. Creating a complex view in SQL.
5. Drop a View in SQL.
6. Rules for performing DML operations in Views.
7. Introduction to Index in SQL.
8. Creating Index in SQL.
9. Removing Index in SQL.

**Introduction to Views in SQL**

- A view is a virtual table that does not physically exist. Rather, it is created by a query joining one or more tables.
- 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.
- Views are very powerful and handy since they can be treated just like any other table but do not occupy the space of a table.
- A view is a virtual table that is based on the result set of a SELECT query.
- Views do not store data themselves but instead provide a way to present data from one or more tables or other views in a structured format.
- Views can be used to simplify complex queries, hide the complexity of underlying tables,

and provide a layer of security by restricting access to certain columns or rows of data.

- Views can be queried and manipulated like tables, but any changes made to the data through a view affect the underlying tables.
- They are useful for providing a predefined and consistent way to access and present data to users or applications without exposing the underlying database structure.

There are two classification for views:

- **simple and complex:** The basic difference is related to the DML 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 Views in SQL.

Here is syntax for a simple view for your existing table.

```
CREATE VIEW view_name AS
```

```
SELECT column1, column2
```

```
FROM table_name
```

```
WHERE condition;
```

### Example:

```
CREATE VIEW employee_names AS
SELECT
    employee_id,
    first_name,
    last_name
FROM
    employees;
```

```
8 • SELECT * FROM employee_names;
```

Result Grid    Filter Rows: <input type="text"/>		
employee_id	first_name	last_name
100	Steven	King
101	Neena	Kochhar
102	Lex	De Haan
103	Alexander	Hunold
104	Bruce	Ernst
105	David	Austin
106	Valli	Pataballa
107	Diana	Lorentz
108	Nancy	Greenberg

```
CREATE VIEW employee_names AS
SELECT
    employee_id, first_name, last_name
FROM
    employees;
```

```
SELECT * FROM employee_names;
```

**Example:** Create VIEW that contains details of employees in deptno 10.

```
271 • CREATE VIEW EMPVU10
272 AS SELECT employee_id, first_name, department_id
273 FROM employees
274 WHERE department_id=10;
275
276 • SELECT * FROM EMPVU10;
277
278
```

Result Grid    Filter Rows: <input type="text"/>   Export:    Wrap Cell Co			
	employee_id	first_name	department_id
▶	200	Jennifer	10

```
CREATE VIEW EMPVU10
AS SELECT employee_id, first_name, department_id
FROM employees WHERE department_id=10;
```

```
SELECT * FROM EMPVU10;
```

## Modifying Views in SQL.

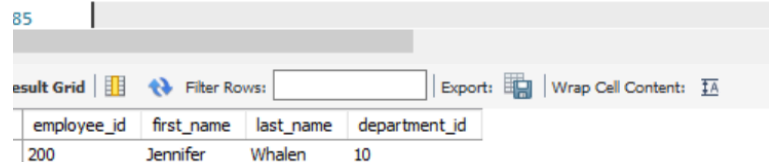
- Note: OR REPLACE option allows a view to be created even if one exists with this name already. Thus, replacing the old version of the view.

### Example:

```
CREATE OR REPLACE VIEW EMPVU10
AS SELECT employee_id, first_name, last_name, department_id
FROM employees WHERE department_id=10;
```

```
SELECT * FROM EMPVU10;
```

```
78 • CREATE OR REPLACE VIEW EMPVU10
79 AS SELECT employee_id, first_name, last_name, department_id
80 FROM employees
81 WHERE department_id=10;
82
83 • SELECT * FROM EMPVU10;
84
85
```



The screenshot shows a SQL IDE interface. The top part displays the SQL code entered in the previous blocks. Below the code, there is a toolbar with options like 'Filter Rows', 'Export', and 'Wrap Cell Content'. At the bottom, a table titled 'result Grid' shows the output of the SELECT statement. The table has four columns: employee\_id, first\_name, last\_name, and department\_id. The first row of data shows employee\_id 200, first\_name Jennifer, last\_name Whalen, and department\_id 10.

employee_id	first_name	last_name	department_id
200	Jennifer	Whalen	10

Note: the ALTER VIEW statement allows you to redefine the underlying SELECT query of an existing view without dropping and recreating the view.

### Example:

```
ALTER VIEW EMPVU10 AS
SELECT
    employee_id, first_name,
    last_name, -- Adding the last_name column
    phone_number, -- Adding the phone_number column
    department_id
FROM
    employees
WHERE
    department_id = 10;
```

```
SELECT * FROM EMPVU10;
```

**Example:**

```
create view job as select
```

```
job_id, job_title, min_salary from jobs;
```




```
select * from job;
```

## Creating a complex view in SQL.

Simple views are typically based on a single table, have straightforward SELECT queries, and allow direct DML operations. On the other hand, complex views involve multiple tables, may include functions or aggregations, and may have limitations on DML operations due to their complexity.

**Example:**

```
302 CREATE VIEW dept_sum_vu
303 AS SELECT d.department_name, MIN(e.salary), MAX(e.salary), AVG(e.salary)
304 FROM employees e, departments d
305 WHERE e.department_id= d.department_id
306 GROUP BY d.department_name;
307 • select * from dept_sum_vu;
308
```

Result Grid    Filter Rows: <input type="text"/>   Export:    Wrap Cell Content: 				
department_name	MIN(e.salary)	MAX(e.salary)	AVG(e.salary)	
Accounting	8300.00	12000.00	10150.000000	
Administration	4400.00	4400.00	4400.000000	
Executive	17000.00	24000.00	19333.333333	
Finance	6900.00	12000.00	8600.000000	
Human Resources	6500.00	6500.00	6500.000000	

## Rules for performing DML operations in Views.

You can perform DML operations on simple views

You can not remove a row if the view contains the following:

- Group functions
- GROUP by clause
- DISTINCT keyword

**Example:**

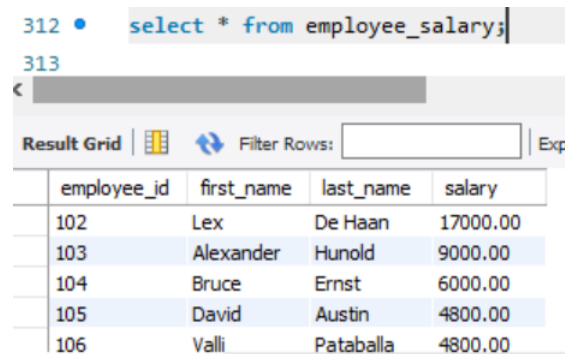
```
CREATE VIEW employee_salary AS
SELECT
    employee_id,
    first_name,
    last_name,
    salary
FROM
    employees;
```

```
CREATE VIEW employee_salary AS
SELECT
    employee_id,
    first_name,
    last_name,
    salary
FROM
    employees;
```

- This view selects the employee\_id, first\_name, last\_name, and salary columns from the employees table.

312 • `select * from employee_salary;`

313

< 

employee_id	first_name	last_name	salary
102	Lex	De Haan	17000.00
103	Alexander	Hunold	9000.00
104	Bruce	Ernst	6000.00
105	David	Austin	4800.00
106	Valli	Pataballa	4800.00

- Now, let's say we want to delete a row from this view. Since it's a simple view that selects data directly from the employees table without any aggregation or complexity, we can perform DELETE operations on this view:
- `DELETE FROM employee_salary WHERE employee_id = 110;`

Also perform update.

### Example:

```
UPDATE employee_salary
SET salary = 55000
```

```
WHERE employee_id = 120;
```

```
select first_name , salary from employee_salary where employee_id = 120;
```

Note: We cannot directly use insert to add values using Views.

```
-- Create the departments table
```

```
CREATE TABLE departinfo (  
    department_id INT(11) UNSIGNED PRIMARY KEY,  
    department_name VARCHAR(50) NOT NULL,  
    location VARCHAR(50)  
);
```

```
-- Insert sample values into the departments table
```

```
INSERT INTO departinfo (department_id, department_name, location)  
VALUES  
(1, 'Sales', 'New York'),  
(2, 'Marketing', 'Los Angeles'),  
(3, 'Finance', 'Chicago'),  
(4, 'Human Resources', 'Houston');
```

```
-- Check the inserted values
```

```
SELECT * FROM departinfo;
```

```
create view deptview as select department_id, department_name, location from departinfo;
```

```
select * from deptview;
```

```
delete from deptview where department_id=4;
```

```
INSERT INTO deptview (department_id, department_name, location)  
VALUES
```

```
(7, 'Sales', 'New York');
```

```
update deptview set department_name= 'Inventory' where department_id=7;
```

## DROP View in SQL.

### Example:

DROP VIEW viewname;

## Introduction to Index in SQL.

- Is used by the oracle, MySQL server to speed up the retrieval of rows by using a pointer.
- Is used and maintained automatically by the Oracle and MySQL server. Once an index is created, no direct activity is required by the user.
- Indexes are physically and logically independent of the table they index. This means they can be created or dropped at any time and have no effect on the base tables or other indexes.
- Two types of indexes can be created .
- One type is a **unique index**: the Oracle server automatically creates this index when you define a column in a table to have a PRIMARY KEY or UNIQUE key constraint. The name of the index is the name given to the constraint.
- The other type of index is a **non unique index**, which a user can create.

## Creating Index in SQL.

### Example:

```
CREATE INDEX emp_ename_idx
```

```
ON emp (ename);
```

Example:

```
CREATE INDEX idx_department_id ON employees (department_id);
```

```
CREATE INDEX idx_department_id ON employees (department_id);
```

This statement creates a new index named `idx_department_id` on the `department_id` column of the `employees` table. The index will allow the database system to quickly locate rows in the `employees` table based on their `department_id` values, improving query performance for queries that involve filtering or sorting by `department_id`.

You can also create composite indexes that involve multiple columns. For example, if you frequently query the `employees` table based on both `department_id` and `last_name`, you can create a composite index:

### Example:



```
CREATE INDEX idx_department_id_last_name ON employees (department_id, last_name);
```

## Removing Index in SQL.

### Example:

```
DROP INDEX idx_department_id ON employees;
```

## Exercises (Class)

1. Add here all the tasks performed in lab.

## Exercises (Weekly)

1. Create a view call employee\_vu based on the employee numbers, employee names and department numbers from the emp table. Change the heading for the ename to EMPLOYEE.
2. Display the contents of the employee\_vu view.
3. Create a view called SALARY\_VU based on the employee names, salaries and salary grades for all employees. Use the EMP and SALGRADE tables. Label the columns Employee, salary and grade respectively.
4. Create a view named DEPT10 that contains the emp numbers, enames and department numbers for all employees in department 10. Do not allow an employee to be reassigned to another department through the view.
5. Create an index on the deptno of the Emp table. Drop the newly created index.