1. Introduction

Imagine a university database contains a `students` table with many columns:

- Student ID
- Student Name
- Father Name
- CNIC
- Email
- Phone Number
- Department
- Program
- Admission Date

Sometimes, a university employee only needs to see the student's name, ID, and department.

Instead of writing the same SELECT query again and again, we can create a View.

A view allows us to save a SQL query and use it like a virtual table.

2. What is a View in PostgreSQL?

A View is a `virtual table` based on the result of a `SQL query`.

It does not normally store the actual data separately. Instead, PostgreSQL executes the underlying query when we access the view.

Example:

Suppose we frequently need to display student names and departments.

Instead of writing:

```
SELECT student_id, student_name, department
FROM students;
```

every time, we can create a view named `student_details`.

Then we can simply write:

```
SELECT * FROM student_details;
```

3. Why Do We Use Views?

Views are useful for several reasons:

1. Simplify queries: Save complex SQL queries for repeated use.
2. Data presentation: Display only the required columns.
3. Security: Restrict access to selected columns or rows through appropriate permissions.

4. Reusability: Use the same query from different applications or reports.
5. Maintainability: Make frequently used queries easier to manage.

4. Create a Sample Table

First, create a table named `students` with the following columns:

Column	Data Type	Description
student_id	SERIAL	Unique student ID
student_name	VARCHAR(100)	Student's full name
department	VARCHAR(100)	Department name
marks	NUMERIC(5,2)	Student marks
semester	INTEGER	Current semester

Execute the following SQL query:

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    department VARCHAR(50),
    semester INT,
    marks INT
);
```

Insert sample records:

```
INSERT INTO students
    (student_id, student_name, department, semester, marks)
VALUES
    (1, 'Ali', 'Computer Science', 1, 85),
    (2, 'Sara', 'Mathematics', 2, 72),
    (3, 'Ahmed', 'Computer Science', 3, 65),
    (4, 'Usman', 'Physics', 1, 45),
    (5, 'Ayesha', 'Mathematics', 4, 90);
```

Display all records:

```
SELECT * FROM students;
```

5. Creating a View

Let's create a view that displays student names and their departments.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name;
```

Example:

```
CREATE VIEW student_details AS
SELECT
    student_id,
    student_name,
    department
FROM students;
```

The view `student_details` has been created.

6. Displaying Data from a View

We can retrieve data from a view using a normal SELECT statement.

```
SELECT * FROM student_details;
```

Expected output:

student_id	student_name	department
1	Ali	Computer Science
2	Sara	Mathematics
3	Ahmed	Computer Science
4	Usman	Physics
5	Ayesha	Mathematics

Important: We use a view in a SELECT statement just like a table.

7. Creating a View with WHERE

We can create a view using a WHERE condition.

For example, create a view that displays only Computer Science students.

```
CREATE VIEW cs_students AS
SELECT
    student_id,
    student_name,
    semester,
    marks
```

```
FROM students
WHERE department = 'Computer Science';
```

Display the view:

```
SELECT * FROM cs_students;
```

Expected output:

student_id	student_name	semester	marks
1	Ali	1	85
3	Ahmed	3	65

This view displays only students belonging to the Computer Science department.

8. Creating a View with ORDER BY

We can also use ORDER BY in a view definition.

```
CREATE VIEW students_by_marks AS
SELECT
    student_id,
    student_name,
    marks
FROM students
ORDER BY marks DESC;
```

Retrieve the records:

```
SELECT * FROM students_by_marks;
```

Note: If the order matters, specify `ORDER BY` in the SELECT statement that retrieves data from the view. Do not rely on a view's stored ordering.

For example:

```
SELECT *
FROM students_by_marks
ORDER BY marks DESC;
```

9. Using Aggregate Functions in a View

Views can also contain aggregate functions.

For example, create a view that shows the average marks of each department.

```sql
CREATE VIEW department_average AS
SELECT
    department,
    AVG(marks) AS average_marks
FROM students
GROUP BY department;
```

Display the view:

```sql
SELECT * FROM department_average;
```

Expected output:

department	average_marks
Computer Science	75
Mathematics	81
Physics	45

We can use a view to store the definition of a report query and retrieve its results whenever required.

10. Using WHERE with a View

We can apply conditions when retrieving data from a view.

For example, display Computer Science students who have marks above 70.

```sql
SELECT *
FROM student_details
WHERE department = 'Computer Science';
```

Another example:

```sql
SELECT *
FROM student_details
WHERE student_id > 2;
```

Remember: We can use WHERE, ORDER BY, GROUP BY (where appropriate), and other SQL features with views just as we do with tables.

11. Updating Data Through a View

Some views allow us to insert, update, and delete records.

For example, our `student_details` view is based on a single table and contains simple columns.

We can update a student's department through this view:

```
UPDATE student_details
SET department = 'Information Technology'
WHERE student_id = 1;
```

Check the original table:

```
SELECT * FROM students;
```

The department of student Ali will also be updated in the original `students` table.

Important: A simple view based on one table is generally updatable. Views containing GROUP BY, aggregate functions, DISTINCT, UNION, or certain other operations are generally not directly updatable.

12. Replacing an Existing View

Suppose we want to add the semester column to our existing `student_details` view.

We can use `CREATE OR REPLACE VIEW`.

Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2
FROM table_name;
```

Example:

```
CREATE OR REPLACE VIEW student_details AS
SELECT
    student_id,
    student_name,
    department,
    semester
FROM students;
```

Now retrieve the updated view:

```
SELECT * FROM student_details;
```

The view now displays the semester column as well.

Note: When replacing a view, existing columns must retain their names, order, and data types. New columns can generally be added at the end.

13. Renaming a View

We can rename an existing view using `ALTER VIEW`.

Syntax:

```
ALTER VIEW old_view_name
RENAME TO new_view_name;
```

Example:

```
ALTER VIEW student_details
RENAME TO university_student_details;
```

Now use the new view name:

```
SELECT * FROM university_student_details;
```

14. Deleting a View

We can delete a view using `DROP VIEW`.

Syntax:

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW university_student_details;
```

To avoid an error if the view does not exist:

```
DROP VIEW IF EXISTS university_student_details;
```

Important: Dropping a view does not delete the original table or its records.

15. Views in PostgreSQL Using the DVD Rental Database

16. View vs Table

View	Table
Virtual table based on a query	Stores actual data
Usually does not store data separately	Stores data in the database

View	Table
Created using CREATE VIEW	Created using CREATE TABLE
Displays data from underlying tables	Contains its own records
Can simplify complex queries	Used to store and manage data

17. Important PostgreSQL Commands

Task	SQL Command
Create a view	`CREATE VIEW`
Display view data	`SELECT`
Replace a view	`CREATE OR REPLACE VIEW`
Rename a view	`ALTER VIEW ... RENAME TO`
Delete a view	`DROP VIEW`
Create a materialized view	`CREATE MATERIALIZED VIEW`
Refresh a materialized view	`REFRESH MATERIALIZED VIEW`

20. Quick Revision

- A view is a virtual table based on a SQL query.
- `CREATE VIEW` creates a view.
- `SELECT` retrieves data from a view.
- `CREATE OR REPLACE VIEW` modifies its definition.
- `ALTER VIEW` can rename a view.
- `DROP VIEW` deletes a view.

Remember: A view saves a query definition, making it easier to reuse SQL queries without writing them again.