VIEWS

* Result of query stored as virtual relation and available for users

↳ what's the use case?

1. Access Control for Sensitive Information

- **Use Case**: A company's HR database has a `Employees` table with sensitive columns like `salary`, `personal_info`, and `SSN`.
- **Solution**: Create a view that omits these sensitive columns and grants access to departments or personnel who need to view basic employee information without accessing sensitive data.
- **Example**:

sql Copy code

```sql
CREATE VIEW public_employee_info AS
SELECT emp_id, name, department, position
FROM Employees;
```

2. Simplifying Complex Joins in Reporting

- **Use Case**: A retail business needs monthly sales reports that combine data from `Sales`, `Customers`, and `Products` tables.
- **Solution**: Create a view that joins these tables, calculating totals and aggregating data by month. This view allows the analytics team to query monthly sales data directly, without needing to perform complex joins each time.
- **Example**:

sql Copy code

```sql
CREATE VIEW monthly_sales AS
SELECT p.product_name, c.customer_region, SUM(s.amount) AS total_sales, DATE_TRUNC('mo
FROM Sales s
JOIN Customers c ON s.customer_id = c.customer_id
JOIN Products p ON s.product_id = p.product_id
GROUP BY p.product_name, c.customer_region, sale_month;
```

* CREATE VIEW

AS < Any SQL Query >

View

Simple ⇓ Simple Queries

Complex ⇓ Queries with joins etc.

Materialized ⇓ Views that are stored in DB ⇓ Some database provider

⇓ Dynamic/on demand

1. Simple View

- **Definition**: A view that draws data from a single table without using functions, grouping, or advanced operations.
- **Purpose**: Provides a straightforward way to limit access to certain columns or rows.
- **Example**:

```sql
CREATE VIEW simple_employee_view AS
SELECT emp_id, name
FROM Employees
WHERE department = 'HR';
```

2. Complex View

- **Definition**: A view that pulls data from multiple tables and may include `JOINs`, aggregate functions, grouping, and advanced filtering.
- **Purpose**: Used for more sophisticated queries to aggregate or transform data, often for reporting and analytics.
- **Example**:

```sql
CREATE VIEW employee_sales_summary AS
SELECT e.emp_id, e.name, SUM(s.amount) AS total_sales
FROM Employees e
JOIN Sales s ON e.emp_id = s.emp_id
GROUP BY e.emp_id, e.name;
```

Certain database systems allow view relations to be stored, but they make sure that, if the actual relations used in the view definition change, the view is kept up-to-date. Such views are called **materialized views**.

3. **Materialized View** (available in some databases, like PostgreSQL, Oracle)

- **Definition**: A view that stores the query result physically in the database, unlike regular views that retrieve data dynamically.
- **Purpose**: Improves performance for complex, resource-heavy queries, as the data is stored and only needs to be refreshed periodically.
- **Example**:

sql

Copy code

```sql
CREATE MATERIALIZED VIEW monthly_sales_report AS
SELECT product_id, SUM(amount) AS monthly_sales
FROM Sales
WHERE sale_date >= DATE_TRUNC('month', CURRENT_DATE)
GROUP BY product_id;
```

The process of keeping the materialized view up-to-date is called **materialized view maintenance** (or often, just **view maintenance**) and is covered in Section 13.5. View maintenance can be done immediately when any of the relations on which the view is defined is updated. Some database systems, however, perform view maintenance lazily, when the view is accessed. Some systems update materialized views only periodically; in this case, the contents of the materialized view may be stale, that is, not up-to-date, when it is used, and should not be used if the application needs up-to-date data. And some database systems permit the database administrator to control which of the above methods is used for each materialized view.