Yogesh Tyagi
@ytyagi782

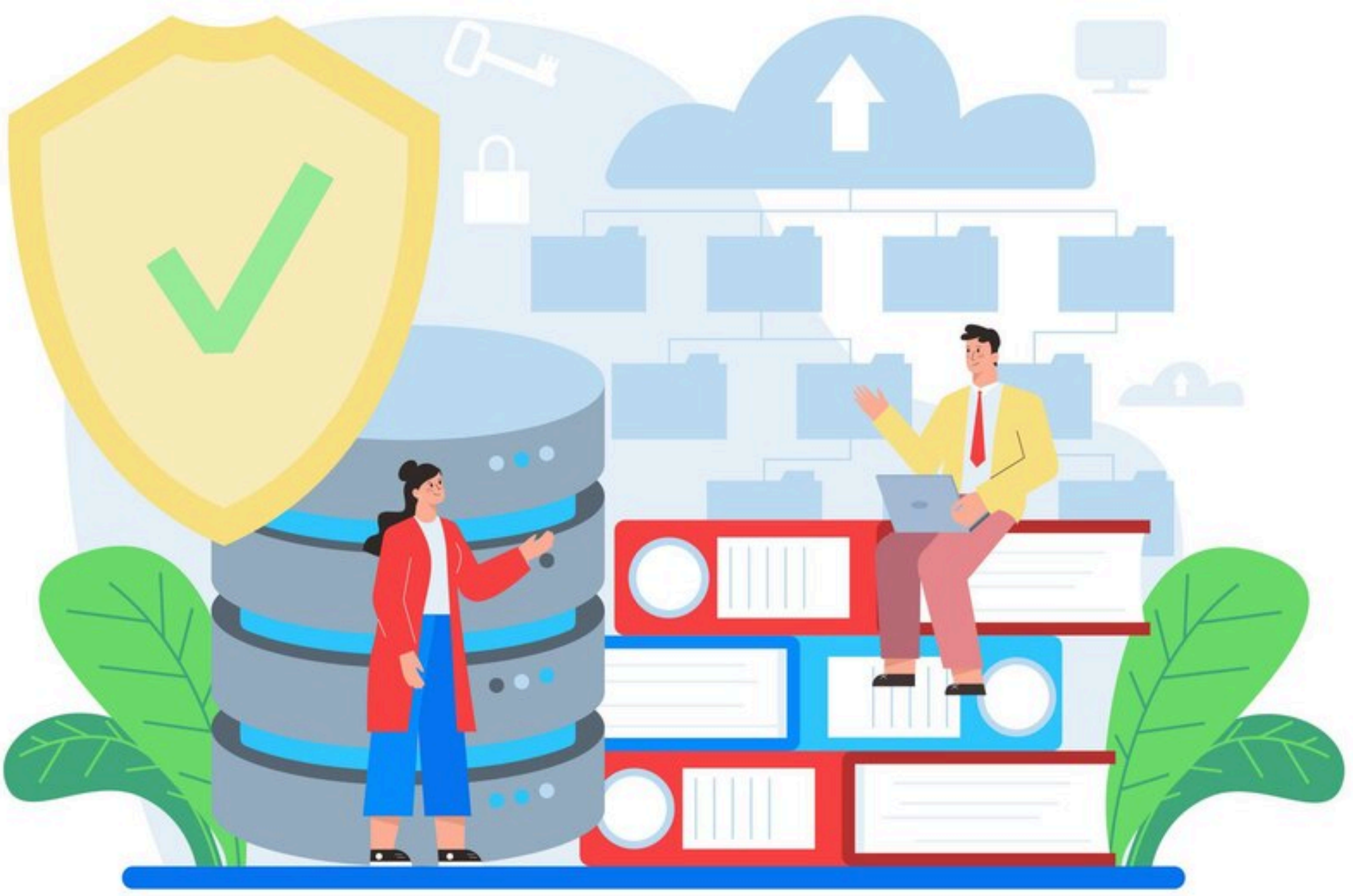
SQL Views Guide

The Key to Query Simplification


Yogesh Tyagi
@ytyagi782

SQL Views: The Key to Query Simplification

Learn how SQL Views help create reusable, efficient, and secure database query structures.



Yogesh Tyagi
@ytyagi782

What is a SQL View?

A view is a virtual table representing the result of a query. It does not store data itself but simplifies complex queries.

Examples:

```sql
CREATE VIEW employee_details AS
SELECT emp_id, name, department FROM
employees;
```

Querying the view

```sql
SELECT * FROM employee_details WHERE
department = 'HR';
```

Yogesh Tyagi
@ytyagi782

Why Use SQL Views?

1. Simplify complex queries.
2. Enhance data security by exposing only required columns.
3. Improve code reusability and consistency across applications.

Examples:

A view hides sensitive data like salaries:

```
CREATE VIEW public_employees AS
SELECT emp_id, name, department FROM
employees;
```


Yogesh Tyagi
@ytyagi782

Types of Views

Simple Views

Derived from a single table.

```sql
CREATE VIEW dept_employees AS
SELECT name, department FROM
employees;
```

Complex Views

Derived from multiple tables using joins or subqueries.

```sql
CREATE VIEW sales_summary AS
SELECT s.sales_id, c.customer_name,
s.amount
FROM sales s
JOIN customers c ON s.customer_id =
c.customer_id;
```


Yogesh Tyagi
@ytyagi782

Updating Data Through Views

You can update data through updatable views, provided the view references a single table.

Example (Updatable View)

```sql
CREATE VIEW update_employees AS
SELECT emp_id, name FROM employees;
```

Update query

```sql
UPDATE update_employees SET name =
'John Doe' WHERE emp_id = 101;
```

Yogesh Tyagi
@ytyagi782

Creating Indexed Views (Materialized Views)

Materialized views store query results physically to improve performance for frequent, complex queries.

Example (PostgreSQL)

```
CREATE MATERIALIZED VIEW sales_summary
AS
SELECT customer_id, SUM(amount) AS
total_sales FROM sales GROUP BY
customer_id;
```

Refreshing the view

```
REFRESH MATERIALIZED VIEW
sales_summary;
```

Yogesh Tyagi
@ytyagi782

Advantages of SQL Views

1. Simplify complex queries into reusable components.
2. Enhance security by limiting data exposure.
3. Improve readability and maintainability of query logic.

Example (Simplified Query)

Instead of writing complex joins repeatedly:

```sql
SELECT * FROM sales_summary WHERE
total_sales > 1000;
```


Yogesh Tyagi
@ytyagi782

Limitations of SQL Views

1. Views cannot store data (except materialized views).
2. Performance depends on the underlying query execution.
3. Cannot include certain constructs like ORDER BY without TOP in some databases.

Example (Non-Updatable View)

If a view contains aggregated data, you cannot directly update it:

```
CREATE VIEW aggregated_sales AS
SELECT customer_id, SUM(amount) AS
total_sales FROM sales GROUP BY
customer_id;
```

Yogesh Tyagi
@ytyagi782

Best Practices for SQL Views

1. Use views to encapsulate frequently used logic.
2. Name views meaningfully for clarity (employee_summary_view).
3. Avoid overly complex views to reduce execution overhead.

Example (Meaningful Name)

```sql
CREATE VIEW active_customers_view AS
SELECT customer_id, name FROM
customers WHERE is_active = 1;
```


Yogesh Tyagi
@ytyagi782

Querying Data from Views

Query views just like a regular table.

Example

```sql
SELECT * FROM employee_details WHERE
department = 'Sales';
```

Use Case

Simplifies access to pre-filtered data for applications or reporting.

Yogesh Tyagi
@ytyagi782

Dropping a View

Remove unnecessary or outdated views using DROP VIEW.

Example:

```
DROP VIEW employee_details;
```

For materialized views

```
DROP MATERIALIZED VIEW sales_summary;
```


Yogesh Tyagi
@ytyagi782

Advanced View Concepts

Recursive Views

Solve hierarchical problems like organizational structures.

Examples:

```
CREATE VIEW hierarchy AS
WITH RECURSIVE org_chart AS (
    SELECT emp_id, manager_id FROM
employees WHERE manager_id IS NULL
    UNION ALL
    SELECT e.emp_id, e.manager_id
    FROM employees e
    JOIN org_chart o ON e.manager_id =
o.emp_id
) SELECT * FROM org_chart;
```

Parameterized Views (Workaround):

Use stored procedures to emulate views with parameters.


Yogesh Tyagi
@ytyagi782

Managing Performance with Views

1. Use materialized views for performance-intensive queries.
2. Index base tables referenced by the view for faster execution.
3. Regularly refresh materialized views to keep data up-to-date.

Example (Indexing)

```
CREATE INDEX idx_customer_id ON
sales(customer_id);
```

Yogesh Tyagi
@ytyagi782

Comparing Views and Tables

Feature	Views	Tables
Data Storage	Virtual (no physical storage)	Physically stores data.
Performance	Depends on query execution.	Faster due to pre-stored data.
Use Cases	Simplifying query logic.	Permanent data storage.

Yogesh Tyagi
@ytyagi782

Security with Views

Restrict access to sensitive data by exposing only necessary columns through views.

Example (Restricted View)

```
CREATE VIEW public_orders AS
SELECT order_id, customer_id FROM
orders;
```

Refreshing the view

```
GRANT SELECT ON public_orders TO
reporting_user;
```


Yogesh Tyagi
@ytyagi782

Use Cases of SQL Views

1. Use materialized views for performance-intensive queries.
2. Index base tables referenced by the view for faster execution.
3. Regularly refresh materialized views to keep data up-to-date.

Example (Indexing)

```sql
CREATE VIEW active_sales AS
SELECT order_id, amount FROM sales
WHERE status = 'Active';
```

Yogesh Tyagi
@ytyagi782

When to Avoid Views

Avoid using views when:

1. Performance-critical applications need optimized queries.
2. Complex views cause overhead due to nested queries.
3. Real-time updates are required (use materialized views instead).


Yogesh Tyagi
@ytyagi782

Wrap-Up

"SQL Views: Simplify, Secure, and Optimize!"

Use SQL Views to simplify query logic, protect sensitive data, and enhance database readability and usability.

Yogesh Tyagi

@ytyagi782

Follow for More