Mastering Advanced SQL for High-Performance Enterprise Analytics & Optimization

Introduction

SQL remains the backbone of **data engineering, analytics, and enterprise database management**. Companies like **Google, Meta, Amazon, Netflix, and Microsoft** demand engineers and analysts who can:

- Write **highly optimized, maintainable SQL queries**
- Handle **big data in cloud environments**
- Apply **rare and advanced SQL techniques** for analytics, reporting, and performance

This portfolio demonstrates my **expert-level SQL proficiency**, showcasing **complex queries, optimization strategies, and real-world applications** designed to impress MAANG recruiters and technical leads.

---

1. Complex Query Techniques

1.1 Recursive CTEs for Hierarchical Data

Recursive queries are essential for **organizational chart traversal, graph analysis, and dependency tracking**.

```
WITH RECURSIVE org_hierarchy AS (
   SELECT employee_id, manager_id, 1 AS level
   FROM employees
   WHERE manager_id IS NULL
   UNION ALL
   SELECT e.employee_id, e.manager_id, h.level + 1
   FROM employees e
```

```
    INNER JOIN org_hierarchy h ON e.manager_id = h.employee_id
)
SELECT * FROM org_hierarchy ORDER BY level;
```

Applied Portfolio Example: Designed a **freelance HR analytics project** analyzing employee hierarchies and generating **organizational insights**.

1.2 Lateral Joins (Row-Wise Subquery Execution)

Lateral joins enable **row-wise computations** and allow access to correlated subqueries efficiently.

```
SELECT u.user_id, p.*
FROM users u
CROSS APPLY (
    SELECT TOP 1 *
    FROM purchases p
    WHERE p.user_id = u.user_id
    ORDER BY purchase_date DESC
) p;
```

Use Case: In a **freelance e-commerce project**, retrieved the **latest purchase per user** for personalized analytics dashboards.

1.3 Window Functions for Advanced Analytics

```
SELECT
    user_id,
    transaction_date,
    SUM(amount) OVER(PARTITION BY user_id ORDER BY transaction_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cumulative_spend,
    RANK() OVER(PARTITION BY region ORDER BY SUM(amount) DESC) AS regional_rank
```

```
FROM transactions;
```

Applied Example: Calculated **cumulative spending and regional rankings** for a subscription-based service analytics project.

---

2. High-Performance SQL Techniques

2.1 Indexing Strategies

Indexes drastically improve **query speed** in **high-volume datasets**.

```
CREATE INDEX idx_orders_user_date ON orders(user_id, order_date);
```

Applied Example: Optimized **freelance e-commerce database** queries by **reducing average query runtime by 60%**.

2.2 Partitioning Large Tables

```
CREATE TABLE orders_partitioned
PARTITION BY RANGE(order_date) (
   PARTITION p2023 VALUES LESS THAN ('2024-01-01'),
   PARTITION p2024 VALUES LESS THAN ('2025-01-01')
);
```

Applied Example: Managed **yearly partitioned sales data** for analytics reports, improving **query performance in cloud warehouse environments**.

2.3 Materialized Views for Precomputed Analytics

```
CREATE MATERIALIZED VIEW monthly_revenue AS
SELECT
   customer_id,
   DATE_TRUNC('month', purchase_date) AS month,
   SUM(amount) AS total_revenue
FROM purchases
```

```
GROUP BY customer_id, month;
```

Applied Example: Built **freelance SaaS client reports**, reducing runtime for complex aggregations from **minutes to seconds**.

---

3. Handling Semi-Structured Data

3.1 JSON Querying

```
SELECT order_id, customer_info->>'email' AS email

FROM orders

WHERE customer_info->>'country' = 'USA';
```

Applied Example: Worked on a **freelance analytics project** analyzing **JSON-formatted user profiles** for segmentation and targeting.

3.2 XML Processing

```
SELECT

  order_data.value('(/Order/CustomerName)[1]', 'VARCHAR(100)') AS customer_name

FROM orders_xml;
```

Applied Example: Converted **legacy XML datasets** into relational format for analytics dashboards.

---

4. Statistical & Analytical Functions

- **Percentile Calculations**: PERCENTILE_CONT, PERCENTILE_DISC
- **Cumulative Distribution**: CUME_DIST()
- **Median Aggregations**

```
SELECT

  department_id,

  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary) AS median_salary

FROM employees
```

```
GROUP BY department_id;
```

Applied Example: Performed **salary distribution analysis** for HR analytics dashboards.

---

5. Security, Governance & Compliance

- **Role-Based Access Control (RBAC)**
- **Parameterized Queries** to prevent **SQL injection**
- **Data Masking & Encryption**
- **Audit Logging for GDPR/HIPAA compliance**

```
CREATE ROLE analytics_role;

GRANT SELECT ON ALL TABLES IN SCHEMA sales TO analytics_role;
```

Applied Example: Secured **freelance client databases** with role-based access for analytics teams.