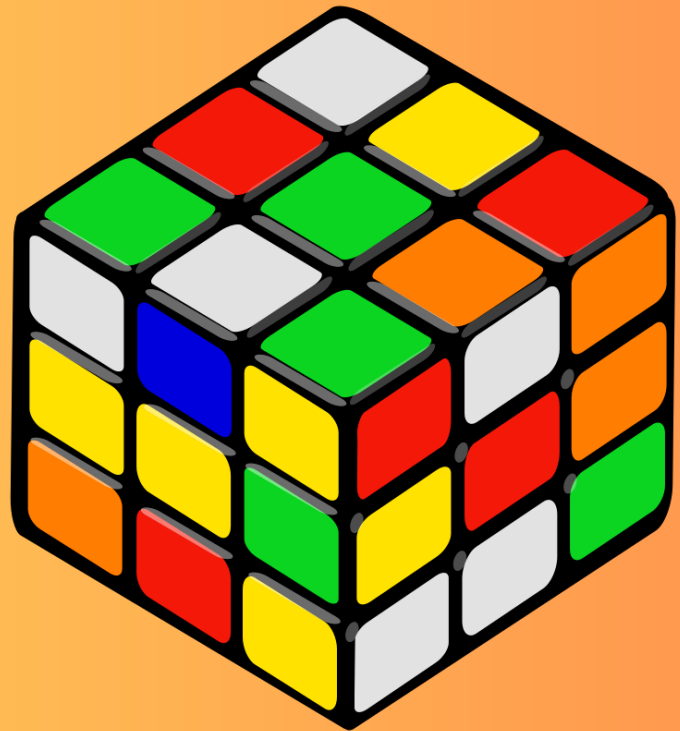




# SQL Optimization Techniques



- *SQL optimization techniques are methods to improve the performance of SQL databases by reducing query execution time and reducing resource usage.*

**Some common SQL optimization techniques are :**

- *Indexing.*
- *Avoiding using Wildcards.*
- *Normalization.*
- *Using appropriate Data types.*
- *Using correct Join types.*
- *Writing efficient WHERE clauses.*
- *Using aggregate functions wisely.*
- *Avoiding using subqueries.*
- *Use proper storage & backup methods.*
- *Monitoring performance.*



# 1.Indexing :

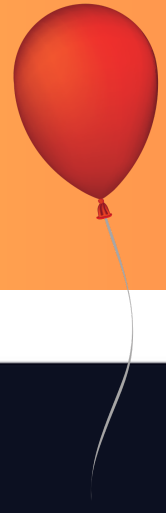
- *Using indexes to quickly find data in tables, reducing the number of rows that need to be scanned.*
- *Eg - Creating an index on a column used in the WHERE clause of a query can speed up the query execution time.*

```
CREATE INDEX idx_employee_id ON  
employees (employee_id);
```



## 2. Avoid using wildcards :

- *Wildcards slow down queries and should be used sparingly.*

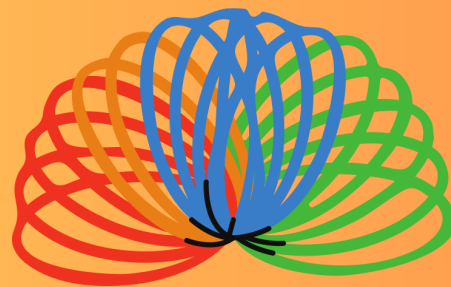


-- Instead of:

```
SELECT * FROM customers WHERE name LIKE  
'%Smith%';
```

-- Use:

```
SELECT * FROM customers WHERE name LIKE  
'Smith%';
```



### 3. Normalization :

- *Normalization divides data into smaller, more manageable tables, reducing data redundancy and improving data integrity.*

```
-- Non-normalized table:  
CREATE TABLE orders (order_id INT, customer_id  
INT, customer_name VARCHAR(50), order_total  
FLOAT);  
  
-- Normalized tables:  
CREATE TABLE customers (customer_id INT,  
customer_name VARCHAR(50));  
  
CREATE TABLE orders (order_id INT, customer_id  
INT, order_total FLOAT);
```



## 4. Use appropriate data types :

- *Use the smallest data type that can accommodate your data to save disk space and improve performance.*



-- Instead of:

```
CREATE TABLE customers (customer_id INT,  
is_active TINYINT);
```

-- Use:

```
CREATE TABLE customers (customer_id  
INT, is_active BOOLEAN);
```



## 5. Use the correct join type :

- *Choosing the correct join type can improve query performance and return the expected results.*



-- Inner join:

```
SELECT customers.*, orders.*  
FROM customers  
JOIN orders  
ON customers.customer_id = orders.customer_id;
```

-- Left join:

```
SELECT customers.*, orders.*  
FROM customers  
LEFT JOIN orders  
ON customers.customer_id = orders.customer_id;
```



## 6. Write efficient WHERE clauses :

- Use the correct comparison operators and avoid using complex expressions in WHERE clauses to improve query performance.

-- Instead of:

```
SELECT * FROM customers WHERE (age > 30 AND salary > 50000) OR (age < 30 AND salary < 30000);
```

-- Use:

```
SELECT * FROM customers WHERE (age > 30 OR age < 30) AND (salary > 50000 OR salary < 30000);
```





## 7. Use aggregate functions wisely :

- *Aggregate functions like SUM, AVG, and COUNT can simplify complex queries and improve performance.*

```
-- Find the average order total:  
SELECT AVG(order_total)  
FROM orders;  
  
-- Find the total order count:  
SELECT COUNT(order_id)  
FROM orders;
```



## 8. Avoid using subqueries :

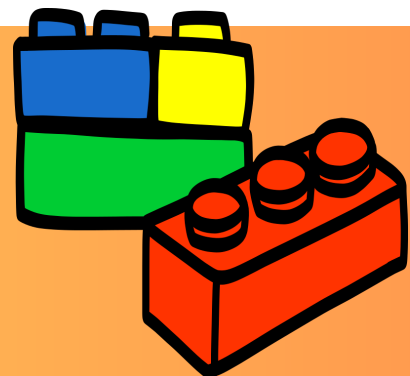
- *Subqueries can slow down queries and reduce performance. Consider alternative methods like joins or temporary tables.*

-- Instead of:

```
SELECT * FROM customers WHERE customer_id IN  
(SELECT customer_id FROM orders);
```

-- Use:

```
SELECT customers.*, orders.*  
FROM customers  
JOIN orders  
ON customers.customer_id = orders.customer_id;
```

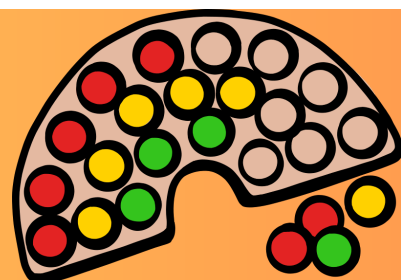


## 9. Use proper storage and backup methods :

- *Proper storage and backup methods can ensure data availability and recoverability in case of data loss.*



```
-- Store data in multiple locations:  
CREATE TABLESPACE primary_tablespace  
LOCATION '/path/to/primary/location';  
  
CREATE TABLESPACE secondary_tablespace  
LOCATION '/path/to/secondary/location';  
  
-- Backup data regularly:  
CREATE BACKUP TO '/path/to/backup/location';
```



## 10. Monitor performance :

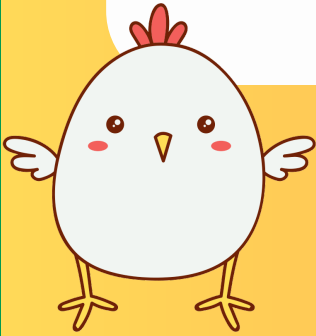
- *Monitor performance regularly to detect and resolve performance issues, and to identify opportunities for optimization.*



```
-- Use the EXPLAIN command:  
EXPLAIN SELECT * FROM customers WHERE age > 30;  
  
-- Use performance monitoring tools:  
SET
```



**DID YOU FIND THIS POST  
HELPFUL ?**



***Follow us on***



**@Maheshpal Singh Rathore**



**@mpsrathore2020**



**@Kiran Kanwar Rathore**



**@kiranrathore123**



***Like , Save, and Share  
with  
your friends !!!***

**Credit - Internet**

