Database Administration (MySQL-Focused)

By Djakba Faustin

1. Introduction to Database Administration (DBA)

What is a database administrator (DBA)?

A database administrator, or DBA for short, designs, implements, administers, and monitors data management systems and ensures consistency, quality, security, and compliance with rules and regulations.

Database administrator responsibilities

Now that we have defined what a database administrator is, we can take a deeper dive into the responsibilities that might be part of a database administrator job description.

The day-to-day activities that a DBA performs may include:

Security: Implementing security measures, managing user authentication, and protecting data from unauthorized access.

•**Backup and Recovery:** Designing and implementing backup and recovery plans to prevent data loss.

•**Performance:** Monitoring database performance, identifying and resolving issues, and tuning th system for optimal speed.

•**Installation and Maintenance:** Installing, configuring, and maintaining database software and related hardware.

•**Data Management:** Overseeing data integrity, managing data extraction, transformation, and loading (ETL), and assisting with data modeling.

•**User Support:** Providing support to users and working with developers on database design and implementation.

Introduction to DBMS and RDBMS?

A database is a collection of organized or arranged data that can be easily accessed, updated/ modified or controlled. Information within the database can be easily placed into rows and columns, or tables. Meanwhile, database management enables the user to store, manage and access data.

Database Management System (DBMS) is software used to identify, manage, and create a database that provides administered access to the data.

Relational Database Management System (RDBMS) is a more advanced version of a DBMS system that allows access to data in a more efficient way. It is used to store or manage only the data that are in the form of tables.

What is the Difference between DBMS and RDBMS?

DBMS stands for Database Management System, and RDBMS is the acronym for the Relational Database Management system. In DBMS, the data is stored as a file, whereas in RDBMS, data is stored in the form of tables. To know what is the difference between RDBMS and DBMS, check out the table below.

RDBMS	DBMS
Data stored is in table format	Data stored is in the file format
Multiple data elements are accessible together	Individual access of data elements
Data in the form of a table are linked together	No connection between data
Normalisation is not achievable	There is normalisation
Support distributed database	No support for distributed database
Data is stored in a large amount	Data stored is a small quantity
Here, redundancy of data is reduced with the help of key and indexes in RDBMS	Data redundancy is common
RDBMS supports multiple users	DBMS supports a single user
It features multiple layers of security while handling data	There is only low security while handling data
The software and hardware requirements are higher	The software and hardware requirements are low
Oracle, SQL Server.	XML, Microsoft Access.

MySQL: History, Features, and Editions

MySQL is an open-source relational database management system (RDBMS) that was first released on May 23, 1995. It was initially developed by Michael Widenius and David Hughes. MySQL AB, a Swedish company, originally developed and supported MySQL, which was later acquired by Sun Microsystems, and subsequently by Oracle Corporation.

Key Features:

- **Open-source:** Available under the GNU General Public License, offering flexibility and community support.
- **Relational Model:** Organizes data into tables with rows and columns, enforcing relationships between them.
- **Scalability:** Capable of handling large volumes of data and high numbers of concurrent users.
- **High Performance:** Optimized for fast data retrieval and manipulation.
- **Cross-platform Compatibility:** Runs on various operating systems, including Windows, Linux, macOS, and more.
- **Security:** Provides robust security features for data protection.
- **Support for various programming languages:** Integrates with languages like PHP, Python, Java, C++, and others.

Editions:

- **MySQL Community Edition:** The free, open-source version for general use and development.
- **MySQL Standard Edition:** Commercial edition offering enhanced features and support.
- **MySQL Enterprise Edition:** Comprehensive commercial offering with advanced features, tools, and support for enterprise-level applications.
- **MySQL Cluster Carrier Grade Edition:** Designed for high-availability and real-time applications requiring extreme performance and fault tolerance.

MySQL Architecture: Components and Client-Server Model

MySQL is a Relational Database Management system which is free Open Source Software Under GNU License. It is also supported by Oracle Company . It is fast , scalable, easy to use database management System. MySQL support many operating system like Windows, Linux, MacOS etc.

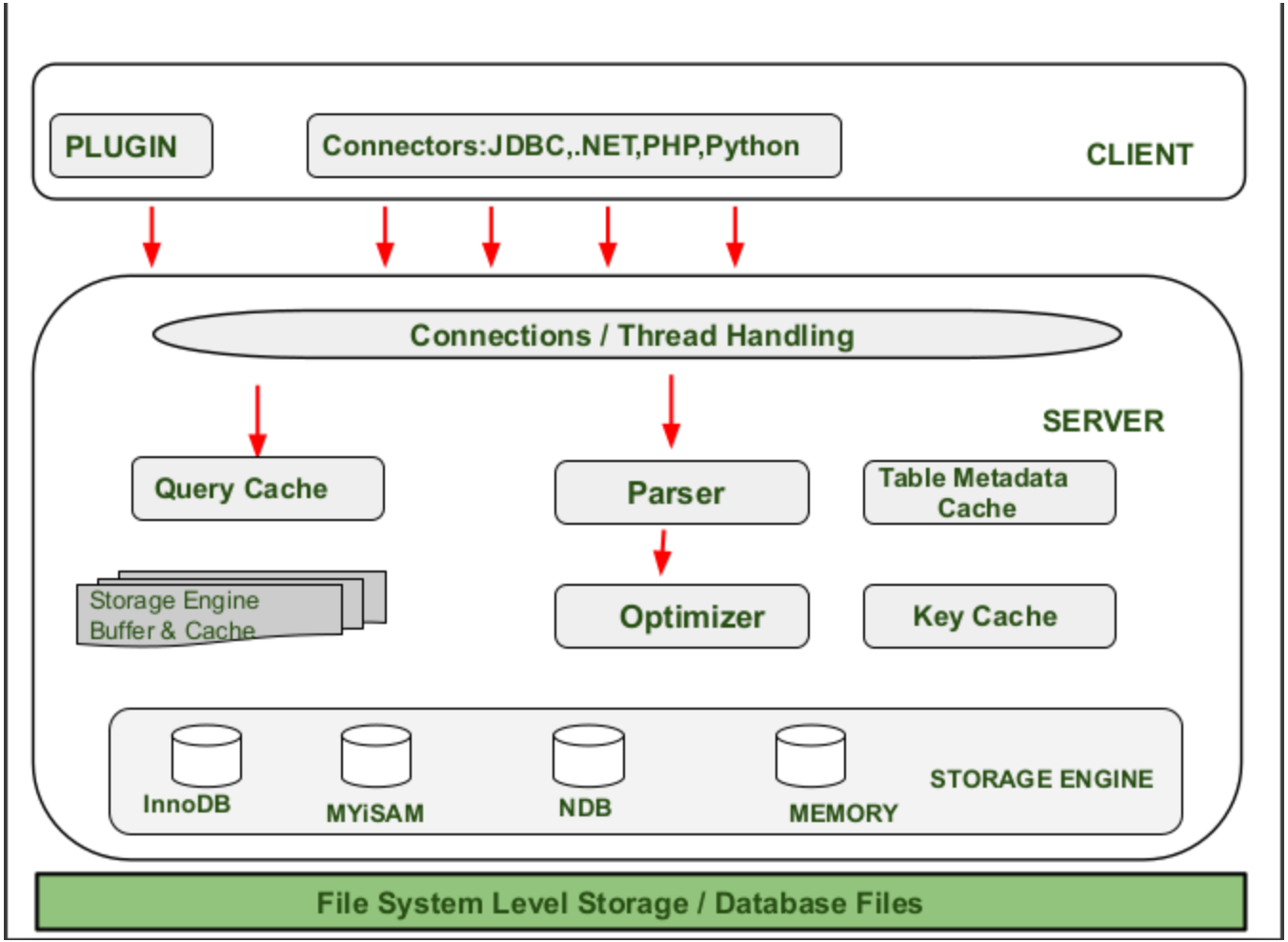
MySQL is a Structured Query Language which is used to manipulate, manage and retrieve data with the help of various Queries.

MySQL is developed and supported by MySQL AB, which is a Swedish Company and written in C and C++ programming language. It was developed by Michael Widenius and David Hughes . It is often used to say that MySQL is named after the name of the daughter of the co-founder Michael Widenius whose name is 'My'.

Architecture of MySQL:

Architecture of MySQL describes the relation among the different components of MySQL System. MySQL follow Client-Server Architecture. It is designed so that end user that is Clients can access the resources from Computer that is server using various networking services. The Architecture of MY SQL contain following major layer's :

- Client
- Server
- Storage Layer


PLUGIN
Connectors:JDBC,.NET,PHP,Python
CLIENT
Connections / Thread Handling
SERVER
Query Cache
Parser
Table Metadata Cache
Storage Engine Buffer & Cache
Optimizer
Key Cache
InnoDB
MYiSAM
NDB
MEMORY
STORAGE ENGINE
File System Level Storage / Database Files

Components:

MySQL Server (mysqld): The core component responsible for handling database operations. It includes:

- **Connection Handling:** Manages client connections and authentication.
- **SQL Parser:** Interprets and validates SQL queries.
- **Optimizer:** Determines the most efficient way to execute queries.
- **Query Cache:** Stores results of frequently executed queries for faster retrieval.
- **Storage Engines:** Pluggable modules (e.g., InnoDB, MyISAM) that handle data storage and retrieval.

Components:

MySQL Clients: Applications or tools that connect to the MySQL server to interact with databases. Examples include:

Command-line clients: mysql (for interactive SQL), mysqladmin (for server administration).

Graphical User Interface (GUI) tools: MySQL Workbench, phpMyAdmin.

Application programming interfaces (APIs): Connectors for various programming languages (e.g., Connector/J for Java, Connector/Python for Python)

Client-Server Model:

Client-Server Model:

In this model, clients send SQL queries to the MySQL server. The server processes these queries, retrieves or modifies data using the appropriate storage engine, and then sends the results back to the client. This distributed structure allows clients and servers to operate on different machines, communicating over a network using protocols like TCP/IP.

Installation and Setup.

To install MySQL, you can go to https://dev.mysql.com/downloads/installer/. But for our purposes, we will install XAMPP instead.

XAMPP is a very easy to install Apache Distribution for Linux, Solaris, Windows, and Mac OS X. The package includes the Apache web server, MySQL, PHP, Perl, a FTP server and phpMyAdmin. Clic on this link to download it https://www.apachefriends.org/

Configuration of MySQL Server: Running and Shutting down

Running and Shutting Down MySQL Server

Starting MySQL: Linux/macOS (Systemd/SysVinit).

```
sudo systemctl start mysql # For systemd-based systems
sudo service mysql start # For SysVinit-based systems
```

Windows (Service):

```
net start MySQL## # Replace ## with your MySQL version number
```

Shutting Down MySQL:

Linux/macOS (Systemd/SysVinit).

```
sudo systemctl stop mysql
sudo service mysql stop
```

Windows (Service).

```
net stop MySQL##
```

Command-line vs. Configuration files

Command-line Options:

Parameters can be passed directly when starting mysqld or mysqld_safe.

Example: mysqld --port=3307 --datadir=/var/lib/mysql_custom

Pros: Quick for testing specific settings, overrides configuration file settings for the current session.

Cons: Not persistent across server restarts, can lead to long and complex commands.

Configuration Files (Option Files):

MySQL reads settings from configuration files (e.g., my.cnf on Unix-like systems, my.ini on Windows) during startup.

Configuration files

These files contain key-value pairs defining server parameters.
Example (my.cnf):
[mysqld]

```
port=3306
datadir=/var/lib/mysql
innodb_buffer_pool_size=1G
```

•Pros:

•Persistent settings across restarts, centralized management of configurations, easier to read and maintain.

•Cons:

•Requires manual editing and server restart for changes to take effect (for most parameters).

Storage Engine: Importance of Storage Engines (e.g., InnoDB vs. MyISAM) and their features

Storage engines are critical database components that manage data storage, retrieval, and organization, significantly impacting a system's performance, scalability, and reliability. They handle tasks like data compression, caching, transaction management, and crash recovery, with different engines being optimized for specific workloads such as transactional processing or real-time analytics.

Importance of storage engines

•**Performance:** A storage engine's design directly affects read and write speeds. For example, in-memory engines provide ultra-low latency by eliminating disk I/O for specific use cases, while others are optimized for specific data types like time-series data.

•**Reliability and durability:** Storage engines are responsible for ensuring data integrity through features like transaction management, crash recovery, and checkpointing.

•**Resource optimization:** They implement techniques like data compression to reduce storage space and memory buffering to improve performance.

•**Scalability:** The underlying architecture of the storage engine determines how well a database scales.

•**Flexibility and modularity:** Many database systems, like MySQL, use a pluggable architecture that allows developers to switch storage engines.

InnoDB storage engine in MySQL

InnoDB is a general-purpose, transactional storage engine for MySQL and MariaDB, designed for high reliability and performance. It is the default storage engine in both MySQL (since version 5.5.5) and MariaDB, making it the standard choice for most database workloads.

Key features of InnoDB include:

- **ACID(**atomicity, consistency, isolation, and durability**) Compliance**: Supports transactions with commit, rollback, and crash-recovery capabilities to protect user data.
- **Row-Level Lockin**g: Enhances multi-user concurrency and performance by allowing multiple users to read and write to different rows simultaneously.

•**Foreign Key Suppor**t: Ensures data integrity by enforcing relationships between tables.

•**Clustered Indexes**: Data is stored in a primary key index (clustered index), optimizing primary key lookups and minimizing I/O.

•**Buffer Pool**: Caches frequently accessed data in memory for faster access.

•**Online DDL(**Data Definition Language**)**: Allows schema changes to be performed without locking tables.

•**Full-Text and Spatial Indexes**: Supports advanced indexing for text and geospatial data..

The MyISAM Storage Engine

MyISAM is based on the older (and no longer available) ISAM storage engine but has many useful extensions.

MyISAM Storage Engine Features

Feature	Support
B-tree indexes	Yes
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	No
Compressed data	Yes (Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.)
Data caches	No
Encrypted data	Yes (Implemented in the server via encryption functions.)
Foreign key support	No
Full-text search indexes	Yes
Geospatial data type support	Yes
Geospatial indexing support	Yes
Hash indexes	No
Index caches	Yes
Locking granularity	Table

Metadata

Metadata is **“data about data.”** It’s structured information that describes and gives context to a data asset so that it can be **understood, organized, and managed**.

Why Metadata Matters in Data Management? Modern enterprises generate **massive volumes of data**—from customer records to IoT sensor logs. Without proper metadata, this data quickly becomes unmanageable.

Metadata provides three critical advantages in data management:

1. **Faster Discovery and Access**: Users can search using metadata tags (e.g., “customer invoices Q1 2025”) instead of manually scanning thousands of files.
2. **Better Governance and Compliance**: Metadata tracks access permissions, lineage, and audit trails, making it easier to meet regulations like GDPR,
3. **Improved Decision-Making**: With metadata, business leaders can trust that their reports are based on accurate, well-documented data.

MySQL Clients and Interaction

MySQL offers various client tools for administration and interaction

- **mysql command-line client:** A basic SQL shell for interactive and batch mode execution of SQL statements.
- **MySQL Shell (mysqlsh):** An advanced client and code editor supporting SQL, JavaScript, and Python, offering enhanced features for administration and development.
- **mysqladmin:** Used for administrative tasks like creating/dropping databases, reloading grant tables, flushing logs, and retrieving server status.
- **mysqlcheck:** A table maintenance program for checking, repairing, analyzing, and optimizing tables.
- **mysqldump:** For creating backups by dumping database content into SQL, text, or XML files.
- **mysqlimport:** For importing data from text files into tables.
- **MySQL Workbench:** A graphical tool for database design, development, and administration.

Usage of the mysql Command-Line Client:

To connect to a MySQL server using the mysql client:

```
mysql -u username -p -h hostname -P portnumber databasename
```

`-u username:` Specifies the user to connect as.

`-p:` Prompts for the password.

`-h hostname:` Specifies the host where the MySQL server is running.

`-P portnumber:` Specifies the port number for the connection (default is 3306) .

`databasename:` (Optional) Specifies the database to connect to directly.

Administrative MySQL Commands:

- **SHOW DATABASES;**: Lists all databases on the server.
- **USE database_name;**: Selects a specific database to work with.
- **SHOW TABLES;**: Lists tables within the currently selected database.
- **SHOW STATUS;**: Displays server status information.
- **KILL process_id;**: Terminates a running server process identified by its ID.
- **FLUSH PRIVILEGES;**: Reloads the grant tables, applying any changes to user permissions.

Usage of Secured Connections:

MySQL clients can establish secure connections using SSL/TLS to encrypt communication between the client and the server. This typically involves:

Server Configuration: The MySQL server must be configured to support SSL/TLS, including having appropriate certificate and key files.

Client Connection Options: When connecting with the mysql client or other tools, specific options are used to enable SSL(Secure Sockets Layer)/TLS(Transport Layer Security):

```
mysql -u username -p --ssl-mode=REQUIRED --ssl-ca=/path/to/ca.pem --ssl-cert=/path/to/client-cert.pem --ssl-key=/path/to/client-key.pem
```

--ssl-mode=REQUIRED: Ensures an encrypted connection is established.

--ssl-ca: Specifies the path to the Certificate Authority (CA) certificate.

--ssl-cert: Specifies the path to the client's SSL certificate.

--ssl-key: Specifies the path to the client's SSL key.

Data Management, Types, and Structuring

Presentation of Data Types:

MySQL supports various data types, each suited to different kinds of data:

Numeric Types: Used for numerical data.

Examples include INT (integers), DECIMAL (fixed-point numbers for financial data), FLOAT, and DOUBLE (floating-point numbers for approximate values).

String Types: Used for text and binary data.

Examples include VARCHAR (variable-length strings, common for names or addresses), CHAR (fixed-length strings), TEXT (longer text blocks like descriptions), and BLOB (binary data like images).

Presentation of Data Types:

Date/Time Types: Used to store temporal data.

Examples include DATE (date only), TIME (time only), DATETIME (date and time),

TIMESTAMP (date and time with automatic update features), and YEAR (year only).

MySQL Transactions: ACID properties

Transaction Management in MySQL refers to the ability to group a set of database operations into a single unit of work that must be executed as a whole, either entirely or not at all, to ensure data consistency and integrity.

ACID properties

ACID stands for atomicity, consistency, isolation, and durability. These four properties are essential for ensuring that database transactions are processed reliably, making them fundamental to the design and operation of database management systems.

- **Atomicity**: All operations within a transaction are completed, or none are.
- **Consistency**: A transaction brings the database from one valid state to another.
- **Isolation**: Concurrent transactions do not interfere with each other.
- **Durability**: Once a transaction is committed, the changes are permanent, even in the event of system failure.

Transaction Management Commands:

```
START TRANSACTION;

-- SQL statements to be executed as part of the transaction
-- INSERT, UPDATE, DELETE, or SELECT statements can be used here

COMMIT; -- commit the transaction if everything went well
-- OR
ROLLBACK; -- rollback the transaction if something went wrong
```

Example

```
START TRANSACTION;

-- Insert a new record into the users table
INSERT INTO users (username, email) VALUES ('john', 'john@example.com');

-- Update the balance of a bank account
UPDATE bank_accounts SET balance = balance - 100 WHERE account_number = '123456';
UPDATE bank_accounts SET balance = balance + 100 WHERE account_number = '789012';

-- Check if both updates were successful
IF (SELECT ROW_COUNT() = 2) THEN
    COMMIT;
    SELECT 'Transaction successful!';
ELSE
    ROLLBACK;
    SELECT 'Transaction failed!';
END IF;
```

Example

This code first starts a transaction using START TRANSACTION. It then inserts a new record into the user's table and updates the balance of two accounts in the bank_accounts table.

After performing the updates, it checks if both updates were successful using the IF statement. If both updates are successful, it commits the transaction using COMMIT and prints a message indicating that the transaction was successful. If either of the updates fails, it rolls back the transaction using ROLLBACK and prints a message indicating that the transaction failed.

Presentation and advantages of partitioning.

MySQL table partitioning is a database optimization technique that divides a large table into smaller, more manageable sub-tables (partitions) to improve performance and data management.

Benefits of MySQL Table Partitioning

1. Improved Query Performance
2. Enhanced Data Management
3. Increased Availability and Reliability
4. Optimized Indexing and Storage Efficiency

Creating a partition table (Range, List, Hash)

Range partitioning divides data into partitions based on the specified range of values. This is useful for partitioning data based on the dates or numeric ranges.

```
CREATE TABLE orders (
  order_id INT,
  order_date DATE,
  customer_id INT,
  amount DECIMAL(10, 2)
)
PARTITION BY RANGE (YEAR(order_date)) (
  PARTITION p0 VALUES LESS THAN (2019),
  PARTITION p1 VALUES LESS THAN (2020),
  PARTITION p2 VALUES LESS THAN (2021),
  PARTITION p3 VALUES LESS THAN (2022)
);
```

- In this example, the orders table is partitioned by the year with each partition containing orders from the specific year.

Creating a partition table (Range, List, Hash)

The List partitioning divides data based on a predefined list of the values. This is useful when partitioning data by categories or specific values.

```
CREATE TABLE employees (
    emp_id INT,
    emp_name VARCHAR(50),
    department VARCHAR(50)
)
PARTITION BY LIST COLUMNS(department) (
    PARTITION p0 VALUES IN ('HR', 'Finance'),
    PARTITION p1 VALUES IN ('Engineering', 'Sales'),
    PARTITION p2 VALUES IN ('Marketing', 'Support')
);
```

Here, the employees table is partitioned by the department with each partition containing employees from the specific departments.

Creating a partition table (Range, List, Hash)

Hash partitioning divides data based on a hash function, ensuring an even distribution of rows across multiple partitions for balanced performance and efficient data access

```
CREATE TABLE users (
    user_id INT,
    user_name VARCHAR(50),
    email VARCHAR(50)
)
PARTITION BY HASH(user_id) PARTITIONS 4;
```

Here, the users table is partitioned using a hash function on the user_id column, evenly distributing the data across four partitions.

Managing and Maintaining Table Partitions

Managing partitions involves performing operations such as adding, dropping, merging and splitting partitions. These operations help maintain the performance and manageability of the partitioned tables.

Adding a Partition, To add a new partition to an existing table:

```
ALTER TABLE orders ADD PARTITION (
 PARTITION p4 VALUES LESS THAN (2023)
);
```

Dropping a Partition

```
ALTER TABLE orders DROP PARTITION p0;
```

Splitting Partitions

```
ALTER TABLE orders REORGANIZE PARTITION p3 INTO (
PARTITION p3_1 VALUES LESS THAN (2022),
PARTITION p3_2 VALUES LESS THAN (2023)
);
```

Exporting and Importing Data

Data transfer is crucial for backups, migrations, and analysis.

.mysqldump: A command-line utility used to export a database structure and data into a single SQL script file.

```
mysqldump -u username -p database_name > backup.sql
```

To import, use the MySQL client:

```
mysql -u username -p database_name < backup.sql
```

Exporting and Importing Data

.SELECT ... INTO OUTFILE and LOAD DATA INFILE: Efficient MySQL commands for exporting and importing data into delimited text files (like CSV), useful for large datasets

```
-- Export to a CSV file
SELECT * INTO OUTFILE '/tmp/data.csv'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM sales;

-- Import from a CSV file
LOAD DATA INFILE '/tmp/data.csv'
INTO TABLE sales
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Security Risks in a Database Environment

- **Unauthorized Access:** Gained through weak passwords, default credentials, or compromised accounts.
- **SQL Injection:** Attackers exploit application vulnerabilities to inject malicious SQL queries, leading to data manipulation or exposure.
- **Privilege Abuse:** Legitimate users (e.g., DBAs, developers) misusing their high-level permissions to access data they shouldn't.
- **Malware and Ransomware:** Malicious software that can infect database servers, encrypt data, and disrupt operations.

Security Risks in a Database Environment

- **Data Leakage:** Accidental exposure or unsecured transmission of sensitive information.
- **Outdated Software:** Failing to apply patches for known vulnerabilities, leaving systems open to exploits.
- **Misconfigurations:** Using default settings, having open network ports, or insecure installations create security loopholes.
- **Insider Threats:** Employees or contractors, either maliciously or accidentally, compromising data security.

Security Measures and Auditing

•**Strong Authentication and Access Controls:** Enforce complex passwords, Multi-Factor Authentication (MFA), and the **Principle of Least Privilege** (PoLP), granting users only the minimum permissions necessary for their jobs.

•**Encryption:** Encrypt data both **at rest** (stored on disk) and **in transit** (moving across a network via TLS/SSL).

•**Network Security:** Isolate database servers on separate network segments, use firewalls, and restrict public internet access to database ports.

Security Measures and Auditing

- **Regular Patching and Updates:** Promptly apply security patches and updates released by database vendors to fix known vulnerabilities.
- **Database Activity Monitoring (DAM) and Auditing:** Continuously monitor and log all database activities (logins, queries, admin actions) to detect anomalies and provide an audit trail for investigations and compliance.
- **Secure Backups and Recovery Plans:** Encrypt backups, store them securely (preferably offsite or air-gapped), and regularly test recovery procedures to ensure data integrity and availability in case of a breach or disaster.
- **Security Testing:** Regularly perform vulnerability assessments and penetration testing to proactively identify and address weaknesses.

Privileges: Understanding and Assigning

Privileges (permissions) control what operations a user can perform. They are managed using GRANT to assign permissions and REVOKE to remove them.

GRANT <privileges> ON <object> TO <user>;

REVOKE <privileges> ON <object> FROM <user>;

The WITH GRANT OPTION clause allows the recipient user to further grant those same privileges to other users.

Access Levels: Defining and Managing

Access levels can be defined at various scopes using the GRANT statement:

•**User Accounts:** Privileges are assigned to individual users or, more efficiently, to roles/groups which users belong to (Role-Based Access Control).

•**Databases:** Granting access to an entire database.

Example (MySQL): GRANT ALL PRIVILEGES ON sales.* TO 'salesadmin'@'localhost’;

.Tables: Limiting access to specific tables within a database.

Example (MySQL): GRANT SELECT, INSERT ON website.eventlog TO 'weblogger'@'localhost';

Access Levels: Defining and Managing

Columns: Restricting access to specific columns within a table.

Example (SQL Server): GRANT UPDATE (EMPNO, DEPT) ON TABLE EMP TO NATZ;

Stored Routines (Procedures/Functions): Granting the ability to execute specific stored procedures or functions.

Example (MySQL): GRANT EXECUTE ON FUNCTION CalculateSalary TO 'Amit'@'localhost';

Management of User Accounts

Database administrators manage the lifecycle of user accounts:

Creation: Use CREATE USER (e.g., CREATE USER 'username'@'host' IDENTIFIED BY 'password'; in MySQL).

Modification: Alter user properties, such as passwords or assigned roles (e.g., ALTER USER or RENAME USER in some systems).

Deletion: Remove accounts when no longer needed (e.g., DROP USER 'username'@'host'; in MySQL). Inactive accounts should be promptly deactivated or deleted to minimize insider threats.

Database Maintenance and Optimization

Maintenance of Tables

Regular table maintenance helps to reclaim wasted space, maintain data integrity, and improve performance. Specific commands are often used depending on the database system (e.g., MySQL):

CHECK TABLE: Analyzes a table for errors and integrity issues. It is a fundamental step in proactive maintenance .

REPAIR TABLE: Attempts to fix a corrupted table. This is often necessary following a system crash or power outage .

OPTIMIZE TABLE: Reclaims fragmented space and defragments the data file. This is most beneficial for tables with frequent inserts, updates, and deletes, especially for storage engines like MyISAM or specific configurations in InnoDB

Syntax

sql

```sql
-- Check the 'products' table for errors
CHECK TABLE products;

-- Repair the 'orders' table if corrupted
REPAIR TABLE orders;

-- Optimize the 'users' table to reclaim space and improve performance
OPTIMIZE TABLE users;

-- You can also run these commands on multiple tables at once
CHECK TABLE products, orders, users;
```

Backup and Recovery Types

Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.

Logical backups save information represented as logical database structure (CREATE DATABASE, CREATE TABLE statements) and content (INSERT statements or delimited-text files). This type of backup is suitable

for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture

Monitoring

Basic monitoring provides vital insights into the health and performance of the database system, allowing administrators to identify and address bottlenecks proactively.

Key Metrics to Monitor:

Connections: Track the number of active and idle connections to prevent connection pool exhaustion .

Throughput: Monitor the rate of queries (reads/writes) per second [1].

Latency: Measure the time it takes for queries to execute to pinpoint slow operations .

Resource Utilization: Keep track of CPU, memory, disk I/O, and storage capacity.

Tools: Built-in status variables (e.g., SHOW STATUS in MySQL), Prometheus, Grafana, and cloud-provider monitoring solutions like Amazon RDS Performance Insights or Google Cloud Monitoring