\*\*ORMS (Object-Relational Mapping Systems)\*\*

ORMs (Object-Relational Mapping Systems) are tools or frameworks that facilitate the conversion of data between incompatible type systems in object-oriented programming languages. This allows developers to interact with a relational database using the object-oriented paradigm of their programming language, without writing direct SQL queries.

\*\*Key Features and Benefits of ORMs:\*\*

1. \*\*Abstraction of Database Interactions\*\*: ORMs abstract the database interactions, allowing developers to manipulate database entries as if they were working with native objects in their programming language.

2. \*\*Productivity\*\*: By reducing the amount of boilerplate code required for CRUD (Create, Read, Update, Delete) operations, ORMs significantly enhance developer productivity.

3. \*\*Maintainability\*\*: ORMs can lead to more maintainable codebases by centralizing and standardizing data access logic.

4. \*\*Database Agnosticism\*\*: Many ORMs support multiple database systems, making it easier to switch from one database to another without extensive code changes.

5. \*\*Security\*\*: ORMs can help mitigate SQL injection attacks by using parameterized queries.

\*\*Examples of Popular ORMs:\*\*

- \*\*Hibernate\*\* (Java)

- \*\*Entity Framework\*\* (C#/.NET)

- \*\*SQLAlchemy\*\* (Python)

- \*\*Doctrine\*\* (PHP)

- \*\*Active Record\*\* (Ruby on Rails)

---

\*\*Databases\*\*

Databases are structured collections of data that are managed and stored electronically. They enable the efficient retrieval, insertion, update, and deletion of data.

\*\*Types of Databases:\*\*

1. \*\*Relational Databases (RDBMS)\*\*: Store data in tables with rows and columns. They use SQL (Structured Query Language) for data manipulation.

- \*\*Examples\*\*: MySQL, PostgreSQL, Oracle, Microsoft SQL Server

2. \*\*NoSQL Databases\*\*: Designed for specific data models and include key-value, document, column-family, and graph databases. They are often used for large-scale data storage and real-time web applications.

- \*\*Examples\*\*: MongoDB (document), Redis (key-value), Cassandra (column-family), Neo4j (graph)

3. \*\*In-Memory Databases\*\*: Store data in RAM for faster read/write operations.

- \*\*Examples\*\*: Redis, Memcached

4. \*\*NewSQL Databases\*\*: Combine the scalability of NoSQL systems with the ACID guarantees of traditional relational databases.

- \*\*Examples\*\*: CockroachDB, Google Spanner

\*\*Key Features and Concepts of Databases:\*\*

### ACID Properties

ACID properties are a set of principles that ensure the reliable processing of database transactions. They are crucial for maintaining data integrity and consistency in relational databases.

1. \*\*Atomicity\*\*:

- Ensures that all operations within a transaction are completed successfully. If any part of the transaction fails, the entire transaction fails and the database state is left unchanged.

- Example: Transferring money between bank accounts involves debiting one account and crediting another. Both operations must succeed or fail together.

2. \*\*Consistency\*\*:

- Guarantees that a transaction can only bring the database from one valid state to another, maintaining database rules and constraints.

- Example: Inserting a row into a table must not violate any predefined constraints such as unique keys or foreign keys.

3. \*\*Isolation\*\*:

- Ensures that concurrent transactions do not interfere with each other. Intermediate states of a transaction are invisible to other transactions.

- Example: While one transaction is updating a record, another transaction cannot read that record until the first transaction is complete.

4. \*\*Durability\*\*:

- Once a transaction has been committed, it remains committed even in the case of a system crash. This is achieved through mechanisms like transaction logs.

- Example: After completing a purchase, the transaction is recorded permanently even if the system crashes immediately afterward.

### Schema

A schema defines the structure of data in a relational database, including tables, fields, and relationships between them.

1. \*\*Tables\*\*:

- Structured collections of rows and columns where data is stored.

- Example: A table called `Employees` might have columns like `EmployeeID`, `FirstName`, `LastName`, `Position`, and `Salary`.

2. \*\*Fields (Columns)\*\*:

- Define the type of data that can be stored in each column.

- Example: `EmployeeID` might be an integer, `FirstName` and `LastName` might be strings, and `Salary` might be a decimal.

3. \*\*Relationships\*\*:

- Define how data in one table relates to data in another table. Common relationships include one-to-one, one-to-many, and many-to-many.

- Example: A one-to-many relationship between `Departments` and `Employees` where each department can have multiple employees.

### Indexes

Indexes improve the speed of data retrieval operations by providing quick access paths to data.

1. \*\*Types of Indexes\*\*:

- \*\*Primary Index\*\*: Automatically created when a primary key is defined.

- \*\*Secondary Index\*\*: Created by users to speed up queries on non-primary key columns.

2. \*\*Usage\*\*:

- Example: An index on the `LastName` column of an `Employees` table can speed up searches for employees by their last names.

### Normalization

Normalization is the process of organizing data to reduce redundancy and improve data integrity.

1. \*\*Normal Forms\*\*:

- \*\*First Normal Form (1NF)\*\*: Ensures that each column contains only atomic (indivisible) values and each column contains values of a single type.

- \*\*Second Normal Form (2NF)\*\*: Ensures that all non-key columns are fully functionally dependent on the primary key.

- \*\*Third Normal Form (3NF)\*\*: Ensures that all non-key columns are not dependent on other non-key columns.

2. \*\*Benefits\*\*:

- Reduces data redundancy and prevents anomalies during data operations.

### Replication

Replication involves copying data across multiple databases or servers to ensure data availability and reliability.

1. \*\*Types of Replication\*\*:

- \*\*Master-Slave Replication\*\*: Data is copied from a master database to one or more slave databases. Writes happen only on the master, while reads can occur on any server.

- \*\*Multi-Master Replication\*\*: Allows writes on multiple databases, which then synchronize with each other.

2. \*\*Usage\*\*:

- Example: An e-commerce site might use replication to ensure that data is available even if one server fails.

### Sharding

Sharding involves distributing data across multiple machines to handle large datasets and high-transaction volumes.

1. \*\*Types of Sharding\*\*:

- \*\*Horizontal Sharding\*\*: Splits data into rows, distributing different rows across multiple shards.

- \*\*Vertical Sharding\*\*: Splits data into columns, distributing different columns across multiple shards.

2. \*\*Usage\*\*:

- Example: A large social media platform might shard user data based on user ID ranges to distribute load and improve performance.

### Backup and Recovery

Processes to ensure data can be restored in case of corruption, loss, or disasters.

1. \*\*Backup\*\*:

- Regularly creating copies of the database and storing them in a secure location.

- Types: Full backup, differential backup, incremental backup.

2. \*\*Recovery\*\*:

- Restoring the database from backups to ensure data continuity.

- Example: If a database is corrupted, a recent backup can be restored to minimize data loss.

### Transactions

Transactions are sequences of one or more SQL operations that are executed as a single unit.

1. \*\*Properties\*\*:

- Begin with a `START TRANSACTION` or similar command.

- End with a `COMMIT` to save changes or `ROLLBACK` to undo changes.

2. \*\*Usage\*\*:

- Example: A banking application might use a transaction to ensure that transferring funds between accounts is completed entirely or not at all.

These concepts and features are fundamental to understanding how databases work and are used in practice to ensure data integrity, performance, and reliability in various applications.