CHAPTER 1 - RDBMS & DATABASES

INTRODUCTION

For a successful business, fast access to information is critical. You extract information from the existing data. Important decisions are based on the information available at any point in time. To get the right information at the right time, you store business-related data, in the form of text, numbers, pictures and sound, on a computer system. This aids in fast and easy access to information, besides data access, a computerized system enables you to manage data efficiently. Managing data involves storing, organizing, adding, modifying, and deleting data.

What is Database?

A database is a collection of data that is organized in a specific manner and stored electronically.

Purpose: The main purpose of a database is to store and manage data in a way that is both efficient and easy to use.
Data Organization: Data in a database is stored in tables, fields, and records.
Management: Databases are managed by a Database Management System (DBMS), which provides tools and interfaces for creating, updating, and querying the database.
Data Consistency, Accuracy, and Security: The DBMS ensures that the data stored in the database is consistent, accurate, and secure.
Types of Databases: There are two main types of databases: relational databases and non-relational databases (NoSQL databases).
Relational Databases: Relational databases use the relational model to organize data into tables, columns, and rows.
NoSQL Databases: NoSQL databases use alternative data structures, such as key-value pairs, documents, and graphs, to store and retrieve data.
Use Cases: Databases are used for various purposes, such as data analysis, data mining, and decision-making. They are also used for big data applications and real-time data processing.

DBMS

DBMS stands for Database Management System. It is a software system that provides a comprehensive and efficient solution for managing and organizing data stored in a database.

The history of DBMS dates back to the 1960s, when IBM introduced the first commercially available DBMS system, known as System R. This system was based on the relational model, which was proposed by E.F. Codd in 1970. The relational model provided a way to organize data into tables and define relationships between the data using keys.

Main Points:

Data Organization: A DBMS organizes data into tables, which are composed of columns (or fields) and rows (or records). The relationships between tables are established using keys.

Types of DBMS: There are several different types of DBMS, including relational DBMSs, object-oriented DBMSs, and NoSQL databases.

SQL: SQL (Structured Query Language) is a standardized language for accessing and manipulating data in a DBMS. Most relational DBMSs support SQL, which allows users to write queries to retrieve and manipulate data in the database.

Advantages: DBMSs are widely used because of their scalability, reliability, and ability to handle complex relationships between data elements. They are also preferred over other types of databases because they provide a wide range of tools and features for managing and querying data.

RDBMS

RDBMS stands for Relational Database Management System. It is a type of database management system that is based on the relational model. A relational database stores data in tables and defines relationships between the data using keys.

In an RDBMS, data is organized into tables, also known as relations, where each table has a unique name and is composed of columns (or fields) and rows (or records). Each column has a specific data type, and each row represents a single record in the table.

RDBMSs are widely used because of their scalability, reliability, and ability to handle complex relationships between data elements. They are also preferred over other types of databases because they provide a wide range of tools and features for managing and querying data.

NoSQL Databases

NoSQL databases are a type of database management system that do not use the traditional relational model for organizing data. Instead, NoSQL databases use alternative data structures, such as key-value pairs, document-based, column-based, and graph-based, to store and retrieve data.

Data Organization: NoSQL databases do not rely on the relational model and do not use tables, columns, and rows to organize data. Instead, data is stored in a more flexible and dynamic structure that allows for easier scalability and more efficient data retrieval.

Scalability: NoSQL databases are designed to be highly scalable and can handle large amounts of data and high levels of traffic. They are often used for applications that need to handle big data and real-time data processing.

Flexibility: NoSQL databases are flexible and allow for the storage of data in different formats, such as key-value pairs, documents, and graphs. This allows for the representation of complex data structures and relationships between data elements.

Performance: NoSQL databases are optimized for fast data retrieval and are often faster than traditional relational databases for certain types of queries.

Some popular examples of NoSQL databases include MongoDB, Cassandra, CouchDB, and Redis.

Comparison between RDBMS and NoSQL Databases

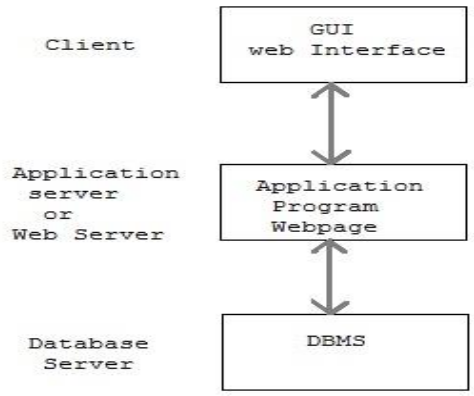
Feature	RDBMS	NoSQL
Data Model	Relational	Non-Relational (Document-based, Key-Value, Graph, Column-based)
Structure	Tables with rows and columns	No fixed structure, flexible and dynamic
Scalability	Vertically Scalable	Horizontally Scalable
Consistency	Strong consistency	Eventual consistency
ACID Compliance	Yes	No
Use Cases	Transactional systems, enterprise applications, reporting systems	Big Data, real-time web applications, content management systems
Examples	Oracle, Microsoft SQL Server, MySQL, PostgreSQL	MongoDB, Cassandra, CouchDB, Amazon DynamoDB

COMPARISON OF DIFFERENT RDBMS

Feature	MySQL	PostgreSQL	Microsoft SQL Server	Oracle Database	SQLite
Cost	Open source (free)	Open source (free)	Commercial	Commercial	Open source (free)
Platform Support	Cross-platform	Cross-platform	Windows and Linux	Windows and Linux	Cross-platform
Data Types	Supports a wide range of data types, including numeric, date and time,	Supports a wide range of data types, including advanced data types such as arrays, hstore	Supports a wide range of data types, including XML and spatial data.	Supports a wide range of data types, including multimedia and XML.	Supports a limited subset of data types, including text and numeric.

	string, and binary data.	(key-value store), and JSONB (binary JSON).			
Scalability	Supports horizontal and vertical scalability.	Supports horizontal and vertical scalability.	Supports horizontal and vertical scalability.	Supports horizontal and vertical scalability.	Not designed for large-scale applications.
Performance	Known for high performance.	Offers good performance for a wide range of workloads.	Known for high performance.	Known for high performance.	Simple and lightweight, optimized for read-intensive applications.
Security	Strong data security features, including password protection and encryption.	Strong data security features, including role-based access control and encryption.	Built-in security features, including transparent data encryption and advanced auditing.	Advanced security features, including database firewall and data masking.	Limited security features.
SQL Support	Supports SQL (Structured Query Language).	Supports SQL and NoSQL.	Supports SQL.	Supports SQL.	Supports a limited subset of SQL.
Concurrency	Supports concurrent access to the database.	Supports concurrent access to the database.	Supports concurrent access to the database.	Supports concurrent access to the database.	Supports concurrent access to the database.

Database Architecture

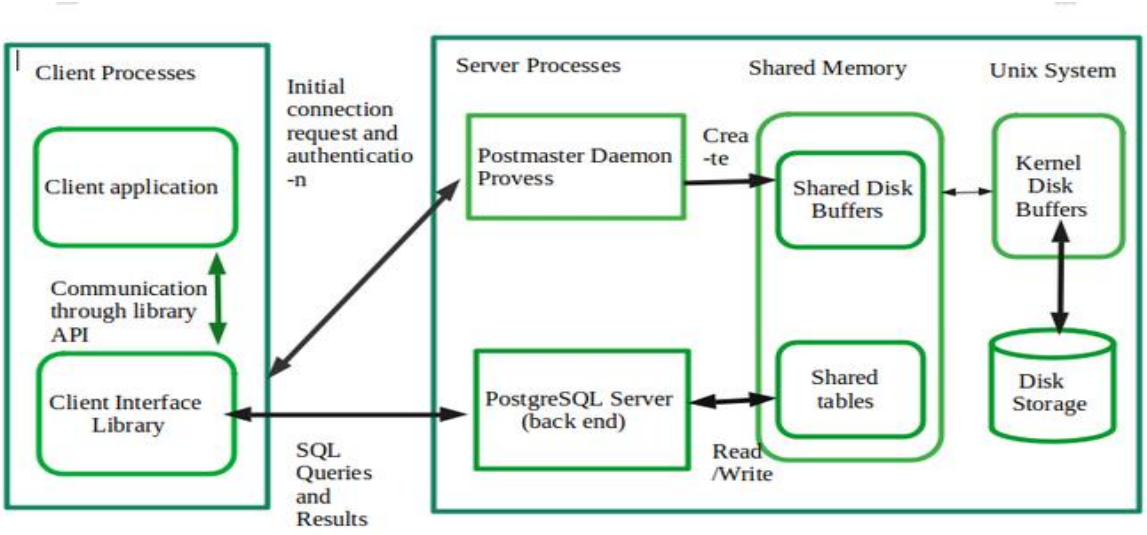


3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

- **User (Presentation) Tier** –End users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

PostgreSQL Architecture



PostgreSQL is an open-source Database Management System that has an object-relational nature. PostgreSQL is a successor of one of the earliest systems i.e. POSTGRES system. It is one of the most widely used open-source database management systems.

PostgreSQL has a Client-server model of architecture. In the simplest term a PostgreSQL service has 2 processes:

- **Server-side process:** This is the "Postgres" application that manages connections, operations, and static & dynamic assets.
- **Client-side process (Front-end applications):** These are the applications that users use to interact with the database. It generally has a simple UI and is used to communicate between the user and the database generally through APIs.

The other features of PostgreSQL RDBMS are:

Client-server architecture

Process-based architecture

Multi-version concurrency control (MVCC)

Object-relational data model

Extensible architecture

Multiple storage engines

ACID compliance

What is ACID?

ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that ensure the reliability and consistency of transactions in a relational database management system.

Atomicity requires that transactions be treated as a single unit of work and either fully committed or fully rolled back, so the database is either in its original state or the state after the transaction.

Consistency requires that a transaction only be committed if it doesn't violate any integrity constraints, so it brings the database from one consistent state to another.

Isolation ensures that transactions are executed in isolation from each other, so the changes made by one transaction are not visible to other transactions until the first transaction is committed.

Durability ensures that once a transaction is committed, its effects are permanent and survive any subsequent failures, using write-ahead logging and other techniques.

ACID provides data reliability and consistency, helping to prevent data corruption and maintain the integrity of the data stored in the database.

Example using Day-to-Day Bank Transactions:

Atomicity: If the transfer of $100 is a single transaction, it must be completed in its entirety or not at all.

In other words, the system would ensure that your savings account isn't debited, and your checking account isn't credited if the transaction is not completed.

Consistency: The transfer of $100 must bring the state of your accounts from one consistent state to another.

For example, if your savings account has $500 before the transfer, and the checking account has $100, the transaction would ensure that your savings account has $400 and your checking account has $200 after the transfer, if these amounts are within the allowed limits.

Isolation: The transfer of $100 should be isolated from other transactions that may be happening at the same time.

For example, if another user is trying to transfer money from their account to your savings account at the same time, the system should ensure that the transactions are executed one after the other and not simultaneously.

Durability: The changes made by the transaction should be permanent and survive any system crashes or failures.

For example, the system should ensure that the debit from your savings account and the credit to your checking account are recorded in the database and are not lost even if the system crashes.

Data Model

A Data model is a conceptual representation of the data in a database, including the relationships and constraints between the data elements. A data model serves as a blueprint for designing a database and can help to ensure that the data is organized in a consistent and meaningful way.

There are several types of data models, including:

- **Conceptual data models:** This type of data model provides a high-level view of the data and its relationships. It is used to define the overall structure of the database, including the entities, attributes, and relationships between entities.
- **Logical data models:** This type of data model provides a more detailed view of the data and its relationships. It is used to define the structure of the database in terms of tables, columns, keys, and relationships between tables.
- **Physical data models:** This type of data model provides a low-level view of the data and its relationships. It is used to define the physical storage of the data, including the file structure, indexes, and access methods.

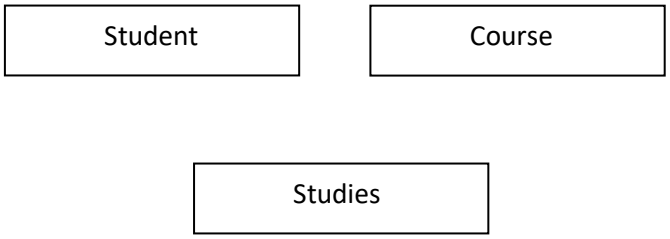
A data model is an important step in the design of a database, as it provides a clear understanding of the data that needs to be stored and the relationships between the data elements. This information can be used to design an effective database schema and ensure that the data is stored in a consistent and meaningful way.

Entity-Relationship Model:

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes, and constraints.

Entity (Tables):

An Entity is an object with a distinct set of properties that is easily identified. Entities are the building blocks of a database. Some examples of entities are student, Course and Grade. You represent an entity using a rectangular box that contains the name of the entity.

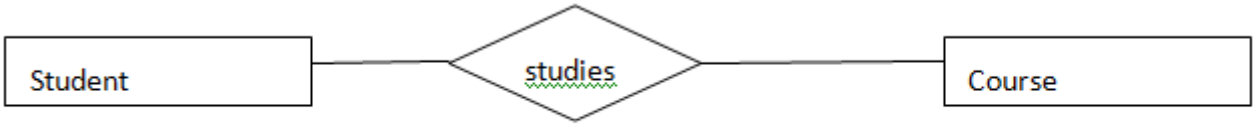


An entity instance is a specific value of an entity. For example, if John and Laura are students, they are instances of the entity Student. Similarly, Science and Mathematics are instances of the entity Course.

Typically, in a problem scenario, you may treat the nouns as entities.

RELATIONSHIPS

A relationship is a crucial part of the design of a database. It is used to establish a connection between a pair of logically related entities. It is an association between entities. Separate entities can have relationships with each other. For example, if students study various courses; the entities are student and course, while the relationship between them is Studies. You represent a relationship between two entities using a diamond labelled with the name of the relationship.



Types of Relationships:

There are three types of relationships that can exist between entities:

- One-to-One
- One-to-many
- Many-to-many

One-to-One (1:1) Relationship:

Two entities have a one-to-one relationship if for every instance of the first entity; there is only one instance of the second entity.

Example:

- One student has one address
- One person has one passport

One-to-Many (1: M) Relationship: This type of relationship exists when one record in one table corresponds to multiple records in another table, but each record in the second table corresponds to only one record in the first table.

For example, in a school database, you could have a student's table and a Courses table. Each student could be enrolled in many courses, but each course could have only one student, so you would have a 1: M relationship between the Students and Courses tables.

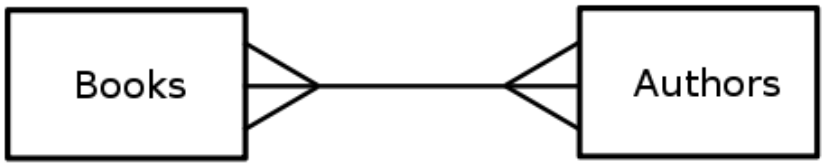
Many-to-one:

Two entities are related in a many-to-one relationship when for multiple instances of the first entity. when more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.

Example:

Many-to-Many (M:M) Relationship: This type of relationship exists when multiple records in one table correspond to multiple records in another table.

For example, in a library database, you could have a Books table and an Authors table. Each book could have multiple authors, and each author could have written multiple books, so you would have a N:M relationship between the Books and Authors tables.



Multiple authors can write multiple books, and multiple books can be written by multiple Authors.

TABLES OR ENTITY

Once the database is designed using an ER diagram, you map the entities and relationships on to tables. **A table is set of rows and columns.** You represent the attributes of the entity as column headings of the table and the data about the entities as rows.

Table Name: Student Personal Details

StudentId	StudentName	StudentLocation	EmployeeContact
1	Siva	Vellore	1234568912
2	Ravi	Gudiyattam	6523217892
3	John	Madhanur	9876223222
4	Jessi	Ambur	1237895672

You can represent the table structure of the table **Student_Personal_Details** as follows:

Student_Personal_Details
StudentId
StudentName
StudentLocation
StudentContact

***

Table is also mentioned as **Entity**
Column is also mentioned as **Attribute**
Row is also mentioned as **Records or Tuples**

Rules to provide Table Name and Attribute Naming Conventions

- **Length:** Table names typically have a length limit, which varies between different DBMSs. For example, in MySQL, the limit is 64 characters, whereas in Oracle it's 30 characters.
- **Characters:** Table names can contain letters, numbers, and underscore characters. Some DBMSs also allow other characters, such as hyphens or dollar signs.
- **Case sensitivity:** Whether table names are case sensitive or not depends on the specific DBMS. In some systems, such as MySQL, table names are case-insensitive, whereas in others, such as PostgreSQL, they are case-sensitive.
- **Keywords:** Some characters are reserved keywords in the database and cannot be used as table names. For example, in SQL, "select" is a reserved keyword and cannot be used as a table name.
- **Naming convention:** It is recommended to follow a consistent naming convention for your table names, such as using snake_case (e.g., "customer_order") or camelCase (e.g., "customerOrder").

KEYS

Enforcing data integrity ensures that the data in the database is valid and correct. Keys play an important role in maintaining data integrity.

The various types of keys that have been identified are the:

- Primary Key
- Foreign Key
- Candidate Key
- Alternate Key
- Composite Key

PRIMARY KEY:

A primary key is a unique identifier for each record in a database table and is used to enforce referential integrity.

Primary Key will have only the Unique values. It does not allow any Null or duplicate values.

StudentId	Student Name	DepartmentID	ContactNumber
1	Arun	DPT_01	1234567893
2	Vijay	DPT_03	2728299393
3	Komal	DPT_01	2292829238
4	Shalini	DPT_02	3728294829

FOREIGN KEY:

When a primary key of one table appears as an attribute in another table, it is called the foreign key in the second table. A foreign key is used to relate two tables.

Consider the tables, **Department** and **Student**

Department:

DepartmentID	DepartmentCode	DepartmentHead	Location
DPT_01	ECE	Jeeva	A-block
DPT_02	CSE	Lalitha	B-block
DPT_03	IT	Aarti	C-block
DPT_04	EEE	Harthi	D-block

Student:

StudentId	Student Name	DepartmentID	ContactNumber
1	Sheeba	DPT_01	123456789
2	Vijay	DPT_03	987654321
3	Vishnu	DPT_01	875435789
4	Kirti	DPT_02	982462827

Since DepartmentID is unique in the Department table, you can choose it as a primary key. Since it appears in the student table as an attribute. It will be the foreign key in the student table.

Candidate Key

It is important to have an attribute in a table that uniquely identifies a row. An attribute or set of attributes that uniquely identifies a row is called a Candidate Key. This attribute has values that are unique. Consider the table **Student.**

Example:

StudentId	Student Name	DepartmentID	ContactNumber
1	Arun	DPT_01	1234567893
2	Vijay	DPT_03	2728299393
3	Vishnu	DPT_01	2292829238
4	Shalini	DPT_02	3728294829

The values of the attributes StudentId, StudentName and ContactNumber are unique in every row. Therefore, all three are candidate keys.

A candidate key is also referred to as a ***Surrogate key.***

Alternate Key

A candidate key that is not chosen as a primary key is an Alternate key.

In the table Student, if you choose StudentId as the primary key. ContactNumber is the Alternate Key.

An alternate key may have null values. A Null value is not to be permitted in a primary key since it would be difficult to uniquely identify rows containing NULL values.

Composite Key

In certain tables, a single attribute cannot be used to identify rows uniquely and a combination of two or more attributes is used as a primary key. Such keys are called as Composite Key.

Consider the following tables, Suppliers.

Composite Key

ITEM

Supplier_ID	Item_ID	Item_Name	Quantity
S_1	I_1	AC	5
S_1	I_2	Inverter	8
S_2	I_2	Inverter	4
S_2	I_3	UPS	15
S_2	I_4	Generator	5
S_3	I_3	UPS	10

Table: 5.5

You can see not all values are unique for any attributes. However, a combination of Supplier_ID and Item_ID results in unique values. Hence, the combination can be used as composite Primary Key.

********** END **********