1. Introduction to SQL

Topics covered
What are Relational Databases?
RDBMS Benefits and Limitations
SQL vs. NoSQL Databases

1.1 What are Relational Databases?

- A relational database is a type of database that stores and organizes data in tables. These tables are made of rows and columns and can be connected through relationships. This model makes it easier to organize and access related information.
- Imagine a school needs to keep track of students. Each student has a name, age, ID and class. Instead of writing this all over the place, we put it into a table with rows and columns. That's the basic idea of a **relational database**.

Relational Database vs Relational Database Management System (RDBMS)

- A **DBMS** (Database Management System) is a software. It is a system that manages databases by providing tools and services for storing, retrieving, and manipulating data. Unlike an RDBMS, it doesn't necessarily use tables. It can store data in various formats such as files, objects, or key-value pairs. DBMS is often used for smaller applications where data relationships aren't complex and doesn't require the structure of relational tables. This makes it suitable for simpler data storage needs where you don't need advanced features like data integrity or complex relationships between data.

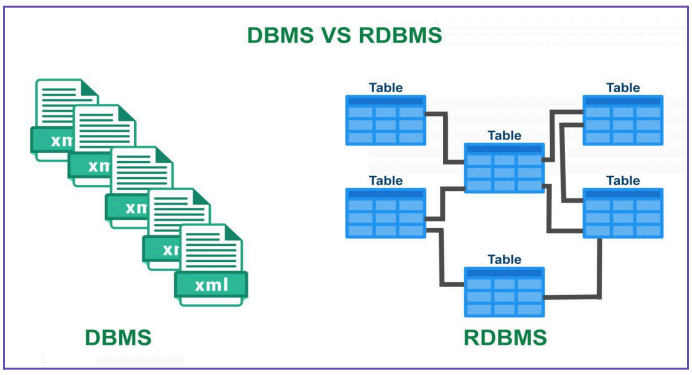

Image source: Google Cloud

- **DBMS** includes many types of databases ranging from older models like flat files or network databases to modern **NoSQL** databases. NoSQL databases

are designed to store large amounts of unstructured or semi-structured data that doesn't fit well into tables. Some common types of NoSQL databases are:

- **Document-based databases** (storing data as documents, often in formats like JSON)
- **Key-value pair databases** (storing data as key-value pairs, like a dictionary)
- **Graph-based databases** (representing data as nodes and edges)

So an **RDBMS** (Relational Database Management System) is a more specific type of **DBMS**. It uses tables to store data and is built for situations where relationships between data are important. It allows users to create, update, insert, or delete data in the system. It also provides data structure, multi-user access, privilege control, and network access. Some well-known RDBMS include MySQL PostgreSQL Oracle and SQLite. RDBMSs follow strict rules to ensure that the data remains consistent and accurate. These rules are known as **ACID properties**: **Atomicity**, **Consistency**, **Isolation**, and **Durability**. These properties ensure that transactions are reliable and that data integrity is maintained even in the case of system failures or power loss.

- In short, while all **RDBMSs** are **DBMSs**, not all **DBMSs** are **RDBMSs**. **RDBMSs** are specialized systems designed for relational data, while **DBMSs** can be more flexible and support various data types.

1.2 RDBMS Benefits and Limitations

- A Relational Database Management System (RDBMS) is one of the most commonly used ways to store and manage data. It has been around since the late 1970s and was originally built to help businesses organize financial records on mainframe computers. Even though other technologies have come along since then, RDBMSs are still popular because of how reliable and organized they are.
- RDBMS works best with structured data. It stores information in tables which makes it easy to connect related data. For example, if one table has a list of customers and another has a list of orders, you can connect them through a shared field like a customer ID. This setup helps to avoid storing the same data repeatedly and keeps everything organized.
- One major benefit of RDBMS is that it follows ACID properties (1.3). These are a set of rules that make sure data stays accurate and reliable even if something goes wrong during a transaction like a system crash. This is especially important when the data needs to stay consistent at all times.
  - When you have multiple pieces of information that need to be related to one another, then it is important to store them in this type of format;

otherwise, you would just end up with a bunch of unrelated facts and features without any ties between them. For instance, if you want to view all the contacts in your phone book then all you need to do is enter one query into the search bar, and instantly see every contact listed there. Saves time from having to manually go through all.

Below is a table outlining the benefits and limitations of relational databases as presented by *Database Town.*

Benefits	Explanation
1. Simplicity of Model	In contrast to other types of database models, the relational database model is much simpler. It does not require any complex queries because it has no query processing or structuring so simple SQL queries are enough to handle.
2. Ease of Use	Users can easily access/retrieve their required information within seconds without indulging into complexity of the database. Structured Query Language (SQL) is used to execute the complex queries of the users
3. Accuracy	A key feature of relational databases is that they're strictly defined and well- organized, so data doesn't get duplicated. Relational databases have accuracy because of their structure with no data duplication
4. Data Integrity	RDBMS databases are also widely used for data integrity as they provide consistency across all tables. The data integrity ensures the features like accuracy and ease of use.

5. Normalization	Database normalization also ensures that a relational database has no variety or variance in its structure and can be manipulated accurately. This ensures that integrity is maintained when using data from this database for your business decisions.
6. Collaboration	Multiple users can access the database to retrieve information at the same time and even if data is being updated
7. Security	Data is secure as Relational Database Management System allows only authorized users to directly access the data. No unauthorized user can access the information

Source: Database Town, *Relational Database Benefits and Limitations*. Retrieved from: https://databasetown.com/relational-database-benefits-and-limitations/

1.3 SQL Databases vs. NoSQL Databases

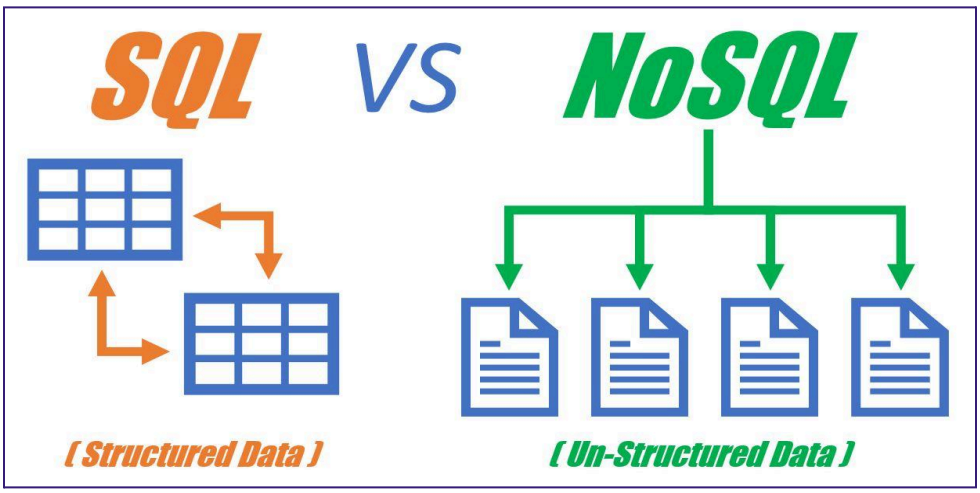

Image source: gowithcode.com

SQL Databases

- Relational Database Structure
    - SQL databases are based on a **relational database structure**. Just like what was explained in the earlier pages about relational databases, it means data is stored in tables which consist of rows and columns. SQL databases are best suited for situations where the relationships between data are important and need to be maintained accurately.
- Pre-planning Required
    - SQL databases require more **pre-planning** when designing the database and the reason is because you must define how the data is structured, how different tables will relate to each other, and the constraints needed to ensure data integrity. This upfront design helps with organizing and querying data but can be rigid when changes need to be made later on.
- Vertical Scaling
    - As your database grows and experiences more traffic, SQL databases typically scale vertically. This means adding more resources such as RAM or CPU power to a **single server**. For example, you can increase the capacity of your MySQL server to handle more requests and data. Since SQL databases are relational and follow a strict schema, it's harder to split data across multiple servers. So instead of spreading the load, they scale **upward** by upgrading the existing machine.
    - To visualize: it's like upgrading one big truck to carry more packages instead of using many smaller trucks. However, this method has limitations. There's only so much you can add to one truck (server), and the upgrades can be expensive.
- ACID Properties
    - SQL databases follow ACID properties which stands for **Atomicity, Consistency, Isolation, Durability.**
    - In totality, it provides a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

NoSQL Databases

- NoSQL databases are non-relational

- NoSQL or Not Only SQL refers to **non-relational databases**. Unlike SQL databases, NoSQL databases don't use tables with rows and columns. Instead they store data in different formats which typically fall into the main types of NoSQL data models:

    1. **Column-Oriented Stores:** Data is stored in cells grouped in a virtually unlimited number of columns rather than rows. This structure is especially efficient for querying large datasets, particularly when handling millions of rows. Popular column-oriented databases include **Apache Cassandra**, **HBase**, and **Google Bigtable**.

    2. **Key-value Stores:** Each piece of data is stored as a key-value pair, also known as dictionaries or maps. The **key** is a unique identifier (like a variable name), and the **value** is the associated data. These databases are commonly used for caching, session storage, and handling large volumes of simple data. Popular key-value store databases include **Redis**, **Riak**, and **DynamoDB**.

    3. **Document-Stores:** Data is stored in documents, which can be in formats like **JSON**, **BSON**, or **XML**. These documents can have nested fields (data stored inside other data) and don't need to follow a strict schema. This flexibility is particularly useful for semi-structured data. Popular document store databases include **MongoDB**, **CouchDB**, and **Firebase Firestore**.

    4. **Graph databases:** Data is represented as **nodes** (entities) and **edges** (relationships between entities). Graph databases are ideal for understanding how different pieces of data are connected and for managing complex relationships. Popular graph databases include **Neo4j**, **OrientDB**, and **ArangoDB**.

- Less Pre-planning Required

    - Unlike SQL, NoSQL databases don't need as much pre-planning. You don't have to define all relationships and structures upfront, which gives you more flexibility. This makes NoSQL databases great for applications where data needs to evolve quickly or where relationships between data aren't complex.

- Horizontal Scaling

    - Instead of relying on a single powerful server (vertical scaling), NoSQL systems handle more data and users by **adding more servers** to share the load. This is known as **horizontal scaling**. Each server manages part of the data, and they work together as a system. For example, in a

MongoDB cluster, different nodes handle different parts of the database. This is possible because NoSQL databases are non-relational and don't rely on a strict schema, making it easier to distribute data.

- ➢ To visualize: it's like using many smaller trucks to deliver packages instead of upgrading one big truck. This approach is often more cost-effective and allows the system to grow easily as data increases.

- ❖ CAP theorem, or CAP principle

  - ➢ NoSQL databases follow the **CAP Theorem**, a fundamental principle for comprehending these trade-offs in distributed systems which says a distributed system can only guarantee two out of three: **Consistency, Availability, Partition Tolerance.**

  - ➢ The key point is that network problems and large-scale data cause challenges. You must choose two that matter most based on your needs. For example, if you choose consistency over partition tolerance, you might sacrifice availability because you can't always respond to requests. If you choose availability over partition tolerance, you might return inconsistent data. And if you choose consistency and availability, you might lose the ability to function during network issues.

    - 1. Consistency vs. Availability

      - Problem: The combination of consistency and availability is not possible in distributed systems. If the system prioritizes consistency, it ensures that all data across servers is the same. However synchronizing the data takes time and during this process the system might not be able to respond quickly. This means the system could be unavailable for fast replies (no availability).

    - 2. Availability vs. Partition Tolerance

      - Problem: If the system prioritizes availability, it ensures that the system always responds even if some servers can't communicate with each other. However, when this happens, the system might not guarantee partition tolerance. This means that even if parts of the network are unreachable, the system will still respond. But this can lead to incomplete or outdated data being returned (no consistency).

    - 3. Consistency vs. Partition Tolerance

- Problem: In the case of a network issue, the system needs to decide between maintaining consistency (ensuring all servers have the same data) or partition tolerance (allowing some servers to fail but still responding to requests). Consistency is chosen over availability for critical applications where latest data plays an important role such as stock market application, ticket booking application, banking, etc. where problems will arise due to old data present to users of application.