# Fundamental Relational Databases Concepts.

## Review of Data Fundamentals

Data includes facts, observations, numbers, symbols, images or a mix of everything.

Data structure

We have:

* Structure Data: which follows a predefined format and adheres to strict schema, it includes tables with rows and columns and ensures consistency and easy retrieval. Ex: SQL database, spreadsheet, online forms.
* Unstructured data: Lacks a specific format or organisation, does not conform to any predefined rules. Ex: text file that contain free form document, Media files, such video, images, audio and video, web pages, and social media contents.
* Semi structure data: it possess some organisational properties, does not follow tabular structure, and employs hierarchical structures or tags. Provides a balance between flexibility and structure. Ex: Json file, XML documents, email.

Data sources

Traditional databases, web scraping, API, flat files and XML data sets, IOT devices with sensors, social media platforms data streams and feeds.

**Common Files format**

We have delimited:

* Delimited text files: Rows of variables separated by character such as csv, tsv files.
* Spreadsheet: data exists in rows and columns for manipulation, creates csv files.
* Language files: includes XML and JSON, set rules and structures for encoding data.

Data Repositories

They actively store, manage and organise data, they offer a structured framework for retrieval and administration, categories include: Relational databases and non relational databases.

1. Relational Databases:

Includes structured data stored in related tables to minimise duplication, along with supporting system known as RDBMS such IBM DB2, Microsoft SQL Server Oracle and MySQL.

Relational Databases are primarily designed for OLTP systems, store high volumes of operational data. Ensures transactional integrity. OLAP includes various storage solutions, focuses on querying and analysing large data sets. For example Sourcing data form CRM generating sales.

1. Non relational databases:

Offer flexibility in handling diverse and unstructured data, examples include MongoDB, Cassandra and redis.

## Information and Data models

An Information Model is an abstract, formal representation of entities that includes their properties, relationships and the operations that can be performed on them.

An Information Model is at the conceptual level and defines relationships between objects. Data Models are defined at a more concrete level, are specific and include details. A data model is the blueprint of any database system.

There are several types of Information Models. The most familiar is the Hierarchical, typically used to show organization charts.

The root of the tree is the parent node followed by child nodes. A child node cannot have more than one parent; however, a parent can have many child nodes. The first hierarchical database management system was the Information Management System released by IBM in 1968 and was originally built as the database for the Apollo space program.

Relational Model

The Relational Model is the most used data model for databases because this model allows for data independence. Data is stored in a simple data structure, tables. This provides logical data independence, physical data independence, and physical storage independence.

An Entity-Relationship Data Model, or ER Data Model, is an alternative to a relational data model. Using a simplified library database as an example, this figure shows an Entity-Relationship Diagram, or ERD, that represents entities (called tables) and their relationships.

Entity Relationship Model

An Entity-Relationship Model proposes thinking of a database as a collection of entities. Rather than being used as a model on its own, the ER Model is used as a tool to design relational databases.

In the ER Model, entities are objects that exist independently of any other entities in the database. It is simple to convert an ER Diagram into a collection of tables. The building blocks of an ER Diagram are **entities and attributes**. Entities have attributes, which are the data elements that characterise the entity. Attributes tell us more about the entity. In an ER Diagram, an entity is drawn as a **rectangle**, and attributes are drawn as **ovals**. Entities can be a noun (person, place, or thing).

## ERDs and types of relationship

Building Blocks

The building blocks of relationship are:

* Entities
* Relationship sets
* Crows foot notations

The building blocks of a relationship are entities, relationship sets, and crows foot notations. In a one-to-one relationship, one entity is associated with one and only one instance of another entity. For example, when one book has only one author. In a one-to-many relationship, one entity is associated with one or more instances of another entity. For example, when one book has many authors. In a many to many relationship, many instances of an entity are associated with many instances of another entity. For example, when many authors write many different books.

## Mapping Entities to tables

Entity-Relationship Diagrams (ERD) are the basic foundation for designing a database To translate an ERD into a relational database table: The entity becomes the table The attributes become columns in the table.

## Data types

In addition to the various built-in data types covered in this video, many relational databases also allow you to create your own custom or “user defined” data types (UDTs) that are derived or extended from the built in types.

INT, Boolean, Varchar, char, binary string, large object, date, time, timestamp.

Advantages of using data types

* Data integrity
* Data sorting
* Range selection
* Data calculations
* Use of standard functions

Data types define the type of data that can be stored in a column. There are many different data types for all kinds of data. Using the correct data type for a column has many advantages.

## Relational Model Concepts

The relational model introduced in 1970 offers a powerful approach to organizing and understanding data. It centers around two fundamental concepts, sets and relations. Let us define each of these terms.

Sets

A set is a collection of unique elements without a specified order, comprising items of a similar type, usually curly braces denote sets with the elements listed inside or described using set builder notation with a condition. Sets play a critical role across various disciplines in modern mathematics, influencing algebra, geometry, and probability.

Set operations, see videos.

Aspects of relation

In addition to sets, the study of relations is essential, it is a mathematical concept based on the idea of sets. Relations describe connections between elements of sets and are crucial in set theory and logic. Common types include binary relations and ordered pairs. A binary relation establishes a connection between two elements. For instance, the less than relation illustrates the relationship between two numbers like 3 is less than 5. Ordered pairs a subset of the cartesian product A times B represent a binary relation on sets A and B, denoted as parentheses A comma B parentheses. Relations demonstrate various properties contributing to their significance in mathematical analysis.

Sets are collections without a specified order. Key set operations, including membership, subsets, union, and intersection, aid in understanding logical relations between sets. Relations describe connections between set elements, impacting set theory and logic. Common types of relation include binary relations and ordered pairs. A relation consists of two essential components, the relation schema and the relation instance. Degree indicates the number of columns, and cardinality is the count of row.

Congratulations! You have completed this lesson. At this point in the course, you know:

* The relational model is the most used data model for databases because this model allows for logical data independence, physical data independence, and physical storage independence.
* Entities are objects that exist independently of any other entities in the database, while attributes are the data elements that characterize the entity.
* The building blocks of a relationship are entities, relationship sets, and crows foot notations.
* Relationships can be one-to-one, one-to-many, or many-to-many.
* When translating an Entity-Relationship Diagram to a relational database table, the entity becomes the table and the attributes become columns in the table.
* Data types define the type of data that can be stored in a column and can include character strings, numeric values, dates/times, Boolean values and more.
* The advantages of using the correct data type for a column are data integrity, data sorting, range selection, data calculations, and the of standard functions.
* In a relational model, a relation is made up of two parts: A relation schema specifying the name of a relation and the attributes and a relation instance, which is a table made up of the attributes, or columns, and the tuples, or rows.
* Degree refers to the number of attributes, or columns, in a relation.
* Cardinality refers to the number of tuples, or rows in a relation.

# Introducing Relational Database Products

## Data Architecture

### Deployment Topologies

The deployment topology you use for your database is determined by how it will be used and accessed.

--Local-- For example, you can deploy a small database which requires limited user access on a local desktop. The database resides on the user's system and access is often limited to a single user. This deployment topology is sometimes known as single-tier architecture.

It is useful for **development and testing** or when the database is embedded in a local application.

--Client/Server-- You can deploy a  larger database that many users must access in a client-server architecture. In this scenario, the database resides on a remote server and users access it from client systems, often through a web page or local application. Some scenarios employ **a middle-tier (or an application server layer)** between the application client and the remote database server. These client/server deployments are commonly used for multi-user scenarios and typical of production environments

--Cloud-- Deploying a database in the Cloud is an increasingly **popular option**. In a Cloud deployment, the database resides in a Cloud environment and has all the **advantages of a cloud-based service**. **You don’t have to download or install the database software**, **you don’t have to maintain the supporting infrastructure, and it is easy for users to access from wherever they are**, whatever they are doing, so long as they have an internet connection. In Cloud deployments, client apps and users typically **access the database through an application server layer or interface** in the cloud. Cloud deployments are **very flexible**; you can use them for development, testing, and full production environments.

### Client/Server topology or 2-tier Database Architecture

In a 2-Tier database architecture, the database server and the application run in 2 separate tiers. The application in the client tier connects to the database server through some sort of **database interface such as an API or Framework**, which can be dependent on the programming language the application is written in. The database interface communicates with the database server through a **Database Client or API** that is installed on client system. **The database management system software (DBMS)** on the server includes multiple layers which on a high level can be categorized as: **Data Access layer Database Engine layer Database Storage layer**. The Data Access layer server includes interfaces for different types of clients which can include **data industry standard APIs such as JDBC and ODBC, Command Line Processor (CLP) interfaces as well vendor specific or proprietary interfaces**. The database server also contains an Engine which **compiles queries, and retrieves and processes the data and returns the result set**. The database storage or persistence layer **is where the data is stored**, which may be on local storage on the same device, or reside physically on network storage or specialized storage appliances.

### 3-tier Database Architecture

In most production environments, especially in the last 20-25 years, the database server is typically not accessed directly (except by administrators). The client applications and users typically go through **a middle tier** such as a **web application server, a BI server**, etc, and hence referred to as 3-tier architecture. In this architecture **the application presentation layer and the business logic layer reside in different tiers**. The presentation layer is the interface with which end-users interact, which could be a traditional desktop application, a web browser or a mobile application. The client application communicates with an application server over the network. The application server encapsulates the application and business logic and communicates with the database server through **a database API or driver**.

### Recap

Databases are deployed in different topologies, depending on which best suits the processing and access requirements. **A single-tier topology** is one where the database is installed on a user’s local desktop. It is useful for small databases that only require single user access. In **2-tier database topologies** the database resides on a remote server and users access it from client systems. **In 3-tier database topologies** the database resides on a remote server and users access it through an application server or a middle-tier.

In Cloud deployments the database resides in the cloud, and users access it through an application server layer or another interface that also resides in the cloud.

## Distributed Architecture and Clustered Databases

### Distributed Architecture

In our exploration of database architectures, we've primarily focused on single-server configurations. However, **for critical or large-scale workloads where high availability or scalability is important**, relational database management systems, RDBMSs, offer **distributed architectures**. These distributed database architectures involve clusters of machines interconnected through a network, distributing data processing and storage tasks.

The approach brings about notable benefits including enhanced **scalability, fault tolerance, and overall performance improvements**.

Let's now discuss the types of database architecture. The common types of database architecture include **shared disk architecture, shared nothing architecture, and combination and specialized architectures.**

### Shared Disk Architecture

Shared disk architecture involves **multiple database servers processing workloads in parallel**. Each server establishes a connection to **shared storage and communications with other servers using high-speed interconnection**. The shared disk architecture also facilitates the **effective distribution of workloads**, ensuring scalability as the demand for processing power grows. In the event of a server failure, **a mechanism is in place to reroute clients seamlessly to other servers**, maintaining **high availability and minimizing service disruptions**.

### Shared Nothing Architecture

Shared nothing architecture which utilizes **either replication or partitioning techniques**. The approach allows for **the effective distribution of client workloads across multiple nodes**, promoting **parallel processing and efficient resource utilization**. One of the key advantages lies in **enhanced fault tolerance achieved by rerouting clients to alternative nodes** in the event of a server failure.

### Combination and specialised architectures

Certain distributed database architectures employ a combination of **shared disk, shared nothing, replication or partitioning techniques**. Additionally, they integrate **specialized hardware components** to achieve specific goals related to availability and scalability.

### Common Techniques

Some of the common techniques include **database replication, database partitioning and sharding.**

**Database Replication:** Database replication is a technique that involves copying changes from one database server to one or more replicas. This process distributes the client workload across servers, leading to improved performance. When the replica resides in the same location, we call it a **high availability, HA, replica**. If the primary database server experiences a failure due to software or hardware issues, the system redirects clients to HA replica. To mitigate broader disasters, organizations establish replicas in geographically distributed locations. This guarantees that during instances of complete data center outages, be it due to power loss, fire, earthquake or flood, clients can be rerouted to disaster recovery replicas.

**Database Partitioning:** An alternative strategy involves partitioning tables with **substantial data into logical segments**, each containing a subset of the overall data, e.g., sales records for different quarters. This technique, known as **sharding**, places these partitions on separate nodes in a cluster. Each shard possesses its compute resources, processing, memory, and storage to operate on its specific subset of data. When a client issues a query, it is processed in parallel across multiple nodes or shards, and the results from different nodes are synthesized and returned to the client. As data or query workloads increase, additional shards and nodes can be seamlessly added to the database cluster, facilitating increased parallel processing and improved overall performance. **Database partitioning and sharding are particularly prevalent in handling data warehousing and business intelligence workloads that involve extensive volumes of data.**

### Recap

RDBMSs offer distributed architectures for critical or large-scale workloads. Shared disk allows parallel processing with mechanisms for high availability during server failures. Shared nothing employs replication or partitioning for optimized performance. Database replication involves copying changes from one database server to one or more replicas. Sharding involves placing partitions on separate nodes, facilitating increased parallel processing and improved overall performance.