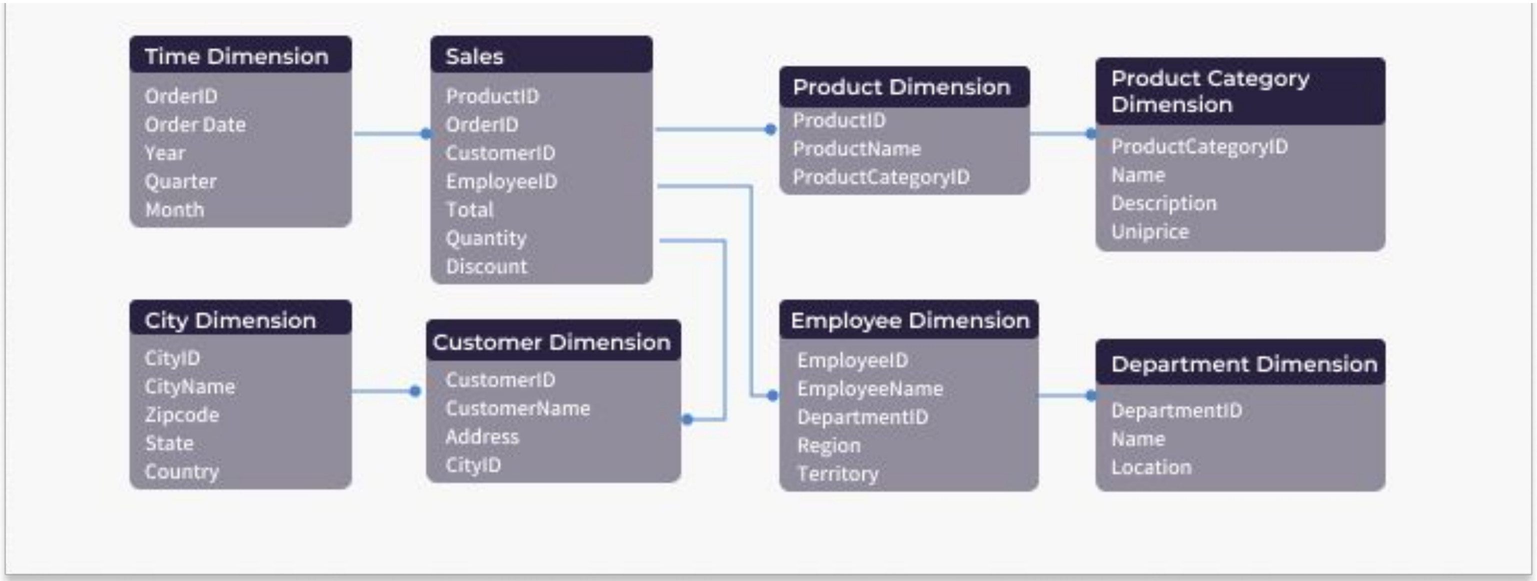
Databases

Database

A database is an organized collection of data that can be easily accessed, managed, and updated. Databases are crucial for storing and managing information in various applications, ranging from websites to large enterprise systems.



What is DBMS?

Database Management Systems (DBMS) are software systems used to store, retrieve, and run queries on data.

A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.

DBMS manage the data, the database engine, and the database schema, allowing for data to be manipulated or extracted by users and other programs.

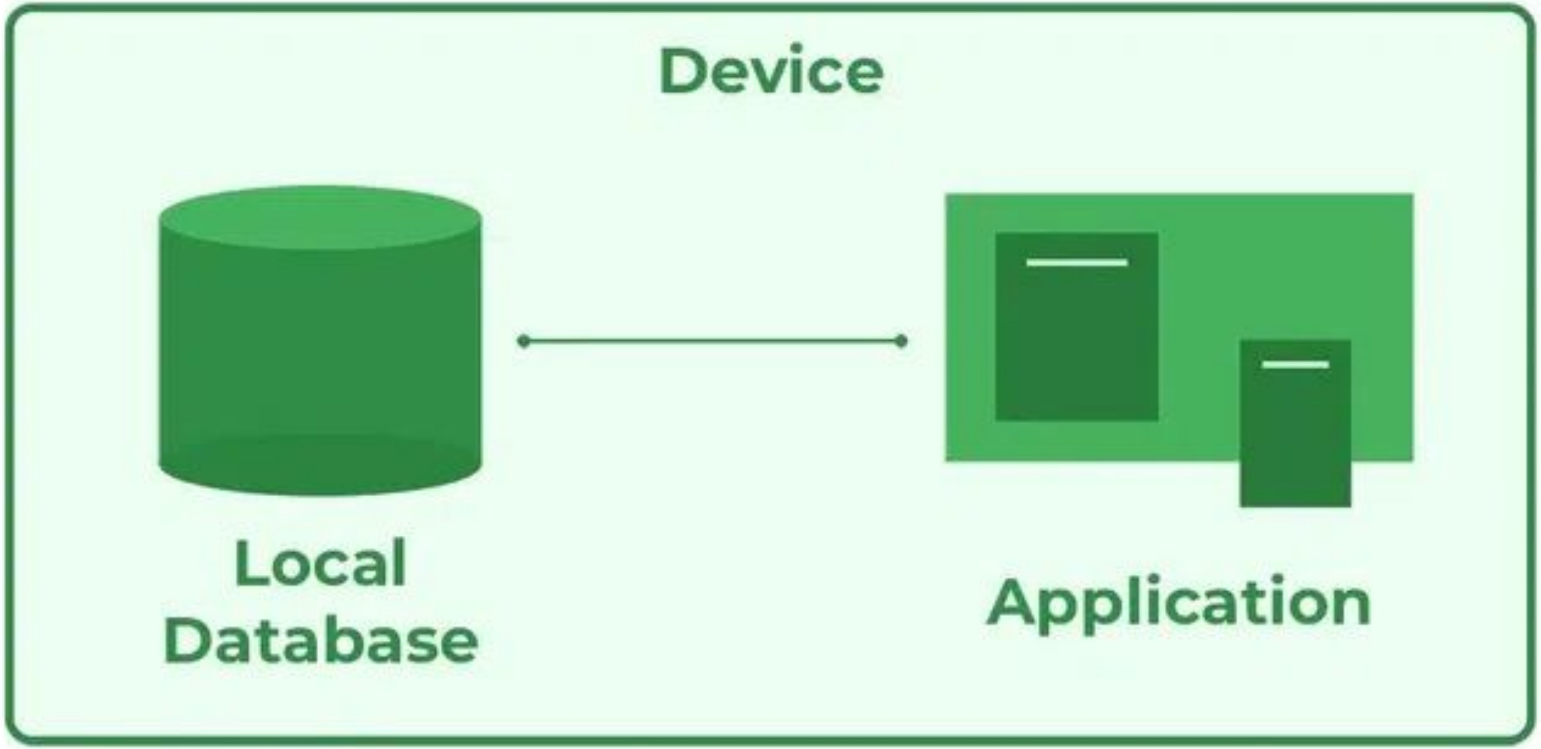
This helps provide data security, data integrity, concurrency, and uniform data administration procedures.

Database Architecture

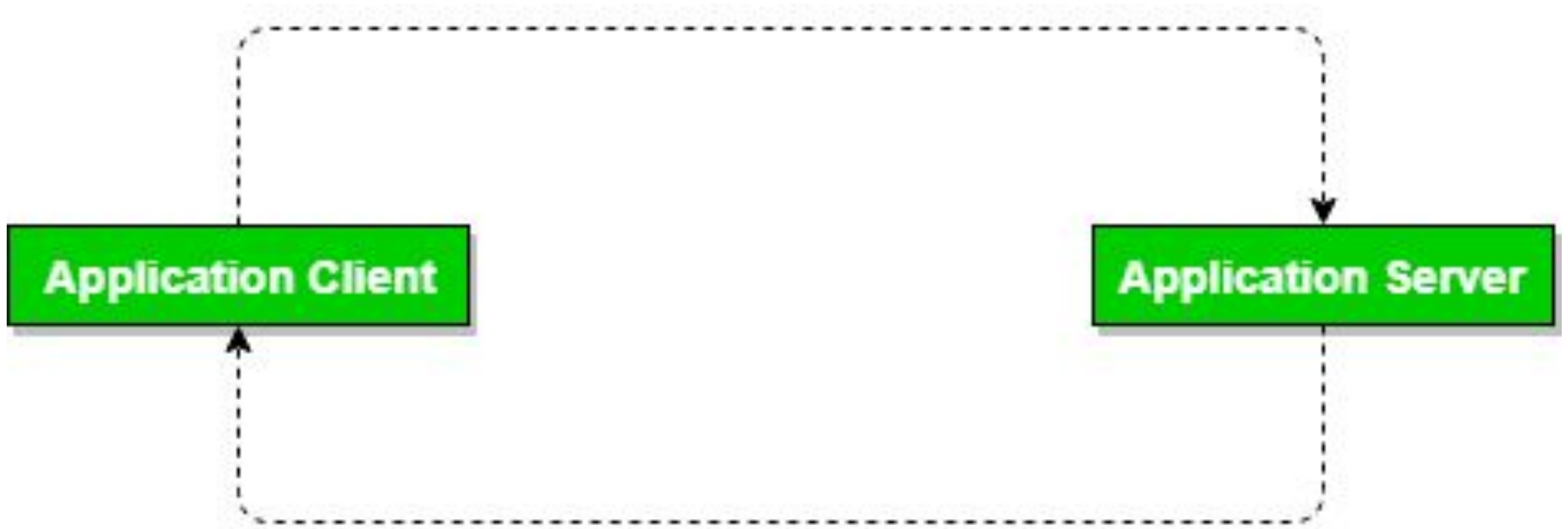
Database architecture refers to the design and structure of a database system, including how data is stored, accessed, and managed. There are two main types of database architecture:

1. **Single-Tier Architecture:** The database and the application accessing it reside on the same machine. This architecture is usually used for small-scale applications. It is simple to manage, cost-effective but limited scalability, prone to performance bottlenecks.
2. **Multi-Tier Architecture:** In this architecture, the database is separated from the application, often residing on different servers. Typically includes:
    - **Presentation Tier:** User interface layer.
    - **Application Tier:** Business logic layer.
    - **Database Tier:** Data storage layer.

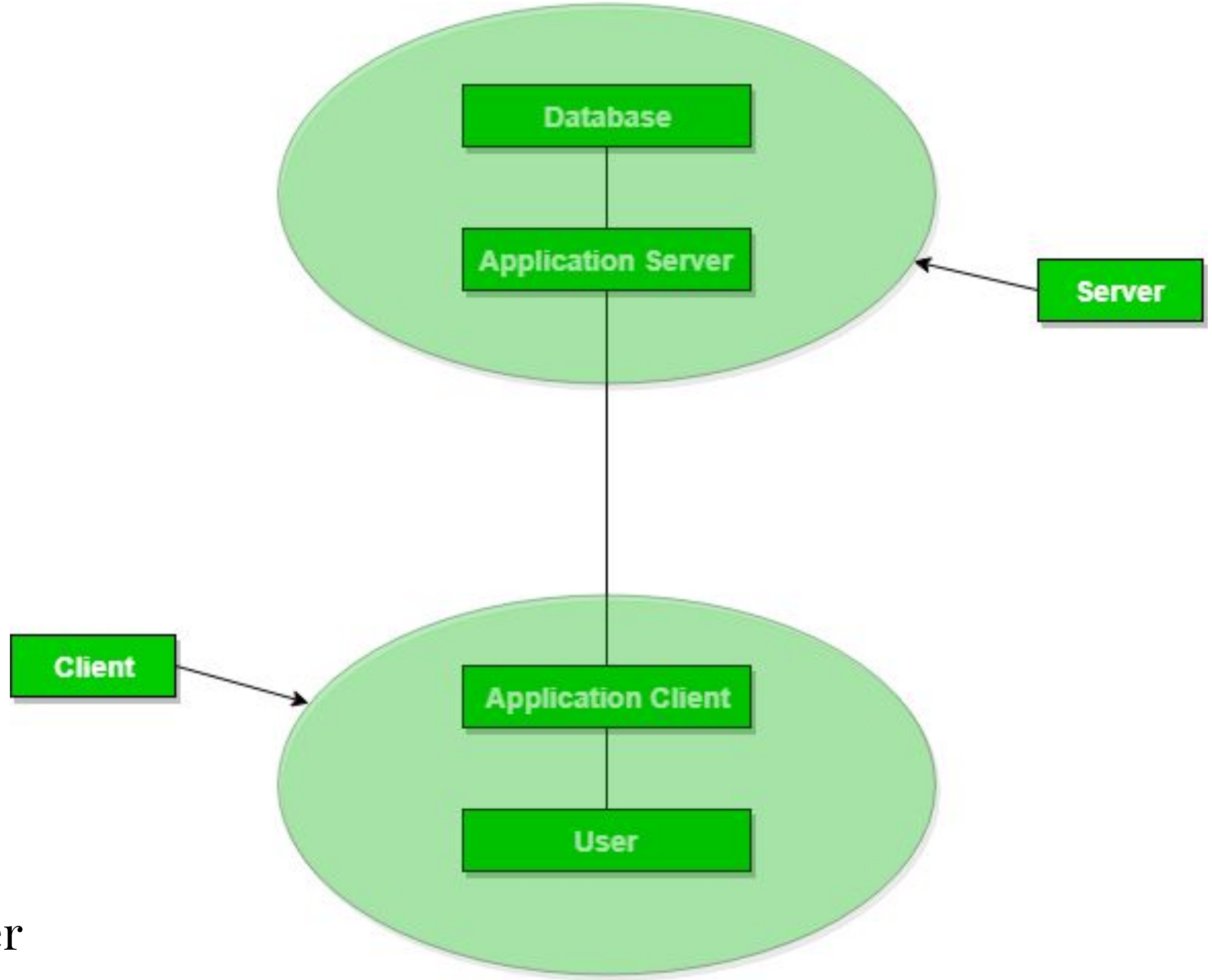
It helps for improved scalability, better performance, enhanced security but can be more complex to setup and manage, higher cost



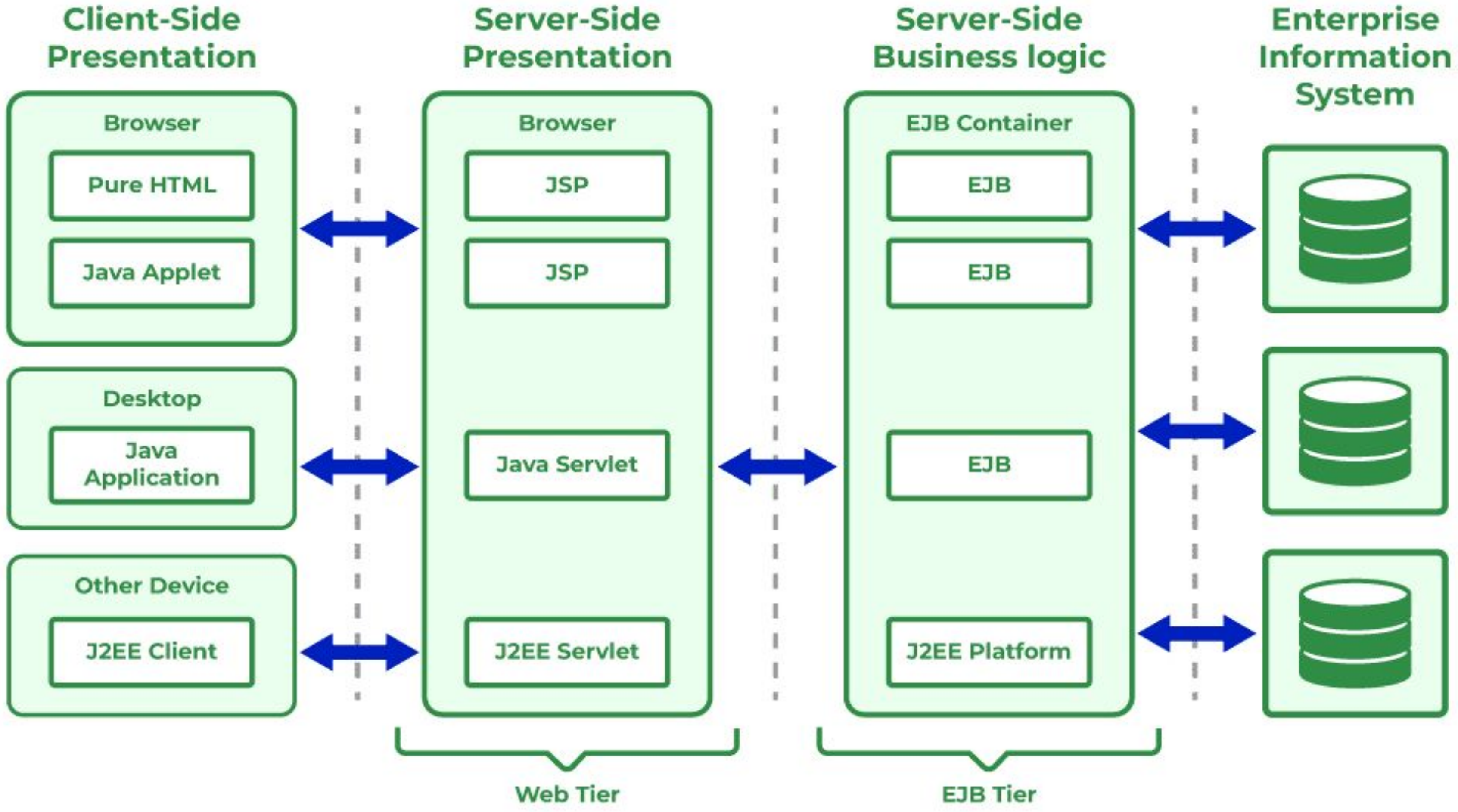
1 tier



2 tier


Database
Application Server
Server
Client
Application Client
User

3 tier

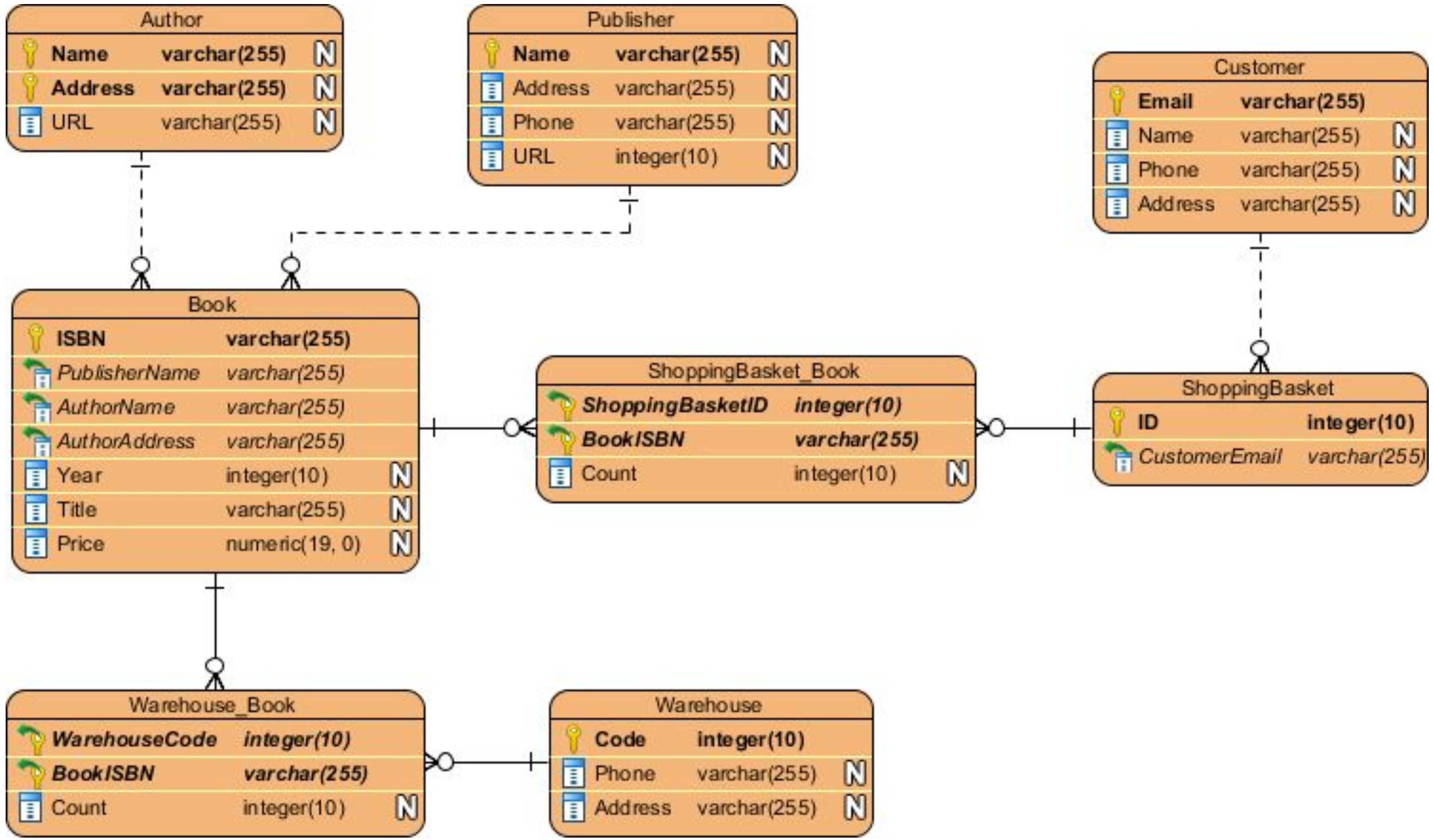

Client-Side Presentation
Server-Side Presentation
Server-Side Business logic
Enterprise Information System
Browser
Pure HTML
Java Applet
Desktop
Java Application
Other Device
J2EE Client
Browser
JSP
JSP
Java Servlet
J2EE Servlet
EJB Container
EJB
EJB
EJB
J2EE Platform
Web Tier
EJB Tier

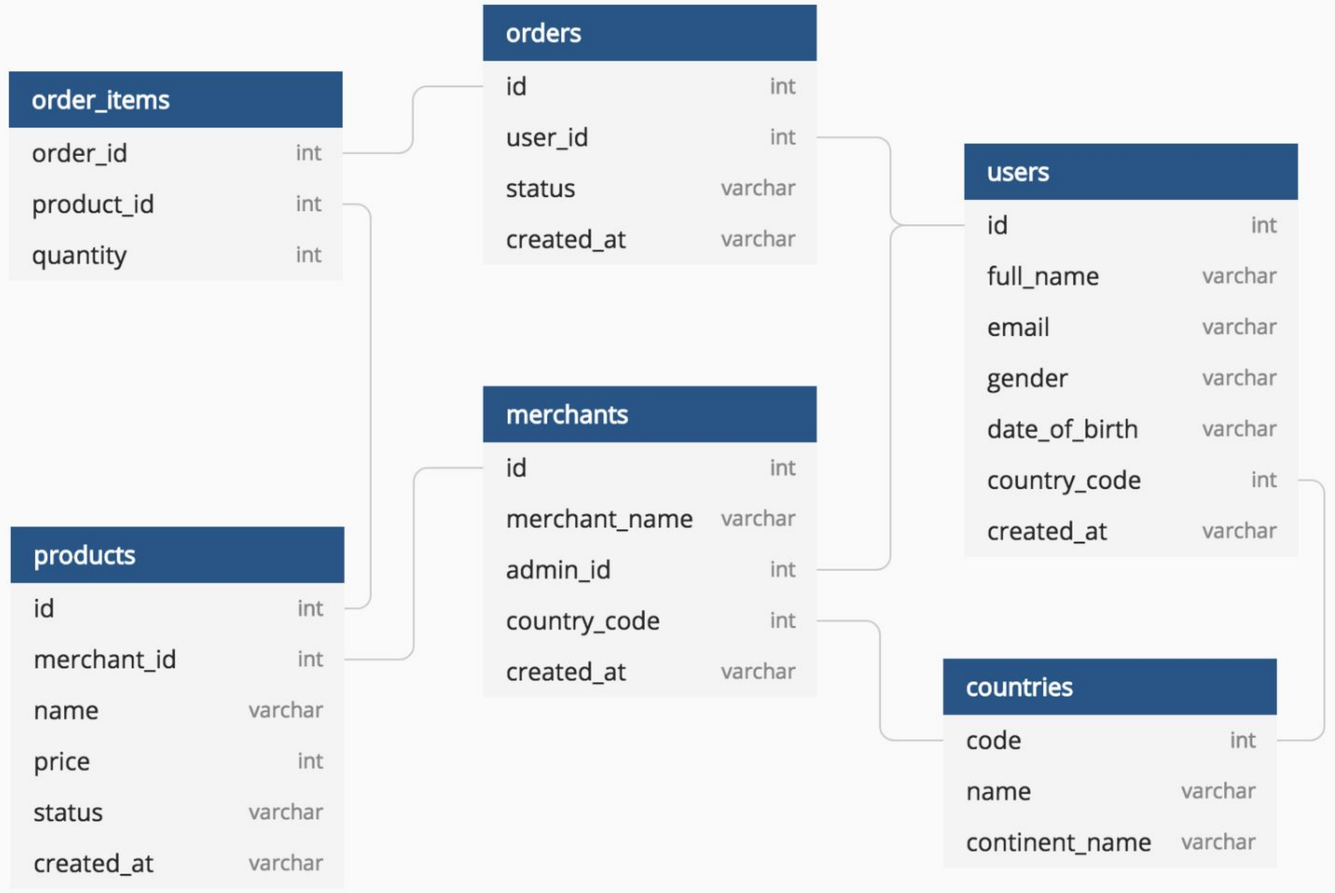
Types of Database

There are several types of databases, each designed to handle different data management needs:

1. **Relational Databases (RDBMS):** These databases store data in tables, which consist of rows and columns. Each table is related to others through keys.

- **Examples:** MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.
- **Use Cases:** They are ideal for applications requiring structured data, such as customer relationship management (CRM) systems, e-commerce platforms, and financial applications.


Author
Name varchar(255) N
Address varchar(255) N
URL varchar(255) N
Publisher
Name varchar(255) N
Address varchar(255) N
Phone varchar(255) N
URL integer(10) N
Customer
Email varchar(255)
Name varchar(255) N
Phone varchar(255) N
Address varchar(255) N
Book
ISBN varchar(255)
PublisherName varchar(255)
AuthorName varchar(255)
AuthorAddress varchar(255)
Year integer(10) N
Title varchar(255) N
Price numeric(19, 0) N
ShoppingBasket_Book
ShoppingBasketID integer(10)
BookISBN varchar(255)
Count integer(10) N
ShoppingBasket
ID integer(10)
CustomerEmail varchar(255)
Warehouse_Book
WarehouseCode integer(10)
BookISBN varchar(255)
Count integer(10) N
Warehouse
Code integer(10)
Phone varchar(255) N
Address varchar(255) N



order_items	
order_id	int
product_id	int
quantity	int

orders	
id	int
user_id	int
status	varchar
created_at	varchar

users	
id	int
full_name	varchar
email	varchar
gender	varchar
date_of_birth	varchar
country_code	int
created_at	varchar

merchants	
id	int
merchant_name	varchar
admin_id	int
country_code	int
created_at	varchar

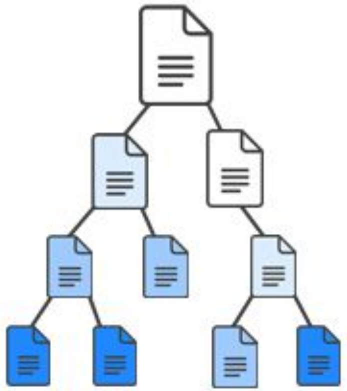
products	
id	int
merchant_id	int
name	varchar
price	int
status	varchar
created_at	varchar

countries	
code	int
name	varchar
continent_name	varchar

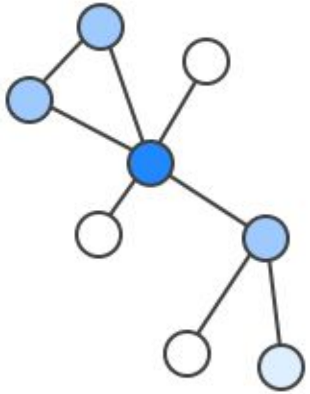
2. **Non-relational Databases:** Unlike relational databases, NoSQL databases do not store data in tables. They are designed for unstructured or semi-structured data.

- **Types:**
    - **Document Stores:** Store data in documents, typically JSON or BSON. Examples: MongoDB, CouchDB.
    - **Key-Value Stores:** Store data as key-value pairs. Examples: Redis, Amazon DynamoDB.
    - **Column-Family Stores:** Store data in columns instead of rows. Examples: Cassandra, HBase.
    - **Graph Databases:** Focus on relationships between data points. Examples: Neo4j, Amazon Neptune.
- **Use Cases:** NoSQL databases are used for big data applications, real-time web apps, and scenarios requiring scalability and flexibility.

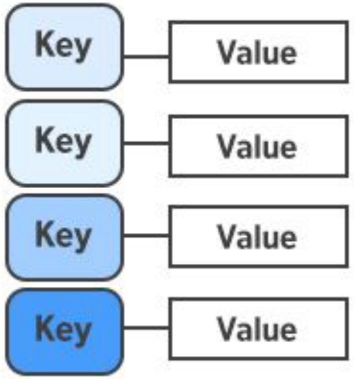
Document



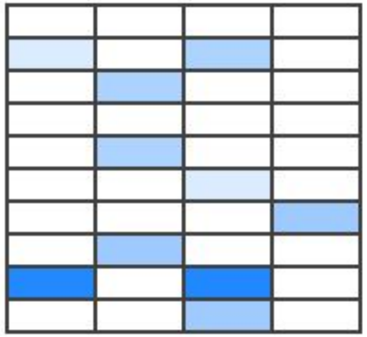
Graph

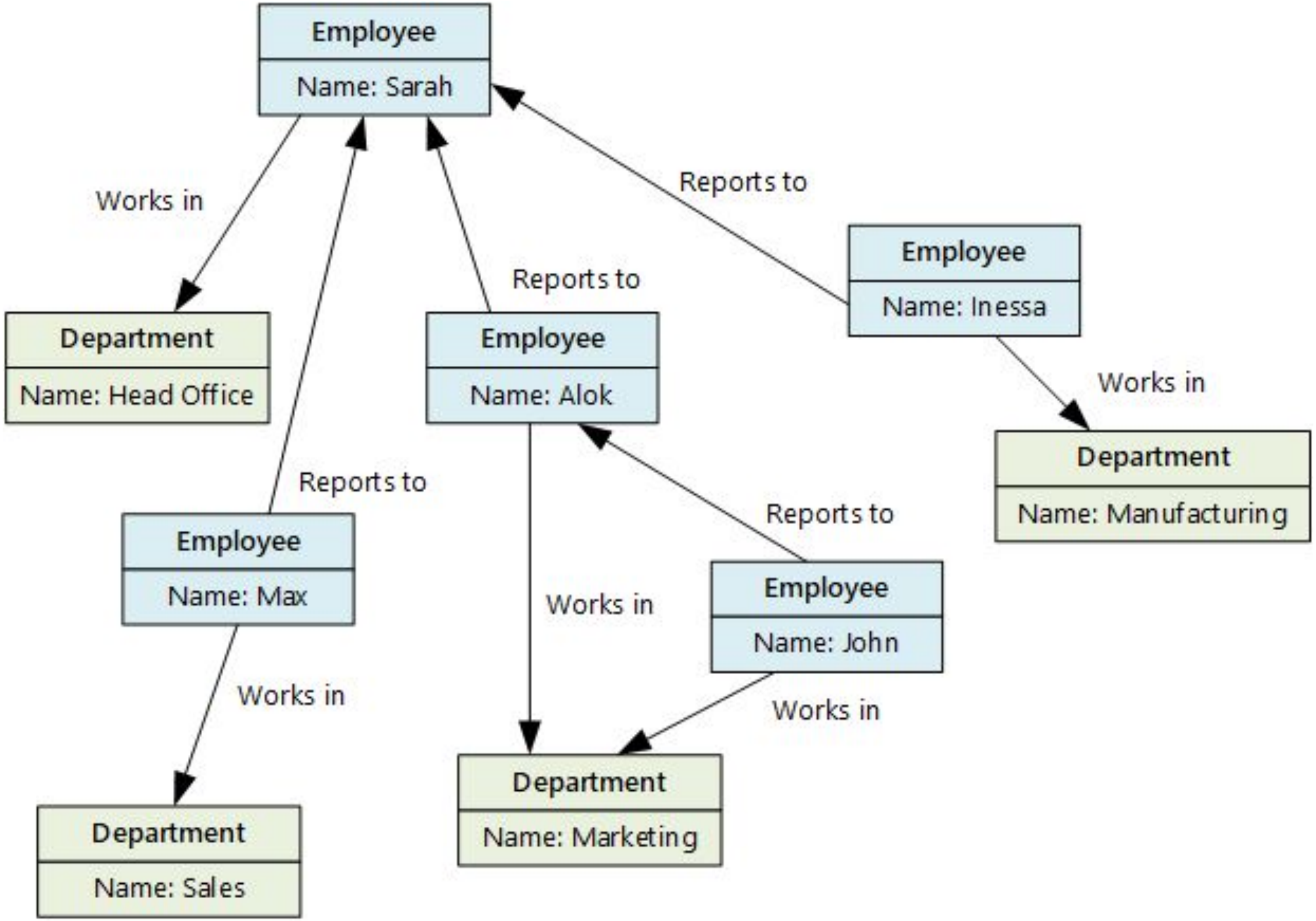


Key-Value



Wide-column



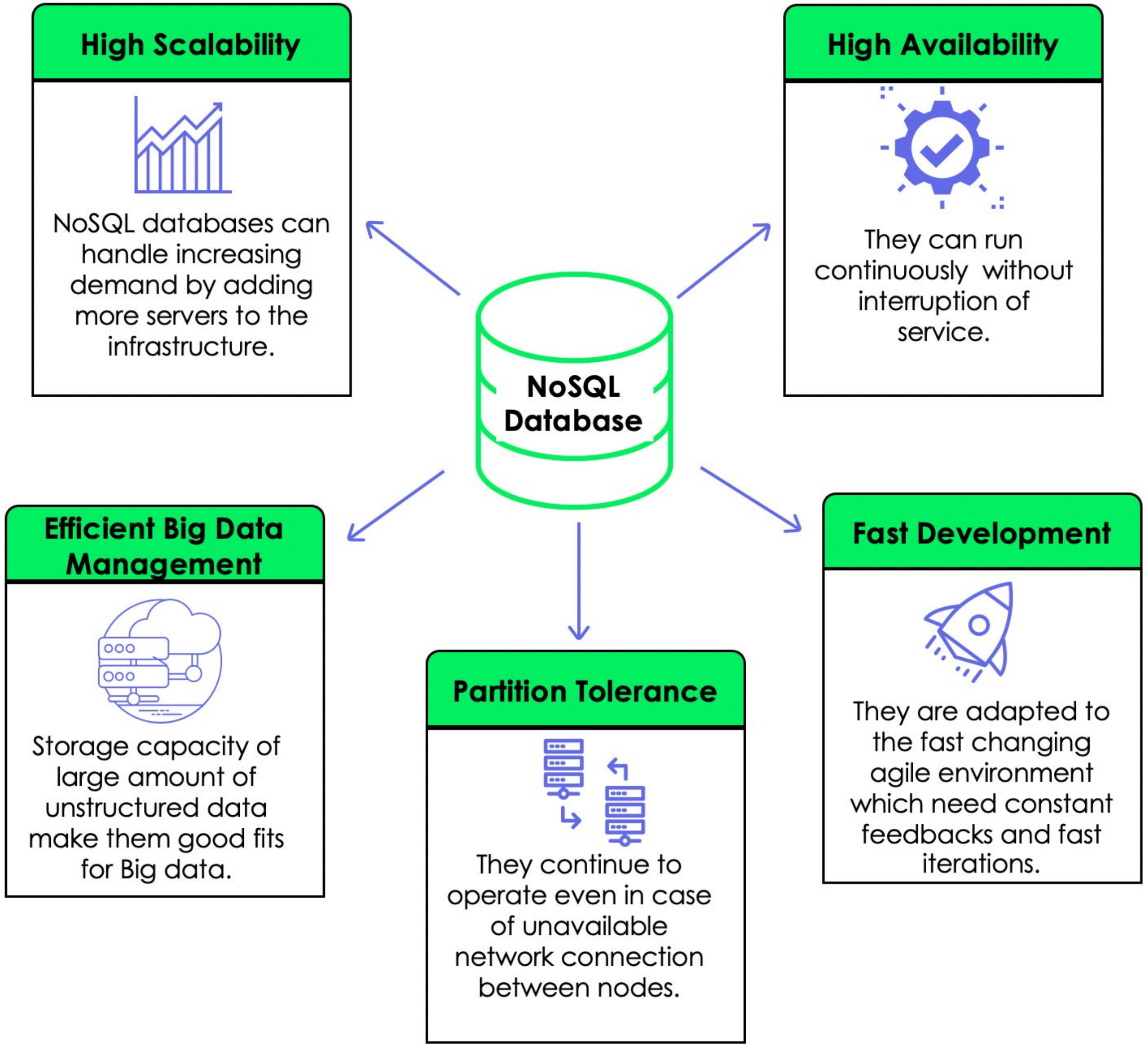

Employee
Name: Sarah
Works in
Reports to
Department
Name: Head Office
Reports to
Employee
Name: Alok
Employee
Name: Inessa
Works in
Department
Name: Manufacturing
Reports to
Employee
Name: Max
Reports to
Works in
Employee
Name: John
Works in
Works in
Department
Name: Sales
Department
Name: Marketing

Non-relational DB

Non-relational Database

Non-relational databases, often referred to as NoSQL databases, differ from traditional relational databases in that they do not require a fixed schema and can store data in a variety of formats such as key-value pairs, documents, wide-column stores, and graphs.

These databases are designed to handle large volumes of unstructured or semi-structured data, making them ideal for modern applications that require flexibility, scalability, and fast access to data.



High Scalability

NoSQL databases can handle increasing demand by adding more servers to the infrastructure.

High Availability

They can run continuously without interruption of service.

Efficient Big Data Management

Storage capacity of large amount of unstructured data make them good fits for Big data.

Partition Tolerance

They continue to operate even in case of unavailable network connection between nodes.

Fast Development

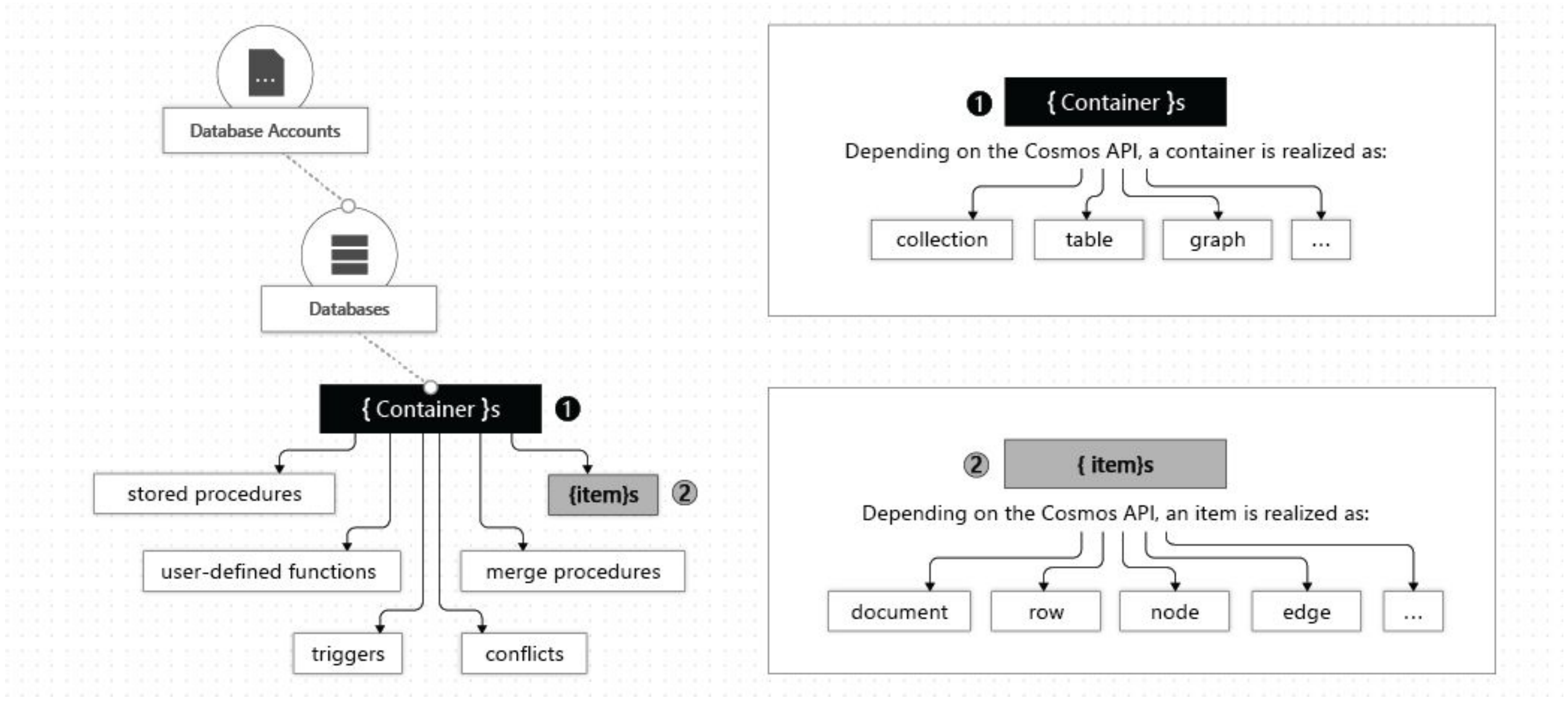
They are adapted to the fast changing agile environment which need constant feedbacks and fast iterations.

Azure Cosmos DB

Azure Cosmos DB is Microsoft's globally distributed, multi-model database service designed to provide high availability, low latency, and scalability.

It supports various NoSQL data models, including document, key-value, wide-column, and graph.

- **Key Features:**
  - Global distribution
  - Multi-model support (e.g., document, key-value, graph)
  - Automatic scaling
  - Enterprise-grade security


Database Accounts
Databases
{ Container }s
1
stored procedures
user-defined functions
triggers
conflicts
merge procedures
{item}s
2
1
{ Container }s
Depending on the Cosmos API, a container is realized as:
collection
table
graph
...
2
{ item}s
Depending on the Cosmos API, an item is realized as:
document
row
node
edge
...

APIs Supported

Core (SQL) API:

- Query data using SQL-like syntax.
- Supports JSON documents.

Cassandra API:

- Compatible with Cassandra workloads, ideal for those migrating from Cassandra to Azure.

MongoDB API:

- Supports MongoDB queries, useful for MongoDB workloads.

Gremlin API:

- For graph-based queries and data.

Table API:

- NoSQL key-value storage, compatible with Azure Table Storage.

Global Distribution (Replicas)

Global distribution is one of the standout features of Cosmos DB. It allows you to replicate your data across multiple Azure regions with just a few clicks. This ensures that your application can serve users from the nearest data center, reducing latency. Each region acts as a replica, and Cosmos DB handles the synchronization of data across these replicas. This global distribution provides high availability, disaster recovery, and geo-redundancy.

Throughput

Throughput in Cosmos DB is measured in Request Units per second (RU/s).

It defines the performance level of the database, including the number of reads, writes, and queries that can be performed per second.

You can provision throughput on a per-container (collection) or database level.

Cosmos DB automatically scales the throughput based on the demands of your application, ensuring optimal performance.

Partitioning

Partitioning in Cosmos DB is essential for handling large datasets and ensuring performance at scale.

Data is distributed across multiple partitions, each responsible for a subset of the data.

The partition key is a critical component in this process, as it determines how data is distributed across partitions.

A well-chosen partition key helps ensure even distribution of data and workload, avoiding bottlenecks and maximizing performance.

Consistency Level

Azure Cosmos DB offers five consistency levels, allowing you to balance between performance and consistency based on your application's needs:

1. **Strong**: Guarantees the most recent write is visible for all reads, ensuring no replicas will see an outdated value.
2. **Bounded Staleness**: Allows a lag in the data read, defined by a time interval or a number of operations.
3. **Session**: Guarantees consistency within a single user session, ensuring that read-after-write operations within the session are consistent.
4. **Consistent Prefix**: Guarantees that reads never see out-of-order writes. Partial updates are seen in the correct order.
5. **Eventual**: Provides the weakest consistency but offers the lowest latency, where reads may eventually reflect the most recent write.

Indexing

- **Automatic Indexing:**
    - By default, every property in a document is indexed, allowing fast queries without requiring a schema.
- **Manual Indexing:**
    - Control which properties are indexed to optimize performance and reduce storage costs.

Security Features

Data Encryption:

- Both at rest and in transit using industry-standard encryption protocols.

Role-Based Access Control (RBAC):

- Granular access control with Azure Active Directory (AAD) integration.

Network Security:

- Virtual Network Service Endpoints, IP firewall, and Private Link to restrict access to your database.

Use Cases of Azure Cosmos DB

Real-Time Personalization:

- Tailoring user experiences in real-time for retail and e-commerce.

IoT Applications:

- Ingesting and processing large amounts of data from IoT devices globally.

Gaming:

- Managing player profiles, leaderboards, and multiplayer sessions in real-time.

Social Media:

- Managing relationships, feeds, and interactions in social networks.

Azure Cosmos DB Pricing

Provisioned Throughput Mode:

- Pay based on the provisioned RU/s and storage used.

Serverless Mode:

- Pay only for the RUs consumed by database operations, ideal for small workloads.

Free Tier:

- Provides 400 RU/s and 5 GB of storage for free, ideal for development and testing.

Azure Cosmos DB Lab

https://docs.google.com/document/d/1Ast6spJHbjEvs8IFIrthXH5VY4I-ENk4d9W_EJX3Rvc/edit?usp=sharing

Relational DB

Relational Database

A relational database is a type of database that stores and provides access to data points that are related to one another. This model was first proposed by Edgar F. Codd in 1970 and has since become one of the most widely used database models in the world.

The core concept behind a relational database is the organization of data into tables (or relations), where each table consists of rows (records) and columns (attributes).

Key Concepts in Relational Databases

1. **Tables (Relations):**
   - **Definition:** A table is a collection of related data entries organized in rows and columns. Each table in a relational database represents an entity, such as "Customers" or "Orders."
   - **Columns (Attributes):** Each column in a table represents a specific attribute of the entity, such as "CustomerID," "FirstName," or "OrderDate."
   - **Rows (Records):** Each row in a table represents a single record, or a single instance of the entity. For example, a row in the "Customers" table might contain all the information about one customer.

Customer

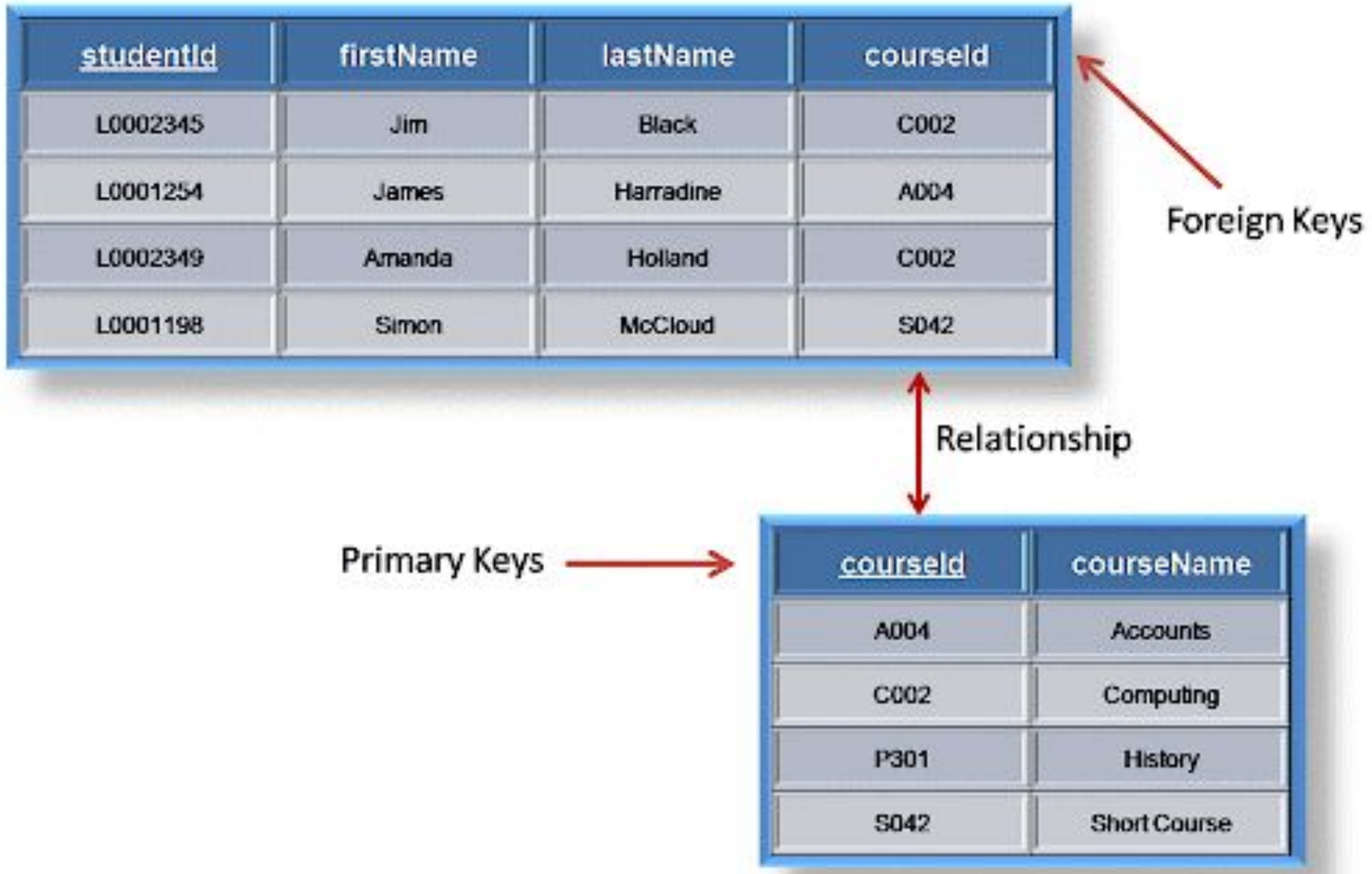
CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

2. **Primary Key:**

- **Definition:** A primary key is a unique identifier for each record in a table. It ensures that each row can be uniquely identified and prevents duplicate entries.
- **Example:** In a "Customers" table, "CustomerID" might be the primary key, as each customer should have a unique ID.

3. **Foreign Key:**

- **Definition:** A foreign key is a field (or a set of fields) in one table that uniquely identifies a row of another table. It creates a relationship between two tables, enforcing referential integrity.
- **Example:** In an "Orders" table, "CustomerID" might be a foreign key linking to the "CustomerID" primary key in the "Customers" table, indicating which customer placed the order.



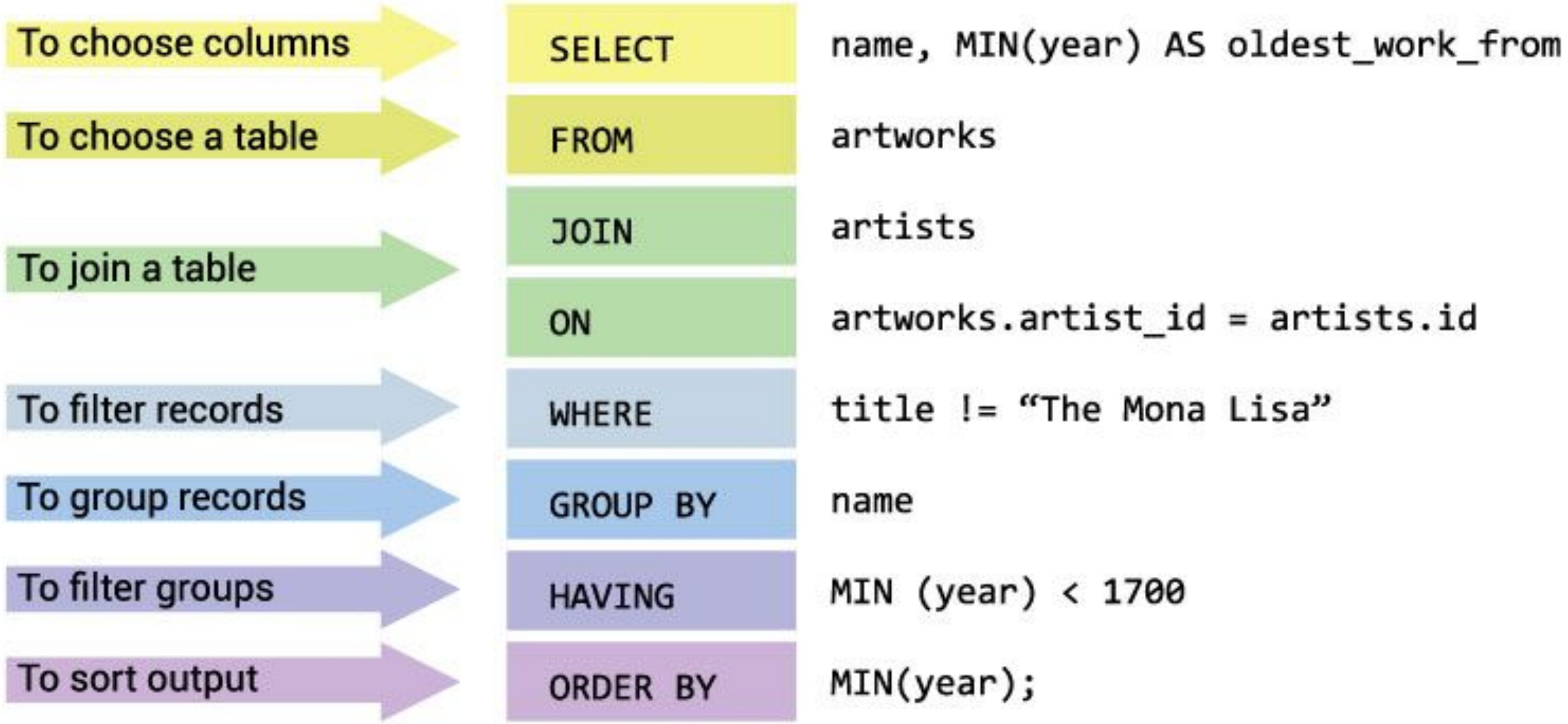
studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

SQL (Structured Query Language)

SQL is the standard language for managing and manipulating relational databases. It allows users to perform various operations on the data, such as:

- **Data Retrieval:** Using SELECT statements to query and retrieve data.
- **Data Insertion:** Using INSERT statements to add new records to a table.
- **Data Updating:** Using UPDATE statements to modify existing records.
- **Data Deletion:** Using DELETE statements to remove records from a table.
- **Database Management:** Using CREATE, ALTER, and DROP statements to manage the structure of the database.



```
SELECT     name, MIN(year) AS oldest_work_from
FROM       artworks
JOIN       artists
ON         artworks.artist_id = artists.id
WHERE      title != “The Mona Lisa”
GROUP BY   name
HAVING     MIN (year) < 1700
ORDER BY   MIN(year);
```

SQL Syntax Rules

- SQL statements are case-insensitive, but table and column names may be case-sensitive depending on the database.
- SQL statements end with a semicolon (;).
- Clauses are used to specify conditions, such as WHERE and HAVING.
- Parentheses are used to group expressions and clarify order of operations.
- Commas separate items in a list, such as column names or values.

Basic SQL Syntax

- **SELECT**: Retrieves data from a database table.
- **FROM**: Specifies the table(s) to retrieve data from.
- **WHERE**: Filters data based on conditions.
- **GROUP BY**: Groups data by one or more columns.
- **HAVING**: Filters grouped data based on conditions.
- **ORDER BY**: Sorts data in ascending or descending order.
- **INSERT INTO**: Adds new data to a table.
- **UPDATE**: Modifies existing data in a table.
- **DELETE**: Deletes data from a table.

- **CREATE TABLE**: Creates a new table in the database.
- **ALTER TABLE**: Modifies an existing table (e.g., adding or dropping columns).
- **DROP TABLE**: Deletes a table from the database.
- **JOIN**: Combines rows from two or more tables based on a related column between them.
- **INNER JOIN**: Returns rows when there is at least one match in both tables.
- **LEFT JOIN**: Returns all rows from the left table and matching rows from the right table.
- **RIGHT JOIN**: Returns all rows from the right table and matching rows from the left table.
- **FULL OUTER JOIN**: Returns all rows when there is a match in either table.
- **DISTINCT**: Retrieves unique values in a column.

- **AS**: Renames a column or table with an alias.
- **LIKE**: Searches for a specified pattern in a column using wildcard characters.
- **IN**: Specifies multiple values for a column.
- **BETWEEN**: Selects values within a given range.
- **NULL**: Represents NULL values in the database.
- **IS NULL**: Checks if a value is NULL.
- **NOT NULL**: Checks if a value is not NULL.

Resources

- Watch and Install SQL Server and SSMS - https://www.youtube.com/watch?v=7zXtA0LwoHs
- Download SQL Server - https://www.microsoft.com/en-us/sql-server/sql-server-downloads
- Download SSMS - https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16#download-ssms
- Download Azure Data Studio - https://learn.microsoft.com/en-us/azure-data-studio/download-azure-data-studio?tabs=win-install%2Cwin-user-install%2Credhat-install%2Cwindows-uninstall%2Credhat-uninstall#download-azure-data-studio
- Download AdventureWorksDatabase - https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms
- SQL compiler - https://programiz.com/sql/online-compiler
- Learn SQL - https://www.codecademy.com/learn/learn-sql

Hands-on Lab

SQL Lab - https://docs.google.com/document/d/1dPD4bBSAi_VGFZsJf6iuIVZZ3t3MSZ8ryNpgSYe_XXQ/edit?usp=sharing

Assignment

- Practice SQL - https://www.codecademy.com/learn/learn-sql