Software Engineering

A Faculty of Engineering Course: CSEN 303

Databases

6

Dr. Iman Awaad

iman.awaad@giu-uni.de

GERMAN INTERNATIONAL UNIVERSITY
الجامعة الألمانية الدولية

Acknowledgments

The slides are **heavily** based on the **slides** by **Prof. Dr. John Zaki**.

His contribution is gratefully acknowledged.

Any additional sources are referenced.

Databases

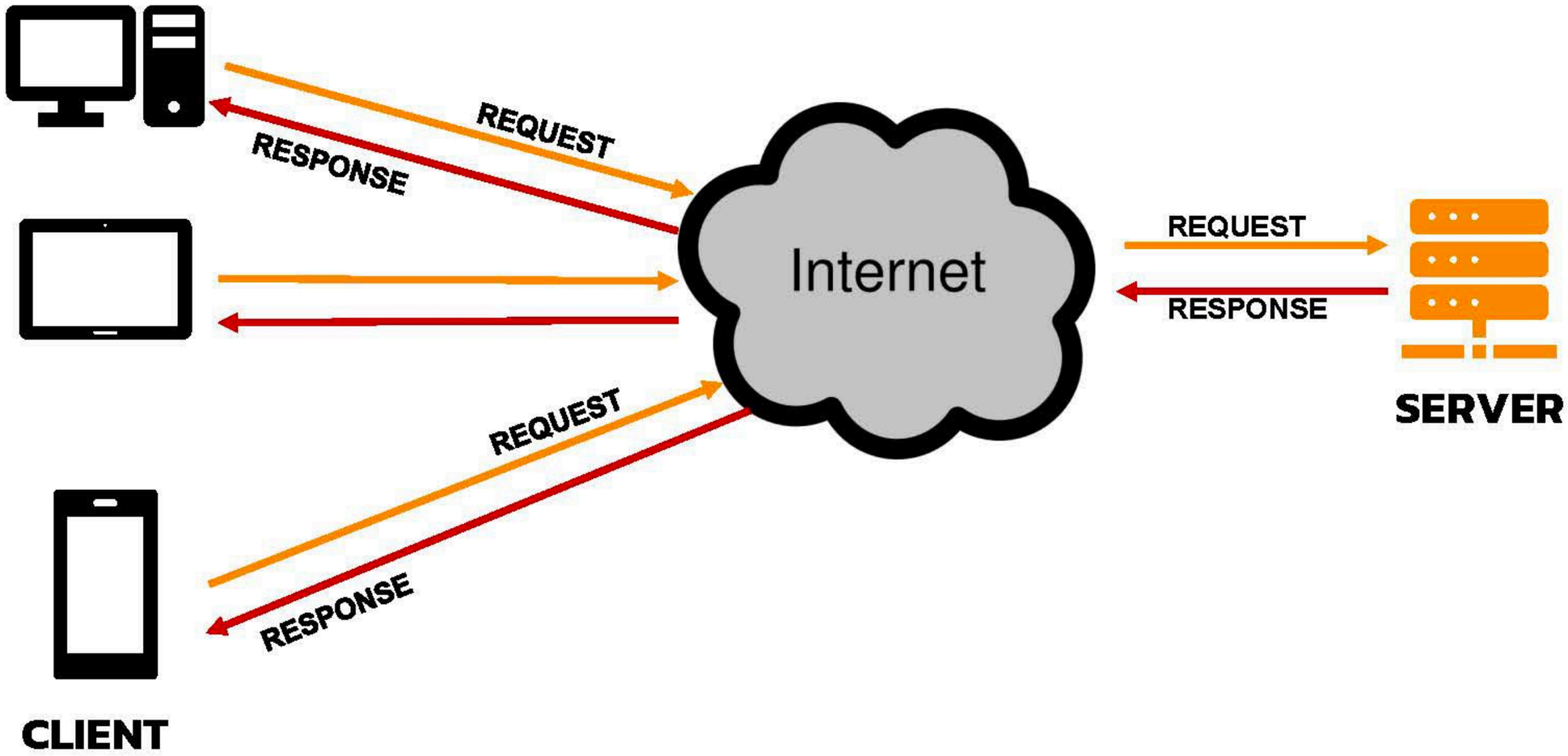
- Databases
- DBMS
- Relational v Non-relational DBs
- Key concepts in Relational DBs
- Visualising Relational DBs with Enity-Relationship Diagrams
- SQL Basics

How do we
organise and store data?

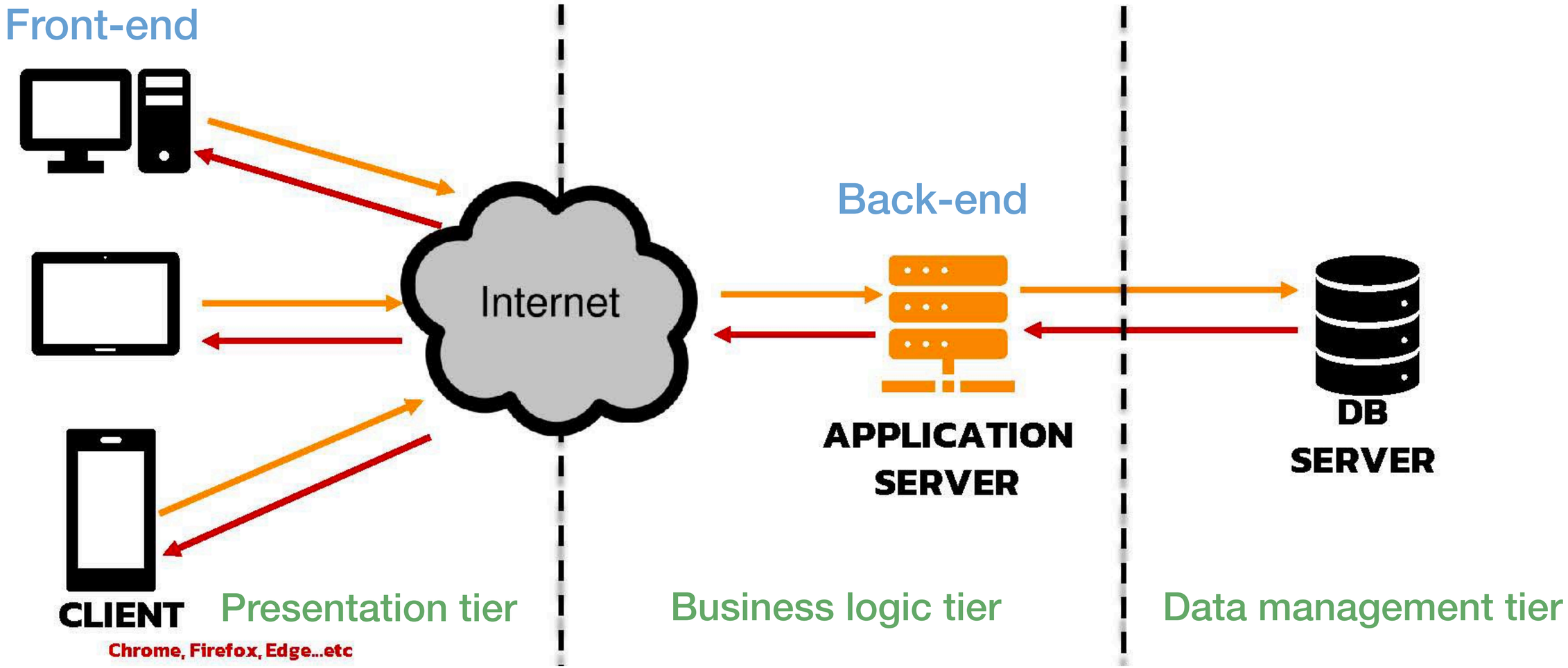
How do we manipulate it
(*e.g.* insert, delete, search,
analyse and update)?

The Internet in a nutshell…

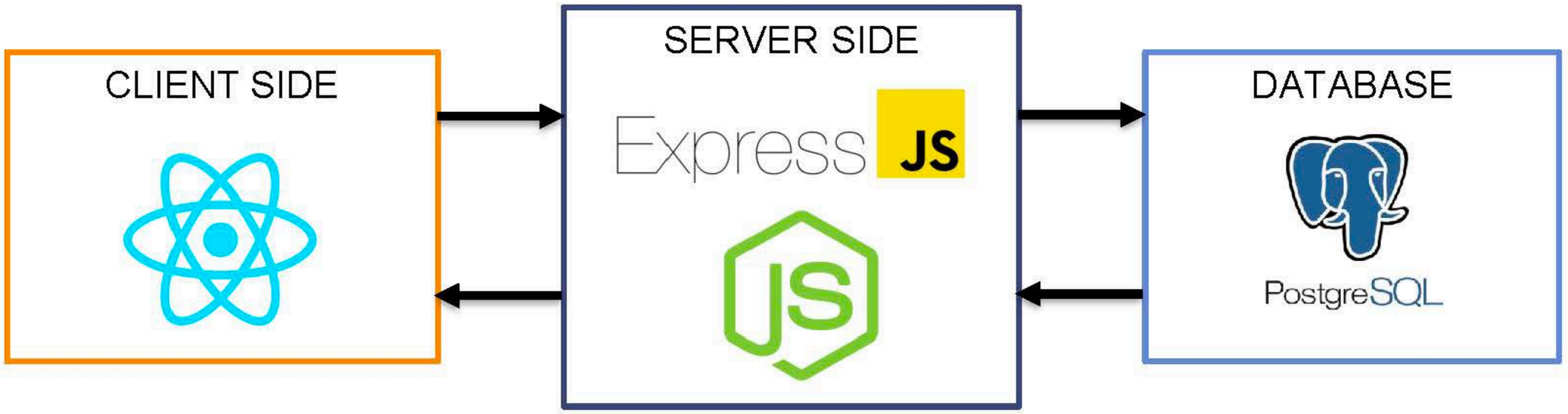
Remember this?



Web application



The PERN stack



What is a **Database**?

… is an **organized**, **electronic** collection of **structured information** (**data**) that is **managed** and **controlled** by a computer system, typically a **database management system** (**DBMS**). Databases store data in **formats** like **rows** & **columns** within **tables** to make it **efficient** to **access**, **manage**, and **update**.

Database management system (DBMS)

Software that allows you to
create a database and all the objects within it
(*e.g.* tables, views,…)

…**retrieve**, **modify** and **secure** data

Database management system (DBMS)

Software that allows you to
create a database and all the objects within it (*e.g.* tables, views,…)

Retrieve, **modify** and **secure** data

Relational DB

that use structured data

e.g. MySQL, PostgreSQL, Oracle…

Non-relational DB

that use unstructured data

e.g. MongoDB, Cassandra,
Redis, Apache Hbase

How do we **create**/**read**/**update**/**delete** (**CRUD**) data in the DB?

…use

SQL (**Structured Query Language**)

…it depends…

aka **NoSQL** DBs

Types of DBs

Relational DBs

store data in tables with rows and columns…

query using

SQL (**Structured Query Language**)

Non-relational/NoSQL DBs

NoSQL DB

Data structures can vary, making them ideal for rapidly evolving data; great at horizontal scaling (distributing data across multiple servers (nodes)) to handle massive datasets and high traffic loads

Object-oriented DB

data stored in object-oriented format: organized into objects, which can be related to one another. Used for complex data structures, such as 3D models or geographical information

Graph DB

for data best represented as a network of interconnected nodes. Used to store data about relationships between people or things. *e.g.* social networks or relationships between different system parts

Example DBs

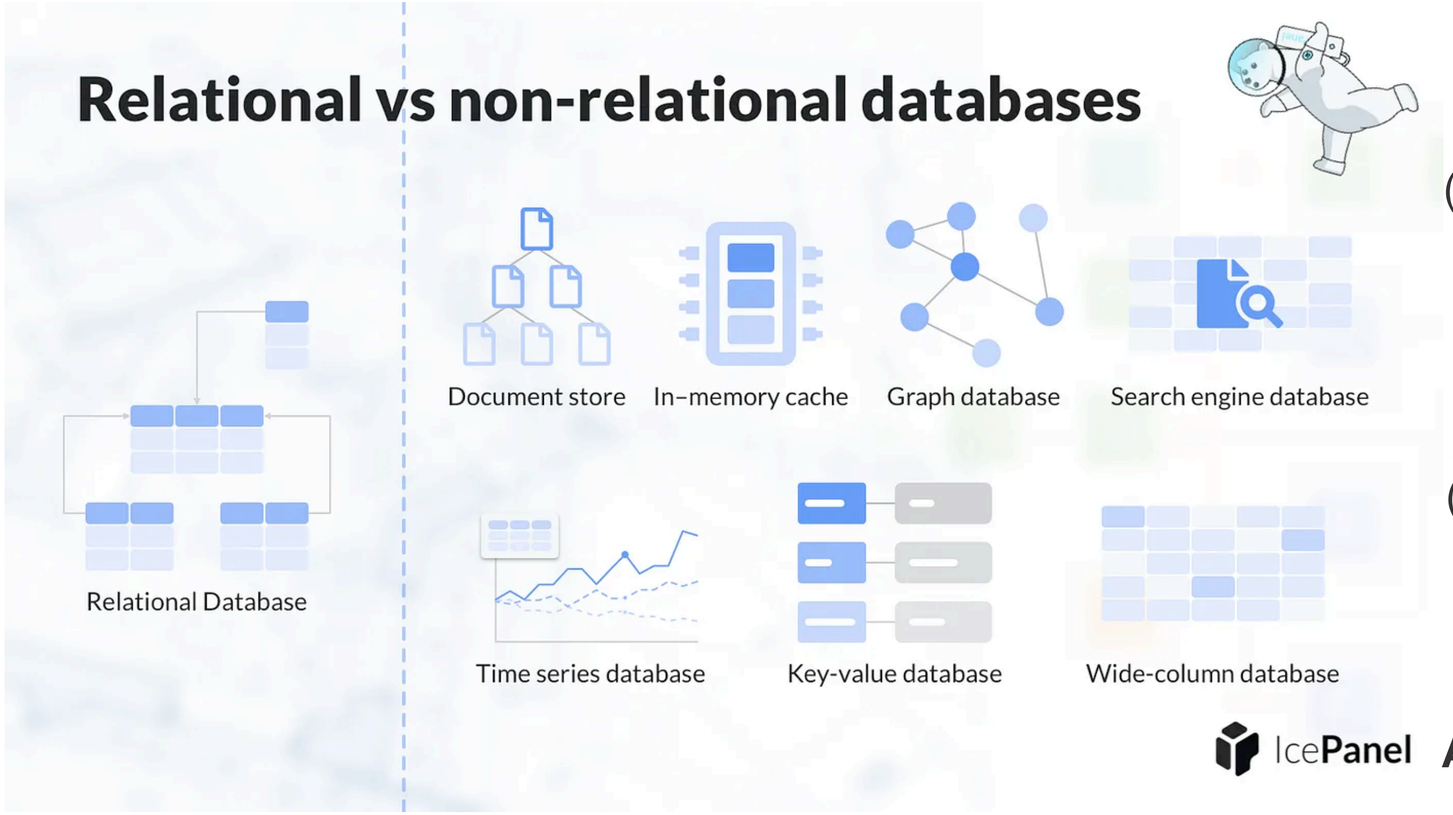
MySQL

PostgreSQL

Amazon RDS

Google Cloud SQL

Azure SQL



MongoDB
(document database)

Redis
(key-value store)

Apache Cassandra
(column-family store)

Neo4j
(graph database)

Amazon DynamoDB
(key-value and document database)

https://icepanel.medium.com/relational-sql-vs-non-relational-nosql-databases-a66ac93f9aa0

Example DBs

JSON — Javascript Object Notation

My friends

text | tree | table

```
{
  "name": "Richard",
  "age": "33",
  "hobbies": ["Biking", "Gaming", "Squash"],
  "city": "Port Land",
  "friends": [
    {"name": "Chris"   , "age": 23, "city": "New York"   },
    {"name": "Emily"   , "age": 19, "city": "Atlanta"    },
    {"name": "Joe"     , "age": 32, "city": "New York"   },
    {"name": "Kevin"   , "age": 19, "city": "Atlanta"    },
    {"name": "Michelle", "age": 27, "city": "Los Angeles"},
    {"name": "Robert"  , "age": 45, "city": "Manhattan"  },
    {"name": "Sarah"   , "age": 31, "city": "New York"   }
  ]
}
```

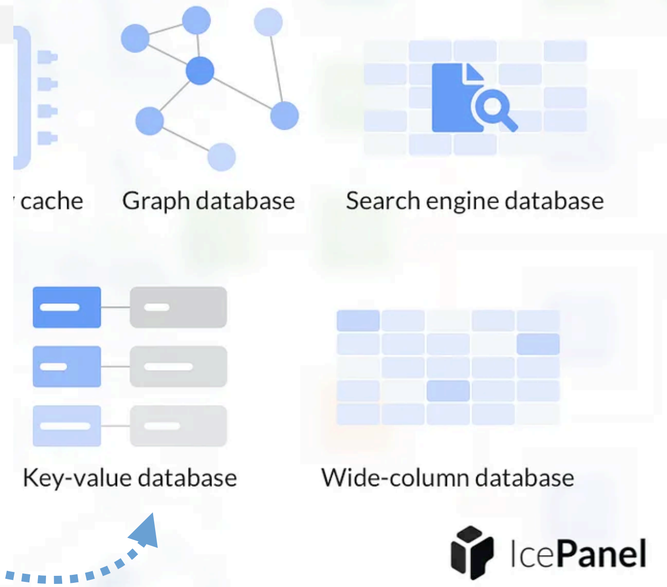
MongoDB
(document database)

Redis
(key-value store)

Apache Cassandra
(column-family store)

Neo4j
(graph database)

Amazon DynamoDB
(key-value and document database)



https://icepanel.medium.com/relational-sql-vs-non-relational-nosql-databases-a66ac93f9aa0

Example DBs

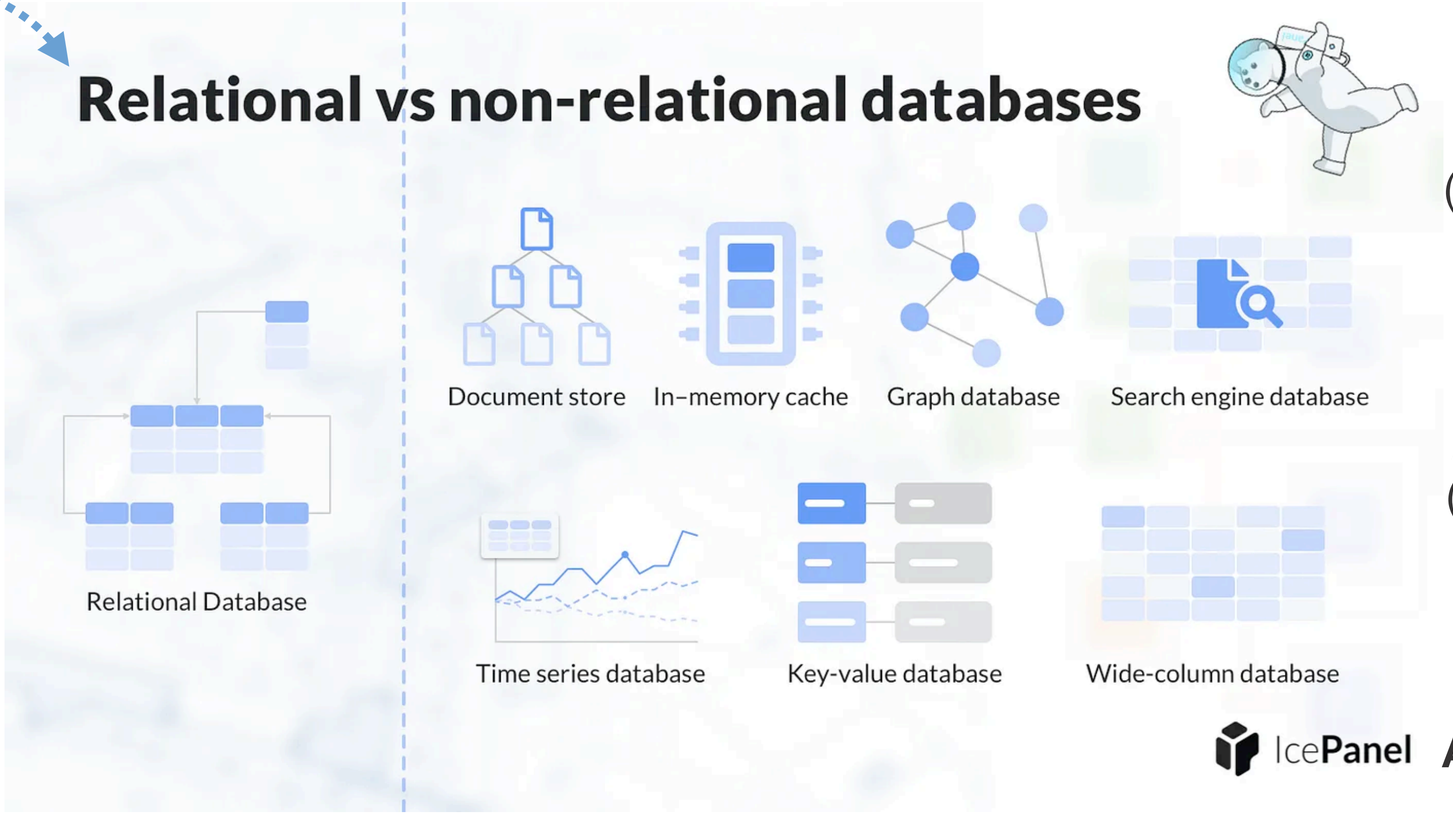
MySQL

PostgreSQL

Amazon RDS

Google Cloud SQL

Azure SQL



MongoDB
(document database)

Redis
(key-value store)

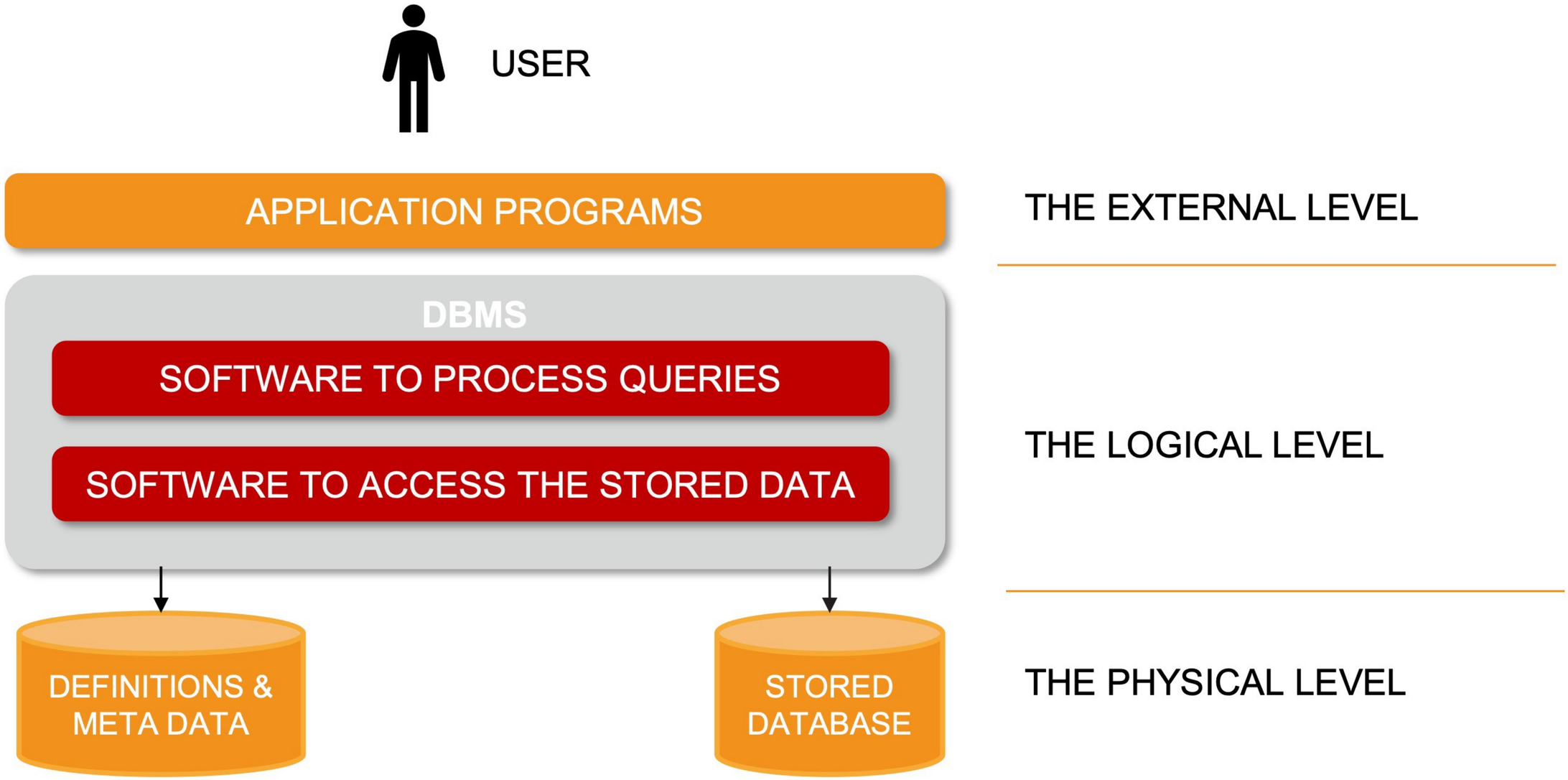
Apache Cassandra
(column-family store)

Neo4j
(graph database)

Amazon DynamoDB
(key-value and document database)

https://icepanel.medium.com/relational-sql-vs-non-relational-nosql-databases-a66ac93f9aa0

DB system environment



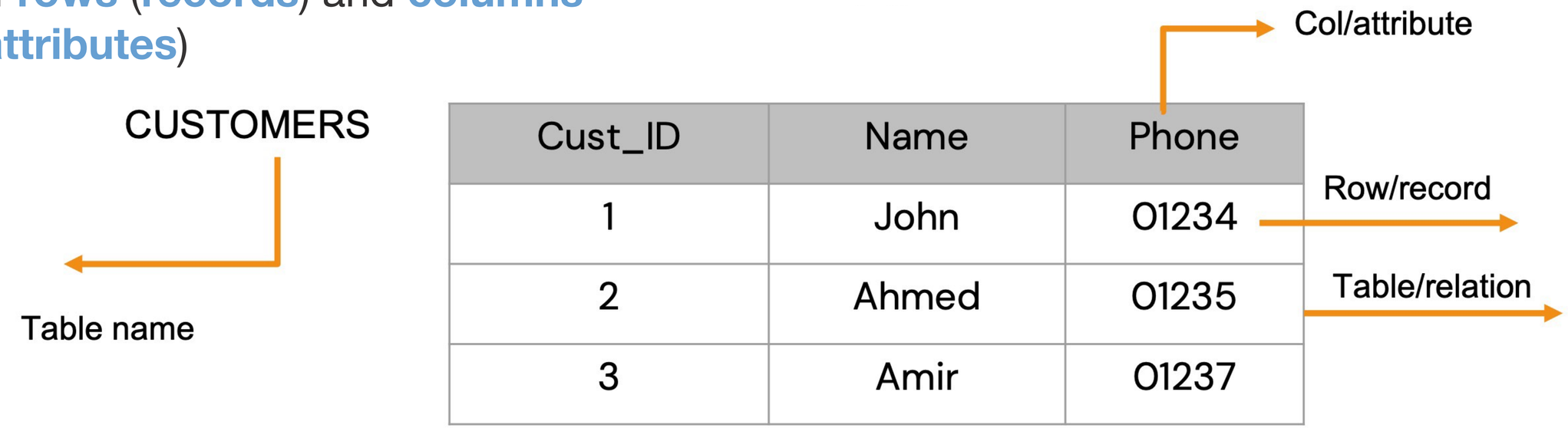
Relational model

…**Tables** are the fundamental structure of relational DBs; **Data** is **structured** using **relations**

Tables **represent** a specific **entity type** (*e.g.* "Customers," "Products"). Consists of **rows** (**records**) and **columns** (**attributes**)

Each **relation** (**table**) has a **name**, **attributes** and **records**, where each record holds one data per attribute

DB schema: the logical representation of the DB



Cust_ID	Name	Phone
1	John	01234
2	Ahmed	01235
3	Amir	01237

Relational model

Row (**Tuple**/**Record**): A single entry or **instance** within a table

Represents a **complete set of related data** for a specific entity (*e.g.* a single customer's details)

Column (**Attribute**/**Field**): A vertical component of a table

Represents **a specific characteristic** or **property** of the entity type (*e.g.* "CustomerName," "ProductID").

Domain: Set of **all possible values** that an **attribute** can take (*e.g.* the domain for "Age" might be positive integers).

Key: An **attribute** or **set of attributes** used to **uniquely identify** or **establish relationships** between **records within a table** or **across tables**

	Attribute	Attribute	Attribute	attribute
Entity				
Entity				
Entity				

Relational model

Primary Key: **unique** **number** identifying the records. Provided automatically by the DBMS… cannot be NULL

Foreign key: a **link** **between two tables**. Enforces **data integrity**.

A foreign key in one table that references a primary key in another table and is used to set relationships.

Structured Query Language

Enables the use of an SQL **Join** query to pull data from two related tables (thus, creating a **view**)

Views: A virtual table based on the result-set of a SQL query…used to **securely pull data from tables** (*i.e.* without actually giving access to tables)… It does not store data itself but provides a dynamic window into the underlying data.

Index: A data structure that improves the speed of data retrieval operations on a database table.

Stored **Procedures**: similar to views, container for SQL code. They are stored and you can pass parameters to them so they can retrieve the data.

Relational model

Relationship: A logical connection between two or more tables, established through common attributes (often foreign keys referencing primary keys).

Cardinality: This specifies the number of instances of one entity that can be associated with the number of instances of another entity…

One-to-One (1:1)

Each record in one table relates to exactly one record in another table

One-to-Many (1:N)

Each record in one table can relate to multiple records in another table

Many-to-Many (N:M)

Multiple records in one table can relate to multiple records in another table (usually implemented using a **linking** or **junction table**)

Relational model

Relationship: A logical connection between two or more tables, established through common attributes (often foreign keys referencing primary keys).

Cardinality: This specifies the number of instances of one entity that can be associated with the number of instances of another entity…

One-to-One (1:1)

Each record in one table relates to exactly one record in another table

One-to-Many (1:N)

Each record in one table can relate to multiple records in another table

Many-to-Many (N:M)

Multiple records in one table can relate to multiple records in another table (usually implemented using a **linking** or **junction table**)

If we want to visualise a relational DB
…all these should be represented…

What is an **Entity Relation** Diagram (**ERD**)?

…a **visual tool** used in **DB design** to show the **logical structure of data** by depicting **how different entities** (e.g. customers, products, or orders) **are related** to each other…

What is an **Entity Relation** Diagram (**ERD**)?

… is a **conceptual data model** that visually maps out the **structure** of a DB by illustrating the **entities** (tables), their **attributes** (columns), and the **relationships** (connections) between them. ER diagrams serve as a **high-level blueprint** for DB **design**, helping to **organize** data **early in the project** and **communicate** complex data structures between **stakeholders** and **developers**.

https://www.atlassian.com/work-management/project-management/entity-relationship-diagram#:~:text=Understanding%20ER%20diagrams,data%20model%20during%20database%20development.

Entity Relationship Diagram (ERD)

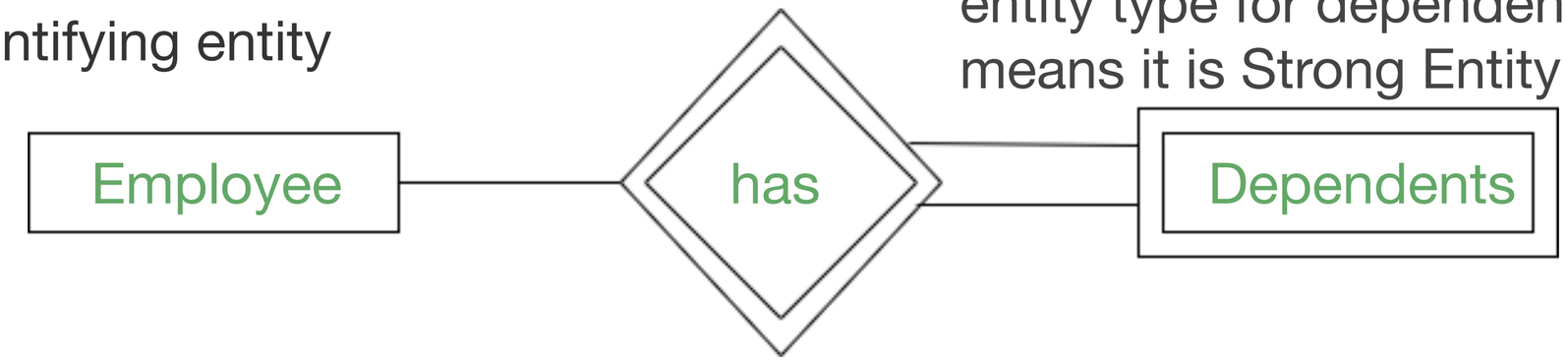
Entities: represent real-world objects or concepts; typically depicted as rectangles
e.g. "Customer," "Product," or “Order"

Strong: has a key Attribute that can uniquely identify each instance

Weak: cannot be uniquely identified by its own attributes alone… associated with an identifying entity

e.g. A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents can't exist without the employee.

So dependent will be a Weak Entity Type and Employee will be identifying entity type for dependent, which means it is Strong Entity Type.



https://www.geeksforgeeks.org/dbms/introduction-of-er-model/

https://www.atlassian.com/work-management/project-management/entity-relationship-diagram#:~:text=An%20entity%20relationship%20diagram%20is,and%20how%20to%20create%20them.

Entity Relationship Diagram (ERD)

Entities: represent real-world objects or concepts; typically depicted as rectangles *e.g.* "Customer," "Product," or “Order"

Strong: has a key Attribute that can uniquely identify each instance

Weak: cannot be uniquely identified by its own attributes alone… associated with an identifying entity

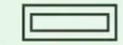
Attributes: facts or characteristics that describe an entity, often becoming the columns in a database table; usually listed inside the entity's rectangle.

Relationships: These are the connections or associations between entities, represented by verbs and depicted by lines connecting the entities. *e.g.* "places" (a customer places an order) or "belongs to" (a student belongs to a course).

Cardinality: This specifies the number of instances of one entity that can be associated with the number of instances of another entity.

https://www.atlassian.com/work-management/project-management/entity-relationship-diagram#:~:text=An%20entity%20relationship%20diagram%20is,and%20how%20to%20create%20them.

ERD Model: Attributes

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

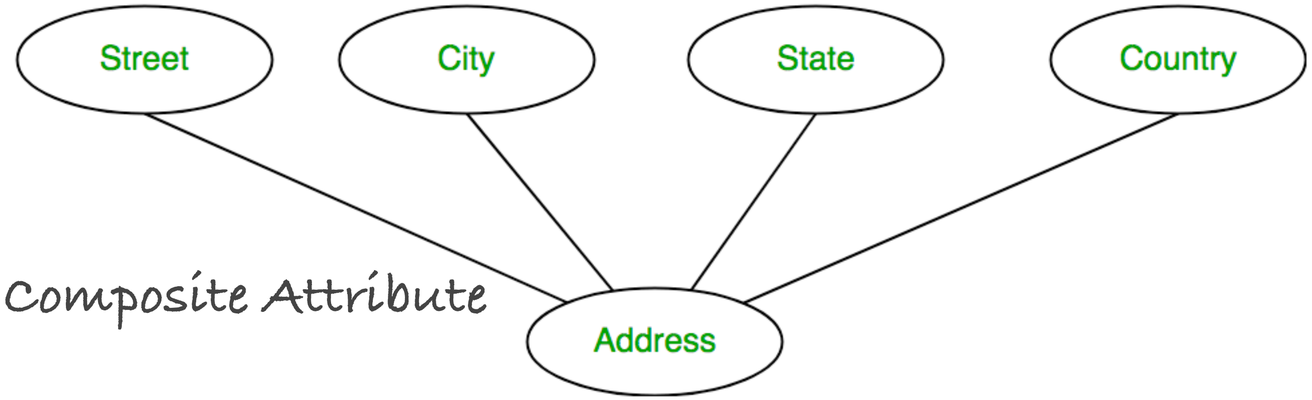
Key Attribute

Multivalued Attribute

(can be more than one for a given student)

Derived Attribute

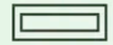
(can be derived from DOB)



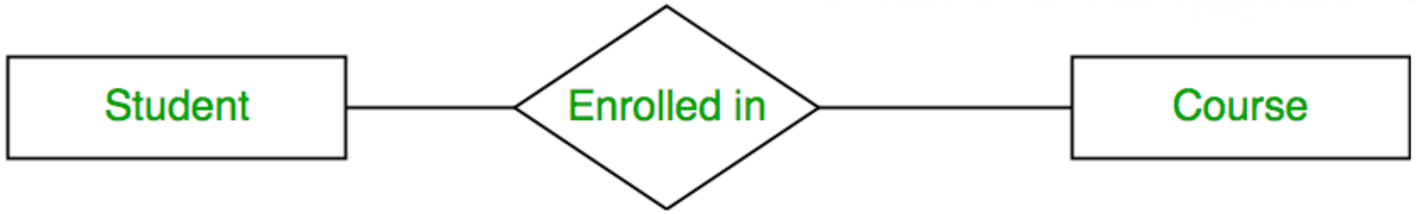
Composite Attribute

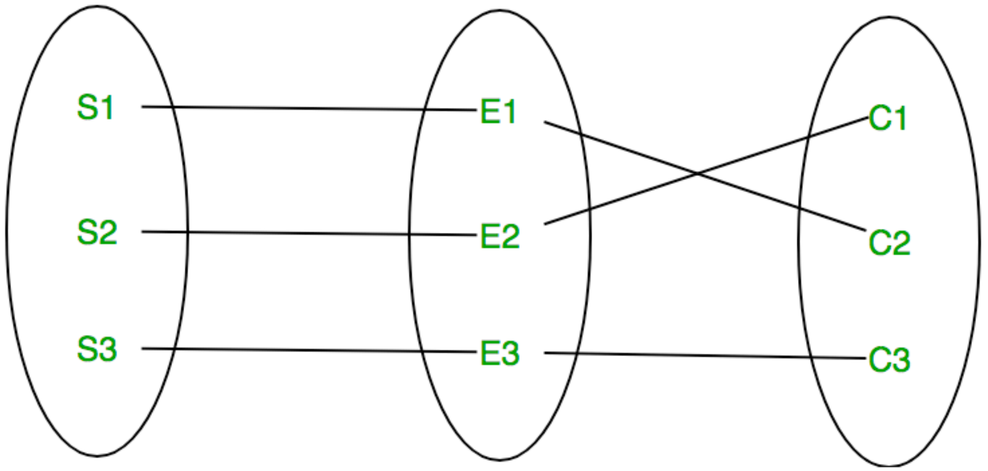
https://www.geeksforgeeks.org/dbms/introduction-of-er-model/

ERD Model: Relationships

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

Relationship





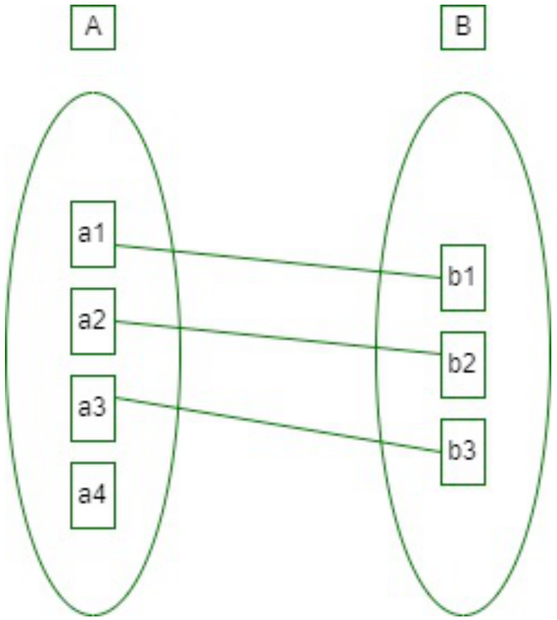
Relationship Set

(Set of relationships of the same type:
S1 as enrolled in C2,
S2 as enrolled in C1,
and S3 as registered in C3)

https://www.geeksforgeeks.org/dbms/introduction-of-er-model/

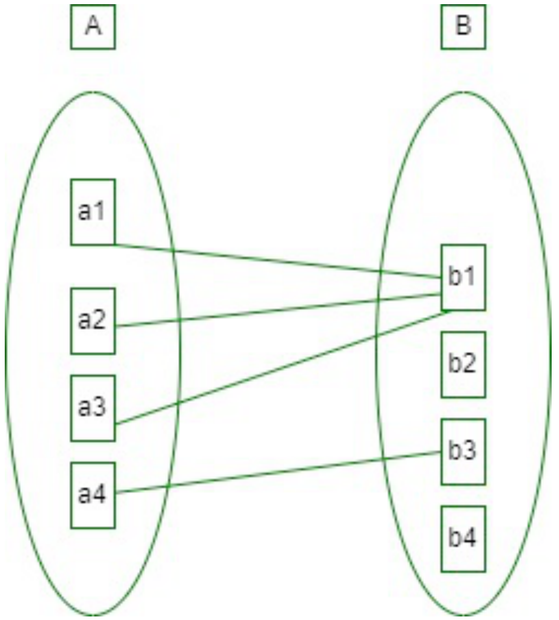
ERD Model: Cardinality

1:1



Many-to-1

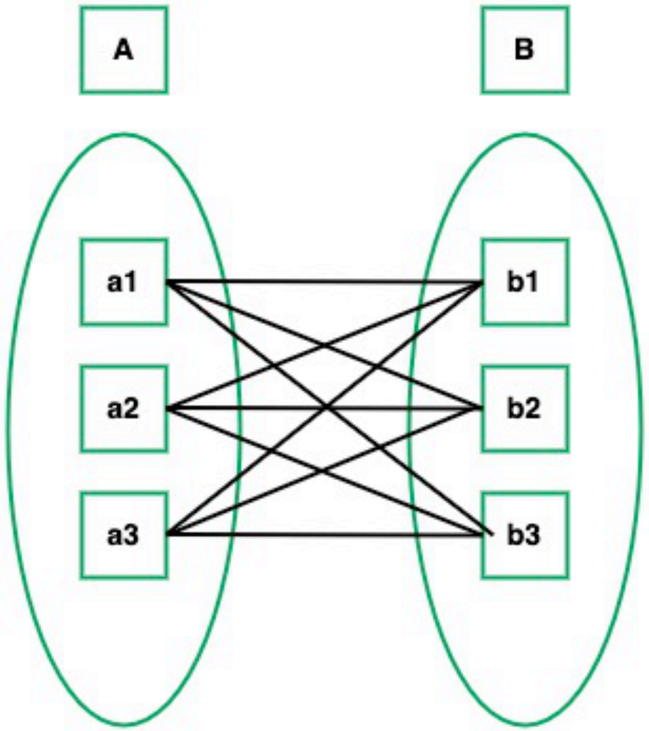
(entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set)



https://www.geeksforgeeks.org/dbms/introduction-of-er-model/

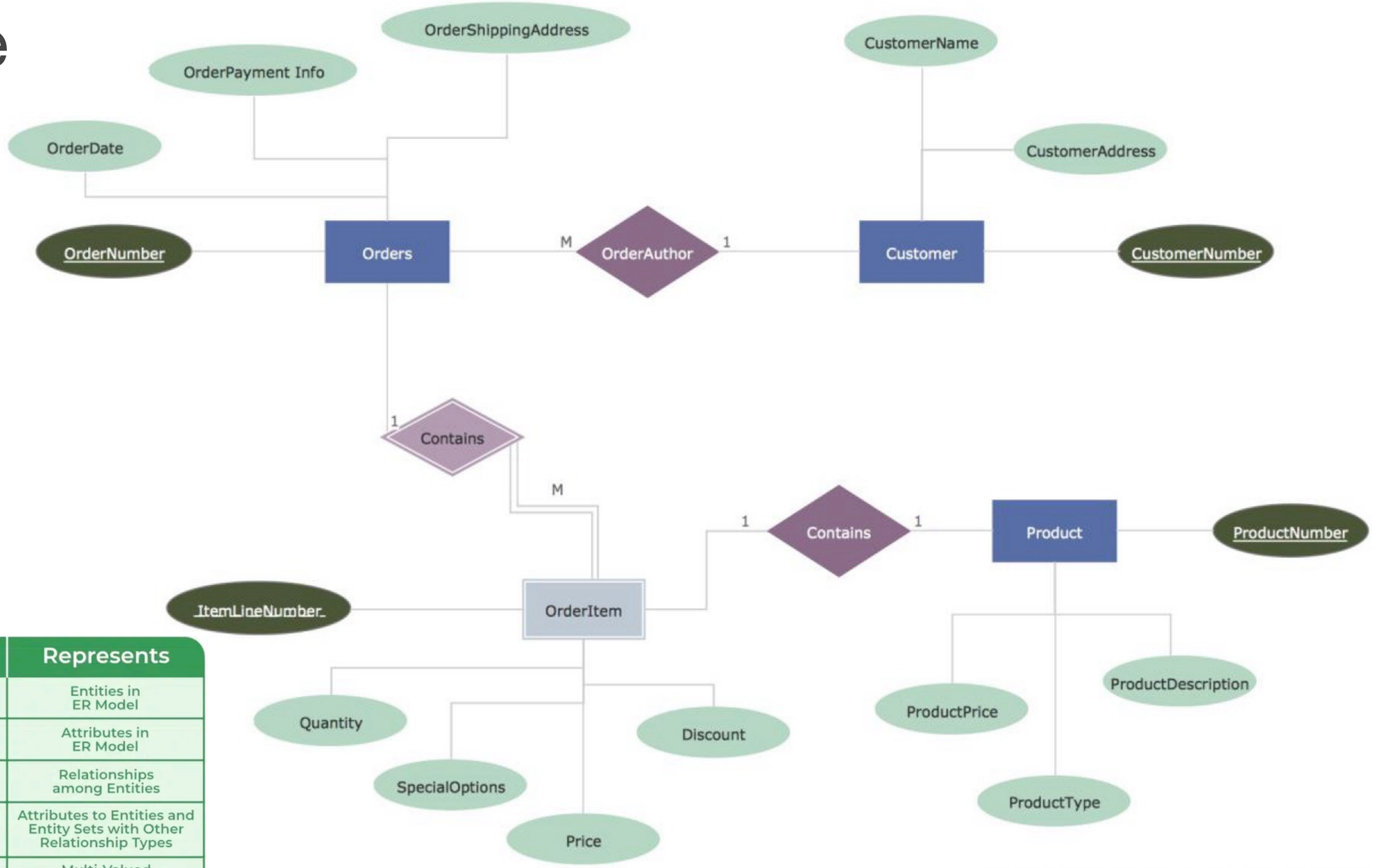
ERD Model: Cardinality

Many-to-Many
(entities in all entity sets can take part more than once in the relationship)



https://www.geeksforgeeks.org/dbms/introduction-of-er-model/

Example ERD

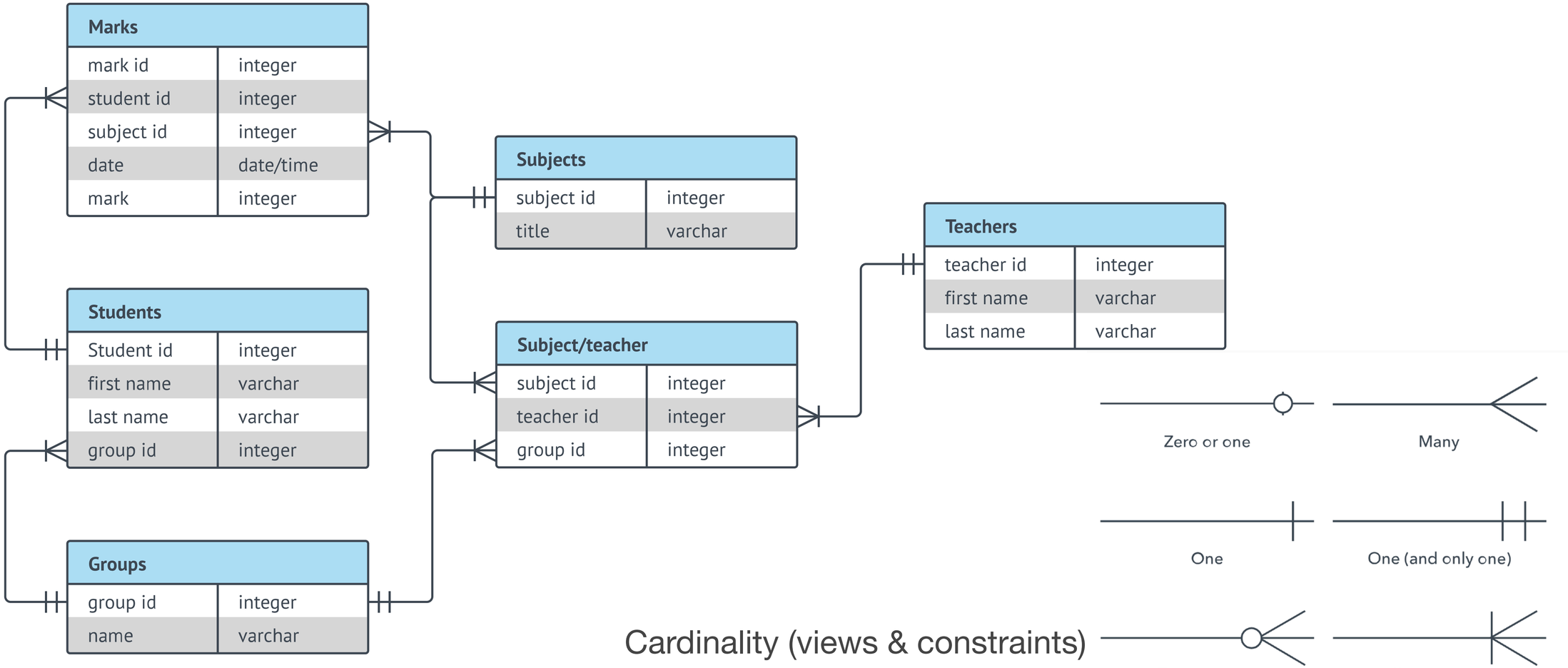


Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

PICTURE COURTSEY OF ERMODELEXAMPLE.COM

ER Models focus on the **relationships of elements within entities**,

instead of relationships between entities themselves

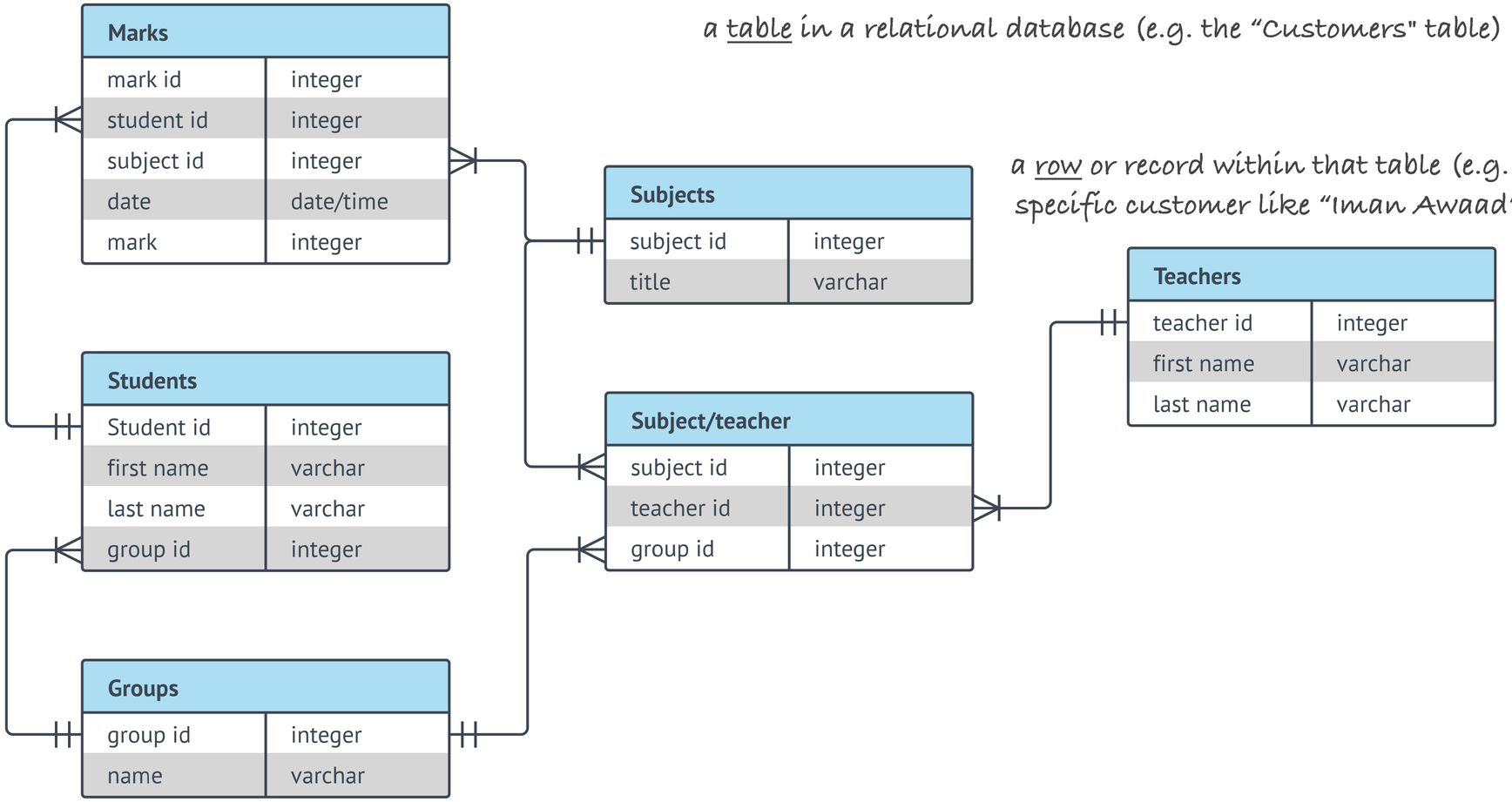


Cardinality (views & constraints)

https://www.lucidchart.com/pages/er-diagrams

ER Models focus on the **relationships of elements within entities**,

instead of relationships between entities themselves



a table in a relational database (e.g. the "Customers" table)

a row or record within that table (e.g. a specific customer like "Iman Awaad")

Common noun: Entity type. *e.g.* student

Proper noun: Entity. *e.g.* Sally Smith

Verb: Relationship type. *e.g.* Enrols. (*e.g.* in a course, which would be another entity type)

Adjective: Attribute for entity. *e.g.* sophomore

Adverb: Attribute for relationship. *e.g.* digitally

https://www.lucidchart.com/pages/er-diagrams

ERD Model

Entity
Field
Field
Field

Entity	
Key	Field
Key	Field
Key	Field

Entity		
Key	Field	Type
Key	Field	Type
Key	Field	Type

Customers
CustomerID
FirstName
LastName
Street
City
ZipCode
Phone

Customers	
Key	CustomerID
Key	FirstName
Key	LastName
Key	Street
Key	City
Key	ZipCode
Key	Phone

Customers		
Key	CustomerID	Type
Key	FirstName	Type
Key	LastName	Type
Key	Street	Type
Key	City	Type
Key	ZipCode	Type
Key	Phone	Type

ERD Model

Customers		
PK	CustomerID	int
	FirstName	varchar(50)
	LastName	varchar(50)
	Street	varchar(50)
	City	varchar(50)
	ZipCode	varchar(50)
	Phone	varchar(11)

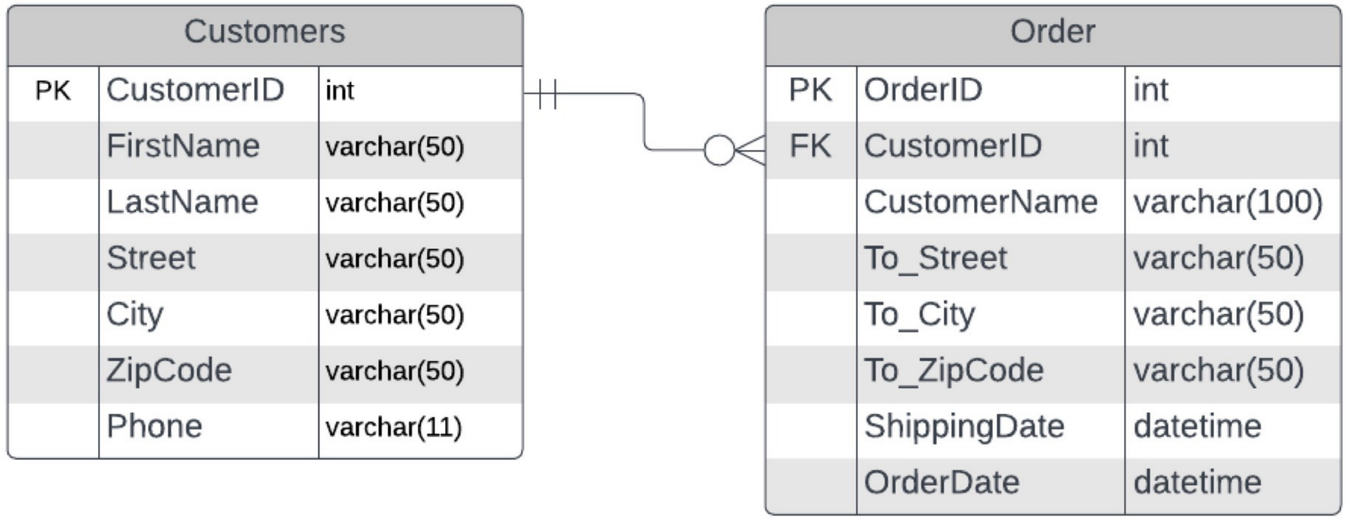
1. **Primary Key:** is an attribute or field that uniquely identifies a single record in a table
   - Unique
   - Never changing
   - Never null

2. **Data Types:**
   a. Numeric data types
   b. String data types
   c. Date & Time data types
   d. XML data type
   e. Spatial data type

ERD Model

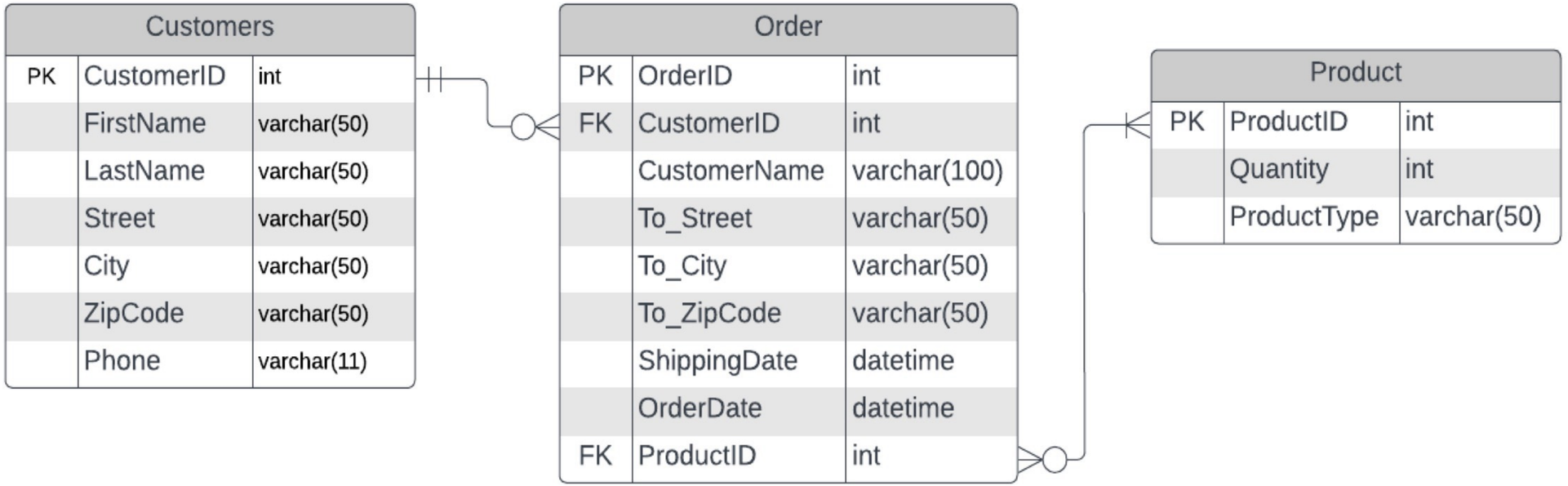
1. **Foreign Key:** it is the primary key of a table but exists in a foreign table (another table). Can be repeated or duplicated.

2. **Composite Primary Key:** two or more foreign keys are used as a Primary key.



ERD Model

1. **Cardinality (multiplicity):** the relationship between two entities, specifically the number of instances of one entity that can be associated with a single instance of another entity



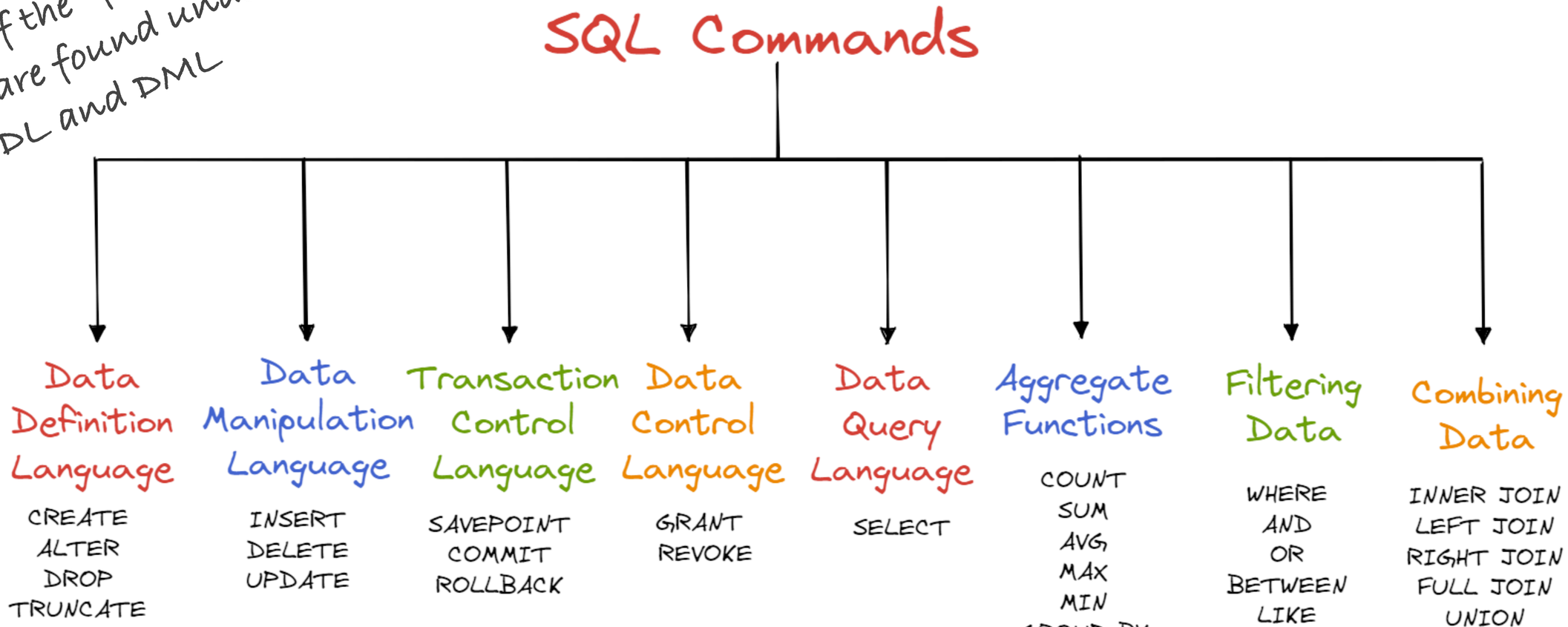
Structured **Query** Language (SQL)

sometimes, pronounced "sequel"

…is the standard language used to **interact with** and **manage** relational DBs

(e.g. **defining**, **manipulating**, and **querying** data)

Most of the "famous ones", are found under DDL and DML



https://storage.googleapis.com/algodailyrandomassets/curriculum/sql_tutorials/sql_commands.png

DDL & DML

DDL

Used to create the database structure are known as data definition language (DDL).

CREATE: creates a new table, view, index or other object in the DB

ALTER: modifies an existing DB object such as a table or a view…

DROP: deletes a table or a view or other objects.

DML

Update the DB with new data

INSERT: creates a record

UPDATE: modifies existing records

DELETE: deletes existing records

SELECT: retrieves certain records from one or more tables. Considered as DQL

DCL & DQL

DCL

Used to control or grant other users access to DB

GRANT: give authorization or privilege to a user

REVOKE: removes an authorization or privilege

DQL

Used to fetch data from the DB

SELECT: define what data you want to fetch

COMBINING DATA

Used to fetch data from the DB

JOIN: used to combine rows from two or more tables based on a related column between them

Data types

Numeric

bigint, int, smallint, tinyint, bit, numeric, money, smallmoney, float, real,

`INT`: *e.g.* `123, -45`

`SMALLINT`: *e.g.* `50, -10`

`BIGINT`: *e.g.* `9223372036854775807`

`TINYINT`: *e.g.* `5, 120`

`DECIMAL(p, s)` or `NUMERIC(p, s)`: *e.g.* `DECIMAL(5, 2)` for `123.45`

String

(non-Unicode) char, varchar, varchar(max), text

(Unicode) nchar, Nvarchar,nvarchar(max)

`VARCHAR(n), n is max length, e.g. VARCHAR(255) for "Hello World"`

`CHAR(n),e.g. CHAR(10) for "Name "`

Date and time

Date, time, datetime

`DATE`: *e.g.* `2025-10-04`

`TIME`: *e.g.* `05:10:00`

`DATETIME` or `TIMESTAMP`: *e.g.* `2025-10-04 05:10:00`

Data types

Binary Data Types

Used for storing binary data *e.g.* images or files

`BINARY(n)`: Fixed-length binary data

`VARBINARY(n)`: Variable-length binary data

`BLOB` (Binary Large Object): Stores large binary data

Boolean

Used for storing true/false values

`BOOLEAN`: Stores `TRUE` or `FALSE`. Some systems might use `BIT(1)` to represent this.

e.g.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE,
    salary DECIMAL(10, 2),
    is_active BOOLEAN
);
```

CREATE TABLE

```
CREATE TABLE table_name (
Column1 datatype (size),
Column2 datatype (size),
.....);
```

CREATE: creates a new table, view, index or other object in the DB

Customers		
PK	CustomerID	int
	FirstName	varchar(50)
	LastName	varchar(50)
	Street	varchar(50)
	City	varchar(50)
	ZipCode	varchar(50)
	Phone	varchar(11)

```
CREATE TABLE "Customers" (
  "CustomerID" INT,
  "FirstName" varchar(50),
  "LastName" varchar(50),
  "Street" varchar(50),
  "City" varchar(50),
  "ZipCode" varchar(50),
  "Phone" varchar(11),
PRIMARY KEY ("CustomerID")
);
```

PRIMARY KEY

1. UNIQUE
2. AUTO INCREMENT
3. NOT NULL

CREATE TABLE

```
CREATE TABLE Order (

“OrderID” INT,
PRIMARY KEY(“OrderID”),

FOREIGN KEY(“CustomerID”),
REFERENCES Customers(“CustomerID”),

“CustomerName” varchar(100),
“To_Street” varchar(50),
“To_City” varchar(50),
“To_ZipCode” varchar(50),
“ShippingDate” datetime,
“OrderDate” datetime,
“ProductID” INT,
);
```

CREATE: creates a new table, view, index or other object in the DB

CREATE TABLE

CREATING FROM EXISTING TABLE AND COPYING DATA
CREATE TABLE *new_table_name* AS
SELECT * FROM *old_table_name;*

CREATING FROM EXISTING TABLE WITHOUT COPYING DATA
CREATE TABLE *new_table_name* (LIKE *old_table_name*);

EX.
CREATE TABLE Administrators (LIKE Customers);

CREATE: creates a new table, view, index or other object in the DB

ALTER TABLE

ALTER: modifies an existing DB object such as a table or a view…

ALTER TABLE table_name ACTION;	RENAME, DROP, ADD COLUMN, DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT, SET DEFAULT, ADD CHECK….
ALTER TABLE old_table_name RENAME TO new_table_name;	ALTER TABLE Customers RENAME TO Users;
ALTER TABLE table_name RENAME COLUMN old_col_name TO new_col_name;	ALTER TABLE Customers RENAME FirstName TO Fname;
ALTER TABLE *table_name* ADD COLUMN *col_name datatype(size)* CONSTRAINT, ADD COLUMN *col_name datatype(size)* CONSTRAINT ……;	ALTER TABLE Customers ADD COLUMN Email VARCHAR(50)); ALTER TABLE Customers ADD COLUMN Email VARCHAR(50) **NOT NULL DEFAULT 'N/A';**
ALTER TABLE *table_name* DROP COLUMN *col_name*, DROP COLUMN *col_name* ……;	ALTER TABLE Customers DROP COLUMN Email;

DROP TABLE

DROP TABLE [IF EXISTS] *table_name*
[CASCADE | RESTRICT];

DROP TABLE customers;
DROP TABLE [IF EXISTS] customers

DROP: deletes a table or a view or other objects

Restrict: won't delete the table if there are other dependencies

Cascade: will delete the table and the objects which depend on the table.

INSERT

INSERT: creates a record

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

INSERT ONE RECORD

```
INSERT INTO Customers (CustomerID, FirstName, LastName, Street, City, ZipCode, Phone)
VALUES (1, 'John', 'Zaki', 'ABC', 'Mansoura', '35511', '01000100100');
```

INSERT MORE THAN ONE RECORD WITH PK

```
INSERT INTO Customers (CustomerID, FirstName, LastName, Street, City, ZipCode, Phone)
VALUES (1, 'John', 'Zaki', 'ABC', 'Mansoura', '35511', '01000100100'),
VALUES(2, 'Ahmed', 'Sherif', 'DEF', 'Cairo', '51144', '01111101111'),
VALUES(3, 'Amir', 'Haitham', 'XYZ', 'Ismailia', '33311', '01222112222');
```

INSERT MORE THAN ONE RECORD WITHOUT PK

```
INSERT INTO Customers (FirstName, LastName, Street, City, ZipCode, Phone)
VALUES ( 'John', 'Zaki', 'ABC', 'Mansoura', '35511', '01000100100'),
VALUES( 'Ahmed', 'Sherif', 'DEF', 'Cairo', '51144', '01111101111'),
VALUES( 'Amir', 'Haitham', 'XYZ', 'Ismailia', '33311', '01222112222');
```

Customers		
PK	CustomerID	int
	FirstName	varchar(50)
	LastName	varchar(50)
	Street	varchar(50)
	City	varchar(50)
	ZipCode	varchar(50)
	Phone	varchar(11)

UPDATE

UPDATE: modifies existing records

```
UPDATE table_name
SET column1 = value1,
column2 = value2, ...

WHERE condition;
RETURNING *;
```

EX

```
UPDATE Customers
SET Firstname='Jo'
WHERE CustomerID=1;
```

If you do not use a WHERE clause,
you will update all columns in the table!!

DELETE

DELETE: deletes existing records

```
DELETE FROM table_name
WHERE condition;
```

```
DELETE FROM Customers
WHERE CustomerID=1;
```

If you do not use a WHERE clause,
all the data in the table will be deleted!!

SELECT

SELECT: retrieves certain records from one or more tables. Considered a DQL…

```
SELECT *
FROM table_name;
```

```
SELECT col1, col2,col3
FROM table_name;
```

```
SELECT col1, col2,col3
FROM table_name
WHERE condition;
```

```
SELECT *
FROM Customers;
```

```
SELECT CustomerID, FirstName, LastName
FROM Customers;
```

```
SELECT CustomerID, FirstName, LastName
FROM Customers
WHERE FirstName= 'john' ;
```

You have a plan!

STEPS TO DESIGN YOUR DB TABLES

1. **Understand the Purpose:** Determine the main function of your database to guide your data organization.
2. **Gather All Data Items:** Make a comprehensive list of all the pieces of information you think you'll need to store.
3. **Group Data into Entities:** Organize these data items into logical subjects or entities that represent a real-world concept, such as "Students" or "Modules". Each of these entities will become a table.
4. **Define Table Attributes (Columns):** Within each table, identify the specific attributes (data items) that describe the entity. For example, a "Products" table might have columns for "Name," "Price," and "SKU".
5. **Establish Primary Keys:** For each table, select a primary key—a column or set of columns that uniquely identifies each row. This could be a natural key (like an email address) or a surrogate key (like an auto-incrementing ID).
6. **Map Table Relationships:** Determine how tables are connected. If you have a table for "Customers" and another for "Orders," you'll likely use a foreign key in the "Orders" table (referencing the customer's primary key) to link each order to its customer.
7. **Design for Data Integrity:** Use foreign key constraints to ensure that related records are valid and that the relationships between tables are maintained.
8. **Refine and Normalize:** Review your design to eliminate redundant information and ensure that each table serves a specific purpose, following principles of database normalization.
9. **Consider Future Needs:** Design for flexibility and extensibility, allowing you to easily add new data types or expand the system as it grows.

Google

“what to do in a scrum session”

“

what to do in a scrum session

”