New Generation Academy
Transformed for community

Coding Academy

Trade: Software Programming and Embedded System SPES

SUBJECT NAME: BASICS OF DATABASE DEVELOPMENT

SUBJECT CODE : SPEDD302

Introduction

This module covers the skills, knowledge and attitude to maintain a website which facilitates the requirement as a front-end website developer. The module will allow the learner to resolve website issues, to respond to the customer requests, to add new features and to execute customer service support.

Database development is a crucial aspect of modern software engineering and data-driven decision-making. It involves designing, implementing, and maintaining databases that store, organize, and retrieve large volumes of structured and unstructured data. Databases are at the heart of many business applications, e-commerce websites, social media platforms, and scientific research projects, among others. They enable developers to store and manage data efficiently, secure it from unauthorized access, and retrieve it quickly and accurately.

To develop a database, developers need to follow a systematic approach that involves several stages, such as requirement analysis, conceptual modelling, logical and physical design, implementation, testing, and maintenance. They need to select an appropriate database management system (DBMS) that matches the requirements and constraints of the project, such as scalability, performance, security, and compatibility. Moreover, they need to use standard query languages, such as SQL (Structured Query Language), to interact with the database and manipulate data. Overall, database development requires a blend of technical skills, analytical thinking, and creativity to design and implement effective data management solutions that meet the needs of various stakeholders.

What is a Database?

A database is a systematic collection of data. They support electronic storage and manipulation of data. Databases make data management more efficient by allowing users to store, retrieve, and manipulate large amounts of information quickly and securely. Typically, databases are managed using a Database Management System (DBMS), which is a software tool that provides the necessary interfaces for database interaction.

Types of Databases

1.Relational Databases: These are the most common type of databases. They store data in tables, with each table consisting of rows and columns. Each row represents a record, and each column represents a data field. SQL (Structured Query Language) is typically used to manage and query data in these databases. Examples include MySQL, PostgreSQL, and Oracle.

1.NoSQL Databases: These databases are designed to handle large volumes of unstructured data. They are known for their flexibility, as they do not require a fixed schema and are scalable. Common types of NoSQL databases include document-oriented (MongoDB), key-value stores (Redis), wide-column stores (Cassandra), and graph databases (Neo4j).

1.Object-oriented Databases: These databases store data in the form of objects, as used in object-oriented programming. They are beneficial when complex data relationships need to be represented.

2.Distributed Databases: These are spread across multiple physical locations, either on multiple computers or across a network. They are useful for large organizations with decentralized operations.

New Generation Academy

Transformed for community

Coding Academy

Why Do We Need Databases?

1.Data Organization and Retrieval: Databases provide a structured way to store data, making it easier to organize, search, and retrieve information. This organization is crucial for businesses and organizations that handle large amounts of data.

2.Data Integrity and Security: Databases enforce rules that ensure data accuracy and consistency. They also come with security features that protect sensitive data from unauthorized access or breaches.

3. Efficient Data Management: With databases, multiple users can access and manipulate data concurrently without compromising its integrity. This is essential in a collaborative environment where data is continuously updated.

4. Decision Making and Reporting: Databases enable organizations to analyze their data effectively, leading to more informed decision-making. They are integral in generating reports, forecasts, and trends essential for strategic planning.

5. Automation and Integration: Databases can automate regularly tasks like data entry, updating, and reporting. They can also be integrated with other applications, enhancing functionality and efficiency.

Challenges in Database Management

While databases are powerful tools, they come with challenges. These include ensuring data security, maintaining data integrity, and managing large volumes of data effectively. Additionally, with the advent of big data, databases must handle increasingly large and complex datasets, requiring constant updates and scalability.

New Generation Academy

Transformed for community

Coding Academy

Shortcomings of Traditional File Processing System

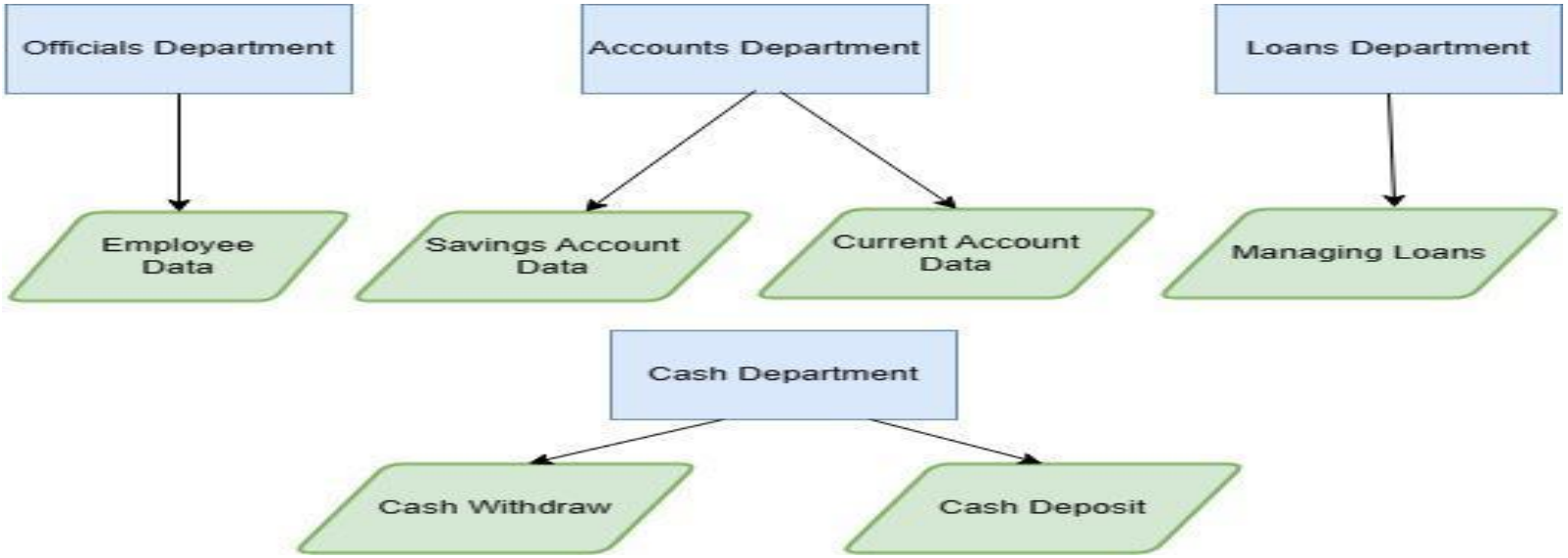
We have seen the importance of a Database management system and how it is better than the Traditional file systems that were used before DBMS to store data.

Before the origin of Computer Systems or before their heavy usage, the data were used to be stored in files manually. This means that the same data could have been present multiple times in a single institution. Although it was easy to find those data as one piece of data was present in multiple places (Data Redundancy) but accessing those data was not an easy task to do, every time someone who wanted to access those files in computer should have to write a file name, this technique was only suitable for small organizations where the amount of data to be stored was relatively lesser. Each unit of information stored in files was known as a "flat file".

What is the Traditionally used "File System"?

Considering a scenario of a bank before the introduction of DBMS, for example, say someone went to the bank to deposit a certain amount in their account. So as the DBMS is not available so the bank employee has to manually register their account number, name, and amount in either a written manner or type and store them locally in the computer as a file.

The problem which might arise that while writing if the employee mistakenly writes any digit of their account number or amount wrong then there would be a major issue and as there is no Database so it would be really hard to know what was the last state of that person's account before this deposit.



In the diagram above we can see how the details used to be stored by bank employees before the introduction of DBMS. Each department would handle some specific tasks and store the data locally in their computers or registers without knowing what is happening in the other department.

Characteristics of Traditional File Systems

- The data of certain companies or organizations were kept as "Files".
- The files stored in different departments were independent of each other, which caused severe data redundancy.

- Each file includes information for a particular department or region, such as the library, tuition, and students' exams.
- The traditional file system is way less flexible than DBMS and has many disadvantages.
- The maintenance of those files was also of high cost.
- Each of the units of "Files" used to be known as "Flat Files".

Disadvantages of the Traditional File Systems

Distinguished and Isolated Data:

Imagine a user needs information that is not possible to be provided using a single file, multiple different files were required, which are situated in different departments. So all the employees first need to manually and carefully check each of the files in each department and find the relationship between them to figure out the information that the user wants.

- **Data Duplication / Data Redundancy**

•Storing the same data multiple times not only wastes resources in every machine but also is costly to maintain and wastes time.
•Loss of data integrity is another major issue of Data Redundancy, imagine someone's address is present in multiple systems and he has applied to update the address, in one system the address gets updated but in the rest of them it remains the same, so the if someone from any different department where the data is not updated tries to send them any letter or something then it would go to a wrong address.

•**Data Protection -** Data protection was very less due to different reasons like Data Redundancy, manual storing of data, easy access of confidential data by unauthorized parties, etc.

•**Issues with Transactions -** It didn't follow the ACID (Atomicity, Consistency, Isolation, and Durability) properties, for that if in the middle of any transaction the system crashed then it would leave the system in an inconsistent state.

•**Concurrent issues -** Two or more users can view the same file simultaneously, but the problem arises when they try to update the same file simultaneously.

Data

Data are just structured facts and measurements which are True and construct meaning. Data is used for many different purposes like calculations, different discussions, proofs, etc. It doesn't only include some textual information, data can be numbers, events, certain actions, etc. Data can be stored in many different places which include -

- Spreadsheets
- Folders
- Lists
- Digital drives like Hard Disk, SSD, Floppy Disk, CD etc

A database is like a bag that can hold different types of information,
no matter in which format they come.

Advantages of using Database System in place of Traditional File Systems

•**Searching Data -** In the case of the Traditional file systems, the programmer needs to write lengthy programs everytime to fetch certain information, with DBMS some 2-3 line query is enough to fetch as many data as we want. Also, one language is supported by many databases (with a slight variety of syntax).

•**Data Reliability -** Data reliability is also high in DBMS as it supports user-defined data types also apart from the traditional in-built data types.

•**System Failure -** As DBMS follows ACID properties, even if a system failure happens in between a transaction, nothing will be lost, and we can restart that transaction from its previous stable state.

•Data Protection :

DBMS comes up with lots of methods to protect the data stored inside it rather than just Passwords

•Backup of Data :

Data Backup is possible in DBMS, which was not present in the Traditional File systems.

•Variation of Interfaces :

DBMS comes with different kinds of interfaces like Graphicals or Tabular.

•Maintenance of the Database:

As DBMS is a centralized structure it is easier to maintain it rather than the Traditional File systems

New Generation Academy

Transformed for community

Coding Academy

Database Users

A Database User is defined as a person who interacts with data daily for updating, reading, and modifying the given data. Database users can access and retrieve data from the database through the Database Management System (DBMS) applications and interfaces.

Types of Database Users

Database users are categorized based on their interaction with the database. There are seven types of database users in DBMS. Below mentioned are the types of database users:

1. Database Administrator (DBA)

A Database Administrator (DBA) defines the schema and manages all three levels of the database architecture. They create user accounts, control access, and ensure data security by authorizing only trusted users. DBAs are also responsible for handling security breaches and optimizing system performance.

- DBA also monitors the recovery and backup and provides technical support.
- The DBA has a DBA account in the DBMS which is called a system or superuser account.
- DBA repairs damage caused due to hardware and/or software failures.

2. Naive / Parametric End Users

Parametric End Users are the unsophisticated who lacks DBMS knowledge but they frequently use the database applications in their daily life to get the desired results. For example, Railway's ticket booking users, Clerks in any bank are naive user because they lacks DBMS knowledge but they still use the database.

3. A System Analyst

A system Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.

4. Sophisticated Users

Sophisticated users like engineers or analysts interact directly with the database using SQL queries. They don't write full application code but use the query processor to access and manipulate data. These users develop custom database applications based on their specific needs.

5. Database Designers

Database Designers create the structure of a database, including tables, views, indexes, triggers, and constraints. They decide what data to store and how it should be related based on user requirements. Their design is finalized before the database is built or populated with data.

6. Application Programmers

Application Programmers, also known as System Analysts or Software Engineers, write backend code for application programs. They use languages like C, Visual Basic, COBOL, PHP etc., to design, test, and maintain programs. These programs, called “ canned transactions," help naive users interact with the database efficiently.

7. Casual Users / Temporary Users

Casual Users are the users who occasionally use/access the database but each time when they access the database they require the new information, for example, Middle or higher level manager.

8. Specialized users

Specialized users are sophisticated users who write specialized database application that does not fit into the traditional data-processing framework. Among these applications are computer aided-design systems, knowledge-base and expert systems etc.

Data Independence

Definition: Ability to change schema at one level without affecting the next level.

In simpler terms, it means that **changes in data storage or structure do not affect how users or applications access the data**. It is one of the most important goals of a **Database Management System (DBMS)** — to separate data from the programs that use it.

Levels of Data Independence

There are **two types** of data independence:

A. Physical Data Independence

This means that **changes in the physical storage** of data **do not affect the logical structure** of the database.

•**Physical level**: how the data is actually stored in files (for example, indexing, compression, data placement, storage devices).

•**Logical level**: describes *what data* is stored (tables, fields, relationships).

Example of Physical Data Independence:

•Suppose a database administrator decides to move data from **HDD** to **SSD** for faster access or adds **indexes** to improve performance. Applications that query the data (like SELECT * FROM students WHERE id = 5;) **don’t need to change** — they still work the same way.

B. Logical Data Independence

This means that **changes in the logical structure** of the database **do not affect the external views or application programs**.

Example of Logical Data Independence:

Suppose a university adds a new column email to the Students table or merges FirstName and LastName into FullName. The applications or reports that only use other columns (like StudentID, Class, Marks) ***will continue to work*** *without modification.*

Importance of Data Independence

Reduced maintenance : user can change the database structure without rewriting application code.
Flexibility : Easier to evolve and improve database performance or structure.
Data abstraction : Users focus on what data they need, not how it's stored.
Security and control: DBAs can optimize storage or structure without exposing internal details to users.

Quick Examples data Independence

1.Bank Database

- Adding an index on AccountNumber (physical change) doesn't affect how tellers access account data.
  (Physical Data Independence)

2.Library System

- Adding a new attribute Genre to the Books table doesn't affect the existing book search system.
  (Logical Data Independence)

3.Hospital Database

- Moving patient records to cloud storage doesn't affect the doctor's patient portal.
  (Physical Data Independence)

Database Languages

A **database language** is a specialized language used to **define, manipulate, and control data** in a database. These languages allow users and applications to interact with the **Database Management System (DBMS)** to perform operations like storing, retrieving, updating, and deleting data. We have **three main types**
Data Definition Language (DDL), Data Manipulation Language (DML) and Data Control Language (DCL)

1. Data Definition Language (DDL) this is used To **define the structure** of the database, including tables, schemas, indexes, and constraints. It also Determine what data needs to be stored, relationships, and how it will be used (Requirements). DDL Used by database administrators and developers to create the database structure.

New Generation Academy
Transformed for community

How it work ?

Coding Academy

- **Identify Requirements:** Determine what data needs to be stored, relationships, and how it will be used.
- **Define Database Schema:** Outline tables, relationships, and overall database organization.
- **Create Tables:** Design the tables based on the schema.
- **Define Columns:** Specify each column and its attributes (name, type, length).
- **Set Primary Keys:** Decide unique identifiers for each table.
- **Set Foreign Keys:** Establish relationships between tables.
- **Define Indexes:** Create indexes for faster search and queries.
- **Define Constraints:** Include rules like NOT NULL, UNIQUE, CHECK to maintain data integrity.
- **Review & Optimize:** Check design for efficiency, redundancy, and normalization.

Common DDL Commands are the following :

Command	Purpose	Example
CREATE	Create a new table or database	CREATE TABLE Students (ID INT, Name VARCHAR(50), Age INT);
ALTER	Modify the structure of an existing table	ALTER TABLE Students ADD COLUMN Email VARCHAR(100);
DROP	Delete a table or database	DROP TABLE Students;
TRUNCATE	Remove all data from a table	TRUNCATE TABLE Students;

B. Data Manipulation Language (DML)

- It used To **insert, update, delete, and retrieve data** in the database.

DML Used by end-users or application programs.

How it work?

- **Connect to Database:** Establish a connection to the database.
- **Select Operation:** Choose whether you want to Insert, Update, Delete, or Retrieve.
- **Insert Data:** Input new data, validate it, and run an INSERT query.
- **Update Data:** Select existing data, modify it, validate, and run an UPDATE query.
- **Delete Data:** Select data to remove, confirm, and run a DELETE query.
- **Retrieve Data:** Run a SELECT query to fetch and display data.
- **Handle Success/Error:** Show messages based on whether the query was successful.

Common DML Commands are :

Command	Purpose	Example
INSERT	Add new records	INSERT INTO Students VALUES(1, 'Alice', 20);
UPDATE	Modify existing records	UPDATE Students SET Age = 21 WHERE ID = 1;
DELETE	Remove records	DELETE FROM Students WHERE ID = 1;
SELECT	Retrieve data	SELECT * FROM Students;

C. Data Control Language (DCL)

- Used to **control access and permissions** in a database.

It Used by database administrators to ensure security and access control. This Commands grant or revoke rights to users.

How it work?

- **Connect to Database:** Establish connection to manage access.
- **Identify Users/Roles:** Determine individual users and/or groups (roles).
- **Define Access Levels:** Decide what each user/role can do in the database.
- **Assign Permissions:** Apply permissions like SELECT, INSERT, UPDATE, DELETE either directly to users or through roles.
- **Validate Access Rules:** Make sure permissions are correctly applied.
- **Test Access:** Verify that users can only perform allowed actions.
- **Grant or Adjust Access:** Allow access if correct; adjust if incorrect.

Common DCL Commands are:

Command	Purpose	Example
GRANT	Give permissions	GRANT SELECT, INSERT ON Students TO user1;
REVOKE	Remove permissions	REVOKE INSERT ON Students FROM user1;

D. Transaction Control Language - TCL

Transaction Control Language – TCL used to manage **transactions** in a database, ensuring **data integrity**. It Ensures that operations like updates, inserts, and deletes happen completely or not at all.

How it work ?

- **Connect to Database:** Establish a connection.
- **Begin Transaction:** Start a transaction block so multiple operations can be treated as one unit.
- **Perform Operations:** Execute data manipulation queries (INSERT, UPDATE, DELETE).
- **Check for Errors:** Verify whether all operations succeed.
- **Commit Transaction:** If everything is correct, save all changes permanently.
- **Rollback Transaction:** If there's any error, undo all changes to maintain data integrity.
- **Confirm Success / Show Error:** Provide feedback on transaction result.

Common TCL Commands are:

Command	Purpose	Example
COMMIT	Save changes permanently	COMMIT;
ROLLBACK	Undo changes	ROLLBACK;
SAVEPOINT	Set a point to roll back to	SAVEPOINT sp1;

Summary

Type	Purpose	Examples
DDL	Define database structure	CREATE, ALTER, DROP
DML	Manipulate data	INSERT, UPDATE, DELETE, SELECT
DCL	Control access	GRANT, REVOKE
TCL	Manage transactions	COMMIT, ROLLBACK, SAVEPOINT

Real-life Examples

1. School Database

DDL: Create Teachers and Students tables.
DML: Add a new student record.
DCL: Allow teachers to update only marks.
TCL: Commit marks after final approval.

2. Bank Database

DDL: Define Accounts table.
DML: Deposit or withdraw money.
DCL: Give bank clerks only read permission.
TCL: Ensure money transfer happens fully or not at all.

Examples of Databases

1. **School Database:** Stores student info, marks, attendance, and teachers.
2. **Bank Database:** Stores customer accounts, transactions, and loan info.
3. **Hospital Database:** Stores patient records, appointments, and medicines.

Data Model

Definition

A **data model** is a **conceptual framework** that describes **how data is organized and how relationships are maintained** in a database. It provides a **structure for data representation** and helps in designing the database. It ensures **consistency, integrity, and easy retrieval** of data.

More Explanation:

- **Connect to Database:** Establish a connection to manage models.
- **Select Source Data Model:** Choose the existing data model you want to copy.
- **Check Structure:** Review tables, columns, constraints, and indexes.
- **Create New Data Model:** Initialize a new model in the database.
- **Copy Tables:** Duplicate table structures.
- **Copy Columns & Data Types:** Ensure all columns and data types are accurately copied.
- **Copy Constraints & Indexes:** Transfer primary keys, foreign keys, unique constraints, and indexes.
- **Verify New Model:** Check that the new data model matches the original.
- **Success / Adjust & Retry:** Confirm the copy; if issues exist, fix and retry.

Example

Suppose you have a database for a **school management system** called SchoolDB with the following structure:

- **Tables:** Students, Teachers, Classes
- **Columns:**
  - Students: student_id, name, age, class_id
  - Teachers: teacher_id, name, subject
  - Classes: class_id, class_name, teacher_id
- **Constraints:**
  - Primary keys: student_id, teacher_id, class_id
  - Foreign keys: Students.class_id → Classes.class_id, Classes.teacher_id → Teachers.teacher_id

Steps to Copy the Data Model

1.Connect to Database
- Use a database tool like MySQL Workbench or SQL Server Management Studio.

2.Select Source Data Model
- Choose SchoolDB as the source.

3.Analyze Structure
- Check all tables, columns, primary keys, foreign keys, and indexes.

4.Create New Data Model
- Create a new database SchoolDB_Copy.

5.Copy Tables
- Create tables Students, Teachers, Classes in SchoolDB_Copy.

6.Copy Columns & Data Types
- Copy all columns and their types exactly

Types of Data Models

A. Hierarchical Data Model

•Data is organized in a **tree-like structure**.

•Each record has a **parent-child relationship**.

Example: **Company database** (Departments → Employees)

B. Network Data Model

•Data is organized as a **graph**.

•Each record can have **multiple parent and child records** (many-to-many relationships).

Example of **Network Data Model** : **Airline reservation system** (Flights ↔Passengers)

C. Relational Data Model

•Data is organized in **tables (relations)** with **rows (tuples)** and **columns (attributes)**.

•Most widely used model in DBMS.

Example: **Student Table where** | StudentID | Name | Age | Class |

D. Entity-Relationship (ER) Model

•Uses **entities** (objects) and **relationships** between them.

Example: Students (entity) take Courses (relationship)

Importance of Data Models

1.Helps in **database design**.
2.Ensures **data consistency**.
3.Simplifies **querying and reporting**.
4.Provides a **blueprint** (Plan / Design / Map) for developers and DBAs.

New Generation Academy
Transformed for community

Coding Academy

RELATIONAL DATA MODEL AND RELATIONAL DATABASE CONSTRAINTS

Storing and managing information is one of the most important tasks for computers. The way in which information is organized can have a profound effect on how easy it is to access and manage. Perhaps the simplest but most versatile way to organize information is to store it in the forms of tables. Table is the backbone of the relational model.

In general : The Relational Model is a way of organizing and managing data in a database using tables.

Brief History

The relational model of database was introduced in 1970, by English **Dr. Edgar Frank "Ted" Codd** (19 August 1923 – 18 April 2003), a mathematician and computer scientist working at **IBM**.

The relational data model was developed for databases — that is, information stored over a long period of time in a computer system — and for database management systems, the software that allows people to store, access, and modify this information. Databases still provide us with important motivation for understanding the relational data model. They are found today not only in their original, large-scale applications such as train booking systems or hospital management systems, but in desktop computers handling individual activities such as maintaining expense records, homework grades, and many other uses.

Advantages of Relational Model

1.Simplicity

Data is organized in tables (rows and columns), making it easy to understand and use.

2.Reduced Data Redundancy

Data is stored only once and shared using relationships, which minimizes duplication.

3.Data Integrity and Consistency

Integrity constraints (primary key, foreign key, domain constraints) ensure accurate and consistent data.

4.Easy Data Retrieval

Data can be easily accessed and manipulated using **SQL** without knowing physical storage details.

5.Flexibility

Tables can be modified (add/remove columns or tables) with minimal impact on the entire database.

6.Security
Access control and authorization can be applied at table, row, or column level.
7. Data Independence
Changes in physical storage do not affect the logical structure or applications.
8. Support for Relationships
Relationships between tables are easily created and maintained using keys.
9. Standardization
Uses standard query languages like SQL, supported by most DBMSs.
10. Scalability and Reliability
Suitable for small to very large databases with proper performance and reliability.

☞ **In short:** The relational model is easy to use, reliable, secure, and efficient for managing structured data.

New Generation Academy

Transformed for community

Coding Academy

Limitations of Relational Model

Cost

The underlying cost involved in a relational database is quite expensive. For setting up a relational database, there must be separate software which needs to be purchased. And a professional technician should be hired to maintain the system. All these can be costly, especially for businesses with small budget.

Performance

Always the performance of the relational database depends on the number of tables. If there are a greater number of tables, the response given to the queries will be slower. Additionally, more data presence not only slows down the machine, it eventually makes it complex to find information. Thus, a relational database is known to be a slower database.

Physical Storage

A relational database also requires tremendous amount (**very large quantity** or **a huge amount**) of physical memory since it is with rows and columns. Each of the operation depends on separate physical storage. Only through proper optimization, the targeted applications can be made to have maximum physical memory.

Complexity

Although a relational database is free from complex structuring, occasionally it may become complex too. When the amount of data in a relational database increases, it eventually makes the system more complicated.

Information Loss

Large organizations tend to use a greater number of database systems with more tables. This information can be used to be transferred from one system to another. This could pose a risk of data loss.

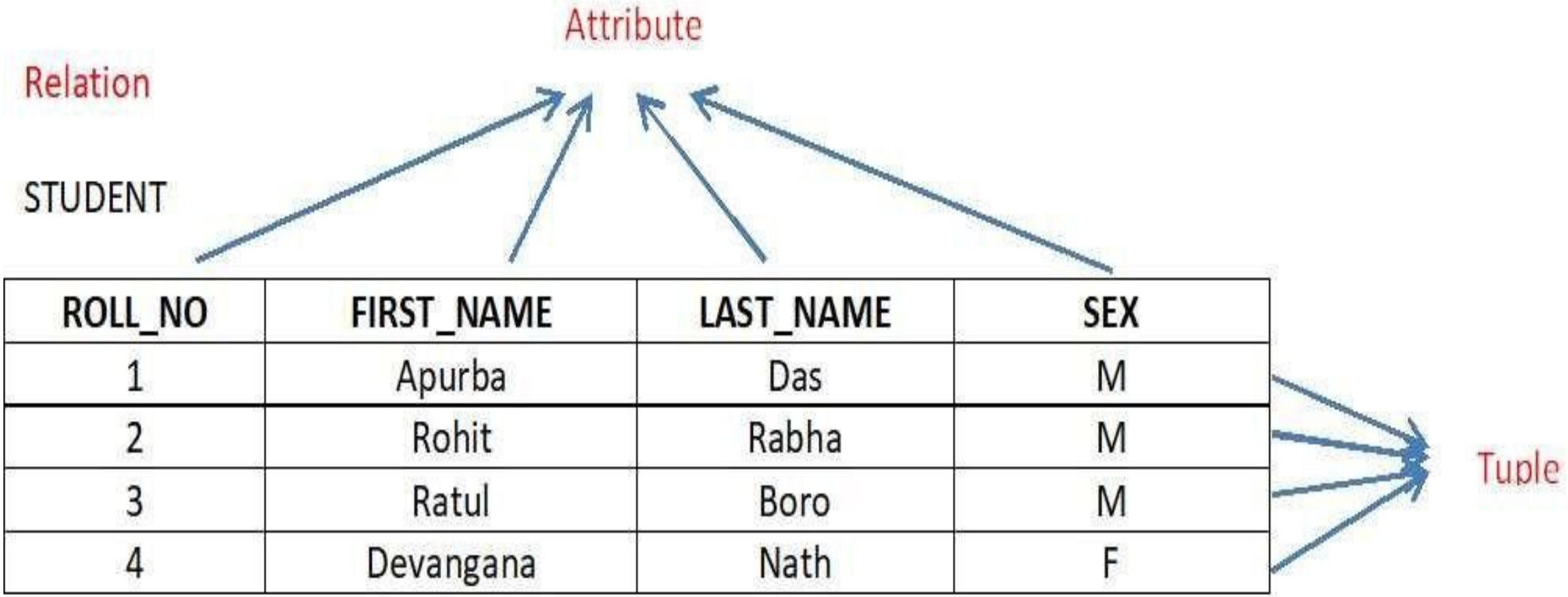
Structure Limitations

The fields that are present on a relational database has limitations. Limitations are in that sense that it cannot accommodate more

COMPONENTS AND RELATIONAL TERMINOLOGIES

In the **Relational Model**, data is organized in a structured way using specific components and terminologies.

Domain: A domain is a set of values permitted for an attribute in a table. Domain is atomic. For example, ROLL_NO can only be a positive integer. A data type or format is also specified for each domain. It is possible for several attributes to have the same domain.



ROLL_NO	FIRST_NAME	LAST_NAME	SEX
1	Apurba	Das	M
2	Rohit	Rabha	M
3	Ratul	Boro	M
4	Devangana	Nath	F

Attribute: Attributes are the characteristics of a relation. Each column in a table is the attribute. Attributes are the properties which define a relation. e.g.,ROLL_NO, FIRST_NAME etc of the relation Student. Each attribute in a relational model must have domain information. Domain information contains the following:

- **Data Type:** Databases provide support for different types of data and their variants. For example, integer, float etc.
- **Length:** Length means number of characters or digits that an attribute value has. For example, when we assign a PIN code, it has 6 digits.
- **Date format:** A date contains day, month and year. These three must be given in combination. Such as DD/MM/YYYY or MM/DD/YYYY or YYYY/MM/DD etc.
- **Range:** A range is specified by lower and upper bounds of data values that an attribute may have.
- **Constraints:** These are particular type of conditions that put restrictions on values that are allowed.
- **NULL support:** There is a support for NULL values in relational model. Some particular attribute may remain blank. For example, in a relation the column “PAN_Number” may be blank as a person may not have a Permanent Account Number.
- **Default Value:** If nothing is entered, database assigns a default value. Relational model supports the facility that the default value may be set for every attribute.

Tuple – It is nothing but a single row of a table, which contains a single record.

Relations- are in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

Relation Schema- A relational schema is the design for the table. It includes none of the actual data, but is like a blueprint or design for the table, so describes what columns are on the table and the data types. It may show basic table constraints (e.g., if a column can be null) but not how it relates to other tables.

Degree- is the number of attributes n of its relation schema. A relation ofdegree four, which stores information about college students, would contain four attributes describing each student as follows:

STUDENT(Roll_No, First_Name, Last_name, Sex)

Cardinality: Total number of rows present in the Table.

Relation Instance–Relation instance is a finite set of tuples at a given time. Relation instances do not have duplicate tuples.

Null Value: Afield with a NULL value is a field with no value. Primary key can't be a null value.

KEYS IN RELATIONAL MODEL

Illustration of Relational Schema

Keys are very important part of Relational database model. A Key can be a single attribute or a group of attributes, where the combination may act as a key. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

Why do we need a Key?

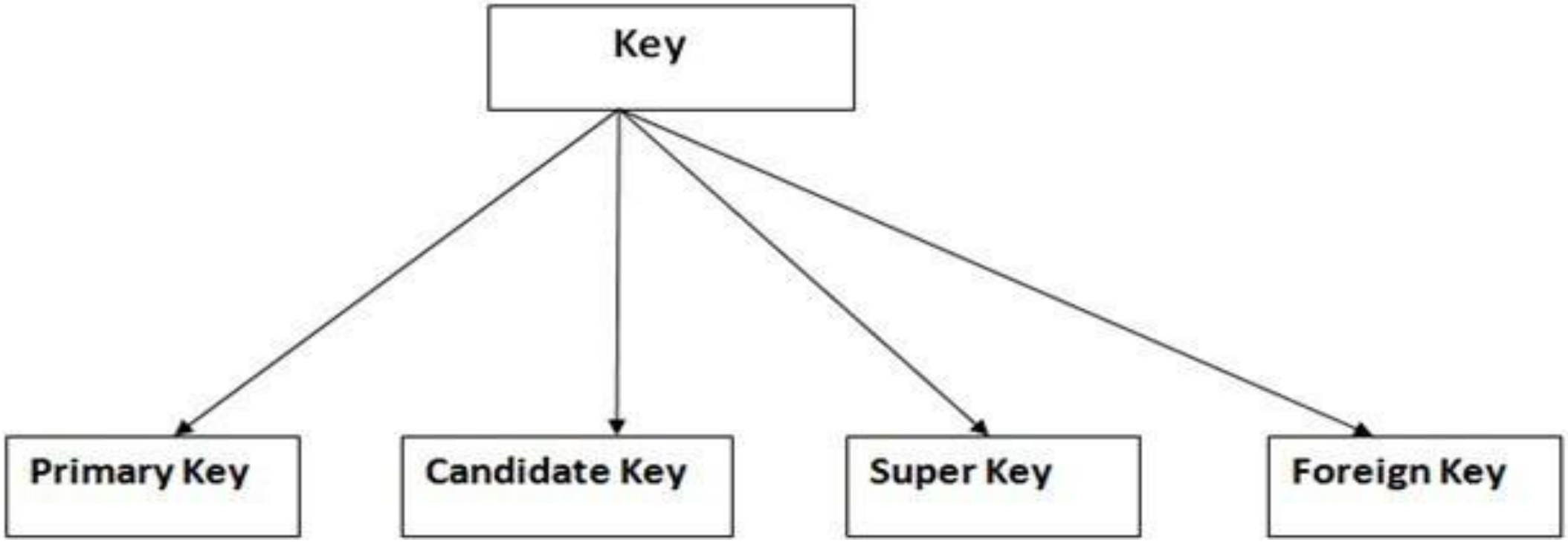
In real world applications, number of tables required for storing the data is huge, and the different tables are related to each other as well. Also, tables store a lot of data in them. A table generally extends to thousands of records stored in them, unsorted and unorganised.

Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data?

To avoid all this, Keys are defined to easily identify any row of data in a table.

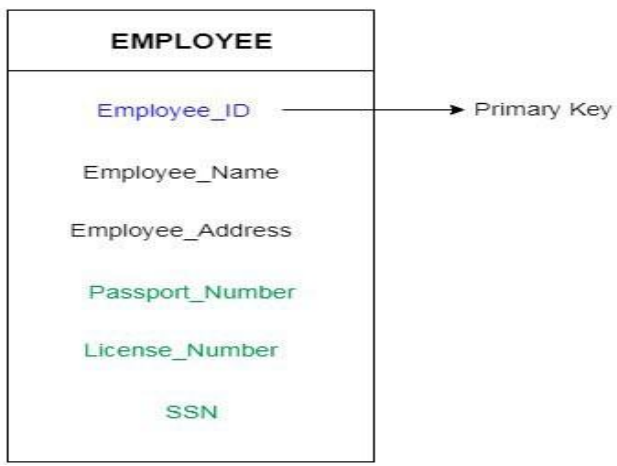
For example: In STUDENT table above, The attribute ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

Types of Keys: Different types of keys are shown in the following figure



- **Primary key**
  - It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table . The key which is most suitable from those lists become a primary key.

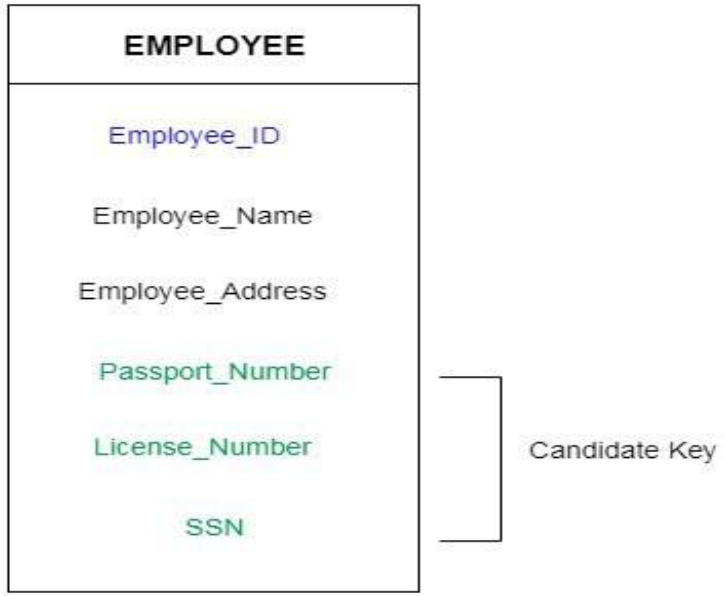
- In the EMPLOYEE table below , ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.



- **Candidate key**

  Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key.

  For example: In the EMPLOYEE table, Employee_id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.



- **Super Key**

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.

For example: In the above EMPLOYEE table , for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key. The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE- NAME), etc.

Let's take a simple **STUDENT** table with the attributes: **student_id, name**, **phone** and **age**.

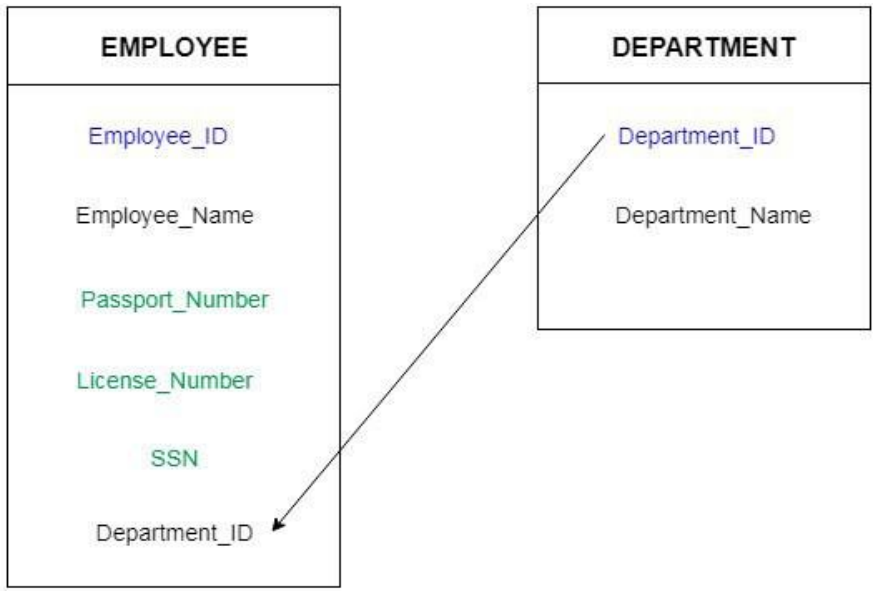
STUDENT Relation

student_id	name	phone	age
1	Alpha	1234567890	17
2	Alpha	1234567891	18
3	Kheilla	1234567892	19

In the table defined above super key would include student_id, (student_id, name), phone etc. The first one is pretty simple as student_id is unique for every row of data; hence it can be used to identity each row uniquely. Next comes, (student_id, name), now name of two students can be same, but their student_id can't be same hence this combination can also be a key (**Super Key).**

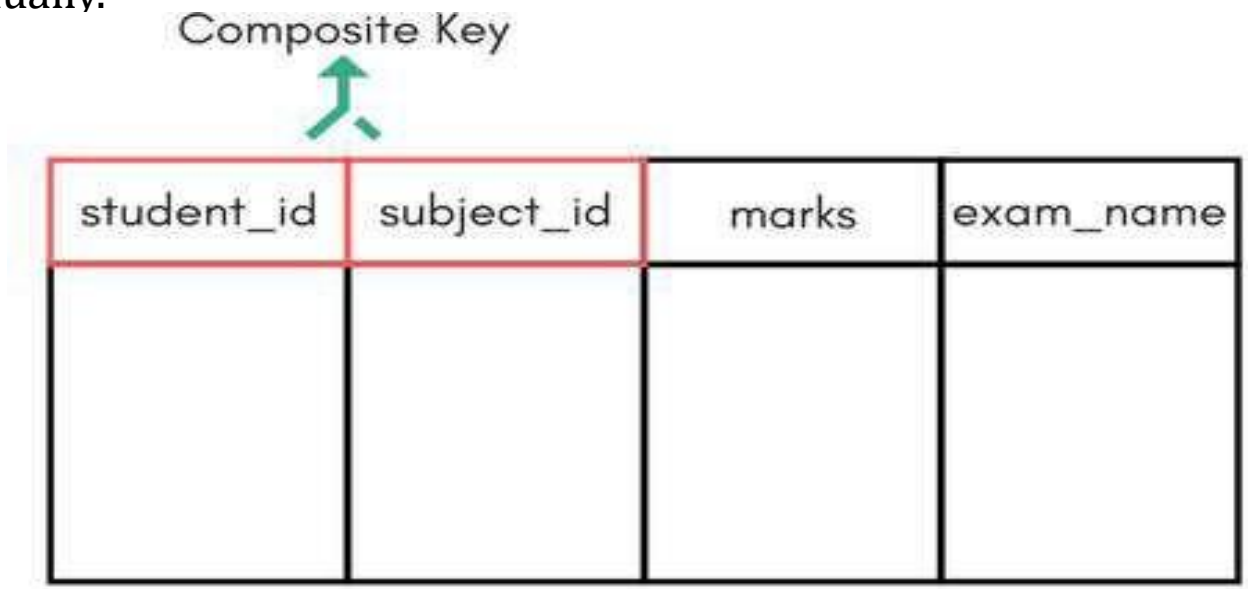
- **Foreign key**
  - Foreign keys are the column of the table which is used to point to the primary key of another table. If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers.

In a company, every employee works in a specific department, and employee and department are two different entities. So, we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table. We add the primary key of the DEPARTMENT table , Department_Id as a new attribute in the EMPLOYEE table. Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independently or individually.



Score Table – To save scores of the student for various subjects.

In the above picture we have a Score table which stores the marks scored by a student in a particular subject. In this table, student_id and subject_id together will form the primary key and hence it is a composite key.

Other Keys you must know are :

Secondary or Alternative Key

The candidate keys which are not selected as primary key are known as secondary keys or alternative keys.

Non-key Attributes

Non-key attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table.

Non-prime Attributes

Non-prime Attributes are attributes other than Primary Key attribute(s)**.**

Summary Table of Terms

Term	Meaning	Example
Relation	Table	Student
Tuple	Row	(1, Alice, 20)
Attribute	Column	Name
Domain	Allowed values for attribute	Age: 0–120
Primary Key	Unique identifier	StudentID
Foreign Key	Links to another table	StudentID in Enrollment
Candidate Key	Possible unique identifiers	StudentID, Email
Composite Key	Multiple attributes for uniqueness	(StudentID, CourseID)
Super Key	Set of attributes that uniquely identify	(StudentID, Name)
Relation Schema	Structure of the table	Student(StudentID, Name…)
Relation Instance	Actual data	All rows in the table

New Generation Academy

Transformed for community

Coding Academy

Entity-Relationship Modeling

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

What is an Entity Relationship Diagram (ER Diagram)?

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

Facts about ER Diagram Model:

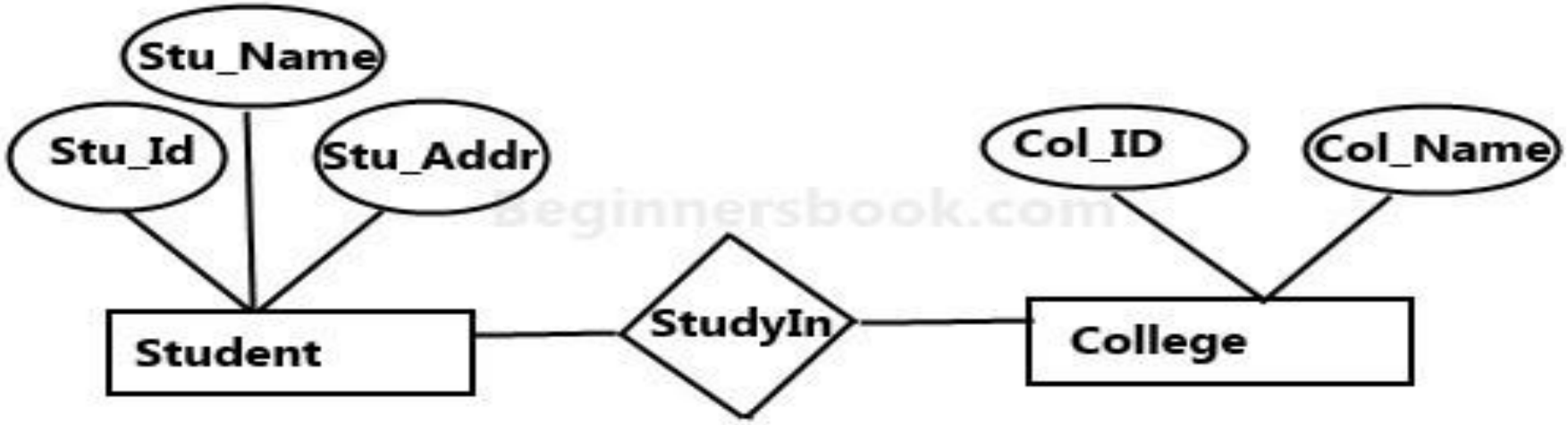
ER model allows you to draw Database Design
It is an easy to use graphical tool for modeling data Widely used in Database Design
It is a GUI representation of the logical structure of a Database
It helps you to identifies the entities which exist in a system and the relationships between those entities

Why use ER Diagrams?

Here, are prime reasons for using the ER Diagram
Helps you to define terms related to entity relationship modeling
Provide a preview of how all your tables should connect, what fields are going to be on each table
Helps to describe entities, attributes, relationships
ER diagrams are translatable into relational tables which allows you to build databases quickly ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications

A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.



Sample E-R Diagram

Explanation:

1. Rectangles (Entity Sets)

In E-R modeling, a rectangle represents an **Entity Set**. An entity is a "thing" or object in the real world that is distinguishable from others.

•**Student:** Represents the collection of all students in the system.

•**College:** Represents the collection of all colleges in the system.

2. Ellipses (Attributes)

The ovals or ellipses represent **Attributes**, which are the specific properties or characteristics that describe an entity.

•**Student Attributes:** Stu_Id, Stu_Name, and Stu_Addr. These are the data points you will store for every student.

•**College Attributes:** Col_ID and Col_Name. These describe the college entity.

•**Note on Keys:** Although not underlined in your specific image, Stu_Id and Col_ID typically serve as **Primary Keys** to uniquely identify each record.

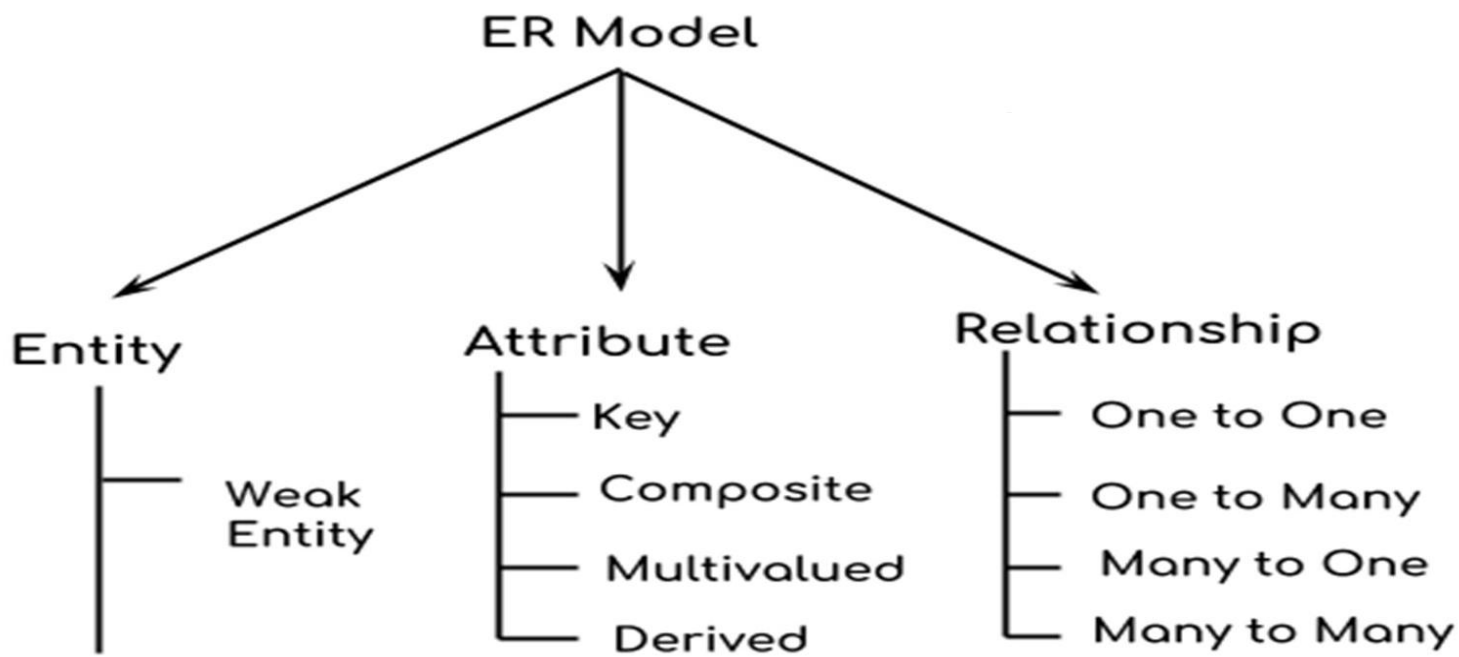
3. Diamonds (Relationship Sets)

The diamond shape represents a **Relationship Set**, which defines how two entities interact or relate to one another.

•**StudyIn:** This relationship connects "Student" and "College". It tells us that students have a relationship with a college because they study there.

4. Lines (Links)

The lines are used to connect attributes to their respective entity sets and to link entity sets to the relationship.



Components of ER Diagram

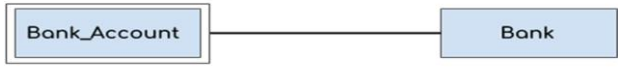
As shown in the above diagram, an ER diagram has three main components:
1.Entity 2. Attribute 3. Relationship

2.Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college.

Weak Entity:

An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.



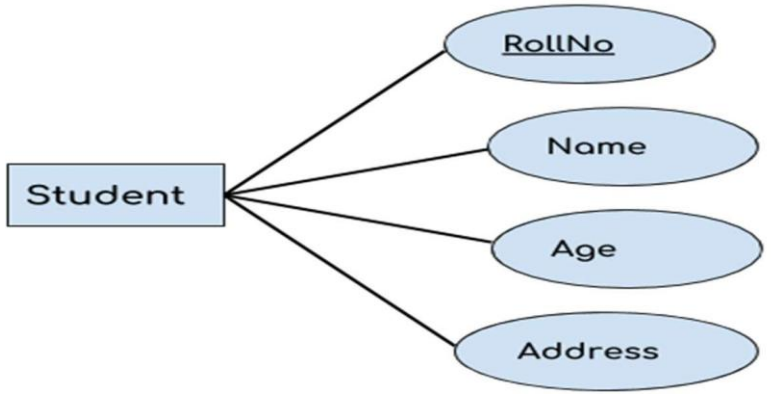
Weak Entities

A weak entity is a type of entity which doesn't have its key attribute. It can be identified uniquely by considering the primary key of another entity. For that, weak entity sets need to have participation.

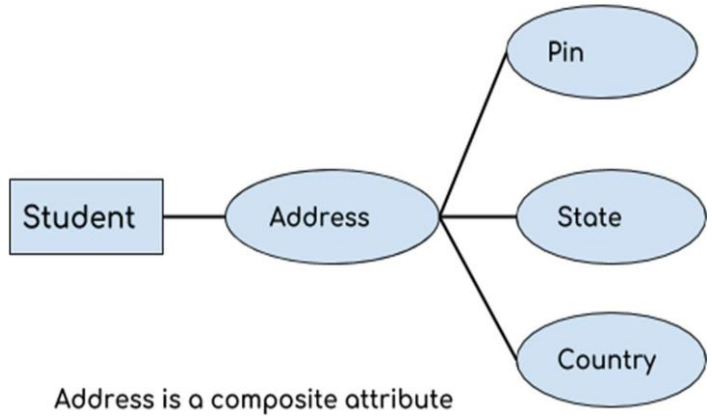
Strong Entity Set	Weak Entity Set
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
In the ER diagram the relationship between two strong entity set shown by using a diamond symbol.	The relationship between one strong and a weak entity set shown by using the double diamond symbol.
The connecting line of the strong entity set with the relationship is single.	The line connecting the weak entity set for identifying relationship is double.

1. Key attribute:

A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.



2. Composite attribute: An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.

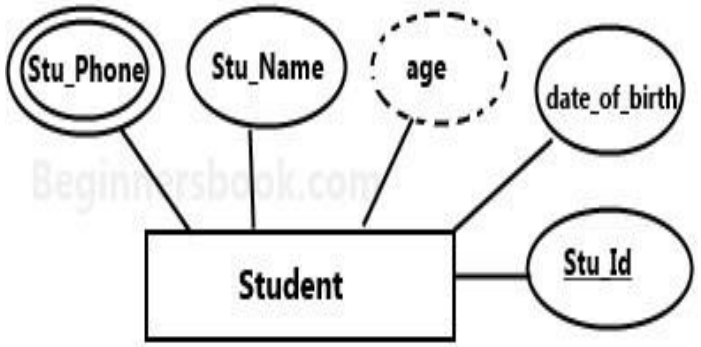


3.Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4.Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).



3.Relationship

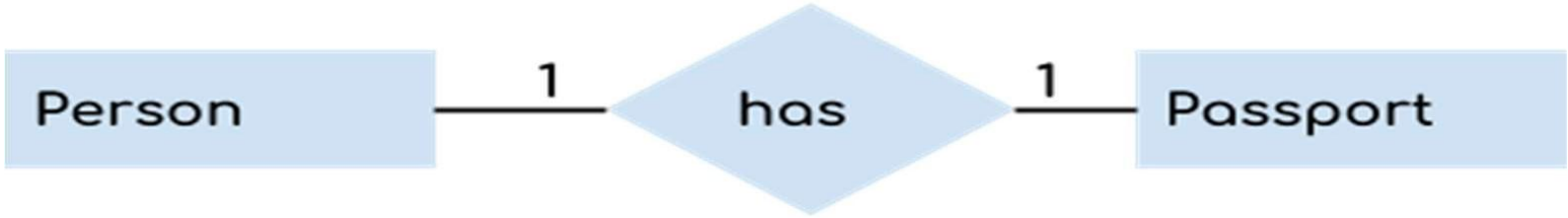
Cardinality: Defines the numerical attributes of the relationship between two entities or entity sets.

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of cardinal relationships:

1.One to One 2. One to Many 3. Many to One 4. Many to Many

1.One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called **one to one relationship**. For example, a person has only one passport and a passport is given to one person.

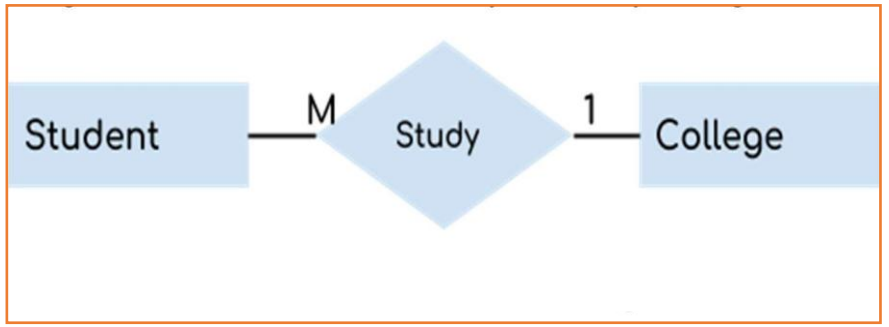


2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.

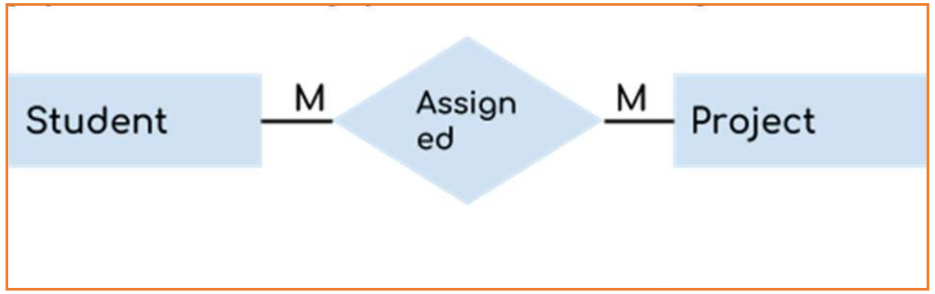
3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.



4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a student can be assigned to many projects and a project can be assigned to many students.



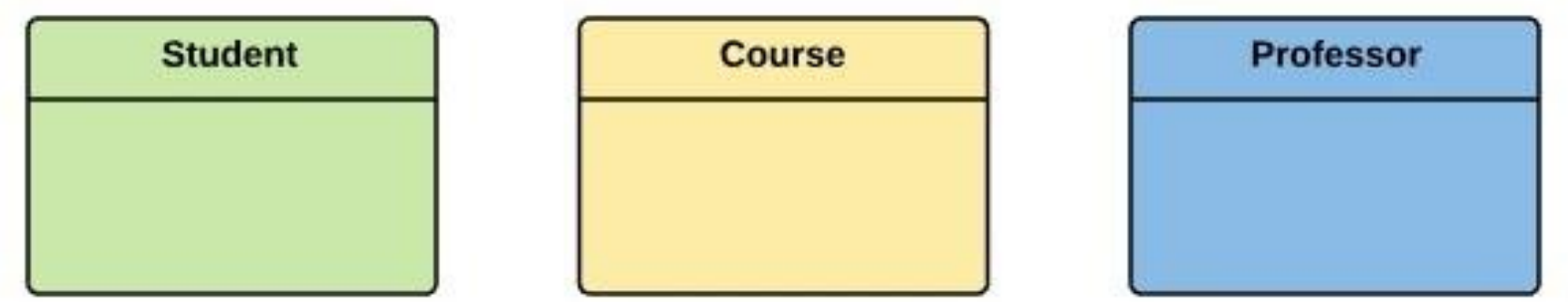
Let's study them with an example:

In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course

Step 1) Entity Identification

We have three entities

- Student
- Course
- Professor



Step 2) Relationship Identification

We have the following two relationships

- The student is **assigned** a course
- Professor **delivers** a course

Step 3) Cardinality Identification

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course

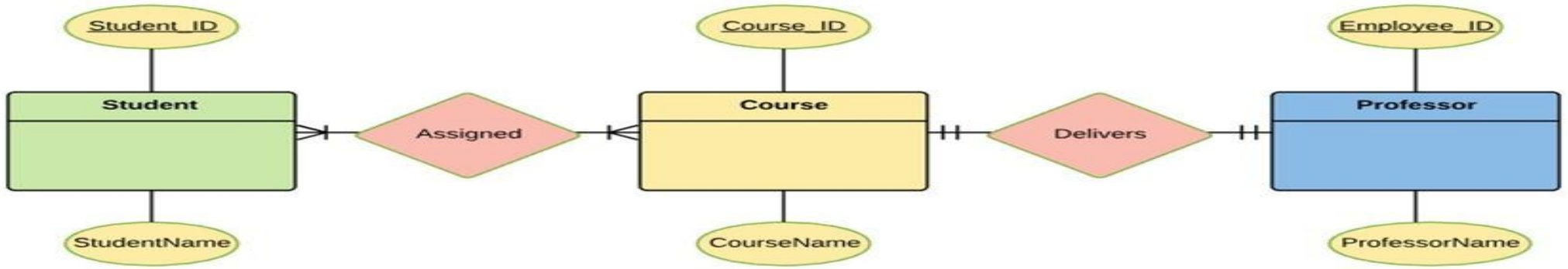
Step 4) Identify Attributes

You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

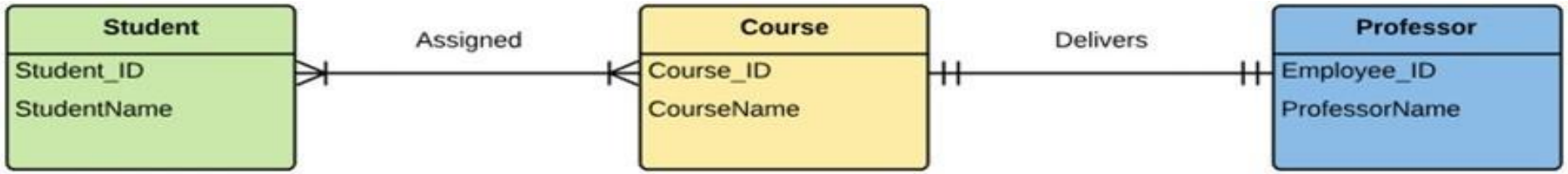
Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Step 5) Create the ERD

A more modern representation of ERD Diagram



Best Practices for Developing Effective ER Diagrams

- Eliminate any redundant entities or relationships
- You need to make sure that all your entities and relationships are properly labeled
- There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
- You should assure that each entity only appears a single time in the ER diagram
- Name every relationship, entity, and attribute are represented on your diagram
- Never connect relationships to each other
- You should use colors to highlight important portions of the ER diagram

Summary

- The ER model is a high-level data model diagram
- ER diagrams are a visual tool which is helpful to represent the ER model
- Entity relationship diagram displays the relationships of entity set stored in a database
- ER diagrams help you to define terms related to entity relationship modeling
- ER model is based on three basic concepts: Entities, Attributes & Relationships
- An entity can be place, person, object, event or a concept, which stores data in the database

New Generation Academy

Transformed for community

Coding Academy

Install and Configure DBMS

Installing a DBMS is more than just running an installer; it requires careful planning of hardware resources and a deep understanding of how the system will manage data.

1. Installation Requirements

Before installation, you must ensure the host system meets the necessary specifications to handle the expected workload.

Hardware & Storage

•**CPU:** The processor handles query execution and data processing. Multi-core processors are preferred for handling concurrent user requests.

•**Storage (Disk Space):** You need space for the **DBMS binaries**, **Data files**, and **Log files**. Solid State Drives (SSDs) are highly recommended over HDDs for faster Read/Write operations.

Memory (RAM)

RAM is the most critical factor for performance. The DBMS uses memory to cache data (Buffer Pool), reducing the need to access the slow physical disk. DBMS often requires minimum RAM (e.g., 4GB for small setups, 16GB+ for enterprise).

Software:

- Operating system compatibility (Windows, Linux (like Ubuntu), macOS).
- Supporting libraries (Java, .NET, or C++ runtime depending on DBMS).
- Network drivers for remote access.

DBMS Features

A DBMS is designed to provide a systematic way to manage data. Its core features include:

- **Data Structuring:** Organizing data into tables, rows, and columns to ensure clarity, consistency, and efficient data management. This includes defining appropriate data types, establishing relationships between datasets, applying keys (such as primary and foreign keys), and ensuring data is normalized to reduce redundancy. Proper data structuring improves data accuracy, simplifies analysis, enhances query performance, and supports scalable data storage and retrieval.
- **Database Customization:**

Allowing administrators to tune and configure the database system to meet specific application and performance requirements. This includes selecting and configuring storage engines, optimizing indexing strategies, adjusting memory and cache settings, defining security rules, and enabling or disabling features as needed. Effective database customization improves performance, scalability, reliability, and resource utilization.

- Data Retrieval:

The ability to efficiently fetch specific records from a database using queries and indexes. This involves optimizing search operations, filtering data based on conditions, sorting results, and minimizing response time. Proper use of indexes and query optimization techniques ensures fast, accurate access to data even in large datasets.

- Query Languages:

Use of standardized languages such as SQL (Structured Query Language) to interact with databases. Query languages enable users to create, read, update, and delete data, define database structures, manage permissions, and perform complex queries involving filtering, sorting, grouping, and joining multiple tables. Standardization ensures consistency, portability, and compatibility across different database systems.

Multi-user Access:
Allowing multiple users to access and manipulate data simultaneously without conflicts through concurrency control mechanisms. This includes managing transactions, locking strategies, isolation levels, and conflict resolution to ensure data consistency, integrity, and reliability while maintaining optimal system performance.

Data Integrity:
Ensuring data remains accurate, valid, and consistent throughout its lifecycle by enforcing constraints within the database. These constraints include rules such as **NOT NULL**, **UNIQUE**, **PRIMARY KEY**, **FOREIGN KEY**, and **CHECK**, which prevent invalid data entry, maintain relationships between tables, and protect the reliability of stored information.

Metadata:
Data about data that describes the structure, organization, and properties of stored information. A DBMS maintains metadata in a **Data Dictionary**, which defines table schemas, column names, data types, constraints, indexes, relationships, and access privileges. This metadata helps users and administrators understand, manage, and maintain the database efficiently while ensuring consistency and control.

DBMS Installation Steps

A **Database Management System (DBMS)** is software that allows users to create, manage, and interact with databases.

Installation Steps:

1.Download the DBMS installer (***e.g., MySQL (https://dev.mysql.com/downloads/mysql/), PostgreSQL (https://www.postgresql.org/download/).***

2.Run the installer and follow setup wizard.

3.Choose installation type (Typical, Developer, Custom).

4.Set root/admin password.

5.Configure networking and security options.

6.Complete installation and verify by connecting to the server.

NB: If you already have **XAMPP installed**, then you **already have a SQL database server included**—specifically **MariaDB**, which is almost the same as MySQL

So no need to install MySQL separately unless:

*- You want a **standalone MySQL server** for specific features or more control.*
*- You want to practice **advanced database administration** outside the web server environment.*
- You want to use tools that specifically require a full MySQL installation.

Create Database and Set Up User Privileges

1. Create a new database.
2. Set its properties (character set, collation, etc.).
3. Create user accounts.
4. Grant privileges to users based on roles (admin, read-only, etc.).

Syntax to CREATE DATABASE (Manage)

```
CREATE DATABASE database_name;
```

Where :
database_name → Name of the new database.

REPLACE
- Some DBMS support CREATE OR REPLACE DATABASE to **replace the existing database** if it exists.
- MariaDB/MySQL typically uses DROP DATABASE IF EXISTS before creating a database.

```
DROP DATABASE IF EXISTS school;
CREATE DATABASE school;
```

IF NOT EXISTS

Prevents an error if the database already exists.

```
CREATE DATABASE IF NOT EXISTS school;
```

When creating a database, you can **specify character sets and collation**:

```
CREATE DATABASE school
CHARACTER SET utf8mb4
COLLATE utf8mb4_general_ci;
```

- **CHARACTER SET** → Determines which characters can be stored (e.g., utf8mb4 supports emojis).
- **COLLATION NAME** → Determines **how text is compared and sorted** (_ci = case-insensitive).

After creating a database, you **grant access to users** using the GRANT command.

```
GRANT ALL PRIVILEGES ON signup_db.* TO 'username'@'localhost' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

*This gives username full control on **all tables** inside signup_db.*

```
GRANT SELECT, INSERT, UPDATE ON signup_db.users TO 'username'@'localhost' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

This allows the user to only read, insert, and update data in the **users** table.

Sample command to create a user and grant privileges:

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'UserPass123';
GRANT ALL PRIVILEGES ON signup_db.* TO 'newuser'@'localhost';
FLUSH PRIVILEGES;
```

Full access to users table

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'UserPass123';
GRANT SELECT, INSERT ON signup_db.users TO 'newuser'@'localhost';
FLUSH PRIVILEGES;
```

limited access to users table

New Generation Academy
Transformed for community

Coding Academy

Learning Out come 3: Manipulate databases

1. INSERT, UPDATE, AND DELETE RECORDS (DML)

DML (Data Manipulation Language) is used to **add, change, and remove data** inside tables.

Main DML commands:

- INSERT → add new records
- UPDATE → modify existing records
- DELETE → remove records

1.1 SQL INSERT STATEMENT

The **INSERT statement** is used to **add new records (rows)** into a table.

Insert Syntax (All Columns)

```
INSERT INTO table_name
VALUES (value1, value2, value3);
```

Example:

```
INSERT INTO students
VALUES (1, 'Jean Paul', 18, 'S6');
```

Insert into Specified Columns

INSERT INTO table_name (column1, column2)
VALUES (value1, value2);

Example:

```
INSERT INTO students (id, full_name)
VALUES (2, 'Aline');
```

Insert Multiple Records

```
INSERT INTO table_name (column1, column2)
VALUES
(value1, value2),
(value3, value4);
```

Example:

```
INSERT INTO teachers (id, name)
VALUES
(1, 'Mukamana'),
(2, 'Niyonzima');
```

Insert Records from Another Table

INSERT INTO new_table (column1, column2)
SELECT column1, column2 FROM old_table;

Example:
INSERT INTO alumni (id, name)
SELECT id, full_name FROM students;

1.2 SQL UPDATE STATEMENT

The **UPDATE statement** is used to **change existing data** in a table.

Update Syntax

```
UPDATE table_name
SET column = value
WHERE condition;
```

Ex: Update a Single Record

```
UPDATE students
SET class = 'S5'
WHERE id = 1;
```

Ex2: Update a Set of Records

```
UPDATE students
SET class = 'S4'
WHERE age < 15;
```

1.3 SQL DELETE STATEMENT

The **DELETE statement** removes records from a table.

Delete Syntax

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

Delete a Single Record

```
DELETE FROM students
WHERE id = 5;
```

Delete Multiple Records

```
DELETE FROM students
WHERE class = 'S1';
```

2. RETRIEVE RECORDS FROM ONE TABLE (SELECT)
2.1 SELECT STATEMENT BASICS

SELECT Syntax

```
SELECT column_name
FROM table_name;
```

The **SELECT statement** retrieves data from a database.

Select Specific Columns

```
SELECT full_name, class
FROM students;
```

Simple Calculation Using SELECT

```
SELECT salary * 12
FROM teachers;
```

Select All Records

```
SELECT *
FROM students;
```

2.2 SELECT Records Based on Condition (WHERE)

WHERE Clause Syntax

```
SELECT column
FROM table
WHERE condition;
```

WHERE Clause Operators

Comparison Operators

= equal
!= not equal
< less than
> greater than
<= less than or equal
>= greater than or equal

Example:
SELECT * FROM students WHERE age >= 18;

New Generation Academy
Transformed for community

Coding Academy

BETWEEN Operator
SELECT * FROM students
WHERE age BETWEEN 15 AND 18;

DISTINCT
Used to remove duplicates.
SELECT DISTINCT class
FROM students;

ALIAS
Used to rename columns temporarily.
SELECT full_name AS name
FROM students;

LIMIT
Used to restrict number of records.
SELECT * FROM students
LIMIT 5;

DESC (Descending)
ORDER BY age DESC;

ASC (Ascending)
ORDER BY full_name ASC;

Logical Operators
AND → all conditions true
OR → one condition true

ORDER BY Clause
Default Order
ORDER BY column; (ASC)

Example:
SELECT * FROM students
WHERE age > 15 AND class = 'S3';

3. RETRIEVE RECORDS FROM DIFFERENT TABLES (JOINS)

A **JOIN** combines records from two or more tables using a related column.

Advantages of Joins

✓ Avoid data duplication
✓ Improve database design
✓ Faster queries

SQL JOIN TYPES

1. INNER JOIN

Returns matching records only.

Eg:

```
SELECT *
FROM students
INNER JOIN classes ON students.class_id = classes.id;
```

2. LEFT JOIN

Returns all records from left table.

- The **LEFT table** = the table written **before** JOIN
- The **RIGHT table** = the table written **after** JOIN

structure:

SELECT …
FROM table_A ← **LEFT table**
JOIN table_B ← **RIGHT table**

3. INNER JOIN

INNER JOIN returns only records that match in BOTH tables.

Syntax

```
SELECT columns
FROM students
INNER JOIN classes
ON students.class_id = classes.class_id;
```

4. LEFT JOIN (LEFT OUTER JOIN)

LEFT JOIN returns ALL records from the LEFT table and matching records from the RIGHT table.

Syntax

```
SELECT columns
FROM students
LEFT JOIN classes
 ON students.class_id = classes.class_id;
```

5. FULL OUTER JOIN

FULL OUTER JOIN returns ALL records from BOTH tables.

Syntax

```
SELECT columns
FROM students
FULL OUTER JOIN classes
ON students.class_id = classes.class_id;
```

1. SQL UNION Operator

The **UNION** operator is used to **combine the result sets of two or more SELECT statements** into a single result set.

Important Rules

•Each SELECT statement must have:

- The **same number of columns**
- **Compatible data types**
- Columns in the **same order**

•**Duplicate rows are removed** by default.

Syntax:

```
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

2. UNION ALL

UNION ALL combines result sets **without removing duplicate records**.

- Faster than UNION (no duplicate check)
- Includes **all rows**, including duplicates

Syntax

```
SELECT column1, column2 FROM table1
UNION ALL
SELECT column1, column2 FROM table2;
```

3. UNION with ORDER BY

When using **ORDER BY** with UNION:

- It must be placed **at the end**
- Applies to the **final combined result**

```
SELECT name FROM students
UNION
SELECT name FROM teachers
ORDER BY name;
```

4. SQL INTERSECT Operator

The **INTERSECT** operator returns **only the rows that are common** to both SELECT statements.

- It Removes duplicates automatically
- The Same column rules as UNION apply

Syntax:

```
SELECT column1 FROM table1
INTERSECT
SELECT column1 FROM table2;
```

5. Manipulating Data Using Nested Queries (Subqueries)

A **subquery** is a query written **inside another SQL query**.

- Also called **inner query**
- The outer query is called the **main query**
- The subquery executes **first**

Syntax of Subqueries

```
SELECT column_name
FROM table_name
WHERE column_name operator ( SELECT column_name FROM table_name WHERE condition);
```

6. Subquery Used with IN / NOT IN

6.1 IN Operator

Returns records where the value **matches any value** from the subquery result.

```
SELECT name FROM students WHERE course_id IN (SELECT course_id  FROM courses WHERE course_name = 'SQL');
```

6.2 NOT IN Operator

Returns records **not matching** the subquery result.

```
SELECT name
FROM students
WHERE course_id NOT IN (
   SELECT course_id
   FROM courses
);
```

7. Subquery with Comparison Operators

Syntax :

```
SELECT name
FROM employees
WHERE salary > (
   SELECT AVG(salary)
   FROM employees
);
```

8. Subquery with EXISTS / NOT EXISTS

```
SELECT name
FROM students s
WHERE EXISTS (
   SELECT 1
   FROM enrollments e
   WHERE s.student_id = e.student_id
);
```

Common Operators

=, >, <, >=, <=

EXISTS Operator

Returns TRUE if the subquery **returns at least one row**.

NOT EXISTS Operator

Returns records where the subquery **returns no rows**.

```
SELECT name
FROM students s
WHERE NOT EXISTS (
   SELECT 1
   FROM enrollments e
   WHERE s.student_id = e.student_id
);
```

Summary Table

Topic	Purpose
UNION	Combine results, remove duplicates
UNION ALL	Combine results, keep duplicates
INTERSECT	Return common records
Subquery	Query inside another query
IN / NOT IN	Match values from subquery
Comparison Operators	Compare with single subquery value
EXISTS / NOT EXISTS	Check presence or absence of records

Examples1

Students_2023

id	name
1	Amit
2	Neha

Students_2024

id	name
3	Rahul
2	Neha

Query

```
SELECT name FROM Students_2023
UNION
SELECT name FROM Students_2024;
```

Output

name
Amit
Neha
Rahul

✓ Duplicate value **Neha** appears only once.

Query 2

```
SELECT name FROM Students_2023
UNION ALL
SELECT name FROM Students_2024;
```

Output

name
Amit
Neha
Rahul
Neha

✓ Duplicate values are **not removed**.

Query 3

```
SELECT name FROM Students_2023
UNION
SELECT name FROM Students_2024
ORDER BY name;
```

Output

name
Amit
Neha
Rahul

Query 4

```
SELECT name FROM Students_2023
INTERSECT
SELECT name FROM Students_2024;
```

name
Neha

✓ Only the **common record** is returned.

Subquery Using IN

Students

student_id	name	course_id
1	Aman	101
2	Riya	102
3	Mohan	101

Courses

course_id	course_name
101	SQL
102	Java

Query

```
SELECT name
FROM Students
WHERE course_id IN (
   SELECT course_id
   FROM Courses
   WHERE course_name = 'SQL'
);
```

Output

name
Aman
Mohan

Query 2

```
SELECT name
FROM Students
WHERE course_id NOT IN (
   SELECT course_id
   FROM Courses
   WHERE course_name = 'SQL'
);
```

name
Riya

Subquery with Comparison Operator

Employees

emp_id	name	salary
1	Raj	40000
2	Seema	60000
3	Anil	50000

Query

```
SELECT name
FROM Employees
WHERE salary > (
   SELECT AVG(salary)
   FROM Employees
);
```

Output

name
Seema

✓ Finds employees earning **more than average salary**.

Subquery with EXISTS

Students

student_id	name
1	Aman
2	Riya

Enrollments

student_id	course
1	Database

Query

```
SELECT name
FROM Students s
WHERE EXISTS (
   SELECT 1
   FROM Enrollments e
   WHERE s.student_id = e.student_id
);
```

name
Aman

New Generation Academy
Transformed for community

Coding Academy

EXCERCIES TO DO IN CLASS

New Generation Academy
Transformed for community

Coding Academy

Senario

The school wants a system to manage:

- Students
- Classes
- Subjects
- Teachers
- Marks
- Payments

Tables

- students(student_id, full_name, class_id)
- classes(class_id, class_name)
- teachers(teacher_id, teacher_name)
- subjects(subject_id, subject_name, teacher_id)
- marks(marks_id, student_id, subject_id, score)
- payments(payment_id, student_id, amount, payment_date)

1.Display all students with their class names.

2. Display only students who are assigned to a class.

3.Display all classes and the students in them, including empty classes.

4.Display the number of students in each class.

5.Display classes that have no students.

6.Display all subjects with the teacher who teaches them.

7.Display teachers who teach at least one subject.

8.Display all teachers including those without subjects.

9.Display subjects that are not assigned to any teacher.

10. Display the total number of subjects per teacher.

11.Display student names, subject names, and scores.

12.Display students who already have marks.