LESSON 1: INTRODUCTION TO DATABASE SYSTEMS

1. DATA

- **Definition:** Raw, unprocessed facts, figures, or symbols without context or meaning.
- **Characteristics:**
  - Unorganized
  - Can be numbers, text, images, or sounds
  - Has no interpretation yet

2. INFORMATION

- **Definition:** Processed or organized data that has meaning, relevance, and context.
- **Characteristics:**
  - Data is arranged or structured
  - Answers "who, what, where, when"
  - Useful for decision-making at a basic level

3. KNOWLEDGE

- **Definition:** Information that has been understood, internalized, and applied based on experience, rules, or insights.
- **Characteristics:**
  - Answers "how" and "why"
  - Enables predictions and decision-making
  - Often tied to human expertise

Database Fundamentals

What is a Database?

- A **structured collection of data** stored electronically.
- It is usually managed by software called a **Database Management System (DBMS)**.

Importance in Organizations

Databases are crucial for:

- **Efficiency:** They allow for quick storage, retrieval, and updating of data.
- **Reliability:** They ensure data is stored consistently and accurately.
- **Scalability:** They can handle growing amounts of data as an organization expands.

Common Examples of Database Use:

- Banking Systems

- Inventory Management
- Hospital Records
- School Systems

LESSON 2: DATABASE MODELS

What is a Database Model?

A **Database Model** is the framework that determines how data is **stored, organized, and manipulated** within a database.

Why do database models matter?

- They organize complex data.
- They improve efficiency and reliability.
- They provide a structure for queries and updates.

Types of Database Models

1. Hierarchical Model

- **What it is:** Data is organized in a **tree-like structure** with parent-child relationships. One parent can have many children, but each child has only one parent.
- **Pros:** Simple and efficient for 1-to-many relationships.
- **Cons:** Very rigid and hard to update.

2. Network Model

- **What it is:** Uses a **graph structure** to represent relationships. This is more flexible than the hierarchical model, allowing for multiple connections between data records (a "many-to-many" relationship).
- **Pros:** Handles complex relationships well.
- **Cons:** Can be very complex and hard to maintain.

3. Relational Model

- **What it is:** This is the most common model. It represents data in **tables (relations)** with **rows (records)** and **columns (attributes)**. Relationships are defined by common values (keys) shared between tables.
- **Pros:** Easy to use, flexible, strong support for **SQL** (Structured Query Language), and scalable.
- **Cons:** Can have performance issues at a very large scale.

4. Object-Oriented Model

- **What it is:** Stores data as **"objects"** (like in object-oriented programming). An object includes both the data (attributes) and the methods (functions) that operate on that data.

- **Pros:** Great for complex applications and handling multimedia (images, videos, etc.).
- **Cons:** Less widespread and can be more complex than the relational model.

LESSON 3: DBMS AND ITS FUNCTIONS

What is a Database Management System (DBMS)?

A **Database Management System (DBMS)** is a software that allows users to **create, access, manage, and control** databases efficiently. Think of it as the "operating system" for a database.

Key Functions of a DBMS

A DBMS has several critical functions to manage data properly:

- **Data Definition:** Allows users to define the database's structure.
- **Data Storage, Retrieval, and Update:**
    - Stores large volumes of data systematically.
    - Provides efficient retrieval of data using queries.
    - Allows records to be updated without affecting other data.
- **Data Security and Integrity:**
    - **Security:** Ensures that only authorized users can access or modify the data.
    - **Integrity:** Maintains the accuracy and consistency of the data.
- **Data Administration:** Provides tools for monitoring performance, as well as for data backup and recovery.
- **Multi-user Access & Concurrency Control:**
    - Supports multiple users working at the same time.
    - Prevents conflicts when two or more users try to update the same record simultaneously.
- **Data Independence:**
    - Separates the logical design (how the user sees the data) from the physical storage (how or where the data is actually stored).
    - This means users don't need to know the physical storage details to access the data.
- **Backup & Recovery:** Ensures that data is recoverable after a system failure.

Real-World Examples

You can find DBMS applications in many large-scale systems, including:

- Banking Systems
- E-Commerce
- University Enrollment Systems

LESSON 4: RELATIONAL MODEL

What is the Relational Model?

The **Relational Model** represents how data is stored and managed in a Relational Database. It organizes data into tables, where each table is known as a **"relation"**.

- **Origin:** Introduced by E.F. Codd of IBM in 1970.
- **Basis:** It is based on mathematical set theory.
- **Structure:** Each table consists of rows (tuples) and columns (attributes), and the relationships between tables are well-defined.
- **Foundation:** It is the foundation for all modern **Relational Database Management Systems (RDBMS)**.

Relational Model Concepts

- **Relation (Table):** A two-dimensional table used to store a collection of data.
- **Tuple (Row):** A single row in a table. It represents one complete record.
- **Attribute (Column):** A column in a table that contains values for a specific data point.
- **Relation Schema:** The metadata or "blueprint" that defines the structure of a table (e.g., `Customer (Customer_ID, Name, Address)`).
- **Domain:** The set of all possible, valid, and atomic (indivisible) values for an attribute (e.g., the domain for a "Month" attribute would be January, February, etc.).
- **Degree:** The number of **columns** (attributes) in a table.
- **Cardinality:** The number of **rows** (tuples) in a table.
- **Primary Key:** A column (or set of columns) that **uniquely identifies** each row in a table.
- **Foreign Key:** A column in one table that **links to the primary key** of another table, which establishes a relationship between them.

Relational Integrity Constraints

These are rules that maintain the **accuracy, consistency, and reliability** of data in a database.

- **Domain Constraints:** Restricts the values an attribute can have to its defined domain (e.g., enforcing data types like `INTEGER` or `DATE`).
- **Key Constraints:** Ensures that every tuple (row) in a relation is unique, which is enforced by the primary key.
- **Entity Integrity Constraints:** A crucial rule stating that the **primary key of a table cannot have a NULL value**.
- **Referential Integrity Constraints:** Ensures that a foreign key value in one table **must have a matching primary key value** in the table it references. This prevents "orphaned" records.

Operations in the Relational Model

There are two main types of operations:

1. Data Manipulation Language (DML)

This is part of SQL and is used for managing existing data *within* the tables.

- **INSERT:** Adds a new row (tuple) to a table.
- **UPDATE:** Modifies the values in one or more existing rows.
- **DELETE:** Removes one or more rows from a table.
- **SELECT:** Retrieves a specific range of data based on certain criteria.

2. Relational Algebra

This is a *procedural query language* based on a set of mathematical operations.

- **Selection (σ):** Filters and selects *rows* based on a condition (a horizontal subset).
- **Projection (π):** Selects specific *columns* from a table (a vertical subset).
- **Union (∪):** Combines the rows of two tables, removing duplicates.
- **Set Difference (-):** Returns rows from the first table that are *not* in the second table.
- **Cartesian Product (×):** Combines every row from the first table with every row from the second table.
- **Join (⋈):** Combines related rows from two or more tables based on a common attribute (like a foreign key).

Advantages and Disadvantages

Advantages

- **Simple Model:** It is simple and easy to use.
- **Flexible:** More flexible than other models.
- **Secure:** It is a very secure model.
- **Data Accuracy:** Data is more accurate.
- **Data Integrity:** The integrity of the data is well-maintained through constraints.
- **Operations:** Operations are easy to apply.

Disadvantages

- **Performance:** Can have performance issues with *very large* databases.
- **Complexity:** Struggles with hierarchical or complex data (where Graph or Document models might be better).
- **Normalization Overhead:** Extensive normalization (splitting tables) can lead to complex queries that are slower to run.

Best Practices for Creating a Relational Model

1. **Understand Business Requirements:** Before you start, clarify what data is needed and how it will be used.
2. **Follow Normalization Principles:** Minimize data redundancy (repetition) by normalizing tables, usually to at least the Third Normal Form (3NF).
3. **Define Clear Naming Conventions:** Use consistent and descriptive names for tables and columns.
4. **Use Appropriate Data Types:** Select the correct data type for each column (e.g., `INTEGER`, `DATE`, `TEXT`) to ensure accuracy and save storage.
5. **Utilize Constraints:** Enforce data integrity by properly defining primary keys, foreign keys, and other constraints.
6. **Optimize with Indexing:** Add indexes to columns that are frequently used in queries to speed up data retrieval.
7. **Document the Design:** Clearly document the schema, relationships, and business rules for future maintenance.
8. **Plan for Scalability:** Consider future growth by planning strategies like partitioning or sharding data.

LESSON 5: ENTITY - RELATIONSHIP MODELING

What is an Entity-Relationship (ER) Model?

The **Entity-Relationship Model (ERM)** is a diagram that illustrates how data is related within a database. It's a conceptual blueprint that uses three main components:

- Entities
- Attributes
- Relationships

Why use an ER Model?

- It serves as a **blueprint** before building the actual database.
- It helps **plan and organize** the database's structure.
- It clearly **shows how data is connected**.
- It helps **prevent errors** and **reduces data redundancy** (unnecessary repetition).
- It makes the database design **easier to understand**.

Core Component: Entities

An **Entity** is any real-world object, concept, or thing that can be clearly identified and stored in a database (e.g., a `Person`, `Place`, or `Product`).

- **Entity Type:** This is the *definition* or *collection* of entities that share the same properties or attributes. For example, `Student` is an entity type.
- **Entity Set:** This is the *actual collection* of all entities (all the rows) of a specific type that are stored in the database.

Kinds of Entities

There are two main kinds of entities: Independent (Strong) and Dependent (Weak).

The Tip: To tell the difference, ask this one question: **"Can I add a row to this table without the need for another table?"**

- If **YES** It's an **Independent (Strong) Entity**.
- If **NO** It's a **Dependent (Weak) Entity**.

1. Independent (Strong) Entity

- This entity is the **backbone** of the database and can **exist on its own**.
- Its primary key is independent and is *not* a foreign key.
- **Example:** A PRODUCTS table with (ID(PK), NAME, PRICE, DETAILS) can have a row added at any time.

2. Dependent (Weak) Entity

- This entity **cannot exist without** being associated with a strong entity.
- Its existence *depends* on another entity.
- It uses a **foreign key** to identify the strong entity it's associated with.
- It is often used to connect multiple strong entities.
- **Example:** A TRANSACTIONS table with (TRANSAC_ID(PK), NAME, TAX(FK) (NOT NULL)) is dependent because it *cannot* be created without a valid TAX(FK).
- **Best Practice for Primary Key:** The industry standard for creating a primary key for a dependent entity is to simply add a new, single, non-meaningful **surrogate key**.
- **ERD Representation:** A dependent entity is shown with a **dashed line**, while an independent entity uses a **solid line**.

Core Component: Attributes

An **Attribute** is a **property or characteristic** of an entity. It's used to describe the entity and represents an individual piece of data. In a database table, an attribute is a **column or field**.

Types of Attributes

1. **Simple Attribute:** A single-valued attribute that cannot be broken down into smaller parts (it's "atomic").
   - **Example:** Age, Gender.
   - **Symbol:** A single oval.
2. **Composite Attribute:** An attribute that can be broken down into multiple, smaller attributes.
   - **Example:** Address can be broken down into *Street*, *City*, and *Region*.
   - **Symbol:** An oval with smaller ovals branching out of it.
3. **Derived Attribute:** An attribute whose value is **calculated** from other attributes. It is *not* stored directly in the database.

    - **Example:** Age is derived by calculating from a stored Birthdate.
    - **Symbol:** A **dashed oval**.
4. **Multivalued Attribute:** An attribute that can have a **set of values** for a single entity.
    - **Example:** A STUDENT can have multiple PhoneNumbers or Email addresses.
    - **Symbol:** A **double oval**.
5. **Key Attribute:** An attribute that **uniquely identifies** each entity instance (row).
    - **Example:** StudentID.
    - **Symbol:** An oval with the attribute name **underlined**.
6. **Stored Attribute:** An attribute whose value is **stored directly** in the database. This is the opposite of a derived attribute.
    - **Example:** Birthdate.
7. **NULL Attribute:** An attribute that **may not have a value**. It can be empty or unknown because the data is optional or not applicable.
    - **Example:** MiddleName (NULL) or Nickname (NULL).

NULL and Keys

What is NULL?

- In a database, NULL represents a value that is **missing, unknown, or inapplicable**.
- It is **NOT** the same as zero (0), an empty string (""), or the value false.
- It simply means "we don't know the value (yet)". It is an absence of data, not an absence of meaning.

Types of Keys

Keys are special fields used to **identify records** and **establish relationships** between tables.

1. **Primary Key (PK):** The main, unique identifier for each record in a table. It **must be UNIQUE** and **CANNOT be NULL**.
2. **Candidate Key:** *Any* attribute (or set of attributes) that *could* be used to uniquely identify a record. There can be multiple candidate keys in one table. It also **cannot be NULL**.
3. **Alternate Key:** A candidate key that was *not* chosen to be the primary key. It acts as a backup unique identifier.
4. **Foreign Key (FK):** A column in one table that **references the Primary Key** of another table. This is what **establishes the relationship**. A foreign key **CAN be NULL**.
5. **Composite Key (Compound Key):** A key that uses **two or more columns together** to uniquely identify a record. This is used when no single column is unique.
6. **Super Key:** *Any* set of attributes that can uniquely identify a record, even if it includes extra, unnecessary attributes. (e.g., (StudentID, Name) is a super key, but StudentID is the minimal candidate key).
7. **Unique Key:** A key that ensures all values in its column are unique, but with one difference from a PK: it **CAN contain NULL values**. This is used for fields that are optional but must be unique if a value *is* entered.

Core Component: Relationships

Relationships are the **"glue"** that holds tables together by connecting related information.

- **Relationship:** The connection itself (e.g., "A Student *enrolls in* a Class").
- **Cardinality:** The **MAXIMUM** number of entities that can be in the relationship (e.g., *One* or *Many*).
- **Modality:** The **MINIMUM** number of entities (e.g., *Zero* or *One*). This tells you if the relationship is **optional** (Zero) or **mandatory** (One).

Types of Relationships

1. **One-to-One (1:1):** One instance of an entity is associated with *exactly one* instance of another.
   - **Example:** One StudentNumber is issued to only One Student.
2. **One-to-Many (1:N):** A single instance of one entity can be linked to *multiple* instances of another.
   - **Example:** One Customer can place Many Orders.
3. **Many-to-Many (N:M):** Multiple instances of one entity can relate to *multiple* instances of another.
   - **Example:** Many Students can develop Many Projects.
   - **Note:** In a physical database, this requires a third table, called a **Junction Table**, in the middle.
4. **Recursive (Unary) Relationship:** An entity has a relationship *with itself*.
   - **Example:** In an Employee table, one employee *supervises* another employee. The SupervisorID would point back to the EmployeeID in the same table.
5. **Ternary (N-ary) Relationship:** A single relationship that involves **more than two entities**.
   - **Example:** A Prescription entity connects a Doctor, a Patient, and a Medication.

ERD Symbols (MySQL Workbench)

- **ER Model:** The conceptual, logical representation of the database.
- **ER Diagram (ERD):** The physical, graphical tool used to visualize the model.

In MySQL Workbench, you'll see these symbols:

- **Attributes (Nullability):**
  - **Solid Diamond:** The attribute is **NOT NULL** (mandatory).
  - **Not-Filled Diamond:** The attribute **can be NULL** (optional).
- **Keys (Colors):**
  - **Blue:** Simple Key.
  - **Red:** Foreign Key.

LESSON 6: BASIC SQL

SQL (Structured Query Language) is a standard language used to manage and manipulate relational databases. It allows users to store, retrieve, update, and delete data within a database.

SQL commands are typically divided into several main categories:

- **Data Definition Language (DDL)**
- **Data Manipulation Language (DML)**
- **Data Query Language (DQL)**
- **Data Control Language (DCL)**

Data Definition Language (DDL)

DDL comprises a set of SQL commands used to **define, modify, and delete the structure** of database objects. These commands are essential for managing the database's **schema**, which includes tables, views, indexes, and other components.

The main DDL commands are `CREATE`, `ALTER`, `RENAME`, `COMMENT`, `TRUNCATE`, and `DROP`.

CREATE

The `CREATE` command is used to build new database objects, such as databases, tables, views, or indexes. It defines the structure of the object, including column names, data types, and constraints.

For example, if you run this command:

```
CREATE TABLE Students ( StudentID INT, LastName VARCHAR(255), FirstName VARCHAR(255) );
```

You create this empty table structure:

StudentID	LastName	FirstName

Types of `CREATE` statements include:

- `CREATE DATABASE`
- `CREATE TABLE`
- `CREATE VIEW`

- CREATE INDEX
- CREATE USER
- CREATE PROCEDURE
- CREATE TRIGGER
- CREATE SCHEMA

ALTER (ALTER TABLE)

The ALTER TABLE command modifies the structure of an existing table without losing the data already stored in it.

Common uses for ALTER include:

- **ADD:** Used to add a new column to a table. You must specify the column's name and data type.
- **MODIFY:** Used to change the definition of an existing column, such as its data type or size.
- **DROP:** Used to remove a column from a table, which permanently removes the column and all its data.
- **RENAME:** Used to change the name of an existing column while keeping its data type and content.

For example, starting with our Students table:

StudentID	**LastName**	**FirstName**

If you run the ADD command:

ALTER TABLE Students ADD Email VARCHAR(255);

The table structure changes to this:

StudentID	**LastName**	**FirstName**	**Email**

RENAME

This command is used to change the name of an existing table in the database. This is useful for restructuring or updating table names to better reflect their content without losing data.

COMMENT

Comments are annotations in SQL code that are **not executed** by the database engine. They serve as notes for human readers to make the code easier to understand and maintain.

There are three types of comments:

1. **Single-line:** Starts with two hyphens (`--`) and continues to the end of the line.
2. **Multi-line:** Enclosed between `/*` and `*/` and can span multiple lines.
3. **In-line:** Used *within* a query, also enclosed between `/*` and `*/`.

DROP

The `DROP` command is used to **permanently remove an object** from a database, such as a table, database, index, or view. When a table is dropped, all its data and its structure are permanently removed, and this action cannot be undone without a backup.

- **Foreign Key (FK) Situation:** You cannot `DROP` a table if it is being referenced by a foreign key constraint in another table. This is a safety feature to protect data consistency.
- **How to Fix:**
    - **Option 1:** Drop the "child" table (the one containing the foreign key) *first*, then drop the "parent" table.
    - **Option 2:** Remove the foreign key constraint from the child table using `ALTER TABLE ... DROP FOREIGN KEY`, and then drop the parent table.

TRUNCATE

This command removes **all rows** from a table but preserves the table's structure for future use. While similar to a `DELETE` command, `TRUNCATE` is much faster because it bypasses certain integrity checks.

For example, if your table looked like this:

StudentID	**LastName**	**FirstName**
101	Smith	Jane

102	Doe	John

After running TRUNCATE TABLE Students;, the table would be empty, but the structure remains:

StudentID	LastName	FirstName

Data Manipulation Language (DML)

DML comprises a set of SQL commands used to **manage and manipulate the actual data** stored in database tables.

The main DML commands are SELECT, INSERT, UPDATE, and DELETE.

SELECT

The SELECT statement is used to retrieve data from one or more tables. It can return all records or specific results based on conditions you set. The output is returned in a tabular format called a result set.

For example, using a table with data:

```
SELECT FirstName, LastName FROM Students WHERE StudentID = 101;
```

You would see this as the result:

FirstName	LastName
Jane	Smith

INSERT

The INSERT statement is used to add new rows (records) to an existing table. It can be used to insert values into all columns or only specific columns.

If you run this command on our empty Students table:

```
INSERT INTO Students (StudentID, LastName, FirstName) VALUES (101, 'Smith', 'Jane');
```

The table now contains that data:

StudentID	LastName	FirstName
101	Smith	Jane

UPDATE

The UPDATE statement is used to modify existing records in a table. It allows you to update one or multiple columns to keep data accurate and consistent.

Let's start with this table:

StudentID	LastName	FirstName
101	Smith	Jane

If you run this command:

```
UPDATE Students SET LastName = 'Jones' WHERE StudentID = 101;
```

The table is modified and now looks like this:

StudentID	LastName	FirstName
101	Jones	Jane

-

  Note on Safe Updates: When MySQL is in "safe update mode," it may return **Error 1175** if you try to run an `UPDATE` or `DELETE` command without a `WHERE` clause that uses a key. To proceed with the operation, you may need to temporarily disable this feature by running `SET SQL_SAFE_UPDATES = 0;`.

DELETE

The `DELETE` statement is used to remove specific rows from a table. It keeps the table structure intact, which makes it different from the `DROP` command.

Let's start with this table:

StudentID	LastName	FirstName
101	Jones	Jane
102	Doe	John

If you run this command:

```
DELETE FROM Students WHERE StudentID = 102;
```

The table now looks like this:

StudentID	LastName	FirstName
101	Jones	Jane

Differences Between DDL and DML

DDL (Data Definition Language)

- Used to define database objects (tables, views, etc.).
- Changes affect the **structure** of the database.
- Statements are not transactional (meaning they cannot be rolled back).

- Typically used during the **design and setup phase** of a database.

DML (Data Manipulation Language)

- Used to manipulate **data** within the database.
- Changes affect the **data stored** in the database.
- Statements are executed by developers or end-users.
- Used during the **normal operation** of a database.