CHAPTER II: RELATIONAL MODEL

1. Definition of the Relational Model

The relational model, introduced by Edgar Codd in 1970, is based on the mathematical set theory. Codd proposed using this universally known theory to formalize a database (DB) and the operations that can be applied to it.

2. Basic Concepts

2.1. Attribute

- **Definition**: An **attribute** is a characteristic or property that describes an **entity** in a database. Each attribute has a **name** and an associated **value**.
- **Example**: In a **student database**, an entity "Student" could have attributes such as **"Name", "Age", "Identification Number"**, etc.

2.2. Tuple

- **Definition**: A **tuple** is an **ordered collection of attributes** representing a **specific instance** of an entity in a **relational database**.
- **Example**: For the entity "Student," a specific tuple could be **(Name: "Farouk", First Name: "Samir", Age: 22, ID: 12345)**.

2.3. Domain

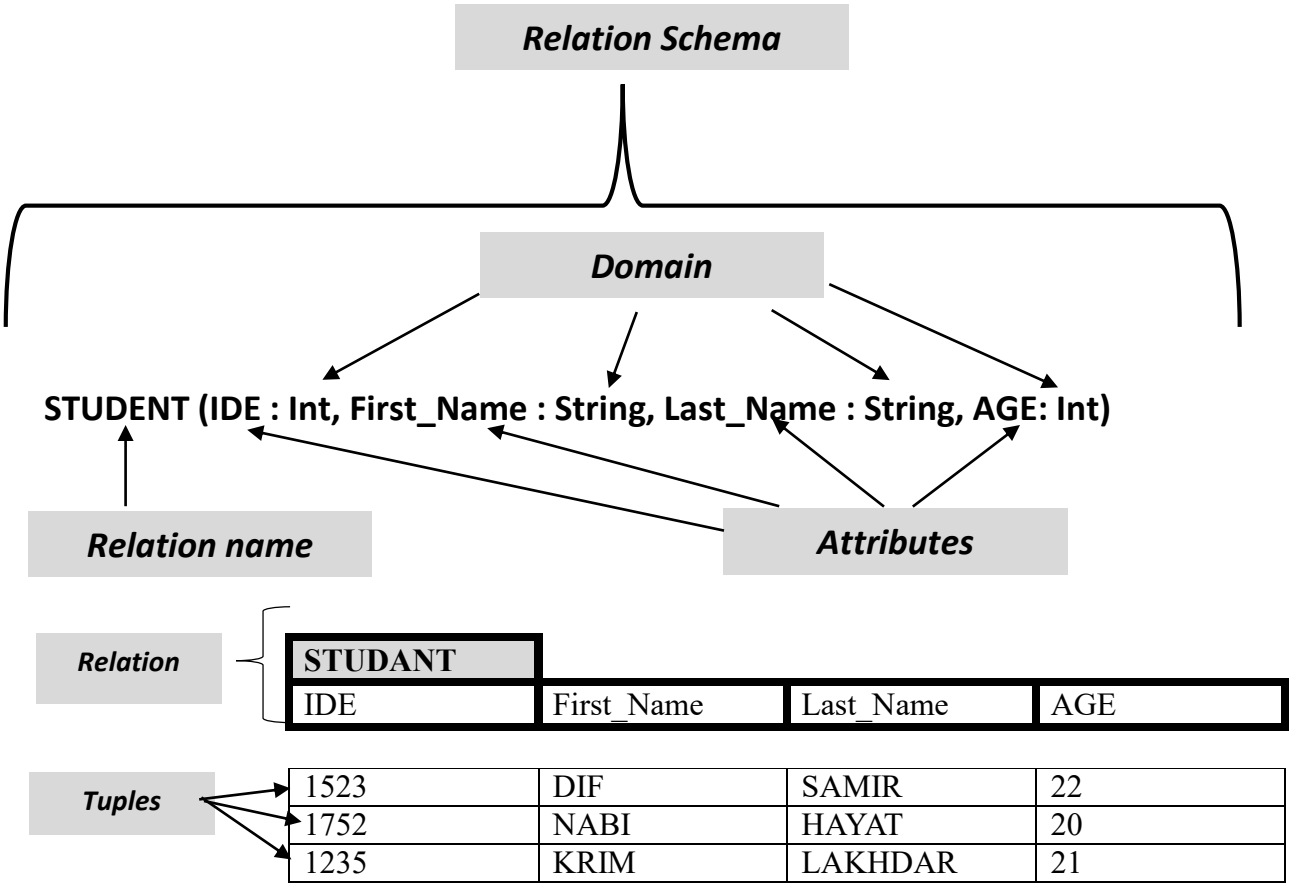
- **Definition**: A **domain** is the **set of allowed values** for a particular attribute in a database. It defines the possible range of values for an attribute.
- **Example**: If the attribute "Age" has a domain defined between **18 and 30 years**, then all age values for students must fall within this range.

2.4. Relation

- **Definition**: A **relation** is a **table** in a relational database. It consists of **tuples**, where each row represents a **specific record**, and each column represents an **attribute**.
- **Example**: In a **school database**, the relation "Student" could be a **table** where each tuple represents a specific student with attributes such as **name, age, and ID**.

3. Schema of a Relation

The **schema of a relation** consists of the **name of the relation**, followed by the **list of attributes** and the **definition of their domains**.



4. Normalization

Database normalization is a design process aimed at efficiently organizing data by eliminating redundancies and optimizing table structures. The primary objective of normalization is to improve the quality, integrity, and performance of relational databases. This process follows a set of rules called normal forms, such as **1NF, 2NF, 3NF, and BCNF**, each addressing specific redundancy and data integrity issues.

Some key benefits of database normalization include:

- **Reduction of data redundancy**: It eliminates unnecessary duplications by properly distributing information across different tables, saving storage space and preventing inconsistencies.
- **Improved data integrity**: By reducing redundancy, normalization helps maintain data integrity by avoiding inconsistencies and anomalies that might arise in non-normalized structures.
- **Easier updates and modifications**: Changes are easier to manage since they only need to be applied in one table instead of multiple locations.
- **Performance optimization**: Queries can be more efficient in a well-normalized database, as they leverage well-defined relationships between tables.
- **Better data structure understanding**: Organizing data logically through normalization simplifies database schema comprehension and maintenance.

5. Functional Dependency and Keys (Primary & Foreign Keys)

5.1. Functional Dependency

The concept of **functional dependency** was introduced by Edgar Codd at the inception of the relational model to characterize relationships that can be decomposed without losing information.

A functional dependency is defined as follows:
Given a relation schema $\mathbf{R(A_1, A_2, ..., A_n)}$, and subsets **X** and **Y** of $\{A_1, A_2, ..., A_n\}$, we say that **X** → **Y** (X determines Y or Y is functionally dependent on X) if, for a given value of X, there exists a unique value of Y at any given time.

Formally:
For any extension **r** of **R**, and for all tuples $\mathbf{t_1}$ and $\mathbf{t_2}$ in **r**, we have:
$\mathbf{\Pi X(t_1) = \Pi X(t_2) \Rightarrow \Pi Y(t_1) = \Pi Y(t_2)}$

5.1.1. Properties of Functional Dependencies (Armstrong's Axioms)

Functional dependencies follow several inference rules, known as **Armstrong's axioms**:

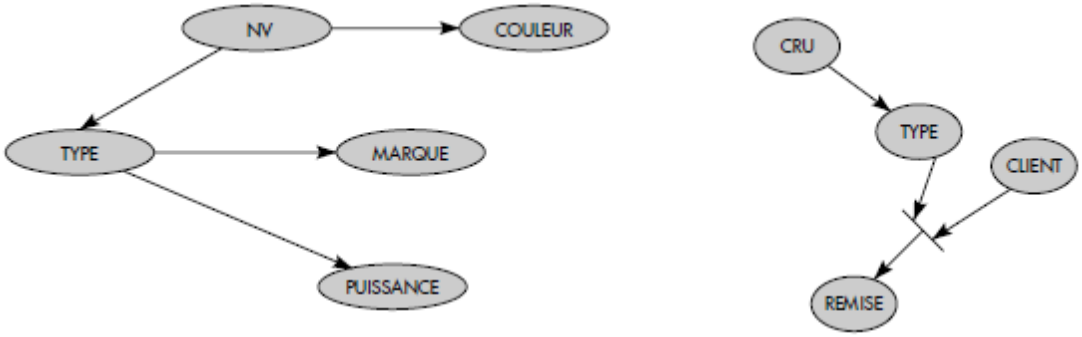
- **Reflexivity**: If Y is a subset of X, then X → Y.
- **Augmentation**: If X → Y, then XZ → YZ.
- **Transitivity**: If X → Y and Y → Z, then X → Z.
- **Union**: If X → Y and X → Z, then X → YZ.
- **Pseudo-transitivity**: If X → Y and WY → Z, then WX → Z.
- **Decomposition**: If X → Y and Z is a subset of Y, then X → Z.

5.1.2. Elementary Functional Dependency

A **functional dependency X → A** is elementary if **A** is a single attribute **not belonging to X**, and there is no **X' ⊂ X** such that **X' → A**.

5.1.3. Functional Dependency Graph

A **functional dependency graph** visually represents dependencies between attributes. Each node represents an attribute set, while directed edges indicate functional dependencies, providing a clearer understanding of relationships between attributes in a database schema.

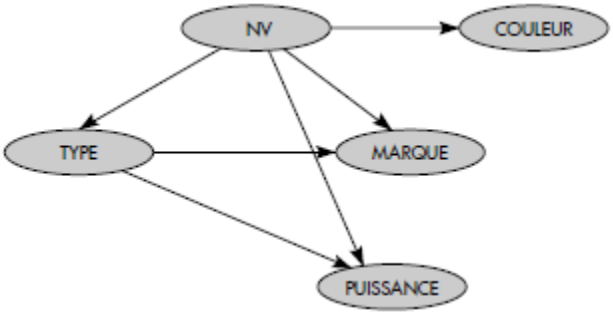


5.1.4. Transitive Closure and Minimal Cover

The **transitive closure** of a set of functional dependencies (FDs) is the set of all functional dependencies derived from the initial set using transitivity rules.

The **minimal cover** of a set of functional dependencies is an equivalent set that has the same transitive closure but is simplified by eliminating redundant and non-essential functional dependencies.

Example: The transitive closure of the previous example is as follows:



5.2. Relation Key

Relation keys, such as the **primary key** and **foreign key**, are fundamental concepts in relational database modeling. They are used to establish links between different tables and ensure data integrity. A more formal definition can be given based on **functional dependency**, as follows:

A subset **X** of attributes in a relation **R (A_1, A_2, ..., A_n)** such that:

1. $X \rightarrow A_1, A_2, \ldots, A_n$
2. There is no subset $Y \subseteq X$ such that $Y \rightarrow A_1, A_2, \ldots, A_n$

5.3. Primary Key

- A **primary key** is a column (or a set of columns) that uniquely identifies each record in a table.
- It ensures **data uniqueness** within the table and serves as a reference point for establishing relationships with other tables.
- The primary key **cannot contain NULL values**, and each table can have only **one primary key**.

5.4. Foreign Key

- A **foreign key** is a column (or a set of columns) in a table that references the **primary key** of another table.
- It establishes a relationship between two tables, allowing information from one table to be linked to another.
- Foreign keys **ensure referential integrity**, meaning that every foreign key value must correspond to an existing value in the primary key of the referenced table.

6. Integrity Constraints

Integrity constraints are rules that must be verified by the data stored in a database. They ensure the **validity and consistency** of stored data. These constraints are essential for maintaining the **quality and integrity** of information within a database.

6.1. Domain Integrity

The values stored in a column must comply with the **data type** defined for that column. For example, a numeric column should contain only numbers.

- In some cases, **advanced domain integrity** is enforced, which includes additional constraints such as **allowed value ranges**, **regular expressions**, etc.

6.2. Key Integrity (Entity Integrity)

Each record in a table must be **uniquely identifiable** by a **primary key**.

- This ensures that no records are duplicated.
- It **implicitly** means that **primary key values must be unique and non-NULL**.

6.3. Referential Integrity

Foreign keys are used to link two tables. The **referential integrity rule** states that:

- A value in a **foreign key** must correspond to an **existing value** in the **primary key** of the referenced table.

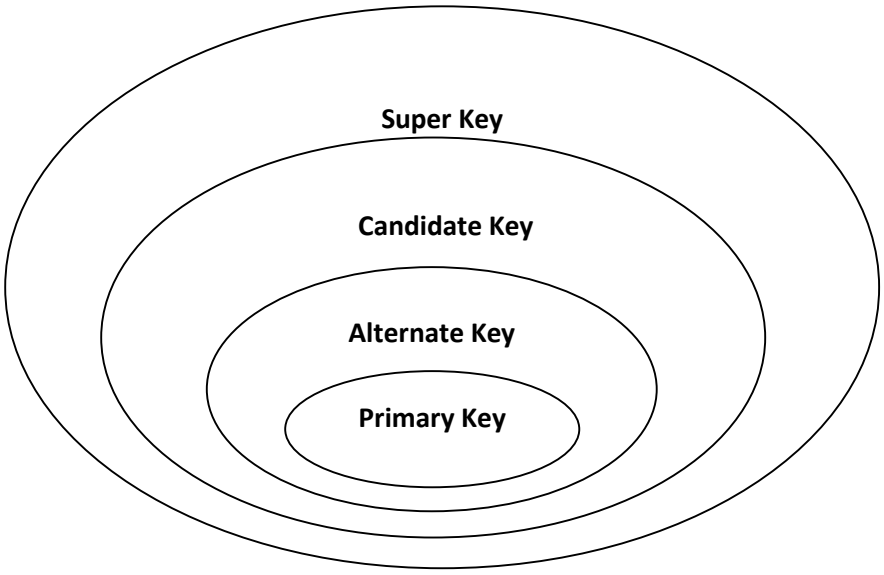
7. Normal Forms

Normal forms are **design rules** in relational databases that define the **minimal and optimal** structure of a database.

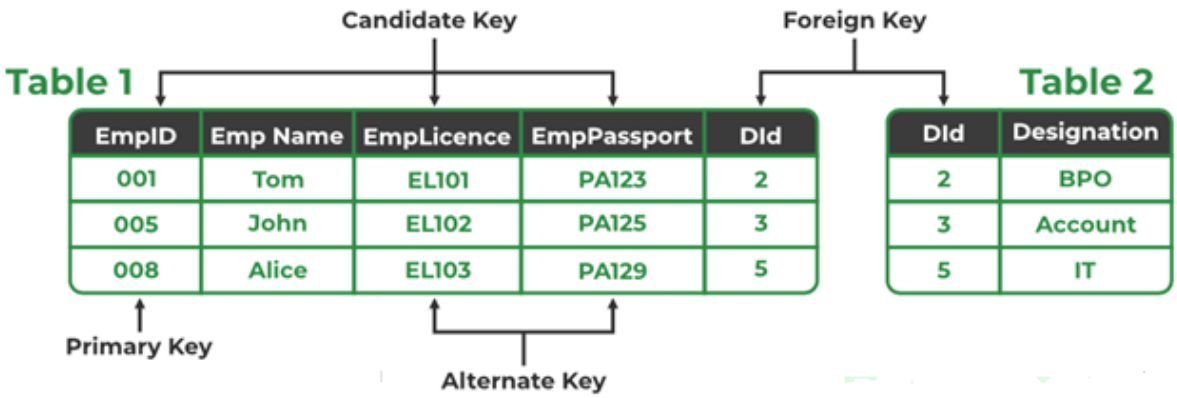
- Their goal is to **eliminate data redundancy** and **prevent update anomalies**.
- Each normal form establishes specific criteria for structuring tables, focusing on how data is **organized** and how **functional dependencies** exist between attributes.

Before discussing normal forms, here is the typology of keys in the relational model:

1. **Super Key**: A set of attributes that, when taken together, can uniquely identify a row in a table.
2. **Candidate Key**: A **minimal super key**, meaning a super key without unnecessary attributes that **always guarantees uniqueness**. It is a candidate to become the primary key.
3. **Primary Key**: A **chosen candidate key** that serves as the main identifier of each row in a table. It must be **minimal** and **cannot contain NULL values**.
4. **Alternate Key**: A **candidate key** that was **not selected** as the primary key.
5. **Composite Key**: A **primary key** or **candidate key** consisting of **two or more attributes**.
6. **Foreign Key**: A set of attributes in a table that **references the primary key** of another table, establishing a **relationship** between tables.



Example:



Moreover, **EmpID, EmpLicence, EmpPassport, {EmpID, Emp_Name}, {EmpID, EmpLicence}, {EmpLicence, EmpPassport}**, and so on constitute the **set of super keys** for this relation.

7.1. First Normal Form (1NF)

A relation is in **First Normal Form (1NF)** if **every attribute contains an atomic (indivisible) value and does not have multiple values**.

This normal form is justified by **simplicity and aesthetics**. It ensures that each column holds a **single value** rather than sets or lists of values.

Example:

ID_EMP	Name	Profession
1	Soltani Rabeh	Ingénieur, Expert
2	Selami Younes	Professeur, Administrateur

This relation must be decomposed as follows:

ID_EMP	F_Name	L_Name
1	Soltani	Rabeh
2	Selami	Younes

ID_Prof	Designation
1	Ingénieur
2	Expert
3	Professeur
4	Administrateur

ID_EMP	ID_Prof
1	1
1	2
2	3
2	4

7.2. Second Normal Form (2NF)

A relation R is in the Second Normal Form if and only if:

1. It is in the First Normal Form.
2. Every non-key attribute does not depend on a part of a composite key.

The verification of this normal form applies only to relations with composite primary keys.

Example:

Consider the relation:
Project (project_number, employee_number, role, employee_name)

This relation is not in 2NF because **employee_number → employee_name**, which leads to two problems:

- **Problem 1:** An employee is only recorded if they are part of a project.
- **Problem 2:** Redundancy occurs if an employee is involved in multiple projects.

To resolve these issues, we must decompose this relation as follows:

- **Project (project_number, employee_number, role)**
- **Employee (employee_number, employee_name)**

7.3. Third Normal Form (3NF)

A relation R is in the Third Normal Form if and only if:

1. It is in the Second Normal Form.
2. Every non-key attribute does not depend on another non-key attribute.

Example:

Consider the relation:
CAR (Car_Number, Brand, Model, Power, Color)

This relation is not in the Third Normal Form because the non-key attribute **Model** determines **Brand** and **Power**.
To eliminate this issue, we decompose the relation as follows:

- **CAR (Car_Number, Model, Color)**
- **MODEL (Model, Brand, Power)**

7.4. Boyce-Codd Normal Form (BCNF)

- The **2nd Normal Form (2NF)** eliminates anomalies caused by dependencies between parts of a key and non-key attributes.
- The **3rd Normal Form (3NF)** eliminates anomalies caused by dependencies between non-key attributes.
- However, what about dependencies where parts of a key depend on each other or when a non-key attribute determines part of a key? The **3NF is not sufficient** in such cases.

To address redundancy caused by dependencies between key parts and those already eliminated by 3NF, **Boyce and Codd** introduced a stricter normal form called **Boyce-Codd Normal Form (BCNF).**

Definition: Boyce-Codd Normal Form (BCNF):
A relation is in **BCNF** if and only if the only elementary functional dependencies are those in which a full key determines an attribute.

Example:

Consider the relation:
HARVEST (Dates, Country, Region, Quality)

Dates	Country	Region	Quality
Medjool	Morocco	Drâa Valley	Average
Deglet Nour	Algeria	Biskra	Excellent
Ajwa	Saudi Arabia	Medina	Good
Khodri	Saudi Arabia	Medina	Average
Barhi	Egypt	Al-Minya	Average
Medjool	United States	California	Poor

This relation is in **3rd Normal Form (3NF)** but **not in BCNF** due to the existence of the following functional dependency:
Region → Country

Therefore, the relation should be decomposed as follows:

1. **HARVEST (Dates, Country, Quality)**
2. **REGIONS (Region, Country)**

8. Database Schema

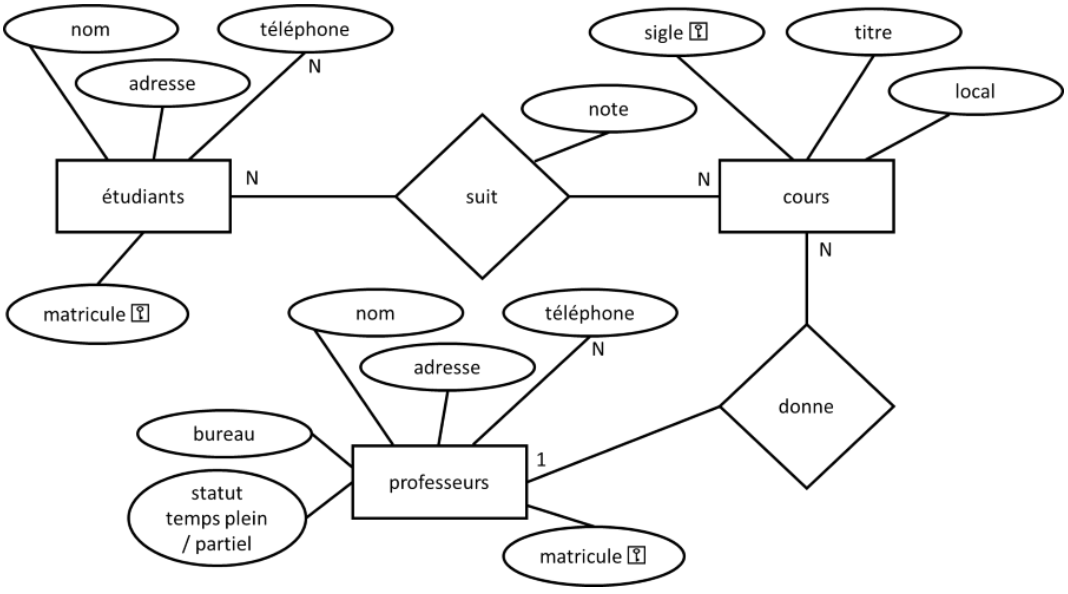
A **database schema** is defined as the set of relation schemas that compose it. In other words, a **database schema** is a visual representation of the structure of a database. It describes how data is organized and the relationships between the different tables within the database.

A **database schema** can include information about tables, columns, primary and foreign keys, as well as integrity constraints.
It can be represented **textually** or **graphically** (e.g., **Entity-Relationship Diagram (ERD)**, **Conceptual Data Model (CDM)**, **Physical Data Model (PDM)**, etc.).

Example:

1. Student (matricule, name, phone, address)
2. Professor (matricule, name, status, phone, office, address)
3. Course (code, title, location, professor_matricule, assistant_matricule)
4. Enrollment (matricule, course_code, grade)

The Entity-Relationship Diagram (ERD) for this schema is as follows:



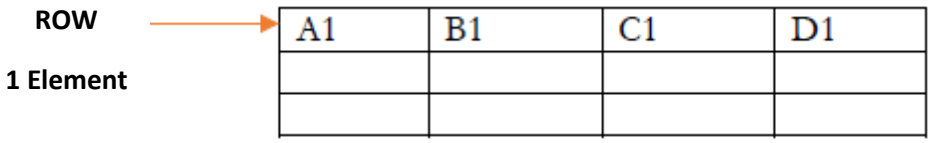
9. Relational Model in SQL

9.1. Table, Column, and Row

- **RELATION = TABLE**

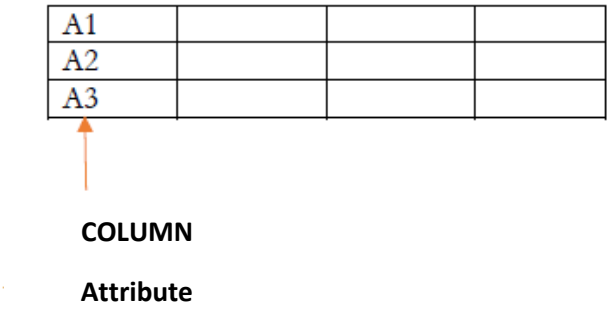
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4

- **ELEMENT or n-tuple = ROW**



We cannot have two identical rows.

- **ATTRIBUTE = COLUMN**



9.2. SQL (Structured Query Language) Description

SQL (Structured Query Language) is a programming language used to manage and manipulate relational databases. It facilitates communication with RDBMS (Relational Database Management Systems) and has been standardized by ANSI (American National Standards Institute) and ISO (International Standards Organization).

SQL provides a set of commands that allow defining and managing the structure of a database, inserting, updating, and deleting data, as well as retrieving data from the database.

The SQL language is divided into several categories, each with a specific set of commands for performing different operations on a database. The three main categories of SQL are as follows:

9.2.1. DDL (Data Definition Language):

- This category deals with defining the structure of the database. DDL commands are used to create, modify, and delete database objects such as tables, indexes, views, etc.

9.2.2. DML (Data Manipulation Language):

- This category focuses on manipulating the data stored in the database. DML commands are used to perform operations such as inserting, updating, deleting, and selecting data.

9.2.3. DCL (Data Control Language):

- This category manages access rights to data. DCL commands allow defining access permissions and controlling user privileges.

In addition to these three main categories, there is a fourth category:

9.2.4. TCL (Transaction Control Language):

- This category of the SQL language deals with transaction control in a database. TCL commands help manage the beginning and end of transactions, as well as commit or roll back changes made to data within a transaction.

The table below summarizes the main SQL commands and organizes them according to their category: DDL, DML, DCL, and TCL. Each command serves a specific function in managing and manipulating relational databases.

Category	SQL Command	Description
DDL (Data Definition Language)		
Table Creation	`CREATE TABLE`	Creates a new table in the database.
Table Modification	`ALTER TABLE`	Modifies the structure of an existing table.
Table Deletion	`DROP TABLE`	Deletes a table from the database.
Index Creation	`CREATE INDEX`	Creates an index on one or more columns.
View Creation	`CREATE VIEW`	Creates a virtual view based on a `SELECT` query.
Database Creation	`CREATE DATABASE`	Creates a new database.
DML (Data Manipulation Language)		
Data Selection	`SELECT`	Retrieves data from the database.
Data Insertion	`INSERT`	Inserts new rows into a table.
Data Update	`UPDATE`	Modifies existing data in a table.

Category	SQL Command	Description
Data Deletion	`DELETE`	Deletes rows from a table.
DCL (Data Control Language)		
Granting Permissions	`GRANT`	Grants access rights to a user or role.
Revoking Permissions	`REVOKE`	Revokes previously granted access rights.
TCL (Transaction Control Language)		
Transaction Commit	`COMMIT`	Commits a transaction, making it permanent.
Transaction Rollback	`ROLLBACK`	Cancels a transaction, undoing its effects.

9.3. Data Definition

9.3.1. Table Creation (CREATE)

The CREATE TABLE statement allows defining a new table, its fields, and field constraints. If NOT NULL is specified for a field, new records must contain valid data in that field. The general structure of this statement with possible constraints is as follows:

```
CREATE TABLE table_name (
   column1 data_type [NULL | NOT NULL] [DEFAULT default_value],
   column2 data_type [NULL | NOT NULL] [DEFAULT default_value],
    ...

    CONSTRAINT constraint_name PRIMARY KEY (column1, column2,
...),
    CONSTRAINT constraint_name UNIQUE (column1, column2, ...),
    CONSTRAINT constraint_name CHECK (condition),
    CONSTRAINT constraint_name FOREIGN KEY (column) REFERENCES
other_table(reference_column)
);
```

Microsoft Access supports multiple data types that can be used when creating tables using the `CREATE TABLE` command. Below are some commonly used data types in Access:

- **Boolean**: BIT
- **Integer Numbers**: SHORT (short integer), SMALLINT (small integer), LONG (long integer), INTEGER (long integer).
- **Real Numbers**: SINGLE (single precision float), DOUBLE (double precision float), NUMERIC (double precision float).
- **Monetary Values**: CURRENCY, MONEY.

- **Date/Time**: DATE, TIME, DATETIME.
- **Text**: VARCHAR(255) (variable size), CHAR(n) or TEXT(n) where n is the number of characters.

Example in Microsoft Access

```
CREATE TABLE Department (
DepartmentID AUTOINCREMENT PRIMARY KEY,
DepartmentName TEXT(50) NOT NULL
);

CREATE TABLE Employee (
EmployeeID AUTOINCREMENT PRIMARY KEY,
Name TEXT(50) NOT NULL,
Salary CURRENCY,
HireDate DATE,
DepartmentID INT,
CONSTRAINT FK_DepartmentLink FOREIGN KEY (DepartmentID)
REFERENCES Department(DepartmentID)
);
```

Note:
A table can be created from an existing one using the following query:

```
SELECT * INTO NewTable
FROM OldTable;
```

9.3.2. Schema Modification (DROP, ALTER)

Deleting a Table

The `DROP TABLE` statement is used to completely remove a table from the database.

Syntax:

```
DROP TABLE table_name;
```

This command deletes the table structure, including indexes, constraints, and all associated objects, along with permanently deleting all data stored in the table. It is important to ensure that no foreign key from another table references the table being deleted.

Modifying an Existing Table

The `ALTER TABLE` statement is used to modify an existing table by adding, modifying, or deleting columns. Below are various operations that can be performed using `ALTER TABLE`:

Adding a Column

```
ALTER TABLE MyTable
ADD COLUMN NewColumn INT;
```

Modifying a Column Data Type

```
ALTER TABLE MyTable
ALTER COLUMN MyColumn VARCHAR(100);
```

Deleting a Column

```
ALTER TABLE MyTable
DROP COLUMN ColumnToDelete;
```

Adding a Primary Key Constraint

```
ALTER TABLE MyTable
ADD CONSTRAINT PK_MyTable PRIMARY KEY (MyColumn);
```

Removing a Primary Key Constraint

```
ALTER TABLE MyTable
DROP CONSTRAINT PK_MyTable;
```

Adding a Foreign Key Constraint

```
ALTER TABLE ChildTable
ADD CONSTRAINT FK_Child_Parent FOREIGN KEY (child_column)
REFERENCES ParentTable(parent_column);
```

Removing a Foreign Key Constraint

```
ALTER TABLE ChildTable
DROP CONSTRAINT FK_Child_Parent;
```

Adding an Index

```
CREATE INDEX IndexName ON MyTable (ColumnName);
```

Deleting an Index

```
DROP INDEX IndexName ON MyTable;
```

Changing Column Data Type

```
ALTER TABLE MyTable
ALTER COLUMN MyColumn NEW_DATA_TYPE;
```

9.3.3 Data Manipulation (INSERT, DELETE, UPDATE)

The `INSERT`, `DELETE`, and `UPDATE` statements belong to the **Data Manipulation Language (DML)**.

INSERT – Adding Records

The `INSERT` command is used to add new rows (records) to a table in the database.

Syntax:

```
INSERT INTO TableName (Column1, Column2, Column3, ...)
VALUES (Value1, Value2, Value3, ...);
```

Example:

```
INSERT INTO Departement (DepartementID, NomDepartement)
VALUES (1, 'Informatique');
```

UPDATE – Modifying Records

The `UPDATE` statement is used to modify existing records in a table.

Syntax:

```
UPDATE table_name
SET
    column_name_1 = {expression_1 | (SELECT ...)},
    column_name_2 = {expression_2 | (SELECT ...)},
    ...
    column_name_n = {expression_n | (SELECT ...)}
WHERE condition;
```

Example:

```
UPDATE Departement
SET NomDepartement = 'Physique'
WHERE DepartementID = 1;
```

DELETE – Removing Records

The `DELETE` statement is used to delete records from a table based on a specified condition.

Syntax:

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

Example:

```
DELETE FROM Departement
WHERE DepartementID = 1;
```