**Terminology**

* A **relation** is a table with columns and rows.
* **Attribute** is a named column of a relation.
* **Domain** is a set of allowable values for one or more attributes.
* **Tuple** is a row of a relation.
* **Degree** is the number of attributes in a relation.
* **Cardinality** is the number of tuples in a relation.
* **Relationship** is a link or dependency between relations.
* **Relational Database** is a collection of relations.

**Atomic Values**

It simply means a simple single piece of data. Each value within a tuple (a row in a relational database) is considered a single, indivisible unit, meaning it cannot be further broken down into smaller components; essentially, each value is a basic data element with no internal structure.

For example, storing a full address in one column is not atomic if you later need to query by the city, street, or zip code. It must be a single value.

**Characteristics of Relations**

* Each relation in the same relational database schema has a distinct name
* Each value in a tuple is atomic/single value
* Each attribute in a relation has a distinct name.
* Values of an attribute are all from the same domain.
* Each tuple is distinct.
* Order of attributes has no significance.
* Order of tuples has no significance, theoretically.

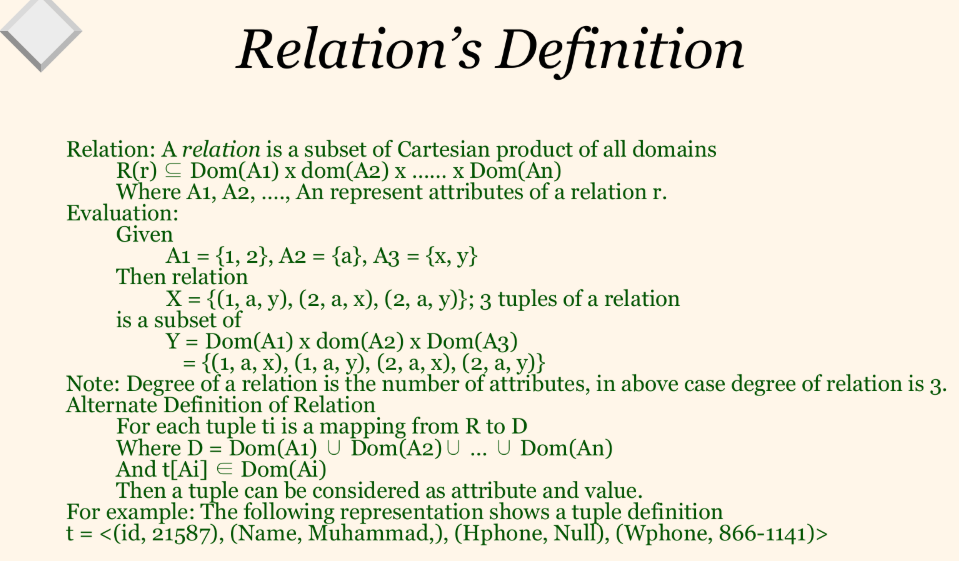
**--Read LEC-3: Entity-Relationship Mode (DBMS\_Full\_Notes.pdf pg. 6)--**

**Relation and Its Attributes**

* **Relation (R):**A relation is defined by a set of attributes (or columns). For instance, consider a relation representing a student table with attributes such as id, Name, Hphone (home phone), and Wphone (work phone).
* Attributes (A1, A2, ..., An):  
  Each attribute has a domain—a set of allowable values. For example:
  + Dom(id) might be all valid integer IDs.
  + Dom(Name) might be all possible strings for names.
  + Dom(Hphone) might include phone number formats (or NULL if not provided).
  + Dom(Wphone) might be similarly defined for work phone numbers.

**Relational Data Model Notations**

* The letters Q, R, S denote the abstract relation names. R(A1, A2, A3, …., An) denotes a relation schema R of degree n.
* Example: STUDENT(RollNo, Name, Phone, Address, Mobile, DOB)
* COURSE(CourseID, Title, CreditHours)
* Both t[Ai] and t.Ai refers to the value vi in t for attribute Ai



**Tuple as a Mapping**

* **Tuple (t):**  
  A tuple is a single row in the relation, representing a complete record. Formally, each tuple ti is considered a **mapping** from the set of attributes R to the set of values D.
* **Domain (D):**  
  Here, D is defined as the union of the domains of all attributes:

D = Dom(A1) ∪ Dom(A2) ∪ … ∪ Dom(An)

This means that the value for any attribute in a tuple must belong to its respective domain.

* **Mapping Definition:**  
  For each attribute Ai​ in the relation, the tuple provides a value t[Ai​] such that:

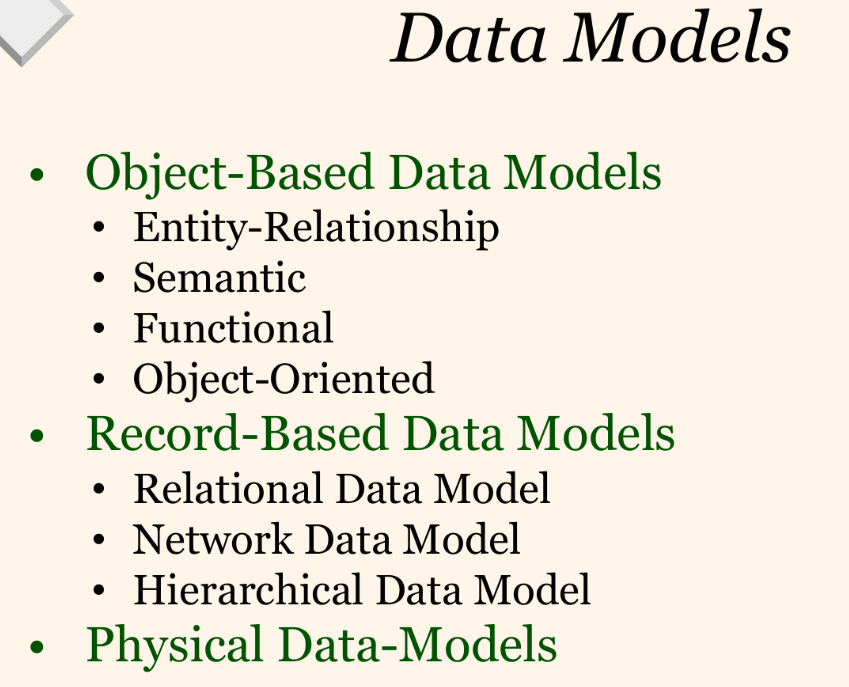
t[Ai] ∈ Dom(Ai)

This condition ensures that each value in the tuple is valid for its corresponding attribute.

**Tuple as a Set of Attribute-Value Pairs**

A tuple can be viewed as a collection of pairs where each pair consists of an attribute and its corresponding value. For example, consider the following tuple representation:

**t=⟨(id,21587), (Name, Muhammad), (Hphone, Null), (Wphone, 866−1141)⟩**



**Data Models**

Data models provide a structured framework for organizing and managing data, outlining how data is stored, accessed, and related to one another. It is the concept of **tools that are developed to summarize the description of the database**.

Data Model is the modelling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction (external/view, logical/conceptual, internal/physical).

**Summary**

* **Object-Based Data Models:**  
  Emphasize real-world entities and their behaviors. They include:
  + **Entity-Relationship (ER):** Uses diagrams to show entities, attributes, and relationships.
  + **Semantic:** Adds meaning, constraints, and business rules to the data.
  + **Functional:** Based on mathematical functions, how data is processed and transformed.
  + **Object-Oriented:** Encapsulate data and behavior, supporting inheritance and polymorphism.
* **Record-Based Data Models:**  
  Focus on how data is organized in records or sets. They include:
  + **Relational:** Tables of rows and columns, based on set theory.
  + **Network:** Graph-like structures with flexible, many-to-many relationships.
  + **Hierarchical:** Tree-like structures with parent-child relationships.
* **Physical Data Models:**
  + Address the actual storage of data on hardware, optimized for performance and scalability.
  + Focus on file structures, indexes, partitioning, and other implementation details.

**I. Object-Based Data Models**

These models emphasize representing real-world entities (objects) along with their properties (attributes) and behaviors (operations or methods).

A screenshot of a computer

Description automatically generated

A screenshot of a black and white text

Description automatically generated

A black screen with white text

Description automatically generated

A screenshot of a black screen

Description automatically generated

**II. Record-Based Data Models**

These models emphasize how data is stored in records or rows and focus on the structure and organization of the data.

A black screen with white text

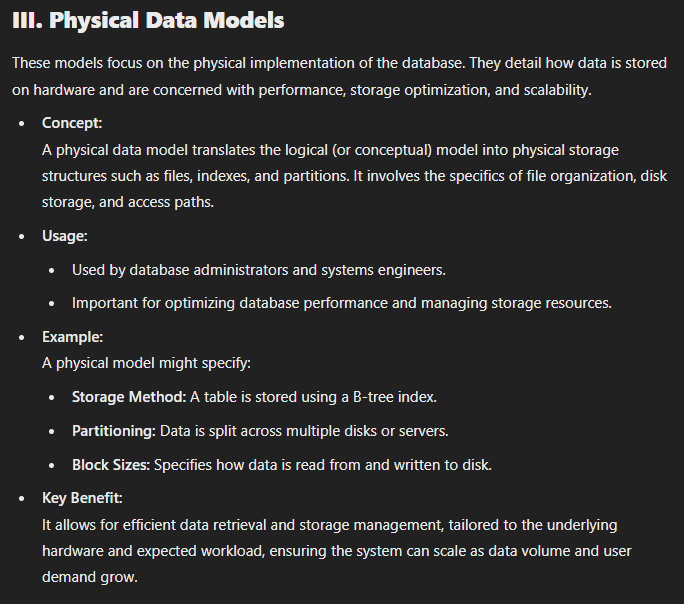
Description automatically generated

A screenshot of a black screen

Description automatically generated

A black screen with white text

Description automatically generated



**Database Keys**

**1. Super Key**

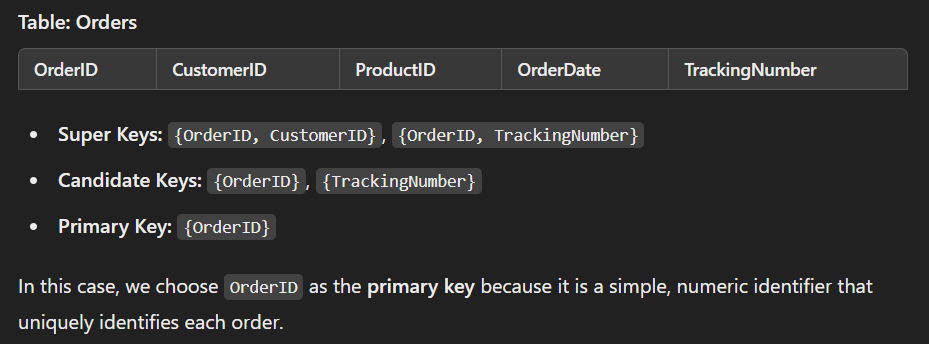
* **Definition:**  
  A **super key** is any combination of attributes in a table that uniquely identifies each record (tuple) in the table. It may contain extra, redundant attributes beyond what is necessary for unique identification.
* **Example:**  
  Consider a Student table with the following attributes: StudentID | Name | Email | Phone
* Examples of super keys:
* {StudentID}
* {StudentID, Name}
* {StudentID, Email, Phone} The first example {StudentID} is enough to uniquely identify a student, while the others contain redundant information.

**2. Candidate Key**

* **Definition:**  
  A **candidate key** is a minimal subset of attributes from a super key that uniquely identifies each record in a table. It does not contain any unnecessary attributes.
* **Example:**  
  From the above table, {StudentID} and {Email} can both uniquely identify a student, so both are **candidate keys**.
* **Key Point:**  
  A table can have **multiple candidate keys**, but each must uniquely identify a record.

**3. Primary Key**

* **Definition:**  
  A **primary key** is a candidate key that has been chosen to uniquely identify records in a table. It enforces uniqueness and prevents null values.
* **Example:**  
  In the Student table, {StudentID} can be selected as the **primary key**.
* **Important Rules for Primary Keys:**
  + Must be unique across all rows.
  + Cannot contain NULL values.
* Why Choose One Candidate Key as the Primary Key?
  + While {Email} is a valid candidate key, {StudentID} is more suitable as the primary key because it is typically a simple, numeric value designed explicitly for unique identification.



**4. Composite Key**

**Definition**

* A **composite key** is a primary key made up of two or more attributes (columns) that together uniquely identify a record in a table.

**When to Use?**

* Used when a single attribute is not enough to uniquely identify a record, but a combination of attributes can.

A screenshot of a computer

Description automatically generated

**Advantages of Composite Key**

* Ensures uniqueness in complex relationships.
* Allows modeling relationships between entities naturally.

**Disadvantages of Composite Key**

* Increases table complexity.
* May slow down database operations due to multiple attributes in joins.

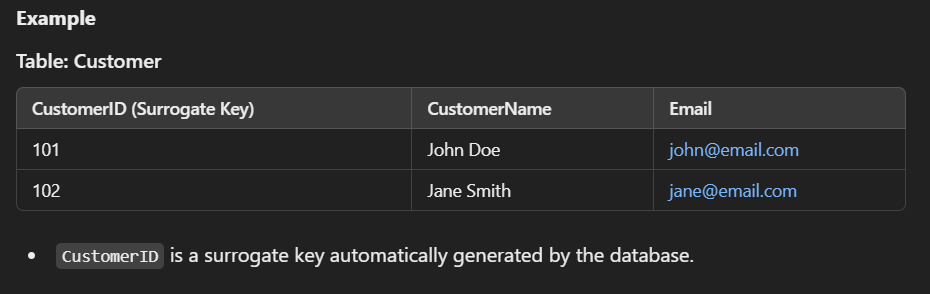
**2. Surrogate Key**

**Definition**

* A **surrogate key** is an artificial, system-generated key (usually a unique integer or UUID) used as a primary key instead of a naturally occurring attribute.

**When to Use?**

* When no natural key is suitable, or when using a composite key becomes complex.

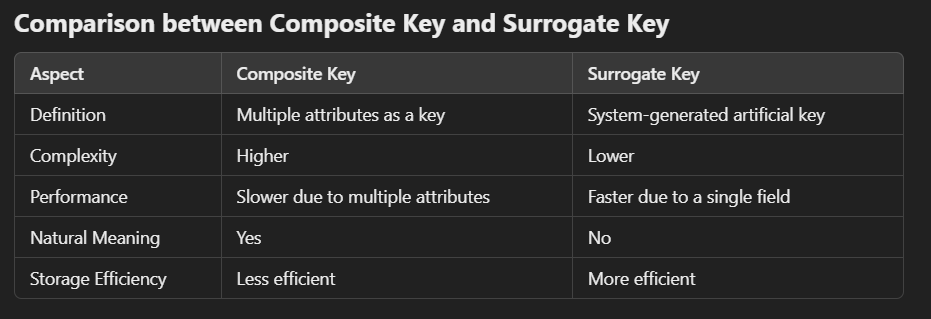


**Advantages of Surrogate Key**

* Simple, efficient, and unique.
* Avoids changes when natural keys are updated.
* Improves performance in joins and indexing.

**Disadvantages of Surrogate Key**

* Can lose human readability and meaningfulness.



**Real-World Example Use Case**

* **Composite Key:** In a **student-course enrollment system**, {StudentID, CourseID} can form a composite key as one student can enroll in multiple courses.
* **Surrogate Key:** In a **social media platform**, PostID (generated by the system) can be a surrogate key to uniquely identify each post instead of relying on multiple attributes like user and timestamp.

**5. Foreign Key**

A **foreign key** is an attribute (or a set of attributes) in one table that provides a link between data in two tables. It establishes a relationship between the data in the current (child) table and data in another (parent) table, ensuring referential integrity within the database.

**Relationship Establishment:**

* A foreign key in a child table refers to the primary key (or a candidate key) in a parent table.
* This relationship enforces that every value in the foreign key column must match an existing value in the referenced column of the parent table or be NULL (if allowed).

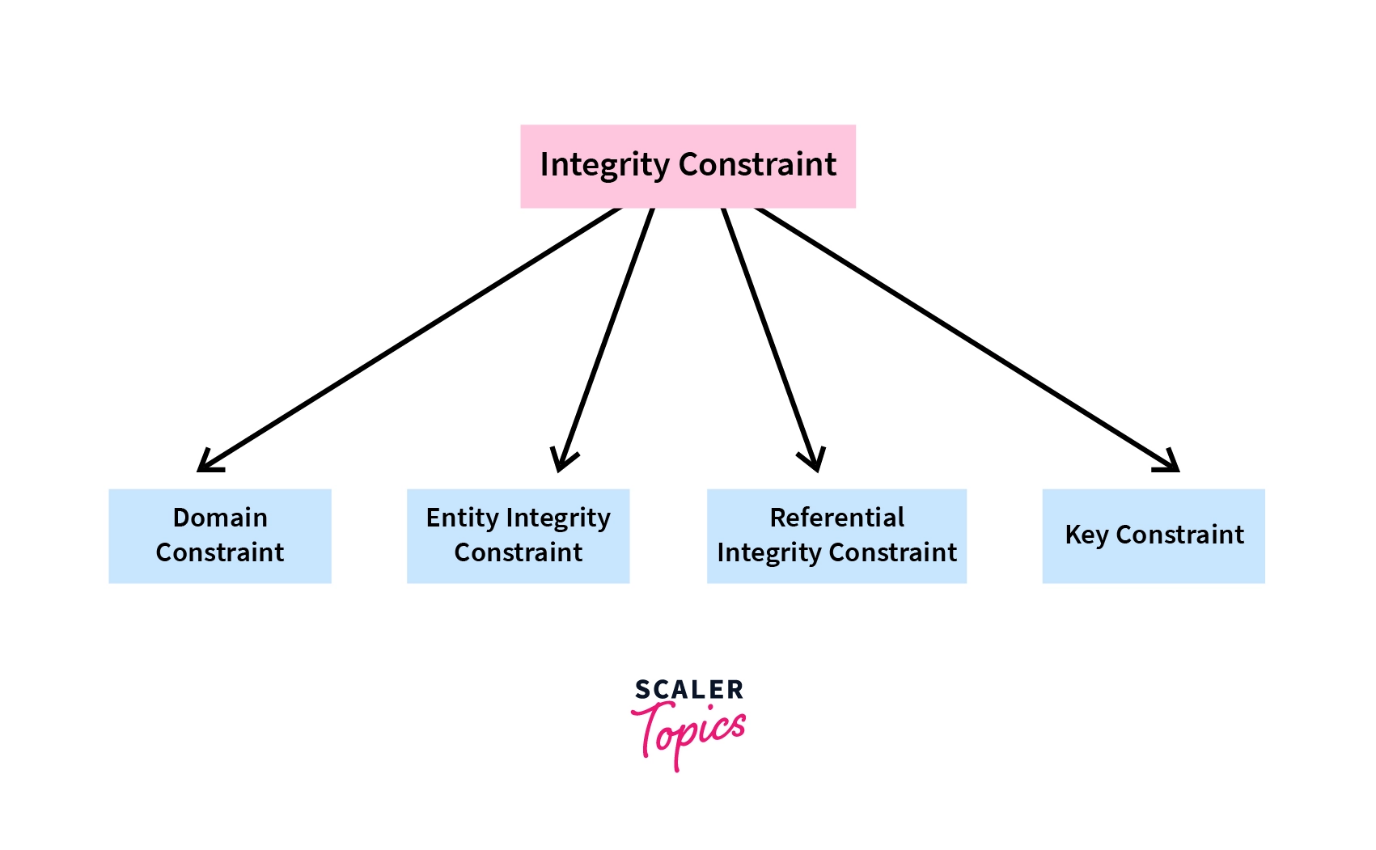
**Referential Integrity:**

* Referential integrity ensures that the relationships between tables remain consistent.
* For example, if a foreign key references a particular record in the parent table, that record must exist. If you try to insert a record into the child table with a non-existent parent key, the database system will reject the operation.
* Similarly, deleting or updating records in the parent table may be restricted or cascade changes to maintain integrity.

**Cascading Actions:**

* Many database management systems allow you to specify actions that occur when a referenced record in the parent table is updated or deleted. Common options include:
  + **CASCADE:** Automatically updates or deletes corresponding records in the child table.
  + **SET NULL:** Sets the foreign key value in the child table to NULL when the referenced record is deleted.
  + **RESTRICT/NO ACTION:** Prevents the deletion or update if there are matching records in the child table.

**Integrity Constraints:**



**1. Domain Constraints**

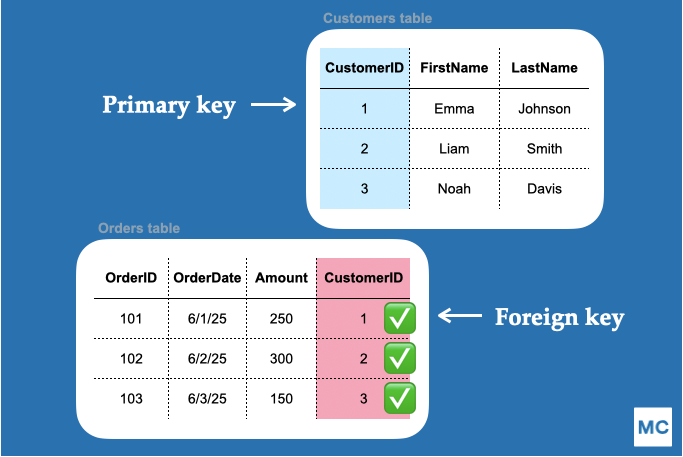
* **Definition**: Ensures that the value of an attribute (column) falls within a specified domain or range.
* **Examples**:
  + A age column must be a positive integer.
  + A gender column can only have values like 'M', 'F', or 'O'.
* **Purpose**: Restricts the type of data that can be stored in a column.

**2. Entity Integrity Constraints**

* **Definition**: Ensures that each row (entity) in a table is uniquely identifiable.
* **Primary Key Constraint**:
  + A primary key is a column or set of columns that uniquely identifies a row.
  + It cannot contain NULL values and must be unique.
* **Example**:
  + In a Students table, the student\_id column is the primary key.

**3. Referential Integrity Constraints**

* **Definition**: Ensures that relationships between tables remain consistent.
* **Foreign Key Constraint**:
  + A foreign key in one table refers to the primary key in another table.
  + It ensures that the value in the foreign key column must exist in the referenced table or be NULL.
* **Example**:
  + In an Orders table, the customer\_id column is a foreign key referencing the customer\_id in the Customers table.

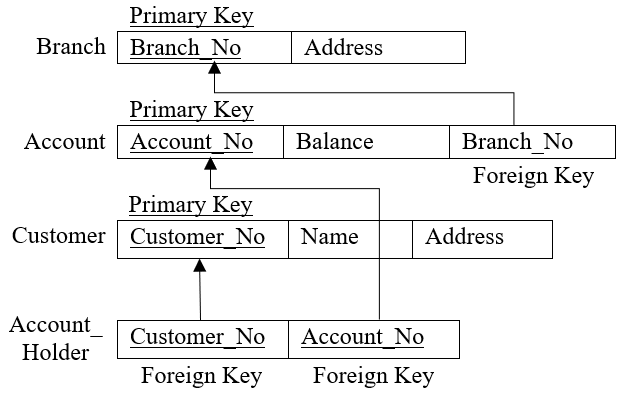


**4. Key Constraints**

* **Definition**: Ensures that a column or set of columns uniquely identifies a row.
* **Types**:
  + **Primary Key**: Uniquely identifies a row (as mentioned above).
  + **Unique Key**: Ensures that all values in a column are unique, but it can contain NULL values.
* **Example**:
  + In an Employees table, the email column can be a unique key.

**5. Check Constraints**

* **Definition**: Ensures that a condition or expression is satisfied for the data in a column.
* **Examples**:
  + A salary column must be greater than **0**.
  + A **date\_of\_birth** column must be earlier than the current date.
* **Purpose**: Enforces custom rules on the data.

**🡪 Referential Integrity Mapping (Primary Key & Foreign Key Mapping)**