4.1

(a) Relation

- In the relational data model, a **relation** represents a table within a database. It's essentially a set of tuples (rows) with a consistent structure across each tuple. Each relation has a unique name, and each row within the relation corresponds to a unique entry in the table. Relations are foundational to relational databases, as they define the organized format through which data is stored and accessed.

(b) Attribute

- An **attribute** is a column within a relation, representing a specific piece of information or property about the entity that the relation represents. For example, in a student relation, attributes might include StudentID, Name, and Age. Each attribute holds values for all tuples (rows) in each relation. Attributes help define the structure of the data within each relation.

(c) Domain

- A **domain** represents the set of valid values that an attribute can take. Each attribute in a relation is associated with a domain that restricts the types of values allowed. For instance, if an attribute is defined to have the domain of integer values between 1 and 100, only values within this range can be stored in that attribute. Domains are crucial for ensuring data integrity by preventing invalid entries.

(d) Tuple

- A **tuple** is a single row within a relation, representing a unique record or instance in the table. Each tuple is an ordered set of values, one for each attribute in the relation. For example, a tuple in a student relation might represent a single student with specific values for attributes like StudentID, Name, and Age. Tuples are the actual data entries stored in each relation.

(e) Intension and Extension

- **Intension** refers to the structure or schema of a relation, describing the relation's name, attributes, and their domains. Intension is static; it defines the relation itself rather than the data it currently holds.
- **Extension**, on the other hand, refers to the set of tuples currently contained in a relation. It's dynamic and changes as tuples (records) are added, updated, or deleted. So, while intension defines what a relation is, extension represents its actual data content at any given point in time.

(f) Degree and Cardinality

- **Degree** of a relation is the number of attributes it contains. For example, if a relation has attributes ID, Name, and Age, its degree is 3.
- **Cardinality** refers to the number of tuples in a relation. For example, if a student relation has 50 rows, its cardinality is 50. Degree remains fixed for a relation unless its schema is altered, but cardinality can change as data is added or removed.

4.4

In the relational data model, a **relation** is a table-like structure with rows and columns that represents data about entities and their relationships. Here are the key properties of a relation:

1. **Uniqueness of Tuples**: Each row, or tuple, in a relation must be unique. No two rows can have the same values across all attributes, which ensures that each entity instance in the table is uniquely represented.
2. **Uniqueness of Attribute Names**: Each column, or attribute, in a relation has a unique name, which helps to identify data fields clearly.
3. **Attribute Values are Atomic**: Each attribute in a relation contains atomic (indivisible) values. This property, also known as the First Normal Form, ensures that attributes don't contain multiple values, making data simpler to query and manipulate.
4. **Order Irrelevance**: In a relation, the order of tuples (rows) and attributes (columns) does not matter. The relational model treats them as a set where the order does not impact the information stored.
5. **Domain Constraints**: Each attribute in a relation is associated with a domain (a set of permissible values). This ensures that data values are consistent, such as integers, strings, or specific enumerated types.
6. **Primary Key**: A relation usually has a primary key, an attribute or a set of attributes that uniquely identify each tuple within the relation. The primary key enforces uniqueness and ensures that each row is distinguishable.

4.5

Candidate Key vs. Primary Key

- **Candidate Key**: A candidate key is an attribute or a combination of attributes that can uniquely identify each tuple (row) in a relation. A relation can have multiple candidate keys, each of which is a potential unique identifier. Candidate keys must follow these properties:
    - Uniqueness: Each tuple must have a unique value or set of values for the candidate key.
    - Irreducibility: A candidate key cannot have any unnecessary attributes; removing an attribute would make it non-unique.
- **Primary Key**: The primary key is one of the candidate keys selected to uniquely identify tuples in the relation. It enforces unique, non-null values across the table and is often chosen based on database design conventions (e.g., simplicity, stability). Only one primary key is chosen per relation, even if there are multiple candidate keys.

Example: Consider a Student relation with attributes StudentID, Email, and Phone Number. If each attribute uniquely identifies a student, then they are all candidate keys. However, if StudentID is selected as the primary key, it becomes the primary identifier for the relation.

2. Foreign Key

- A **foreign key** is an attribute or a set of attributes in one relation that references a candidate key (often the primary key) in another relation. This relationship enforces a referential integrity constraint, meaning the values in the foreign key must match values in the referenced candidate key or be null if permitted. Foreign keys establish links between related tables.

Example of a Foreign Key: Suppose we have a Course relation with a primary key CourseID, and a StudentEnrollment relation where each enrollment entry associates a student with a course. The CourseID in StudentEnrollment serves as a foreign key that references the CourseID primary key in Course:

Course Relation

CourseID	**CourseName**
CS101	Computer Science
MATH200	Calculus

StudentEnrollment Relation

EnrollmentID	StudentID	CourseID
001	001	CS101
002	002	MATH200

Foreign keys in one relation must correspond to candidate keys in another relation (often the primary key). This relationship allows multiple tables to connect logically while maintaining data consistency. For example, the foreign key CourseID in StudentEnrollment must match a candidate key in Course, establishing a valid link between enrollments and courses.

4.6

1. Entity Integrity

- **Definition**: Entity integrity requires that every relation in a database must have a primary key, and this primary key cannot contain null values. Since a primary key uniquely identifies each tuple (row) in a relation, it is essential for maintaining distinct entries and preventing ambiguous records.
- **Why It's Desirable**: Enforcing entity integrity ensures that each record is uniquely identifiable, which helps in accurate data retrieval, updating, and deletion. Null values in a primary key would create ambiguity, as they would imply missing or incomplete information, making it impossible to uniquely identify records. Entity integrity maintains the reliability of database operations and helps prevent accidental data loss or duplication.

Example: In a student table, StudentID is a primary key and cannot be null. This requirement prevents the insertion of a student record without an identifier, ensuring that each student can be uniquely referenced.

2. Referential Integrity

- **Definition**: Referential integrity mandates that any foreign key in a relation must match an existing primary key (or candidate key) value in another relation or be null if allowed. This rule ensures that relationships between tables remain consistent and that records reference valid data in related tables.
- **Why It's Desirable**: Enforcing referential integrity preserves the logical relationships between tables, preventing orphaned records and invalid references. Without referential integrity, a foreign key might reference a nonexistent entry, leading to inconsistent data and potential errors in reporting, querying, and data management.

Example: In an Enrollment table, StudentID is a foreign key referencing the primary key in the student table. Referential integrity requires that each StudentID in Enrollment must match a valid StudentID in Student, preventing entries for enrollments where the student does not exist.

Enforcing these integrity rules is crucial for maintaining data consistency, accuracy, and reliability across relational databases. They help prevent the insertion of invalid data, maintain proper relationships between entities, and reduce errors in data operations, which is especially important in large, interdependent data sets. By ensuring entity and referential integrity, the database remains a trustworthy and accurate source of information, supporting the integrity of applications and analysis that rely on it.

4.7

Definition of Views

In a relational database, a **view** is a virtual table representing the result of a query on one or more tables. Views do not store data physically; instead, they generate data dynamically based on the underlying tables each time they are accessed. A view can include a subset of columns and rows, computed columns, or join data from multiple tables, simplifying data access and presentation.

Importance of Views in a Database Approach

Views are valuable in database systems for several reasons:

1. **Data Security and Access Control**: Views can restrict access to specific data by showing only particular columns or rows to certain users. This allows sensitive information to be hidden from users without having to manage permissions on the underlying tables directly.
2. **Simplified Querying**: Views can simplify complex queries by presenting pre-defined data sets to users. Users can access a complex join or aggregation result as a single virtual table, reducing the need to write complex SQL every time.
3. **Logical Data Independence**: Views provide a layer of abstraction, enabling changes to underlying tables without affecting applications or queries that rely on the view. This promotes data consistency and reduces the impact of structural changes.
4. **Data Integrity and Consistency**: Views can help enforce data consistency by standardizing access to frequently referenced data. By centralizing data access through views, the database can reduce redundancy and ensure that applications retrieve the same data set consistently.

5. **Customized Data Presentation**: Views allow tailored data representations for different user roles or purposes. For instance, a view might format data for a reporting dashboard, showing only relevant information.

   Suppose there's a customer table with columns CustomerID, Name, Email, and CreditCardNumber. A view called CustomerView could exclude the CreditCardNumber to protect sensitive data:

   ```
   CREATE VIEW CustomerView AS
   SELECT CustomerID, Name, Email
   FROM Customer
   ```

4.8

1. Identifying the Foreign Keys

Based on the schema, the foreign keys are as follows:

- **ROOM Table**:
  - HOTEL_NO is a foreign key referencing HOTEL_NO in the HOTEL table. This establishes a relationship between each room and its respective hotel.
- **BOOKING Table**:
  - HOTEL_NO is a foreign key referencing HOTEL_NO in the HOTEL table, linking each booking to a specific hotel.
  - GUEST_NO is a foreign key referencing GUEST_NO in the GUEST table, associating each booking with a specific guest.
  - ROOM_NO could also serve as a foreign key referencing ROOM_NO in the ROOM table (in combination with HOTEL_NO), linking each booking to a specific room in a hotel.

2. Application of Entity and Referential Integrity Rules

- **Entity Integrity**:
  - Each table has a primary key ensuring unique identification of records:
    - **HOTEL**: The primary key is HOTEL_NO, ensuring each hotel is uniquely identifiable.
    - **ROOM**: The combination of ROOM_NO and HOTEL_NO forms the primary key, making each room unique within a specific hotel.
    - **BOOKING**: The combination of HOTEL_NO, GUEST_NO, and DATE_FROM forms the primary key, uniquely identifying each booking entry.
    - **GUEST**: The primary key is GUEST_NO, ensuring each guest record is unique.
  - This enforcement of primary keys maintains **entity integrity** by ensuring that each row can be uniquely referenced, preventing duplicate or ambiguous records within each table.

- **Referential Integrity**:
    - Referential integrity rules ensure that foreign keys must match primary key values in related tables or be null if allowed:
        - In the **ROOM** table, HOTEL_NO must match a valid HOTEL_NO in the HOTEL table, ensuring that each room is linked to an existing hotel.
        - In the **BOOKING** table:
            - HOTEL_NO must refer to a valid HOTEL_NO in the HOTEL table.
            - GUEST_NO must refer to a valid GUEST_NO in the GUEST table, ensuring each booking is associated with a registered guest.
            - ROOM_NO and HOTEL_NO together should match an entry in the ROOM table, ensuring that a booking only references valid rooms in specific hotels.

4.9

HOTEL

HOTEL_NO	HOTEL_NAME	CITY
101	Grand Plaza	New York
102	Oceanview Resort	Miami
103	Mountain Lodge	Denver

ROOM

ROOM_NO	HOTEL_NO	TYPE	PRICE
1	101	Suite	250
2	101	Single	100
3	102	Double	150
4	103	Suite	300
5	103	Single	120

GUEST

GUEST_NO	GUEST_NAME	GUEST_ADDRESS
201	Kira Solo	456 Oak Ave, Miami
202	Fani Abreha	123 Elm St, NY
203	John Doe	789 Pine Rd, Denver

BOOKING

HOTEL_NO	GUEST_NO	DATE_FROM	DATE_TO	ROOM_NO
101	201	2024-12-01	2024-12-05	1
102	202	2024-12-10	2024-12-15	3
103	203	2024-11-01	2024-11-26	4

5.1

Difference Between Procedural and Non-Procedural Languages

- **Procedural Language**:
    - In a procedural language, the user specifies a sequence of operations to be performed on the data to obtain the desired result. The focus is on *how* to obtain the result.
    - Examples include programming languages like C, Java, and SQL's procedural extensions (e.g., PL/SQL).
    - **Relational Algebra** is considered procedural because it defines a step-by-step method to retrieve data by applying a sequence of operations like selection, projection, join, etc.
- **Non-Procedural Language**:
    - In a non-procedural language, the user specifies *what* data they want, without specifying the steps to retrieve it. The system determines the optimal method for data retrieval.
    - Examples include SQL (in its standard form) and query languages like SPARQL.
    - **Relational Calculus** is non-procedural as it focuses on what results should be returned rather than detailing the retrieval steps. It expresses queries using logical expressions and conditions.

Classification of Relational Algebra and Relational Calculus

- **Relational Algebra**: A *procedural* language, as it requires defining the exact operations to manipulate and retrieve data.
- **Relational Calculus**: A *non-procedural* language, as it uses logical expressions to specify conditions, letting the system determine how to execute the query.

5.8(a)

This relation extracts the hotelNo attribute for all rooms where the price is greater than 50.

5.8(b)

This relation performs a natural join between Room and Hotel based on matching hotelNo attributes and then projects the hotel and hotelNo attributes.

5.8(c)

This relation joins Hotel and Room on hotelNo, filters for rooms with a price greater than 50, and then projects the hotelName attribute.

5.8(d)

This relation joins Guest with Booking records where the dateTo is on or before January 1, 2007.