UNIT 3: Relational Database Design

Unit III	Relational Database Design	06 Hours

Relational Model: Basic concepts, Attributes and Domains, CODD's Rules. **Relational Integrity**: Domain, Referential Integrities, Enterprise Constraints. **Database Design**: Features of Good Relational Designs, Normalization, Atomic Domains and First Normal Form, Decomposition using Functional Dependencies, Algorithms for Decomposition, 2NF, 3NF, BCNF.

1. Relational Model

a. Basic Concepts

The **Relational Model** is the most widely used data model for databases today. It was proposed by **Dr. E. F. Codd** in 1970. This model organizes data into **relations**, which are two-dimensional tables. Each relation has a **schema** that defines its name, the attributes (columns), and their data types.

- **Relation (Table)**: A relation consists of **tuples (rows)** and **attributes (columns)**.
- **Tuple**: A row in the table that represents a single data item or record.
- **Attribute**: A column that represents a field of the relation.
- **Degree**: The number of attributes in a relation.
- **Cardinality**: The number of tuples in a relation.

Key Features:

- Data is stored in tables, which are easy to understand and work with.
- Relations are **unordered**; the order of rows and columns doesn't matter.
- A table must have a **primary key**—a set of one or more attributes that uniquely identifies a tuple.

Example:

```
STUDENT(StudentID, Name, Age, Course)
```

Operations on relational models are defined through **Relational Algebra**, such as **Select, Project, Join, Union, Intersection, Difference**, etc.

b. Attributes and Domains

Each attribute in a relation is associated with a **domain**, which is the set of all possible values that attribute can hold.

- **Attribute**: Describes a property of the entity.
- **Domain**: A set of atomic values that an attribute can take.
  - Atomic means indivisible from the database's point of view.
  - Example: Domain of `Age` might be integers from 1 to 120.

Important Points:

- Domains help in enforcing **data type constraints**.
- Ensures that data stored in a column adheres to a certain type and range.
- Prevents incorrect or invalid data from entering the database.

c. Codd's Rules

Dr. Codd proposed **13 rules (0 to 12)** that a database management system must follow to be considered fully relational.

Rule 0 (Foundation Rule)

A system must manage databases entirely through its relational capabilities.

Rule 1: Information Rule

All information is represented logically as data values in tables.

Rule 2: Guaranteed Access Rule

Every data item must be accessible by using table name, primary key, and column name.

Rule 3: Systematic Treatment of Null Values

NULLs must be supported and distinguished from zero or empty strings.

Rule 4: Active Online Catalog

Metadata (schema, structure) must be stored in the same form as ordinary data.

Rule 5: Comprehensive Data Sublanguage Rule

A relational system should support at least one language that supports:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- View definition
- Integrity constraints
- Authorization
- Transaction boundaries

Rule 6: View Updating Rule

All views that are theoretically updatable must be updatable by the system.

Rule 7: High-Level Insert, Update, Delete

These operations must be supported on sets of data, not just single rows.

Rule 8: Physical Data Independence

Changes to physical storage should not affect the logical model.

Rule 9: Logical Data Independence

Changes to the logical structure (e.g., adding a column) should not affect applications.

Rule 10: Integrity Independence

Integrity constraints should be stored separately and modifiable without affecting applications.

Rule 11: Distribution Independence

Users should not know whether data is distributed.

Rule 12: Non-subversion Rule

If a low-level language exists, it should not bypass the integrity rules set through the high-level relational language.

2. Relational Integrity

a. Domain Integrity

- Ensures that values in each column are from the specified domain.
- Enforced using:
    - **Data types** (e.g., INT, DATE)
    - **NOT NULL constraints**
    - **CHECK constraints**
    - **DEFAULT values**

Example:

```sql
CREATE TABLE Employees (
  ID INT,
  Name VARCHAR(50),
  Age INT CHECK (Age >= 18)
);
```

b. Referential Integrity

- Ensures **validity of foreign keys**.
- A foreign key in one table must match a primary key in another, or be NULL.
- Enforced using:
  - **FOREIGN KEY constraints**
  - **ON DELETE CASCADE/SET NULL/RESTRICT**

Example:

```sql
CREATE TABLE Orders (
  OrderID INT,
  CustomerID INT,
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Violating referential integrity leads to:

- **Orphan records**
- Inconsistent data

c. Enterprise Constraints

- Also called **user-defined constraints**, these are business-specific rules.
- Not directly covered by domain or referential integrity.
- Examples:
  - Employee’s salary cannot exceed their manager's.
  - Total order amount should not exceed customer’s credit limit.

Implemented via:

- **Triggers**
- **Stored Procedures**
- **Application code**

These constraints enforce **real-world business logic**.

3. Database Design

a. Features of Good Relational Designs

A well-designed relational database:

- **Minimizes redundancy**
- **Avoids anomalies** (insertion, update, deletion)
- **Ensures data integrity**

- **Supports scalability**
- **Maintains logical clarity**

Good design leads to:

- **Efficient queries**
- **Reduced storage**
- **Simplified maintenance**

b. Normalization

- A process of organizing data to reduce redundancy and improve integrity.
- Involves decomposing tables based on **functional dependencies**.
- Each step in normalization corresponds to a **Normal Form**.

Goals:

- Remove data duplication.
- Avoid anomalies.
- Achieve better data structure.

c. Atomic Domains and First Normal Form (1NF)

- A relation is in 1NF if:
    - All domains contain **atomic (indivisible) values**.

- There are **no repeating groups or arrays**.

Violation Example:

```
Student(ID, Name, PhoneNumbers)
```

- If `PhoneNumbers` = “12345, 67890”, it violates 1NF.

To correct:

- Break into multiple rows or separate tables.

d. Decomposition using Functional Dependencies

- Used to break down tables while maintaining:
  - **Lossless-join** property
  - **Dependency preservation**

A **Functional Dependency (FD)** `X → Y` means:

- If two tuples agree on X, they must agree on Y.

Decomposition:

- Should not lose any data or functional dependencies.
- Should maintain all constraints.

e. Algorithms for Decomposition

Two main goals:

1. **Lossless-Join Decomposition**:
   - Join of decomposed tables must recreate original table.
   - Ensured if the common attribute is a **super key**.
2. **Dependency Preservation**:
   - All original FDs must be enforceable in decomposed tables.
   - Important for **query optimization** and **constraint enforcement**.

Chase Algorithm:

- Tests whether a decomposition is lossless and preserves dependencies.

f. Second Normal Form (2NF)

A relation is in 2NF if:

- It is in **1NF**.
- **No partial dependencies** exist.
  - Partial dependency: Non-prime attribute depends on part of a composite key.

Applies only if:

- Table has a **composite primary key**.

Solution:

- Move partial dependencies to a new table.

Example:

```
StudentCourse(StudentID, CourseID, CourseName)
```

- `CourseName` depends only on `CourseID` ⇒ Partial dependency ⇒ Move to `Course(CourseID, CourseName)`

g. Third Normal Form (3NF)

A relation is in 3NF if:

- It is in **2NF**.
- It has **no transitive dependencies**.

Transitive Dependency:

- Non-key attribute depends on another non-key attribute.

Example:

```
Employee(EmpID, DeptID, DeptName)
FDs: EmpID → DeptID, DeptID → DeptName
```

- Transitive dependency: EmpID → DeptName

Fix:

- Create separate tables for `Department(DeptID, DeptName)` and `Employee(EmpID, DeptID)`

h. Boyce-Codd Normal Form (BCNF)

- Stronger than 3NF.
- A relation is in BCNF if for every FD `X → Y`, **X is a superkey**.

Example of Violation:

```
Table: Teaching(Course, Professor, Room)
FDs: Course → Room, Professor → Room
```

- Neither `Course` nor `Professor` is a superkey.

To fix:

- Decompose into:
  - `CourseRoom(Course, Room)`
  - `ProfessorCourse(Professor, Course)`

Trade-off:

- May lose **dependency preservation** but gains stronger structure.

✅ SUMMARY:

Topic	Key Concepts / Summary
Relational Model - Basic Concepts	Data is organized into **relations (tables)**. Each relation has **tuples (rows)** and **attributes (columns)**.
Attributes and Domains	Each attribute has a specific **domain** (valid set of values). E.g., Age: Integer [0-120].
Codd's Rules	12 rules proposed by E.F. Codd to define a **perfect relational DBMS**. Example: Rule 1 (Information Rule): All data must be represented in table format.
Relational Integrity Constraints	Ensure correctness of data:
→ Domain Integrity	Attribute values must be from their specified **domains**.
→ Referential Integrity	**Foreign key** in one table must match a **primary key** in another (or be NULL).
→ Enterprise Constraints	Custom business rules, e.g., salary < manager's salary.
Database Design Goals	Minimize redundancy, ensure data integrity, support efficient query processing.
Good Relational Design Features	**Minimal redundancy**, **no anomalies**, **logical structure**, **normalization followed**.
Normalization	Process to reduce data redundancy and anomalies by decomposing relations.
Atomic Domains and 1NF	A table is in **1NF** if all attributes contain **atomic (indivisible)** values.
Functional Dependency (FD)	A relationship where **one attribute uniquely determines another**. E.g., RollNo → Name.
Decomposition Using FD	Splitting a table into smaller ones using FDs to remove anomalies.

Algorithms for Decomposition	Used to preserve **lossless join** and **dependency preservation** during normalization.
Second Normal Form (2NF)	Achieved when a table is in 1NF **and** has **no partial dependency**.
Third Normal Form (3NF)	In 2NF **and** has **no transitive dependency**.
Boyce-Codd Normal Form (BCNF)	Stronger version of 3NF: **every determinant must be a candidate key**.