DBMS

Quick Work
Page No.:
Date:
M T W T F S S

Unit-3

Relational Database Design

- Relational model - ensures data accuracy, consistency & easy retrieval.
  - It is a key concept in DBMS
  - It organizes data in a structured and efficient way by representing data & their relationships using tables.

- Terminology -
  - Each table is called a Relation.
  - The column of the table are called attributes
  - The rows of the table are called Tuples

- Management
  - It primarily uses SQL (Structured Query language) to manage & query data stored in tables with predefined relationships

- Relational Integrity -
  - Relational Integrity refers to a set of rules and constraints in a relational database
  - that ensures accuracy, consistency & validity of the data.
  - They are essential for maintaining a high quality relational database.

- They are mainly the foll" types of relational integrity constraints -
  1. Domain
  2. Entity
  3. Referential
  4. Enterprise

Quick Work

Page No.:
Date:
M T W T F S S

① Domain Constraints -

- It ensures that attributes (columns) in a relation (table) have valid values.
- Every column has a defined domain.
- DBMS enforces that values inserted/updated in a column must belong to its domain.
- The domain defines - data type, Range of values, format or pattern, specific set of values.

eg. Age must be an integer & greater than 0
email must follow a valid email format
Gender can only be male, female, other

```
CREATE TABLE Employee(
  EmpID INT Primary key,
  Name varchar (50) NOT NULL,
  Age INT CHECK (Age >18),
  Gender varchar (10) Check (Gender IN ('Male',
  'female', 'Other'))
);
```

② Entity Integrity Constraint -

- This is the rule about the Primary key of table
- Ensures each row in a table is unique.
- Primary key can Never be null & must be unique.
- If primary key is NULL or repeated, the table cannot uniquely identify rows, causing confusion & data inconsistencies

Quick Work

Page No.:
Date:
M T W T F S S

eg. Student table:

Roll_No (PK)	Name
101	Dea
102	Swarn
NULL	Yash

- NULL is not allowed because the row cannot be identified

⑧ Referential Integrity -

- Referential integrity ensures that relationships between tables remain consistent.
- This is enforced using foreign keys

Rules:-
- A foreign key value must match an existing Primary key value in the parent table.
- If a referenced record is deleted, the database must decide what to do -
  - CASCADE (delete child records)
  - SET NULL
  - RESTRICT (stop deletion)

Reason - This avoids 'orphan records' where a record points to a non-existing parent

eg.

```
Create table Department(
DeptID Int Primary key,
DeptName varchar(50)
);
```

```
Create table Employee(
EmpID Int Primary key,
Name varchar(50),
DeptID Int,
Foreign key (DeptID) References
Department(DeptID)
```

This ensures an employee cannot be assigned to a non-existent department

business rules

Quick Work
Page No.:
Date:
M T W T F S S

④ Enterprise (Business) Constraints -

- Enterprise constraints go beyond primary key, foreign key, or domain rules.
- They enforce real-life conditions that the business requires.
- The rules come from the policies, procedures, & requirements of the enterprise (company).

eg- A bank may say min balance must be ₹500
A company may say employee age must be atleast 18
A college may say attendance must be atleast 75%

- These rules are defined using -
  - CHECK constraints
  - Triggers
  - Stored procedures
  - Application logic
- These constraints ensure that the database supports the actual working rules of the organization.

• CODD'S RULES TWELVE [12]

- E.F. Codd proposed 12 rules (also called Codd's 12 commandments) which should be satisfied by the relational model.
- Codd's Rules are basically used to check whether DBMS has the quality to become RDBMS.

• Importance / significance -

- Provide a standard to judge whether a DBMS is truly relational.

Quick Work

Page No.:

Date:

M T W T F S S

- Ensure data consistency, integrity, correctness
- Support high-level non-procedural access (SQL)
- Promote data independence.
- Improve security, accuracy, & flexibility in data management
- Prevents DBMS from using only partial relational features

- In short, Codd's rules were designed to guarantee that a system follows proper relational principles & is not just "relational by name."

Rule 0 Foundation Rule -

A system must be able to manage databases entirely through its relational capabilities to be called an RDBMS.

eg- SQL server or MySQL uses tables, keys & relational operations (like JOIN) - so they qualify

Rule 1 Information Rule :-

- All data should be represented as values in tables.

eg- Student names, roll numbers, & marks are stored as values in rows & columns of table

Roll No	Name	Marks
101	Dea	85
102	Yash	92

Quick Work

Page No.:
Date:
M T W T F S S

Rule 2: Guaranteed Access Rule :-

Every data item must be accessible by table name, primary key, & column name

eg- You can access Dea's marks using
SELECT marks FROM students WHERE Roll no = 101;

Rule 3 Systematic Treatment of Nulls-
Null values must be treated consistently, representing
• missing value • unknown value • Not applicable value
- DBMS must support NULLs without breaking rules.

eg- If Yash's marks are not available, we store it as NULL

Rollno	Name	marks
102	Yash	NULL

Rule 4 Dynamic online Catalog
The database should have a catalog that can be queried using SQL

eg. You can query table structure using;
SELECT * FROM information_schema.tables;

Rule 5 Comprehensive Data Sub-language Rule
The system must support at least one language that supports data definition, manipulation, constraints, and access

eg. SQL can create tables, insert data, define constraints-
CREATE TABLE Students (....);

Quick Work

Page No.:
Date:
M T W T F S S

Rule 6 View Updating Rule

All views that can be updated theoretically should be updatable by the system

eg. Updating a view should update the base table, if logically possible.

Rule 7 :- High-level Insert, Update, Delete

You should be able to manipulate sets of data (not just one row at a time)

eg. Update all students who score less than 50

```
UPDATE Students SET Marks = Marks + 5 WHERE
Marks < 50;
```

Rule 8 :- Physical Data Independence

Changes in physical storage should not affect how data is accessed

eg.- moving a table's storage file shouldn't affect your SQL queries

Rule 9 - Logical Data Independence

Changes in logical schema should not break existing applications

eg.- Adding a new column like Email shouldn't break queries using only Name & Marks.

Quick Work

Page No.:
Date:
M T W T F S S

Rule 10 Integrity Independence -
Integrity constraints should be stored in the catalog & not in the application code.

eg- Defining a constraint in SQL, not in appl^n code:
ALTER TABLE Students ADD CONSTRAINT pk
Primary key (Roll no);

Rule 11 Distribution Independence
A user should not know whether the database is: Centralized, Distributed, Replicated.
The system must handle distribution automatically.

Rule 12 Non-Subversion Rule
- If a system provides low-level access, it must not bypass relational security & integrity rules

eg. You can't insert data that violates constraints using low-level tools like file editors or APIs

Anomaly & Normalization

Quick Work
Page No.:
Date:
M T W T F S S

- An anomaly is an error or irregular behavior in inserting, updating or deleting data because the table is not properly structured.

- When database is poorly designed, it may contain redundant & repeating data, which causes three major anomalies -

INSERT anomaly

- Consider the following table - Student_Course

StudentID	StudentName	CourseID	CourseName	Instructor
101	Amrita	CSE101	DBMS	Dr. Jadhav
102	Bob	CSE102	OS	Dr. Kadam
101	Amrita	CSE101	OS	Dr. Kadam

- This above table violates normalization rules & can lead to anomalies

(1) INSERT Anomaly :-
You cannot insert data into a table because some other data is missing.

eg. Suppose a new course is added (CSE103 - Networks' taught by Dr. Sapkal) but no student has enrolled yet.

∴ We can't insert this course because - The table requires a StudentID & StudentName, but we

don't have any student yet for this course.

- We are forced to insert NULLs or fake student data, which violates data integrity.

(2) UPDATE Anomaly -

Occurs when same data is repeated in multiple rows, & updating it in one place but not in others causes inconsistencies.

eg. Amruta is enrolled in two courses. Her name is stored in multiple rows -

StudentID	StudName	...
101	Amruta	
101	Amruta	

Suppose Amruta changes her name to 'Dea'.

- We need to update all rows where Amruta appears
- If we miss one, the database has inconsistent data

(3) DELETE Anomaly -

Occurs when deleting some data unintentionally causes loss of additional useful data.

eg. Suppose Bob drops all his courses we delete his record

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

- Bob's record is deleted, but also the course CSE102 taught by Dr. Kadam may no longer exist in table if no other student was enrolled in it.
- We lose course data, even though course might still be valid.

Quick Work
Page No.:
Date:
M T W T F S S

Solution - NORMALIZATION

- Overall Impact on Database Design -

- These anomalies lead to -
  - Redundancy
  - Inconsistency
  - Storage waste
  - Complex update operations
  - Poor reliability & bad database structure
  - Data loss due to delete anomalies

  Thus, database become incorrect, unreliable & inefficient.

- Solution - **NORMALIZATION**
  - To avoid these anomalies, we use normalization, which involves dividing data into logical tables & establishing relationships.

- Normalized form eg.

(1) Student Table

Student ID	StudentName
101	Anuta
102	Bob

(2)

Course Table	CourseID	CourseName	Instructor
	CSE101	DBMS	-11-
	CSE102	OS	

Quick Work
Page No.:
Date:
M T W T F S S

③ Enrollment Table

StudentID	CourseID
101	CSE101
102	CSE102
101	CSE102

- **How Normalization remove these anomalies :-**

- Normalization is a step-by-step process of organizing data in a database to remove redundancy and improve integrity.
- It uses functional dependencies to decide how to break tables.

= Normalization uses a series of rules called - Normal Forms :- 1NF, 2NF, 3NF, BCNF

- **1NF** [First Normal form] -

Rule ① Each column should have atomic values
② Each row must be unique
- Helps avoid insert anomalies because data is organized properly.

eg. Before 1NF -

Student ID	Name	subjects
1	Dea	Maths, Science
2	Yash	English, Hindi

After Applying 1NF

StudentID	Name	subject
1	Dea	Maths
1	Dea	Science
2	Yash	English
2	Yash	Hindi

Quick Work

Page No.:

Date:

M T W T F S S

- **2NF [Second Normal form]**

A table is in 2NF if -

Rules -

- It is already in 1NF &
- There is no partial dependency (non-key col depends on only one part of a composite primary key)
- Only applies to tables with composite primary keys.

eg. Table (Not in 2NF)

Student ID	Course ID	Student_Name	Course_Name
1	C1	Ashu	DBMS
1	C2	Ashish	OS

Primary key = {Student_ID, Course_ID}

StudentName depends only on Student_ID.
Course_Name depends only on Course_ID
Both are partial dependencies.

- Convert to 2NF

① Student Table | Student_ID | Student_Name |

② Course Table | Course_ID | Course_Name |

③ Enrollment Table | Student_ID | Course_ID |

Now, - No partial dependencies
- Proper separation of data
- The tables are in 2NF.

Quick Work

Page No.:
Date:
M T W T F S S

- **3NF (Third Normal form)**
  - A table is in 3NF if -
  - It is already in 2NF &
  - There is no transitive dependency. (Non key col should not depend on another non key col) only depend on PK

$A \to B$ &
$B \to C$
$A \to C$ TD.

eg. Not in 3NF -

Rollno (PK)	Student_Name	Class	Class Teacher
1	Tanu	TE	mrs. Patil
2	Swarn	TE	mrs. Patil

- transitive dependency - RollNo → Class → ClassTeacher

- Convert to 3NF -

T. ① student -

Rollno	student_Name	Class

T. ② Class -

Class	Class Teacher

- **BCNF (Boyce Codd Normal form)**
  - A stricter version of 3NF
  - In BCNF, the left side of every functional dependency must be a key.
  - If the left side is not a key, the relation is not in BCNF.

eg.

Teacher	Subject	Room
A	DBMS	101
B	OS	102
A	OS	103

Teacher → Room
Subject, room → Teacher
PK = (Teacher, Subject)

9004395223 7 /

Quick Work
Page No.:
Date:
M T W T F S S

Teacher → Room
Teacher is not a key
Violates BCNF

Convert to BCNF

T-1 | Teacher | Room |

T-2 | Subject | Teacher |

Now every functional dependency has a key on the left, so these tables are in BCNF

- Difference -

	3NF	BCNF
①	3NF is less strict	① BCNF is more strict
②	Allows some anomalies to remain	② Removes almost all anomalies
③	There should be no transitive dependency	③ Left side of functional dependency must be a key
④	Every relation in BCNF is also in 3NF	④ Every 3NF relation is not always BCNF
⑤	Easier to achieve	⑤ Harder to achieve
⑥	Removes most redundancy	⑥ Removes all redundancy

Quick Work

Page No.:
Date:
M T W T F S S

DECOMPOSITION -

- Decomposition is the process of breaking a relation (table) into two or more smaller relations in such a way that the data becomes easier to manage, redundancy reduces, & anomalies are removed.
- It's used primarily during normalization to ensure the database is well structured.

- Desirable Properties of Decomposition -
  A good decomposition must satisfy the following properties -

(1) Lossless - Join Property -
- No info should be lost during decomposition.
- After joining the decomposed tables, we should get back the exact original relation (table).

(2) Dependency Preservation -
- All functional dependencies of the original table should be preserved in the decomposed tables
- This help in easy enforcement of constraints without needing joins

(3) No Redundancy -
- The decomposition should minimize duplicate data & avoid anomalies.

(4) Higher Normal forms
- The resulting relations should ideally be in 2NF, 3NF, or BCNF, ensuring fewer anomalies & better structure.

When you split one big table into smaller tables, lossless condition ensures that no data is lost

Types- (1) Lossless Decomposition :-
A decomposition is lossless if joining the decomposed tables returns exactly the original data with no extra or missing tuples (rows).

- Lossless condition -
for decomposition $R \rightarrow R1, R2$:
$(R1 \cap R2) \rightarrow R1$ OR $(R1 \cap R2) \rightarrow R2$
- The common attributes (col) must be a superkey for at least one of the decomposed relations.

(2) Lossy Decomposition -
A decomposition is lossy when-
You lose some original information
OR You get extra (wrong) rows when you join the tables again.
- This happens when the common attributes are not superkey in either of the decomposed tables

Q. Relation - f (FN, PN, C, D)

Functional dependencies-
(1) FN, PN → C
(2) C → D
(3) D → F

Quick Work

Page No.:
Date:
M T W T F S S

Decompositions: F1 (FN, PN, C)
F2 (C, D)

Step1 - Find Common Attributes
F1 (FN, PN, C)
F2 (C, D)
Common attribute = C
Thus. R1 ∩ R2 = C

Step2 - Check if C is a key

F2 (C, D) :
FD : C → D
so C determines D
Therefore, C is a key for F2

Condition satisfied: (R1 ∩ R2) → R2
ie C → (C, D) is true

The decomposition is lossless

Reason - Because the common attribute C functionally determines all attributes of relation F2 (C → D), meaning C is a key for F2

Thus, the decomposition,
F → F1 (FN, PN, C) & F2 (C, D)
is lossless decomposition.

Quick Work

Page No.:
Date:
M T W T F S S

FUNCTIONAL DEPENDENCY (FD)

- Functional Dependency is a relationship between attributes (columns) in a relational database.
- When one column (attribute) in a table uniquely decides the value of another column, it is called a functional dependency
- This relationship is denoted as:

$$X \rightarrow Y$$

X is determinant (attribute on left side of FD)
Y is dependent attribute (attribute on right side of FD)

eg- Rollno → Name
Rollno = X → determinant
Name = Y → dependent attribute

- **Use of FD in database design** -
FD are used to design a good database tables.
FD's are useful in the foll^n ways-

① **Identifying keys** -
FD's help to find- Candidate keys
Primary keys
eg. If Roll NO → Name, Address, Branch
Then Roll no is candidate key

② **Removing Redundancy** -
FDs show where unnecessary repeating data exists. Using FDs we can split tables to

remove duplicate data.

(3) Avoiding Anomalies -
FDs help remove - Update, Insert, Delete anomalies
So the database becomes more consistent

(4) Normalization -
FDs are the basis of normalization -
1NF - remove repeating groups
2NF - remove partial dependency
3NF - remove transitive dependency
BCNF - Strengthen key dependencies
Every decomposition is done using FDs

Q. Consider the instance of the relation market (MarketName, Product, stock):

MarketName	Product	Stock
s1	Toothpaste	14
s1	Biscuits	8
s1	Shampoo	8
s2	Toothpaste	30
m1	Chocolates	50
m2	Cakes	14

Identify the functional dependencies that can be found in the given instance

→ To identify the functional dependencies we observe the repeating values in table,

Given - Market (MarketName, Product, Stock)

Observation - A market can have many products
A product can appear in many markets

Quick Work
Page No.:
Date:
M T W T F S S

Stock value is different for each Market-Product Pair

eg. Toothpaste in S1 → stock = 14
Toothpaste in S2 → stock = 30
Therefore, neither MarketName alone nor Product alone can determine stock

∴ valid FD - (MarketName, Product) → stock

The combination of MarketName & Product uniquely determines the stock value

∴ FD = {(MarketName, Product) → stock}

Q. Student (RollNo, Branch_code, Marks_Obtained, Exam_Name, Total_Marks)
Identify FD & chuck whether the given schema is in 3NF or no. If not justify & convert into 3NF

→

Step1 - Identify FDs :-

① RollNo → Branch_code

② (RollNo, Exame_Name) → Marks_obtained

③ Exam_Name → Total_Marks

Quick Work

Page No.:
Date:
M T W T F S S

Step 2:- Check for 3NF

FD1:- RollNO → Branch_code — 3NF OK
FD2:- (RollNo, Exam_Name) → Marks_obtained → 3NF OK
FD3:- Exam_Name → Total_Marks —
Not a key

Step 3 - Convert into 3NF

① Student Table Student(RollNO, Branch_code)
② Exam Table Exam(Exam_name, Total_Marks)
③ Marks table Marks(RollNO, Exam_name, Marks_Obtained)

- **FD Types -**

① **Full FD -**
A FD X → Y is full if Y depends on all (col) attributes of X, and not on any part of X

eg - Relation - Student(RollNO, Subject, Marks)
Candidate key - (RollNO, Subject)
FD: (RollNO, Subject) → Marks

Hence, it is a full functional dependency.

② **Partial FD:-**
A FD X → Y is Partial if Y depends on ~~at~~ only part of a composite key, not the whole key

eg - Relation - Student(RollNO, Subject, StudentName)

Quick Work

Page No.:
Date:
M T W T F S S

Composite key - (RollNo, Subject)
FD: RollNo → StudentName.
StudentName depends only on RollNo, not on subject
- This is a partial dependency.

(3) Transitive Dependency -
A dependency X → Z is transitive if
X → Y and Y → Z
& Y is not a key attribute

eg. Relation - Employee (EmpID, DepID, DeptName)
FDs - EmpID → DeptID
DeptID → DeptName
So, EmpID → DeptName through DeptID)
This is transitive dependency

LHS = {A,B}
RHS = {A}
RollNo → Name

(4) Trivial - If RHS is inside LHS
(5) Non-Trivial - If RHS is outside LHS.

Q. Features of good relational database design
minimal redundancy
No update anomalies
Proper use of functional dependencies
Proper normalization
Clear & simple schema
Data Integrity
efficient query Performance
Flexibility & Expandability
Strong Relationship Between Tables