Database Constraints and Normalization

4. a) Explain the working of triggers along with its syntax and an example. [5]
   b) What is a partial dependency? Define 2NF and 3NF with suitable examples. [1+2+2]
5. What are the different steps in

4. a) Define an extraneous attribute in a Functional dependency. Given a relation R = {A, B, C, D} and corresponding FDs: F = {A→BC, B→D, A→B, AB→C}. Find attribute closure $\{A\}^+$, $\{BC\}^+$. Which of these attribute deserve the property of candidate key? Justify it. [2+2+2]
   b) Define a partial and transitive functional dependency with example. How do you achieve a relation in a BCNF? Describe the algorithm. [3+3]

5. Suppose that we decompose the schema R = (A,B,C,D,E) into (A,B,C) and (C,D,E). Is it lossless decomposition? Does it preserve dependencies? Consider the following set F of functional dependencies hold A→BC; CD→E; B→D; E→A [2+2]
6. What is Normalization? Why is it important? Differentiate between 3NF and BCNF with suitable examples. [1+1+6]
7. Define query processing and briefly explain its steps. How is

4. a) What do you mean by closure of functional dependency? Explain about referential integrity constants and illustrate with suitable examples. [3+3]
   b) Wha is the purpose of Normalization? Explain 3NF and BCNF. [2+4]

[illegible] department. (Dptid='ELX')

5. a) What do you mean by closure of functional dependency? What is a trigger in DBMS? Is it safe or risky to use triggers? Explain. [3+3]
   b) Define normalization and levels of normalization 1NF, 2NF and 3NF. Compare the advantage of BCNF over 3NF. [4+2]
6. Explain the basic steps in

4. a) Explain the conditions to be satisfied for lossless decomposition using FD sets. [3]
   b) Define Assertion with its SQL syntax. What is a view in database system? How does materialized view differ from normal view? [2+2+1]
   c) Define Multi Valued Dependency (MVD). Explain Fourth normal form with an example. [2+3]
5. Describe the basic steps in

4. a) Why do we need normalization? Differentiate primary key and foreign key. Differentiate between 3NF and BCNF. [2+2+3]
   b) Consider the relation Treatment with the schema: Treatment (doctorID, doctorName, PatientID, diagnosis) and functional dependencies; [5]

   doctorID→doctorName and (doctorID, patientID)→diagnosis.

   Describe different types of problem that can arise for this relation with records

5. a) What is a functional dependency? List the various integrity constraints and explain about the referential integrity along with an example. [3+3]

b) Define 1NF, 2NF and 3NF. Illustrate your answer with suitable example. [6]

5. a) What are Triggers? Define Domain constraint and Referential Integrity constraint with an example. [1+4]

b) What is the role Functional dependencies in Normalization? Explain trivial and non-trivial dependencies. Explain BCNF. [2+2+3]

4. a) Define functional dependency. Explain partial and transitive functional dependency with example. [1+4]

b) Define decomposition and its desirable properties. Explain 3NF and BCNF. [3+4]

4. a) What do you mean by functional dependencies? Define formally. What is BCNF? [3+3]

b) What is normalization? Explain INF, 2NF, 3NF and 4NF. [2+4]

4. a) What is the advantages of 3NF over BCNF? Suppose that we decompose the scheme R = (A, B,C) into R1 = (A,B), R2 = (A,C). Show that this decomposition is a lossless join decomposition and not dependency preserving if the F = {A->B, B->C} [3+4]

b) What do you mean by integrity constraints? Explain any four constraints that can be enforced to database tables. [1+4]

4. a) What is lossless decomposition and dependency preservation? Suppose that we decompose the schema R = (A, B, C, D, E) into (A, B, C) and (C, D, E). Is it lossless decomposition? Is it dependency preserving? [3+4]
Consider that the following set F of functional dependencies hold.
$A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$

b) What is the importance of normalization? Define BCNF. [2+3]

5. a) What are integrity constraints in a database? Explain with example. What is a trigger in DBMS? When is it risky to use triggers? [3+3]

b) Define what a functional dependency is. Explain BCNF in terms of functional dependencies. [3+3]

5. a) Explain what is referential integrity constraint along with an example? Briefly explain cascading actions in referential integrity constraints. [3+3]

b) Briefly explain how to normalize a database from un-normalized form to 1NF, 2NF, 3NF and 4NF? [6]

5. a) Explain the necessary condition for decomposing a relational database table into two tables. Why is normalization needed? [4+4]

b) Compare 3NF and BCNF normal forms? [4]

4. a) What is a lossless-join decomposition? What is a functional dependency? Explain. [4+4]

b) What is the advantage of 3NF over BCNF? [4]

5. What do you mean by term functional dependency? Discuss various types of functional dependencies. [6]

4. What do you mean by integrity constrains? Explain any four constraints that can be enforced to database tables. [6]

5. What are the advantages of normalization of database? Explain 1NF, 2NF and 3NF. When database de-normalization is preferred? [2+3+1]

5. What is a functional dependency? Explain. What are the criteria for a relational schema to be in BCNF? [4+4]

5. Explain the conditions of BCNF. Compare BCNF and 3NF with example. [3+5]

Integrity Constraints

Integrity constraints are a set of rules applied to database tables to maintain data accuracy, consistency, and reliability. These rules ensure that database operations such as insertion, deletion, and updates do not lead to data corruption or inconsistency.

There are four main types of integrity constraints:

1. Domain Constraint

-> Ensures that the values in a column belong to a specific data type or domain.

-> Each column in a table must have values within a specific range, data type, or format.

Example:

Consider a Students table:

StudentID	Name	Age	Gender
101	Sushant	21	M
102	Aditi	19	F
103	Rahul	22	X

Invalid

Constraints Applied:

-> Only ages between 18 and 30 are allowed.

-> Only 'M' or 'F' is allowed.

Invalid Case: The third row contains 'X' in the Gender column, violating the domain constraint.

2. Entity Integrity Constraint

-> Ensures that each row in a table has a unique and non-null primary key.

-> Prevents duplicate or null values in the primary key column.

-> Each row is uniquely identified and cannot have NULL values.

Example:

Consider a Customers table:

CustomerID(PK)	Name	Email
1	Ram	ram@email.com
2	Sita	sita@email.com
NULL	Hari	hari@email.com

Invalid

Constraints Applied:

->Each CustomerID must be unique and not NULL.

Invalid Case: The third row has NULL in CustomerID, violating the entity integrity constraint.

3. Referential Integrity Constraint

-> Ensures that a foreign key in one table refers to a valid primary key in another table.

-> Prevents orphaned records (i.e., records that reference a non-existent entity).

Example:

CustomerID (PK)	Name
1	Ram
2	Sita

OrderID	CustomerID (FK)	Product
101	1	Laptop
102	2	Phone
103	3 ✗	Tablet

Invalid Case: The third row in Orders has CustomerID = 3, but there is no such CustomerID in the Customers table, violating referential integrity.

4. Key Constraint

-> Ensures that a candidate key (Primary Key or Unique Key) uniquely identifies a record.

-> A Primary Key must be unique and not NULL.

-> A Unique Key must be unique, but can have NULL values.

Example:

Consider a Users table:

UserID (PK)	Username	Email (Unique)
1	user1	user1@email.com
2	user2	user2@email.com
3	user3	user1@email.com ✗ (Invalid - Duplicate Email)

Constraints Applied:

-> UserID INT PRIMARY KEY

-> Email VARCHAR(100) UNIQUE

Invalid Case: The third row has a duplicate email (user1@email.com), violating the Unique Key constraint.

Triggers

-> A trigger is a database object that automatically executes a specified action when certain events (such as INSERT, UPDATE, or DELETE) occur on a table.

-> Triggers execute automatically when the specified event occurs.

-> Triggers are written using SQL

Example of a Trigger

-> Let's say we want to prevent inserting a negative salary in an employees table.

-> This trigger runs before inserting or updating an employees record.

-> If the new salary is negative, it prevents the operation.

Assertions

->An assertion is a database constraint that ensures a certain condition holds true for the entire database at all times.

-> Assertions apply to the entire database, not just a single table.

-> If an assertion's condition is violated, the operation causing the violation is rejected.

Example of an Assertion

-> Let's say we want to ensure that the total salary expense in a company does not exceed $1,000,000.

-> assertion prevents inserting or updating salaries if the total salary crosses $1,000,000.

```
CREATE TRIGGER prevent_negative_salary
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
WHEN (NEW.salary < 0)
BEGIN
    RAISE EXCEPTION 'Salary cannot be negative';
END;
```

Example of trigger

```
CREATE ASSERTION total_salary_limit
CHECK (
    (SELECT SUM(salary) FROM employees) <= 1000000
);
```

assertion example

Risks of Using Triggers

-> Triggers execute automatically on every affected row, which can slow down database operations.

-> If not optimized, triggers can increase query execution time, especially on large tables.

-> If multiple triggers modify the same table, this may create circular dependencies.

-> Poorly written triggers can allow unintended data modifications.

-> If multiple triggers fire on the same table, tracking which one caused an issue can be challenging

Functional Dependencies

-> A functional dependency (FD) in a database management system (DBMS) is a relationship between two attributes (or sets of attributes) in a relation (table).

-> It expresses a constraint that the value of one attribute uniquely determines the value of another attribute.

-> If X and Y are attributes of a relation, then X → Y (X determines Y) means that for every unique value of X, there is only one corresponding value of Y.

Example

Consider a Student table:

Student_ID	Name	Department
101	Alex	CSE
102	Bob	ECE
103	Charlie	CSE

Here, Student_ID → Name because each Student_ID uniquely determines a Name.

Composite Determinant:

-> A determinant requiring multiple attributes to uniquely determine another attribute.

-> Example: In a table with (Student_ID, Course) → Grade, both Student_ID and Course form a composite determinant.

Full Functional Dependency:

-> Occurs when all attributes in a composite determinant are needed to uniquely identify the dependent attribute.

-> Formula: If A→B, and no proper subset of A can determine B, then B is fully dependent on A.

-> Example: (Student_ID, Course) → Grade (cannot determine Grade using only Student_ID or Course alone).

Partial Functional Dependency:

-> Occurs when a subset of a composite determinant is sufficient to determine the dependent attribute.

-> Example: In (Student_ID, Course) → Student_Name, Student_ID alone determines Student_Name, making it a partial dependency.

Types of Functional Dependencies (FDs)

1. Trivial Functional Dependency:

-> A dependency Y → X is trivial if X is a subset of Y.

-> Example: {Student_ID, Name} → Name is trivial because Name is already part of the determinant.

2. Non-Trivial Functional Dependency:

-> A dependency Y → X is non-trivial if X is not a subset of Y.

-> Company → CEO (non-trivial, as CEO is not a subset of Company).

Table:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46

3. Transitive Dependency

-> If X → Y and Y → Z, then X → Z is a transitive dependency.

-> Example:

Student_ID → Name

Name → Department

So, Student_ID → Department (transitive dependency).

4. Multivalued Dependency (MVD)

-> Occurs when an attribute determines multiple independent values.

-> Notation: X→→Y (read as "X" multidetermines "Y").

Example:

Name	Project	Hobby
Yuvaraj	MS	Reading
Yuvaraj	Oracle	Music

MVDs:

Name →→ Project (Yuvaraj has multiple projects).

Name →→ Hobby (Yuvaraj has multiple hobbies).

5. Join Dependency (JD)

-> A relation R satisfies a JD if it can be losslessly decomposed into smaller relations, and rejoining them reconstructs R

-> Example:

- Original relation: $R(A, B, C, D)$
- Decompositions: $R_1(A, B, C, D)$ and $R_2(C, D)$
- If $R_1 \bowtie R_2 = K$, then R satisfied JD.

Closure of a Set of FDs (F^+)

-> The closure F^+ is the set of all possible FDs that can be derived from a given set of FDs, using inference rules (e.g., Armstrong's Axioms).

Extraneous Attribute

-> An extraneous attribute in a functional dependency (FD) is an attribute that can be removed from the left-hand side (LHS) or right-hand side (RHS) of an FD without changing the closure.

Example: $AB \to C$, if A can be removed (i.e $B \to C$ is already derivable) then A is extraneous

Closure Example:

Given $R = \{A, B, C, D\}$ and $F = \{A \to BC, B \to D, A \to B, AB \to C\}$

closure of $\{A\}^+$

1. start $\{A\}^+ = \{A\}$
2. Apply $A \to BC$: $\{A\}^+ = \{A, B, C\}$
3. Apply $B \to D$: $\{A\}^+ = \{A, B, C, D\}$
4. no further attributes can be added

So, $\{A\}^+ = \{A, B, C, D\}$

→ A determines all attributes and it is minimal (no subset of A can act as a key)

So, $\{A\}$ is the candidate key

Closure of $\{BC\}^+$

1. Start : $\{BC\}^+ = \{B, C\}$
2. Apply $B \rightarrow D$: $\{BC\}^+ = \{B, C, D\}$
3. no FDs can derive A

So, $\{BC\}^+ = \{B, C, D\}$

→ missing A, so BC cannot determine all attributes, so $\{BC\}$ is not a candidate key

---

Normalization

Normalization in DBMS (Database Management System) is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves decomposing large tables into smaller ones and establishing relationships among them using primary keys and foreign keys.

🔷 Goals of Normalization

-> Eliminate redundancy
-> Ensure data consistency
-> Improve efficiency
-> Simplifies Extensions
-> Saves Storage
-> Avoids Anomalies: Prevents insertion/update/deletion issues.

Stages of Normalization

1. First Normal Form (1NF)

A table is in First Normal Form (1NF) if it satisfies the following conditions:

-> **Atomic Values**: Each column contains no multi-valued or composite attributes.

-> **Unique Rows**: Each row is uniquely identifiable (via a primary key).

-> **No Repeating Groups**: Eliminate repeating groups/columns by splitting multi-value cells into separate row

Columns

Examples:

ID	Name	Subject
101	Alice	Math, Science
102	Bob	History, art, English

(non-1NF -> multi-valued cell)

ID	Name	Subject
101	Alice	Math
101	Alice	Science
102	Bob	History
102	Bob	art
102	Bob	English

1NF form

P.K (ID, subject)

repeating groups

ID	Name	Phone1	Phone2
201	John	555-01	555-02

↓ 1NF

ID	Name	Phone
201	John	555-01
201	John	555-02

P.K (ID, Phone)

Advantages of 1NF

✓ Prevents data redundancy.

✓ Makes searching and filtering more efficient.

✓ Ensures data integrity by maintaining atomicity.

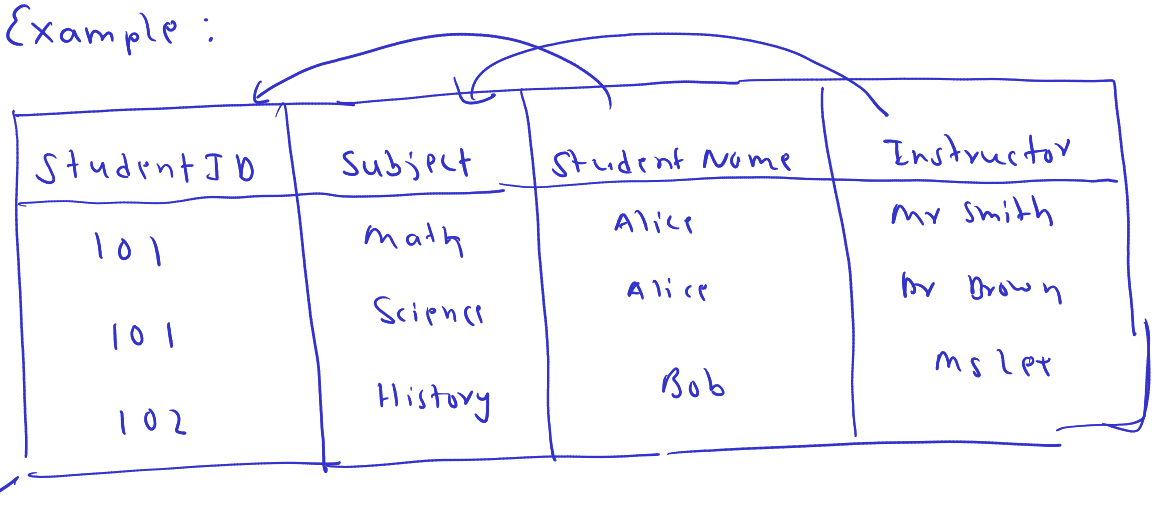
Second Normal Form (2NF)

A table is in Second Normal Form (2NF) if:

-> It is already in 1NF.

-> Have no partial dependency: Every non-key attribute must depend on the entire primary key, not just a part of it.

Example :



StudentID	Subject	Student Name	Instructor
101	Math	Alice	Mr Smith
101	Science	Alice	Mr Brown
102	History	Bob	Ms Lee

P.K must be unique

P.K → (StudentID, subject)

(1NF - compliant but not 2NF)

StudentName depends on studentID (partial dependency)

Instructor depends on subject (partial dependency)

-> Split the table to remove partial dependencies.

Table 1 : Students

Student ID	Name
101	Alice
102	Bob

PK : {ID}

Table 2 : subjects

Subject	Instructor
Math	Mr Smith
Science	Dr. Brown
History	Ms. Lee

PK : {subject}

Table 3 :

Student ID	Subject
101	math
101	science
102	History

PK : {ID, subject)

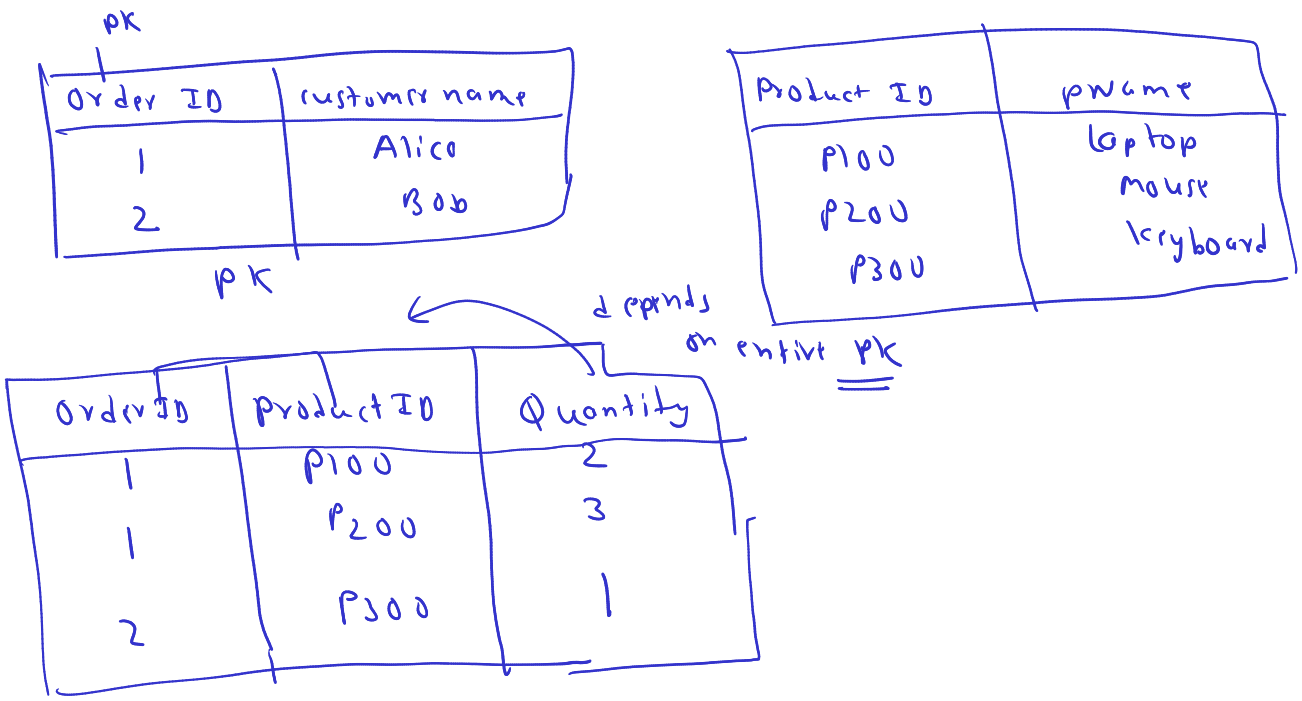
→ no partial dependencies

Order ID	Product ID	PName	Quantity	Customer Name
1	P100	laptop	2	Alice
1	P200	mouse	2	Alice
2	P300	keyboard	3	Bob

PK : (order ID, Product ID)

Customer Name → Order ID
(only depends on)
(partial dependency)

PName → Product ID
Only depends on



Advantages of 2NF

✓ Eliminates partial dependency, ensuring better data consistency.

✓ Reduces data duplication and improves database efficiency.

✓ Makes updates easier (If an PName changes, it needs to be updated in one place only).

Third Normal Form (3NF)

A table is in Third Normal Form (3NF) if:

-> It is already in 2NF.

-> **No Transitive Dependency:** Non-key attributes must depend directly on the primary key, not on other non-key attributes.

Transitive Dependency: When a non-key attribute depends on another non-key attribute.

Example :

depend

Employee ID	Name	Dpart	Dpart location
101	Alice	Sales	New York
102	Bob	HK	Los Angeles
103	Charlie	Sales	New York

key attribute

p.k → Employee ID (uniquely identifies others)

So, it is in 2NF

→ Dpart location depends on Dpart

(non key attribute depends on another non key)
not directly on Employee ID
(transitive dependency)

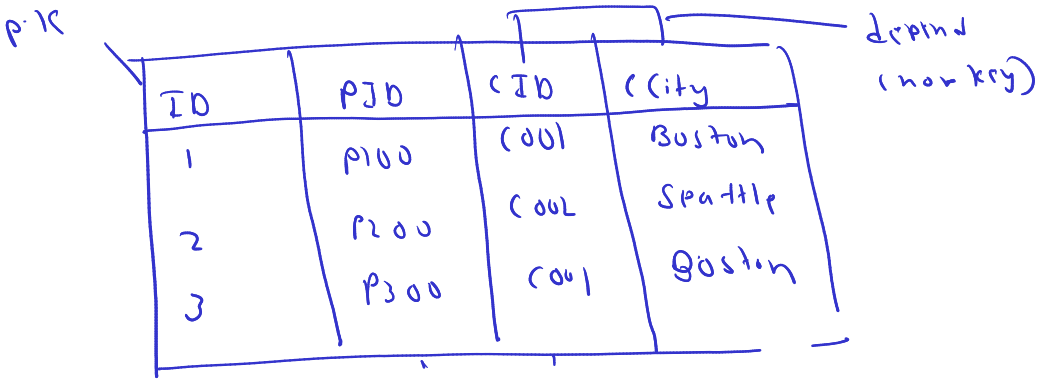
-> Split the table to remove transitive dependencies.

p.k

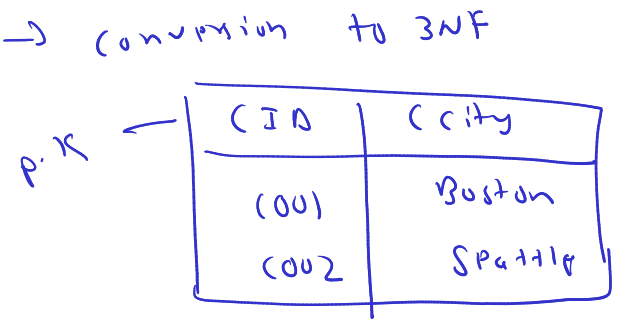
Department	Department location
Sales	New York
HR	Los Angeles

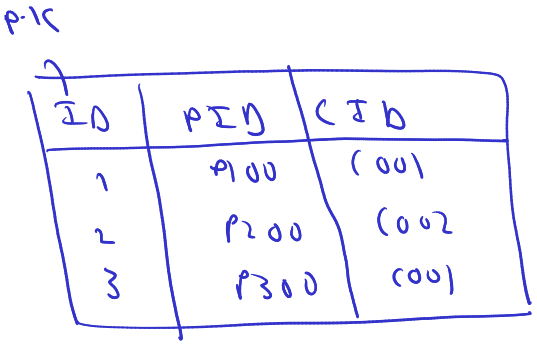
p.k

Employee ID	Name	Dpart
101	Alice	Sales
102	Bob	HR
103	Charlie	sales



→ conversion to 3NF





Advantages of 3NF

- ✓ Removes transitive dependencies, reducing redundancy.
- ✓ Improves data integrity and ensures better organization.
- ✓ Easier to update (If an HOD changes, update only in the Departments table).

Boyce-Codd Normal Form (BCNF)

→ advanced 3NF

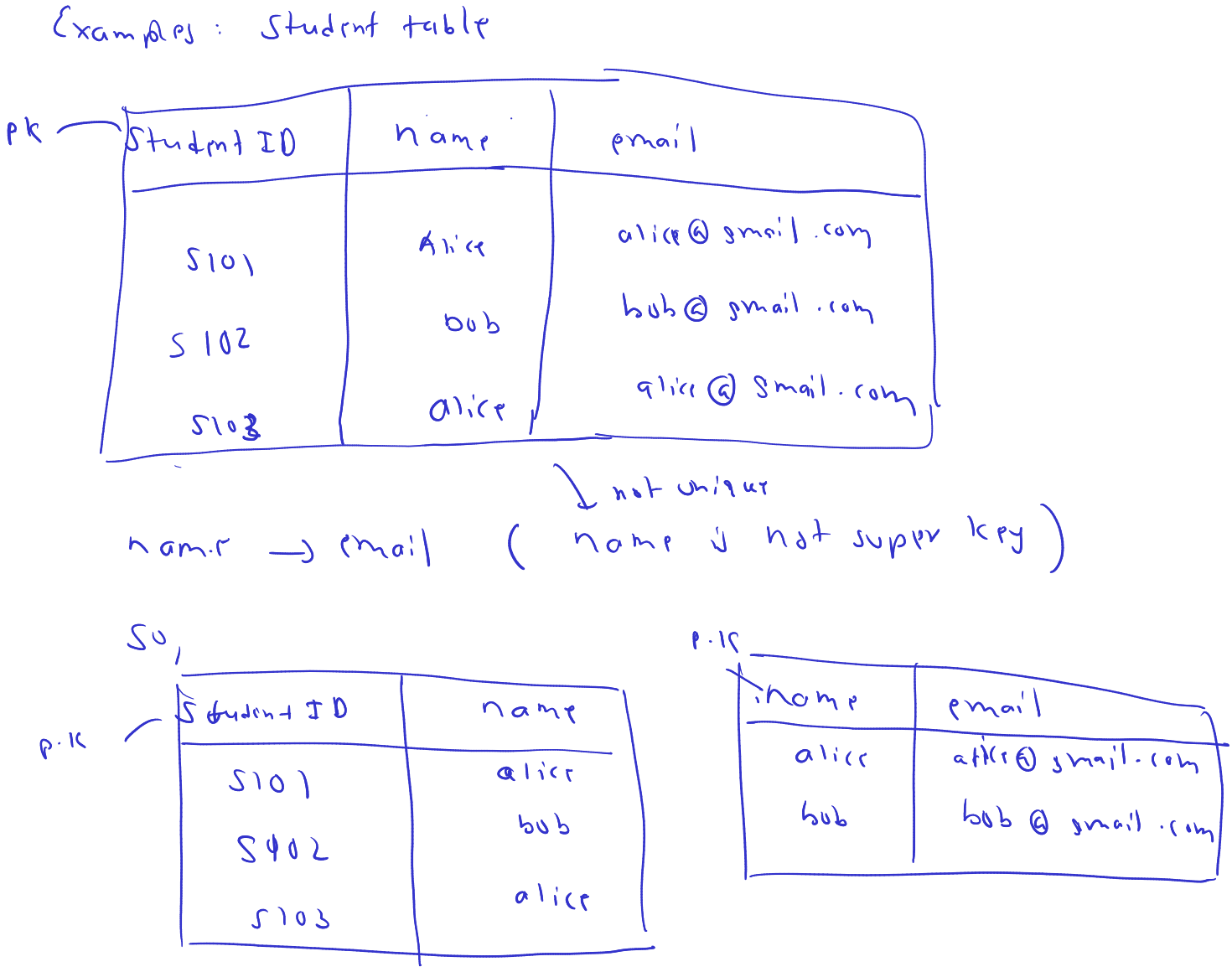
For a table to be in BCNF, it must:

-> It is already in 3NF.

-> No Non-Trivial Dependencies i.e Every determinant must be a super key.

(A determinant is any attribute that determines another attribute.)

-> Every non-trivial functional dependency X→Y must have X as a super key (i.e., X uniquely determines all other attributes in the table).



Key Takeaways for BCNF:

1. Stricter than 3NF: BCNF eliminates all non-trivial dependencies where the determinant is not a super key.

2. Super Key Requirement: Every determinant (left side of a functional dependency) must be a super key.

3. No Overlapping Dependencies: Fixes cases where a non-key attribute determines another attribute.

Difference Between 3NF and BCNF

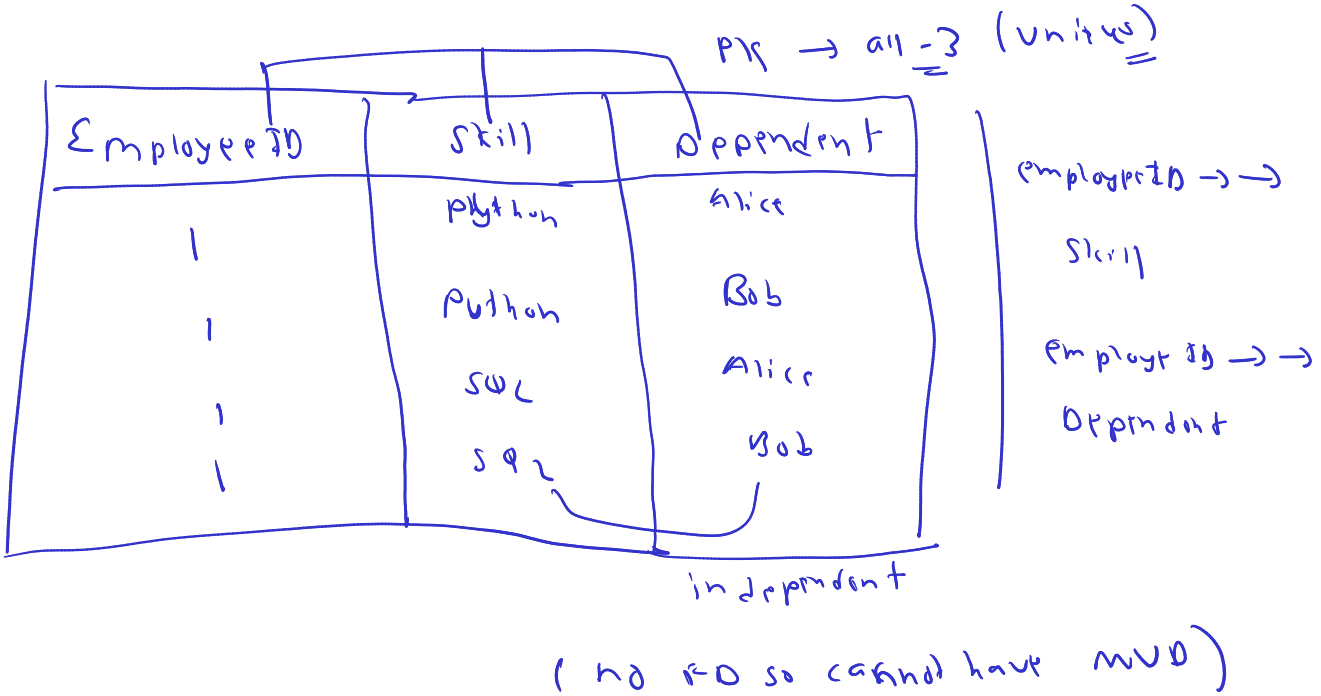
1. Must be in **2NF** 2. **No transitive dependency** (Non-key attributes must depend **only** on the primary key)	1. Must be in **3NF** 2. Every **determinant** must be a **super key**
Suitable for **most practical applications** where transitive dependencies are an issue	Used in **complex cases** where multiple candidate keys create redundancy
Easier to achieve and implement	More restrictive and requires more decomposition
If a **non-key attribute depends on another non-key attribute**, 3NF is needed	If **a candidate key determines another candidate key**, BCNF is required
Less strict	More strict (Stronger version of 3NF)

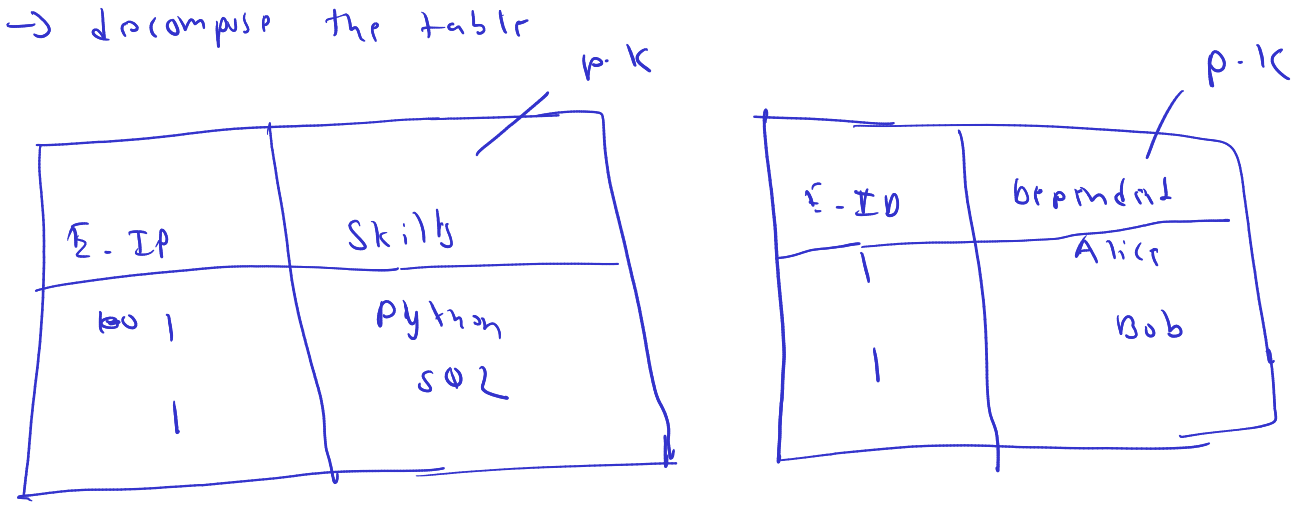
Fourth Normal Form (4NF)

-> The relation must be in BCNF (Boyce-Codd Normal Form).

-> It does not have multi-valued dependencies unless they are functional dependencies.

(A multi-valued dependency occurs when one attribute in a table determines multiple independent values of another attribute.)





Domain-Key Normal Form (DKNF)

-> DKNF (or DCNF) is the highest level of normalization. A table is in DKNF if all constraints and dependencies can be enforced by domain constraints and key constraints alone, meaning there are no additional integrity constraints.

Example (Before DKNF)

Consider a Loan Application table:

Loan_ID	Customer_Name	Loan_Amount	Interest_Rate
201	Ram	5000	5%
202	Sita	12000	10%

Business Rule: If Loan_Amount > 10000, Interest_Rate must be at least 8%.

This is not DKNF because this business rule cannot be enforced by keys or domain constraints alone.

Relational Decomposition in DBMS

Relational decomposition is the process of breaking a large relation (table) into smaller, more manageable relations to eliminate redundancy and anomalies while preserving data integrity. Decomposition can be either lossless or lossy.

1. Lossless Decomposition

Lossless decomposition ensures that no information is lost when a relation is decomposed into two or more relations. After performing a natural join operation on the decomposed relations, we should be able to reconstruct the original relation without losing any data.

A decomposition of $R(A,B,C)$ into $R_1(A,B)$ and $R_2(B,C)$ is lossless if

$$R_1 \cap R_2 \rightarrow R_1 \quad \text{or} \quad R_1 \cap R_2 \rightarrow R_2$$

2. Lossy Decomposition

Lossy decomposition means that when we decompose a relation into smaller relations, we lose some original data, making it impossible to perfectly reconstruct the original relation.

Some Old Questions Not in Note

mani ico Qn ma garya cha

Suppose that we decompose the schema R = (A,B,C,D,E) into (A,B,C) and (C,D,E). Is it lossless decomposition? Does it preserve dependencies? Consider the following set F of functional dependencies hold A→BC; CD→E; B→D; E→A [2+2]

What is Normalization? Wh...

Soln:

$R_1(ABC)$ $R_2((,D,E)$

FDs

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

we know

for lossless

$R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

$R_1 \cap R_2 = \{C\}$

So we must check $\{C\}^+ \rightarrow$ closure of $\{C\}$

$\{C\}^+ = \{C\}$ (no FDs with $C \rightarrow$ SU)

SU, it doesn't include all the attributes of R_1 (A, B, C) ↓ only C

or R_2 (C, D, E) → only C

So it is not lossless it lossy

a) What is lossless decomposition and dependency preservation? Suppose that we decompose the schema R = (A, B, C, D, E) into (A, B, C) and (C, D, E). Is it lossless decomposition? Is it dependency preserving? [3+4]
Consider that the following set F of functional dependencies hold.
A → BC
CD → E
B → D
E → A

A decomposition is dependency preserving if the functional dependencies in the original relation can be enforced without computing a join between the decomposed relation

Sum Qn as above → not lossless

$A \rightarrow BC$ (ABC in R_1 so preserved)

$CD \rightarrow E$ (CDE in R_2 so preserved)

$B \rightarrow D$ (B in R_1 and D in R_2) not preserved

$E \rightarrow A$ (E in R_2, A in R_1 not preserved)

Since, we lose $B \rightarrow D$ and $E \rightarrow A$, this decomposition is not dependency preserving.

a) What is the advantages of 3NF over BCNF? Suppose that we decompose the scheme R = (A, B,C) into R1 = (A,B), R2 = (A,C). Show that this decomposition is a lossless join decomposition and not dependency preserving if the F = {A->B, B->C} [3+4]

$R_1 = (A, B)$ $R_2 = (A, C)$

$R_1 \cap R_2 = A$

$\{A\}^+ = \{A\}$

$\{A\}^+ = \{AB\}$ $A \to B$

$\{A\}^+ = \{ABC\}$ $B \to C$

$\therefore R_1 \cap R_2 \to R_1$ contains all attributes of $R_1 (A, B)$ or $R_2 (A, C)$

or $R_1 \cap R_2 \to R_2$

(So it is lossless)

$A \to B$ (A, B in R_1) → preserved

$B \to C$ (B in R_1 and C in R_2) → not preserved

$B \to C$ not preserved it is not dependency preserving.

Explain the conditions to be satisfied for lossless decomposition using FD sets. [3]

Define Assertion with its SQL

(Explain like above,

$R_1 \cap R_2 \to R_1$ or $R_1 \cap R_2 \to R_2$

$R(A, B, C)$ $R_1(A, B)$ $R_2(A, C)$ F.D

$A \to B$

$B \to C$

then lossless

Consider the relation Treatment with the schema: Treatment (doctorID, doctorName, PatientID, diagnosis) and functional dependencies; [5]

doctorID→doctorName and (doctorID, patientID)→diagnosis.

Describe different types of problem that can arise for this relation with records

1. Since the functional dependency doctorID → doctorName means that every record for a given doctor will include the same doctorName. This redundancy increases storage requirements and the risk of inconsistencies.

2. If a doctor's name changes (or is corrected), every record with that doctorID must be updated.

3. Adding a new doctor who hasn't yet seen any patients is problematic because the primary key (doctorID, patientID) requires a patient ID.

4. If the only record(s) linking a doctor to a patient is deleted (for example, if a patient leaves the system), then the doctor's name might also be lost from the relation—even though the doctor is still practicing

5. Because the diagnosis depends on both doctorID and patientID ((doctorID, patientID) → diagnosis), any error in recording either attribute can lead to incorrect or missing diagnosis data.

Define Assertion with its SQL syntax. What is a view in database system? How does materialized view differ from normal view? [2+2+1]

Feature	Normal View	Materialized View
Storage	No physical storage	Physically stored
Query Execution	Executes SQL every time	Precomputed and stored result
Performance	Slower for complex queries	Faster since data is precomputed
Data Freshness	Always up-to-date	Needs refresh (manual or scheduled)
Use Case	Used for logical abstraction	Used for performance optimization