# DBMS Unit-04

By:
Mani Butwall,
Asst. Prof. (CSE)

## Contents

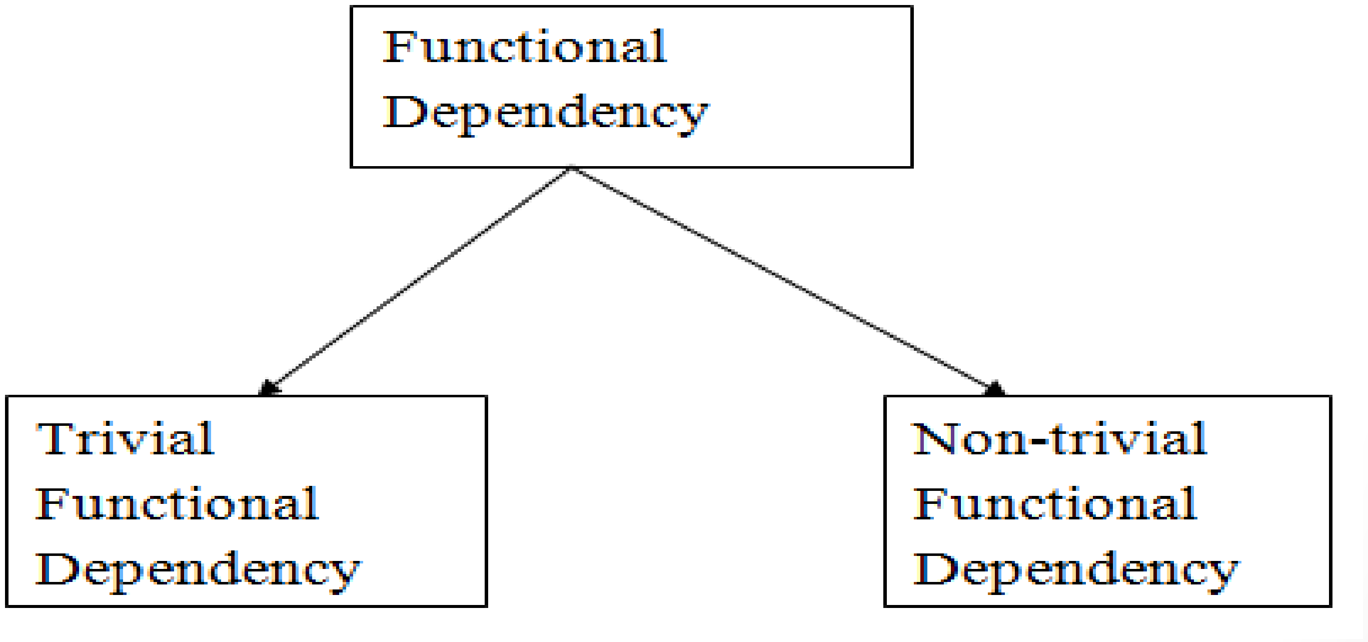
- Relational database design
- Domain and data dependency
- Armstrong's axioms
- Normal forms
- Dependency preservation
- Lossless design

# Functional Dependency

- The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.
- **For example:** Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it.
- This is known as functional dependency and can be written as **Stu_Id->Stu_Name**

or in words we can say Stu_Name is functionally dependent on Stu_Id.

- **Formally**:
  If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)


Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

# Trivial

- The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute.
- **Symbolically**: A ->B is trivial functional dependency

  if B is a subset of A.
- The following dependencies are also trivial: A->A & B->B

- **For example**: Consider a table with two columns Student_id and Student_Name.
- **{Student_Id, Student_Name} -> Student_Id**

is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}.  That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined.

- Also,
- **Student_Id -> Student_Id**
- **Student_Name -> Student_Name**

are trivial dependencies too.

# Non Trivial

- If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.
- **For example**:
  An employee table with three attributes:
  emp_id, emp_name, emp_address.

The following functional dependencies are non-trivial:
**emp_id -> emp_name (emp_name is not a subset of emp_id)**
**emp_id -> emp_address (emp_address is not a subset of emp_id)**

- On the other hand, the following dependencies are trivial:
  **{emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]**

- **Completely non trivial FD**:
  If a FD X->Y holds true where X intersection Y is null then this dependency is said to be completely non trivial function dependency.

## Multivalued

- Multivalued dependency occurs when there are more than one **independent** multivalued attributes in a table.
- **For example**: Consider a bike manufacture company, which produces two colors (Black and white) in each model every year.

| bike_model | manuf_year | color |
|---|---|---|
| M1001 | 2007 | Black |
| M1001 | 2007 | Red |
| M2012 | 2008 | Black |
| M2012 | 2008 | Red |
| M2222 | 2009 | Black |
| M2222 | 2009 | Red |

- Here columns manuf_year and color are independent of each other and dependent on bike_model.
- In this case these two columns are said to be multivalued dependent on bike_model.
- These dependencies can be represented like this:
  - **bike_model ->> manuf_year**

- **Example**
- Let us see an example &mins;
- **<Student>**

| StudentName | CourseDiscipline | Activities |
|---|---|---|
| Amit | Mathematics | Singing |
| Amit | Mathematics | Dancing |
| Yuvraj | Computers | Cricket |
| Akash | Literature | Dancing |
| Akash | Literature | Cricket |
| Akash | Literature | Singing |

In the above table, we can see Students **Amit** and **Akash** have interest in more than one activity.
This is multivalued dependency because **CourseDiscipline** of a student are independent of Activities, but are dependent on the student.
Therefore, multivalued dependency –

- StudentName ->-> CourseDiscipline
  StudentName ->-> Activities

## Transitive

- A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies.
- For e.g.

**X -> Z** is a transitive dependency if the following three functional dependencies hold true:

1. X->Y
2. Y does not ->X
3. Y->Z

- **Note:** A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF (3rd Normal Form).

| Book | Author | Author_age |
|---|---|---|
| Game of Thrones | George R. R. Martin | 66 |
| Harry Potter | J. K. Rowling | 49 |
| Dying of the Light | George R. R. Martin | 66 |

1. {Book} ->{Author} (if we know the book, we knows the author name)
2. {Author} does not ->{Book}
3. {Author} -> {Author_age}

- Therefore as per the rule of **transitive dependency**:

{Book} -> {Author_age} should hold,

that makes sense because if we know the book name we can know the author’s age.

# Closure of Attribute

- **Step-01:**
- Add the attributes contained in the attribute set for which closure is being calculated to the result set.
- **Step-02:**
- Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

- **Example-**
- Consider a relation R ( A , B , C , D , E , F , G ) with the functional dependencies-

1. $A \rightarrow BC$
2. $BC \rightarrow DE$
3. $D \rightarrow F$
4. $CF \rightarrow G$

- Now, let us find the closure of some attributes and attribute sets-

- <u>**Closure of attribute A-**</u>

- $A^+$ = { A }
- = { A , B , C } ( Using $A \rightarrow BC$ )
- = { A , B , C , D , E } ( Using $BC \rightarrow DE$ )
- = { A , B , C , D , E , F } ( Using $D \rightarrow F$ )
- = { A , B , C , D , E , F , G } ( Using $CF \rightarrow G$ )

Thus,

- **$A^+$ = { A , B , C , D , E , F , G }**

- **Closure of attribute set {B, C}-**

- $\{B, C\}^+ = \{B, C\}$
- $= \{B, C, D, E\}$ ( Using $BC \rightarrow DE$ )
- $= \{B, C, D, E, F\}$ ( Using $D \rightarrow F$ )
- $= \{B, C, D, E, F, G\}$ ( Using $CF \rightarrow G$ )

Thus,

- **$\{B, C\}^+ = \{B, C, D, E, F, G\}$**

- **Closure of attribute D-**

- $D^+$ = { D }
- = { D , F } ( Using $D \rightarrow F$ )
- We can not determine any other attribute using attributes D and F contained in the result set.

Thus,

- **$D^+$ = { D , F }**

# Finding the Keys Using Closure

- **Super Key-**
- If the closure result of an attribute set contains all the attributes of the relation, then that attribute set is called as a super key of that relation.
- Thus, we can say-
- **"The closure of a super key is the entire relation schema."**

- **Example-**
- In the above example,
- The closure of attribute A is the entire relation schema.
- Thus, attribute A is a super key for that relation.

- <u>**Candidate Key-**</u>
- If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.
- <u>**Example-**</u>
- In the above example,
- No subset of attribute A contains all the attributes of the relation.
- Thus, attribute A is also a candidate key for that relation.

- Consider the given functional dependencies-

1. $AB \rightarrow CD$
2. $AF \rightarrow D$
3. $DE \rightarrow F$
4. $C \rightarrow G$
5. $F \rightarrow E$
6. $G \rightarrow A$

- Which of the following options is false?
- (A) $\{ CF \}^+ = \{ A , C , D , E , F , G \}$
- (B) $\{ BG \}^+ = \{ A , B , C , D , G \}$
- (C) $\{ AF \}^+ = \{ A , C , D , E , F , G \}$
- (D) $\{ AB \}^+ = \{ A , C , D , F ,G \}$

- **Solution-**
- 
- Let us check each option one by one-
- 
- **Option-(A):**
- 
- $\{ CF \}^+ = \{ C , F \}$
- $= \{ C , F , G \}$ ( Using $C \rightarrow G$ )
- $= \{ C , E , F , G \}$ ( Using $F \rightarrow E$ )
- $= \{ A , C , E , E , F \}$ ( Using $G \rightarrow A$ )
- $= \{ A , C , D , E , F , G \}$ ( Using $AF \rightarrow D$ )
- 
- Since, our obtained result set is same as the given result set, so, it means it is correctly given.

- <u>Option-(B):</u>
-
- $\{BG\}^+ = \{B, G\}$
- $= \{A, B, G\}$ ( Using $G \rightarrow A$ )
- $= \{A, B, C, D, G\}$ ( Using $AB \rightarrow CD$ )
-
- Since, our obtained result set is same as the given result set, so, it means it is correctly given.

- <u>**Option-(C):**</u>
- 
- $\{ AF \}^+ = \{ A , F \}$
- $= \{ A , D , F \}$ ( Using $AF \rightarrow D$ )
- $= \{ A , D , E , F \}$ ( Using $F \rightarrow E$ )
- 
- Since, our obtained result set is different from the given result set, so,it means it is not correctly given.
-

- **Option-(D):**
-
- $\{ AB \}^+ = \{ A , B \}$
- $= \{ A , B , C , D \}$ ( Using $AB \rightarrow CD$ )
- $= \{ A , B , C , D , G \}$ ( Using $C \rightarrow G$ )
-
- Since, our obtained result set is different from the given result set, so,it means it is not correctly given.

- Thus,
- **Option (C) and Option (D) are correct.**

- **Example-2 :** Consider a relation R(A,B,C,D,E) having below mentioned functional dependencies.

1. FD1 : A -> BC
2. FD2 : C -> B
3. FD3 : D -> E
4. FD4 : E -> D

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

- **Example-1 :** Consider the relation R(A,B,C) with given functional dependencies :
- FD1 : A -> B
- FD2 : B -> C

- Now, calculating the closure of the attributes as :
- $\{A\}^+ = \{A, B, C\}$
- $\{B\}^+ = \{B, C\}$
- $\{C\}^+ = \{C\}$
- Clearly, “A” is the candidate key as, its closure contains all the attributes present in the relation “R”.

- **Example-2 :** Consider another relation R(A, B, C, D, E) having the Functional dependencies :
- FD1 : A  BC
- FD2 : C  B
- FD3 : D  E
- FD4 : E  D
-

- Now, calculating the closure of the attributes as :
- $\{A\}^+ = \{A, B, C\}$
- $\{B\}^+ = \{B\}$
- $\{C\}^+ = \{C, B\}$
- $\{D\}^+ = \{E, D\}$
- $\{E\}^+ = \{E, D\}$

- In this case, a single attribute is unable to determine all the attribute on its own like in previous example. Here, we need to combine two or more attributes to determine the candidate keys.
  - $\{A, D\}^+ = \{A, B, C, D, E\}$
  - $\{A, E\}^+ = \{A, B, C, D, E\}$
- Hence, "AD" and "AE" are the two possible keys of the given relation “R”. Any other combination other than these two would have acted as extraneous attributes.
- NOTE : Any relation “R” can have either single or multiple candidate keys.

## Closure Of Functional Dependency : Key Definitions

1. **Prime Attributes :** Attributes which
   are indispensable part of candidate keys. For example : "A, D, E" attributes are prime attributes in above example-2.
2. **Non-Prime Attributes :** Attributes other than prime attributes which does not take part in formation of candidate keys.
3. **Extraneous Attributes :** Attributes which does not make any effect on removal from candidate key.

- For example : Consider the relation R(A, B, C, D) with functional dependencies :
- FD1 : A  BC
- FD2 : B  C
- FD3 : D  C
- Here, Candidate key can be “AD” only. Hence,

1. Prime Attributes : A, D.
2. Non-Prime Attributes : B, C
3. Extraneous Attributes : B, C(As if we add any of the to the candidate key, it will remain unaffected). Those attributes, which if removed does not affect closure of that set.

# Armstrong axioms

- **Armstrong axioms** are a complete set of inference rules or axioms, introduced and developed by William W. Armstrong in 1974.
- The inference rules are sound which is used to test logical inferences of functional dependencies. The axiom which also refers to as sound is used to infer all the functional dependencies on a relational database.
- The Axioms are a set of rules, that when applied to a specific set, generates a closure of functional dependencies.

- **Armstrong's Axioms** has two different set of rules,

**1. Axioms or primary rules**

  - Axiom of Reflexivity
  - Axiom of Augmentation
  - Axiom of Transitivity

**2. Additional rules or Secondary rules**

  - Union
  - Composition
  - Decomposition
  - Pseudo Transitivity

# Axioms or primary rules

- Let suppose **T (k)** with the set of attributes **k** be a relation scheme. Subsequently, we will represent subsets of **k** as **A**, **B**, **C**. The standard notation in database theory for the set of attributes is **AB** rather than **A∪B**.

**1. Reflexive Rule ($IR_1$)**

- In the reflexive rule, if Y is a subset of X, then X determines Y.
- If $X \supseteq Y$ then $X \rightarrow Y$
- X = {a, b, c, d, e}
- Y = {a, b, c}

## 2. Augmentation Rule ($IR_2$)

- The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.
- If $X \rightarrow Y$ then $XZ \rightarrow YZ$
- **Example:**
- For R(ABCD),
- **if** $A \rightarrow B$ then
- $AC \rightarrow BC$

## 3. Transitive Rule ($IR_3$)

- In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.
- If $X \rightarrow Y$ and
- $Y \rightarrow Z$ then
- $X \rightarrow Z$

# Secondary Rules

**4. Union Rule ($IR_4$)**

- Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.
- If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
- **Proof:**
- 1. $X \rightarrow Y$ (given)
  2. $X \rightarrow Z$ (given)
  3. $X \rightarrow XY$ (using $IR_2$ on 1 by augmentation with X. Where $XX = X$)
  4. $XY \rightarrow YZ$ (using $IR_2$ on 2 by augmentation with Y)
  5. $X \rightarrow YZ$ (using $IR_3$ on 3 and 4)

## 5. Decomposition Rule ($IR_5$)

- Decomposition rule is also known as project rule. It is the reverse of union rule.
- This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.
- If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$
- **Proof:**
- 1. $X \rightarrow YZ$ (given)
  2. $YZ \rightarrow Y$ (using $IR_1$ Rule)
  3. $X \rightarrow Y$ (using $IR_3$ on 1 and 2)

## 6. Pseudo transitive Rule ($IR_6$)

- In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.
- If $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$
- **Proof:**
- 1. $X \rightarrow Y$ (given)
  2. $WY \rightarrow Z$ (given)
  3. $WX \rightarrow WY$ (using $IR_2$ on 1 by augmenting with W)
  4. $WX \rightarrow Z$ (using $IR_3$ on 3 and 2)

# Normalization

- **Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.
- **Anomalies in DBMS**
- There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.
- **Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

| emp_id | emp_name | emp_address | emp_dept |
|---|---|---|---|
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Maggie | Agra | D890 |
| 166 | Glenn | Chennai | D900 |
| 166 | Glenn | Chennai | D004 |

1. **Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.
2. **Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn’t allow nulls.
3. **Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

- To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

- **Normalization**
- Here are the most commonly used normal forms:

1. First normal form(1NF)
2. Second normal form(2NF)
3. Third normal form(3NF)
4. Boyce & Codd normal form (BCNF)

# First normal form (1NF)

- As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.
- **Example**: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

| emp_id | emp_name | emp_address | emp_mobile |
|---|---|---|---|
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212<br>9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123<br>8123450987 |

- Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.
- This table is **not in 1NF** as the rule says **“each attribute of a table must have atomic (single) values”**, the emp_mobile values for employees Jon & Lester violates that rule.
- To make the table complies with 1NF we should have the data like this:

| emp_id | emp_name | emp_address | emp_mobile |
|---|---|---|---|
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123 |
| 104 | Lester | Bangalore | 8123450987 |

# Second normal form (2NF)

- A table is said to be in 2NF if both the following conditions hold:

1. Table is in 1NF (First normal form)
2. No non-prime attribute is dependent on the proper subset of any candidate key of table.

- An attribute that is not part of any candidate key is known as non-prime attribute.
- **Example**: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

| teacher_id | subject | teacher_age |
|---|---|---|
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

- **Candidate Keys**:
- {teacher_id, subject}
  **Non prime attribute**: teacher_age
- The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says **"no non-prime attribute is dependent on the proper subset of any candidate key of the table"**.
- To make the table complies with 2NF we can break it in two tables like this:

- **teacher_details table:**

| teacher_id | teacher_age |
|---|---|
| 111 | 38 |
| 222 | 38 |
| 333 | 40 |

- **teacher_subject table:**

| teacher_id | subject |
|---|---|
| 111 | Maths |
| 111 | Physics |
| 222 | Biology |
| 333 | Physics |
| 333 | Chemistry |

# Third Normal form (3NF)

- A table design is said to be in 3NF if both the following conditions hold:

1. Table must be in 2NF
2. **Transitive functional dependency** of non-prime attribute on any super key should be removed.

- In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

1. X is a **super key** of table
2. Y is a prime attribute of table

- **Example**: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

| emp_id | emp_name | emp_zip | emp_state | emp_city | emp_district |
|---|---|---|---|---|---|
| 1001 | John | 282005 | UP | Agra | Dayal Bagh |
| 1002 | Ajeet | 222008 | TN | Chennai | M-City |
| 1006 | Lora | 282007 | TN | Chennai | Urrapakkam |
| 1101 | Lilly | 292008 | UK | Pauri | Bhagwan |
| 1201 | Steve | 222999 | MP | Gwalior | Ratan |

- **Super keys**:
1. {emp_id},
2. {emp_id, emp_name},
3. {emp_id, emp_name, emp_zip}…so on

- **Candidate Keys**: {emp_id}

- Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.
- To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

- employee table:

| emp_id | emp_name | emp_zip |
|---|---|---|
| 1001 | John | 282005 |
| 1002 | Ajeet | 222008 |
| 1006 | Lora | 282007 |
| 1101 | Lilly | 292008 |
| 1201 | Steve | 222999 |

- **employee_zip table:**

| emp_zip | emp_state | emp_city | emp_district |
|---|---|---|---|
| 282005 | UP | Agra | Dayal Bagh |
| 222008 | TN | Chennai | M-City |
| 282007 | TN | Chennai | Urrapakkam |
| 292008 | UK | Pauri | Bhagwan |
| 222999 | MP | Gwalior | Ratan |

# Boyce Codd normal form (BCNF)

- It is an advance version of 3NF that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF.
- A table complies with BCNF if it is in 3NF and for every **functional dependency** X->Y, X should be the super key of the table.
- **Example**: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

| emp_id | emp_nationality | emp_dept | dept_type | dept_no_of_emp |
|---|---|---|---|---|
| 1001 | Austrian | Production and planning | D001 | 200 |
| 1001 | Austrian | stores | D001 | 250 |
| 1002 | American | design and technical support | D134 | 100 |
| 1002 | American | Purchasing department | D134 | 600 |

- **Functional dependencies in the table above**:
  emp_id -> emp_nationality
  emp_dept -> {dept_type, dept_no_of_emp}

- **Candidate key**: {emp_id, emp_dept}
- The table is not in BCNF as neither emp_id nor emp_dept alone are keys.
- To make the table comply with BCNF we can break the table in three tables like this:
  **emp_nationality table:**

| emp_id | emp_nationality |
|---|---|
| 1001 | Austrian |
| 1002 | American |

## emp_dept table:

| emp_dept | dept_type | dept_no_of_emp |
|---|---|---|
| Production and planning | D001 | 200 |
| stores | D001 | 250 |
| design and technical support | D134 | 100 |
| Purchasing department | D134 | 600 |

- **emp_dept_mapping table:**

| **emp_id** | **emp_dept** |
|---|---|
| 1001 | Production and planning |
| 1001 | stores |
| 1002 | design and technical support |
| 1002 | Purchasing department |

- **Functional dependencies**:
  emp_id -> emp_nationality
  emp_dept -> {dept_type, dept_no_of_emp}
- **Candidate keys**:
  For first table: emp_id
  For second table: emp_dept
  For third table: {emp_id, emp_dept}
- This is now in BCNF as in both the functional dependencies left side part is a key.

## 4NF

- Fourth Normal Form comes into picture when **Multi-valued Dependency** occur in any relation
- Rules for 4th Normal Form
- For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.

- Below we have a college enrolment table with columns s_id, course and hobby.

| s_id | course | hobby |
|---|---|---|
| 1 | Science | Cricket |
| 1 | Maths | Hockey |
| 2 | C# | Cricket |
| 2 | Php | Hockey |

- To make the above relation satisfy the 4th normal form, we can decompose the table into 2 tables.
- **CourseOpted Table**
-

| **s_id** | **course** |
|---|---|
| 1 | Science |
| 1 | Maths |
| 2 | C# |
| 2 | Php |

- **Hobbies Table**

| s_id | hobby |
|---|---|
| 1 | Cricket |
| 1 | Hockey |
| 2 | Cricket |
| 2 | Hockey |

- Now this relation satisfies the fourth normal form.
- A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.
- If you design your database carefully, you can easily avoid these issues.

## 5NF

- A relation is in 5NF if it is in 4NF.
- If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

| SUBJECT | LECTURER | SEMESTER |
|---|---|---|
| Computer | Anshika | Semester 1 |
| Computer | John | Semester 1 |
| Math | John | Semester 1 |
| Math | Akash | Semester 2 |
| Chemistry | Praveen | Semester 1 |

- In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.
- Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.
- So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

| SEMESTER | SUBJECT |
|---|---|
| Semester 1 | Computer |
| Semester 1 | Math |
| Semester 1 | Chemistry |
| Semester 2 | Math |

| SUBJECT | LECTURER |
|---|---|
| Computer | Anshika |
| Computer | John |
| Math | John |
| Math | Akash |
| Chemistry | Praveen |

| SEMSTER | LECTURER |
|---|---|
| Semester 1 | Anshika |
| Semester 1 | John |
| Semester 1 | John |
| Semester 2 | Akash |
| Semester 1 | Praveen |

- Now, each of combinations is in three different tables. If we need to identify who is teaching which subject to which semester, we need join the keys of each table and get the result.
- For example, who teaches Physics to Semester 1, we would be selecting Physics and Semester1 from table 3 above, join with table1 using Subject to filter out the lecturer names. Then join with table2 using Lecturer to get correct lecturer name. That is we joined key columns of each table to get the correct data. Hence there is no lose or new data – satisfying 5NF condition.

## GATE practice Questions

**Q.1** Let R (A, B, C, D, E, P, G) be a relational schema in which the following functional dependencies are known to hold: $AB \rightarrow CD$, $DE \rightarrow P$, $C \rightarrow E$, $P \rightarrow C$ and $B \rightarrow G$. The relational schema R is

- **(A)** in BCNF
  **(B)** in 3NF, but not in BCNF
  **(C)** in 2NF, but not in 3NF
  **(D)** not in 2NF

- AB → CD,
- DE → P,
- C → E,
- P → C and
- B → G
- X->y

- **Answer: (D)**

  **Explanation:** Candidate key = AB
  B->G is partial dependency
  So, not in 2NF

**Q.2** A table has fields Fl, F2, F3, F4, F5 with the following functional dependencies

F1 $\rightarrow$ F3

F2 $\rightarrow$ F4

(F1 . F2) $\rightarrow$ F5

In terms of Normalization, this table is in

-
  (A) 1 NF
  (B) 2 NF
  (C) 3 NF
  (D) none

- **Answer: (A)**

  **Explanation:**
- **First Normal Form**
  A relation is in first normal form if every attribute in that relation is singled valued attribute.
- **Second Normal Form**
- A relation is in 2NF iff it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
- This table has **Partial Dependency f1->f3, f2-> f4** given (F1, F2) is Key
- **So answer is A**

**Q.3** Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values. F =

{CH -> G,

A -> BC,

B -> CFH,

E -> A,

F -> EG} is a set of functional dependencies (FDs) so that F+ is exactly the set of FDs that hold for R.

- How many candidate keys does the relation R have?
- (A) 3
  (B) 4
  (C) 5
  (D) 6

- **Answer: (B)**

  **Explanation:** A+ is ABCEFGH which is all attributes except D.
- B+ is also ABCEFGH which is all attributes except D.
- E+ is also ABCEFGH which is all attributes except D.
- F+ is also ABCEFGH which is all attributes except D.
- So there are total 4 candidate keys AD, BD, ED and FD

**Q.4** Consider the FDs given in above question. The relation R is
**(A)** in 1NF, but not in 2NF.
**(B)** in 2NF, but not in 3NF.
**(C)** in 3NF, but not in BCNF.
**(D)** in BCNF

- **Answer: (A)**

  **Explanation:** The table is not in 2nd Normal Form as the non-prime attributes are dependent on subsets of candidate keys.
- The candidate keys are AD, BD, ED and FD. In all of the following FDs, the non-prime attributes are dependent on a partial candidate key.
- A -> BC
  B -> CFH
  F -> EG

**Q.5** Consider the relation scheme R = {E, F, G, H, I, J, K, L, M, M} and the set of functional dependencies

{{E, F} -> {G},

{F} -> {I, J},

{E, H} -> {K, L},

K -> {M},

L -> {N} on R. What is the key for R?

(A) {E, F}
(B) {E, F, H}
(C) {E, F, H, K, L}
(D) {E}

- **Answer: (B)**

  **Explanation:** All attributes can be derived from {E, F, H}

**Q.6** For a database relation R(A, B, C, D) where the domains of A, B, C and D include only atomic values, only the following functional dependencies and those that can be inferred from them are :

$A \rightarrow C$

$B \rightarrow D$

The relation R is in _______.

**(A)** First normal form but not in second normal form.

**(B)** Both in first normal form as well as in second normal form.

**(C)** Second normal form but not in third normal form.

**(D)** Both in second normal form as well as in third normal form.

- **Answer: (A)**

  **Explanation:** A relation is in first normal form if every attribute in that relation is single valued attribute. It is in 1NF.
  {A,B} are prime attribtes and {C,D} are non-prime attribute.
  $A^+ = \{A,C\}$
  $B^+ = \{B,D\}$
  $\{A,B\}^+ = \{A,B,C,D\}$ so AB is the key.
  But $A^+ = \{A,C\}$
  $B^+ = \{B,D\}$ makes it partial dependency.
  So, this relation is not in 2NF.
  So, option (A) is correct.

**Q.7** Consider the following dependencies and the BOOK table in a relational database design. Determine the normal form of the given relation.

- ISBN → Title
- ISBN → Publisher
- Publisher → Address
- (A) First Normal Form
  (B) Second Normal Form
  (C) Third Normal Form
  (D) BCNF

- **Answer: (B)**

  **Explanation:** Candidate key = ISBN
  For a relation having functional dependencies of the form $\alpha \rightarrow \beta$, a relation is in 2-NF if:
  i) $\alpha$ should **not** be a **proper** subset of the candidate key, or,
  ii) $\beta - \alpha$ should be a prime attribute.
- First condition satisfies as the candidate key contains only one attribute.
- So, this relation is in 2-NF

**Q.8** Consider the relation R (ABCDE):
FD = { $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$}
Find out the highest normal form.
**(A)** 1 NF
**(B)** 2 NF
**(C)** 3 NF
**(D)** BCNF

- **Answer: (B)**

  **Explanation:** Here candidate Key is A and B -> C , C -> D , D -> E all are. (Non prime attribute -> Non prime attribute.).
- This type of FD must not be present in 3NF therefore highest normal form of this FDs are 2NF.
- Option (B) is correct.

# Lossless Join Decomposition

- Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.
- **Lossless Join Decomposition**

1. If we decompose a relation R into relations R1 and R2,
2. Decomposition is lossy if $R1 \bowtie R2 \supset R$
3. Decomposition is lossless if $R1 \bowtie R2 = R$

- **To check for lossless join decomposition using FD set, following conditions must hold:**

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2. **Att(R1) U Att(R2) = Att(R)**
2. Intersection of Attributes of R1 and R2 must not be NULL. **Att(R1) ∩ Att(R2) ≠ Φ**
3. Common attribute must be a key for at least one relation (R1 or R2)

   **Att(R1) ∩ Att(R2) -> Att(R1) or**

   **Att(R1) ∩ Att(R2) -> Att(R2)**

- For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:
- First condition holds true as
- **Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).**
- Second condition holds true as
- **Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ**
- Third condition holds true as

**Att(R1) ∩ Att(R2) = A is a key of R1(ABC)** because A->BC is given.

## Dependency Preserving Decomposition

- If we decompose a relation R into relations R1 and R2,

1. All dependencies of R either must be a part of R1 or R2 or
2. must be derivable from combination of FD's of R1 and R2.

For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of R1(ABC).

# Dependency Preservation

- **Dependency Preservation**
- A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if
- **(F1 ∪ F2 ∪ … ∪ Fm)+ = F+.**
- Consider a relation R R ---> F{...with some functional dependency(FD)....} R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then there can be three cases:

1. **f1 U f2 = F** -----> Decomposition is dependency preserving.
2. **f1 U f2** is a subset of F -----> Not Dependency preserving.
3. **f1 U f2** is a super set of F -----> This case is not possible.

- *Example:*
- Let a relation R(A,B,C,D) and set a FDs
- F = { A -> B ,  A -> C  , C -> D}  are given. A relation R is decomposed into –

1. **$R_1$ = (A, B, C) with FDs $F_1$ = {A -> B, A -> C},**
2. **$R_2$ = (C, D) with FDs $F_2$ = {C -> D}.**

- F' = $F_1 \cup F_2$ = {A -> B, A -> C, C -> D} so,
- F' = F. And so,
- $F'^+ = F^+$.