UNIT-IV
SCHEMA REFINEMENT and NORMAL FORMS

SCHEMA REFINEMENT: The purpose of Schema refinement is used for a refinement approach based on decompositions. Redundant storage of information (i.e. duplication of data) is main cause of problem. This redundancy is eliminated by decompose the relation.

PROBLEMS CAUSED BY REDUNDANCY: Redundancy is a method of storing the same information repeatedly. It means, storing the same data more than one place within a database is can lead several problems. Such as

1) **Redundant Storage:** It removes the multi-valued attribute. It means, some tuples or information is stored repeatedly.

2) **Update Anomalies:** Suppose, if we update one row (or record) then DBMS will update more than one similar row, causes update anomaly.

For example, if we update the department name those who getting the salary 40000 then that will update more than one row of employee table is causes update anomaly.

3) **Insertion Anomalies:** In insertion anomaly, when allows insertion for already existed record again, causes insertion anomaly.

4) **Deletion Anomalies:** In deletion anomaly, when more than one record is deleted instead of specified or one, causes deletion anomaly.

Consider the following example,

Eno	ename	salary	rating	hourly_wages	hours_worked
111	suresh	25000	8	10	40
222	eswar	30000	8	10	30
333	sankar	32000	5	7	30
444	padma	31000	5	7	32
555	aswani	35000	8	10	40

In this example,

1) Redundancy storage: The rating value 8 corresponds to the hourly_wages 10 and there is repeated three times.
2) Update anomalies: If the hourly_wages in the first tuple is updated, it does not make changes in the corresponding second and last tuples. Because, the key element of tuples is emp_id.
   i.e. update employee set hourly_wages = 12 where emp_id = 140; But the question is update hourly_wages from 10 to 12.
3) Insertion anomalies: If we have to inset a new tuple for an employee, we should know both the values rating value as well as hourly_wages value.
4) Deletion anomalies: Delete all the tuples through a given rating value, causes deletion anomaly.

DECOMPOSITIONS: A relation schema (table) R can be decomposition into a collection of smaller relation schemas (tables) (i.e. R_1, R_2, ... R_m) to eliminate anomalies caused by the redundancy in the original relation R is called Decomposition of relation. This shown in Relational Algebra as

$R_1 \subseteq R$ for $1 \leq i \leq m$ and $R_1 \cup R_2 \cup R_3 \ldots R_m = R$.

(or)

A decomposition of a relation schema R consists of replacing the relation schema by two or more relation schemas that each contains a subset of the attributes of R and together include all attributes in R.

(or)

In simple, words, "The process of breaking larger relations into smaller relations is known as decomposition".

Consider the above example that can be decomposing the following hourly_emps relation into two relations.

Hourly_emp (eno, ename, salary, rating, houly_wages, hours_worked)

This is decomposed into

Hourly_empsd(eno, ename, salary, rating, hours_worked) **and**
Wages(rating, houly_wages)

Eno	ename	salary	rating	hours_worked
111	suresh	25000	8	40
222	eswar	30000	8	30
333	sankar	32000	5	30
444	padma	31000	5	32
555	aswani	35000	8	40

rating	hourly_wages
8	10
5	7

Problems Related to Decomposition:

The use of Decomposition is to split the relation into smaller parts to eliminate the anomalies. The question is

1. What are problems that can be caused by using decomposition?
2. When do we have to decompose a relation?

The answer for the **first** question is, two properties of decomposition are considered.

1) **Lossless-join Property:** This property helps to identify any instance of the original relation from the corresponding instance of the smaller relation attained after decomposition.
2) **Dependency Preservation**: This property helps to enforce constraint on smaller relation in order to enforce the constraints on the original relation.

The answer for the **second** question is, number of normal forms exists. Every relation schema is in one of these normal forms and these normal forms can help to decide whether to decompose the relation schema further or not.

The **Disadvantage** of decomposition is that it enforces us to join the decomposed relations in order to solve the queries of the original relation.

What is Relation?: A relation is a named two–dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows.

For example, a relation named Employee contains following attributes, emp-id, ename, dept name and salary

Emp-id	ename	dept name	salary
100	Simpson	Marketing	48000
140	Smith	Accounting	52000
110	Lucero	Info-systems	43000
190	Davis	Finance	55000
150	Martin	Marketing	42000

What are the Properties of relations?:

The properties of relations are defined on two dimensional tables. They are:

- Each relation (or table) in a database has a unique name.
- An entry at the intersection of each row and column is atomic or single. These can be no multiplied attributes in a relation.
- Each row is unique; no two rows in a relation are identical.
- Each attribute or column within a table has a unique name.
- The sequence of columns (left to right) is insignificant the column of a relation can be interchanged without changing the meaning use of the relation.
- The sequence of rows (top to bottom) is insignificant. As with column, the rows of relation may be interchanged or stored in any sequence.

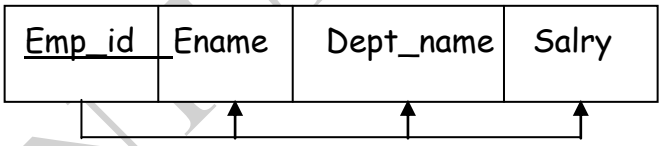
Removing multi valued attributes from tables: the "second property of relations" in the above is applied to this table. In this property, there is no multi valied attributes in a relation. This rule is applied to the table or relation to eliminate the one or more multi valued attribute. Consider the following the example, the employee table contain 6 records. In this the course title has multi valued values/attributes. The employee 100 taken two courses vc++ and ms-office. The record 150 did not taken any course. So it is null.

Emp-id	name	dept-name	salary	course_title
100	Krishna	cse	20000	vc++, msoffice
140	Rajasekhar	cse	18000	C++, DBMS,DS

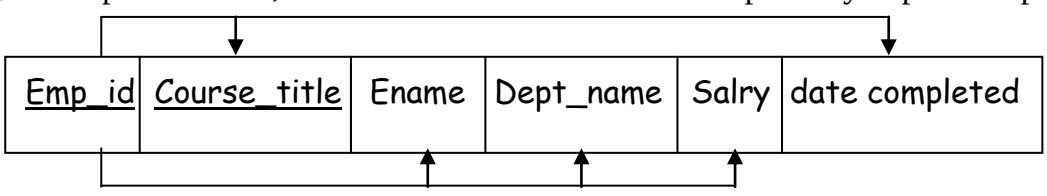
Now, this multi valued attributes are eliminated and shown in the following employee2 table.

Emp-id	name	dept-name	salary	course_title
100	Krishna	cse	20000	vc++
100	Krishna	cse	20000	MSoffice
140	Rajasekhar	cse	18000	C++
140	Rajasekhar	cse	18000	DBMS
140	Rajasekhar	cse	18000	DS

FUNCTIONAL DEPENDENCIES: A functional dependency is a constraint between two attributes (or) two sets of attributes.
For example, the table EMPLOYEE has 4 columns that are functionally dependencies on EMP_ID.



PARTIAL FUNCTIONAL DEPENDENCY: It is a functional dependency in which one or more non-key attributes are functionally dependent on part of the primary key. Consider the following graphical representation, in that some of the attributes are partially depend on primary key.



In this example, Ename, Dept_name, and salary are fully functionally depend on Primary key of Emp_id. But Course_title and date_completed are partial functional dependency. In this case, the partial functional dependency creates redundancy in that relation.

What is Normal Form? What are steps in Normal Form?

NORMALIZATION: Normalization is the process of decomposing relations to produce smaller, well-structured relation.

To produce smaller and well structured relations, the user needs to follow six normal forms.

Steps in Normalization:

A *normal form* is state of relation that results from applying simple rules from regarding functional dependencies (relationships between attributes to that relation. The normal forms are

1. First normal form
2. Second normal form
3. Third normal form
4. Boyce/codd normal form
5. Fourth normal form
6. Fifth normal form

1) **First Normal Form:** Any multi-valued attributes (also called repeating groups) have been removed,
2) **Second Normal Form:** Any partial functional dependencies have been removed.
3) **Third Normal Form:** Any transitive dependencies have been removed.
4) **Boyce/Codd Normal Form:** Any remaining anomalies that result from functional dependencies have been removed.
5) **Fourth Normal Form:** Any multi-valued dependencies have been removed.
6) **Fifth Normal Form:** Any remaining anomalies have been removed.

advantages of normalized relations over the un-normalized relations: The advantages of normalized relations over un-normalized relations are,

1) Normalized relation (table) does not contain repeating groups whereas, un-normalized relation (table) contains one or more repeating groups.
2) Normalized relation consists of a primary key. There is no primary key presents in un-normalized relation.
3) Normalization removes the repeating group which occurs many times in a table.
4) With the help of normalization process, we can transform un-normalized table to First Normal Form (1NF) by removing repeating groups from un-normalized tables.
5) Normalized relations (tables) gives the more simplified result whereas un-normalized relation gives more complicated results.
6) Normalized relations improve storage efficiency, data integrity and scalability. But un-normalized relations cannot improvise the storage efficiency and data integrity.
7) Normalization results in database consistency, flexible data accesses.

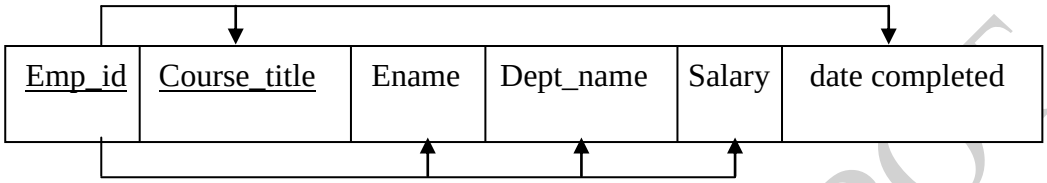
FIRST NORMAL FORM (1NF): A relation is in first normal form (1NF) contains no multi-Valued attributes. Consider the example employee, that contain multi valued attributes that are removing and converting into single valued attributes **Multi valued attributes in course title**

Emp-id	name	dept-name	salary	course_title
100	Krishna	cse	20000	vc++,
				msoffice
140	Raja	it	18000	C++,
				DBMS,
				DS

Removing the multi valued attributes and converting single valied using First NF

Emp-id	name	dept-name	salary	course_title
100	Krishna	cse	20000	vc++
100	Krishna	cse	20000	msoffice
140	Raja	it	18000	C++
140	Raja	it	18000	DBMS
140	Raja	it	18000	DS

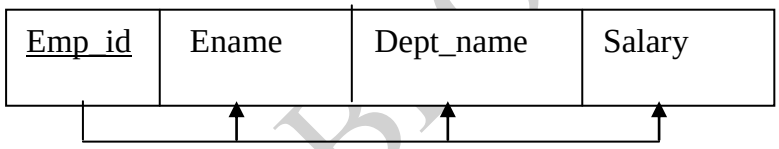
SECOND NORMAL FORM(2NF): A relation in Second Normal Form (2NF) if it is in the 1NF and if all non-key attributes are fully functionally dependent on the primary key. In a functional dependency X → Y, the attribute on left hand side (i.e. x) is the primary key of the relation and right side attributes on right hand side i.e. Y is the non-key attributes. In some situation some non key attributes are partial functional dependency on primary key. Consider the following example for partial functional specification and also that convert into 2 NF to decompose that into two relations.



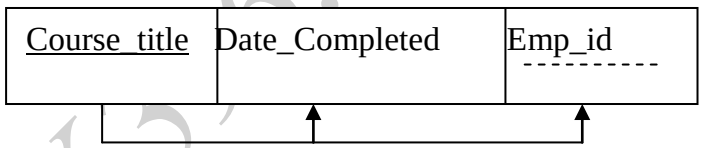
In this example, Ename, Dept_name, and salary are fully functionally depend on Primary key of Emp_id. But Course_title and date_completed are partial functional dependency. In this case, the partial functional dependency creates redundancy in that relation.

- To avoid this, convert this into Second Normal Form. The 2NF will decompose the relation into two relations, shown in graphical representation.

EMPLOYEE



COURSE

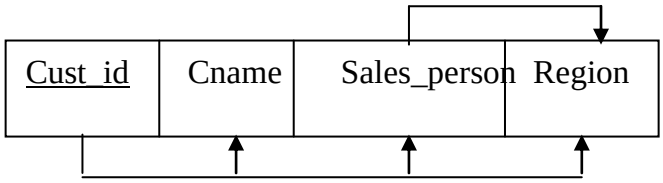


In the above graphical representation

- the EMPLOYEE relation satisfies rule of 1 NF in Second Normal form and
- the COURSE relation satisfies rule of 2 NF by decomposing into two relation.

THIRD NORMAL FORM(3NF): A relation is said to be in 3NF, if it is already in 2NF and there exists no transitive dependency in that relation

Transitive dependency : A transitive is a functional dependency between two non key attributes. For example, consider the relation Sales with attributes cust_id, name, sales person and region that shown in graphical representation.



CUST_ID	CNAME	SALESPERSON	REGION
1001	Anand	Smith	South
1002	Sunil	kiran	West
1003	Govind	babu rao	East
1004	Manohar	Smith	South
1005	Madhu	Somu	North

In this example, to insert, delete and update any row that facing Anomaly.

a) Insertion Anomaly: A new salesperson is assigned to North Region without assign a customer to that salesperson. This causes insertion Anomaly.
b) Deletion Anomaly: If a customer number say 1003 is deleted from the table, we lose the information of salesperson who is assigned to that customer. This causes, Deletion Anomaly.
c) Modification Anomaly: If salesperson Smith is reassigned to the East region, several rows must be changed to reflect that fact. This causes, update anomaly.

To avoid this Anomaly problem, the transitive dependency can be removed by decomposition of SALES into two relations in **3NF**.

Consider the following example, that removes Anomaly by decomposing into two relations.

Cust_Id	Name	Sales Person
1001	Anand	Smith
1002	Sunil	kiran
1003	Govind	babu rao
1004	Manohar	Smith
1005	Madhu	Somu

Sales Person	Region
Smith	South
kiran	West
babu rao	East
Smith	South
Somu	North

SALES PERSON

Sales_person	Region

CUSTOMER

Cust_id	Cname	Salry

BOYCE/CODD NORMAL FORM(BCNF): A relation is in BCNF if it is in 3NF and every determinant is a candidate key.
FD in F^+ of the form $X \rightarrow A$ where $X \subseteq S$ and $A \in S$, X is a super key of R.

Boyce-Codd normal form removes the remaining anomalies in 3NF that are resulting from functional dependency; we can get the result of relation in BCNF.
For example, STUDENT-ADVIDSOR IN 3NF

STUDENT-ADVISOR

Student-id	major-subject	faculty-advisor

STUDENT-ADVISOR relstion with simple data.

STYDENT-ID	MAJOR-SUBJECT	FACULTY-ADVISOR
123	Physics	Faculty-1
123	Math's	Faculty-2
456	Bio	Faculty-3
789	Physics	Faculty-4
999	Physics	Faculty-1

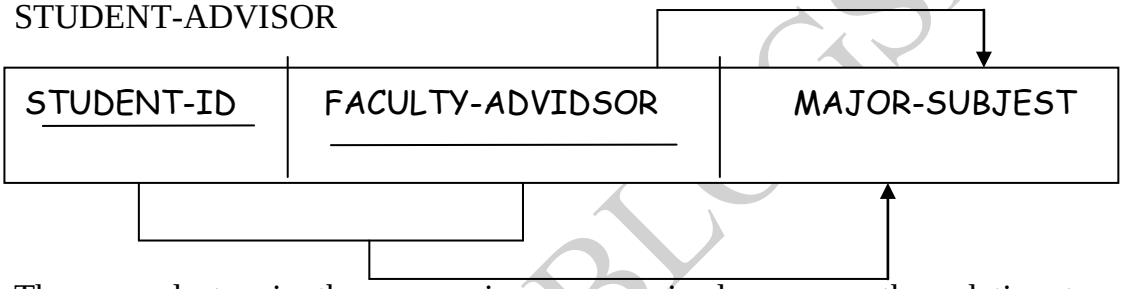
In the above relation the primary key in student-id and major-subject. Here the part of the primary key major-subject is dependent upon a non key attribute faculty–advisor. So, here the determenant the faculty-advisor. But it is not candidate key.

Here in this example there are no partial dependencies and transitive dependencies. There is only functional dependency between part of the primary key and non key attribute. Because of this dependency there is anomely in this relation.

Suppose that in physics subject the advisor' 1' is replaced by X. this change must be made in two or more rows in this relation. This is an updation anomely.

To convert a relation to BCNF the first step in the original relation is modified that the determinant(non key attributes) becoms a component of the primary key of new relation. The attribute that is dependent on determinant becomes a non key attributes .

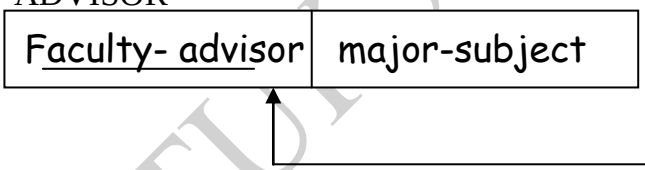
STUDENT-ADVISOR



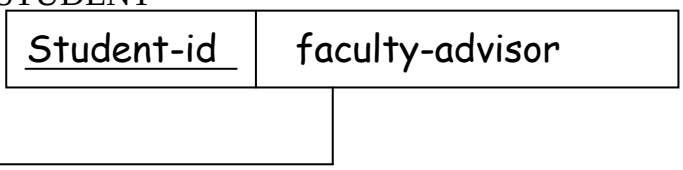
The second step in the conversion process is decompose the relation to eleminate the partial functional dependency. This results in two relations. these relations are in 3NF and BCNF . since there is only one candidate key. That is determinant.

Two relations are in BCNF.

ADVISOR



STUDENT



In these two relations the student relation has a composite key , which contains attributes student-id and faculty-advisor. Here faculty–advisor a foreign key which is referenced to the primary key of the advisor relation.

Two relations are in BCNF with simple data.

Faculty Advisor	Major_subject
Faculty-1	Physics
Faculty-2	Maths
Faculty-3	Bio
Faculty-4	Maths

Student_id	Faculty_Advisor
123	Faculty-1
123	Faculty-2
456	Faculty-3
789	Faculty-4
999	Faculty-1

FOURTH NORMAL FORM (4NF) : A relation is in 4NF that contain no multi-valued dependency. In this case, **1 NF** will repeated in this step. For example, R be a relation schema, X and Y be attributes of R, and F be a set of dependencies that includes both FDs and MVDs. (i.e. Functional Dependency and Multi-valued Dependencies). Then R is said to be in Fourth Normal Form (4NF) if for every MVD X →→ Y that holds over R, one of the following statements is true.
1) Y c X or XY = R, or 2) X is a super key.

multi-valued dependency is a typical kind of dependency in which each and every attribute within a relation depends upon the other, yet none of them is a unique primary key.

We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organization. The following are the assumptions:

A vendor is capable of supplying many items.
A project uses many items.
A vendor supplies to many projects.
An item may be supplied by many vendors.

A multi valued dependency exists here because all the attributes depend upon the other and yet none of them is a primary key having unique value.

Vendor Code	Item Code	Project No.
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P3

The given relation has a number of problems. For example:

If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.

The information about item I1 is stored twice for vendor V3.

Observe that the relation given is in 3NF and also in BCNF. It still has the problem mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multi valued dependency or one independent multi valued dependency with a functional dependency.

The table can be expressed as the two 4NF relations given as following. The fact that vendors are capable of supplying certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

Vendor-Supply(vendor code→→Item code)
Vendor-Project(vendor code→→project no)

Vendor Code	Project No.
V1	P1
V1	P3
V2	P1
V3	P2
V3	P3

Vendor Code	Item Code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

FIFTH NORMAL FORM (5 NF) :

A relation is in 5NF it is 4NF and not having any join dependency and joining should be lossless.

Any remaining anomalies from 4 NF relation have been removed. It is also called as project join Normal form.

A relation schema R is said to be in Fifth Normal Form (5NF) if, for every join dependency * (R1, ... R_n) that holds over R, one of the following statements is true.

*R_i = R for some I, or R1 ⋈ (R2 ⋈ R3) or

* The JD is implied by the set of those FDs over R in which the left side is a key for R. It deals with a property loss less joins.

LOSSELESS-JOIN DECOMPOSITION:

Let R be a relation schema and let F be a set FDs (Functional Dependencies) over R. A decomposition of R into two schemas with attribute X and Y is said to be lossless-join decomposition with respect to F, if for every instance r of R that satisfies the dependencies in F_r.

$$\pi_x (r) \bowtie \pi_y (r) = r$$

In simple words, we can recover the original relation from the decomposed relations.

In general, if we take projection of a relation and recombine them using natural join, we obtain some additional tuples that were not in the original relation.

S	P	D
s_1	p_1	d_1
s_2	p_2	d_2
s_2	p_1	d_3

S	P
s_1	p_1
s_2	p_2
s_2	p_1

P	D
p1	d_1
p_2	d_2
p_1	d_3

S	P	D
s_1	p_1	d_1
s_1	p_1	d_3
s_2	p_2	d_2
s_2	p_1	d_1
s_2	p_1	d_3

The decomposition of relation schema r i.e. SPD into SP i.e. PROJECTING π_{sp} (r) and PD i.e., projecting π_{PD} (r) is therefore lossless decomposition as it gains back all original tuples of relation 'r' as well as with some additional tuples that were not in original relation 'r'.

DEPENDENCY-PRESERVING DECOMPOSITION:

Dependency-preserving decomposition property allows us to check integrity constraints efficiently. In simple words, a dependency-preserving decomposition allows us to enforce all FDs by examining a single relation on each insertion or modification of a tuple.

Let R be a relation schema that is decomposed in two schemas with attributes X and Y and let F be a set of FDs over R. The projection of F on X is the set of FD s in the closure F^+ that involves only attributes in X. We denote the projection of F on attributes X as F_x. Note that a

dependency U → V in F^+ is in F_x only if all the attributes in U and Y are in X. The decomposition of relation schema R with FDs F into schemas with attributes X and Y is dependency-preserving if $(F_x \cup F_y)^+ = F^+$.

That is, if we take the dependencies in F_x and F_y and compute the closure of their union, then we get back all dependencies in the closure of F. To enforce F_x. we need to examine only relation X (that inserts on that relation) to enforce F_y, we need to examine only relation Y.

Example: Suppose that a relation R with attributes ABC is decomposed into relations with attributes AB and BC. The set F of FDs over r includes A → B , B → C and C → A. Here, A → B is in F_{AB} and B → C is in F_{BC} and both are dependency-preserving. Whereas C → A is not implied by the dependencies of F_{AB} and F_{BC}. Therefore C → A is not dependency-preserving.

Consequently, F_{AB} also contains B → A as well as A → B and F_{BC} contains C → B as well as B → C. Therefore $F_{AB} \cup F_{BC}$ contain A → B , B → C, B → A and C → B.

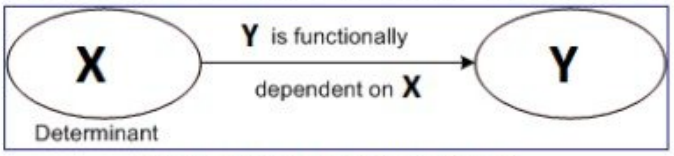
Now, the closure of the dependencies in F_{AB} and F_{BC} includes C → A (because, from C → B, B → A and transitivity rule, we compute as C → A).

Ex: consider the fallowing functional dependences in a database
DOB-->Age
Age-->Eligibility
Name-->Rollno
Rollno-->Name
Course no-->course name
courseno-->instructor
the relation (Rollno,name,DOB,Age) is in
A) 2NF but not in 3NF B) 3NF but not in BCNF C)BCNF D)None of the above
Answer:
For the given relation fallowing FDs are applicable
DOB-->Age
Name-->Rollno
Rollno-->Name
Candidate key for the above are
(DOB,Name) and (DOB,Rollno)
but there is a partial dependency here as DOB-->Age
So it is only in 1NF
so answer is option(D).

Ex: Let R=(A,B,C,D,E,F) be a relation schema with the fallowing FDs C-->F, E-->A, EC-->D, A-->B which of the fallowing is a key for R?
A) CD B)AC C)AE D)EC
answer:
here EC is the key for R in FDs E and C are not coming on the RHS of any FD.
so both of them must be present in key and also
$(EC)^+ = \{A,B,C,D,E,F\}$

Concept of Functional Dependency:

Functional Dependencies are fundamental to the process of Normalization i.e., Functional Dependency plays key role in differentiating good database design from bad database designs. A functional dependency is a "**type of constraint that is a generalization of the notation of the key**".
Functional Dependency describes the **relationship between attributes (columns) in a table**.
Functional dependency is represented by an arrow sign **(→)**.
In other words, a dependency FD: "**X → Y**" means that the values of Y are determined by the values of X. Two tuples sharing the same values of X will necessarily have the same values of Y. An attribute on left hand side is known as "Determinant". Here X is a Determinant.



Functional dependency between X and Y

Example [Identifying the FD's]

A	B	C	D
A1	B1	C1	D1
A1	B2	C1	D2
A2	B2	C2	D2
A2	B2	C2	D3
A3	B3	C2	D4

Case1: A →B
Here A1 belongs to **B1 & B2.** So **A1** does not have unique value in **B**. So it is not in FD.
Case1: A →C
Here **A1→C1 and A2, A3→C2.** So **A** has unique values in **B**. So it is in FD.
Note: try to find all the possibilities. i.e., **A→D, B→C, B→D, and C→D**

Reasoning about functional dependencies:

Armstrong Axioms (Inference Rules) : The term Armstrong axioms refers to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test logical implication of **functional dependencies**.

Armstrong axioms define the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

Various axioms rules or inference rules:
Primary axioms:

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

Secondary or derived axioms:

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Closure of a Set of Attributes:

Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.
The set of FD's that is logically implied by F is called the closure of F and written as F^+. And it is defined as "If F is a set FD's on a relation R, the F^+, the closure of F by using the inferences axioms that are not contained in F^+.
Example: R (A, B, C, D) and set of Functional Dependencies are **A→B, B→D, C→B** then what is the Closure of A, B, C, D?
Solution: A^+ is
A^+→ {A, B, D} i.e., A→B, B→D is exists and C is not FD on A. So it is eliminated.
B^+→ { B, D} i.e., B→D is exists and A, C is not FD on A. So it is eliminated.
C^+→ {C, B, D} i.e., C→B, B→D is exists and A is not FD on C. So it is eliminated.

The algorithm for computing the attribute closure of a set X of attributes is shown below

$closure = X;$
repeat until there is no change: {
  if there is an FD $U \rightarrow V$ in F such that $U \subseteq closure$,
    then set $closure = closure \cup V$
}

Types of functional dependencies:

1. **Fully Functional Dependency**: A functional dependency is said to be full dependency "if and only if the determinant of the functional dependency if either candidate key or super key, and the dependent can be either prime or non-prime attribute".

   (OR)

   Let's take the functional dependency **X → Y (**i.e., X determines y). Here Y is said to be fully determinant, if it cannot determine any subset of X.
   Example: Consider the following determinant **ABC → D** i.e., ABC determines D but D is not determined by any subset of A/ BC/C/B/AB i.e., **BC→D, C→D, A→D** Functional dependencies are **not exists. So D is Fully Functional Dependent.**

2. **Partial Functional Dependency**: If a non-prime attribute of the relation is getting derived by only a part of the candidate key, then such dependency is known as Partial Dependency.

   (OR)

   In a relation having more than one key field, a subset of non key fields may depend on all key fields but another subset or a particular non-key field may depend on only one of the key fields. Such dependency is defined as **Partial Dependency**.
   Example: Consider the following determinants **AC→P, A→D, D→P. From these determinants P is not fully FD on AC**. Because, If we find **A+** (means A's Closure) **A→D, D→P** i.e., **A→P.** But we don't have any requirement of **C. C** attribute is removed completely. So **P** is Partially Dependent on **AC.**
   Under the following conditions a table cannot have partial F.D
   (1) If primary key consists a single attribute
   (2) If table consists only two attributes
   (3) If all the attributes in the table are part of the primary key

3. **Transitive Functional Dependency:** If a non-prime attribute of a relation is getting derived by either another non-prime attribute or the combination of the part of the candidate key along with non-prime attribute, then such dependency is defined as **Transitive dependency**. i.e., in a relation, there may be dependency among non-key fields. Such dependency is called Transitive Functional Dependency.
   Example: X→Y, and **Y→Z** then we can determine **X→Z holds.**
   Under the following Circumstances, a table cannot have transitive F.D
   (1) If table consists only two attributes
   (2) If all the attributes in the table are part of the primary key.

4. **Trivial Functional Dependency:** It is basically related to Reflexive rule. i.e., if X is a set of attributes, and Y is subset of X then **X→Y** holds.
   Example: ABC→BC is a Trivial Dependency.

5. **Multi-Valued Dependency:** Consider 3 fields X, Y, and Z in a relation. If for each value of X, there is a well-defined set of values Y and Well-defined set of values of Z and set of values of Y is independent of the set values of Z. This dependency is Multi-valued Dependency. **i.e., X →Y / Z.**

Prime and non-prime attributes

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

Candidate Key:

Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.
Consider student table: student(**sno**, sname,sphone,age)
we can take **sno** as candidate key. we can have more than 1 candidate key in a table.
types of candidate keys:
1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)

Super Key:

Super Key is set of attributes of a relation which can be used to identify a tuple uniquely.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Consider student table: student(sno, sname,sphone,age)
we can take **sno**, (**sno, sname**) as super key

Operations performed functional dependencies (applications of closure set of attributes):

(1) To identify the additional F.D's.
(2) To identify the keys.
(3) To identify the equivalences of the F.D's
(4) To identify irreducible set (minimal set) of F.D's or canonical forms of F.D's or standard form of F.D's.

(1) To identify the additional F.D's :

To check any F.D's like A→B can be determined from F1 or not. Complete A^+ from F1 is A^+ includes B also then; A→B can be derived as a F.D in F1.

Examples:

1. **In a schema with attributes A,B,C,D and E the following set of attributes are given A→B, A→C, CD→E, B→D, E→A. Find CD→AC determines from the given FDs or not.**

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Concept of Surrogate Key:

✓ Alternate of Primary Key that allows duplication of data's/records.

✓ Surrogate key is a unique identification key, it is like an artificial key to production key, because the production key may be alphanumeric or composite key but the surrogate key is always single numeric key.

✓ A surrogate key has the following characteristics:

i. The value is never reused and is unique within the whole system.

ii. It is system generated and an integer.

iii. The value cannot be manipulated by the user or application.

iv. The value is not an amalgam of different values from multiple domains.

✓ A Surrogate Keys can be generated in a variety of ways, and most databases offers ways to generate surrogate keys.

Example: Oracle uses **SEQUENCE**,
MYSQL uses **Auto_Increment**,
and SQL Server uses **IDENTITY**.

Lossless join and Dependency preserving decomposition:

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.

➔ If we decompose a relation R into relations R1 and R2, There are 2 types of decomposition:

1. **Decomposition is lossy**, if **R1 ⋈ R2 ⊃ R.**
   - ✓ The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not yield the same relation as in R." One of the disadvantages of decomposition into two or more relational schemes (or tables) is that some information is lost during retrieval of original relation or table.
   - ✓ Consider that we have table STUDENT with three attribute roll_no , sname and department.

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

- ✓ This relation is decomposed into two relation no_name and name_dept:

Roll_no	Sname
111	parimal
222	parimal

Sname	Dept
parimal	COMPUTER
parimal	ELECTRICAL

- ✓ In lossy decomposition, spurious tuples are generated when a natural join is applied to the relations in the decomposition.

stu_joined :

Roll_no	Sname	Dept
111	parimal	COMPUTER
111	parimal	ELECTRICAL
222	parimal	COMPUTER
222	parimal	ELECTRICAL

- ✓ The above decomposition is a bad decomposition or Lossy decomposition.

2. **Lossless Join Decomposition:** Decomposition is lossless if **R1 ⋈ R2 = R**
   - ✓ This is also referred as **non-additive decomposition**.
   - ✓ The lossless-join decomposition is always defined with respect to a specific set F of dependencies
   - ✓ **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).

   - ✓ **Example 1:,** 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:
     1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
     2. Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
     3. Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.