RELATIONAL DATABASE DESIGN / NORMALIZATION CONCEPTS

Informal Design Guidelines for Relation Schema:

We will discuss *four informal measures* of quality for relation schema design:

- Semantics of the attribute
- Reducing the redundant values in tuples
- Reducing the null values in tuples
- Disallowing spurious tuples

1. **Semantics of the Attributes:**

Whenever we group attributes to form a relation schema, we assume that a certain meaning is associated with the attributes. This meaning or semantics specifies how to interpret the attribute values stored in a tuple of the relation.

Guideline 1: Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation.

2. Redundant Information in Tuples & Update Anomalies:

One goal of schema design is to minimize the storage space that the base relations (files) occupy. Grouping attributes into relation schemas has a significant effect on storage space.

For example: the space used by the two relations EMPLOYEE and DEPARTMENT is less than the space for an EMP_DEPT relation shown below:

EMP_DEPT

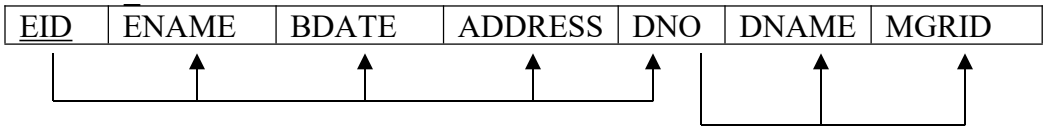


Fig: 1

In EMP_DEPT the attribute values pertaining to a particular department (DNO, DNAME, MGRID) are repeated for every employee who works for that department. While in EMPLOYEE and DEPARTMENT only the DNO is repeated in the EMPLOYEE relation for each employee who works in that department.

Another serious problem with EMP_DEPT is the problem of *update anomalies: Insertion, deletion, and modification anomalies.*

Insertion anomalies: These can be differentiated into two types:

1. To insert a new employee into EMP_DEPT, we must include either the value for all the department's attributes (DNO, DNAME, MGRID) or null (if employee does not works for a department).

   Ex.: To insert a new tuple for an employee who works in dept. no. 5,

   We must enter all the information of dept. no. 5 **correctly** so that they are **consistent** with values for department 5 in other tuples in relation.

   While in EMPLOYEE and DEPARTMENT relation, we do not have to worry about this consistency problem because we enter *only the value for DNO* in the

EMPLOYEE relation. All other information of department are recorded only **once** in the DEPARTMENT relation.

2. It is difficult to insert a new department that has no employee. The only way to do this is to place null values in the attributes for employee, but this cause a problem because EID is primary key and its value can't be null.

Deletion Anomalies: If we delete an employee tuple from EMP_DEPT relation and he is the last employee who works for that department. It means we have lost the information about that department. This problem does not occur in the separate EMPLOYEE and DEPARTMENT relation because department information is stored separately.

Modification Anomalies: In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5 – we must update the tuple of all employee who work in that department. Otherwise database will become inconsistent.

Guideline 2: Design the base relation schemas so that **no** insertion, deletion, modification anomalies occur in the relations. If any anomalies are present, note them clearly so that the programs that update the database will operate correctly.

3.Null values in tuples:

Like EMP_DEPT, we may group many attributes together into a single 'fat' relation. If many of the attributes do not have values for some tuple in the relation, we end up with many nulls in those tuples. This can waste storage space unnecessarily. Null values have multiple interpretations:

- The attribute *does not apply* to this tuple.
- The attribute value for this tuple is *unknown*.
- The value is *known but absent*.

Guideline 3: Avoid placing attributes in a base relation whose values may be null. If nulls are unavoidable, they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

4Spurious (False/Fake) tuples:

When we combine the tuple from two relations, we get spurious or wrong tuples/information that is not valid. This is because of poor relation schema design.

Guideline 4: Design relational schemas so that they can be JOINED with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

Functional Dependency (FD):

A functional dependency is a constraint between two sets of attributes from the database.

Formally, a functional dependency ($X \rightarrow Y$) between two sets of attributes X and Y that are subsets of relation schema R specifies a constraint on the possible tuples. The constraint states that for any two tuples t_1 and t_2 such that $t_1[X]=t_2[X]$, we must also have $t_1[Y]=t_2[Y]$.

This means that the values of the Y component of a tuple depends on/determined by the values of the X component (or Y is **functionally dependent** on X). Alternatively, the values of the X component of a tuple **uniquely/functionally determine** the values of the Y component. The set of attributes X is called the **left-hand side** of the FD and Y is called the **right-hand side.**

Example: consider the following relational schema: EMP_DEPT and EMP_PROJ.
In EMP_DEPT, there are two FDs:

fd1: EID→{PNO, BDATE, ADDRESS, DNO}
fd2: DNO→{DNAME, MGRID}

EMP_DEPT

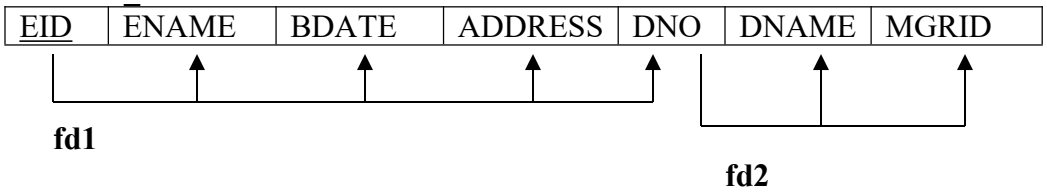


Fig: 2 (a) Relational schema and their dependencies

In EMP_PROJ, there are three FDs

fd1: {EID, PNO}→HOURS
fd2: EID→NAME
fd3: PNO→{PNAME, PLOCATION}

EMP_PROJ

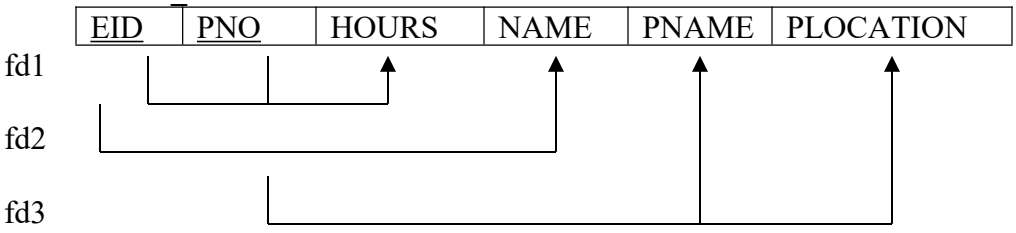


Fig: 2 (b)- Relational schema and their dependencies

Set of FDs (F): Collection of all FDs that are specified on relational schema R.

Closure of F (F^+): It is the set of all FDs that can be inferred from F.

Inference rule for FDs:

1. Reflexive rule: If $X \supseteq Y$, then $X \rightarrow Y$
2. Augmentation rule: This rule says that adding the same set of attributes to both sides of a dependency results in another valid dependency.
   $\{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ$
3. Transitive rule: Function dependencies are transitive. i.e.
   $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z$
4. Decomposition/projective rule: This says that we can remove attributes from the right-hand side of a dependency.
   $\{X \rightarrow YZ\} \Rightarrow X \rightarrow Z$
5. Union/additive rule: We can combine a set of dependencies into the single FD.
   $\{X \rightarrow Y, X \rightarrow Z\} \Rightarrow X \rightarrow YZ$
6. Pseudo-transitive rule:
   $\{X \rightarrow Y, WY \rightarrow Z\} \Rightarrow WX \rightarrow Z$

Minimal sets of FDs: A set of FDs (F) is minimal if it satisfies the following conditions:

1. Every dependency in F has a single attribute for its right hand side.
2. We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.
3. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.

Full Functional Dependency: A FD $X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the **dependency does not exist/hold any more.**

Example: In the above fig

{EID, PNO} → HOURS is a full functional dependency.

Because neither EID → HOURS nor PNO → HOURS holds

Partial Dependency: A FD $X \rightarrow Y$ is a partial dependency if removal of any attribute A from X means that the **dependency still holds**.

Example: In the above fig

{EID, PNO} → NAME is partial dependency because EID →NAME holds.

Transitive Dependency: A FD $X \rightarrow Y$ is a transitive dependency if there is a set of attributes Z that is not a subset of any key, and both $X \rightarrow Z$ and $Z \rightarrow Y$ holds.

Example: The dependency

EID → MGRID is transitive through DNO

in EMP_DEPT relation (Fig above) because both the dependency

EID → DNO and DNO → MGRID hold and DNO is not subset of the key (EID).

Normalization:

Normalization is a process in which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that posses desirable properties.

Codd's Definition: The normalization process takes a relation schema through a series of tests to 'certify' whether or not it belongs to a certain **normal form**.

One objective of the normalization is to ensure that the **update anomalies** do not occur.

Normal Forms: Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A strong definition of 3NF was proposed later by **Boyce and Codd** and is known as Boyce-Codd Normal Form (BCNF). All these normal form are based on functional dependencies among the attributes of a relation.

Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed. These are based on the concept of multi-valued dependencies and join dependencies.

Prime attribute: An attribute A in a relation R is called a prime attribute if it is a member/part of any key (candidate key) of the relation.

Nonprime attribute: If A is not a member/part of any key (candidate key) of relation R.

Example: In WORKS_ON relation both EID and PNO are prime attributes, whereas other attributes are nonprime.

WORKS_ON

EID	PNO	HOURS	…

Fig: 3

First Normal Form (1NF):

It states that the domain of attributes must include only **atomic (simple, individual) values** and the values of attribute in a tuple must be a **single value** from the domain of that attribute. It means, it is defined to disallow **multivalued attributes**, **composite attributes** and their combinations.

1NF is now considered to be the part of the formal definition of a relational. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.

Example: Consider the DEPARTMENT relation shown below; whose PK is DNO. Here, we assume that each department can have a number of locations.

DEPARTMENT

DNO	DNAME	MGRID	DLOCATION

DEPARTMENT

DNO	**DNAME**	**MGRID**	**DLOCATION**
1	Research	102	{A-Block}
4	Administration	104	{B-Block}
5	Head office	108	{C-Block, D-Block, A-Block}

Fig.-4(a)-A relation schema & its instance that is not in 1 NF

As we can see, this is not in 1NF because DLOCATION is not an atomic attribute. There are two ways to normalize it into 1NF:

1. To have a tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. In this case, the primary key becomes the combination of DNO & DLOCATION. But redundancy exists in the tuples.
2. We break up its attributes into two relations DEPARTMENT and DEPT_LOCATIONS. In DEPARTMENT relation DNO will be the PK and in DEPT_LOCATION relation PK will be the combination of DNO & DLOCATION.

DEPARTMENT

DNO	**DLOCATION**	**DNAME**	**MGRID**
1	A-Block	Research	102
4	B-Block	Administration	104
5	C-Block	Head Office	108
5	D-Block	Head Office	108
5	A-Block	Head Office	108

DEPARTMENT

DNO	**DNAME**	**MGRID**
1	Research	102
4	Administration	104
5	Head office	108

DEPT_LOCATIONS

DNO	**DLOCATION**
1	A-Block
4	B-Block
5	C-Block
5	D-Block
5	A-Block

Fig.-4(b)- 1 NF relation with & without redundancy

Second Normal form (2NF):

A relation schema R is in 2NF if every **nonprime attribute** A in R is **fully functionally dependent** on the primary key of R.

General Definition of 2NF: A relational schema R is in 2NF if every nonprime attribute A in R is not partially dependent on any key of R.

Example: The EMP_PROJ relation (fig.-7.5) is in 1NF but not in 2NF. The nonprime attribute NAME violates 2NF because of fd2. Similarly, nonprime attributes PNAME and PLOCATION because of fd3.

The functional dependencies fd2 and fd3 make NAME, PNAME, and PLOCATION partially dependent on the primary key {EID, PNO}.

To make fd2 and fd3 fully functionally dependent, the functional dependencies fd1, fd2, and fd3 lead to the decomposition of EMP_PROJ into three relation schemas EP1, EP2, and EP3.

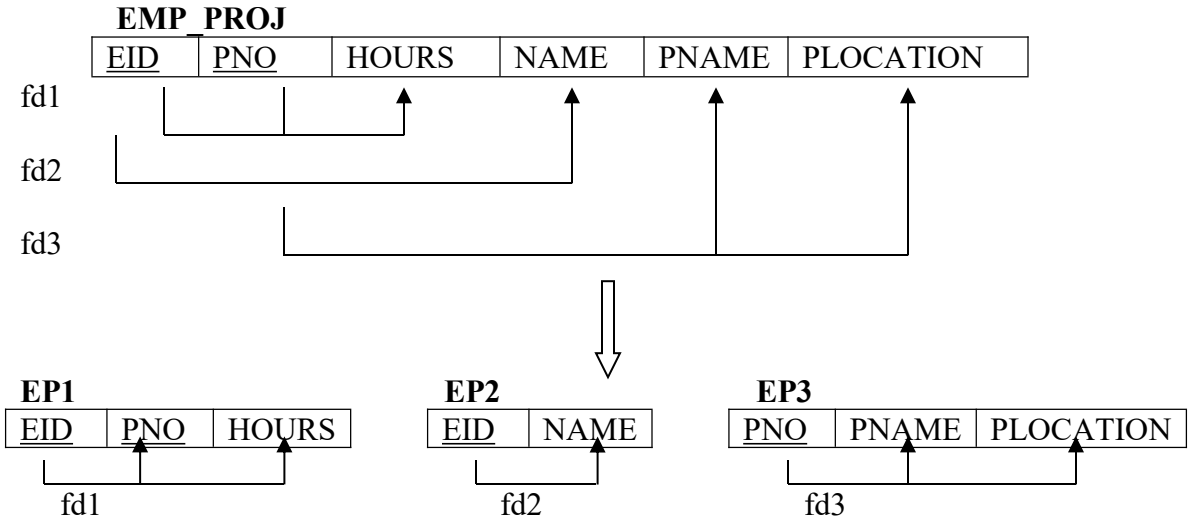


Fig: 5- Normalizing EMP_PROJ into 2NF

Third Normal Form (3NF):

3NF is based on the concept of transitive dependency.

A relational schema R is in 3NF if it in 2NF and no nonprime attribute of R is transitively dependent on primary key.

General Definition of 3NF: A relational schema R is in 3NF if whenever a functional dependency $X \rightarrow A$ holds in R, either (a) X is super key of R or (b) A is a prime attribute of R.

Example: The relational schema EMP_DEPT is in 2NF, but not in 3NF because of the transitive dependencies on MGRID (and also DNAME) on EID via DNO.

We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2.

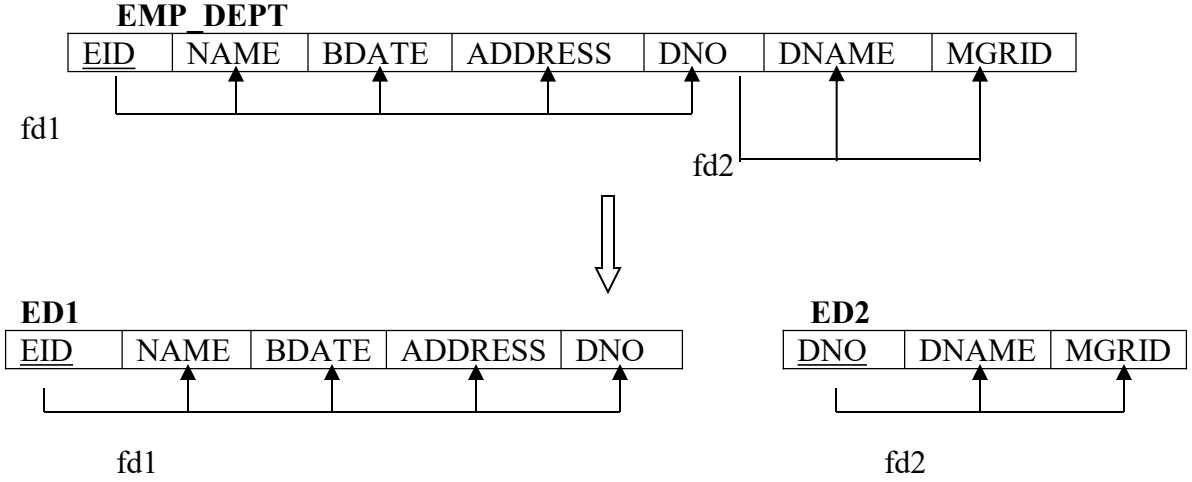


Fig: 6- Normalizing EMP_DEPT into 3 NF

Boyce-Codd Normal Form (BCNF):

A relational schema R is in BCNF if whenever a functional dependency $X \rightarrow A$ holds in R, then X is a super key of R.

Note: 1) The only difference between BCNF and 3NF is that condition (b) of 3NF (which allows A to be prime if x is not a super key) is absent from BCNF.
2) BCNF is stricter than 3NF, meaning that every relation in BCNF is also in3NF. But a relation in 3NF is **not necessarily** in BCNF.
3) In practice, most relation schemas that are in 3NF are also in BCNF.
Example: Let PLOTS be a relational schema, which have two candidate keys PROPERTY_ID# and {COUNTY_NAME, PLOT#}. Relational schema PLOTS is in 3NF {fd1 & fd2 holds due to condition (a) of definition and fd3 holds due to condition (b) of definition} but not in BCNF {because condition (b) does not exits in the definition of BCNF}. We can decompose PLOTS into two BCNF relations PLOTS1 and PLOTS2 as shown below.

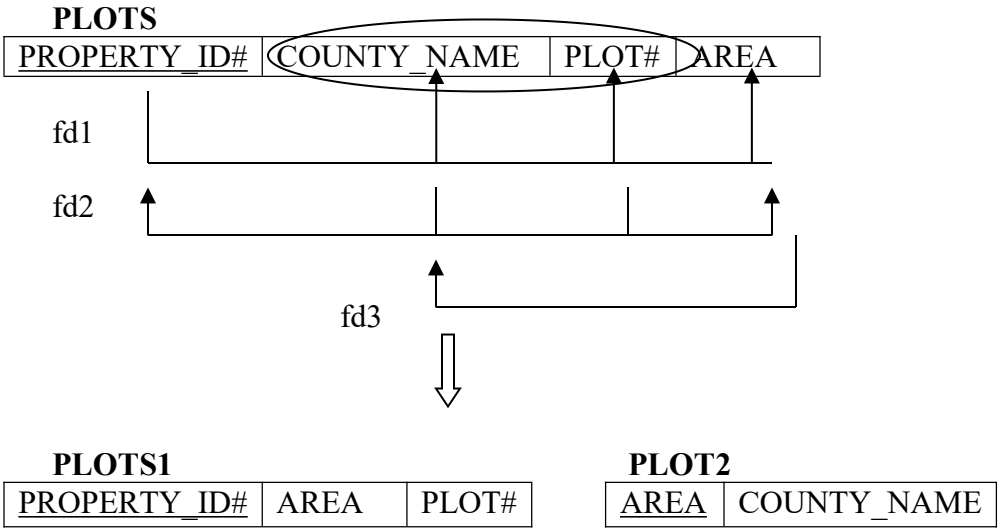


Fig. 7(a)- BCNF normalization with lost of fd2.

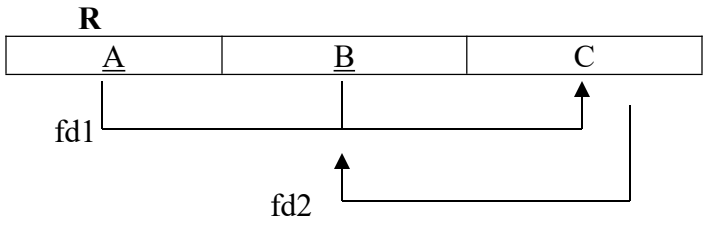


Fig. 7 (b): Relation R is in 3 NF but not in BCNF

Multi-valued Dependency (MVD):

Multi-valued dependencies are a consequence of 1NF, which disallow a tuple to have a set of values. If we have two or more multi-valued independent attribute in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation instance **consistent.** This constraint is specified by a multi-valued dependency.

Informally, whenever two independent 1:N relationships are mixed in the same relation, an MVD may arise.

Formally, a multi-valued dependency (MVD) $X \rightarrow Y$ specified on relation schema R, (where X and Y are subset of R), specifies the following constraint on any relation r of R:

If two tuples t1 and t2 exists in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exists in r with the following properties:

t3[X] = t4[X] = t1[X] = t2[X].

t3[Y] = t1[Y] and t4[Y] = t2[Y].

t3[R−(XY)] = t2[R−(XY)] and t4[R−(XY)] = t1[R−(XY)]

Whenever $X \rightarrow Y$ holds, we say that X **multi-determines** Y. $R-(XY)$ is same as $R-(X \cup Y) = Z$. Hence, $X \rightarrow Y$ implies $X \rightarrow Z$ and therefore it is sometime written as $X \rightarrow Y/Z$.

Example: Consider the relation EMP shown below. A tuple in this relation represents the fact that an employee whose name is ENAME works on the project whose name is PNAME and has a dependent whose name is DEP_NAME. An employee may work on several projects and may have several dependents. Employee's projects and dependents are not directly related to one other.

EMP

ENAME	PNAME	DEP_NAME
Rohit	X	Vikash
Rohit	Y	Shikha
Rohit	X	Shikha
Rohit	Y	Vikash

Fig 8 (a):

In the above fig.6.8 (a) two MVDs ENAME → PNAME and ENAME → DEP_NAME hold in EMP relation. The employee with ENAME 'Rohit' works on projects with PNAME 'X' and 'Y' and has two dependents with DEP_NAME 'Vikash' and 'Shikha'. This information will be stored in four tuples as shown in fig 6.8 (a).

Trivial MVD:

An MVD $X \rightarrow Y$ is called a trivial MVD if (a) Y is a subset of X or (b) $X \cup Y = R$.

For example, the relation EMP_PROJECT in fig 6.8 (b) has the trivial MVD ENAME → PNAME.

Non-trivial MVD:

An MVD that satisfies neither (a) nor (b) is called a non-trivial MVD

EMP_PROJECT

ENAME	PNAME
Rohit	X
Rohit	Y

EMP_DEPENDENT

ENAME	DNAME
Rohit	Vikash
Rohit	Shikha

Fig 8 (b):

Fourth Normal Form (4 NF):

A relation schema R is in 4NF w.r.t. a set of dependencies F if:
Either (a) $X \rightarrow Y$ is a trivial MVD
Or (b) X is a super key of R

The EMP relation of fig 6.8 (a) is not in 4NF. We decompose EMP into EMP_PROJECT and EMP_DEPENDENT as shown in fig. 6.8(b). Both EMP_PROJECT and EMP_DEPENDENT are in 4NF because ENAME $\rightarrow$ PNAME is a trivial MVD in EMP_PROJECT and ENAME $\rightarrow$ DEP_NAME is a trivial MVD in EMP_DEPENDENT.

Lossless(Nonadditive) Joins:

Lossless join ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations in the decomposition. The lossless join property is always defined w.r.t. a specific set F of dependencies.

Formally, a decomposition D={R1, R2, …, Rm} of R has the lossless (nonadditive) join property w.r.t. the set of dependencies F on R if for every relation state r of R that satisfies F, the following holds:

$$*(\pi_{<R1>}(r), \ldots, \pi_{<Rm>}(r))=r$$

The word lossless refers to *loss of information*, not to loss of tuples. If decomposition does not have the lossless join property, we may get additional spurious tuples after the PROJECT (π) and NATURAL JOIN (*) operations are applied. The term nonadditive means no wrong information is added to the result after the PROJECT and NATURAL JOIN operation are applied.

Join Dependency (JD):

A join dependency, denoted by JD(R1, R2, …, Rn) on R, specifies a constraint on instance r of R. The constraint states that every legal instance r of R should have a lossless join decomposition into R1, R2, …, Rn.
i.e. $*(\pi_{<R1>}(r), \pi_{<R2>}(r), \ldots, \pi_{<Rn>}(r)) = r$

Trivial JD:

A join dependency, denoted by JD(R1, R2, …, Rn) on R, is a trival JD if one of the relation schemas Ri in JD(R1, R2, …, Rn) is equal to R.

Trival JD has the lossless join property for any relation instance r of R and does not specify any constraint on R

Fifth Normal Form/Project-Join Normal Form (5NF/PJNF):

A relation schema R is in 5NF/PJNF w.r.t. F (set of functional, multi-valued, and join dependency) if for every JD (R1, R2, …, Rn):

Either (a) The JD is trivial

Or (b) Every Ri is a super key of R.

Example: Consider the SUPPLY relation having constraint that 'whenever a supplier s supplies part p and a prject j uses part p, and the supplier s supplies at least one part to project j, then supplier s will also be supplying part p to project j.

SUPPLY

SNAME	PART_NAME	PROJ_NAME
Rohit	Bolt	X
Rohit	Nut	Y
Amit	Bolt	Y
Vikram	Nut	Z
Amit	Nail	X
Amit	Bolt	X
Rohit	Bolt	Y

Fig. 9 (a): SUPPLY relation

If this constraint holds, the tuples below the dotted line in the above fig must exist in any legal instance of the SUPPLY relation.

This constraint can be restated in other way and specifies a join dependency JD (R1, R2, R3) among the three projections R1(SNAME, PART_NAME), R2(SNAME, PROJ_NAME), and R3(PART_NAME, PROJ_NAME) of SUPPLY.

Fig 6.9 (b) shows how the SUPPLY relation with the join dependency is decomposed into three relations R1, R2, and R3 that are each in 5NF.

R1

SNAME	PART_NAME
Rohit	Bolt
Rohit	Nut
Amit	Bolt
Vikram	Nut
Amit	Nail

R2

SNAME	PROJ_NAME
Rohit	X
Rohit	Y
Amit	Y
Vikram	Z
Amit	X

R3

PART_NAME	PROJ_NAME
Bolt	X
Nut	Y
Bolt	Y
Nut	Z
Nail	X

Fig 9 (b): Decomposition into 5NF

Canonical Cover

In database management systems (DBMS), a canonical cover is a set of functional dependencies that is equivalent to a given set of functional dependencies but is minimal in terms of the number of dependencies. The process of finding the canonical cover of a set of functional dependencies involves three main steps:

- **Reduction:** The first step is to reduce the original set of functional dependencies to an equivalent set that has the same closure as the original set, but with fewer dependencies. This is done by removing redundant dependencies and combining dependencies that have common attributes on the left-hand side.
- **Elimination:** The second step is to eliminate any extraneous attributes from the left-hand side of the dependencies. An attribute is considered extraneous if it can be removed from the left-hand side without changing the closure of the dependencies.
- **Minimization:** The final step is to minimize the number of dependencies by removing any dependencies that are implied by other dependencies in the set.

Example

Consider a set of Functional dependencies: F={A→BC, B→C, AB→C}. Here are the steps to find the canonical cover –

Step 1:Decompose FDs to have a single attribute on the right-hand side

- A→BC becomes A→B and A→C.
- Therefore, we have {A→B, A→C, B→C, AB→C}.

Step 2:Remove extraneous attributes from the left-hand side of FDs

- Checking AB→C First, check if A or B is extraneous.
- We can reach C without using AB→C with other functional dependencies; therefore, we remove AB→C.
- Finally, we have {A→B, A→C, B→C}.

Step 3:Remove redundant FDs

- Check each functional dependency to see if it can be reached without using it. For example, A→C can be reached with A→B and B→C. Therefore, A→C is redundant and can be removed.
- Hence, Canonical Cover = {A→B, B→C}.

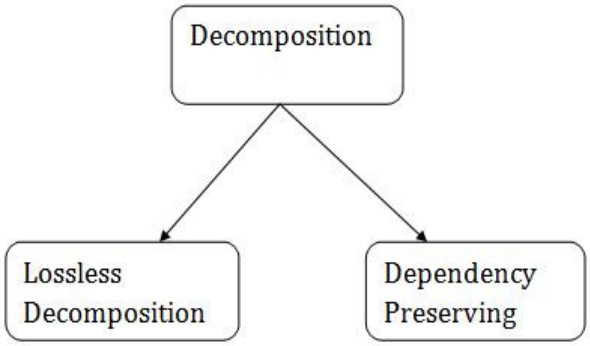
Extraneous attributes

An attribute of an FD is said to be extraneous if we can remove it without changing the closure of the set of FD.

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Join Decomposition

If we decompose a relation R into relations R1 and R2, ⋈ is natural join.

Decomposition is **lossy** if R1 ⋈ R2 ⊃ R

Decomposition is **lossless** if R1 ⋈ R2 = R

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

1. The Union of Attributes of R1 and R2 must be equal to the attribute of R. Each attribute of R must be either in R1 or in R2.

   Att(R1) U Att(R2) = Att(R)

2. The intersection of Attributes of R1 and R2 must not be NULL.

   Att(R1) ∩ Att(R2) ≠ Φ

3. The 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:

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. The third condition holds as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.

Dependency Preserving Decomposition

Let R is decomposed into {R1, R2,....,Rn} with projected FD set {F1,F2,.....Fn}. This decomposition is dependency preserving if $F^+ = \{F1 \cup F2 \cup .....Fn\}^+$.

Example

Let the relation R{A,B,C,D,E} F:{AB->C, C->D, AB->D} R is decomposed to R1(A,B,C), R2(D,E). Prove decomposition is dependency preserving.

F1={AB->C}

F2={C->D}

=> (F1 u F2) = {AB->C, C->D}

AB+ under (F1 U F2) = {A,B,C,D} => AB->D is under (F1 U F2)

$F+ = (F1 \cup F2)^+$

=> Decomposition is **dependency preserving**.

Let the relation R{A,B,C,D,E,F,G,H,I,J} where

F: {AB->C, A->DE, B->F, F->GH. D->IJ}

R is decomposed to R1(A,B,C,D), R2(D,E), R3(B,F), R4(F,G,H) AND R5(D,I,J). Check decomposition is dependency preserving or not.

F1={AB->C} F4={F->GH}

F2={} F5={D->IJ}

F3={B->F}

=> (F1 U F2 U F3 U F4 U F5) = {AB->C, B->F, F->GH, D->IJ}

A+ under (F1 U F2 U F3 U F4 U F5) = {AB->C, B->F, F->GH, D->IJ}

=>A->DE is not under (F1 U F2 UF3 U F4 U F5)

=>F+ ≠ $(F1 \cup F2 \cup F3 \cup F4 \cup F5)^{+}$

=> Decomposition is **not dependency preserving**.

If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from a combination of functional dependency 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).

Advantages of Lossless Join and Dependency Preserving Decomposition

- **Improved Data Integrity:** Lossless join and dependency preserving decomposition help to maintain the data integrity of the original relation by ensuring that all dependencies are preserved.
- **Reduced Data Redundancy:** These techniques help to reduce data redundancy by breaking down a relation into smaller, more manageable relations.
- **Improved Query Performance:** By breaking down a relation into smaller, more focused relations, query performance can be improved.
- **Easier Maintenance and Updates:** The smaller, more focused relations are easier to maintain and update than the original relation, making it easier to modify the database schema and update the data.
- **Better Flexibility:** Lossless join and dependency preserving decomposition can improve the flexibility of the database system by allowing for easier modification of the schema.

Disadvantages of Lossless Join and Dependency Preserving Decomposition

- **Increased Complexity:** Lossless join and dependency-preserving decomposition can increase the complexity of the database system, making it harder to understand and manage.
- **Costly:** Decomposing relations can be costly, especially if the database is large and complex. This can require additional resources, such as hardware and personnel.
- **Reduced Performance:** Although query performance can be improved in some cases, in others, lossless join and dependency-preserving decomposition can result in reduced query performance due to the need for additional join operations.
- **Limited Scalability:** These techniques may not scale well in larger databases, as the number of smaller, focused relations can become unwieldy.