APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

STUDY MATERIALS

MODULE IV

Different Anomalies in Designing a Database, The Idea of Normalization, Functional Dependency, Armstrong's Axioms (Proofs Not Required), Closures and Their Computation, Equivalence of Functional Dependencies (FD), Minimal Cover (Proofs Not Required).

First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce Codd Normal Form (BCNF), Lossless Join and Dependency Preserving Decomposition, Algorithms for Checking Lossless Join (LJ) and Dependency Preserving (DP) Properties.

Reference:

Elmasri R. and S. Navathe, Database Systems: Models, Languages, Design and Application Programming, Pearson Education, 2013.

Different Anomalies in Designing a Database

Consider the simplified relational schema of COMPANY database.

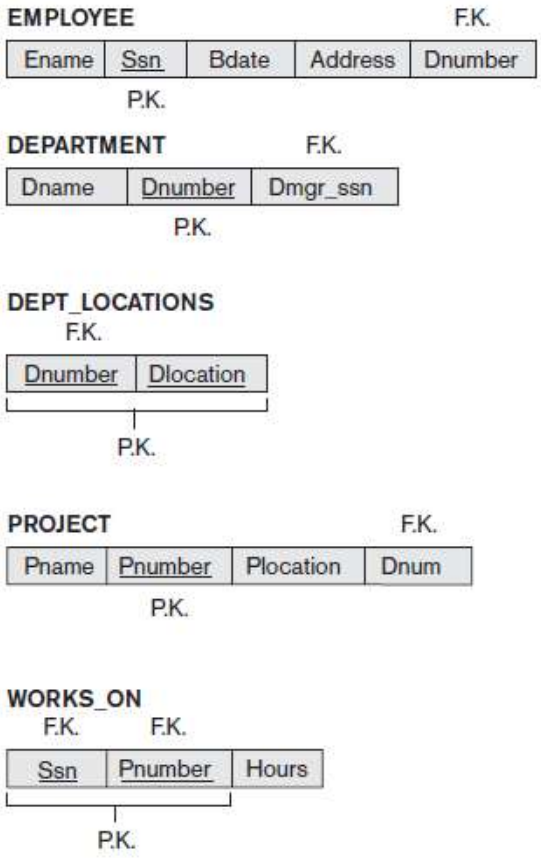
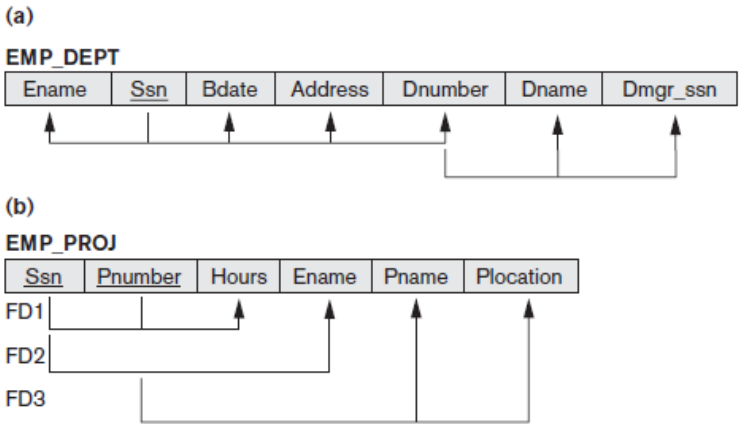


Figure 15.1
A simplified COMPANY relational database schema.

Suppose we are applying the NATURAL JOIN operator to EMPLOYEE and DEPARTMENT relations and also to EMPLOYEE and PROJECT and WORKS_ON relations.



(The students should ignore the lines under the relations for now; they are used to illustrate functional dependency notation, discussed in the next section)

A tuple in the EMP_DEPT relation schema represents a single employee but includes additional information—namely, the name (Dname) of the department for which the employee works and the Social Security number (Dmgr_ssn) of the department manager. For the EMP_PROJ relation, each tuple relates an employee to a project but also includes the employee name (Ename), project name (Pname), and project location (Plocation).

Although there is nothing wrong logically with these two relations, they are mixing attributes from distinct real-world entities: EMP_DEPT mixes attributes of employees and departments, and EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship. They may be used as views, but they cause problems when used as base relations.

Storing natural joins of base relations leads to an additional problem referred to as **update anomalies**. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies. Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

- To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for a department as yet). For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are *consistent* with the corresponding values for department 5 in other tuples in EMP_DEPT. In the design of Figure 1, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation.
- It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP_DEPT because Ssn is its primary key.Moreover, when the first employee is assigned to that department, we do not need this tuple with NULL values any more. This problem does not occur in the design of Figure 1 because a department is entered in the DEPARTMENT relation whether or not any employees work for it, and whenever an employee is assigned to that department, a corresponding tuple is inserted in EMPLOYEE.

Deletion Anomalies. The problem of deletion anomalies is related to the second insertion anomaly situation just discussed. If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database. This problem does not occur in the database of Figure 1 because DEPARTMENT tuples are 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 tuples of *all* employees who work in that department; otherwise, the database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong.

Normalization of Relations

The normalization process, as first proposed by Codd, takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**. The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as *relational design by analysis.* Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively.

Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

(1) minimizing redundancy and

(2) minimizing the insertion, deletion, and update anomalies.

It can be considered as a "filtering" or "purification" process to make the design have successively better quality. Unsatisfactory relation schemas that do not meet certain conditions—the **normal form tests**—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.

Normalization is the successive reduction of relation to evolve more desirable relation

- ***Reversibility should be possible***
- ***No information is lost***
- ***Should not generate extra information***

Definition. The **normal form** of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

Functional Dependencies

A formal tool for analysis of relational schemas that enables us to detect and describe some of the above-mentioned problems in precise terms. The single most important concept in relational schema design theory is that of a functional dependency.

Definition of Functional Dependency

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes $A1, A2, ..., An$; let us think of the whole database as being described by a single **universal** relation schema $R = \{A1, A2, ... , An\}$

Definition. A **functional dependency**, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a *constraint* on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples $t1$ and $t2$ in r that have $t1[X] = t2[X]$, they must also have $t1[Y] = t2[Y]$.

This means that the values of the Y component of a tuple in r depend on, or are *determined by*, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or **functionally**) *determine* the values of the Y component.
We also say that there is a functional dependency from X to Y, or that Y is **functionally dependent** on X. The abbreviation for functional dependency is **FD** or **f.d.** The set of attributes X is called the **left-hand side** of the FD, and Y is called the **right-hand side**.

Definitions of Keys and Attributes Participating in Keys

Definition. A **superkey** of a relation schema $R = \{A1, A2, ... , An\}$ is a set of attributes $S \subseteq R$ with the property that no two tuples $t1$ and $t2$ in any legal relation state r of R will have $t1[S] = t2[S]$.
A **key** K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey any more.
The difference between a key and a superkey is that a key has to be *minimal;* that is, if we have a key $K = \{A1, A2, ..., Ak\}$ of R, then $K - \{Ai\}$ is not a key of R for any Ai, $1 \leq i \leq k$.

{Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys.

If a relation schema has more than one key, each is called a **candidate key**. One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called secondary keys. In a practical relational database, each relation schema must have a primary key.

Definition. An attribute of relation schema *R* is called a **prime attribute** of *R* if it is a member of *some candidate key* of *R*. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key.

Functional Dependencies

Single most important concept in relational schema design theory.

Definition

A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$ they must also have $t_1[Y] = t_2[Y]$.

Eg:- Consider the relation

Ssn	Last	First
111	Smith	Bob
222	Jones	David
333	Smith	Joe
111	Smith	Bob
444	Jones	Sue
555	White	David

Ssn → Last
Ssn → First
} Valid FDs

Last → First
Not a valid FD

Closure

Set of all FDs that include F as well as all the dependencies that can be inferred

from F.

Denoted as F^+.

Consider the relation R(ABC)

Let $F = \{A \rightarrow B, B \rightarrow C\}$

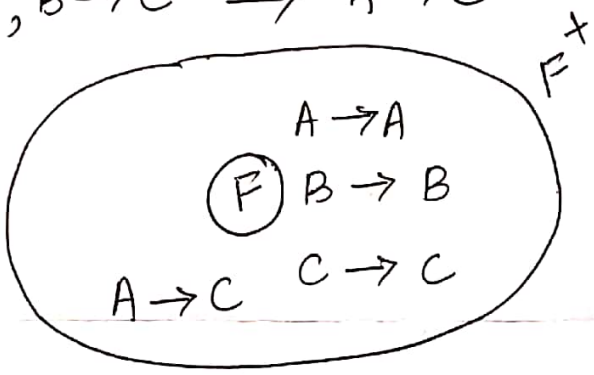
Using reflexivity, we can infer

$A \rightarrow A$

$B \rightarrow B$

$C \rightarrow C$

$A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$



∴ F logically implies F^+.

In the case of large databases, F^+ is huge. So we need mechanisms to see if an FD is in F^+.

Armstrong's Axioms

1) Reflexivity

   ABC ABC ; $ABC \rightarrow AB$

2) Augmentation

   $A \rightarrow B$; $AC \rightarrow BC$

3) Transitivity

   $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$

Armstrong's Axioms are sound.

↳ if we apply AA to FDs in F, we get FDs which are in F^+.

Armstrong's Axioms are complete.

↳ if we repeatedly apply AA to FDs in F, we get the complete FDs in F^+.

From AA, we can derive Inference Rules (IR)

IR1

Union

$$\frac{X \rightarrow Y,\ X \rightarrow Z}{X \rightarrow YZ}$$

IR2

Pseudotransitivity

$$\frac{X \rightarrow Y,\ WY \rightarrow Z}{WX \rightarrow Z}$$

IR3

Decomposition

$$\frac{X \rightarrow YZ}{X \rightarrow Y,\ X \rightarrow Z}$$

Equivalence of Sets of Functional Dependencies

Cover

A set of functional dependencies F is said to cover another set of functional dependencies E if every FD in E is also in F^+.

Definition

Two sets of functional dependencies E and F are equivalent if $E^+ = F^+$.

- that means every FD in E can be inferred from F and every FD in F can be inferred from E. ie., if both E covers F and F covers E hold.

ie., $E^+ \equiv F^+$

Eg:- Let R(ABC) be a given relation.

$F = \{A \to B, B \to C, C \to A\}$

$G = \{C \to B, B \to A, A \to C\}$

Nothing in common between F and G.

To show that $F \equiv G$, we have to show that

1) $F \subseteq G^+$ 2) $G \subseteq F^+$

For that, take every FD in F, take its LHS and find its closure with respect to G.

A^+ = ACB, B^+ = BAC, C^+ = CBA $\Rightarrow F \subseteq G^+$

$\therefore$ G covers F.

$\frac{C^+}{CAB}$ $\frac{B^+}{BCA}$ $\frac{A^+}{ABC}$ $\Rightarrow G \subseteq F^+$

$\therefore$ F covers G.

$\therefore F \equiv G$

Minimal Covers

To prevent the d/b becoming inefficient, we should revise FDs.

Eg:- R(ABC)

$F = \{A \to B, AB \to C\}$

$\frac{A^+}{ABC} \Rightarrow A \to B, A \to C$

$\therefore$ We can eliminate B from LHS of $AB \to C$.

Eg:- R(ABC)

$F = \{A \to B, B \to C, A \to C\}$

$\frac{A^+}{ABC}$

$A \to C$ is obvious. $\therefore$ We can delete it.

Formal Definition

A set of FDs F is minimal if

1. Every FD in F has a singleton RHS.
2. No extraneous LHS attributes.
3. No redundant FDs. (that can be inferred from other FDs.)

Eg:- R (ABCDE)

$F = \{A \to D, BC \to AD, C \to B, E \to A, E \to D\}$

Singleton RHS

$BC \to A$

$BC \to D$

Condition 2

① $BC \to A$

② $BC \to D$

$\frac{B^+}{B}$ $\quad \frac{C^+}{CBAD}$ $\quad$ For ①

⇓

This includes B.

∴ Eliminate B from $BC \to A$

For ②

$\frac{B^+}{B}$ $\quad \frac{C^+}{CABD}$

⇓

includes B

∴ Eliminate B from $BC \to D$.

Condition 3

Take each FD

1) $A \to D$ $\quad \frac{A^+}{A}$ $\quad$ ∴ Keep $A \to D$

2) $C \to A$ $\quad \frac{C^+}{CDB}$ $\quad \Rightarrow$ does not include A.

∴ Keep $C \to A$

3) $C \rightarrow D$ $\quad \frac{C^+}{CADB} \Rightarrow$ include D

$\therefore$ Eliminate $C \rightarrow D$

4) $C \rightarrow B$ $\quad \frac{C^+}{CAB} \Rightarrow$ Only B available even if it includes B.

$\therefore$ Keep $C \rightarrow B$.

5) $E \rightarrow A$ $\quad \frac{E^+}{ED}$ $\therefore$ Keep $E \rightarrow A$

6) $E \rightarrow D$ $\quad \frac{E^+}{EAD} \Rightarrow$ Includes D.

$\therefore$ Eliminate $E \rightarrow D$.

$\therefore$ Minimal Cover of $F = \{A \rightarrow D, C \rightarrow A, C \rightarrow B, E \rightarrow A\}$

There can be 2 or more minimal covers for a set of FDs.

How to determine key values from functional dependencies

Prime attribute – An attribute is prime if it is part of any key.

Non-prime attribute – An attribute is non-prime if it is not part of any key.

We use LMR method to find the key.

If an attribute is present only on the LHS of given sets of FDs it is included in L.

If an attribute is present on both LHS and RHS, it is included in M.

If an attribute is present only on RHS, it is in R.

Attributes coming under L are part of every key.

Attributes coming under R are not part of any key.

To find the key, the closure of attributes in L ~~per~~ is taken. If closure contains all the attributes of R, it is a key.

Otherwise attributes in M are also considered.

Eg:- R(ABC) be a relation.

Let $F = \{A \to B, B \to C\}$

Find the key.

L	M	R
A	B	C

A^+

ABC

∴ A is key.

Prime

A

NP

BC

Eg:-

Eg:- R(ABCD)

$F = \{AB \rightarrow C, C \rightarrow B, C \rightarrow D\}$

L	M	R
A	BC	D

A^+	AB^+	AC^+
A	ABCD	ACBD

∴ AB and AC are keys.

P	NP
ABC	D

Eg:- R(ABC)

$F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

L	M	R
	ABC	

A, B, C → keys

A^+	B^+	C^+
ABC	BCA	CAB

P	NP
ABC	–

Types of Functional Dependencies

1. Trivial Functional Dependency

* $A \rightarrow B$ has trivial functional dependency if B is a subset of A.
* The following dependencies are also trivial.

$A \rightarrow A$
$B \rightarrow B$

Eg:-
Consider a table

Employee_id	Emp_name

{Employee_id, Emp_name} → Employee_id is a trivial FD as Employee_id is a subset of {Employee_id, Emp_name}

Also Employee_id → Employee_id
Emp_name → Emp_name

are trivial FDs.

2. Non-trivial Functional Dependency

* $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.
* When A intersection B is null, then $A \rightarrow B$ is called as complete non-trivial.

Eg:-

ID → Name

Name → DOB

Fully Functional Dependency

An attribute is fully functional dependent on another attribute, if it is functionally dependent on that attribute and not on any of its proper subset.

Consider X → Y

Any proper subset of X should not determine Y.

Consider ABC → D

Then A → D, B → D, C → D, AB → D, BC → D or AC → D should not hold.

Partial Functional Dependency

Partial FD occurs when a non-prime attribute is functionally dependent on part of a candidate key.

The 2nd Normal Form (2NF) eliminates the partial dependency.

Consider the FD

$$AB \rightarrow C$$

Suppose AB is the primary key.

If $A \rightarrow C$ or $B \rightarrow C$ holds, then it is said to be partially functionally dependent.

Transitive Functional Dependency

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

$X \rightarrow Z$ is a transitive FD, if $X \rightarrow Y$ and $Y \rightarrow Z$ holds.

Note: A transitive dependency can only occur in a relation of three or more attributes. This dependency helps in normalizing the database in 3NF.

Closures

The set of all dependencies that include F as well as all dependencies that can be inferred from F is called the closure of F, it is denoted by F+
Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

Algorithm to Find the Attribute Closure:

attribute closure()
{
X+ = X
Repeat
{
for each FD U→V in F do
If U ⊆ X+ then
X+ = X+ U V
Until no change
}
}

Several Uses of Attribute Closure:

1. Used to check whether an attribute (or set of attributes) forms a key or not.
2. Used to find easily all the functional dependencies hold in a relation.
3. Used as an alternate way to find Closure of FDs (F+).

FIRST NORMAL FORM

- states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.

- disallows multivalued attributes, composite attributes and their combinations.

Eg:- Consider the relation

Course	Student
Database	Bob
	Joe
	Sue
Maths	Tim
	Mary

To make it in 1NF

Course	Student
Database	Bob
Database	Joe
Database	Sue
Maths	Tim
Maths	Mary

SECOND NORMAL FORM (2NF)

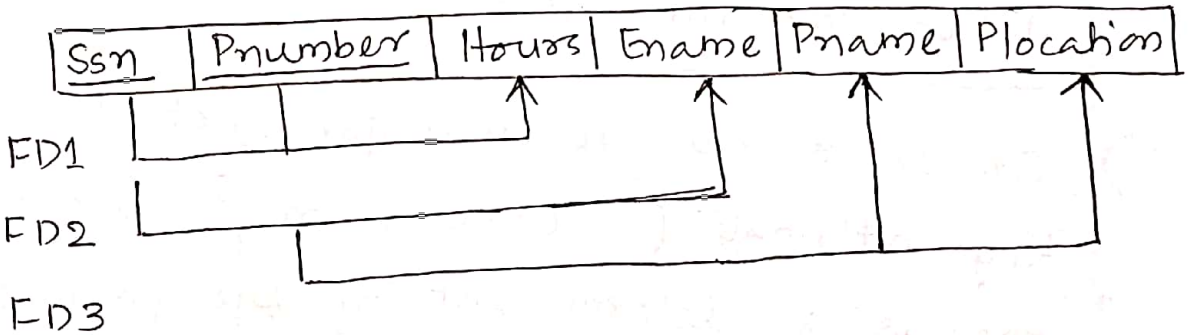
– is based on the concept of full functional dependency.

A functional dependency $X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute $A \in X$, $(X - \{A\})$ does not functionally determine Y.

A functional dependency $X \rightarrow Y$ is a partial dependency if some attribute A can be removed from X and the dependency still holds; ie; for some $A \in X$, $(X - \{A\}) \rightarrow Y$.

Eg:- Consider the relation

EMP_PROJ



{Ssn, Pnumber} → Hours Full FD

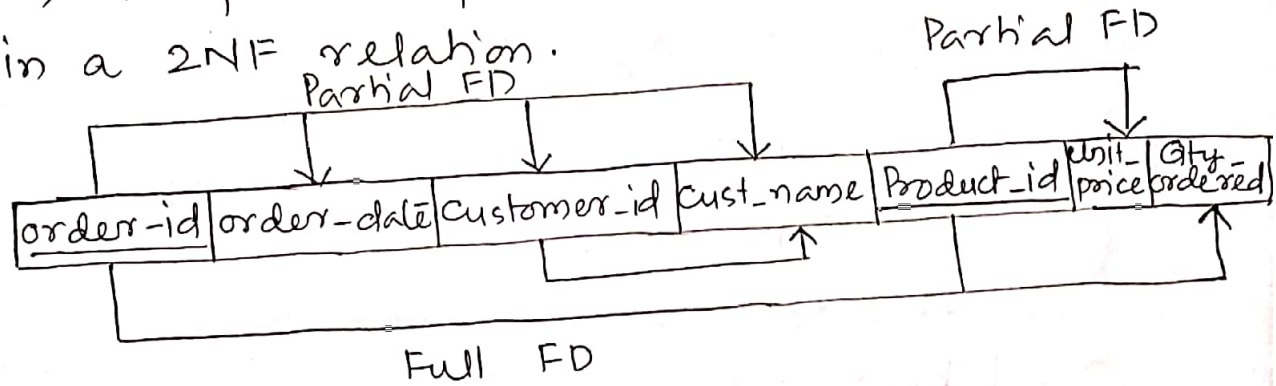
{Ssn} → Ename Partial FD.

{Pnumber} → Pname, Plocation Partial FD

A relation is in 2NF if it is in 1NF and every non-key attribute is fully functionally dependent on the ENTIRE primary key.

→ every non-key attribute must be defined by the entire key, not by only part of the key.

→ no partial functional dependencies exist in a 2NF relation.

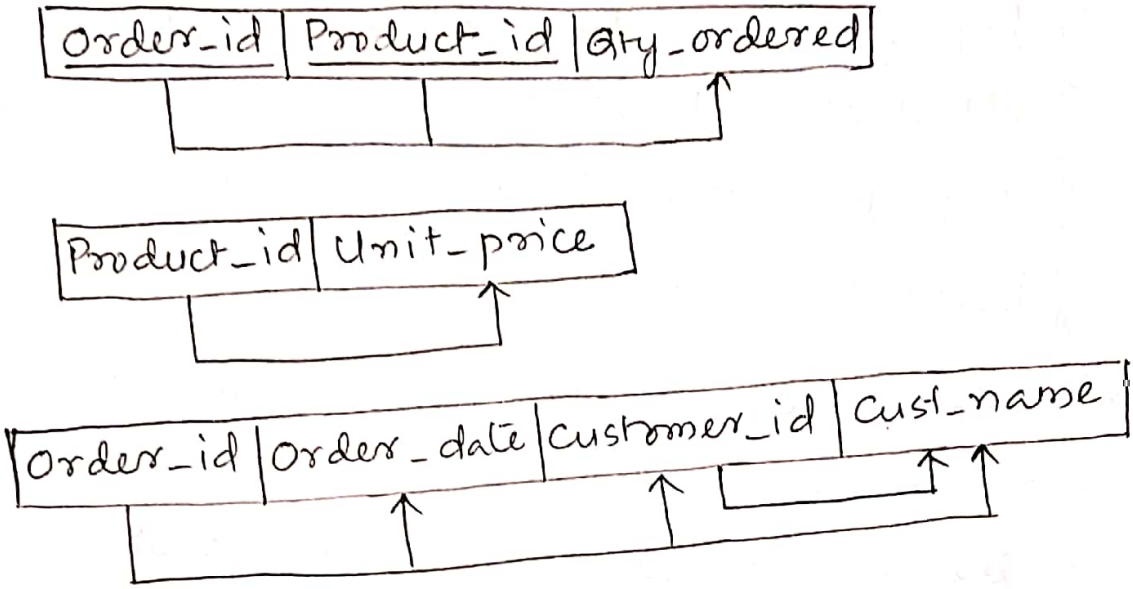


Partial FD exists when there is a composite primary key.

To convert into 2NF

1) Create a new relation for each primary key attribute (or combination of attributes) that is a determinant in the partial FD. That attribute is the 1° key in the new relation.
2) Move the non-key attributes that are dependent on this 1° key attributes/s

from the old relation to new relation.



Third Normal Form (3NF)

- based on the concept of transitive dependency.

A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. (An FD between 2 (or more) non-key attributes)

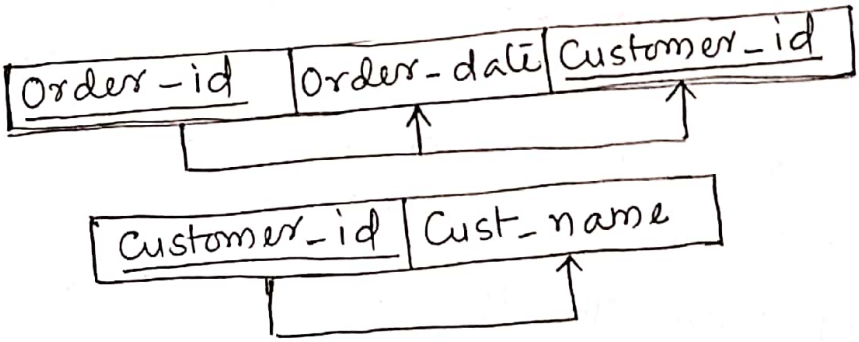
A relation schema R is in 3NF if it satisfies 2NF and no non prime attribute of R is transitively dependent on the primary key.

To convert into 3NF

1) For each non-key attribute (or set of attributes) that is a determinant in the FD, create a new relation. That becomes the 1° key in the new relation.

2) Move all of the attributes that are functionally dependent on the attribute from the old relation into the new relation.

3) Leave the attribute (which serves as the 1° key in the new relation) in the old relation to serve as a foreign key that allows an association between the two relations.



For every FD $X \rightarrow A$, if A is non-prime, then X must be a super key.

BOYCE CODD NORMAL FORM (BCNF)

- Stricter than 3NF.

Every relation in BCNF is also in 3NF; however a relation in 3NF is not necessarily in BCNF.

A relation schema R is in BCNF if whenever a nontrivial FD $X \rightarrow A$ holds in R, then X is a superkey of R.

Shortcuts

1) All attributes are prime $\Rightarrow$ R is in atleast 3NF.
2) Singleton keys $\Rightarrow$ R is in atleast 2NF.

If a relation has only 1 candidate key,

3NF $\equiv$ BCNF.

Properties of Relational Decompositions

Looking at an *individual* relation to test whether it is in a higher normal form does not, on its own, guarantee a good design; rather, a *set of relations* that together form the relational database schema must possess certain additional properties to ensure a good design.

Relation Decomposition and Insufficiency of Normal Forms

The relational database design algorithms start from a single **universal relation schema** $R = \{A1, A2, ..., An\}$ that includes *all* the attributes of the database. We implicitly make the **universal relation assumption**, which states that every attribute name is unique. The set F of functional dependencies that should hold on the attributes of R is specified by the database designers and is made available to the design algorithms. Using the functional dependencies, the algorithms decompose the universal relation schema R into a set of relation schemas $D = \{R1, R2, ..., Rm\}$ that will become the relational database schema; D is called a **decomposition** of R.

We must make sure that each attribute in R will appear in at least one relation schema Ri in the decomposition so that no attributes are *lost*; formally, we have

$$\bigcup_{i=1}^{m} R_i = R$$

This is called the **attribute preservation** condition of a decomposition.

Dependency Preservation Property of a Decomposition

It would be useful if each functional dependency $X \rightarrow Y$ specified in F either appeared directly in one of the relation schemas Ri in the decomposition D or could be inferred from the dependencies that appear in some Ri. Informally, this is the ***dependency preservation condition.*** We want to preserve the dependencies because each dependency in F represents a constraint on the database. If one of the dependencies is not represented in some individual relation Ri of the decomposition, we cannot enforce this constraint by dealing with an individual relation.

If a decomposition is not dependency-preserving, some dependency is **lost** in the decomposition. To check that a lost dependency holds, we must take the JOIN of two or more relations in the decomposition to get a relation that includes all left and right-hand-side attributes of the lost

dependency, and then check that the dependency holds on the result of the JOIN—an option that is not practical.

Nonadditive (Lossless) Join Property of a Decomposition

Another property that a decomposition *D* should possess is the nonadditive join property, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations resulting from the decomposition.

The word loss in *lossless* refers to *loss of information*, not to loss of tuples. If a decomposition does not have the lossless join property, we may get additional spurious tuples after the PROJECT (π) and NATURAL JOIN (*) operations are applied; these additional tuples represent erroneous or invalid information.

The nonadditive join property ensures that no spurious tuples result after the application of PROJECT and JOIN operations. We may, however, sometimes use the term **lossy design** to refer to a design that represents a loss of information.

The decomposition of EMP_PROJ(Ssn, Pnumber, Hours, Ename, Pname, Plocation) in Figure 1 into EMP_LOCS(Ename, Plocation) and EMP_PROJ1(Ssn, Pnumber, Hours, Pname, Plocation) in Figure 2 obviously does not have the nonadditive join property, as illustrated by Figure 3.

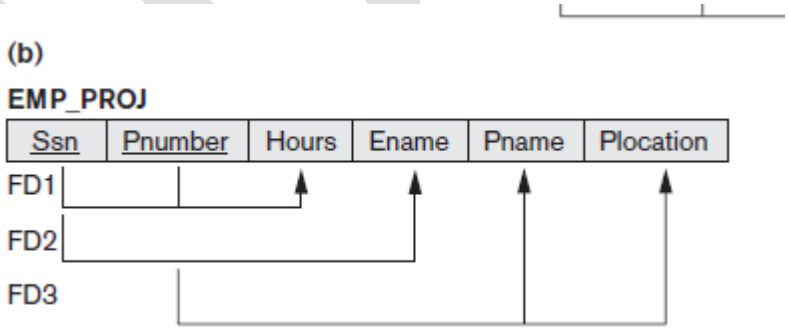


Figure 1

(a)

EMP_LOCS

Ename	Plocation

P.K.

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation

P.K.

Figure 15.5
Particularly poor design for the EMP_PROJ relation in Figure 15.3(b). (a) The two relation schemas EMP_LOCS and EMP_PROJ1. (b) The result of projecting the extension of EMP_PROJ from Figure 15.4 onto the relations EMP_LOCS and EMP_PROJ1.

(b)

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

Figure 2

	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

Figure 15.6
Result of applying NATURAL JOIN to the tuples above the dashed lines in EMP_PROJ1 and EMP_LOCS of Figure 15.5. Generated spurious tuples are marked by asterisks.

Figure 3

Algorithm for Testing Nonadditive Join Property

Input: A universal relation R, a decomposition D = {R1, R2, . . . , Rm} of R, and a set F of FDs

1. Create an initial matrix S with one row i for each relation *Ri* in D, and one column *j* for each attribute *Aj* in *R*.

2. Set *S(i, j):= bij* for all matrix entries.

3. For each row i representing relation schema *Ri*

{for each column j representing attribute *Aj*

{if (relation *R*, includes attribute *Aj)* then set *S(i, j):= aj;};};

4. Repeat the following loop until a *complete loop execution* results in no changes to S

{for each functional dependency X → Y in F

{for all rows in *S that have the same symbols* in the columns corresponding to attributes in X

{make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:

If any of the rows has an *a* symbol for the column, set the other rows to that *same a* symbol in the column.

If no *a* symbol exists for the attribute in any of the rows, choose one of the *b* symbols that appears in one of the rows for the attribute and set the other rows to that same *b* symbol in the column

;} ;} ; } ;

5. If a row is made up entirely of *a* symbols, then the decomposition has the non-additive join property; otherwise, it does not.

Example:

Consider the schema R=ABCD, subjected to FDs F= { A → B, B → C }, and the Non-binary partition D1 = {ACD, AB, BC}. Check the decomposition is lossless

	A	B	C	D
ACD	b_{11}	b_{12}	b_{13}	b_{14}
AB	b_{21}	b_{22}	b_{23}	b_{24}
BC	b_{31}	b_{32}	b_{33}	b_{34}

Step 1. Create initial table, entering $\mathbf{b_{ij}}$ values in each cell

	A	B	C	D
ACD	a_1	b_{12}	a_3	a_4
AB	a_1	a_2	b_{23}	b_{24}
BC	b_{31}	a_2	a_3	b_{34}

Step 2. For each attribute mentioned in each partition R*i* introduce $\mathbf{a_j}$ on column j

	A	B	C	D
ACD	a_1	a_2	a_3	a_4
AB	a_1	a_2	a_3	b_{24}
BC	b_{31}	a_2	a_3	b_{34}

Step 3. Apply FDs to unify $\mathbf{a_j}$, $\mathbf{b_{ij}}$ symbols.
Using B→C we discover $a_2 \rightarrow \{a_3, b_{23}\}$
therefore b_{23} becomes a_3
Using A→B we discover $a_1 \rightarrow \{a_2, b_{12}\}$
hence b_{12} can be replaced by a_2.
First row now has a_j values in each cell. Stop!

Conclusion: Decomposition D_1= {ACD, AB, BC} on <R, F> has a lossless-join

Consider the schema R=ABCD, subjected to FDs F= { A→B, B→C }, and the Non-binary partition D2 = {AB, BC, CD}. Check D2 a Lossless decomposition?

	A	B	C	D
AB	b_{11}	b_{12}	b_{13}	b_{14}
BC	b_{21}	b_{22}	b_{23}	b_{24}
CD	b_{31}	b_{32}	b_{33}	b_{34}

Step 1. Create initial table, entering $\mathbf{b_{ij}}$ values in each cell

	A	B	C	D
AB	a_1	a_2	b_{13}	b_{14}
BC	b_{21}	a_2	a_3	b_{24}
CD	b_{31}	b_{32}	a_3	a_4

Step 2. For each attribute mentioned in each partition R_i introduce $\mathbf{a_j}$ on column j

	A	B	C	D
AB	a_1	a_2	a_3	b_{14}
BC	b_{21}	a_2	a_3	b_{24}
CD	b_{31}	b_{32}	a_3	a_4

Step 3. Apply FDs to unify $\mathbf{a_j}$, $\mathbf{b_{ij}}$ symbols.
Using B→C we discover $a_2 \rightarrow \{a_3, b_{13}\}$ therefore b_{13} becomes a_3.
Using A→B is of no help in discovering new equivalent classes. There are no more FDs to apply and no row shows all a-values. STOP

Conclusion: Decomposition D_2= {AB, BC, CD} on <R, F> is NOT lossless.

try it now

A KTU STUDENTS PLATFORM

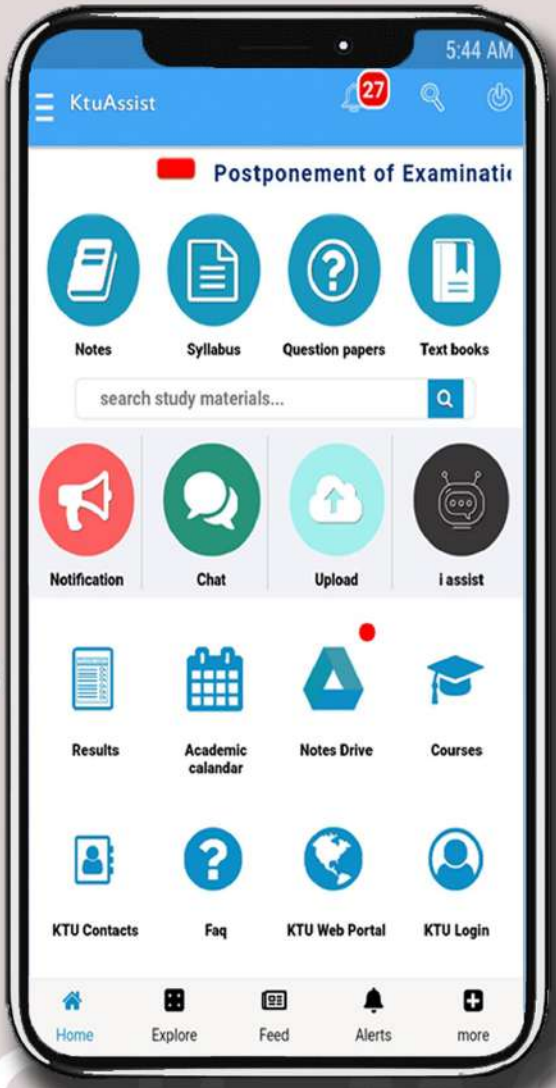
SYLLABUS

NOTES

TEXT BOOKS

QUESTION PAPERS

KTU NOTIFICATION



DOWNLOAD IT FROM GOOGLE PLAY

CHAT

FAQ

LOGIN

CALENDAR

MUCH MORE

DOWNLOAD APP

ktuassist.in

instagram.com/ktu_assist

facebook.com/ktuassist