LECTURE 2

Chapter 6

FUNCTIONAL DEPENDENCY AND NORMALIZATION

OUTLINES

- Introduction
- Informal design guidelines for relation schemas
- Functional dependencies
- Dependencies and logical implications
- Closure of a set of functional dependencies
- Covers
- Keys and functional dependencies
- Decompositions
- Normalisation
- Denormalisation

INTRODUCTION

- Normalization is based on the analysis of functional dependencies.
- A functional dependency is a constraint between two attributes or two sets of attributes.
- The purpose of the database design is to arrange the various data items into an organized structure so that it generates set of relationships and stores the information without any repetition.

INTRODUCTION

- Normalization is a process for deciding which attributes should be grouped together in a relation.
- It is a tool to validate and improve a logical design, so that it satisfies certain constraints that avoid redundancy of data.

INFORMAL DESIGN GUIDELINES FOR RELATION SCHEMAS

- There are four informal measures of quality for relation schema design.
- These measures are not always independent of one another.

(*I*) meaning (semantics) of the relation attributes.

(*Ii*) reducing the redundant (repetitive) values in tuples.

(*Iii*) reducing the null values in tuples.

(*Iv*) not allowing the possibility of generating spuriousزائف-باطل tuples.

MEANING OF THE RELATION ATTRIBUTES

- When the attributes are grouped to form a relation schema, it is assumed that attributes belonging to one relation have certain real-word meaning and a proper interpretation associated with them.
- This meaning (semantics) specifies how the attribute values in a tuple relate to one another.
- The conceptual design should have a clear meaning, if it is done carefully, followed by a systematic mapping into relations and most of the semantics will have been accounted for.

GUIDELINE 1

- Design a relation schema so that it is easy to explain its meaning.
- The attributes from multiple entity types and relationship types should not be combined into a single relation.
- Thus, a relation schema that corresponds to one entity type or one relationship type has a straightforward meaning.

REDUNDANT INFORMATION IN TUPLES AND UPDATE ANOMALIES

- One major goal of schema design is to minimize the storage space needed by the base relations.
- A significant effect on storage space occurred, when we group attributes into relation schemas.
- The second major problem, when we use relations as base relations is the problem of update anomalies.

GUIDELINE 2

- Design the base relation schema in such a way that no updating anomalies (insertion, deletion and modification) are present in the relations.
- If present, note them and make sure that the programs that update the database will operate correctly.

NULL VALUES IN TUPLES

- When many attributes are grouped together into a “fat” relation and many of the attributes do not apply to all tuples in the relation, then there exists many NULL’S in those tuples.
- This wastes a lot of space. It is also not
- Possible to understand the meaning of the attributes having NULL values.

NULL VALUES IN TUPLES

- How to account for them when aggregate operation (*i.E.,* COUNT or SUM) are applied. The null's can have multiple interpretations, like:

(*A*) the attribute does not apply to this rule.

(*B*) the attribute is unknown for this tuple.

(*C*) the value is known but not present *i.E.,* Cannot be recorded.

GUIDELINE 3

- Try to avoid, placing the attributes in a base relation whose value may usually be NULL.
- If null's are unavoidable, make sure that apply in exceptional cases only and majority of the tuples must have some not null value.

GENERATION OF SPURIOUS TUPLES

- The decomposition of a relation schema R into two relations R1 and R2 is undesirable, because if we join them back using NATURAL join, we do not get the correct original information.
- The join operation generates spurious tuples that represent the invalid information.

GUIDELINE 4

- The relation schemas are designed in such a way that they can be joined with equality conditions on attributes that are either **primary key** or **foreign key**.
- This guarantees that no spurious tuples will be generated.
- Matching attributes in relations that are not (foreign key, primary key) combinations must be avoided, because joining on such attributes may produce spurious tuples

FUNCTIONAL DEPENDENCIES

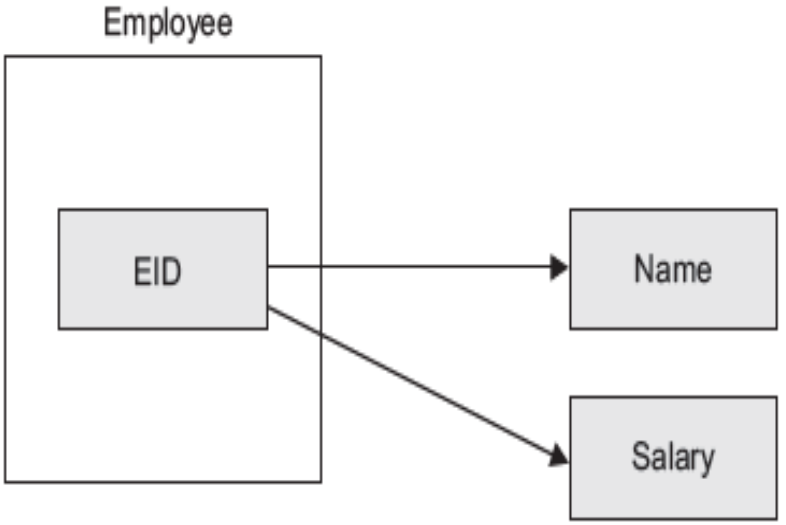
- Functional dependencies are the result of interrelationship between attributes or in between tuples in any relation.
- **Definition :** in relation R, X and Y are the two subsets of the set of attributes, Y is said to be functionally dependent on X if a given value of X (all attributes in X) uniquely determines the value of Y (all attributes in Y).

  It is denoted by X → Y(Y depends upon X).
- **Determinant :** X is known as determinant of functional dependency.

Employee

EID	Name	Salary
1	Aditya	15,000
2	Manoj	16,000
3	Sandeep	9,000
4	Vikas	10,000
5	Manoj	9,000

FIGURE 6.1. Employee relation.



- EID is primary key.
- Suppose you want to know the name and salary of any employee. If you have EID of that employee, then you can easily find information of that employee. So, name and salary attributes depend upon EID attribute.
- Here, x is (EID) and y is (name, salary)
- *X (EID) : Y (name, salary)*
- The determinant is EID

FUNCTIONAL DEPENDENCY CHART/DIAGRAM

- It is the graphical representation of function dependencies among attributes in any relation.
- The four steps are followed to draw FD chart.

1. Find out the primary key attributes.

2. Make a rectangle and write all primary key attributes inside it.

3. Write all non-prime key attributes outside the rectangle.

4. Use arrows to show functional dependency among attributes

EXAMPLE

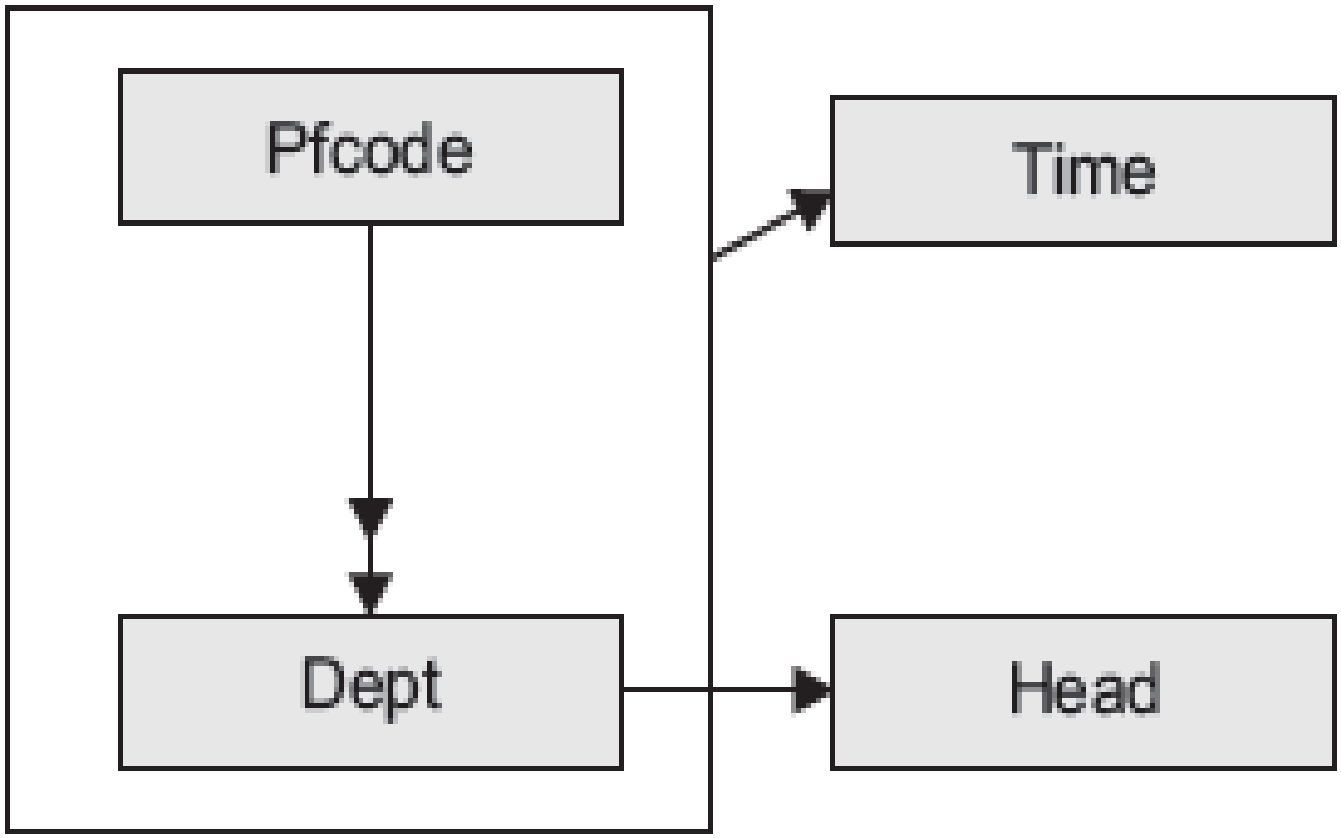
- Consider the following relation:
- Professor (pfcode, dept, head, time) it is assumed that

(*I*) A professor can work in more than one dept.

(*Ii*) the time he spends in each dept is given.

(*Iii*) each dept has only one head.

- Draw the dependency diagram for the given relation by identifying the dependencies.


Pfcode
Time
Dept
Head

TYPES OF FUNCTIONAL DEPENDENCIES

1. **Partial dependency and fully functional dependency**
2. **Transitive dependency and non-transitive dependency**
3. **Single valued dependency and multivalued dependency**
4. **Trival dependency and non-trival dependency**

PARTIAL DEPENDENCY AND FULLY FUNCTIONAL DEPENDENCY

- *Partial dependency :* suppose you have more than one attributes in primary key. Let A be the non-prime key attribute. If A is not dependent upon all prime key attributes, then partial dependency exists.
- *Fully functional dependency :* let A be the non-prime key attribute and value of a is dependent upon all prime key attributes. Then A is said to be fully functional dependent. Consider a relation student having prime key attributes (rollno and game) and non-prime key attributes (grade, name and fee).

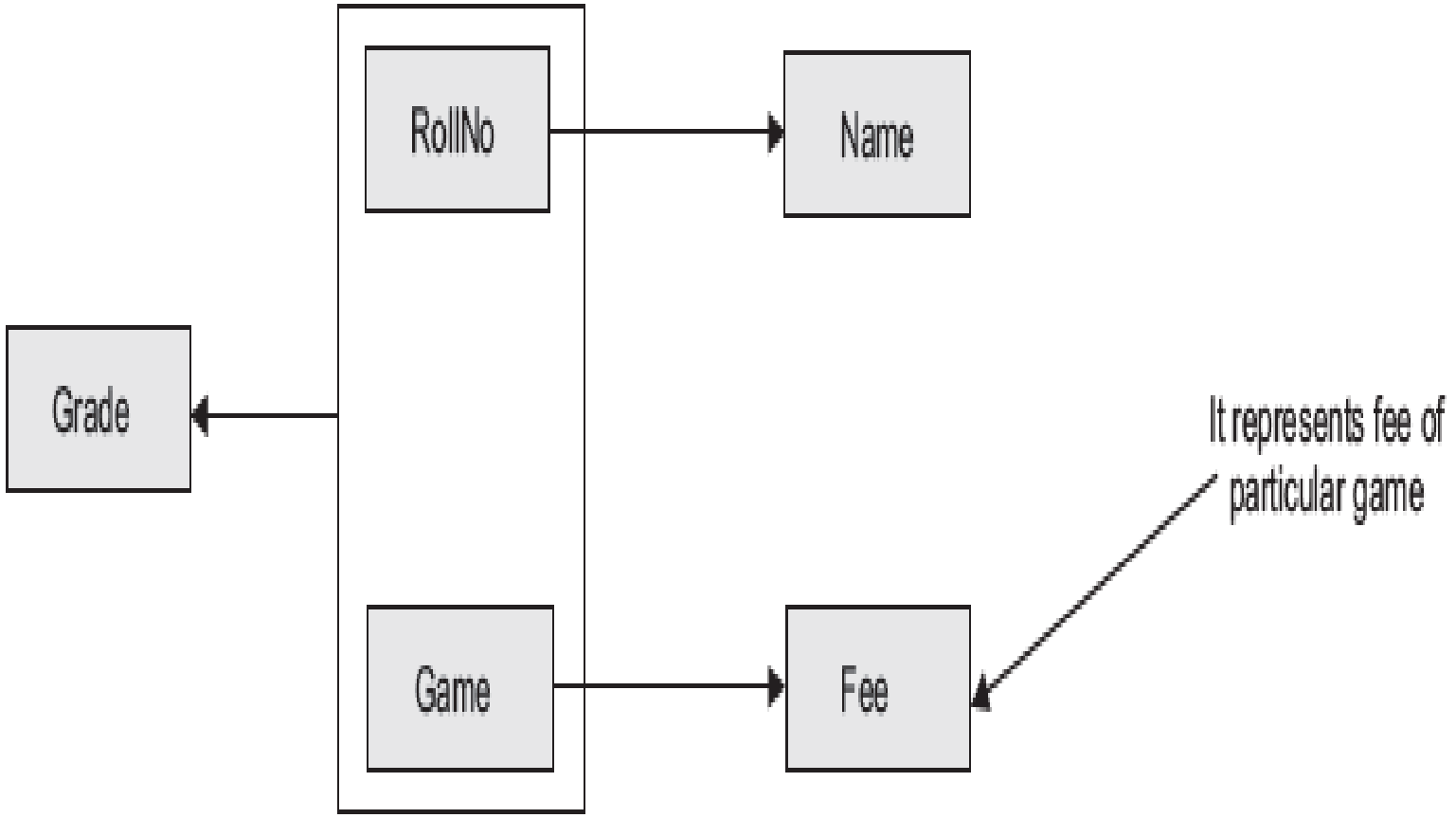


FIGURE 6.4. Functional dependency chart showing partial and fully functional dependency of student relation.

TRANSITIVE DEPENDENCY AND NON-TRANSITIVE DEPENDENCY

- *Transitive dependency :* transitive dependency is due to dependency between non-prime key attributes. Suppose in a relation R, X → Y (Y depend upon X), Y → Z (Z depends upon Y), then X → Z (Z depends upon X). Therefore, Z is said to be transitively dependent upon X.
- *Non-transitive dependency :* any functional dependency which is not transitive is known as non-transitive dependency.
- Non-transitive dependency exists if there is no dependency between non-prime key attributes.

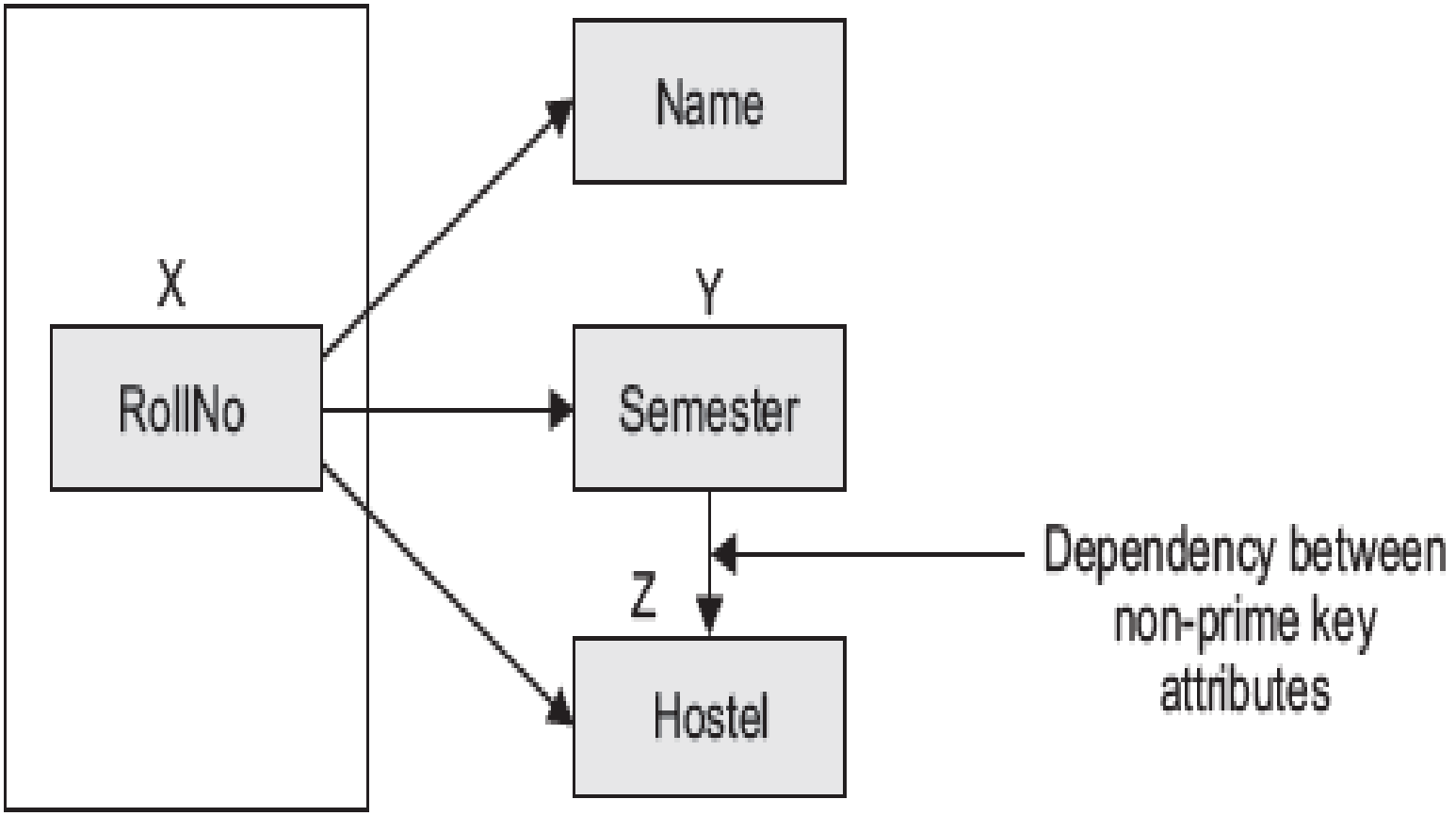


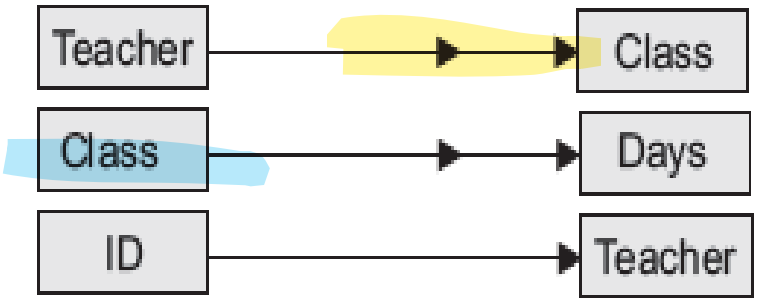
FIGURE 6.5. Functional dependency chart showing transitive and non-transitive dependency on relation student.

SINGLE VALUED DEPENDENCY AND MULTIVALUED DEPENDENCY

- *Single valued dependency :* in any relation R, if for a particular value of X, Y has single value then it is known as single valued dependency.
- *Multivalued dependency (mvd) :* in any relation r, if for a particular value of x, y has more then one value, then it is known as multivalued dependency. It is denoted by X $\rightarrow\!\!\rightarrow$ Y.

ID	Teacher	Class	Days
1Z	Sam	Computer	1
2Z	John	Computer	6
1Z	Sam	Electronics	3
2Z	John	Mechanical	5
3Z	Nick	Mechanical	2

(a)



(b)

FIGURE 6.6. Functional dependency chart showing single valued and multivalued dependency on relation teacher.

TRIVAL DEPENDENCY AND NON-TRIVAL DEPENDENCY

- *Trival FD :* in any relation R, X → Y is trival if Y ⊆ X (Y is the subset of X).
- *Non-trival FD :* in any relation R, X →Y is non-trival if Y(not) ⊆ X (Y is not the subset of X)

S#	City	P#	Quantity
1	Delhi	1P	100
2	Rohtak	8P	200
3	Pune	3P	50
4	Pune	5P	75
5	Rohtak	1P	99
6	Mau	5P	105

EXAMPLE

- Write all non-trivial functional dependencies satisfied by R. Also give an example of an FD that does not hold on R.

O	P	Q
1	2	3
2	1	7
2	1	5
7	3	8

SOL

- For functional dependencies look at the data given in the table.
- – The FD O → P holds on R, since for each value of o there is a single value of P.
- – The FD O → Q does not hold on R, since in the 2nd and 3rd row, O has the same Value, but Q has different values.
- The non-trivial functional dependencies are as follows:
- O → P, Q → O, Q → P, OQ → P, PQ → O, Q → OP, O → OP, OQ → PQ,
- OQ → OPQ, Q → OQ, PQ → OPQ, Q → OPQ, P → O, P → OP

Functional dependence

- Means that the value of one or more attributes determines the value of one or more other attributes.
- Within a relation R, an attribute B is functionally dependent on an attribute A if and only if a given value of attribute A determines exactly one value of attribute B.
- The relationship "B is dependent on A" is equivalent to "A determines B" and is written as A $\longrightarrow$ B.
- In this functional dependency, the attribute whose value determines another is called the determinant or the key.
- The attribute whose value is determined by the other attribute is called the dependent.
- Full Functional Dependence, is used to refer to functional dependencies in which the entire collection of attributes in the determinant is necessary for the relationship.

ANOMALIES IN RELATIONAL DATABASE

- Anomalies refer to the undesirable results because of modification of data.
- **Insertion Anomaly**
- **Deletion Anomaly**
- **Updation Anomaly**

INSERTION ANOMALY

- Suppose you want to add new information in any relation but cannot enter that data because of some constraints.
- When you depend on any other information to add new information then it leads to insertion anomaly.

DELETION ANOMALY

- The deletion anomaly occurs when you try to delete any existing information from any relation, and this causes deletion of any other undesirable information.

UPDATION ANOMALY

- The updation anomaly occurs when you try to update any existing information in any relation and this causes inconsistency of data.

DEPENDENCIES AND LOGICAL IMPLICATIONS

- Given a relational schema R and a set of functional dependencies F.
- A functional dependency X→Y (not in F) on R is said to be logically implied by the set of functional dependencies F on R if for relation R on the relational schema that satisfies F also satisfies X→Y.
- **Example.** Consider the relation schema R= (A, B, C, D) and the set of FD's F = {A→B, B→C, C→D}.
- Then the FD'S A→C, B→D and A→D are logically implied.

ARMSTRONG'S AXIOMS

The following three rules called I**nference axioms** or **Armstrong's axioms** can be used to find all the FDs logically implied by a set of FDs.

Let X, Y, Z, and W be subsets of attributes of a relation R.

The following axioms hold:

1. Reflexivity. If Y is a subset of X, then X → Y. This also implies that X → X always holds. Functional dependencies of this type are called **trivial functional dependencies**.

2. Augmentation. If X → Y holds and Z is a set of attributes, then ZX → ZY.

3. Transitivity. If X → Y holds and Y → Z holds, then X → Z holds.

- These rules are **sound and complete**.
- They are sound because they do not generate any invalid functional dependencies.
- They are complete because they allow us to generate F+ (closure of F) from the given set of functional dependencies F.
- It is very cumbersome and complex to use the **armstrong's axioms** directly for the computation of F+.
- So some more axioms are added to simplify the process of computation of F+.
- These additional axioms can be proved correct by using the **armstrong's axioms.**

4. Additivity or **union**. If X → Y and X → Z, then X → YZ holds.

5. Projectivity or decomposition. If X → YZ holds, then X → Y and X → Z also holds

6. Pseudotransitivity. If X → Y and ZY → W holds, then XZ → W holds.

These additional axioms are also **sound and complete**.

CLOSURE OF A SET OF FUNCTIONAL DEPENDENCIES

- Assume that F is a set of functional dependencies for a relation R.
- The **closure of F, denoted by** F+ , is the set of all functional dependencies obtained logically implied by f *i.E.,* F+ is the set of **FD**'s that can be derived from F.
- Furthermore, the F+ is the smallest set of FD's such that F+ is superset of F and no FD can be derived from F by using the axioms that are not contained in F+.

COVERS

- Consider two sets of FD's F1 and F2 over a relation scheme R.
- The two sets F1 and F2 are equivalent if the closure of F1 is equal to the closure of F2 *i.E.* F1 + = F2+.
- The set F1 covers F2 and F2 covers F1 iff F1 and F2 are equivalent.
- **Importance of cover:** sometimes closure of a set of FD's **F+ can be very large and difficult to compute.**
- **In that case the cover is used.**
- **It acts as a representative of the closure of F.**

TYPES OF COVER

1. **Redundant cover:** consider a set of FD's F1 over a relation schema R. Now if a proper subset F1′ of F1 covers F1, then we can say F1 is redundant cover.
2. **Non-redundant cover:** consider two sets of FD's F1 and F2 over a relation schema R. Suppose F1 covers F2 and no proper subset F1′ of F1 covers F2. This set F1 is called the non-redundant cover.

- .

TYPES OF COVER

- To obtain a non-redundant cover, we have to remove some FD's say X→Y from F1. The non-redundant cover so obtained is not unique and it is possible to obtain more than one non-redundant cover. A non-redundant cover with minimum number of FD's is known as **minimal cover**.

3. Canonical cover: before discussing the canonical cover Fc for the set of FD's F, let Us discuss some of the terms that are associated with canonical cover.

- **Simple FD:** A functional dependency X→A1A2A3……..An can be replaced by an equivalent set of FD's X→A1, X→A2, X→A3…. X→An by using the axioms additivity and projectivity.
- An FD of the form X→Ai, where the right hand side has only one attribute is called a simple FD.
- It is possible to replace every set of FD's by an equivalent set of simple **FD**'s.

- **Extraneous attributes:** an attribute A of a FD A→B in F is said to be extraneous if it can be removed from F(set of FD’s) without changing its closure F+.

OUTLINES

- Introduction
- Informal design guidelines for relation schemas
- Functional dependencies
- **Anomalies in relational database**
- **Dependencies and logical implications**
- Closure of a set of functional dependencies
- Keys and functional dependencies
- Decompositions
- Normalisation
- **De**normalisation

IDENTIFYING REDUNDANT FUNCTIONAL DEPENDENCIES

- Given a set of functional dependencies F.
- An FD in F is redundant if it can be derived from the other FD's in the set F
- *i.E.* FD X→A is redundant if {F – (X→A)} logically implies F.

STEPS TO REMOVE REDUNDANT FD'S.

- **Algorithm:** to determine the redundant FD in a set of FD's
- **Input:** given a relation with a set of FD's F.
- **Output:** non-redundant set of FDs equivalent to the original set
- **Step 1.** For every FD X→A in F do
- **Step 2.** Remove X→A from F, and call the resultant FD's, (*i.E.* F′= {– (x→a)})
- **Step 3.** Compute X+ under F′.
- **Step 4.** If A ε X+, then X→A is redundant. Rename F as F′.

- **Example.** Remove any redundant FD's from the following set of FD's F:
- (1) D → C
- (2) D → B
- (3) BC → A
- (4) DC → B
- **Sol.**
- Try for FD (1), D → C. Remove it to obtain F′. Now F′= {D → B, BC → A, Dc → b}. Further, compute the closure of D *i.E.* D+ under F′. The closure of D is DB. Since C does not belong to D+, hence D → C is not redundant FD.

KEYS AND FUNCTIONAL DEPENDENCIES

- A key or candidate key is a set of attributes that uniquely identify a record in a relation
- **Prime and non-prime attributes:** for a given relation R= {A1,A2,A3,……..,An}, an attribute a is a prime attribute if a is a part of any candidate key of r otherwise a is a non-prime attribute.

- **Example.** Consider a relation R(A, B, C, D, E, F, G, H) having a set of fd's F = {D→AB, B→a, c→a, f→g, h→fgd, e→a}. What are the candidate keys of relation R? Also find Prime and non-prime attributes.
- **Sol.** Consider a subset K={C, E, H} of R. CEH is a candidate key as all attributes does Not appear on right hand side of any FD in F. It can also be proved as follows.
- Compute k+ which gives abcdefh. Now compute closure of every subset of K such As CE, EH and CH.
- First consider y= {c,e}, the closure y+ = ac ≠ r.
- Second consider y={e,h}, the closure y+ = abdefgh ≠ r.
- Finally, consider y= { c,h}, the closure y+ = abcdfgh ≠ r.
- Thus, ceh is a candidate key.
- Prime attributes are = ceh, Non-prime attributes are = ABDFG

DECOMPOSITIONS

- Let U be a relation schema. A set of relation schemas {R1 , R2 , ... , Rn} is a decomposition of U if and only if

$$U = R1 \cup R2 \cup \ldots \cup Rn$$

Lossless-join decomposition

- A decomposition {R, T} of U is a lossless-join decomposition (with respect to a set of Constraints) if the constraints imply that $u = r \infty t$ for all possible instances of R, T, and U.
- The decomposition is said to be lossy otherwise.
- It is always the case for any decomposition {r, t} of u that

$u \subseteq r \infty t$.

PROPERTIES OF A GOOD DECOMPOSITION

- A relational schema R with a set of functional dependencies F is decomposed into R1 and R2.

1. **Lossless-join decomposition**

Test to see if at least one of the following dependencies are in F+

$R1 \cap R2 \rightarrow R1$

$R1 \cap R2 \rightarrow R2$

If not, decomposition may be lossy

PROPERTIES OF A GOOD DECOMPOSITION

2. **Dependency preservation**

Let Fi be the set of dependencies in F+ that include only attributes in Ri

Test to see if (F1 ∪ F2)+= F+

When a relation is modified, no other relations need to be checked to preserve Dependencies.

3. **No redundancy**

EXAMPLE

- R1 = (A, B, C)
- F = {A → B, B → C}
- R1 = (A, B), R2 = (B, C)
- Lossless-join decomposition?
- R1 ∩R2 = {B} AND B → BC ∈ F+
- Dependency preserving?
- F+ = {A → B, A → C, A → AB, A → BC, A → AC, A → ABC, AB → C, AB → AC, AB → BC, AB → ABC, AC → BC, AC → {B, B → C, B → BC} ∪ {many trivial dependencies}
- F1 = {A → B, A → AB} ∪ (many trivial dependencies}
- F2 = {B → C, B → BC}∪(many trivial dependencies}
- (F1 ∪ f2)+ = f+

NORMALISATION

- Normalisation is a process by which we can decompose or divide any relation into more than one relation to remove anomalies in relational database.
- It is a step by step process and each step is known as **normal form**.
- Normalisation is a reversible process.

BENEFITS OF NORMALISATION

1. Normalisation produces smaller tables with smaller rows, this means more rows per page and hence less logical I/O.
2. Searching, sorting, and creating indexes are faster, since tables are narrower, and more rows fit on a data page.
3. The normalisation produces more tables by splitting the original tables. Thus there can be more clustered indexes and hence there is more flexibility in tuning the queries.
4. Index searching is generally faster as indexes tend to be narrower and shorter.

5. The more tables allow better use of segments to control physical placement of data.
6. There are fewer indexes per table and hence data modification commands are faster.
7. There are small number of null values and less redundant data. This makes the database more compact.
8. Data modification anomalies are reduced.
9. Normalization is conceptually cleaner and easier to maintain and change as the needs change.

FIRST NORMAL FORM (1NF)

- A relation is in first normal form if domain of each attribute contains only atomic values.
- It means atomicity must be present in relation.

Employee

EID	Name	Salary	Dept.No	Dept.Name
1	Shivi Goyal	10,000	2	Accounts
2	Amit Chopra	9,000	2	Accounts
3	Deepak Gupta	11,000	1	Sales
4	Sandeep Sharma	8,500	5	Marketing
5	Vikas Malik	7,000	5	Marketing
6	Gaurav Jain	15,000	2	Accounts
7	Lalit Parmar	14,000	5	Marketing
			10	Finance
8	Vishal Bamel	10,500	2	Accounts

Cannot be Inserted

FIGURE 6.8. Employee relation with anomalous data.

Employee

EID	First Name	Second Name	Salary	Dept. No.	Dept. Name
1	Shivi	Goyal	10,000	2	Accounts
2	Amit	Chopra	9,000	2	Accounts
3	Deepak	Gupta	11,000	1	Sales
4	Sandeep	Sharma	8,500	5	Marketing
5	Vikas	Malik	7,000	5	Marketing
6	Gaurav	Jain	15,000	2	Accounts
7	Lalit	Parmar	14,000	5	Marketing
8	Vishal	Bamel	10,500	2	Accounts

FIGURE 6.9. Employee relation in 1NF.

- *Anomalies in first normal form :* first normal form deals only with atomicity. Anomalies described earlier are also applicable here
- **Example.** Given the relation PART (PART_ID, DESCR, PRICE, COMP_ID, NO) having the Following dependencies
- PART_ID → DESCR
- PART_ID → PRICE
- PART_ID, COMP_ID → NO
- Determine the highest normal form of this relation.

- **Sol.** There exists multi-value attributes. The attributes COMP_ID and NO are not determined By the primary key. Hence the relation is **not** in INF.

SECOND NORMAL FORM (2NF)

- A relation is in second normal form if it is in 1NF and all non-primary key attributes must be fully functionally dependent upon primary key attributes.

- The primary key is (rollno., Game). Each student can participate in more than one game.
- Relation student is in 1nf but still contains anomalies.

1. ***Deletion anomaly*** **:** suppose you want to delete student jack. Here you loose information about game hockey because he is the only player participated in hockey.

2. ***Insertion anomaly*** **:** suppose you want to add a new game basket ball having no student participated in it. You cannot add this information unless there is a player for it.

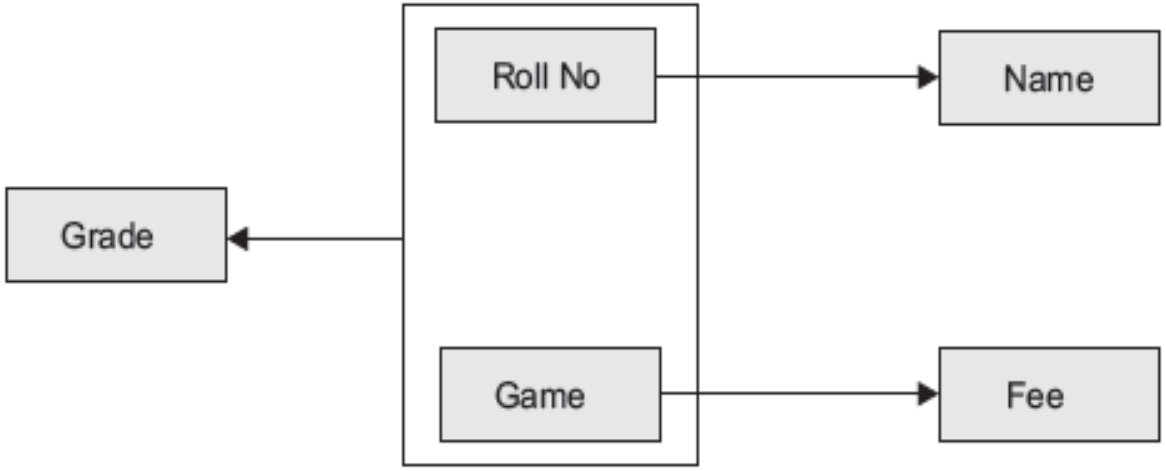
3. ***Updation anomaly*** **:** suppose you want to change fee of cricket. Here, you have to search all the students participated in cricket and update fee individually otherwise it produces inconsistency.

- The solution of this problem is to separate partial dependencies and fully functional Dependencies.
- So, divide student relation into three relations student(rollno., Name), games (Game, fee) and performance(rollno., Game, grade)

Student

RollNo.	Game	Name	Fee	Grade
1	Cricket	Amit	200	A
2	Badminton	Dheeraj	150	B
3	Cricket	Lalit	200	A
4	Badminton	Parul	150	C
5	Hockey	Jack	100	A
6	Cricket	John	200	C

(a)



Dependency Chart

(b)

FIGURE 6.10. Student relation with anomalies.

Student

RollNo.	Name
1	Amit
2	Dheeraj
3	Lalit
4	Parul
5	Jack
6	John

Games

Game	Fee
Cricket	200
Badminton	150
Hockey	100

Performance

RollNo.	Game	Grade
1	Cricket	A
2	Badminton	B
3	Cricket	A
4	Badminton	C
5	Hockey	A
6	Cricket	C

FIGURE 6.11. Relations in 2NF.

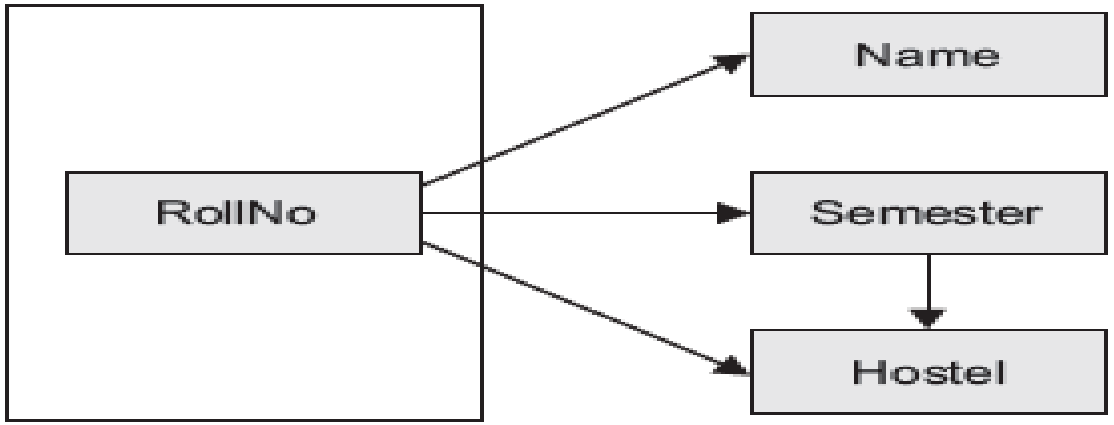
THIRD NORMAL FORM (3NF)

- A relation is in third normal form if it is in 2NF and non-primary key attributes must be non-transitively dependent upon primary key attributes.
- In other words a relation is in 3nf if it is in 2nf and having no transitive dependency.

Student

RollNo.	Name	Semester	Hostel
1	Lalit	1	H1
2	Gaurav	2	H2
3	Vishal	1	H1
4	Neha	4	H4
5	John	3	H3

(a)



(b)

FIGURE 6.12. Student relation.

- The primary key is (ROLLNO.). The condition is different hostel is allotted for different semester. Student relation is in 2NF but still contains anomalies.

1. ***Deletion anomaly*** **:** if you want to delete student gaurav. You loose information about hostel H2 because he is the only student staying in hostel H2.

2. ***Insertion anomaly*** **:** if you want to add a new hostel H8 and this is not allotted to any student. You cannot add this information.

3. ***Updation anomaly*** **:** if you want to change hostel of all students of first semester. You have to search all the students of first semester and update them individually otherwise it causes inconsistency.

- The solution of this problem is to divide relation student into two relations student(ROLLNO. NAME, SEMESTER) and hostels(SEMESTER, HOSTEL)

Student

RollNo.	Name	Semester
1	Lalit	1
2	Gaurav	2
3	Vishal	1
4	Neha	4
5	John	3

Hostels

Semester	Hostel
1	H1
2	H2
3	H3
4	H4

FIGURE 6.13. Relations in 3NF.

DENORMALIZATION

- Denormalization is the process of attempting to optimize the read performance of a database by adding redundant data or by grouping data. In some cases, denormalization helps cover up the inefficiencies inherent in relational database.

- The more usual approach is to denormalize the logical data design.
- You can achieve the same improvement in query response but now the database designer has the responsibility to ensure that the denormalized database does not become inconsistent.
- This is done by adding constraints in the database.
- These constraints specify how the redundant copies of information must be kept synchronized.

Denormalization can be described as a process for reducing the degree of normalization with the aim of improving query processing performance.

Or

Denormalization is the process of putting one fact in numerous places.

- One of the main purposes of denormalization is to reduce the number of physical tables that must be accessed to retrieve the desired data by reducing the number of joins needed to derive a query answer.
- This speeds data retrieval at the expense of data modification.
- Denormalization has been utilized in many strategic database implementations to boost database performance and reduce query response times.

- One of the most useful areas for applying denormalization techniques is in data warehousing implementations for data mining transactions.
- The primary goals of denormalization are to improve query performance and to present the end-user with a less complex and more user-oriented view of data.
- This is in part accomplished by reducing the number of physical tables and reducing the number of actual joins necessary to derive the answer to a query.

THE NEED OF DENORMALIZATION

1. Many critical queries and reports exist that need to be processed in an on-line environment and rely upon data from more than one table.
2. Many repeating groups exist which need to be processed in a group instead of individually.
3. Many calculations need to be applied to one or many columns before queries can be successfully answered.
4. Tables need to be accessed in different ways by different users during the same timeframe.
5. Certain columns are queried a large percentage of the time, which makes them a candidate for denormalization.

THE BASIC ISSUES THAT MUST BE EXAMINED WHEN CONSIDERING DENORMALIZATION

- What are the critical transactions, and what is the expected response time?
- How often are the transactions executed?
- What tables or columns do the critical transactions use? How many rows do they access each time?
- What is the mix of transaction types: select, insert, update, and delete?
- What is the usual sort order?

THE BASIC ISSUES THAT MUST BE EXAMINED WHEN CONSIDERING DENORMALIZATION

- What are the concurrency expectations?
- How big are the most frequently accessed tables?
- Do any processes compute summaries?
- Where the data physically located?

ADVANTAGES OF DENORMALIZATION

1. Denormalization can improve performance by minimizing the need for joins.
2. Denormalization can improve performance by reducing the number of foreign keys on tables.
3. Denormalization can improve performance by reducing the number of indexes, saving storage space and reducing data modification time.

ADVANTAGES OF DENORMALIZATION

4. Denormalization can improve performance by precomputing aggregate values, that is, computing them at data modification time rather than at select time
5. In some cases denormalization can improve performance by reducing the number of Tables.

DISADVANTAGES OF DENORMALIZATION

1. It usually speeds retrieval but can slow data modification.
2. It is always application-specific and needs to be re-evaluated if the application changes.
3. It can increase the size of tables.
4. In some instances, it simplifies coding but in some others, it makes coding more complex.