Database Engineering

Lecture #8

Functional Dependencies and Normalization for Relational Databases

Presented By:
Dr. Suvasini Panigrahi
Associate Professor, Department of CSE,
VSSUT, Burla

Lecture Outline

1. Informal Design Guidelines for Relational Databases
   - 1.1 Semantics of the Relation Attributes
   - 1.2 Redundant Information in Tuples and Update Anomalies
   - 1.3 Null Values in Tuples
   - 1.4 Spurious Tuples
2. Functional Dependencies (FDs)
   - 2.1 Definition of FD
   - 2.2 Inference Rules for FDs
   - 2.3 Equivalence of Sets of FDs
   - 2.4 Minimal Sets of FDs

Pitfalls in Relational Database Design

- Relational database design requires that we find a “good” collection of relation schemas that satisfy the design goals.
- A bad design may lead to:
  - Repetition of information.
  - Inability to represent certain information.
- Design Goals
  - Avoid redundant data
  - Ensure that relationships among attributes are represented
  - Facilitate the checking of updates to prevent the violation of database integrity constraints

Relational Database Design

- What is relational database design?
  - The grouping of attributes to form a "good" relation schema
- There are two levels of relation schema:
  - The logical "user view" level
  - The storage "base relation" level
- Design is concerned mainly with base relations
- What are the criteria for "good" base relations?

Informal Design Guidelines for Relational Databases

- The four informal guidelines that can be used as measures to determine the quality of relation schema design are:
  1. Imparting Clear Semantics to Attributes in Relations
  2. Reducing the Redundant Information in Tuples and Update Anomalies
  3. Reducing the NULL Values in Tuples
  4. Disallowing the generation of spurious tuples

Informal Design Guidelines for Relational Databases

1. Imparting Clear Semantics to Attributes in Relations:

- Design a schema that can be explained easily relation by relation.
- The semantics of attributes should be easy to interpret.
- Do not combine attributes from multiple entity types and relationship types into a single relation.
- Only foreign keys should be used to refer to other entities
- Entity attributes and relationship attributes should be separate

- **GUIDELINE 1:** Informally, each tuple in a relation should represent one entity or relationship instance.

Informal Design Guidelines for Relational Databases

2. Reducing the Redundant Information in Tuples and Update Anomalies:

- One goal of schema design is to minimize the storage space used by the relations
- Mixing attributes of multiple entities through join operation may cause additional problems
- Information is stored redundantly leading to wasting of storage space as well as data inconsistency (Update Anomalies)
- There are three types of update anomalies:
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

Example of an Update Anomaly

Consider the relation:

EMP_PROJ (Emp#, Proj#, Ename, Pname, No_hours)

- **Insertion Anomaly:**
  - Cannot insert a project unless an employee is assigned to it.
  - Inversely we cannot insert an employee unless he/she is assigned to a project.
- **Deletion Anomaly:**
  - When a project is deleted, it will result in deleting all the employees who work on that project.
  - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

Example of an Update Anomaly

- **Modification Anomaly:**
  - Changing the name of project number P1 from "Billing" to "Customer-Accounting" requires this update to be made for all 100 employees working on project P1.
  - If the modification of some of the attribute information is not done for all rows, then it leads to data inconsistency.
- **GUIDELINE 2:** Design a schema that does not suffer from the insertion, deletion and update anomalies. If there are any anomalies present, then note them so that applications can be made to take them into account.

Informal Design Guidelines for Relational Databases

3. Reducing the NULL Values in Tuples:

- In some schema designs we may group many attributes together into a “big” relation.
- If many of the attributes are not applicable to all the tuples in a relation or attribute values are unknown, then there will be many NULL values in those tuples.
- This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes.

- **GUIDELINE 3:** Relations should be designed such that their tuples will have as few NULL values as possible.

Informal Design Guidelines for Relational Databases

4. Disallowing the generation of spurious tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations.
- Spurious Tuples are those rows in a table, which occur as a result of joining two tables in wrong manner. They are extra tuples (rows) which might not be required.
- The "lossless join" property is used to guarantee meaningful results for join operations.

- **GUIDELINE 4:** The relations should be designed to satisfy the lossless join condition. No spurious tuples should be generated by doing a natural join of any relations.

Decomposition In DBMS

- The process of decomposition refers to dividing a relation X into {X1, X2,……Xn}.
- The process of decomposition in DBMS helps in removing redundancy, inconsistencies and anomalies from a database when we divide the table into numerous small tables.
- Decomposition is of two major types in DBMS:
  1. Lossless Decomposition
  2. Lossy Decomposition

Lossless Decomposition

- Lossless join decomposition is a decomposition of a relation R into relations R1, and R2 such that if we perform a natural join of relation R1 and R2, it will return the original relation R.
- This is effective in removing redundancy from databases while preserving the original data.
- **Example of Lossless Decomposition**

  Employee(Emp_Id, Ename, Salary, Dept_Id, Dname)
- It can be decomposed using lossless decomposition as:

  Employee_desc(Emp_Id, Ename, Salary, Dept_Id)
  Department_desc(Dept_Id, Dname)

Lossy Decomposition

- The original relation and relation reconstructed from joining decomposed relations must contain the same number of tuples if the number is increased or decreased then it is Lossy Join decomposition.
- The lossy decomposition would be as joining these tables is not possible so not possible to get back original data.

Employee_desc (Emp_Id, Ename, Salary)
Department_desc (Dept_Id, Dname)

Properties of Decomposition

There are two important properties of decomposition:

(a) Lossless Join/Non-additive Join Property: A lossless Join decomposition ensures two things:

- No information is lost while decomposing the original relation.
- If we join back the decomposed relations, the same relation that was decomposed is obtained as the result of join.

(b) Dependency-preserving Decomposition:

- Dependency Preservation ensures that after decomposing a relation R into R1 and R2, all Functional Dependencies (FDs) of the original relation R must be present either in R1 or R2.

OR

- All the FDs of the original relation can be derived using the combination of FDs present in R1 and R2.

Functional Dependencies

- Functional dependencies (FDs) is a relationship that exists between two sets of attributes.
- A FD is defined as a constraint between two sets of attributes from the relation schema.
- It mostly exists between the primary key and non-key attribute within a table.
- A Functional Dependency between two sets of attributes X and Y is denoted by:

  X → Y

- The attribute set on the left side of the arrow, **X** is called **Determinant**, and **Y** is called the **Dependent**.
- This holds if the value of Y is determined by the value of X (Y is functionally dependent on X).
- Alternatively, the value of X functionally determine the values of Y.

Examples of FD constraints

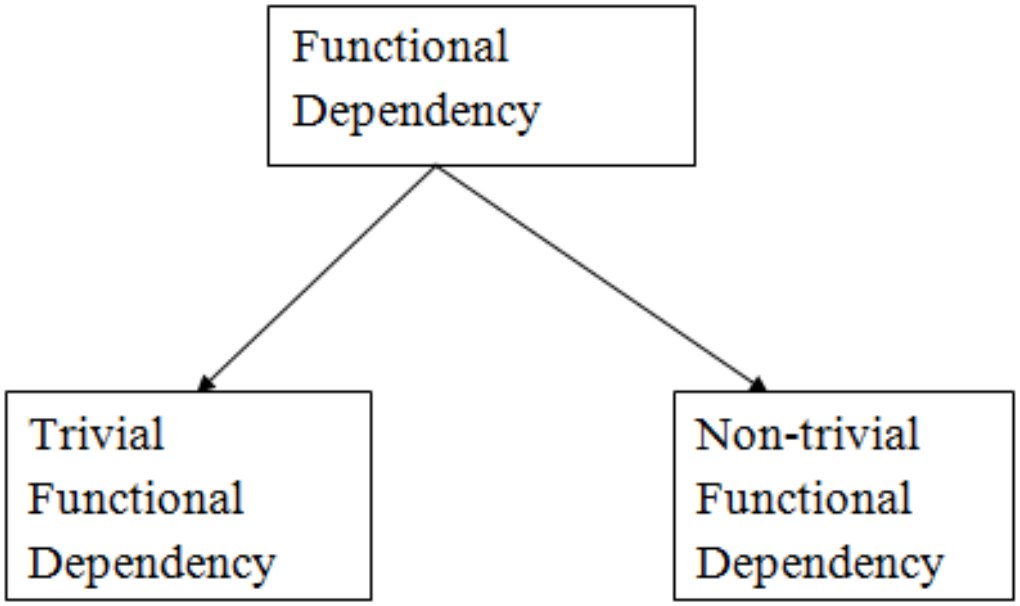
- **Example 1:** Assume that we have an employee table with attributes: *Emp_Id*, *Emp_Name*, *Emp_Address*.
- Here, *Emp_Id* attribute can uniquely identify the *Emp_Name* attribute of employee table because if we know the *Emp_Id*, we can determine the employee name associated with it.
- The Functional dependency can be written as:

  Emp_Id → *Emp_Name*
- **Example 2:** Project Number determines project name and location

  PNUMBER → {PNAME, PLOCATION}
- **Example 3:** Employee SSN and project number determines the hours per week that the employee works on the project

  {SSN, PNUMBER} → HOURS

Types of Functional Dependency



Trivial Functional Dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B.
- **Example:**
  - Consider a table with two columns Emp_Id and Emp_Name
  - {Emp_id, Emp_Name} → Emp_Id is a trivial functional. dependency as Emp_Id is a subset of {Emp_Id, Emp_Name}
  - Also, Emp_Id → Emp_Id and Emp_Name → Emp_Name are trivial dependencies too.

Non-Trivial Functional Dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- **Example:**
  - Consider a table **Employee(Employee_Id, Name, Age)**
  - Here, **{Employee_Id} → {Name}** is a non-trivial FD because Name (*dependent*) is not a subset of Employee_Id (*determinant*).
  - Similarly, **{Employee_Id, Name} → {Age}** is also a non-trivial functional dependency.
- When A ∩ B is NULL, then A → B is called as completely non-trivial. For Example: Roll_No → Name.

Inference Rules for FDs

- Using the inference rules, we can derive additional FDs from the initial FD set.
- Given a set of FDs F, we can *infer* additional FDs that hold whenever the FDs in F hold.
- The notation F ⊨ X → Y is used to denote that the FD X → Y is inferred from the FD set F.
- Armstrong's inference rules
  - **IR1. (Reflexive)** If Y subset-of X, then X → Y
  - **IR2. (Augmentation)** If X → Y, then XZ → YZ (XZ stands for X U Z)
  - **IR3. (Transitive)** If X → Y and Y → Z, then X → Z
- IR1, IR2, IR3 form a *sound* and *complete* set of inference rules

Additional Useful Inference Rules

- **Decomposition (IR4)**
  - If X → YZ, then X → Y and X → Z
- **Union (IR5)**
  - If X → Y and X → Z, then X → YZ
- **Psuedotransitivity (IR6)**
  - If X → Y and WY → Z, then WX → Z
- **Closure** of a set of FDs F is the set F^+ of all FDs that can be inferred from F

Closure of an Attribute Set

- The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.
- Closure of attribute set {X} is denoted as $\{X\}^+$.
- **Steps to Find Closure of an Attribute Set-**
  - **Step-1:** Add elements of attribute set to the result set (Initialization).
  - **Step-2:** Recursively add elements (attributes) to the result set that can be functionally determined from the attributes already in the result set.
  - **Step-3:** Repeat Step 2 until no more attributes can be added to the result set.

Algorithm to Find Closure of an Attribute Set

Algorithm to compute X^+, the closure of attribute set X under F

Result:= X;

while (changes to Result) do

for each FD Y → Z in F do

Begin

if Y ⊆ Result then Result := Result ∪ Z;

End

Example to Find Closure of an Attribute Set

- Given a relation schema R(A, B, C, G, H, I) and the set of FDs F = {A → B, A → C, CG → H, CG → I, B → H}. Compute the closure of (AG) from the given data.
- As per the Attribute Closure Algorithm, the following steps are followed to compute $(AG)^+$:
  - Result = AG [Initially]
  - Result = ABG [A → B, A ⊆ AG]
  - Result = ABCG [A → C, A ⊆ ABG]
  - Result = ABCGH [CG → H, CG ⊆ ABCG]
  - Result = ABCGHI [CG → I, CG ⊆ ABCGH]
  - Result = ABCGHI [B → H, No changes to Result as H ∈ Result]

Closure of an Attribute Set

- For a given relation schema R(A, B, C, D, E, F, G) and a set of FDs F = {A → BC, BC → DE, D → F, CF → G}
- A^+ = {A, B, C, D, E, F, G}
- $(BC)^+$ = {B, C, D, E, F, G}
- D^+ = {D, F}
- $(CF)^+$ = {C, F, G}

Finding the Keys Using Attribute Closure

- Super Key
  - If the closure result of an attribute set contains all the attributes of the relation, then that attribute set is called as a super key of that relation.
- Candidate Key (Minimal Super Key)
  - If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

Finding the Keys Using Attribute Closure

Given a relation schema: **R(E-ID, E-NAME, E-CITY, E-STATE)** with the set of FDs F as follows:

F = {E-ID → E-NAME, E-ID → E-CITY, E-ID → E-STATE, E-CITY → E-STATE}

The attribute closure of various sets of attributes is calculated as:

- **(E-ID)$^+$** = {E-ID, E-NAME,E-CITY,E-STATE}
- **(E-ID,E-NAME)$^+$** = {E-ID, E-NAME,E-CITY,E-STATE}
- **(E-ID,E-CITY)$^+$** = {E-ID, E-NAME,E-CITY,E-STATE}
- **(E-ID,E-STATE)$^+$** = {E-ID, E-NAME,E-CITY,E-STATE}
- **(E-ID,E-CITY,E-STATE)$^+$** = {E-ID, E-NAME,E-CITY,E-STATE}
- **(E-NAME)$^+$** = {E-NAME}
- **(E-CITY)$^+$** = {E-CITY,E-STATE}

Finding the Keys Using Attribute Closure

- **Candidate Key:** **E-ID** as each tuple of EMPLOYEE relation can be uniquely identified by E-ID and it is minimal as well.
- **Super Key:** **(E-ID)**, **(E-ID, E-NAME)**, **(E-ID, E-CITY)**, **(E-ID, E-STATE)**, **(E-ID, E-CITY, E-STATE)** are super keys as the attribute closure of all these attribute sets give set of all attributes of relation EMPLOYEE. So, all of these are super keys of the EMPLOYEE relation.

Closure of Functional Dependencies

- **Closure** of a set *F* of FDs is the set F^+ of all FDs that can be inferred from *F* by applying various Inference Rules.
- Given a set *F* of FDs, there are certain other FDs that are logically implied by *F* or can be inferred from *F*.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$.
- The set of all functional dependencies that can be inferred from *F* is the *closure* of *F* (denoted as $\boldsymbol{F^+}$) and F^+ is a superset of *F*.

Algorithm: Determining X^+, the closure of X under F

Input: Suppose, F be a set of FDs for relation schema R and X is a attribute set of R. The problem is to compute closure of X (X^+)

Steps:

1. $X^+ = X$ //Initialize X^+ to X
2. For each FD : $Y \rightarrow Z$ in F Do
   If $Y \subseteq X^+$ Then //If Y is contained in X^+
   $X^+ = X^+ \cup Z$ //Add Z to X^+
   End If
   End For
3. Return X^+ //Return closure of X

Output: Closure X^+ of X under F.

Example of Computing FD Closure

- Consider the relation schema R = {H, D, X, Y, Z} and the functional dependencies F = {X→YZ, DX→W, Y→H}. Find the closure F^+ of the FDs.
- Solution:
  - Applying Decomposition rule on X→YZ, gives X→Y and X→Z.
  - Applying Transitivity rule on X→Y and Y→H gives X→H.
  - Thus, the closure of the set of FD's F^+ = {X→YZ, DX→W, Y→H, X→Y, X→Z, X→H}.

Key Definitions

- **Prime Attributes:** Prime attributes are the attributes that are part of any of the candidate keys or primary key.
- **Non-Prime Attributes:** Attributes other than prime attributes which does not take part in formation of candidate keys are known as non-prime attributes.
- **Example:** Consider the relation R(A, B, C, D) with functional dependencies F = {A → BC, B → C, D → C}
- Here, the candidate key can be "AD" only as $(AD)^+$ = (A, B, C, D). Hence, we have the following:
  - Prime Attributes: A, D.
  - Non-Prime Attributes: B, C

Key Definitions

- **Extraneous Attributes:** An attribute is considered extraneous if it can be removed from a FD without changing the closure of the FD.
- **Example:** We consider a relation R with schema R(A, B, C) and set of FDs F = {AB → C, A → C}.
  - The closure for F is F^+ = {AB → C, A → C}.
  - In AB → C, B is extraneous attribute.
  - The reason is, there is another FD A → C, which implies that A alone can determine C. Thus, the use of B is considered as extraneous and hence, can be removed from the FD.

Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - Every FD in F can be inferred from G, *and*
  - Every FD in G can be inferred from F
- Hence, F and G are equivalent if $F^+ = G^+$.
- Definition: F **covers** G if every FD in G can be inferred from F.
- F and G are **equivalent** if F covers G and G covers F.
- There is an algorithm for checking equivalence of sets of FDs.

Minimal Sets of FDs

- A set of FDs is **minimal** if it satisfies the following conditions:
  (1) Every dependency in F has a single attribute in its RHS.
  (2) We cannot remove any dependency from F and 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 $Y \subset X$ and still have a set of dependencies that is equivalent to F.
- Every set of FDs has an equivalent minimal set.
- There can be several equivalent minimal sets.

Minimal Cover/Canonical Cover of a set of FDs

- A minimal cover of a set of FDs F is a minimal set of functional dependencies F_{min} that is equivalent to F.
- There can be many such minimal covers for a set of functional dependencies F.

Finding Minimal Cover

- To get the minimal cover of a set of FDs:
  - Right Hand Side (RHS) of all FDs should have a single attribute.
  - Remove extraneous attribute from each FD.
  - Remove redundant FD from the set of FDs.

Example

Consider an example to find canonical cover of F.

The given functional dependencies are as follows −

A -> BC

B -> C

A -> B

AB -> C

First step − Convert RHS attribute into singleton attribute.

A -> B

A -> C

B -> C

A -> B

AB -> C

Second step − Remove the extra LHS attribute

Find the closure of A.

A+ = {A, B, C}

So, AB -> C can be converted into A -> C

A -> B

A -> C

B -> C

A -> B

A -> C

Third step − Remove the redundant FDs.

A -> B

B -> C