Chapter 7.2

Basics of Functional Dependencies and Normalization for Relational Databases

Informal Design Guidelines for Relational Schemas

- The four informal design guidelines that may be used as measures to determine the quality of relation schema design are:
  1. Making sure that the semantics of the attributes is clear in the schema
  2. Reducing the redundant information in tuples
  3. Reducing the NULL values in tuples
  4. Disallowing the possibility of generating spurious tuples

Informal Design Guidelines for Relational Schemas

1. **Imparting Clear Semantics to Attributes in Relations**
   - **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. Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straightforward to explain its meaning. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained

Informal Design Guidelines for Relational Schemas

2. Redundant Information in Tuples and Update Anomalies

- One goal of schema design is to minimize the storage space. Grouping attributes into relation schemas has a significant effect on storage space.
- Storing natural joins of base relations leads to an additional problem referred to as update anomalies (insertion, deletion, and update anomalies).
- Consider the relation: EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- **Insert Anomaly:** Cannot insert a project unless an employee is assigned to it. Conversely, cannot insert an employee unless an he/she is assigned to a project.

Informal Design Guidelines for Relational Schemas

- **Update Anomaly:** Changing the name of project number P1 from "Billing" to "Customer-Accounting" may cause this update to be made for all 100 employees working on project P1.
- **Delete 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.
- **Guideline 2.** Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

Informal Design Guidelines for Relational Schemas

3. NULL Values in Tuples

- When we group many attributes together into a "fat" relation, we end up with many NULL values which can waste space. NULL values also create problems when aggregate operations are applied. If NULL values are present in comparisons, the results may become unpredictable. Moreover, NULLs can have multiple interpretations.
- **Guideline 3.** As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Informal Design Guidelines for Relational Schemas

4. Generation of Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations.
- No spurious tuples should be generated by doing a natural-join of any relations.
- **Guideline 4.** Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Functional Dependencies

- Functional dependencies (FDs) are used to specify *formal measures* of the "goodness" of relational designs
- FDs and keys are used to define **normal forms** for relations
- FDs are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes
- Let X and Y be subsets of R. Then, set of attributes X *functionally determines* set of attributes Y, that is, Y *functionally depends* on X, if the value of X determines a unique value for Y. It is denoted by $X \rightarrow Y$

Functional Dependencies

- $X \rightarrow Y$ holds if whenever two tuples have the same value for X, they *must have* the same value for Y
- For any two tuples t1 and t2 in any relation r:
  If t1[X] = t2[X], *then* t1[Y] = t2[Y]
- $X \rightarrow Y$ in R specifies a *constraint* on all relation instances
- FDs are derived from the real-world constraints on the attributes

Functional Dependencies

- For example, consider the relation schema EMP_PROJ with FDs given below:

ENO	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

  - Employee number determines employee name

    $ENO \rightarrow ENAME$

  - Project number determines project name and location

    $PNUMBER \rightarrow \{PNAME, PLOCATION\}$

  - Employee number and project number determines the hours per week that the employee works on the project

    $\{ENO, PNUMBER\} \rightarrow HOURS$

Functional Dependencies

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every relation instance* r(R)
- If K is a key of R, then K functionally determines all attributes in R (since we never have two distinct tuples with t1[K] = t2[K])

Inference Rules for FDs

- Let F be the set of functional dependencies that are specified on relation schema R. However, we can infer other dependencies from the FDs in F
- The set of all functional dependencies that include F as well as all dependencies that can be inferred from F is called a **closure** of F; it is denoted by F^+.

Armstrong's inference rules:

IR1. (**Reflexive**) If Y *subset-of* X, then $X \rightarrow Y$

IR2. (**Augmentation**) If $X \rightarrow Y$, then $XZ \rightarrow YZ$

(Notation: XZ stands for $X \cup Z$)

IR3. (**Transitive**) If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Inference Rules for FDs

- Armstrong's axioms are **sound** because they do not generate any incorrect functional dependencies
- These are **complete**, because, for a given set F of functional dependencies, they allow us to generate all F^+

Some **additional inference rules** that are useful:

IR4. (**Decomposition**) If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

IR5. (**Union**) If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

IR6. (**Psuedotransitivity**) If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Inference Rules for FDs

- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)
- Some FDs are said to be **trivial** because they are satisfied by all relations. For example, $X \rightarrow X$ is satisfied by all relations involving attribute X. Formally, a functional dependency $X \rightarrow Y$ is trivial if $X \supseteq Y$; otherwise it is nontrivial.

Inference Rules for FDs

Example

$R = (A, B, C, G, H, I)$

$F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

some members of F^+

- $A \rightarrow H$
  - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
- $AG \rightarrow I$
  - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
- $CG \rightarrow HI$
  - from $CG \rightarrow H$ *and* $CG \rightarrow I$ using union rule.

  OR
  - Augmentation of $CG \rightarrow I$ to infer $CG \rightarrow CGI$, augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then by transitivity $CG \rightarrow HI$

Inference Rules for FDs

Algorithm to compute F^+

$F^+ = F$

repeat

for each functional dependency f in F^+

apply reflexivity and augmentation rules on f

add the resulting functional dependencies to F^+

for each pair of functional dependencies f_1 and f_2 in F^+

if f_1 and f_2 can be combined using transitivity

then add the resulting functional dependency to F^+

until F^+ does not change any further

Closure of Attribute Sets

- A systematic way to determine additional functional dependencies from the given set of functional dependencies is first to determine each set of attributes X that appears as a left-hand side of some functional dependency in F and then to determine the set of all attributes that are dependent on X.
- Thus, for each such set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F.
- X^+ is called the closure of X under F.

Closure of Attribute Sets

Algorithm:

$X^+ = X$

repeat

  old $X^+ = X^+$

  for each functional dependency $Y \rightarrow Z$ in F do

  if $X^+ \supseteq Y$ then $X^+ = X^+ \cup Z$

until (X^+ = old X^+)

Closure of Attribute Sets

- For example, consider the relation schema EMP_PROJ as given before with set F of functional dependencies

  ENO $\rightarrow$ ENAME

  PNUMBER $\rightarrow$ {PNAME, PLOCATION}

  {ENO, PNUMBER} $\rightarrow$ HOURS

- Using the above algorithm, we can calculate the following closure of attribute sets with respect of F:

  {ENO}+ = {ENO, ENAME}

  {PNUMBER}+ = {PNUMBER, PNAME, PLOCATION}

  {ENO, PNUMBER}+ = {ENO, ENAME, PNUMBER, PNAME, PLOCATION, HOURS}

Closure of Attribute Sets

Uses of Attribute Closure

- **Testing for superkey** – To test if α is a superkey we compute $\alpha+$ and check if $\alpha+$ contains all attributes of R
- **Testing for functional dependencies** – To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F+), just check if $\beta \subseteq \alpha+$
- **Computing closure of F** – For each $\gamma \subseteq$ R, we find the closure γ^+, and for each s $\subseteq \gamma^+$ we output a functional dependency γ -> s.

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 (i.e., if G^+ *subset-of* 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 (Irreducible) Sets of FDs

- A set of FDs is **minimal** if it satisfies the following conditions:

(1) Every dependency in F has a single attribute for 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 is a proper-subset-of X and still have a set of dependencies that is equivalent to F.

Minimal (Irreducible) Sets of FDs

- A **minimal cover** of a set of functional dependencies E is a minimal set of dependencies F that is equivalent to E.
- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set

Normalization of Relations

- Essentially, an un-normalized relation is a relation that contains repeating information. An un-normalized relation can also contain multivalued attributes, composite attributes, and their combinations.
- Sometimes un-normalized relations are signified by 0NF.
- The un-normalized relation is any relation in its raw state, and they commonly contain repeating values, and other characteristics.

Normalization of Relations

- To normalize a relation, we first decide whether a particular relation *R* is in good (appropriate normal) form.
- In the case that a relation *R* is not in "good" form, we **decompose** it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition

Normalization of Relations

- **Normalization** is the process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations to reduce redundency.
- **Normal form** is a condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form .

Normalization of Relations

- 2NF, 3NF, BCNF are based on keys and FDs of a relation schema
- 4NF based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs
- Additional properties may be needed to ensure a good relational design (**lossless join**, **dependency preservation**)

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are **hard to understand** or to **detect**
- The database designers ***need not*** normalize to the highest possible normal form. (usually up to 3NF, BCNF or 4NF)
- **Denormalization:** the process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

Definitions of Keys and Attributes Participating in Keys

- A **superkey** of a relation schema $R = \{A_1, A_2, ..., A_n\}$ is a set of attributes S *subset-of* R with the property that no two tuples t_1 and t_2 in any legal relation state r of R will have $t_1[S] = t_2[S]$
- 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.

Definitions of Keys and Attributes Participating in Keys

- 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*.
- A **Prime attribute** must be a member of *some candidate key*
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

First Normal Form

- First normal form (1NF) is now considered to be part of the formal definition of a relation.
- 1NF disallows composite attributes, multivalued attributes, and their combinations; attributes whose values *for an individual tuple* are non-atomic.
- An atomic value is considered to be indivisible units.
- For example, consider the DEPARTMENT relation schema as shown below

DNAME	DNUMBER	DMGRENO	DLOCATIONS

- We can see that this relation schema is not in 1NF because DLOCATIONS is not an atomic attribute.

First Normal Form

- To achieve 1NF for such a relation, we remove attribute DLOCATIONS that violets 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key DNUMBER of DEPARTMENT.
- The primary key of this relation is the combination {DNUMBER, DLOCATION} as shown below:

DNAME	DNUMBER	DMGRENO

DNUMBER	DLOCATION

Second Normal Form

- Uses the concepts of **FD**s, **primary key**

Definitions:

- **Prime attribute -** attribute that is member of the primary key K
- **Full functional dependency -** a FD $Y \rightarrow Z$ where removal of any attribute from Y means the FD does not hold any more. Its opposite is **partial dependency**.
- For example, consider EMP_PROJ relation schema as shown below:

ENO	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

Second Normal Form

Examples

-{ENO, PNUMBER} → HOURS is a full FD since neither ENO → HOURS nor PNUMBER → HOURS hold

- {ENO, PNUMBER} → ENAME is *not* a full FD (it is called a *partial dependency*) since ENO → ENAME also holds

- A relation schema R is in **second normal form** (**2NF**) if every non-prime attribute A in R is fully functionally dependent on the primary key
- R can be decomposed into 2NF relations via the process of 2NF normalization

Second Normal Form

- For example, consider EMP_PROJ relation schema as shown before. This relation schema is in 1NF but not in 2NF. The nonprime attribute ENAME functionally depends on ENO; that is ENO $\rightarrow$ ENAME holds.
- Also, PNAME and PLOCATION functionally depend on PNUMBER, that is, PNUMBER $\rightarrow$ {PNAME, PLOCATION} holds. Hence, the decomposition of EMP_PROJ includes the following three schemas:

ENO	**PNUMBER**	**HOURS**

ENO	**ENAME**

PNUMBER	**PNAME**	**PLOCATION**

Third Normal Form

- Based on the concept of transitive dependency.
- **Transitive functional dependency -** a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$ and Y is neither a candidate key nor a subset of any key of R. For example, consider EMP_DEPT relation schema as shown below:

ENAME	ENO	DOB	ADDRESS	DNUMBER	DNAME	DMGRENO

- ENO $\rightarrow$ DMGRENO is a *transitive* FD since

ENO $\rightarrow$ DNUMBER and DNUMBER $\rightarrow$ DMGRENO hold

- ENO $\rightarrow$ ENAME is *non-transitive* since there is no set of attributes X where ENO $\rightarrow$ X and X $\rightarrow$ ENAME

Third Normal Form

- A relation schema R is in **third normal form** (**3NF**) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- The test for 3NF involves that relation should not have a nonkey attribute functionally determined by another nonkey attribute (or a set of nonkey attributes); that is, there should no transitive dependency of a nonkey attribute on the primary key.
- R can be decomposed into 3NF relations via the process of 3NF normalization and set up relation that includes nonkey attribute(s) that functionally determines other nonkey attribute(s).

ENAME	ENO	DOB	ADDRESS	DNUMBER

DNUMBER	DNAME	DMGRENO

General Normal Form Definitions (For Multiple Keys)

- The above definitions consider the primary key only
- The following more general definitions take into account relations with multiple candidate keys
- A relation schema R is in **second normal form** (**2NF**) if every non-prime attribute A in R is fully functionally dependent on *every key* of R

General Normal Form Definitions

Definition:

- **Superkey** of relation schema R - a set of attributes S of R that contains a key of R
- A relation schema R is in **third normal form** (**3NF**) if whenever a FD X $\rightarrow$ A holds in R, then either:

  (a) X is a superkey of R, or

  (b) A is a prime attribute of R

NOTE: Boyce-Codd normal form disallows condition (b) above

BCNF (Boyce-Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form** (**BCNF**) if whenever an FD X $\rightarrow$ A holds in R, then X is a superkey of R
- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- Every relation in BCNF is also in 3NF
- The goal is to have each relation in BCNF (or 3NF)

Properties of Relational Decomposition

Attribute preservation property: Each attribute in R will appear in at least one relation schema R_i in the decomposition so that no attributes are "lost".

Properties of Relational Decomposition

Dependency Preservation Property of a Decomposition :

Definition: Given a set of functional dependencies *F* on *R;* After decomposition the attributes in $X \rightarrow Y$ should be contained in R_i, where *Ri* is one of the decomposition of R. *Hence, the function dependencies on each relation schema* R_i *in the decomposition D is the set of functional dependencies in* F^+*, the closure of F, such that all their left- and right-hand-side attributes are in* R_i.

Properties of Relational Decomposition

Dependency Preservation Property of a Decomposition (cont.):

Dependency Preservation Property: a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R is **dependency-preserving** with respect to F if the union of the functional dependencies on each R_i in D is equivalent to F.

Claim 1: It is always possible to find a dependency-preserving decomposition D with respect to F such that each relation R_i in D is in 3NF.

Properties of Relational Decomposition

Lossless (Non-additive) Join Property of a Decomposition:

Definition:

Lossless join property: a decomposition $D = \{R_1, R_2, ..., R_n\}$ of R has the **lossless (nonadditive) join property** with respect to the set of dependencies F on R if, for *every* relation state r of R that satisfies F, the following holds, where $\bowtie$ is the natural join of all the relations in D:

$$r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n = r$$

Note: The word loss in *lossless* refers to *loss of information*, not to loss of tuples. In fact, for "loss of information" a better term is "addition of spurious information"

Properties of Relational Decomposition

Testing Binary Decompositions for Lossless Join Property:

- **Binary Decomposition**: decomposition of a relation R into two relations.
- **PROPERTY (lossless join test for binary decompositions):** A decomposition $D = \{R_1, R_2\}$ of R has the lossless join property with respect to a set of functional dependencies F on R *if and only if* either
  - The FD $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or
  - The FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+.

Comparison of 3NF and BCNF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

Multivalued Dependencies and Fourth Normal Form

(a) The EMP relation with two MVDs: ENAME —>> PNAME and ENAME —>> DNAME.

(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Multivalued Dependencies and Fourth Normal Form

Definition:

- A **multivalued dependency** (**MVD**) X —>> Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R - (X \cup Y))$:

  $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.

  $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.

  $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

- An MVD X —>> Y in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) $X \cup Y = R$.

Multivalued Dependencies and Fourth Normal Form

Inference Rules for Functional and Multivalued Dependencies:

IR1 (**reflexive rule for FDs**): If $X \supseteq Y$, then $X \rightarrow Y$.

IR2 (**augmentation rule for FDs**): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

IR3 (**transitive rule for FDs**): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

IR4 (**complementation rule for MVDs**): $\{X \twoheadrightarrow Y\} \models X \twoheadrightarrow (R - (X \cup Y))\}$.

IR5 (**augmentation rule for MVDs**): If $X \twoheadrightarrow Y$ and $W \supseteq Z$ then $WX \twoheadrightarrow YZ$.

IR6 (**transitive rule for MVDs**): $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$.

IR7 (**replication rule for FD to MVD**): $\{X \rightarrow Y\} \models X \twoheadrightarrow Y$.

IR8 (**coalescence rule for FDs and MVDs**): If $X \twoheadrightarrow Y$ and there exists W with the properties that (a) $W \cap Y$ is empty, (b) $W \rightarrow Z$, and (c) $Y \supseteq Z$, then $X \rightarrow Z$.

Multivalued Dependencies and Fourth Normal Form

Definition:

- A relation schema R is in **4NF** with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \twoheadrightarrow Y$ in F^+, X is a superkey for R.

Note: F^+ is the (complete) set of all dependencies (functional or multivalued) that will hold in every relation state r of R that satisfies F. It is also called the **closure** of F.

Multivalued Dependencies and Fourth Normal Form

Lossless (Non-additive) Join Decomposition into 4NF Relations:

- The relation schemas R_1 and R_2 form a lossless (non-additive) join decomposition of R with respect to a set F of functional *and* multivalued dependencies if and only if

$$(R_1 \cap R_2) \text{ —>> } (R_1 - R_2)$$

or by symmetry, if and only if

$$(R_1 \cap R_2) \text{ —>> } (R_2 - R_1)).$$