Theory of Relational Database Design and Normalization

(Based on Chapter 14 and some part of Chapter 15 in Fundamentals of Database Systems by Elmasri and Navathe, Ed. 3)

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

3 Normal Forms Based on Primary Keys
   3.1 Introduction to Normalization
   3.2 First Normal Form
   3.3 Second Normal Form
   3.4 Third Normal Form

4 General Normal Form Definitions (For Multiple Keys)

5 BCNF (Boyce-Codd Normal Form)

6 Designing a Set of Relations
1 Informal Design Guidelines for Relational Databases

- What is relational database design?
  The grouping of attributes to form "good" relation schemas

- Two levels of relation schemas:
  - 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?

- We first discuss informal guidelines for good relational design

- Then we discuss formal concepts of functional dependencies and normal forms
  - 1NF (First Normal Form)
  - 2NF (Second Normal Form)
  - 3NF (Third Normal Form)
  - BCNF (Boyce-Codd Normal Form)

- Additional types of dependencies, further normal forms, relational design algorithms by synthesis are discussed in Chapter 15
1.1 Semantics of the Relation Attributes

**GUIDELINE 1:** Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).

- Attributes of different entities (EMPLOYEES, DEPARTMENTs, PROJECTs) should not be mixed in the same relation

- Only foreign keys should be used to refer to other entities

- Entity and relationship attributes should be kept apart as much as possible.

*Bottom Line:* Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.
INSERT FIGURE 14.1
1.2 Redundant Information in Tuples and Update Anomalies

-Mixing attributes of multiple entities may cause problems

- Information is stored redundantly wasting storage

- Problems with update anomalies:
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies
EXAMPLE OF AN UPDATE ANOMALY

Consider the relation:
EMP_PROJ (Emp#, Proj#, Ename, Pname, No_hours)

**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.

**Insert Anomaly:** Cannot insert a project unless an employee is assigned to.

Inversely- Cannot insert an employee unless an he/she is assigned to a project.

**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.
INSERT FIGURE 14.3
GUIDELINE 2: Design a schema that does not suffer from the insertion, deletion and update anomalies. If there are any present, then note them so that applications can be made to take them into account.

1.3 Null Values in Tuples

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

- Attributes that are NULL frequently could be placed in separate relations (with the primary key).

- Reasons for Nulls:
  a. attribute not applicable or invalid
  b. attribute value unknown (may exist)
  c. value known to exist, but unavailable

1.4 Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations.

- 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.

- There are two important properties of decompositions: (a) non-additive or losslessness of the corresponding join, (b) preservation of the functional dependencies. Note that property (a) is extremely important and cannot be sacrificed. Property (b) is less stringent and may be sacrificed. (See Chapter 15).
2.1 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.

- A set of attributes $X$ functionally determines a set of attributes $Y$ if the value of $X$ determines a unique value for $Y$.

- $X \rightarrow Y$ holds if whenever two tuples have the same value for $X$, they must have the same value for $Y$.

- For any two tuples $t_1$ and $t_2$ in any relation instance $r(R)$:
  
  If $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

- $X \rightarrow Y$ in $R$ specifies a constraint on all relation instances $r(R)$.
- Written as $X \rightarrow Y$; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow: $\rightarrow$).

- FDs are derived from the real-world constraints on the attributes.
Examples of FD constraints:
- social security number determines employee name
  \[ \text{SSN} \rightarrow \text{ENAME} \]

- project number determines project name and location
  \[ \text{PNUMBER} \rightarrow \{\text{PNAME, PLOCATION}\} \]

- employee ssn and project number determines the hours per week that the employee works on the project
  \[ \{\text{SSN, PNUMBER}\} \rightarrow \text{HOURS} \]

- 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 \( t_1[K] = t_2[K] \))
2.2 Inference Rules for FDs

- Given a set of FDs $F$, we can infer additional FDs that hold whenever the FDs in $F$ hold.

Armstrong's inference rules:
A1. (Reflexive) If $Y \subseteq X$, then $X \rightarrow Y$
A2. (Augmentation) If $X \rightarrow Y$, then $XZ \rightarrow YZ$
   (Notation: $XZ$ stands for $X \cup Z$)
A3. (Transitive) If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- A1, A2, A3 form a sound and complete set of inference rules.

Some additional inference rules that are useful:
(Decomposition) If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
(Union) If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
(Pseudotransitivity) If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

- The last three inference rules, as well as any other inference rules, can be deduced from A1, A2, and A3 (completeness property)
- **Closure** of a set $F$ of FDs is the set $F^+$ of all FDs that can be inferred from $F$

- Closure of a set of attributes $X$ with respect to $F$ is the set $X^+$ of all attributes that are functionally determined by $X$

- $X^+$ can be calculated by repeatedly applying A1, A2, A3 using the FDs in $F$

**2.3 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 $F^+$) : $G^+$ $\subset$ $F^+$.

- $F$ and $G$ are equivalent if $F$ covers $G$ and $G$ covers $F$

- There is an algorithm for checking equivalence of sets of FDs
2.4 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 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 $Y$ proper-subset-of $X$ ($Y \subset X$) 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

- 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 (e.g., see algorithms 15.1 and 15.4)
3 Normal Forms Based on Primary Keys

3.1 Introduction to Normalization

- **Normalization**: Process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- **Normal form**: Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
- 2NF, 3NF, BCNF based on keys and FDs of a relation schema
- 4NF based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs (Chapter 15)

- Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation; Chapter 14)

3.2 First Normal Form

- Disallows composite attributes, multivalued attributes, and **nested relations**; attributes whose values for an individual tuple are non-atomic
- Considered to be part of the definition of relation
INSERT FIGURE 14.8
INSERT FIGURE 14.9
3.3 Second Normal Form

- Uses the concepts of FDs, primary key

Definitions:
- **Prime attribute** - attribute that is member of the primary key K

- **Full functional dependency** - a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more

Examples:
- \{SSN, PNUMBER\} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold
- \{SSN, PNUMBER\} -> ENAME is not a full FD (it is called a partial dependency ) since SSN -> 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
INSERT FIGURE 14.10
INSERT FIGURE 14.11
3.4 Third Normal Form

Definition:
- **Transitive functional dependency** - a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$

Examples:
- SSN $\rightarrow$ DMGRSSN is a transitive FD since SSN $\rightarrow$ DNUMBER and DNUMBER $\rightarrow$ DMGRSSN hold
- SSN $\rightarrow$ ENAME is *non-transitive* since there is no set of attributes $X$ where SSN $\rightarrow$ X and X $\rightarrow$ ENAME

- 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

- $R$ can be decomposed into 3NF relations via the process of 3NF normalization

**NOTE:**
In $X \rightarrow Y$ and $Y \rightarrow Z$, with $X$ as the primary key, we consider this a problem only if $Y$ is *not* a candidate key. When $Y$ is a candidate key, there is no problem with the transitive dependency.
E.g., Consider EMP (SSN, Emp#, Salary).
Here, SSN $\rightarrow$ Emp# $\rightarrow$ Salary and Emp# is a candidate key.
4 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

**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 -> 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
5 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

- The goal is to have each relation in BCNF (or 3NF)
INSERT FIGURE 14.12
INSERT FIGURE 14.13
6 DESIGNING A SET OF RELATIONS:

The approach of Relational Synthesis:

- Assumes that all possible functional dependencies are known.
- First constructs a minimal set of f.d.s
- Then applies algorithms that construct a target set of 3NF or BCNF relations.

- Additional criteria may be needed to ensure the set of relations in a relational database are satisfactory (see Algorithms 15.1 and 15.4).

Goals:

- Lossless join property (a must) – algorithm 15.2 tests for general losslessness.
- Dependency preservation property – algorithms 15.3 decomposes a relation into BCNF components by sacrificing the dependency preservation.
- Additional normal forms are discussed in Chapter 15
  - 4NF (based on multi-valued dependencies)
  - 5NF (based on join dependencies)