Chapter 7 – Functional dependencies and Normalization for Relational Databases

Database design may be performed using two approaches:

1. bottom-up or
2. top-down
    - A **bottom-up methodology** would consider the basic relationships among individual attributes as starting points, and it would use those to build up relations. This approach suffers from the problem of collecting a large number of binary attribute relationships as the starting point. This approach is also called ***design by synthesis*.**
    - A **top-down methodology** would start with a number of groupings of attributes into relations that have already been obtained from conceptual design and mapping activities. *Design by analysis* is then applied to the relations individually and collectively.

Informal Design Guidelines for Relation Schemas

Following are the informal guidelines that are to be followed while designing a database.

- Design a relation schema so that it is easy to explain its meaning.(Semantics of the attributes.
- Design a relation schema so that no updation anomalies are present in the relations. (Reducing the redundant values in tuples)
- Avoid placing attributes in a relation whose values may frequently be null. (Reducing the null values in tuples)
- Design relation schemas so that they can be JOINed without generating spurious tuples (Disallowing spurious tuples)

Semantics of the Relation Attributes:

- Each tuple in a relation should represent one entity or relationship instance.
- Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation

Guide Line 1: Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.

Redundant Information in Tuples and Update Anomalies:

- Mixing attributes of multiple entities may cause problems. Information is stored redundantly. Redundant Information
    - **Wastes storage**
    - **Causes update anomalies**
        - Insertion anomalies
        - Deletion anomalies
        - Modification anomalies

Consider the relation EMP-DEPT (ENAME, SSN, BDATE, ADDRESS,DNUMBER, DNAME, DMGRSSN) resulting from applying JOIN to EMPLOYEE AND DEPARTMENT relations.

EMP_DEPT

redundancy

ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN
Smith,John B.	123456789	1965-01-09	731 Fondren,Houston,TX	5	Research	333445555
Wong,Franklin T.	333445555	1955-12-08	638 Voss,Houston,TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle,Spring,TX	4	Administration	987654321
Wallace,Jennifer S.	987654321	1941-06-20	291 Berry,Bellaire,TX	4	Administration	987654321
Narayan,Ramesh K.	666884444	1962-09-15	975 FireOak,Humble,TX	5	Research	333445555
English,Joyce A.	453453453	1972-07-31	5631 Rice,Houston,TX	5	Research	333445555
Jabbar,Ahmad V.	987987987	1969-03-29	980 Dallas,Houston,TX	4	Administration	987654321
Borg,James E.	888665555	1937-11-10	450 Stone,Houston,TX	1	Headquarters	888665555

Insertion anomalies :

When inserting a new employee tuple, we need to enter all department information, or null. In the former case, there is a *possibility of inconsistency* if we do not enter all department information correctly.

It is difficult to insert a new department that has no employees. If we put null value for all employee attribute, SSN becomes null too, which *violates the entity integrity constraint*.

Deletion anomalies:

If we delete the last employee of a certain department, the *department information is lost*.

Modification anomalies:

If we change the attribute value(s) of a department, we must update the tuples of all employees who work for the department.

Guide Line 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.

Null Values in Tuples:

- If we have *fat* relations (that have too many attributes), it is possible that some attributes are not applicable to many tuples, which will lead to have many *null* values.
- Problems:
  - Waste Space at the storage level
  - Leads to problems with understanding the meaning of the attributes
  - Leads to problems with specifying JOIN operations.
- nulls can have Multiple interpretations
  - Attribute not applicable or invalid to this tuple
  - Attribute value is unknown (may exist) for this tuple.
  - The value is known to exist, but unavailable

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

Spurious Tuples:

- When a relation is decided to be decomposed into two relations, it is possible that we will *not recover the original information* from the two resulting relations unless the decomposition is carried out carefully.
- Example: Assume that EMP_PROJ is decomposed into EMP_LOCS and EMP_PROJ1. If we perform NATURAL-JOIN on EMP_LOC and EMP_PROJ1, we will end up with a collection of tuples that includes many *spurious (wrong)* tuples
- This bad design is caused by
    1. Decomposition was not done properly
    2. As a result, the JOIN attribute PLOCATION is *neither a primary key nor a foreign key*.

(a)

EMP_LOCS

ENAME	PLOCATION

p.k.

EMP_PROJ1

SSN	PNUMBER	HOURS	PNAME	PLOCATION

p.k.

(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	Product X	Bellaire
123456789	2	7.5	Product Y	Sugarland
666884444	3	40.0	Product Z	Houston
453453453	1	20.0	Product X	Bellaire
453453453	2	20.0	Product Y	Sugarland
333445555	2	10.0	Product Y	Sugarland
333445555	3	10.0	Product Z	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

Result of applying the NATURAL JOIN operation to the tuples above dotted lines in EMP_PROJ1 and EMP_LOCS, with generated spurious tuples marked by an asterisk.

	SSN	PNUMBER	HOURS	PNAME	PLOCATION	
	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.

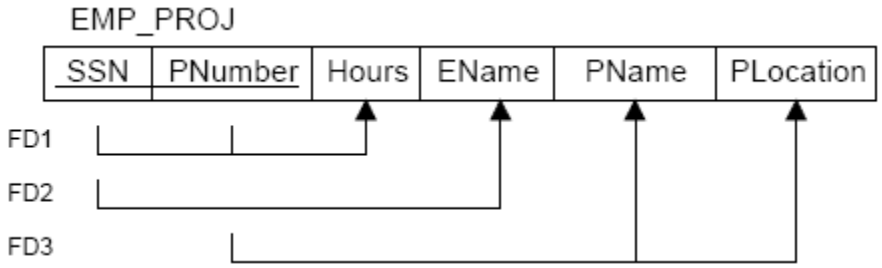
Guide Line 4: Design relation schemas so that they can be JOINed with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that *no* spurious tuples are generated.

Functional Dependencies

Definition of Functional Dependency

- A functional dependency specifies a constraint between two sets of attributes.
- Let R = {A_1, A_2, ..., A_n} be a *universal relation* consisting of all attributes in a database schema (all attributes in a database). A *functional dependency* X → Y, where X ⊆R and Y ⊆R, holds if
  $t_1[X] = t_2[X] \rightarrow t_1[Y] = t_2[Y]$ (_ means "implies")
- X uniquely (functionally) determines Y (Y is dependent on X). Y is functionally dependent on X.
- If X is a candidate key, then X → Y is true for any Y of R.
- X → Y does not imply Y → X.
- Relation extension r(R) that satisfies the functional dependency constraints is called *legal extensions* (or *legal relation states*).
- Functional dependencies should hold *at all times*.
- Functional dependencies are determined by the *semantics of the attributes*.

Example



FD1: {SSN, PNumber} → Hours
FD2: SSN → EName
FD3: PNumber → {PName, PLocation}

Inference Rules for Functional Dependencies

(IR1) (Reflexive rule): If X ⊇ Y, then X → Y.
(IR2) (Augmentation rule): If {X → Y}, then XZ → YZ.

(IR3) (Transitive rule): If {X → Y, Y → Z}, then X → Z.

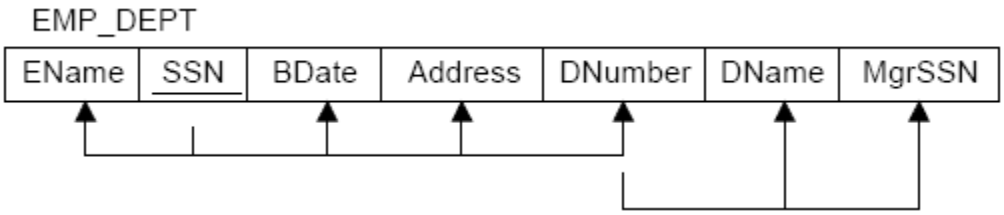
(IR4) (Decomposition (or projective) rule): If {X → YZ}, then X → Y.
(IR5) (Union (or additive) rule): If {X → Y, X → Z}, then X → YZ.
(IR6) (Pseudo transitive rule): If {X → Y, WY → Z}, then WX → Z.

A functional dependency X → Y is **trivial** if X ⊇ Y; otherwise it is **nontrivial**.

IR1 through IR3 are called *sound* and *complete* (called **Armstrong's inference rules**).

- ***Soundness***: Given F on R, any dependency that can be inferred form F using IR1 through IR3 holds in every relation state r of R that satisfies the dependencies in F.
- ***Completeness***: F^+ can be determined from F using only IR1 through IR3.

Example



F = {SSN → {ENAME, BDATE, ADDRESS, DNUMBER},
DNUMBER → {DNAME, DMGRSSN}}
From F of above example we can infer:

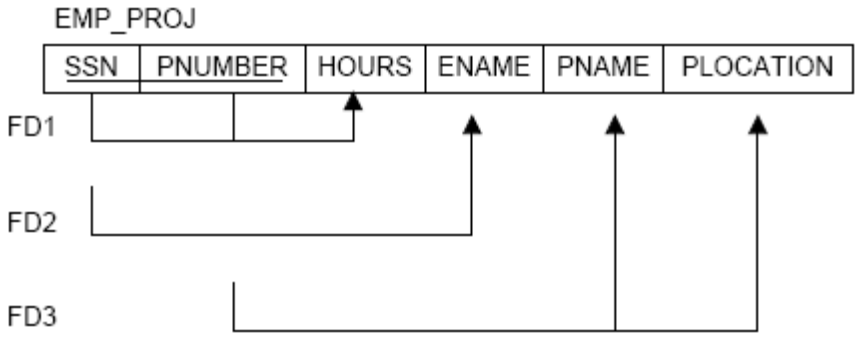
SSN → {DNAME, DMGRSSN},
SSN → SSN,
DNUMBER → DNAME

The **closure** of *F*, denoted by F^+**,** is the set of all functional dependencies that can be inferred from F.

Closure of X under F:

Given a set of functional dependencies F, *closure of X under F*, denoted by X^+ , is the set of all attributes that are functionally determined by X based on F. (Note that X is a set of attributes.)

Example



F = {{SSN, PNUMBER} → HOURS,
SSN → ENAME,
NUMBER → {PNAME, PLOCATION}}

SSN^+ = {SSN, ENAME}

Algorithm: to compute X^+

```
X+ := X;
repeat
    oldX+ = X+;
    for each fd Y → Z in F do
        if X+ ⊇ Y then X+ := X+ ∪ Z;
until (X+ = oldX+);
```

A set of functional dependencies E is **covered** *by* a set of functional dependencies F if every FD in E is also in F^+ (in other words, every FD in E can be inferred from F).

Two sets of functional dependencies E and F are **equivalent** if E covers F and F covers E
(Or if $E^+ = F^+$).

Example
Consider the following 2 sets of FDs: F = { A → C, AC → D, E → AD, E → H} and G = { A → CD, E → AH}. Check whether they are equivalent.

Solution:
F and G are equivalent if $F^+ = G^+$.
First represent F in canonical form. ie.,
F = {A → C, AC → D, E → A, E → D, E → H} and
G = {A → C, A → D, E → A, E → H}
In order to find F^+, find the closure of A, AC, E.
A^+ = {C}, AC+ = {D}, E+ = {A, C, D, H}
F^+ = {A → C, AC → D, E → A, E → C, E → D, E → H}
Similarly compute G^+.
G^+ = {A → C, A → D, E → A, E → A, E → C, E → D, E → H}
[Consider A → D in G^+, that implies, AC → CD ie., AC → C and AC → D.]

Since every dependency that is in F^+ is in G^+ we say that G covers F.
But the dependency A → D Which is in G^+ is not present in F^+.
So, F and G are not equivalent.

A set of functional dependencies F is **minimal** if it satisfies the following conditions:

1. Every functional dependency in F has a single attribute on the right hand side.
2. For every functional dependency X → Y, if we remove any proper subset of X from the left hand side, the functional dependency does not hold any more.
3. If we remove any functional dependency from F, the remaining set of functional dependencies is not equivalent to F.

A **minimal cover** of a set of functional dependencies of F is a minimal set of functional dependencies F_{min} that is equivalent to F.

Algorithm: for finding a minimal cover G for F

(F is a given set of dependencies and G is its minimal cover).

1. Set G := F.
2. Replace each functional dependency X → {A1, A2, … , An} in G by the n dependencies X → A1, X → A2, …, X → An.
3. For each functional dependency X → A in G
   For each attribute B that is an element of X
   If ((G – { X → A}) ∪ {(X – {B}) → A }) is equivalent to G,
   Then replace X → A with (X – {B}) → A in G.
4. for each remaining functional dependency X→ A in G,
   If (G – { X → A}) is equivalent to G, then remove X→ A from G.

Computing the Minimal Sets of FDs

We illustrate the above algorithm with the following example:
Let the given set of FDs be *E:* {$B \rightarrow A$, $D \rightarrow A$, $AB \rightarrow D$}.We have to find the minimum cover of *E*.

- All above dependencies are in canonical form; so we have completed step 1 of Algorithm and can proceed to step 2.
- In step 2 we need to determine if $AB \rightarrow D$ has any redundant attribute on the left-hand side; that is, can it be replaced by $B \rightarrow D$ or $A \rightarrow D$?
- Since B → A, by augmenting with *B* on both sides (IR2), we have $BB \rightarrow AB$, or $B \rightarrow AB$ (i). However, $AB \rightarrow D$ as given (ii).
  Hence by the transitive rule (IR3), we get from (i) and (ii), $B \rightarrow D$.
  Hence $AB \rightarrow D$ may be replaced by $B \rightarrow D$.
- We now have a set equivalent to original *E* , say E' : {$B \rightarrow A$, $D \rightarrow A$, $B \rightarrow D$}.
- No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.
- In step 3 we look for a redundant FD in E′.
- By using the transitive rule on $B \rightarrow D$ and $D \rightarrow A$, we derive $B \rightarrow A$. Hence $B \rightarrow A$ is redundant in E' and can be eliminated.
- Hence the minimum cover of E is {$B \rightarrow D$, $D \rightarrow A$}.

Normal Forms Based on Primary Keys

- The normalization process takes a relation schema through a series of tests to "certify" whether it satisfies a certain normal form. It is a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of
  - Minimizing redundancy
  - Minimizing modification anomalies.
- The normal form of a relation refers to the highest normal form condition that it meets.
- The process of storing the join of higher normal form relations as a base relation is known as **denormalization.**

First Normal Form (1NF): The domains of attributes must include only *atomic (simple, indivisible) values*, and the value of any attribute in a tuple must be a *single value* from the domain of that attribute.

(a)

(b)

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

For Example:

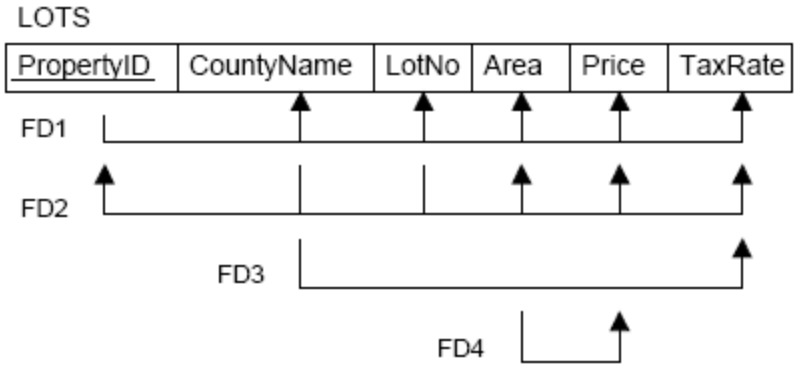
- The DEPARTMENT schema is not in 1NF because DLOCATION is not a single valued attribute.
- The relation should be split into two relations. A new relation DEPT_LOCATIONS is created and the primary key of DEPARTMENT, DNUMBER, becomes an attribute of the new relation. The primary key of this relation is {DNUMBER, DLOCATION}

- Alternative solution: Leave the DLOCATION attribute as it is. Instead, we have one tuple for each location of a DEPARTMENT. Then, the relation is in 1NF, but redundancy exists (Figure c).

Second Normal Form (2NF): A relation schema R is in 2*NF* if every nonprime attribute A in R is fully dependent on the key of R.

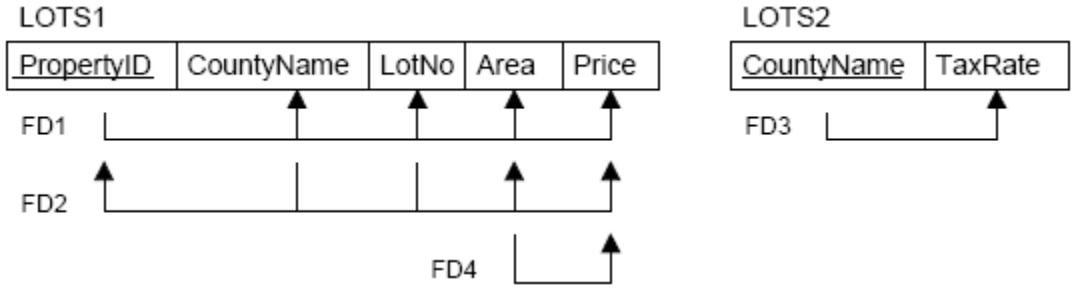
Prime attribute - attribute that is a member of the candidate key K
Full functional dependency - a FD Y → Z where removal of any attribute from Y means the FD does not hold any more

Examples: - {SSN, PNUMBER} → HOURS is a full FD since neither SSN → HOURS nor PNUMBER → HOURS hold. But {SSN, PNUMBER} → ENAME is *not* a full FD (it is called a *partial dependency*) since SSN → ENAME also holds .



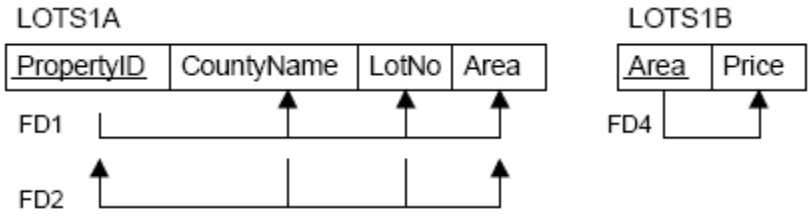
The relation schema LOTS is not in 2NF because TaxRate is partially dependent on the candidate key {CountyName, LotNo}; FD3 violates 2NF.

LOTS must be decomposed into LOTS1 and LOTS2.



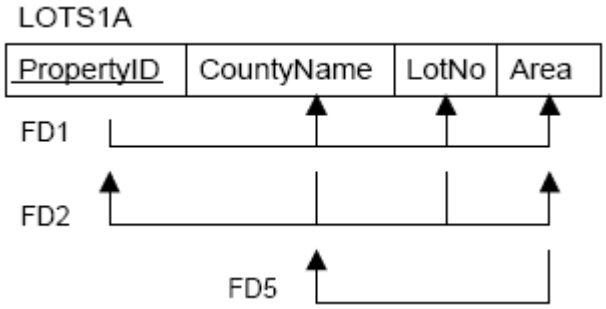
Third Normal Form (3NF): A relation schema R is in 3NF if, whenever a nontrivial functional dependency X → A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

LOTS1 above is not in 3NF because, in fd4, Area is not a superkey and Price is not a prime attribute. LOTS1 must be decomposed into LOTS1A and LOTS1B.



Boyce-Codd Normal Form (BCNF): A relation schema R is in BCNF if whenever a nontrivial functional dependency X → A holds in R, then X is a super key of R. BCNF is stricter than 3NF.

LOTS1 with a new functional dependency FD5 is in 3NF but not in BCNF.



After decomposition as shown below, now they are in BCNF.