Chapter 6

Database Schema Design

Course Notes on Database Systems

By Prof. Rahul Simha

6.1 Database Schema Design: Introduction

- A *Database Administrator* (DBA) is responsible for designing the schema of a relational database
  ⇒ need to decide which attributes go into which relations.
- Other DBA responsibilities:
  - Analysis of user needs.
  - Performance.
  - Security and access (for users as well as dbase programmers).
- We will focus on schema design.
- Schema design usually proceeds in two phases:
  1. Use *informal guidelines* to create an initial design.
  2. Use *formal guidelines* to improve initial design.
- Consider an airline employee dbase:
  - Schema $S1$:

    EMP (NAME, SSN, FLT_ID, START_APT, END_APT, DEPTNO, DNAME, MGRSSN)

  - Schema $S2$:

    EMP (NAME, SSN, DEPTNO)
    CREW (SSN, FLT_ID)
    FLIGHT (FLT_ID, START_APT, END_APT)
    DEPT (DEPTNO, DNAME, MGRSSN)

  Which schema is better?
- Review of terminology:

- **superkey** – any group of attributes that can uniquely identify a tuple *in any instance.*

  e.g., for a particular instance (FNAME, LNAME) may appear to be a key – but it can't be guaranteed.

  ⇒ (FNAME, LNAME) is a poor choice for a key.
- **key** – a minimal superkey, e.g.,
  (NAME, SSN) is a superkey (but not a key)
  (SSN) is a key (and also a superkey)
- **primary key** – one key designated for general use.
- **foreign key** – a set of attributes in relation R that is the primary key for relation S.
- **Domain constraint** – proper typing of values.

- **Key constraint 1** – no two tuples can have identical keys
  If a tuple is found that violates the condition, then either
  - the tuple should be rejected or,
  - the key is not truly a key (i.e., a bad choice)
- **Key constraint 2** – no primary key value can be null.
- **Referential integrity constraint** – can't have a tuple whose foreign key values don't exist in the foreign relation *instance.*

  For example, consider

  EMP1 (NAME, SSN, FLT_ID, START_APT, END_APT, DEPTNO)
  DEPT (DEPTNO, DNAME, MGRSSN)

  e.g. <John, B, Smith, ... , 9> and DEPTNO=9 does not exist at present.

6.2 Informal Guidelines

1. **Try to make user interpretation easy.**

   For example, compare

   - Schema $S1$:

     EMP (NAME, SSN, FLT_ID, START_APT, END_APT, DEPTNO, DNAME, MGRSSN)

   - Schema $S2$:

     EMP (NAME, SSN, DEPTNO)
     CREW (SSN, FLT_ID)
     FLIGHT (FLT_ID, START_APT, END_APT)
     DEPT (DEPTNO, DNAME, MGRSSN)

   Perhaps $S1$ has too much information (to absorb) per tuple

2. **Try to reduce redundancy.**

   Suppose there are only a few departments
   ⇒ MGRSSN and DNAME are unnecessarily repeated too often in EMP.

NAME	...	DEPTNO	DNAME	MGRSSN
Abel	...	5	Crew	111-22-3334
Aitken	...	3	Ticketing	222-33-4445
Al Khwarizmi	...	5	Crew	111-22-3334
Archimedes	...	5	Crew	111-22-3334
⋮	⋮	⋮	⋮	⋮
Zeno	...	3	Ticketing	222-33-4445
Zuse	...	5	Crew	111-22-3334

   On the other hand, $S2$ repeats DEPTNO in DEPT
   ⇒ DEPTNO (integer) is smaller than DNAME (string) and MGRSSN (string)

3. **Try to avoid update anomalies**

   Consider the schemas:

   - Schema $S1$:

     EMP (NAME, SSN, FLT_ID, START_APT, END_APT, DEPTNO, DNAME, MGRSSN)

   - Schema $S2$:

     EMP (NAME, SSN, DEPTNO)
     CREW (SSN, FLT_ID)
     FLIGHT (FLT_ID, START_APT, END_APT)
     DEPT (DEPTNO, DNAME, MGRSSN)

   Both have same attibutes. Unfortunately, $S1$ can create problems called *update anomalies*.

   - **Insertion anomalies**

     (a) Consider inserting "John Smith works in Dept. 5", i.e.,
     <John, Smith, 123456789, ... , 5, <dname>, <mgrssn> >.

     Every time a tuple of this sort is entered, we have to check that DNAME is correct
     ⇒ we have to scan the whole relation (worst-case)

     (b) Consider creating a new department: DEPTNO=9, DNAME='Security' (with no employees yet)
     Only one way to insert this info
     ⇒ create NULL values for employee info
     ⇒ but that means a NULL primary key value (SSN)!

   - **Deletion anomalies**
     If we delete the last employee in the 'Crew' department, e.g.

     ```
     <John, Smith, 123456789, ... , 5, 'Crew', ... >
     ```

     then we will lose the information
     "Department 5 is the Crew department".

- **Modification anomalies**
  Suppose we change the manager of department 5
  ⇒ we have to change MGRSSN for all department 5 employees
  ⇒ full scan of database

Thus, schema $S1$ has many problems. On the other hand:

- $S1$ – has 1 relation.
- $S2$ – has 4 relations.
- For many queries, we will need more joins using $S2$.
- SQL code with $S2$ will be more complicated because of the extra joins
  (One solution: use $S2$ but create views based on needed joins)

4. **Try to avoid too many NULL values**.
   - This may occur in 'fat' relations (with too many attributes).
   - Space is wasted.
   - Problems occur when using aggregate functions like **count** or **sum**.
   - NULLs can have different intentions:
     (a) The attribute does not apply.
     (b) Value is unknown, and will remain unknown.
     (c) Value is unknown at present.

5. **Beware of the Spurious Tuple Problem**.
   Consider the following two schemas:
   - Schema $S1$:
     EMP (NAME, HOME_APT, SSN, FLT_ID)
   - Schema $S2$:
     EMPNEW (NAME, SSN, FLT_ID)
     HOMEBASE (NAME, HOME_APT)

   First, let us see how a relation in $S1$ can be converted to relations in $S2$, e.g., consider this data:

EMP	NAME	HOME_APT	SSN	FLT_ID
	Smith	National	111-22-3333	18
	Smith	JFK	222-33-4444	48
	Jones	La Guardia	333-44-5555	119

To create EMPNEW:

$$\text{EMPNEW} := \Pi_{\text{NAME, SSN, FLT\_ID}} \ (\text{EMP})$$

Thus,

EMPNEW	NAME	SSN	FLT_ID
	Smith	111-22-3333	18
	Smith	222-33-4444	48
	Jones	333-44-5555	119

Similarly, to create HOME_BASE:

$$\text{HOMEBASE} := \Pi_{\text{NAME, HOME\_APT}} \ (\text{EMP})$$

In this case,

HOMEBASE	NAME	HOME_APT
	Smith	National
	Smith	JFK
	Jones	La Guardia

Now, suppose we are using $S2$ and we want to recreate $S1$ (say, as a *view*):

$$\text{EMP} := \text{EMPNEW} * \text{HOME\_BASE}.$$

We get the following join:

EMP	NAME	HOME_APT	SSN	FLT_ID	
	Smith	National	111-22-3333	18	
	Smith	JFK	111-22-3333	18	$*$
	Smith	JFK	222-33-4444	48	
	Smith	National	222-33-4444	48	$*$
	Jones	La Guardia	333-44-5555	119	

Here, the $*$-tuples are *spurious*!

What happened? Since NAME is not a key, a careless join produced wrong results.

Summary of problems:

- Insertion, deletion and modification anomalies.
- Too many NULLs.
- Spurious tuples.

$\Rightarrow$ We need a theory of schema design
$\Rightarrow$ *functional dependencies* and *normalization*

6.3 Functional Dependencies

- First, some convenient notions (and notation):
  - Suppose our relational database has attributes $A_1, \ldots, A_n$.
  - Let R denote the schema $R = (A_1, \ldots, A_n)$.
  - Typically, of course, we will have several relations, e.g., EMP(A_1, A_3, A_9), DEPT(A_9, A_7, A_8) ... etc.
  - However, we will pretend there is a relation with *all* the attributes, i.e., with schema $R = (A_1, A_2, \ldots, A_n)$.
- **Definition**. A **functional dependency (FD)** between two sets of attributes X and Y, denoted by $X \to Y$, specifies a certain relationship or connection between X and Y. Specifically, it says:
  if t_1 and t_2 are any two tuples in any instance of R such that
  $$t_1[X] = t_2[X]$$
  then
  $$t_1[Y] = t_2[Y]$$
  Intuitively, $X \to Y$ means: if you know the X-values of a tuple, then that uniquely determines the Y-values.

  Note: X and Y can be single attributes or *groups* of attributes.
- Example:
  Consider the relational database schema:

  EMP (NAME, SSN, FLT_ID, START_APT, END_APT)

  Suppose
  $$\begin{aligned} X &= \{\text{SSN}\} \\ Y &= \{\text{NAME}\} \end{aligned}$$

Then $X \rightarrow Y$, i.e., SSN uniquely determines NAME

From the definition, if we are given two tuples t_1 and t_2, e.g.,

$$t_1 = \text{<111-22-3333,...,Smith,...>}$$
$$t_2 = \text{<111-22-3333,...,Smith,...>}$$

where

$$t_1[X] = t_2[X] \quad (\text{i.e.,} \;\; t_1[\text{SSN}] = t_2[\text{SSN}])$$

then it must be true that

$$t_1[Y] = t_2[Y] \quad (\text{i.e.,} \;\; t_1[\text{NAME}] = t_2[\text{NAME}]).$$

Thus, we can't have two tuples with the same SSN and different NAME's.

Similarly, the functional dependency

$$\{\text{FLT\_ID}\} \rightarrow \{\text{START\_APT, END\_APT}\}$$

is a reasonable assumption or a reasonable constraint to declare.

- Functional dependencies are specified to capture dependencies for *all* instances.

  It may just happen that employee names (NAME) are <u>all</u> different for a particular instance.

  $\Rightarrow$ we may be led to believe that $\{\text{NAME}\} \rightarrow \{\text{FLT\_ID}\}$.

  But, later on, another 'Smith' might join the airline
  $\Rightarrow$ this would be a poor choice of a FD.

  A FD is a property of the meaning *of attributes*

  $\Rightarrow$ it should hold for all possible instances.

- FD's for specific relations.

  Although we have defined FD's on the universe of attributes, we will often discuss FD's within particular relations.

  For example, the database schema might be:

  (NAME, SSN, FLT_ID, START_APT, END_APT, DEPTNO, DNAME, MGRSSN).

  We might have the relation

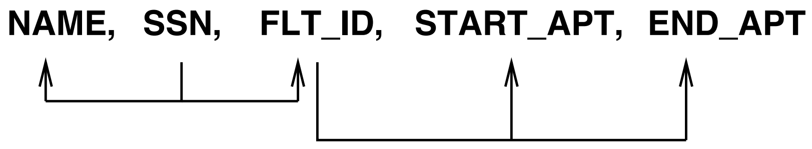
  FLIGHT (FLT_ID, START_APT, END_APT)

  Here, we can identify the FD

  $$\{\text{FLT\_ID}\} \rightarrow \{\text{START\_APT, END\_APT}\}$$

  in FLIGHT.

- Sometimes a diagram is used to show FD's, e.g.,



  Here the FD's shown are:

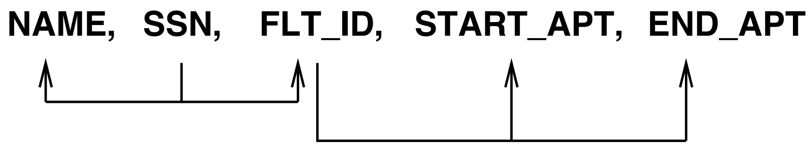
  $$\begin{array}{lll} \{\text{SSN}\} & \rightarrow & \{\text{NAME, FLT\_ID}\} \\ \{\text{FLT\_ID}\} & \rightarrow & \{\text{START\_APT, END\_APT}\} \end{array}$$

- Summary:
  - A FD is defined between sets of attributes.
  - The FD $X \rightarrow Y$ says "The X-attributes completely determine the Y-attributes".

6.4 Sets of Functional Dependencies

- Consider the following example:

  EMP (NAME, SSN, FLT_ID, START_APT, END_APT)



  The obvious FD's are:

$$\begin{array}{lcl} \{\text{SSN}\} & \rightarrow & \{\text{FLT\_ID}\} \\ \{\text{FLT\_ID}\} & \rightarrow & \{\text{START\_APT,END\_APT}\} \end{array}$$

  Let F denote the above *collection* of FD's.

  From F, we can infer

$$\{\text{SSN}\} \rightarrow \{\text{START\_APT, END\_APT}\}.$$

  Why? Because, SSN uniquely determines FLT_ID and FLT_ID uniquely determines {START_APT, END_APT}.

  Also, the following FD's are examples of trivial FD's inferred from F.

$$\begin{array}{lcl} \{\text{SSN}\} & \rightarrow & \{\text{SSN}\} \\ \{\text{SSN, NAME}\} & \rightarrow & \{\text{NAME}\} \end{array}$$

- **Definition**. Suppose F is a set of FD's. Then, F^+, the **closure** of F is the set of FD's that includes F and all the FD's that can be *inferred* from F.

6.5 Inference Rules for FD Sets

- NOTE the following:
  1. When we say $X \rightarrow Y$, X and Y are *subsets* of the universe of attributes.
  2. For convenience, we will sometimes drop the set notation and commas within sets.

     Suppose F is the set of FD's:
     $$X \rightarrow Y$$
     $$X \rightarrow Z.$$
     Then, from F we can infer:
     $$X \rightarrow YZ.$$
     Here, YZ denotes the *union* of Y and Z.

     For example, F is
     $$\begin{array}{rcl} X = \{\text{SSN}\} & \rightarrow & \{\text{NAME }\} = Y \\ X = \{\text{SSN}\} & \rightarrow & \{\text{FLT\_ID}\} = Z. \end{array}$$
     From which we conclude
     $$X = \{\text{SSN}\} \rightarrow \{\text{NAME, FLT\_ID}\} = Y \cup Z = YZ$$

- From above, we can devise a rule: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.

  Such a rule is called an **inference rule**.

- Standard inference rules for FD's:

  1. **Reflexive rule**. *If* $Y \subseteq X$ *then* $X \rightarrow Y$*.*
     e.g. {SSN, NAME} $\rightarrow$ {NAME}
  2. **Augmentation rule**. *If* $X \rightarrow Y$ *then* $XZ \rightarrow YZ$ *for any group of attributes* Z*.*
     e.g.

     $$\begin{array}{lcl} \text{X} & = & \{\text{SSN}\} \\ \text{Y} & = & \{\text{NAME}\} \\ \text{Z} & = & \{\text{START\_APT}\} \end{array}$$

     Then, {SSN} $\rightarrow$ {NAME} implies

     $$\{\text{SSN, START\_APT}\} \rightarrow \{\text{NAME, START\_APT}\}$$

  3. **Transitive rule**. *If* $X \rightarrow Y$ *and* $Y \rightarrow Z$ *then* $X \rightarrow Z$*.*
     e.g., the FD's

     $$\begin{array}{lcl} \{\text{SSN}\} & \rightarrow & \{\text{FLT\_ID}\} \\ \{\text{FLT\_ID}\} & \rightarrow & \{\text{END\_APT}\} \end{array}$$

     together imply

     $$\{\text{SSN}\} \rightarrow \{\text{END\_APT}\}.$$

  4. **Decomposition rule**. *If* $X \rightarrow YZ$ *then* $X \rightarrow Y$*.*
     e.g. the FD

     $$\{\text{SSN}\} \rightarrow \{\text{NAME, FLT\_ID}\}$$

     implies

     $$\{\text{SSN}\} \rightarrow \{\text{NAME}\}.$$

5. **Union rule**. *If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$.*
   e.g. the FD's

$$\begin{array}{lcl} \{\text{SSN}\} & \rightarrow & \{\text{NAME}\} \\ \{\text{SSN}\} & \rightarrow & \{\text{FLT\_ID}\} \end{array}$$

   imply

$$\{\text{SSN}\} \rightarrow \{\text{NAME, FLT\_ID}\}.$$

6. **Pseudotransitive rule**. *If $X \rightarrow Y$ and $WY \rightarrow Z$ then $WX \rightarrow Z$.*
   e.g. consider the relation

   OVERTIME (NAME, SSN, RANK, FLT_ID, START_APT, END_APT, BONUS)

   The FD's

$$\begin{array}{lcl} \{\text{FLT\_ID}\} & \rightarrow & \{\text{START\_APT, END\_APT}\} \\ \{\text{RANK, START\_APT, END\_APT}\} & \rightarrow & \{\text{BONUS}\} \end{array}$$

   imply

$$\{\text{RANK, FLT\_ID}\} \rightarrow \{\text{BONUS}\}.$$

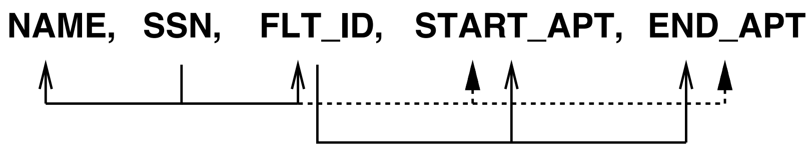
- One can use Rules 1-6 to determine F^+.
- It turns out that Rules 1-3 are sufficient to completely determine F^+.
  Rules 1-3 are called **Armstrong's Rules** in honor of the person who proved this result.
- *Equivalent FD sets.*
  - For most FD-sets F, the closure F^+ is probably quite large.
  - While we are interested in the *theoretical* implications of F^+ for any F, we are rarely going to *compute* F^+.
    $\Rightarrow$ Working with F turns out to be good enough.
  - **Definition**. FD-sets E and F are **equivalent** if $E^+ = F^+$, i.e., if their closures are equal.
  - If E is a set of FD's smaller than F and yet $E^+ = F^+$, then it is easier to work with E.
  - If $E^+ = F^+$ we also say that E **covers** F or is a **cover for** F.

6.6 Attribute Set Closures

- If X is an attribute set, we are often interested in *all* attributes (functionally) determined by X.
  i.e., what is the largest Y for which $X \to Y$?

  For example, consider

  EMP (NAME, SSN, FLT_ID, START_APT, END_APT)



  Here, SSN determines all the other attributes.
  $\Rightarrow$ {SSN}$^+$ = {NAME,SSN,FLT_ID,START_APT,END_APT}

- **Definition**. If F is a set of FD's and X is a set of attributes, then X^+ is the **closure of** X **under** F if X^+ is the largest set of attributes functionally determined by X using inference rules on FD's in F.

- Algorithm for determining X^+.

```
Algorithm:    ATTRIBUTE-SET-CLOSURE ($X$, $F$)

Input: Attribute set $X$, FD set $F$.
Output: Closure of $X$, $X^+$.
  1.  $X^+ := X$;
  2.  repeat
  3.    old_$X^+ := X^+$;
  4.    for each FD $Y \to Z$ in $F$ do
  5.      if $Y \subseteq X^+$ then
  6.        $X^+ := X^+ \cup Z$;
  7.  until old_$X^+ = X^+$;
  8.  return $X^+$;
```

For example, suppose F is:

{SSN}	$\to$	{NAME}
{FLT_ID}	$\to$	{START_APT, END_APT}
{SSN}	$\to$	{FLT_ID}

and suppose we want the closure of X={SSN, NAME} under F.

– Initially, X^+ := {SSN, NAME}.

– In the first iteration of the outerloop, in line 3 old_X^+ = {SSN, NAME}.

– Then, the FD {FLT_ID} $\to$ {START_ID, END_APT} is processed in the **for**-loop.
  $\Rightarrow$ since {FLT_ID} is not in X^+, it is ignored.

– Then, the FD {SSN} $\to$ {FLT_ID} is processed
  $\Rightarrow$ it results in X^+ = {SSN, NAME, FLT_ID}.

– After the first iteration of the **repeat**-loop, X^+={SSN, NAME, FLT_ID} and old_X^+={SSN, NAME}
  $\Rightarrow$ must continue.

- In the second iteration, the FD {FLT_ID} $\rightarrow$ {START_ID, END_APT} is processed in the **for**-loop.
  $\Rightarrow$ it results in X^+ = {SSN, NAME, FLT_ID, START_APT, END_APT}.
- After the second iteration of the **repeat**-loop, X^+={SSN, NAME, FLT_ID, START_APT, END_APT} and old_X^+={SSN, NAME, FLT_ID}
  $\Rightarrow$ must continue.
- No changes in third iteration
  $\Rightarrow$ stop.

Finally, X^+ = {SSN, NAME, FLT_ID, START_APT, END_APT}.

6.7 Normal Forms

- Normal forms are properties of *relations*.
- There are many normal forms: First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-Codd Normal Form (BCNF), etc.
- We say a relation is **in** xNF if its attributes satisfy certain properties (via their FD's).
- Generally, these properties are desirable.
- For example, if we desire the 3NF for a relational database:
  - We will test the relations in the database to see which are in 3NF.
  - Those that are not in 3NF will be decomposed into smaller relations (smaller in numbers of attributes) until we have each relation satisfying the 3NF properties.
- In the real world, most people try to achieve at least 3NF.

  It is slightly better to achieve BCNF (Boyce-Codd Normal Form), but 3NF is considered 'not bad'.
- The end result is: if a database is in BCNF (or 3NF), many anomalies are avoided.
- FD's were designed to test for Normal Forms.
- It is easier to understand normal forms by first considering a simpler version – *normal forms for primary keys.*
- Recall some definitions and notation:

– A **prime attribute** – an attribute belonging to *some candidate key* (not necessarily the primary key).
– A **nonprime attribute** – not belonging to any candidate key.

6.8 1NF: First Normal Form

- **Definition**. A relation is in 1NF if:
    1. the value of any attribute in any tuple is a single value, and
    2. domains of attributes contain only atomic values.
- Example of relations *not* satisfying 1NF:

multiple−valued attribute

PASSENGER	NAME	SSN	FLT_ID	MILES
	Smith	111–22–3333	17	40000
	Jones	222–33–4444	{12, 53, 119}	64000
	Brown	333–44–5555	27	575

⇒ Does not satisfy first part of definition.

Nested structure

PASSENGER	NAME	SSN	FLT_ID	START–APT
	Smith	111–22–3333	17	National
	Jones	222–33–4444	12	JFK
			53	JFK
			119	National
	Brown	333–44–5555	27	Logan

⇒ Does not satisfy second part of definition (contains nested relations).

- It is easy to transform the above relations to satisfy 1NF by:
  1. adding tuples or
  2. creating new relations

  For the first example:

PASSENGER	NAME	SSN	FLT_ID	MILES
	Smith	111–22–3333	17	40000
	Jones	222–33–4444	12	64000
	Jones	222–33–4444	53	64000
	Jones	222–33–4444	119	64000
	Brown	333–44–5555	27	575

Added tuples

  For the second example:

PASSENGER	NAME	SSN	FLT_ID
	Smith	111–22–3333	17
	Jones	222–33–4444	12
	Jones	222–33–4444	53
	Jones	222–33–4444	119
	Brown	333–44–5555	27

FLIGHT	FLT_ID	START–APT
	17	National
	12	JFK
	53	JFK
	119	National
	27	Logan

- 1NF is nowadays taken for granted (i.e., later formulations of relational theory assume relations to be in 1NF).
  $\Rightarrow$ we will assume all relations are in 1NF.
- Sometimes, raw data not in 1NF is called *unnormalized* data.

6.9 2NF: Second Normal Form

- **Definition**. A FD $X \rightarrow Y$ is a **partial dependency** if there exists an attribute $A \in X$ such that

  $$X - A \rightarrow Y.$$

  We say that Y is **partially dependent** on X.

  Example: consider the following relation with primary key {SSN, FLT_ID}

  OVERTIME (NAME, $\underline{\text{SSN, FLT\_ID}}$, BONUS)

  and suppose that bonuses are based on flight duration, and on the crew member's rank and salary.

  Here we can identify some FD's such as

  $$\begin{array}{lcl} \{\text{SSN, FLT\_ID}\} & \rightarrow & \{\text{BONUS}\} \\ \{\text{SSN}\} & \rightarrow & \{\text{NAME}\} \end{array}$$

  Now, neither one of

  $$\begin{array}{lcl} \{\text{SSN}\} & \rightarrow & \{\text{BONUS}\} \\ \{\text{FLT\_ID}\} & \rightarrow & \{\text{BONUS}\} \end{array}$$

  is true.
  Thus, {SSN, FLT_ID} $\rightarrow$ {BONUS} is not a partial dependency.

  But, {SSN} $\rightarrow$ {NAME}
  $\Rightarrow$ NAME is partially dependent on the primary key {SSN,FLT_ID}
  $\Rightarrow$ {SSN, FLT_ID} $\rightarrow$ {NAME} is a partial dependency.

- **Definition**. A relation schema is in 2NF if *no* nonprime attribute is partially dependent on the primary key
  (in other words, depends on part of the primary key).

- Thus, in the above example, OVERTIME is not in 2NF.
  (It is, however, in 1NF).

- Why is this a problem?
  Suppose an instance looks like:

OVERTIME	NAME	SSN	FLT_ID	BONUS)
	Smith	111-22-3333	17	450
	Jones	222-33-4444	28	375

Suppose we want to insert the tuple

<Brown, 111-22-3333, 18, 950>.

Because we have the FD: $\{\text{SSN}\} \rightarrow \{\text{NAME}\}$ we will have to check that

<Brown, 111-22-3333,...>

is valid, i.e., that it matches other SSN,NAME values for SSN=111-22-3333
⇒ we have to scan the whole relation (worst-case) to check
⇒ *insertion anomaly.*

Note that

<Brown, 111-22-3333,...>

is not valid.

Similarly, if Flight # 28 is deleted
⇒ we will lose the information "222-33-4444 is the SSN of Jones"
⇒ *deletion anomaly.*

This relation also has a *modification anomaly*:
⇒ if Smith changes name to Brown
⇒ have to propagate change to all relevant tuples.

To solve the problem, consider the decomposition of

OVERTIME (NAME, <u>SSN, FLT_ID</u>, BONUS)

into

OVERTIME (SSN, FLT_ID, BONUS)
PERSONAL_INFO (SSN, NAME)

These relations are in 2NF, with the important FD's:

$$\begin{array}{lcl} \{\text{SSN, FLT\_ID}\} & \rightarrow & \{\text{BONUS}\} \\ \{\text{SSN}\} & \rightarrow & \{\text{NAME}\} \end{array}$$

There are no partial dependencies of nonprime attributes on primary keys
$\Rightarrow$ the new set of relations is in 2NF.

6.10 3NF: Third Normal Form

- **Definition**. The FD $X \to Y$ is a **transitive dependency** in relation R if there exists a set of attributes Z in R such that

  1. $X \to Z$ and $Z \to Y$
  2. Z is not a subset of any key of R

  Example: consider the relation

  EMP (NAME, <u>SSN</u>, POSITION, DEPTNO, DNAME, MGRSSN).

  – Observe that EMP is in 2NF since no partial dependencies exist at all (and hence partial dependencies on the primary key don't exist).

  – Next, consider these FD's (there are others):

  $$\begin{array}{lcl} \{\text{SSN}\} & \to & \{\text{DEPTNO}\} \\ \{\text{DEPTNO}\} & \to & \{\text{MGRSSN}\} \end{array}$$

  Note that DEPTNO is not part of any key.
  $\Rightarrow$ there is a transitive dependency of MGRSSN on SSN.

- **Definition**. A relation R is in 3NF if

  1. it is in 2NF and,
  2. no nonprime attribute is transitively dependent on the primary key.

  In the above example:

  – EMP is in 2NF

  – MGRSSN is a nonprime attribute transitively dependent on the primary key SSN.

  $\Rightarrow$ EMP is *not* in 3NF.

- Why should we care about 3NF?

  Consider the following instance of EMP:

EMP	NAME	SSN	POSITION	DEPTNO	DNAME	MGRSSN
	Smith	...	...	5	Crew	111-22-3333
	Jones	...	...	6	Ticketing	222-33-4444

  Suppose we insert the tuple <Brown,...,5, Security, 333-44-5555>
  ⇒ we would have to check that the DEPTNO matches the MGRSSN (wrong in this case)
  ⇒ scanning the database
  ⇒ *insertion anomaly.*

  Similarly, if Smith's DEPTNO changes to 6
  ⇒ we will have to also insert the correct MGRSSN in Smith's tuple
  ⇒ *modification anomaly.*

  A deletion anomaly also occurs, if we delete Smith's tuple and Smith is the last Crew employee
  ⇒ we will lose the information "Dept# 6 is Crew"
  ⇒ *deletion anomaly.*

- To solve a 3NF problem, we can decompose relations.

  In the above example:

  E1 (NAME, <u>SSN</u>, POSITION, DEPTNO)
  E2 (<u>DEPTNO</u>, DNAME, MGRSSN)

  Note:

  1. E1 and E2 are in 3NF.
  2. EMP can be recovered by *joining* E1 and E2.
  3. The join will not create spurious tuples.

6.11 General Definitions of 2NF and 3NF

- We have defined 2NF and 3NF using primary keys.
- General definitions based on any candidate key are desirable.
- 2NF:
  - *Primary key version*: A relation schema R is in 2NF if every nonprime attribute A in R is not partially dependent on the primary key.
  - *General version*: A relation schema R is in 2NF if every nonprime attribute A in R is not partially dependent on any key of R.

  Consider the following example:

  AIRCRAFT_PARTS (MANUF, CODE, PART_ID, DESCR, URL, PRICE).

  Here,

  - The airline keeps information about aircraft parts.
  - PART_ID is the primary key
    $\Rightarrow$ it is a a unique number assigned by the airline to each part.
  - Each part is manufactured by a single manufacturer (MANUF) and the manufacturer uses a code (CODE) to identify the part
    $\Rightarrow$ the combination {MANUF, CODE} is a key.
  - The URL is the (internet) address of the manufacturer's webpage.

  Now, each manufacturer has a web page
  $\Rightarrow$ we have the FD {MANUF} $\rightarrow$ {URL}
  $\Rightarrow$ a dependency from part of a key to something else.

  Then, AIRCRAFT_PARTS is in 2NF according to the primary version of the definition (no partial dependency on the primary key).

But the partial dependency on MANUF causes the general definition to fail
$\Rightarrow$ AIRCRAFT_PARTS is not in 2NF.

- We *want* the general definition to hold, because otherwise we will have to check the FD {MANUF} $\rightarrow$ {URL} for every insertion.

  We can decompose

  AIRCRAFT_PARTS (MANUF, CODE, <u>PART_ID</u>, DESCR, URL, PRICE).

  into

  PARTS (MANUF, CODE, <u>PART_ID</u>, DESCR, PRICE)
  WEBSITES (MANUF, URL)

  This schema is in 2NF.

- Thus we see how our elaborate definitions of normal forms helps us catch problems in seemingly innocuous schemas (like AIRCRAFT_PARTS).

- 3NF:
  - *Primary key version*: A relation R is in 3NF if
    1. it is in 2NF, and
    2. no nonprime attribute is transitively dependent on the <u>primary key</u> of R.
  - *General version*: A relation R is in 3NF if
    1. it is in 2NF, and
    2. no nonprime attribute is transitively dependent on <u>any key</u> of R.

  Now observe this:

  - If $X \subseteq \{A_1, \ldots, A_n\}$ is any key then $X \rightarrow Y$ for any $Y \subseteq \{A_1, \ldots, A_n\}$
  - Suppose for some relation R
    1. X is the primary key.

2. Y is some other key.
3. Z is *transitively dependent* on Y, i.e., there are FD's

$$Y \to W \text{ and } W \to Z.$$

But $X \to W$ since X is a key.
$\Rightarrow$ Z is transitively dependent on X (the primary key)
$\Rightarrow$ the two 3NF definitions are identical.

For example, consider

AIRCRAFT_PARTS (MANUF, CODE, <u>PART_ID</u>, EMAIL, URL, PRICE).

Then, the FD {EMAIL} $\to$ {URL} may be a reasonable choice
$\Rightarrow$ there is a transitive dependency PART_ID $\to$ {EMAIL} $\to$ {URL}.
But, since {MANUF, CODE} is a key
$\Rightarrow$ {MANUF, CODE} $\to$ {PART_ID} trivially
$\Rightarrow$ {MANUF, CODE} $\to$ {EMAIL} $\to$ {URL}
$\Rightarrow$ transitive dependency on a nonprimary key.

6.12 BCNF: Boyce-Codd Normal Form

- Consider the relation

  $$\text{PARTS}\ (\underline{\text{MANUF, CODE}}, \text{URL}).$$

  Suppose that each manufacturer has a webpage form that depends on the part being ordered
  $\Rightarrow$ a different URL for each part.

  We can identify the natural FD:

  $$\{\text{MANUF, CODE}\} \rightarrow \{\text{URL}\}.$$

  Note that we also have the FD:

  $$\{\text{URL}\} \rightarrow \{\text{MANUF}\}$$

  since a given URL can only correspond to a unique manufacturer.

  Is PARTS in 2NF?

  – Recall: no nonprime attribute should have a partial dependence on a key.
  – Here, a manufacturer has different URL's
    $\Rightarrow$ no dependence of URL on MANUF.

  $\Rightarrow$ it passes the 2NF test

  Is PARTS in 3NF?

  – It is in 2NF.
  – Recall: we should not have any transitive dependence on a key.
  – Now, {MANUF, CODE} is the only key
    $\Rightarrow$ URL is the only attribute left
    $\Rightarrow$ can't have a transitive dependency with only one nonprime attribute.

  $\Rightarrow$ PARTS is in 3NF.

- So, what is the problem?

  Unfortunately, PARTS (MANUF, CODE, URL) has all the anomalies (insertion, deletion and modification).

  - The FD {URL} $\rightarrow$ {MANUF} is the real problem.
  - Suppose, we delete the tuple

    <Boeing, 3395, http://www.boeing.com/parts/737/wing>.

    If this is the only Boeing tuple in the relation, we will lose Boeing's URL
    $\Rightarrow$ *deletion anomaly.*

  It is easy to check that insertion and modification anomalies are also present.

- The problem appears to be:
  - In 2NF: we did not allow FD's *from* parts of keys *to* nonprime attributes.
  - Here we have an FD *from* an attribute *to* part of a key.

- One option is to introduce the following rule for every relation:
  1. it should be in 3NF
  2. there should be no FD $X \rightarrow Y$ such that Y is part of a key.

  Unfortunately, this rule is too *restrictive.*
  e.g., consider

  AIRCRAFT_PARTS (MANUF, CODE, PART_ID, DESCR, PRICE).

  Here,

  - PART_ID is the primary key.
  - {MANUF, CODE} is another key.
  - The FD {PART_ID} $\rightarrow$ {MANUF} follows.

Thus, if we disallow relations of this sort, we will be essentially barring all non-primary keys from having multiple attributes.
⇒ may be too restrictive in practice.

- Let us try to soften the rule:

  1. it should be in 3NF
  2. for every FD $X \to Y$, such that Y is part of a key, X should itself be a key.

  That is, we do allow *part of keys* to be dependent on things – provided those things are keys.

  This is a reasonable assumption because:

  – If X is a key, we would likely have to check uniqueness anyway (and that's all we have to do – using the **unique** keyword in SQL).

  – Deletion causes less of an anomaly, e.g., in

  AIRCRAFT_PARTS (MANUF, CODE, PART_ID, DESCR, PRICE).

  the FD {PART_ID} $\to$ {MANUF} is not important.
  Deleting the only tuple with 'Boeing', e.g.,

  <Boeing, 423, 12, Coat-rack, $50>,

  we lose the information "Part_id 12 is made by Boeing".
  But, if we delete the tuple

  <Boeing, 423, Coat-rack, $50>

  we are really saying "Boeing is the only company who makes the coat-racks we use, and we don't need coat-racks"
  ⇒ it's OK to lose "Part_id 12 is made by Boeing".

- The softened rule above needs a small modification:
  Observe that in the above example, we have the FD

  {PART_ID} $\to$ {MANUF}.

This passes our new test.

However, the following is also an FD:

$$\{\text{PART\_ID, PRICE}\} \rightarrow \{\text{MANUF}\}.$$

This fails the test because {PART_ID,PRICE} is not a key.

However, it is a *superkey* (contains a key).

- Final form:
  Definition. A relation R is in Boyce-Codd Normal Form (BCNF) if:
  1. it is in 3NF
  2. for every FD $X \rightarrow Y$ such that Y is part of a key, X is a superkey.

6.13 3NF and BCNF: An Alternate Definition

- First recall the 3NF definition:
  1. 2NF
  2. no transitive dependency from a key to a nonprime attribute should exist.

  Here, a transitive dependency means:

  - $X \rightarrow Y$ and $Y \rightarrow Z$
  - Y is <u>not</u> part of any key
  - X is a key
  - Z is a nonprime attribute

  Now, since X is a key, the FD $X \rightarrow Y$ must be true for any Y.

  Thus, the condition is really saying (given X is a key) that for $Y \rightarrow Z$:

  - (a) Y is not part of any key
  - (b) Z is a nonprime attribute

  Next, recall that 2NF is essentially:

  - if $Y \rightarrow Z$ then Y cannot be a proper subset of a key.

  Combine this with the first item (a) in 3NF and write (b) separately:
  A relation R is in 3NF if for every FD $Y \rightarrow Z$ either

  1. Y is a superkey, or
  2. Z is a prime attribute.

- This is an alternate definition of 3NF which does not mention 2NF.

- Note that if Z is a prime attribute, we allow $Y \to Z$ even if Y is not a superkey, e.g. in

  PART (MANUF, CODE, URL)

  we allowed {URL} $\to$ {MANUF} because {MANUF} is a prime attribute (it is part of the key {MANUF, CODE}).

  But BCNF does not allow this.

  Hence, an alternate definition of BCNF is:
  A relation R is in BCNF if for every FD $Y \to Z$, Y is a superkey of R.

  - This definition does not use 3NF.

- NOTE:
  - We must be careful to rule out trivial dependencies from consideration:
    * The dependency $X \to A$ where attribute $A \in X$ is called a *trivial dependency.*
    * Example: {SSN, NAME} $\to$ {SSN}.
    * We rule out trivial dependencies because they occur with any subset of attributes.
  - Suppose $Y = \{A_1, ..., A_k\}$ is a subset of attributes and $X \to Y$.
    * We know that $X \to A_i$ for $i = 1, ..., k$ by the decomposition rule.
    * Thus, the BCNF definition can also be stated as: *a relation is in BCNF if every nontrivial dependency is one in which a superkey determines an attribute.*

- Informally, the only functional dependencies in a BCNF go from keys to other attributes.
- Example:
  - Suppose X is a superkey and $X \to A$ in a BCNF relation, R.
  - Suppose B is some other attribute.
  - Consider the following tuples:

R	X	A	B
	x	a	b
	x	a	?

  - The value $X = x$ determines the value $A = a$.
  - But could we have different B-values?
  - Different B-values raise the familiar problem of $X \to A$ anomalies.
  - But since $R \in BCNF$, X is a superkey and so $X \to B$ (simply by being a superkey).
  - Thus, the B-value must be b.
  - Since we can't have duplicate tuples, it won't be allowed.

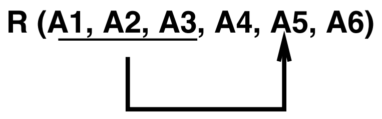
- Finally, observe that

$$\begin{aligned} R \text{ is in BCNF} &\Rightarrow R \text{ is in 3NF} \\ &\Rightarrow R \text{ is in 2NF} \\ &\Rightarrow R \text{ is in 1NF} \end{aligned}$$

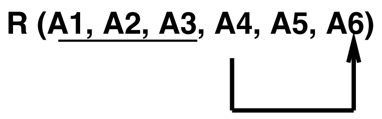
6.14 Another View of Normal Forms

- If the discussion so far has been confusing, let us try to explain normal forms a little differently.
- First, some simplifications:
  - Let us only consider relations with a single key – a primary key.
  - Assume this key has several attributes.

  Note: this simplification is only for conveying the key idea behind normal forms. In practice you would have to use the full definition.
- Let $R(A_1, \ldots, A_6)$ be a relation with primary key $\{A_1, A_2, A_3\}$.
- 2NF says: FD's like $A_2 \rightarrow A_5$ are not allowed.
  $\Rightarrow$ a proper subset of a key should not be on the left side of an FD:

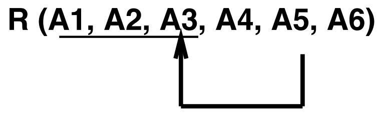


- 3NF says:
  1. at least 2NF, and
  2. FD's like $A_4 \rightarrow A_6$ are not allowed.
     $\Rightarrow$ an FD between non-key attributes is not allowed:



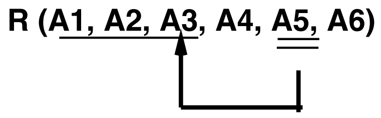
- Next, BCNF:

  Unfortunately 3NF allows an FD like $A_5 \rightarrow A_3$, where A_5 is nonprime and A_3 is part of the key:



  We saw why this was a problem in the BCNF example.

  On the other hand we did not want to be too restrictive: if A_5 happened to be a key we would allow it:



- The key ideas above are generalized to allow for:
  - multiple keys in a relation
  - keys consisting of groups of attributes.

6.15 Decomposition and its Problems

- We have seen that it is desirable to have relations in BCNF (or at least 3NF).
- We have seen how to *test* for BCNF and 3NF.
- But how do we create a BCNF (or 3NF) database?
  - One approach: Ad-hoc
    - Create relations intuitively
    - Test each for BCNF
  - More formal approach:
    - Start with a single large relation with all attributes
    - Systematically decompose relations not in BCNF
    - Repeat until all relations are in BCNF
- Unfortunately, decomposition can create problems:
  - Dependencies may be *lost* after decomposition.
  - Joins of decomposed relations may create *spurious tuples.*
  - Joins of decomposed relations may *lose tuples.*
- Note: thus far, we have identified problems with *individual* relations
  $\Rightarrow$ we have not placed constraints *among* multiple relations.

6.16 Dependency Preservation

- Suppose we decompose $R = (A_1, \ldots, A_n)$ into relations $R_1, \ldots, R_m$.
- Of course, we should have **attribute preservation**, i.e., attributes should not be lost in the shuffle:

$$R_1 \cup R_2 \cup \ldots R_m = R$$

- Unfortunately, FD's can be lost, e.g.,
  - Suppose F is a set of FD's containing the dependency
    {PART_ID} $\rightarrow$ {PRICE}.
  - Suppose also that {PART_ID} is put in relation R_3 and {PRICE} is put in relation R_7
    $\Rightarrow$ we can't check the dependency.
- In the above example, the lost FD would be OK, if the dependency were somehow not important
  $\Rightarrow$ we need to consider the *closure* of FD's.
- **Definition**. Let F be a set of FD's and suppose R is decomposed into relations $R_1, \ldots, R_m$. Let E be the set of FD's in $R_1, \ldots, R_m$. Then the decomposition is **dependency preserving** if $E^+ = F^+$.
- **Fact**: It is always possible to decompose any relation R into 3NF relations $R_1, \ldots, R_m$ such that the decomposition is dependency preserving.

6.17 Nonadditive and Lossless Decompositions

- Suppose we decompose R into $R_1, \ldots, R_m$. Later, we wish to recover R (perhaps as a view).
  - The natural join on $R_1, \ldots, R_m$ should return R.
  - If we're not careful, this join can create spurious tuples

    e.g, consider the relation r with schema R =(CAR,OWNER,COLOR):

CAR	OWNER	COLOR
Toyota	Smith	blue
Ford	Jones	blue

    Suppose we decompose this into r_1 and r_2 with schemas R_1=(CAR,COLOR) and R_2=(OWNER,COLOR).

    How? Let $r_1 = \Pi_{\text{CAR,COLOR}}(r)$ and $r_2 = \Pi_{\text{OWNER,COLOR}}(r)$:

CAR	COLOR
Toyota	blue
Ford	blue

OWNER	COLOR
Smith	blue
Jones	blue

  What happens when we join r_1 and r_2?

CAR	OWNER	COLOR
Toyota	Smith	blue
Toyota	Jones	blue *
Ford	Smith	blue *
Ford	Jones	blue

  $\Rightarrow$ Spurious tuples!

- A decomposition of R into $R_1, \ldots, R_m$ is **nonadditive** if for every instance r of R, the natural join of the corresponding instances $r_1, \ldots, r_m$ is equal to r, i.e.,

$$r_1 * r_2 * \ldots * r_m = r$$

where $r_i = \Pi_{R_i}(r)$.

- Note: **nonadditive** is the same as 'creates no spurious tuples'.
- Sometimes, one can inadvertently *lose* tuples in a join.
- Example:

  Suppose EMP (SSN, NAME, FLT_ID, DEPTNO) has too many NULLs in the DEPTNO attribute (because many employees have no assigned department).

EMP	SSN	NAME	FLT_ID	DEPTNO
	111-22-3333	Smith	12	NULL
	222-33-4444	Jones	55	6
	333-44-5555	Brown	119	NULL

$\Rightarrow$ One solution is to decompose EMP into two relations:

EMP1 (SSN, NAME, FLT_ID)
DEPT (SSN, DEPTNO)

e.g.,

EMP1	SSN	NAME	FLT_ID
	111-22-3333	Smith	12
	222-33-4444	Jones	55
	333-44-5555	Brown	119

DEPT	SSN	DEPTNO
	222-33-4444	6

Here,

- Only those employees assigned to a department will have department numbers
  $\Rightarrow$ DEPT is small.
- Both EMP1 and DEPT are in BCNF.
- The decomposition is nonadditive and dependency preserving.

Consider the join EMP1 $*$ DEPT:

EMP1 $*$ DEPT	SSN	NAME	FLT_ID	DEPTNO
	222-33-4444	Jones	55	6

$\Rightarrow$ the 'Smith' and 'Brown' tuples are lost!
$\Rightarrow$ we have to be careful in letting joins replace relations.

- We call a decomposition is *lossless* if it does not lose tuples in recovering the original relation.
- Note:
  - Nonadditivity and losslessness are two sides of the same coin.
  - We will use the term *lossless* to refer to both.
  - Some books use *additive* to refer to both.
- It would be useful, if given a decomposition, to test whether the decomposition is lossless.
- **A useful fact**. A decomposition of R into R_1 and R_2 is nonadditive with respect to a set of FD's F, if and only if either one of the FD's
  - $R_1 \cap R_2 \ \rightarrow R_1 - R_2$
  - $R_1 \cap R_2 \ \rightarrow R_2 - R_1$

  is in F^+.

  Intuition:
  - Observe: the attributes $R_1 \cap R_2$ are in both R_1 and R_2
    $\Rightarrow$ these are the join attributes.
  - Suppose the FD $R_1 \cap R_2 \ \rightarrow R_1 - R_2$ holds.
    This is the same as $R_1 \cap R_2 \ \rightarrow R_1 - (R_1 \cap R_2)$
    $\Rightarrow$ $R_1 \cap R_2$ is a *key* for R_1.
    $\Rightarrow$ weird, unwanted tuple combinations can't occur.
- In the R=(CAR,OWNER,COLOR) example:

We joined

CAR	COLOR
Toyota	blue
Ford	blue

OWNER	COLOR
Smith	blue
Jones	blue

to get

CAR	OWNER	COLOR
Toyota	Smith	blue
Toyota	Jones	blue *
Ford	Smith	blue *
Ford	Jones	blue

Note that {COLOR} is not a key for either relation.

Here, R_1=(CAR,COLOR) and R_2=(OWNER,COLOR) and,

$$\begin{aligned} R_1 \cap R_2 &= \text{COLOR} \\ R_1 - R_2 &= \text{CAR} \\ R_2 - R_1 &= \text{OWNER} \end{aligned}$$

Clearly, the neither of FD's

$$\begin{aligned} \{\text{COLOR}\} &\rightarrow \{\text{CAR}\} \\ \{\text{COLOR}\} &\rightarrow \{\text{OWNER}\} \end{aligned}$$

hold
$\Rightarrow$ the decomposition is *not* nonadditive.

6.18 Algorithms for Decomposition of Relations

- First, recall that an FD set E *covers* FD set F if $E^+ = F^+$, i.e., the closure of E is the closure of F
  $\Rightarrow$ if E is smaller it will be easier to work with
  $\Rightarrow$ it is useful to determine the *minimal cover* for an FD set F.

- *Minimal covers* can be defined in a number of ways:

  – E is an *FD-minimal cover* of F if E covers F and no other FD set covers F that has fewer FD's than E.

  – E is an *attribute-minimal cover* of F if E covers F and no other FD set covers F with fewer attributes.

  – E is a *left-minimal cover* of F if E covers F and no other FD set covers F with smaller left-hand-sides.

  Note:

  – Computing an attribute-minimal cover is a hard (NP-complete) problem
    $\Rightarrow$ no fast algorithm is known.

  – Computing FD-minimal covers and left-minimal covers is fairly straightforward.

  – Left-minimal covers are all that's needed for 3NF decompositions.

- Key ideas in finding a left-minimal cover:

  – Consider the relation

    EMP (NAME, SSN, FLT_ID, START_APT, END_APT).

    and the FD set F

(1)	{SSN, NAME}	→	{FLT_ID, START_APT}
(2)	{SSN}	→	{FLT_ID}
(3)	{FLT_ID}	→	{START_APT}
(4)	{SSN}	→	{NAME}
(5)	{SSN, NAME}	→	{FLT_ID}

Note that, given (2) and (4), we don't need (5) since the combination of (2) and (4) will imply (5).

– The first step is to break up the FD's so that the right-hand-sides are only single attributes:

{SSN, NAME}	→	{FLT_ID}
{SSN, NAME}	→	{START_APT}
{SSN}	→	{FLT_ID}
{FLT_ID}	→	{START_APT}
{SSN}	→	{NAME}

– Next, see if left-hand-side attributes can be removed.

  ∗ For example, consider the FD {SSN, NAME} → {FLT_ID}.

  ∗ The left-hand-side here is {SSN, NAME}.

  ∗ Suppose we remove SSN: {SSN, NAME} - {SSN}.

  ∗ Can we replace the earlier FD with {NAME} → {FLT_ID}?

  ∗ To check, we compute the attribute closure of the new left-hand-side, i.e., check whether ({SSN, NAME} - {SSN})$^+$ contains the right-hand-side $\{FLT\_ID\}$.

  ∗ In this case, $\{NAME\}^+$ does *not* contain {FLT_ID}
    ⇒ cannot remove {SSN} from {SSN, NAME} → {FLT_ID}.

- **Definition**: An FD $X \rightarrow Y$ where Y has more than one attribute is called a *multiple-RHS* FD.

- Algorithm for computing a minimal cover:

```
Algorithm:    LEFT-MIN-COVER ($F$)

Input: An FD set $F$.
Output: A left-minimal cover $E$.
  1.  $E$ := $F$;
  2.  for each multiple-RHS FD $X \rightarrow A_1 A_2 ... A_k$ in $E$
  3.     $E$ := $E - \{X \rightarrow A_1 A_2 ... A_k\}$;
  4.     for $i \leftarrow 1$ to $k$
  5.        $E$ := $E \cup \{X \rightarrow A_i\}$;
  6.  endfor
      // All multiple-RHS FD's have been replaced by single-RHS FD's
  7.  for each FD $X \rightarrow A$ in $E$
  8.     $X^+$ := ATTRIBUTE-SET-CLOSURE ($X$, $E - \{X \rightarrow A\}$);
  9.     if $A \in X^+$
  10.       $E$ := $E - \{X \rightarrow A\}$;
  11. endfor
      // Now, we are rid of unnecessary FD's. Next, reduce left-hand-sides
  12. for each FD $X \rightarrow A$ in $E$
  13.    for each attribute $B \in X$
  14.       $D$ := $E - \{X \rightarrow A\} \cup \{(X - B) \rightarrow A\}$;
  15.       $(X - B)^+$ := ATTRIBUTE-SET-CLOSURE ($X - B$, $D$);
  16.       if $A \in (X - B)^+$
  17.          $E$ := $E - \{X \rightarrow A\}$;
  18.          $E$ := $E \cup \{(X - B) \rightarrow A\}$;
  19.       endif
  20.    endfor
  21. return $E$;
```

- The following algorithm decomposes a relation R into a set of 3NF relations $R_1, R_2, ...$ that are dependency-preserving and nonadditive.

```
Algorithm:    3NF-DECOMPOSITION ($R$, $F$)

Input: Relation $R = (A_1, ..., A_k)$ with FD set $F$.
Output: 3NF decomposition $R_1, R_2, ...$
  1.   $E$ := LEFT-MIN-COVER ($F$);
  2.   for each left-hand-side $X_i$ in $E$
  3.       $R_i$ := $\{X_i\}$;
  4.       for each $X_i \rightarrow A_j$ in $F$
  5.           $R_i$ := $R_i \cup \{A_j\}$;
  6.   endfor
       // At this point we have a collection of relations $R_1, ..., R_n$
  7.   $R_{n+1}$ := $\emptyset$;
  8.   for each $A_i \notin R_1 \cup ... \cup R_n$
  9.       $R_{n+1}$ := $R_{n+1} \cup \{A_i\}$;
  10.  return $R_1, ..., R_{n+1}$;
```

- Why does this work?
  - First note that all the FD's find their way (see lines 4-5) into the decomposition
    $\Rightarrow$ it is dependency-preserving.
  - Are the resulting relations in 3NF?
    * Consider a transitive dependency $X \rightarrow Y \rightarrow Z$ in the original relation.
    * The FD $X \rightarrow Z$ will be removed in minimal cover since $X \rightarrow Y$ and $Y \rightarrow Z$ are sufficient to generate $Z \in Z^+$.
    * Thus, the decomposition (lines 4-5 above) will not create a relation with attributes (X, Y, Z) and thus transitive dependencies will be removed.
  - Is the decomposition nonadditive?
    * Note this general property: if R is a relation and $X \rightarrow A$ is an FD

then the decomposition $R - A$ and $R' = (X, A)$ is nonadditive. Why? Because a join of $R - A$ and R' only involves X and since X determines A, no spurious tuples will be created.

  * In the above algorithm, all decomposition steps are of the above type.

- An algorithm for decomposing a relation into a collection of *nonadditive BCNF* relations.

```
Algorithm:   NONADDITIVE-BCNF-DECOMPOSITION ($R$, $F$)

Input: Relation $R$, FD set $F$.
Output: A nonadditive BCNF decomposition $R_1, R_2, \ldots$
  1.  $R_1, ..., R_k$ := 3NF-DECOMPOSITION ($R$, $F$);
  2.  while $\exists$ $R_i \in R_1, ..., R_k$ not in BCNF
  3.     if $X \rightarrow Y$ is an FD in $R_i$ that violates BCNF
  4.        $R_i$ := $R_i - Y$;
  5.        $k$ := $k + 1$;
  6.        $R_k$ := $(X, Y)$;
  7.     endif
  8. return $R_1, ..., R_k$;
```

Intuition:

If $X \rightarrow Y$ is in some R_i and it violates BCNF
 $\Rightarrow$ X is not a superkey of R_i (definition of BCNF)
 $\Rightarrow$ we create a relation $R_k = X \cup Y$
Here $X \rightarrow Y$ implies X is a superkey for R_k
 $\Rightarrow$ since Y is removed from R_i, it does not cause the BCNF violation.

- Unfortunately, we can't always decompose R into BCNF relations that are *both* dependency preserving and nonadditive.

  – The nonadditive property is preserved by the above algorithm.

- It may produce a decomposition that is not dependency-preserving.
- In general, it is impossible to achieve both.
- Example: consider the relation

  PARTS ($\underline{\text{MANUF, CODE}}$, URL)

  where each part has a unique URL.

  * The FD's are:

{MANUF, CODE}	$\rightarrow$	{URL}	(unique URL for each part)
{URL}	$\rightarrow$	MANUF	(knowing a URL tells you the manufacturer)

  * The PARTS relation is not in BCNF since we have a dependency from an attribute (URL) to part of a key (MANUF).
  * Any decomposition will have to separate URL from MANUF.
  * This means the FD {MANUF, CODE} $\rightarrow$ {URL} cannot be preserved in the decomposition
    $\Rightarrow$ no BCNF decomposition of PARTS can be dependency-preserving.

6.19 Formal Schema Design: A Summary

- We saw that ad-hoc designs led to anomalies with insertions, deletions and modifications.
- To analyze relations, we developed the theory of normalization:
  - Definition of functional dependency (FD).
  - Properties of FD's.
  - Computation of attribute closures.
  - Definition of Normal forms (primary and general versions).
- In practice: try to achieve BCNF. If not possible, live with 3NF.

  If your design is not in 3NF
  $\Rightarrow$ you have a weird schema.
- Also need to check for *nonadditivity* and *dependency preservation*.
- Before using a join to replace existing relations, check to see tuples don't get lost.
- Sometimes, a BCNF decomposition or a 3NF decomposition can lead to inefficiencies
  $\Rightarrow$ many queries require expensive joins.
  In this case, one sometimes permits BCNF and 3NF violations for efficiency reasons
  $\Rightarrow$ violations can be checked separately at leisure.
- Some issues we have not covered:
  - Formal proofs asserting the correctness of algorithms.
  - Finding minimal FD-sets with other definitions of minimality.

– General mechanisms for testing nonadditivity (an algorithm called the Tableau Chase Method).
– Multivalued FD's and 4NF.
– Other dependencies and normal forms.