Ceng 302
Database Management Systems

Relational Database Design and Normalization

Prof. Dr. Adnan YAZICI
Department of Computer Engineering,
Middle East Technical University

(Fall 2021)

Design of 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 storage **base relations**
- What are the **criteria** for "good" base relations?

Design of Relational Databases

- 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)

Informal Design Guidelines for Good Relation Schemas

1. Semantics of the attributes: it should be easy to explain the meaning of the schema. If a schema correspond to one entity type or one relationship type, its meaning tends to be clear.
2. Reducing the redundant values in tuples: no anomalies.
3. Reducing the null values in tuples: nulls in exceptional cases only.
4. Disallowing the possibility of generating spurious tuples.

Semantics of the Relation Attributes

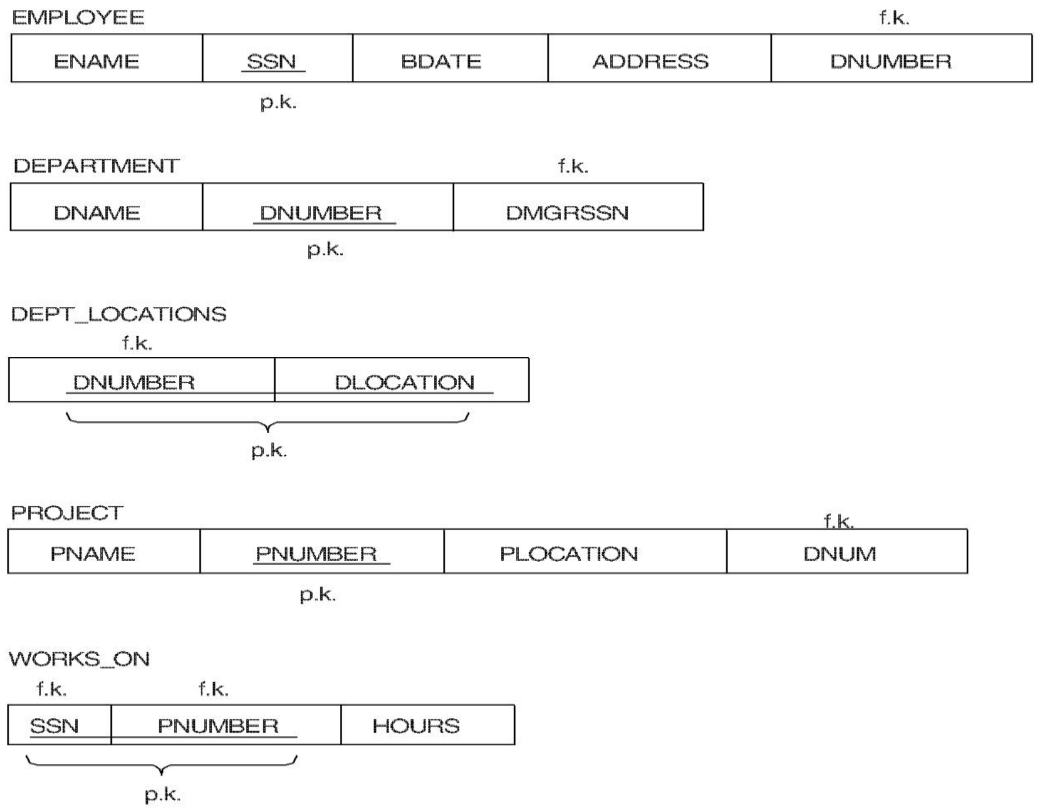
GUIDELINE 1: Informally, each **tuple** in a relation should represent **one entity** or **relationship instance.** (Applies to individual relations and their attributes).

- 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.

A simplified COMPANY relational database schema

Figure 14.1 Simplified version of the COMPANY relational database schema.



Good Database Design wrt Consistency and Anomalies

- no redundancy of *FACT* (!)
- no inconsistency
- no insertion, deletion or update anomalies
- no information loss
- no dependency loss

Redundant Information in Tuples and Update Anomalies

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.

- Mixing attributes of **multiple entities** may cause some problems.
- Information is stored **redundantly** wasting storage.
- Problems with update anomalies
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

Bad Database Design- fact clutter

FLIGHTS

flt#	date	airline	plane#
DL242	10/23/00	Delta	k-yo-33297
DL242	10/24/00	Delta	t-up-73356
DL242	10/25/00	Delta	o-ge-98722
AA121	10/24/00	American	p-rw-84663
AA121	10/25/00	American	q-yg-98237
AA411	10/22/00	American	h-fe-65748

- **insertion anomalies**: how do we represent that TK912 is flown by Turkish Airline without there being a date and a plane assigned.
- **deletion anomalies**: cancelling AA411 on 10/22/00 makes us lose that it is flown by American.
- **update anomalies**: if DL242 is flown by KLM, we must change it everywhere.

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 1000 employees working on project P1.

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:
  - attribute not applicable or invalid
  - attribute value unknown (may not exist)
  - value known to exist, but unavailable

Null Values

CUSTOMER

CUSTOMER#	NAME	MAIDEN NAME	DRAFT STATUS	Telephone
123-45-6789	Lisa Smith	Lisa Jones	**inapplicable**	**unknown**
234-56-7890	George Foreman	**inapplicable**	drafted	**ni**
345-67-8901	**unknown**	Mary Blake	**inapplicable**	**Inapplicaple**

- Null-value **unknown (unk)** reflects that the attribute does apply, but the value is currently unknown. That's ok!
- Null-value **inapplicable (dne)** indicates that the attribute does not apply.
- Null-value **no-information (ni)** results from a no information and not good in database design.

Spurious Tuples

- **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.

- Bad designs for a relational database may result in erroneous results (**spurious tuples**) for certain JOIN operations.
- The "lossless join" property is used to guarantee meaningful results for join operations.

Bad Database Design

- **information loss:** we polluted the database with false facts; we can't find the true facts.

FLIGHTS

flt#	date	airline	plane#
DL242	10/23/00	Delta	k-yo-33297
DL242	10/24/00	Delta	t-up-73356
DL242	10/25/00	Delta	o-ge-98722
AA121	10/24/00	American	p-rw-84663
AA121	10/25/00	American	q-yg-98237
AA411	10/22/00	American	h-fe-65748

FLIGHTS-AIRLINE

flt#	airline
DL242	Delta
AA121	American
AA411	American

DATE-AIRLINE-PLANE

date	airline	plane#
10/23/00	Delta	k-yo-33297
10/24/00	Delta	t-up-73356
10/25/00	Delta	o-ge-98722
10/24/00	American	p-rw-84663
10/25/00	American	q-yg-98237
10/22/00	American	h-fe-65748

Bad Database Design- information loss

FLIGHTS-AIRLINE

flt#	airline
DL242	Delta
AA121	American
AA411	American

DATE-AIRLINE-PLANE

date	airline	plane#
10/23/00	Delta	k-yo-33297
10/24/00	Delta	t-up-73356
10/25/00	Delta	o-ge-98722
10/24/00	American	p-rw-84663
10/25/00	American	q-yg-98237
10/22/00	American	h-fe-65748

FLIGHTS

flt#	date	airline	plane#
DL242	10/23/00	Delta	k-yo-33297
DL242	10/24/00	Delta	t-up-73356
DL242	10/25/00	Delta	o-ge-98722
AA121	10/24/00	American	p-rw-84663
AA121	10/25/00	American	q-yg-98237
AA211	*10/22/00*	*American*	*h-fe-65748*
AA411	*10/24/00*	*American*	*p-rw-84663*
AA411	*10/25/00*	*American*	*q-yg-98237*
AA411	10/22/00	American	h-fe-65748

Spurious Tuples (cont.)

- 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.

Normalization

FLIGHT-SCHEDULE

FLIGHT#	AIRLINE	WEEKDAYS	PRICE
101	delta	mo,fr	156
545	american	mo,we,fr	110
912	scandinavian	fr	450

unnormalized

FLIGHT-SCHEDULE

FLIGHT#	AIRLINE	WEEKDAY	PRICE
101	delta	mo	156
545	american	mo	110
912	scandinavian	fr	450
101	delta	fr	156
545	american	we	110
545	american	fr	110

redundant

FLIGHT-WEEKDAY

FLIGHT#	WEEKDAY
101	mo
545	mo
912	fr
101	fr
545	we
545	fr

just right!

FLIGHT-SCHEDULE

FLIGHT#	AIRLINE	PRICE
101	delta	156
545	american	110
912	scandinavian	450

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

Functional Dependencies (cont.)

An FD X -> Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y.

Defn: *For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], then t1[Y]=t2[Y]*

- X → Y in R specifies a *constraint* on all relation instances r(R)
- FDs are derived from the real-world constraints on the attributes

Examples of FD constraints (cont.)

- Social security number determines employee name

  SSN -> ENAME

- Project number determines project name and location

  PNUMBER -> {PNAME, PLOCATION}

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

  {SSN, PNUMBER} -> HOURS

Examples of FD constraints (cont.)

- 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])

Functional Dependencies and Keys

Definition: Suppose X and Y be sets of attributes subsets of R. A functional dependency between X and Y, denoted by X→Y, specifies a constraint on the possible tuples that can form a relation state r of R.

The constraint is that, for any two tuples t_1 and t_2 in r,

if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$ must also hold.

If $X \rightarrow Y$, Then

R

<u>X</u>	Y

- In another word, Y is **functionally dependent** on X in R **iff** for each $x \in R.X$ there is precisely one $y \in R.Y$.
- We use **keys** to enforce functional dependencies in relations.

How to Compute Meaning

- Armstrong's inference rules

Rules of the computation:

- reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
- Augmentation: if $X \rightarrow Y$, then $WX \rightarrow WY$
- Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Derived rules:

- Union: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Decomposition: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Pseudotransitivity: if $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Armstrong's Axioms:

- sound (generate only functional dependencies that actually hold)
- complete (generate all functional dependencies that hold).

How to Compute Meaning

- Armstrong's inference rules

- **Proof of reflexivity**: if $Y \subseteq X$, then $X \rightarrow Y$.

  Suppose $Y \subseteq X$ and two tuples t_1 and t_2 exist in some relation instance r of R such that $t_1[X] = t_2[X]$ (by defn.).

  Then $t_1[Y] = t_2[Y]$ must be true, because $Y \subseteq X$;

- Hence, $X \rightarrow Y$ must hold in r.

How to Compute Meaning
- Armstrong's inference rules

- **Proof of Augmentation**:

  $\{X \rightarrow Y\} \Rightarrow WX \rightarrow WY$

  Suppose $X \rightarrow Y$ holds in a relation instance r of R, but $WX \rightarrow WY$ does not hold.

  Then, there must exist two tuples t_1 and t_2 in r such that

  (1) $t_1[X] = t_2[X]$,

  (2) $t_1[Y] = t_2[Y]$,

  (3) $t_1[WX] = t_2[WX]$, and

  (4) $t_1[WY] \neq t_2[WY]$.

  This is not possible, since we can deduce from (1) and (3) that

  (5) $t_1[W] = t_2[W]$,

  and from (2) and (5) we deduce

  (6) $t_1[WY] = t_2[WY]$.

  Contradicting (4). So, $\{X \rightarrow Y\} \Rightarrow WX \rightarrow WY$.

How to Compute Meaning
- Armstrong's inference rules

Proof of transitive rule:

$\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z.$

Assume (1) $X \rightarrow Y$ and

(2) $Y \rightarrow Z$ both hold in a relation instance r of R.

Then, for any two tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$, we must have

(3) $t_1[Y] = t_2[Y]$ from assumption (1);

We must also have

(4) $t_1[Z] = t_2[Z]$ from (3) and assumption (2).

Hence $X \rightarrow Z$ must hold in r.

Proof of decomposition (or projection) rule:

$\{X \rightarrow YZ\} \Rightarrow X \rightarrow Y$ and $X \rightarrow Z$

1. $X \rightarrow YZ$ (given)
2. $YZ \rightarrow Y$ (using reflex. rule, $Y \subseteq YZ$)
3. $X \rightarrow Y$ (using transitivity rule)

How to Compute Meaning
- Armstrong's inference rules

- **Proof of union rule:** if $X \to Y$ and $X \to Z$, the $X \to YZ$
  1. $X \to Y$ (given)
  2. $X \to Z$ (given)
  3. $X \to XY$ (usig (1) and augmentation rule, notice that $XX=X$)
  4. $XY \to ZY$ (using (2) and augmentation with Y)
  5. $X \to YZ$ (use (3) and (4) and transitivity rule.)

How to Compute Meaning
- Armstrong's inference rules

- **Proof of pseudotransitive rule:**
  $\{X \rightarrow Y, WY \rightarrow Z\} \Rightarrow WX \rightarrow Z$

1. $X \rightarrow Y$ (given)
2. $WY \rightarrow Z$ (given)
3. $WX \rightarrow WY$ (usig (1) and augmentating W)
4. $WX \rightarrow Z$ (trans. on (3) and (2))

Inference Rules for FDs

- **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

Example:

FD: $a \rightarrow b$; $c \rightarrow \{d,e\}$; $\{a,c\} \rightarrow \{f\}$

$\{a\}^+ = \{a,b\}$

$\{c\}^+ = \{c,d,e\}$

$\{a,c\}^+=\{a,c,f,b,d,e\}$

How to Compute Meaning

when do sets of FDs mean the same?

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

$X^+ = X$;

Repeat

Old $X^+ = X^+$

For each FD, $Y \rightarrow Z$ in F do

If $X^+ \supseteq Y$, Then $X^+ = X^+ \cup Z$;

Until ($X^+ = \text{old } X^+$);

Example: {Ssn → Ename,

Pnumber → {Pname, Plocation}, {Ssn,Pnumber} → Hours}

$\{Ssn\}^+ = \{Ssn, Ename\}$

$\{Pnumber\}^+ = \{Pnumber, Pname, Plocation\}$

$\{Ssn, Pnumber\}^+ = \{Ssn, Pnumber, Ename, Pname, Plocation, Hours\}$

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 $F^+ = G^+$
- F **covers** G if every FD in G can be inferred from F (i.e., if G^+ <u>*subset-of*</u> F^+)
- F and G are **equivalent** if F covers G and G covers F.

Equivalence of Sets of FDs

- We can determine whether F covers E by calculating X^+ with respect to F for each FD $X \rightarrow Y$ in E; then checking whether this X^+ includes the attributes in Y.

- If this is the case for every FD in E, then F covers E.

Equivalence of Sets of FDs

Example:

Given: $F=\{a \rightarrow b;\ c \rightarrow \{d,e\};\ \{a,c\} \rightarrow \{f\}\}$

check if F covers $E=\{\{a,c\} \rightarrow \{d\}\}$

$\{a,c\}^+ = \{a,c,f,b,d,e,f\} \supseteq \{a,c,d\}$,

then F covers E

Finding a Key for a Relation

Algorithm: Finding a Key K for R, given a set of FDs

1. Set K = R.
2. For each attribute A in K {

   Compute $(K\text{-}A)^+$ wrt F;

   If $(K\text{-}A)^+$ contains all the attributes in R,

   Then set K = K – {A}

   };

Example: R = Ssn, Pnumber, Ename, Pname, Plocation, Hours

F = {Ssn → Ename, Pnumber → {Pname, Plocation}, {Ssn,Pnumber} → Hours}

The Key is {Ssn,Pnumber},

Since

$\{Ssn, Pnumber\}^+$ = {Ssn, Pnumber, Ename, Pname, Plocation, Hours}

Examples:

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C , CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG\ (A \rightarrow B \text{ and } A \rightarrow C)$
  3. $result = ABCGH\ \ (CG \rightarrow H$ and $CG \subseteq ABCG)$
  4. $result = ABCGHI\ (CG \rightarrow I$ and $CG \subseteq ABCGH)$
- Is AG a candidate key?
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

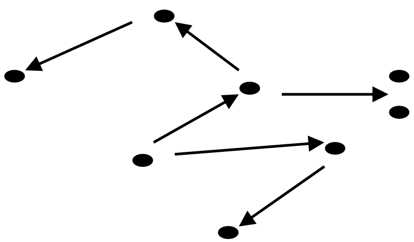
Use of Attribute Closure

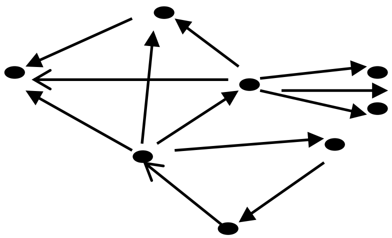
There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute $\alpha^{+,}$ and check if α^{+} contains all attributes of R.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^{+}), just check if $\beta \subseteq \alpha^{+}$.
  - Is a simple and cheap test, and very useful.
- Computing closure of F, that is F^{+}
  - For each $\gamma \subseteq R$, we find the closure γ^{+}, and for each $S \subseteq \gamma^{+}$, we output a FD $\gamma \rightarrow S$.

How to Compute Meaning
-the meaning of a set of FDs, F+

- The set of all FDs implied by a given set F of FDs is called the closure of F, F+.
- Given the ribs of an umbrella, the FDs, what does the whole umbrella, F^+, look like this.





- Determine each set of attributes, X, that appears on a left-hand side of a FD. Determine the set, X^+, the closure of X under F.

Procedure for Computing F^+

- To compute the closure of a set of FDs F:

Algorithm: Computing F^+

$F^+ = F$
repeat
    for each FD f in F^+ apply reflexivity and augmentation rules on f, add the resulting FDs to F^+
    for each pair of FDs f_1 and f_2 in F^+
        if f_1 and f_2 can be combined using transitivity
        then add the resulting FDs to F^+
until F^+ does not change any further

Example: Find F^+, If $F = \{AB \rightarrow C, A \rightarrow D, D \rightarrow E\}$

$F =$

$AB \rightarrow C$

union $AB \rightarrow BCD$

$A \rightarrow D$ aug $AB \rightarrow BD$

trans $AB \rightarrow BCDE$, $AB \rightarrow CDE$

decomp

$D \rightarrow E$ aug $BCD \rightarrow BCDE$

Thus,

$\{AB \rightarrow BD, AB \rightarrow BCD, AB \rightarrow BCDE, AB \rightarrow CDE\} \in F+$

$F+ = \{F, AB \rightarrow BD, AB \rightarrow BCD, AB \rightarrow BCDE, AB \rightarrow CDE$, trivial FDs$\}$

Example: Closure, F+

Example: Contracts (contractid, supplierid, projid, deptid, partid, qty, value).

- We denote the schema for Contracts as CSJDPQV. The meaning of a tuple is that the contract with contractid C is an aggrement that supplier S (supplierid) will suply Q items of part P (partid) to project J (projectid) associated with department D (deptid); the value V of this contract id equal to value.

- The following ICs are known to hold:

  F = {C → CSJDPQV, JP → C, SD → P}

How to Compute Meaning
- minimal cover of a set of FDs

Is there a minimal set of ribs that will hold the umbrella open?

F is minimal if:

1. every dependency in F has a single attribute as right-hand side.
2. we can't replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$ where $Y \subset X$ and still have a set of dependencies equivalent with F. This ensures that there are no redundancies by having redundant attributes on the **left-hand side** of a dependency.
3. we can't remove any dependency from F and still have a set of dependencies equivalent with F. This ensures that there are no redundancies by having dependency that can be inferred from the remaining FDs in F.

Algorithm: Finding Minimal Cover F for a set of FDs.

1. Put the FDs in a standard Form: Obtain a collection G of equivalent FDs with a single attribute on the right side (using the decomposition axiom)

That is; replace each FD $X \rightarrow \{A_1,\ldots, A_n\}$ in F by the FDs, such as $X \rightarrow A_1$, $X \rightarrow A_2,\ldots, X \rightarrow A_n$.

2. Minimize the left side of each FD: For each FD, check each attribute in the left side to see if it can be deleted while preserving equivalence to F.

For each FD, $X \rightarrow A$ in F

For each attribute $B \in X$,

If $((F - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\}) \equiv F$

Then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.

Ex: If $GCD \rightarrow A$ and $G \rightarrow C$ in F, Then $((F-\{GCD \rightarrow A\}) \cup \{(GCD-C) \rightarrow A\}) \equiv F$, Then replace $GCD \rightarrow A$ with $GD \rightarrow A$. That is, C is redundant.

3. Delete redundant FDs: That is; for each remaining FD, $X \rightarrow A$ in F

If $(F - \{X \rightarrow A\}) \equiv F$, then remove $X \rightarrow A$ from F.

How to Compute Meaning
- minimal cover of a set of FDs

Example: **Finding Minimal Cover F for a set of FDs.**

$F = \{A \rightarrow B,\ ABCD \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow H,\ ACDF \rightarrow EG\}.$

- Let us rewrite $ACDF \rightarrow EG$ so that every right side is a single attribute:

  $ACDF \rightarrow E$ and $ACDF \rightarrow G$. (rule 1)

- Next consider $ACDF \rightarrow G$. $ACDF \rightarrow G$ dependency is implied by the following FDs: $\{A \rightarrow B,\ ABCD \rightarrow E$, and $EF \rightarrow G.\}$

  $A \rightarrow B,\ ABCD \rightarrow E \Rightarrow ACD \rightarrow E \Rightarrow ACDF \rightarrow EF,\ EF \rightarrow G \Rightarrow ACDF \rightarrow G$

  Therefore, we can delete $ACDF \rightarrow G$. (rule 2)

- Similarly, we can delete $ACDF \rightarrow E$.

  $A \rightarrow B,\ ABCD \rightarrow E \Rightarrow ACD \rightarrow E \Rightarrow ACDF \rightarrow E$

  Therefore, we can delete $ACDF \rightarrow E$.

- Next consider $ABCD \rightarrow E$.

  Since $A \rightarrow B,\ ABCD \rightarrow E \Rightarrow ACD \rightarrow E$.

  Therefore, we can also delete $ABCD \rightarrow E$.

- A this point one can verify that each remaining FD is minimal and required. Thus, a minimal cover for F is the set:

  $F_{min} = \{A \rightarrow B,\ ACD \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow H\}.$

Normal Forms Based on Primary Keys

- Normalization of Relations
- Practical Use of Normal Forms
- Definitions
- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF

Normalization of Relations

- **Normalization**: The 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
- 1NF, 2NF, 3NF, BCNF, 4NF, 5NF

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 normalize 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

- 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 $t_1[S] = t_2[S]$

  $S^+ = R$

- 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

- 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.