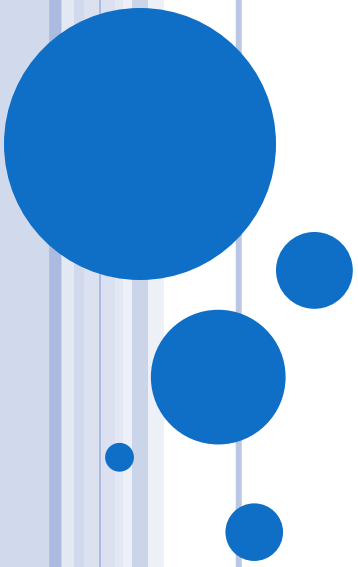


Functional Dependencies and Normalization for Relational Databases

Relational Database Designs

- There are many different relational database "designs" (schemas) that can be used to store the relevant mini-world information.

- Can one schema be much better than another?

Example: Shipment of Parts

- Consider a database containing shipments of parts from suppliers.
  - A supplier belongs to a particular city and a status is associated with each city.
  - A part has a name, color, quantity and weight and is shipped on particular date

- Proposed relation to capture this information:

Ship(<u>S#</u>, sname, status, city, <u>P#</u>, pname, colour, weight, qty, <u>date</u>)

EXAMPLE: SHIPMENT OF PARTS

Ship(S#, sname, status, city, P#, pname, colour, weight, qty, date)

S#	sname	status	city	P#	pname	colour	weight	qty	date
S1	Smith	20	London	P1	Nut	Red	12	200	980620
S1	Smith	20	London	P1	Nut	Red	12	700	980625
S2	Jones	10	Paris	P3	Screw	Blue	17	400	980620
S2	Jones	10	Paris	P5	Bolt	Green	17	300	980620
S2	Jones	10	Paris	P2	Screw	Red	14	200	980621
S3	Clark	20	Rome	P1	Nut	Red	12	300	980612
S3	Clark	20	Rome	P6	Cog	Red	19	600	980612
S4	Blake	30	Athens	P4	Cam	Blue	12	200	980619
S4	Blake	30	Athens	P1	Nut	Red	12	300	980619
S4	Blake	30	Athens	P3	Screw	Blue	17	100	980620
S5	Alex	10	Paris	P1	Nut	Red	12	250	980626

Problems ??

- Redundant Data
- Inability to represent certain information
- Loss of information

Example: Shipment of Parts

Ship(S#, sname, status, city, P#, pname, colour, weight, qty, date)

- INSERT
  - We can not enter a new supplier until the supplier ships a part.
  - We can not enter a new part until it is shipped by a supplier.
- DELETE
  - If we delete the only tuple of a particular supplier, we destroy not only the shipment information.
- UPDATE
  - The city of a supplier may appear many times in Ship.
  - If we change the value of the city in one tuple but not all the other, it creates inconsistency in the database.

Design Anomalies

Ship(S#, sname, status, city, P#, pname, colour, weight, qty, date)

- This Apply relation exhibits three types of anomalies:
  - Redundancy
  - Update Anomaly
  - Deletion Anomaly

- Good designs:
  - No anomalies
  - Can reconstruct all original information

EXAMPLE: SHIPMENT OF PARTS

The solution is to decompose Ship into three relations Shipment, Supplier, and Parts

Supplier(S#, sname, status, city)
Parts(P#, pname, colour, weight)
Shipment(S#, P#, qty, date)

S#	sname	status	city
S1	Smith	20	London
S2	Jones	10	Paris
S3	Clark	20	Rome
S4	Blake	30	Athens
S5	Alex	10	Paris

P#	pname	colour	weight
P1	Nut	Red	12
P2	Screw	Red	14
P3	Screw	Blue	17
P4	Cam	Blue	12
P5	Bolt	Green	17
P6	Cog	Red	19

S#	P#	qty	date
S1	P1	200	980620
S1	P1	700	980625
S2	P3	400	980620
S2	P5	300	980620
S2	P2	200	980621
S3	P1	300	980612
S3	P6	600	980612
S4	P4	200	980619
S4	P1	300	980619
S4	P3	100	980620
S5	P1	250	980626

Example: Shipment of Parts

Supplier(S#, sname, status, city) **Parts(P#, pname, color, weight)**

Shipment(S#, P#, qty, date)

Problems ??

- INSERT
- DELETE
- UPDATE

EXAMPLE: SHIPMENT OF PARTS

The solution is to decompose Supplier into 2 relations Supplier and Status

Supplier(S#, sname, city) Status(city, status)
Parts(P#, pname, colour, weight)
Shipment(S#, P#, qty, date)

S#	sname	city
S1	Smith	London
S2	Jones	Paris
S3	Clark	Rome
S4	Blake	Athens
S5	Alex	Paris

city	status
London	20
Paris	10
Rome	20
Athens	30

P#	pname	colour	weight
P1	Nut	Red	12
P2	Screw	Red	14
P3	Screw	Blue	17
P4	Cam	Blue	12
P5	Bolt	Green	17
P6	Cog	Red	19

S#	P#	qty	date
S1	P1	200	980620
S1	P1	700	980625
S2	P3	400	980620
S2	P5	300	980620
S2	P2	200	980621
S3	P1	300	980612
S3	P6	600	980612
S4	P4	200	980619
S4	P1	300	980619
S4	P3	100	980620
S5	P1	250	980626

Informal Design Guidelines for Relational Databases

- We first discuss informal guidelines for good relational design
- Then we discuss formal concepts of functional dependencies and normal forms

Semantics of the Relation Attributes

- *Each tuple in a relation should represent one entity or relationship instance.*

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

- Attributes of different entities 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.

Redundant Information in Tuples and Update Anomalies

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

- Information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

EXAMPLE : REDUNDANT INFORMATION

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

- Insert Anomaly
- Delete Anomaly
- Update Anomaly

Two relation schemas suffering from update anomalies

Figure 10.3
Two relation schemas suffering from update anomalies.
(a) EMP_DEPT and
(b) EMP_PROJ.

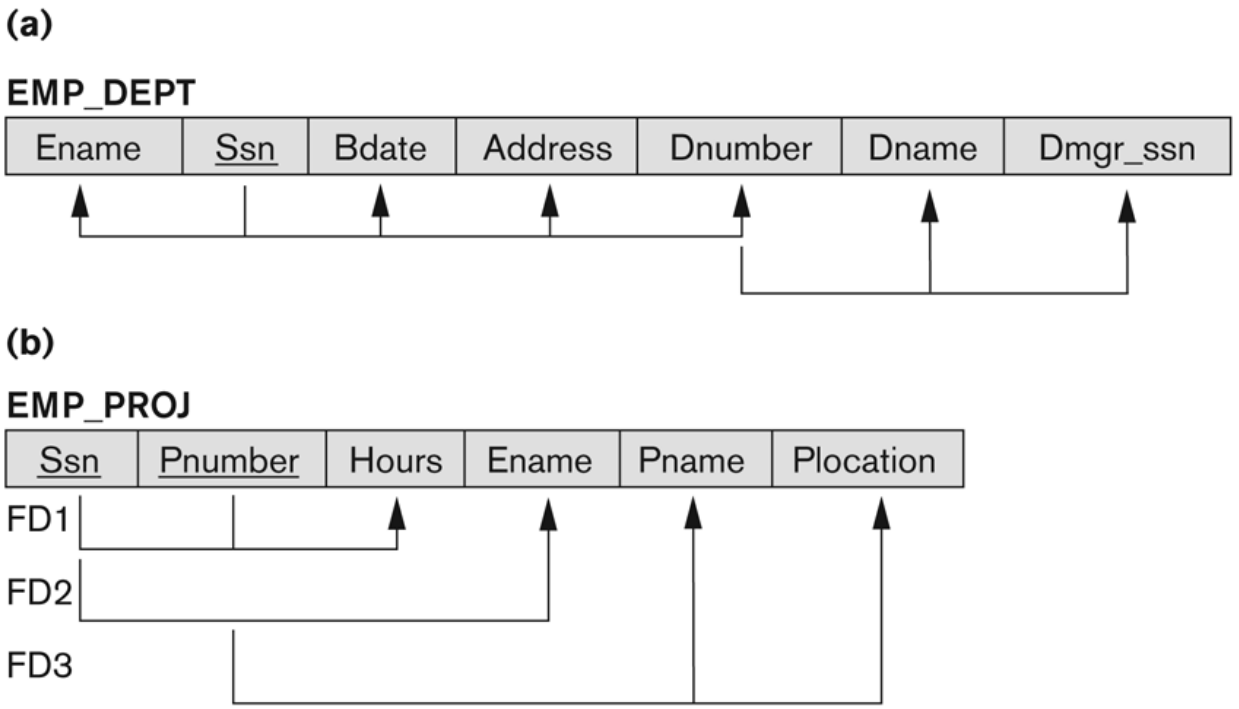


Figure 10.4

Example states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 10.2. These may be stored as base relations for performance reasons.

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
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

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Redundant Information in Tuples and Update Anomalies

- Design a schema that does not suffer from
  - insertion,
  - deletion and
  - update anomalies.

Null Values in Tuples

- Relations should have few NULL values
- 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 exist)
  - Value known to exist, but unavailable

Relation Decomposition

- In order to solve the previous problems we decomposed the relations into smaller relations.
- Relation decomposition can be dangerous we can lose information.
- Loss-less join decomposition
- If we decompose a relation R into smaller relations, the join of those relations should return R, not more, not less.

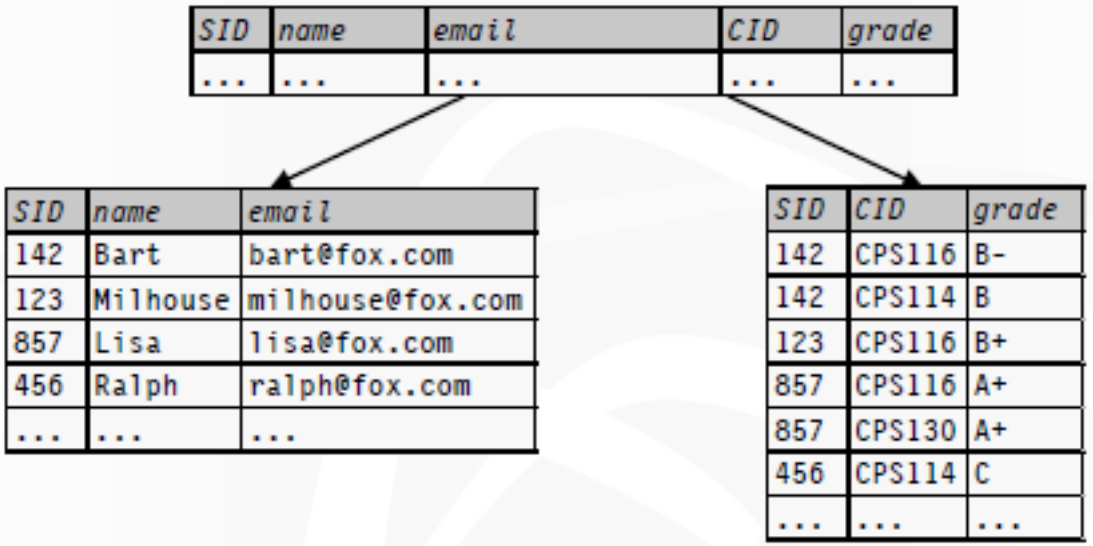
Example: Relation Decomposition

- Redundant Information

SID	*name*	*email*	*CID*	*grade*
142	Bart	bart@fox.com	CPS116	B-
142	Bart	bart@fox.com	CPS114	B
123	Milhouse	milhouse@fox.com	CPS116	B+
857	Lisa	lisa@fox.com	CPS116	A+
857	Lisa	lisa@fox.com	CPS130	A+
456	Ralph	ralph@fox.com	CPS114	C
...	...	...	...	...

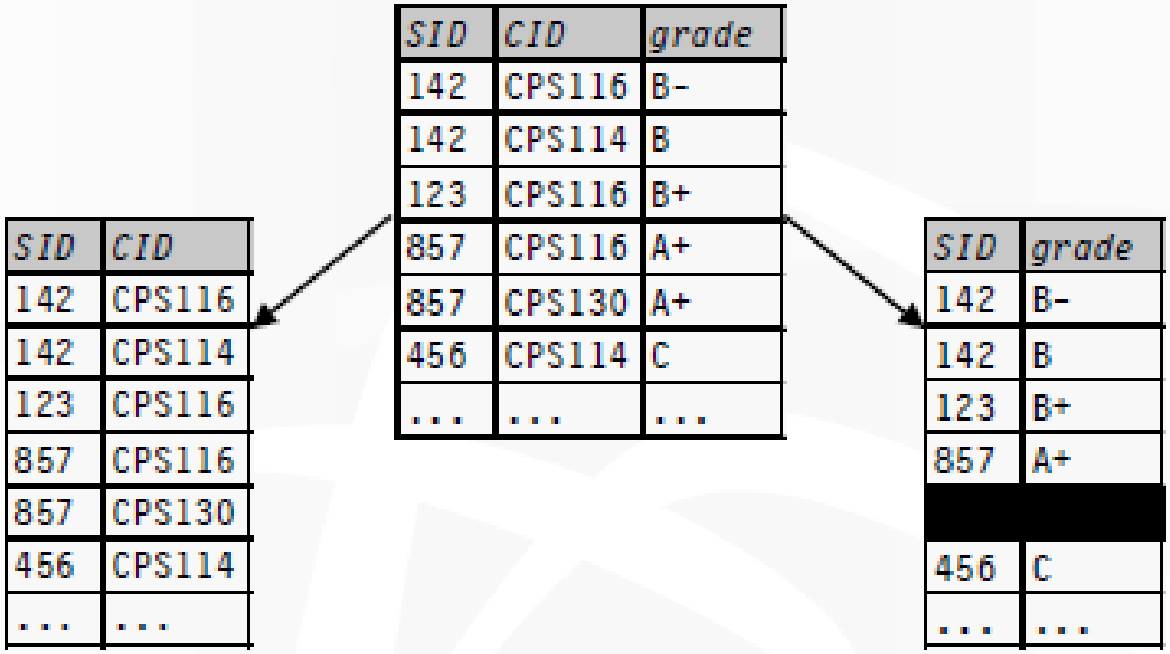
•Eliminates redundancy

•To get back to the original relation use join



Bad decomposition

- Association between *CID and grade is lost*
- Join returns more rows than the original relation



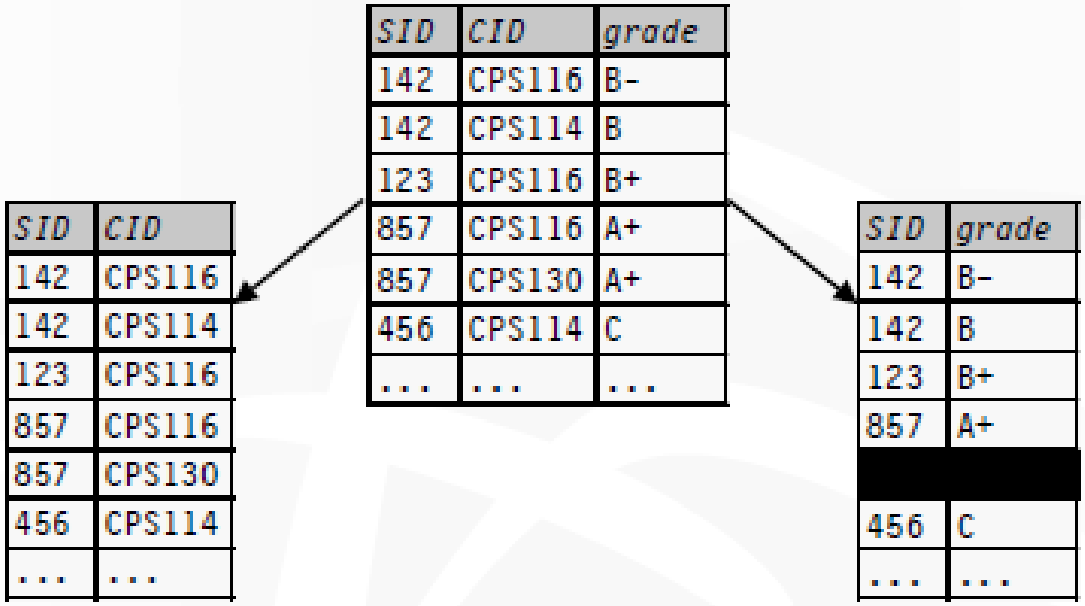
SID	*CID*	*grade*
142	CPS116	B-
142	CPS114	B
123	CPS116	B+
857	CPS116	A+
857	CPS130	A+
456	CPS114	C
...	...	...

SID	*CID*
142	CPS116
142	CPS114
123	CPS116
857	CPS116
857	CPS130
456	CPS114
...	...

SID	*grade*
142	B-
142	B
123	B+
857	A+
456	C
...	...

Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- Relations should satisfy the lossless join condition.
- No spurious tuples should be generated by doing a natural-join of any relations.



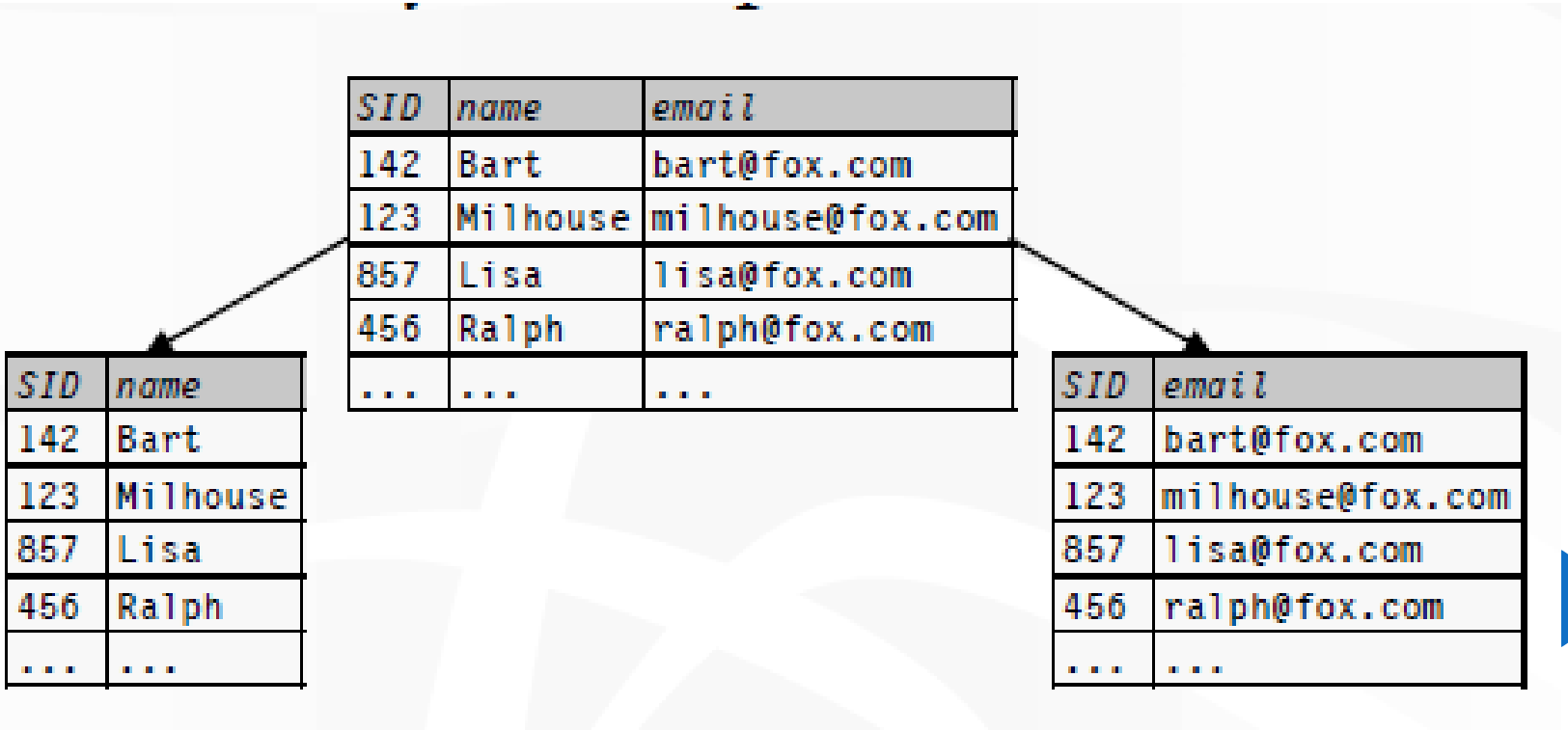
SID	CID	grade
142	CPS116	B-
142	CPS114	B
123	CPS116	B+
857	CPS116	A+
857	CPS130	A+
456	CPS114	C
...	...	...

SID	CID
142	CPS116
142	CPS114
123	CPS116
857	CPS116
857	CPS130
456	CPS114
...	...

SID	grade
142	B-
142	B
123	B+
857	A+
456	C
...	...

UNNECESSARY DECOMPOSITION

- Join returns the original relation
- Unnecessary: no redundancy is removed, and now *SID is stored twice!*



SID	*name*	*email*
142	Bart	bart@fox.com
123	Milhouse	milhouse@fox.com
857	Lisa	lisa@fox.com
456	Ralph	ralph@fox.com
...	...	...

SID	*name*
142	Bart
123	Milhouse
857	Lisa
456	Ralph
...	...

SID	*email*
142	bart@fox.com
123	milhouse@fox.com
857	lisa@fox.com
456	ralph@fox.com
...	...

Functional Dependencies

Motivation

- How do we tell if a design is bad?

SID	*name*	*CID*
142	Bart	CPS116
142	Bart	CPS114
857	Lisa	CPS116
857	Lisa	CPS130
...	...	...

- This design has redundancy and update anomalies

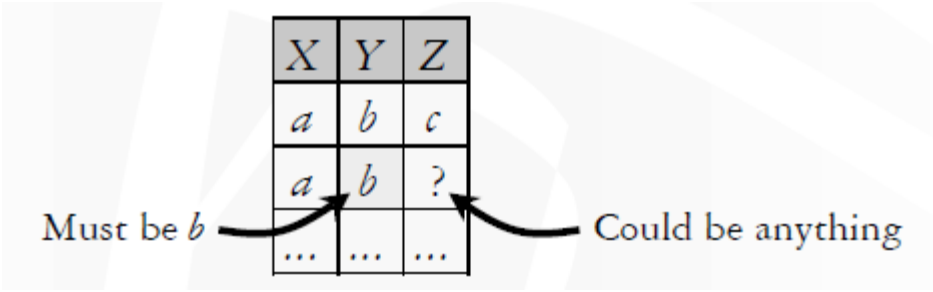
- How about a systematic approach to detecting and removing redundancy in designs?
  - Dependencies, decompositions, and normal forms

Functional Dependencies

- Functional dependencies play an important role in database design.
- They describe relationships between attributes.
- FDs are derived from the real-world constraints on the attributes

Functional Dependencies

- A functional dependency (FD) has the form $X \rightarrow Y$, where X and Y are sets of attributes in a relation R
- $X \rightarrow Y$ means that whenever two tuples in R agree on all the attributes in X, they must also agree on all attributes in Y



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

- *We say X functionally determines Y*
- X -> Y in R specifies a *constraint* on all relation instances r(R)

Example: Student Grade Info

- *StudentGrade* (*SID*, *name*, *email*, *CID*, *grade*)
  - *SID* → *name*, *email*
  - *email* → *SID*
  - *SID*, *CID* → *grade*

- Not a good design

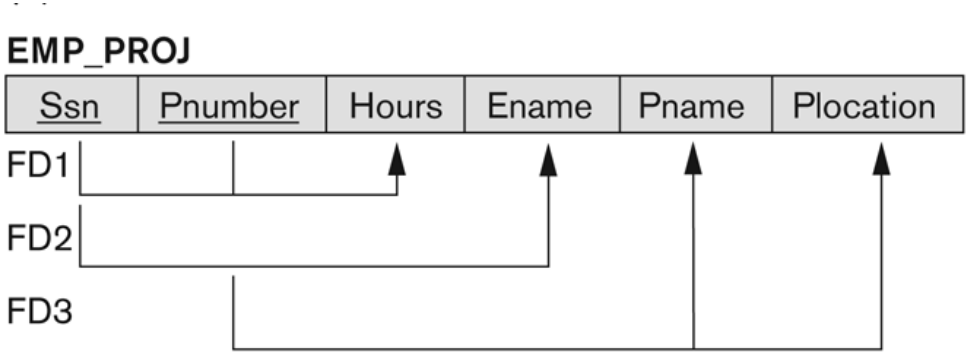
EXAMPLE: ADDRESS

Address (*street_address*, *city*, *state*, *zip*)

- *street_address*, *city*, *state* → *zip*
- *zip* → *city*, *state*
- *zip*, *state* → *zip*?
  - This is a trivial FD
  - Trivial FD: LHS ⊇ RHS
- *zip* → *state*, *zip*?
  - This is non-trivial, but not completely non-trivial
  - Completely non-trivial FD: LHS ∩ RHS = ∅

Examples of FD constraints

- If K is a key of R, then K functionally determines all attributes in R



- SSN -> ENAME
- PNUMBER -> {PNAME, PLOCATION}
- {SSN, PNUMBER} -> HOURS

FD is a property of relational schema R. Must be define by someone who knows the semantic of attributes of R.

FD's hold at all times

Certain FD's can be ruled out based on a given state of the database

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

- Text -> course
  - possible FD
- Teacher -> course
  - ruled out

Functional Dependencies

- Given a relation *R and a set of FD's F*
  - Does another FD follow from F?
  - Are some of the FD's in F redundant (i.e., they follow from the others)?

- Example: **Emp_Dept(ename, ssn, bdate, address, dnumber, dname, dmgrssn)**
  - F ={ssn -> {ename, bdate, address, dnumber),
    dnumber ->{dname, dmgrssn) }
  - Additional FD that can be infer from F
    - ssn-> dname, dmgrssn
    - ssn-> ssn
    - dnumber -> dname

Armstrong's Axioms (Inference Rules)

- Armstrong's Axioms
  - IR1. (**Reflexive**)

    If $Y \subseteq X$, then X -> Y
  - IR2. (**Augmentation**)

    If X -> Y, then XZ -> YZ

    (XZ stands for X U Z)
  - IR3. (**Transitive**)

    If X -> Y and Y -> Z, then X -> Z

These three functional dependencies are known to be sound and complete, i.e., using these three dependencies, we can infer all dependencies from the database.

Armstrong's Axioms

- Some additional inference rules that are useful:
  - **Decomposition:**
    - If X -> YZ, then X -> Y and X -> Z
  - **Union:**
    - If X -> Y and X -> Z, then X -> YZ
  - **Pseudotransitivity:**
    - If X -> Y and WY -> Z, then WX -> Z
- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Proof of IR3, IR4, and IR5 using IR1, IR2, and IR3.

- If X -> YZ, then X -> Y and X -> Z
  - YZ -> Y (reflexive)
  - X -> YZ (given)
  - X -> Y (transitivity)
- If X -> Y and X -> Z, then X -> YZ
  - X ->XY (augmenting X in X-> Y)
  - XY -> YZ (augmenting Y in X-> Z)
  - X -> YZ (transitivity)
- If X -> Y and WY -> Z, then WX -> Z
  - WX->WY (augmenting)
  - WY -> Z (given)
  - WX -> Z (transitivity)

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
- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

Algorithm for computing Attribute Closure

- The closure of *X (denoted X^+) with respect to F is* the set of all attributes functionally determined by *X*
- **Algorithm for computing the closure**

$X^+ = X$

repeat

$OldX^+ = X^+$

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

if $Y \subseteq X^+$ then $X^+ := X^+ \cup Z$

until $(X^+ = oldX^+)$

Example 1: Closure Algorithm

Consider following schema

EMP_PROJ(Ssn, Ename, Pno, Pname, Hours)

And FDs

- F = {Ssn -> Ename,
  Pno -> {Pname, Plocation},
  {Ssn, Pno} -> Hours }

- Find the following Closure Sets
- $\{Ssn\}^+ =$
- $\{Pno\}^+ =$
- $\{Ssn, Pno\}^+ =$

Example 2: Closure Algorithm

StudentGrade (SID, name, email, CID, grade)

F includes:

- *SID → name, email*
- *email → SID*
- *SID, CID → grade*

- ***{ CID, email }⁺ = ?***
- *X+ = {CID, email}*

Consider email → SID

- X+ = {SID, CID, email}

Consider *SID → name, email*

- X+ = {SID, CID, email, name}

Consider SID, CID → grade

- X+ = {SID, CID, email, name, grade}

Minimal Sets of FDs

- A set of FDs is **minimal (also called irreducible or canonical)** if it satisfies the following conditions:

  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot replace any dependency X -> A in F with Y -> A, where Y $\subseteq$ X and still have a set of dependencies that is equivalent to F.
  3. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets

Algorithm for finding Minimal Cover F for a set of FDs E

- Set F:= E
- Replace each FD X-> {A1, A2, …, An} in F by n FDs X-> A1, X->A2, … , X-> An
- For each FD X-> A in F

  For each attribute B of X

  if {{ F- { X -> A }} U { (X- B) -> A}} = F

  then replace X->A with (X- B) -> A in F
- For each remaining FD X->A in F

  if {F – {X->A} } = F

  then remove X->A from F

Example 1: Minimal Cover

- F = { A→BG, G→C, AB→C, AB->A}

- Step1: Remove Trivial Dependencies. Also decompose all dependencies, i.e.,

F={A→B, A→G, G→C, AB→C}

- Step 2: We need to look for redundant attributes on the LHS

F= {A→B, A→G, G→C, A→C}

- Step3: We need to look for FDs that are redundant.

F= {A→B, A→G, G→C}

Example 2: Minimal Cover

- F = { AB -> C, C -> A, BC -> D, ACD -> B, D -> EG,
  BE -> C, CG -> BD, CE -> AG}

- Step1: remove trivial dependencies and decompose.
- Step 2: We need to look for redundant attributes on the LHS
- Step3: We need to look for FDs that are redundant

Example 2: Minimal Cover

- For STEP 2: apply either of the rules below to find redundant attributes on the LHS:
  - 1. In an FD $\{X \rightarrow A\}$ from set F, attribute $B \in X$ is redundant in X if F logically implies

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

  - 2. To test if attribute $B \in X$ is redundant in X
    - Step 1: compute $(\{X\} - B)+$ using the dependencies in F
    - Step 2: check that $(\{X\} - B)+$ contains B; if it does, B is redundant

Example 2: Minimal Cover

- Consider ACD -> B to see if any of the three attributes on the LHS is redundant
- Computing CD+ ={ACDEGB}
  - Closure contains A and thus, A is redundant.
  - The closure contains B, which tells us that CD -> B holds.

- Alternatively, we need to prove that F logically implies CD -> B in place of ACD -> B.
  - C -> A (given in F)
  - CD -> ACD (augment with CD)
  - ACD -> B (given in F)
  - CD -> B (transitivity)

Example 2: Minimal Cover

- Step3: Look for redundant FDs in set F.
  - CE -> A
  - CG -> B
    - (as CG -> D, and CD -> B in F.)
  - No more redundant FDs in F.

- **Minimal Cover : {AB -> C, C -> A, BC -> D, CD -> B, D -> E, D -> G, BE -> C, CG -> D, CE -> G}**

Example 2: Minimal Cover

- **Minimal Cover 2 : {AB -> C, C -> A, BC -> D, D -> E, D -> G, BE -> C, CG -> B, CE -> G}**
- Obtained by eliminating redundant FDs,
  - CE -> A,
  - CG -> D, (CG->B, BC->D)
  - CD -> B (D->G, CG->B)

- We need one of {CG -> D, CG -> B} in the minimal cover – can not eliminate both.

Example 2: Compute Minimal Sets of FDs

Consider set of FDs E : {$B \rightarrow A$, $D \rightarrow A$, $AB \rightarrow D$}. We have to find the minimum cover of E.

- **Step1:** Canonical form
- **Step 2:** Determine if $AB \rightarrow D$ has any redundant attribute?

  Consider B → A

  $BB \rightarrow AB$ (augmenting with B -- IR2)

  $B \rightarrow AB$

  $AB \rightarrow D$ (given)

  We get $B \rightarrow D$ (*transitivity – IR3)*

  Hence, $AB \rightarrow D$ may be replaced by $B \rightarrow D$.

- Now E' : {$B \rightarrow A$, $D \rightarrow A$, $B \rightarrow D$}.

Example 2: Compute Minimal Sets of FDs

Consider set of FDs E : $\{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$. We have to find the minimum cover of E.

- **E' : $\{B \rightarrow A, D \rightarrow A, B \rightarrow D\}$.**
- **Step 3**: Look for a redundant FD in E′

  $B \rightarrow D$

  $D \rightarrow A$,

  Thus, $B \rightarrow A$ (transitive rule)

  Eliminate $B \rightarrow A$ in E'.

- **The minimum cover of E is $\{B \rightarrow D, D \rightarrow A\}$.**

Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - F covers G, and G covers F.

- **Covers**:
  - F **covers** G if every FD in G can be inferred from F
    - (i.e., if $G^+ \subseteq F^+$)

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

Example: Equivalence between sets of Functional Dependencies

- Consider two sets of FDs, F and G,
  - F = {A -> B, B -> C, AC -> D} and
  - G = {A -> B, B -> C, A -> D}
- Are F and G equivalent?

- We need to prove that F+ = G+, i.e., F covers E and E covers F.
  - To determine whether F covers E we calculate X+ with respect to F for each FD X->y in E and then check whether X+ includes the attributes Y.

  - Similarly, check whether E covers F by calculating X+ with respect to E for each FD X->y in F and then check whether X+ includes the attributes Y.

Example: Equivalence between sets of Functional Dependencies

- If F covers E, then it also means that we can infer all dependencies in E (either by applying inference rules on F or by taking closures of attributes on the left hand side of FDs in E w.r.t to F.

- Use attribute closure to infer all FDs in F using G
  - F = {A -> B, B -> C, AC -> D}
  - G = {A -> B, B -> C, A -> D}

- Take the attributes from the LHS of FDs in F and compute attribute closure for each using FDs in G:
  - (A+ using G) = ABCD;
  - (B+ using G) = BC;
  - (AC+ using G) = ABCD;

- All FDs in F are inferred using FDs in G.

Example: Equivalence between sets of Functional Dependencies

- F = {A -> B, B -> C, AC -> D}
- G = {A -> B, B -> C, A -> D}

- Now we see if all FDs in G are inferred by F
  - A+ using F = ABCD;
  - B+ using F = BC;

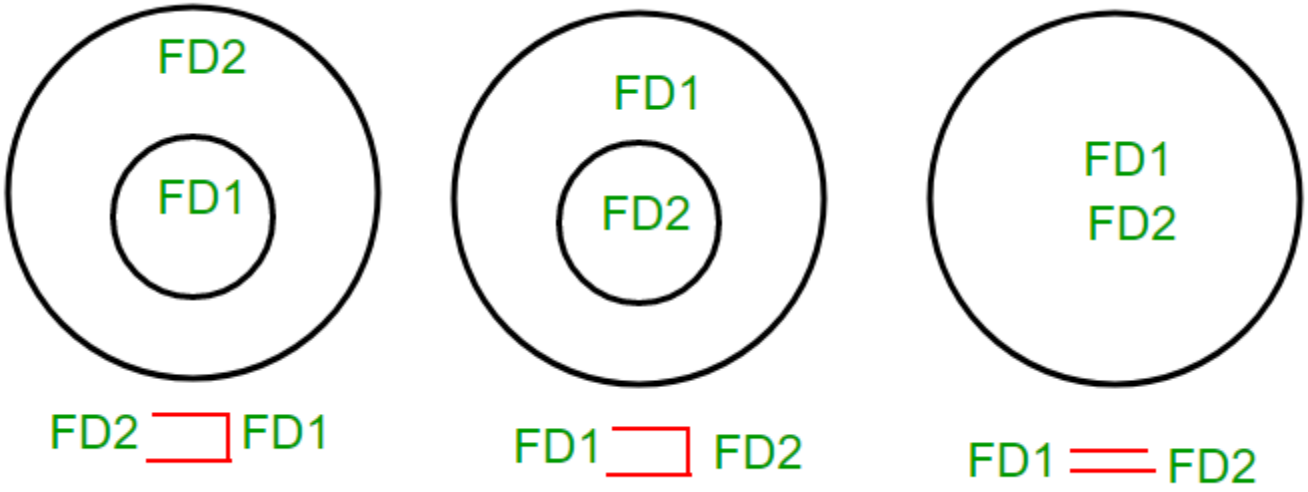
- All FDs in F can be obtained from G and vice versa, so we conclude that F and G are equivalent.

Example: Equivalence between sets of Functional Dependencies

- F = {A -> B, A ->C}
- G = {A -> B, B -> C}
- Are F and G equivalent?

- A+ using G = ABC;
- A+ using F = ABC;
- B+ using F = B,
  - This indicate B -> C in G is not inferred using the FDs from F.

- Thus, F and G are not equivalent, because B -> C in G is not inferred from the FDs in F.


FD2
FD1
FD2 FD1
FD1
FD2
FD1 FD2
FD1
FD2
FD1 FD2

Example: Find the Key for Relation R

- Consider a relation R= {A,B,C,D,E,F,G,H,I,J} and
- The set of FD's {
  - A,B -> C ,
  - B,D->E,F ,
  - A,D-> G,H ,
  - A->I ,
  - H->J }.
- Find the key for Relation R with given FD's.

Algorithm: To Find a Key K for R Given a set F of FD's

Input: A universal relation R and a set of functional dependencies F on the attributes of R.

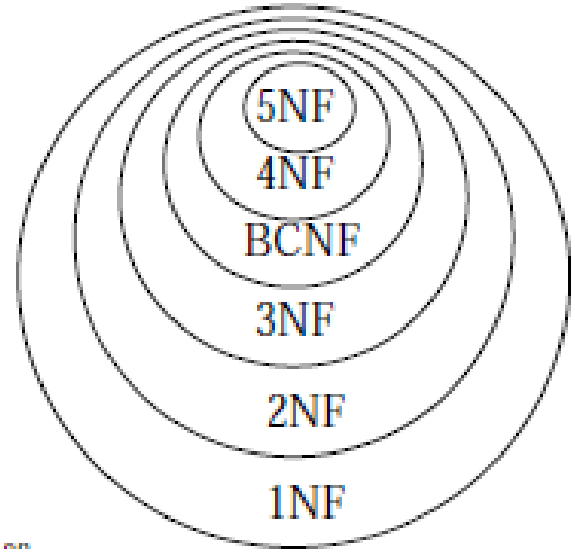
1. Set $K := R$.
2. For each attribute A in K {
   compute $(K - A)^+$ with respect to F;
   If $(K - A)^+$ contains all the attributes in R,
   then set $K := K - \{A\}$;
   }

Normalization

Normalization of Relations

- **Normalization:**
  - The process of decomposing "bad" relation by breaking it into smaller relations

- Normalization should maintain properties like lossless join and dependency preservation to ensure a good relational design



Normal Forms

- The database designers don't have to normalize relations to the highest possible normal form (usually 3NF, BCNF or 4NF is enough)

- **Denormalization**:
  - Opposite of normalization
  - Join relations to form a base relation—which is in a lower normal form

SuperKeys and Keys

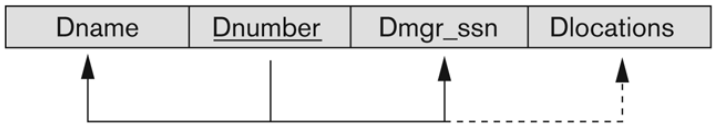
- *Superkey*
- *Key*
- *Candidate key*
- *Primary key*
- *Secondary keys*
- **Prime attribute**: it must be a member of *some* candidate key
- **Nonprime attribute**: it is not a member of any candidate key.

First Normal Form

- It disallows Composite attributes, Multivalued attributes and Nested relations
- Attributes can have atomic values for an *individual tuple*
- It is considered to be a part of the definition of relation

(a)

DEPARTMENT



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Normalization of nested relations into 1NF

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, AliciaJ.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 10.9
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Second Normal Form

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

- **EMP_PROJ(SSN, Ename, Pno, Pname, Hours)**
  - {SSN, Pno} -> Hours is a full FD
    - since neither SSN -> Hours nor Pno -> Hours hold
  - {SSN, Pno} -> Ename is not a full FD, it is called a partial dependency
    - since SSN -> Ename also holds

Second Normal Form

- **Second Normal Form:**

  A relation that is in 1NF and every non-prime attribute is fully functionally dependent on the primary key

- We can decompose a relation into 2NF relations via the process of 2NF normalization

Normalizing into 2NF

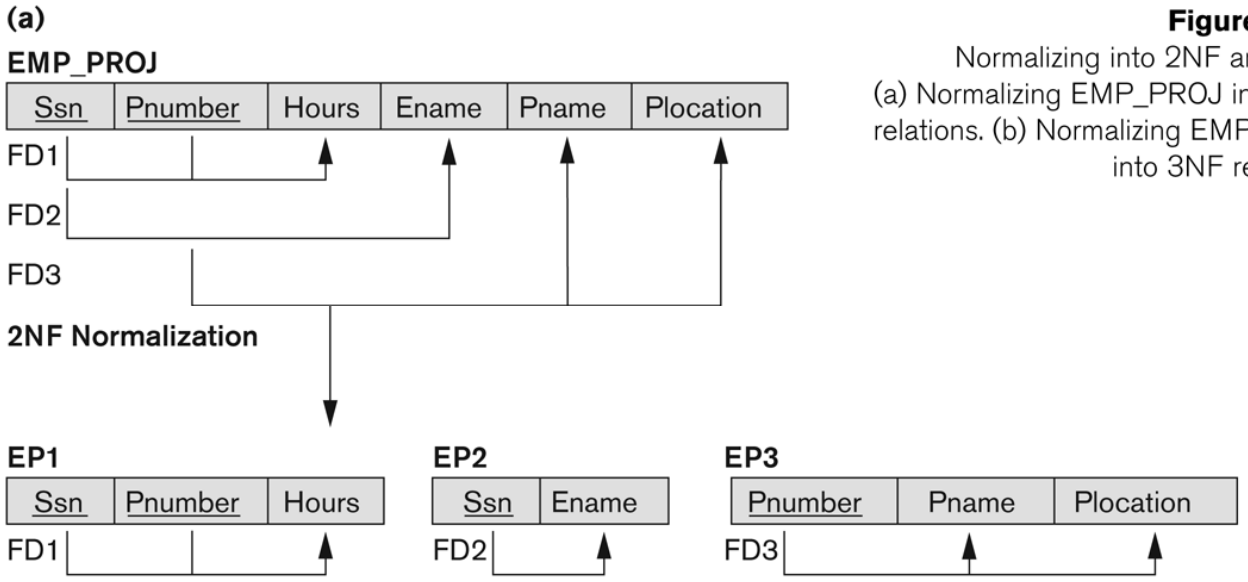


Figure 10.10
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Third Normal Form

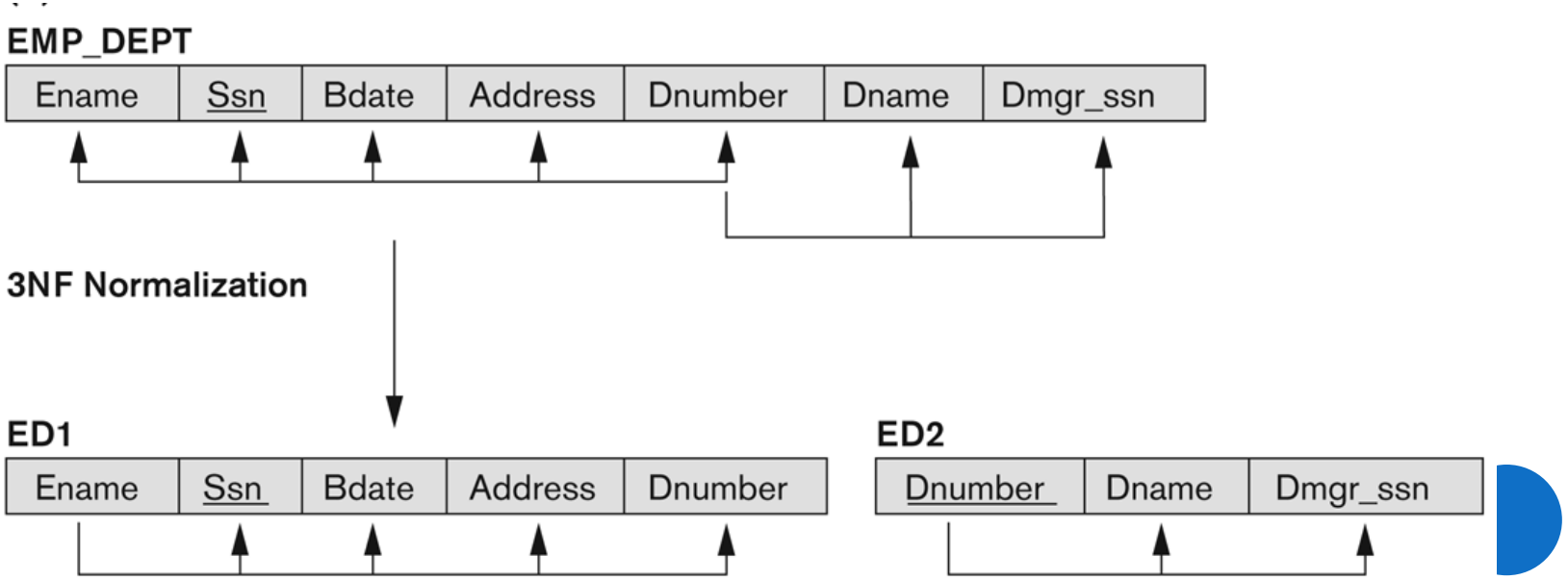
- **Transitive functional dependency:**
  - a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z

  - **Emp_Dept(SSN, Ename, Bdate, Address, Dno, Dname, Dmgrs_ssn)**
  - SSN -> DMGRSSN is a **transitive** FD
    - Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold
  - SSN -> ENAME is **non-transitive**
    - Since there is no set of attributes X where SSN -> X and X -> ENAME

Third Normal Form

- **Third Normal Form:**

  A relation that is in 2NF, and in which no non-prime attribute is transitively dependent on the primary key.



Third Normal Form

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

Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

SUMMARY OF NORMAL FORMS based on Primary Keys

Table 10.1
Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

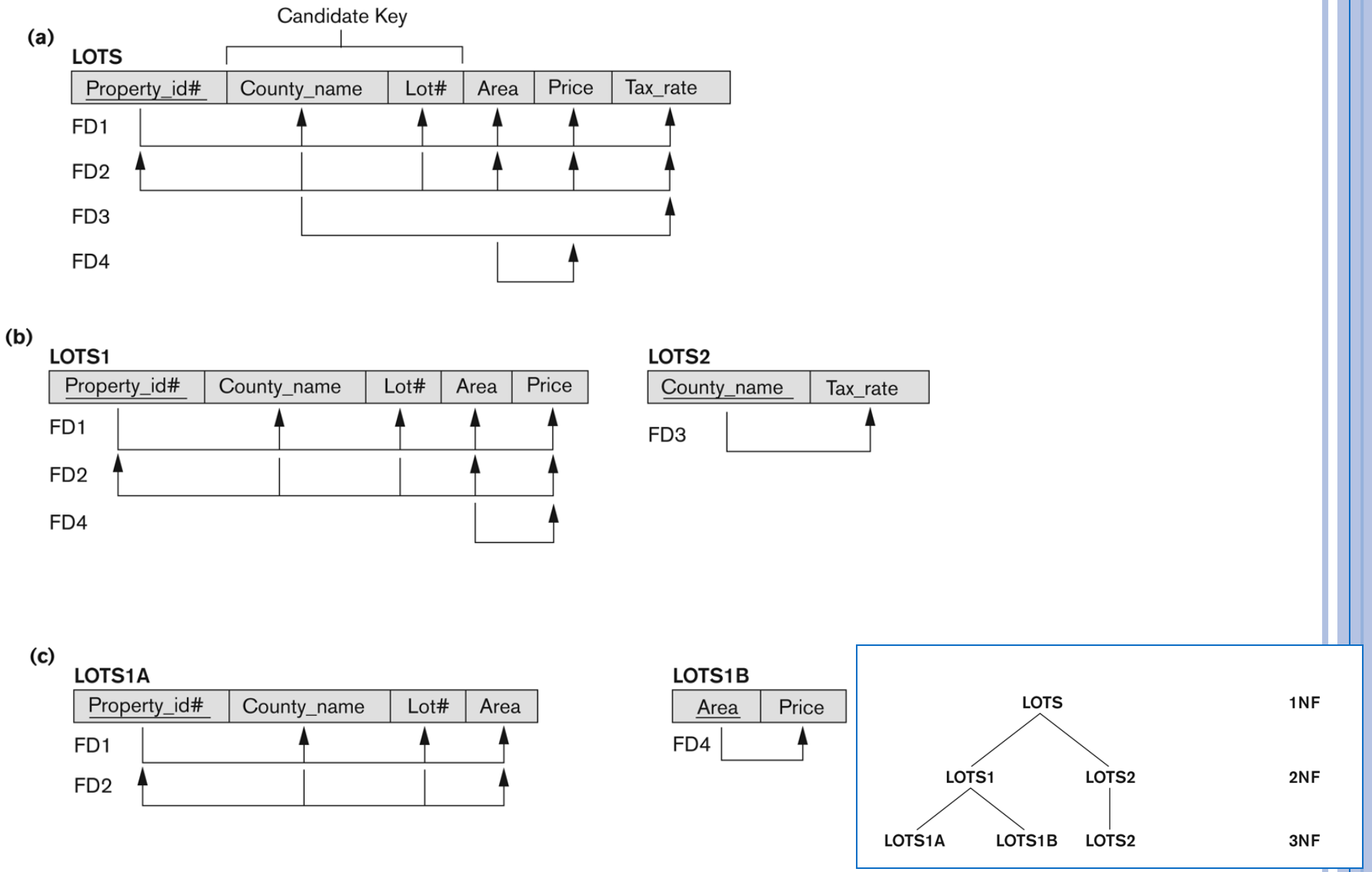
General Normal Form Definitions (For Multiple Keys)

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

General Normal Form Definitions

- A relation R is in **3NF** if whenever a FD X -> A holds in R, then either:
  a) X is a superkey of R, or
  b) A is a prime attribute of R

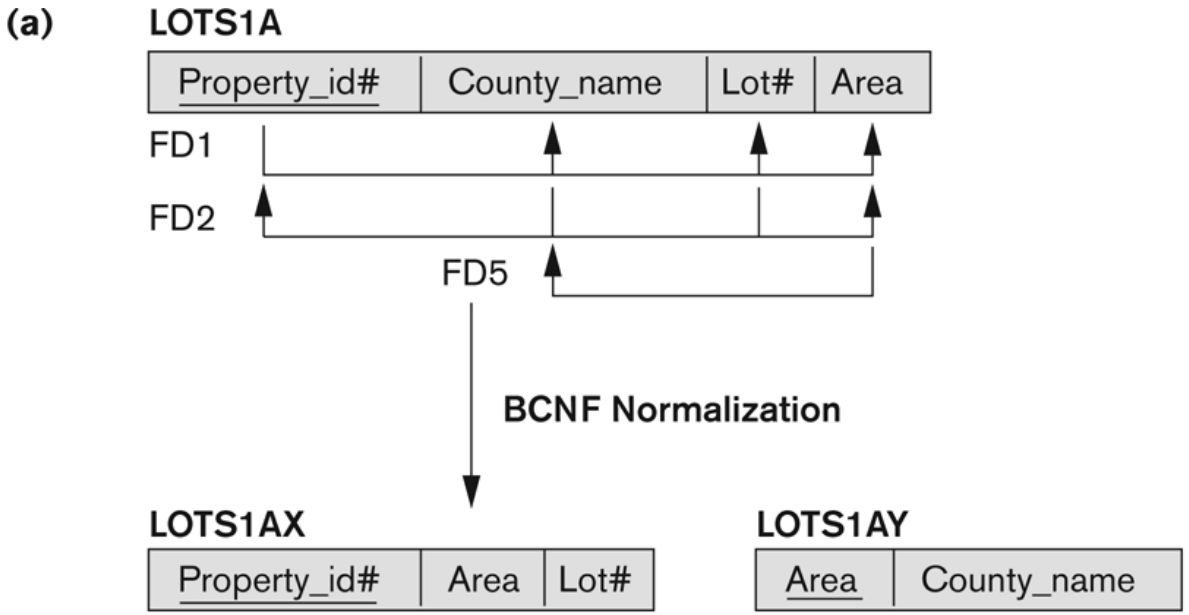
Successive Normalization of LOTS into 2NF and 3NF



BCNF (Boyce-Codd Normal Form)

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

Boyce-Codd Normal Form



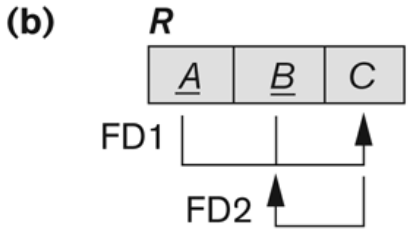


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

A relation that is in 3NF but not in BCNF

Two FDs exist in the relation TEACH:

FD1: { student, course} -> instructor

FD2: instructor -> course

{student, course} is a candidate key

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 10.13
A relation TEACH that is in 3NF but not BCNF.

A relation that is in 3NF but not in BCNF

Two FDs exist in the relation TEACH:

FD1: { student, course} -> instructor

FD2: instructor -> course

Three possible decompositions for relation TEACH

{student, instructor} and {student, course}

{course, instructor } and {student, course}

{course, instructor } and {student, instructor}

{student, course} is a candidate key

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

All three decompositions lose FD1. We sacrifice the FD preservation. But cannot sacrifice the non-additivity property after decomposition.

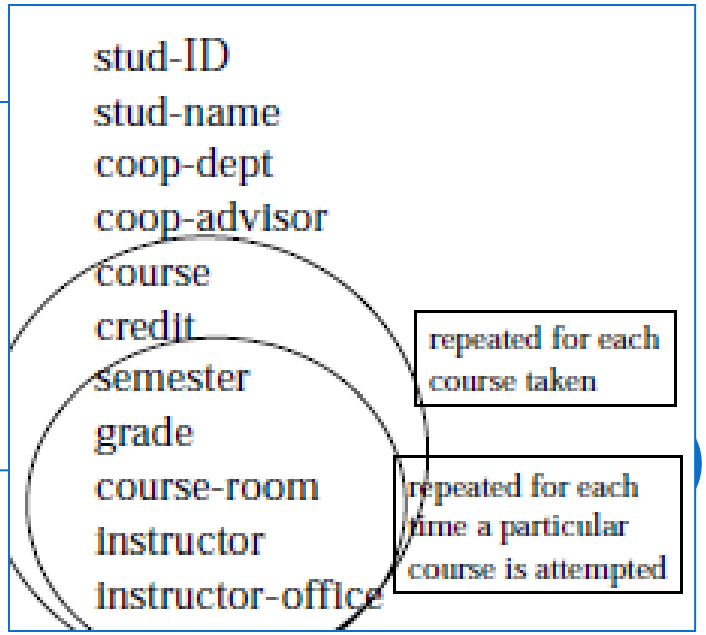
Only the 3rd decomposition will not generate spurious tuples after join

Example: Designing a Student Database

1. Students take courses
2. Students typically take more than one course
3. Students can fail courses and can repeat the same course in different semesters => Students can take the same course more than once.
4. Students are assigned a grade for each course they take.

The data that needs to be retained has repeating groups

Procedure: Remove repeating groups by adding extra rows to hold the repeated attributes. => Add redundancy.



Designing a Student Database -2

- **Student_course(sID, sname, dept, advisor, course, credit, semester, grade, course-room, instructor, instructor-office)**

- 1NF: Tables should have no repeating groups

- Problems:
  - Redundancy
  - Insert anomalies
  - Delete anomalies
  - Update problems

Designing a Student Database -3

- *sname, dept, advisor are dependent only* upon *sID*
- *credit dependent only on course and is independent of which* semester it is offered and which student is taking it.
- *course-room, instructor and instructor-office only depend upon the course and the semester (are independent of which student is* taking the course).
- Only *grade is dependent upon all 3 parts of the original key.*

- **2NF: The non-prime attributes should depend on the whole key.**
- **Procedure: remove partial key dependencies**

Designing a Student Database -4

- Student_course(sID, sname, dept, advisor, course, credit, semester, grade, course-room, instructor, instructor-office)

- **Student (sID, sname, dept, advisor)**
- **Student_Reg (sID, course, semester, grade)**
- **Course (course, credit)**
- **Course_Offering (course, semester, course-room, instructor, instructor-office)**

- 2NF: There are no non-key attributes with partial key dependencies in any table.
- Less redundancy
- Still insert/delete/update problems

Designing a Student Database -5

- **3NF: The non-key attributes should not depend on other non-key attributes.**

- **Procedure: remove non-key dependencies (transitive functional dependencies)**

Designing a Student Database -3NF

- **Student (sID, sname, dept, advisor)**
- **Student_Reg (sID, course, semester, grade)**
- **Course (course, credit)**
- **Course_Offering (course, semester, course-room, instructor, instructor-office)**

- **Student (sID, sname, dept, advisor)**
- **Student_Reg (sID, course, semester, grade)**
- **Course (course, credit)**
- **Course_Offering (course, semester, course-room, instructor)**
- **Instructor (instructor, instructor-office)**

  - More organized, Less redundancy (save space??)
  - performance problems?? (indirect references)

Designing a Student Database -7

- In normalization, we are seeking to make sure that attributes depend:
  - **on the key (1NF)**
  - **on the whole key (2NF)**
  - **on nothing but the key (3NF)**

- We have not checked whether some part of the key itself depends on a non-key attribute(s).

Designing a Student Database -8

- Suppose:
  1. each department may have more than one advisor
  2. a advisor works for only one department (*advisor -> dept)*
  3. a student may be associated with several departmental programs.

- **Student (<u>sID</u>, sname, <u>dept</u>, advisor)**

- Part of the compound key is determined by a non-key attribute

- *advisor -> dept*
  - *sname is function of only sID*
  - *dept is function of only advisor*

Designing a Student Database -9

- **Student (sID, sname)**
- **Advisor (advisor, dept)**
- **Student-Advice (sID, advisor)**
- **Student_Reg (sID, course, semester, grade)**
- **Course (course, credit)**
- **Course_Offering (course, semester, course-room, instructor)**
- **Instructor (instructor, instructor-office)**

BCNF: Table in 3NF and there are no dependencies of part of the compound key on another attribute

Normalization Example -- Sales Order

Sales Order

Fiction Company
202 N. Main
Mahattan, KS 66502

CustomerNumber:	*1001*	**Sales Order Number:**	405
Customer Name:	*ABC Company*	**Sales Order Date:**	2/1/2000
Customer Address:	*100 Points*	**Clerk Number:**	*210*
	Manhattan, KS 66502	**Clerk Name:**	*Martin Lawrence*

Item Ordered	Description	Quantity	Unit Price	Total
800	*widgit small*	*40*	*60.00*	*2,400.00*
801	*tingimajigger*	*20*	*20.00*	*400.00*
805	*thingibob*	*10*	*100.00*	*1,000.00*
	Order Total			*3,800.00*

Fields in the original data table will be as follows:

SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName, ItemNo, Description, Qty, UnitPrice

Normalization: First Normal Form

- Separate Repeating Groups into New Tables.
- ***Repeating Groups*** Fields that may be repeated several times for one document/entity
- The primary key of the new table (repeating group) is always a composite key;

- Relations in 1NF:
  - SalesOrderNo, ItemNo, Description, Qty, UnitPrice
  - SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName

Normalization: Second Normal Form

- Remove Partial Dependencies.
- Note: Do not treat price as dependent on item. Price may be different for different sales orders (discounts, special customers, etc.)

- Relations in 2NF
  - ItemNo, Description
  - SalesOrderNo, ItemNo, Qty, UnitPrice
  - SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName

Normalization: Second Normal Form

- **What if we did not Normalize the Database to Second Normal Form?**
  - Redundancy
  - Delete Anomalies
  - Insert Anomalies
  - Update Anomalies

Normalization: Third Normal Form

- Remove transitive dependencies

- **Relations in 3NF**
  - Customers: CustomerNo, CustomerName, CustomerAdd
  - Clerks: ClerkNo, ClerkName
  - Inventory Items: *ItemNo*, Description
  - Sales Orders: SalesOrderNo, Date, CustomerNo, ClerkNo
  - SalesOrderDetail: SalesOrderNo, ItemNo, Qty, UnitPrice

Normalization: Third Normal Form

- **What if we did not Normalize the Database to Third Normal Form?**
  - Redundancy – Detail for Cust/Clerk would appear on every SO
  - Delete Anomalies – Delete a sales order, delete the customer/clerk
  - Insert Anomalies – To insert a customer/clerk, must insert sales order.
  - Update Anomalies – To change the name/address, etc, must change it on every SO.

Chapter Summary

- Informal Design Guidelines for Relational Databases
- Functional Dependencies (FDs)
  - Definition, Inference Rules, Equivalence of Sets of FDs, Minimal Sets of FDs
- Normal Forms Based on Primary Keys
- General Normal Form Definitions (For Multiple Keys)
- BCNF (Boyce-Codd Normal Form)