Relational Database Design

Why do Relational Database Design?

- ER Model driven database design has been intuitive.
    - ☞ experience and judgement helps determine ER Schema.
- Relational Data Schema can be got from ER Schema.
- But, how “good” is this Relational Data Schema?
- Relational theory helps <u>formally</u> compare one relational schema with another.
- Relational theory deals with design of base relations.

Informal Design Guidelines

- Clear semantics for attributes

  It should be easy to explain the meaning of a relation schema.

  ☞ should not combine attributes from multiple entity types and relationship types.

- Reduce redundant values in tuples

  The base relation schema should not cause insertion, deletion, or update anomalies.

  ☞ Note anomalies clearly so that update programs update the database correctly.

- Reduce null values in tuples -

  Avoid placing attributes in a base relation whose values may be null

  ☞ Nulls should be exceptions -- not common case!

Informal Design Guidelines (cont.)

- Disallow Spurious Tuples

  It should be possible to join relations using equality condition on primary key and foreign key attributes in a way that guarantees that no spurious tuples are generated.

  ☞ Relations should have loss-less join property.

STUDENT_COURSE			
SID	Name	Cnum	Sem
1	Jim	comp231	fall95
1	Jim	comp111	spring94
2	Alice	comp111	spring94
2	Alice	comp211	fall95

STUDENT		
SID	Name	Sem
1	Jim	fall95
1	Jim	spring94
2	Alice	spring94
2	Alice	fall95

COURSE	
Cnum	Sem
comp231	fall95
comp111	spring94
comp111	spring94
comp211	fall95

STUDENT_COURSE_SPURIOUS			
SID	Name	Cnum_	Sem
1	Jim	comp231	fall95
1	Jim	comp211	fall95
1	Jim	comp111	spring94
2	Alice	comp231	fall95
2	Alice	comp211	fall95
2	Alice	comp111	spring94

Functional Dependencies

- a Functional Dependency (FD) is a constraint between two sets of attributes from the database.
- defined using Universal Relation Schema Assumption

  Given a relational database schema with n attributes $A_1, A_2, \ldots, A_n$, then a single universal relation $R = \{A_1, A_2, \ldots A_n\}$ describes the whole database.

  Notation: R - Relation Schema, **r** - Relation Instance

- Functional Dependency

  $X \rightarrow Y$ (X & Y are set of attributes of R)

  for any two tuples $\mathbf{t}_1$ and $\mathbf{t}_2$ in **r** such that

  $\mathbf{t}_1[X] = \mathbf{t}_2[X]$ we must also have $\mathbf{t}_1[Y] = \mathbf{t}_2[Y]$

Example:

HKID $\rightarrow$ DoB

Person

HKID	DoB
K222222	31/4/48
P111111	29/2/29
R333333	31/2/61
K222222	

Use of Functional Dependencies

- To specify constraints on legal relations
  - ☞keys

  If a set of relations *R* satisfy a set *F* of FDs, we say that *F* <u>holds</u> on *R*.

- To test relations to see whether they are legal under a given set of functional dependencies
  - ☞determine if constraints are violated

  If a relation r is legal under a set F of functional dependencies, we say that *r* <u>satisfies</u> *F*.

- To “improve” the relation schema by removing undesirable dependencies
  - ☞decomposition

Inference Rules for FDs

- F denotes the set of FDs specified for R
- Other FDs may also hold on all legal instances that satisfy F
- The set of all such FDs is called <u>closure</u> of F denoted by <u>F</u>$^{+}$

- A FD $X \rightarrow Y$ is <u>inferred</u> from F if $X \rightarrow Y$ <u>holds in every legal instance</u> **r** of R

  ☞ need a set of inference rules that can systematically infer new FDs from a given set of FDs

- <u>Notation:</u>

  $F \models X \rightarrow Y$ (F infers $X \rightarrow Y$); $\{X,Y\} \rightarrow Z$ written as $XY \rightarrow Z$;

  and $\{X,Y,X\} \rightarrow \{UV\}$ written as $XYZ \rightarrow UV$

Reflexivity

IR1: (Reflexivity) If $X \supseteq Y$, then $X \rightarrow Y$.

Proof:

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

Then, $t_1[Y] = t_2[Y]$ because $X \supseteq Y$;

hence $X \rightarrow Y$ must hold.

STUDENT_COURSE			
SID	Name	Cnum	Sem
1	Jim	comp231	fall95
1	Jim	comp111	spring94
2	Alice	comp111	spring94
2	Alice	comp211	fall95

$X = \{SID, Name, Cnum\}$

$Y = \{SID, Name\}$

$X \rightarrow Y$

Augmentation

IR2: (<u>Augmentation</u>) If $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

Proof (by contradiction):

Assume that $X \rightarrow Y$ holds in a relation instance r of R but that $XZ \rightarrow YZ$ does not hold. Then there exists two tuples such that

1). $t_1[X] = t_2[X]$

2). $t_1[Y] = t_2[Y]$

3). $t_1[XZ] = t_2[XZ]$

4). $t_1[YZ] \neq t_2[YZ]$

This is not possible because from (1) and (3) we have (5) $t_1[Z]= t_2[Z]$ and from (2) and (5) we have (6) $t_1[YZ] = t_2[YZ]$ contradicting (4).

Transitivity

IR3: (Transitivity) If $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

Proof:

Assume that (1) $X \rightarrow Y$ and (2) $Y \rightarrow Z$ both hold in a relation r.

Then for any two tuple 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)).

Hence we must also have (4) $t_1[Z] = t_2[Z]$, (from (3) and assumption (2));

hence, $X \rightarrow Z$ must hold in r.

Inference Rules (cont.)

- IR1 to IR3 are

  sound: given a set of FDs F specified on a relation schema R, any FD that we can infer from F by using IR1, IR2, and IR3 holds in every relation state r of R that satisfies the FDs in F

  complete: using IR1, IR2, and IR3 repeatedly to infer FDs until no more FDs can be inferred will result in the complete set of all possible FDs that can be inferred from F.

Additional Inference Rules

IR4 Projectivity

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

IR5 Additive

$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

IR6 Pseudotransitivity

$\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

Definitions

1. <u>Key and Superkey:</u>

   A super key, S, of $R = \{A_1, A_2, \ldots, A_n)$ is a set of attributes $S \subseteq R$ such that for any two tuples t_1 and t_2 in r, $t_1[S] \neq t_2[S]$

   A key, K, is a superkey that is minimal.

2. <u>Prime and non-prime attributes:</u>

   <u>prime:</u> if part of any candidate key

   <u>non-prime:</u> if not part of any candidate key

3. <u>Full dependency:</u>

   Given a relation scheme R and an FD $X \rightarrow Y$, Y is fully functionally dependent on X if there is no $Z \subset X$ such that $Z \rightarrow Y$

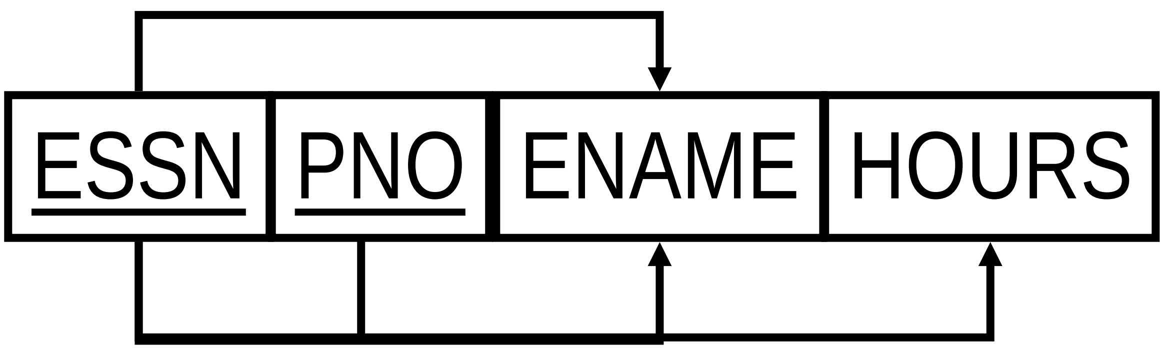
Definitions (cont.)

4. Partial Dependency

Given a relation scheme R, FDs F, a functional dependency $X \rightarrow Y$ is a partial dependency if some attribute $A \in X$ can be removed from X and the dependency still holds;

That is, for some $A \in X$, $(X - \{A\}) \rightarrow Y$.

PRJ



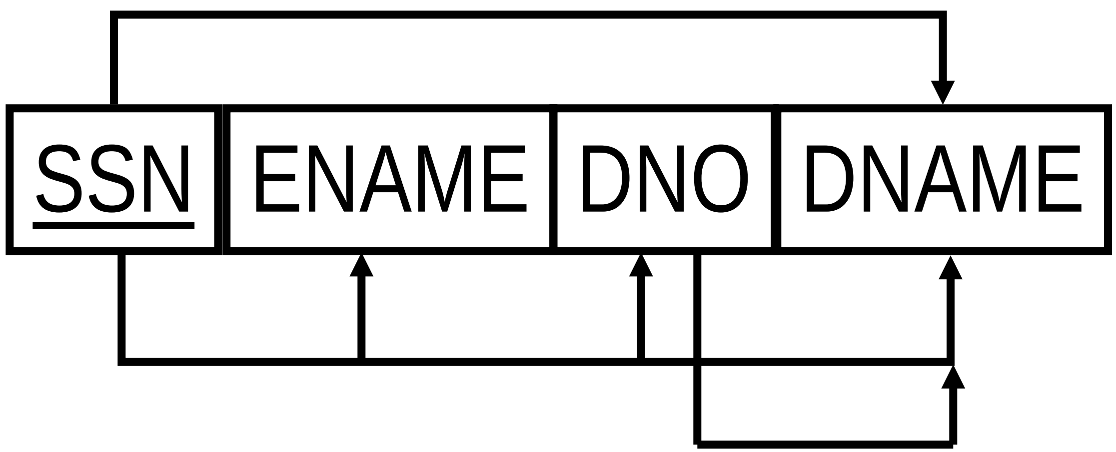
Definitions (cont.)

4. <u>Transitive Dependency</u>

   Given a relation scheme R, FDs F, let X, Y and Z be subsets of R such that $X \not\subset Y$, and $Z \not\subset XY$.

   If the set of FDs $\{ X \rightarrow Y, Y \rightarrow Z \}$ is implied by F, then Z is transitively dependent on X.

DEPT



Normalization

- A systematic process of decomposing unsatisfactory relation schemas into smaller relations schemas that possess desirable properties.

- Provides a series of tests for relation schemas
- expressed in terms of normal forms
  - ☞ first four: only use FDs
  - ☞ additional: use other types of dependencies

- Not necessary to normalize to highest form!

- Normal forms do not guarantee good design
  - ☞ also need to consider
    - ◆ loss less join property
    - ◆ dependency preserving property

First Normal Form (1NF)

A Relation schema is in 1NF if the values in the domain of each attribute are atomic.

R(K, ... M1, M2 ... Mn)

DEPARTMENT			
DID	DName	DMGR	DLOCATIONS
5	Research	101	{HK, Beijing, Shanghai}
2	Admin	105	London
1	Headquarters	108	Hyderabad

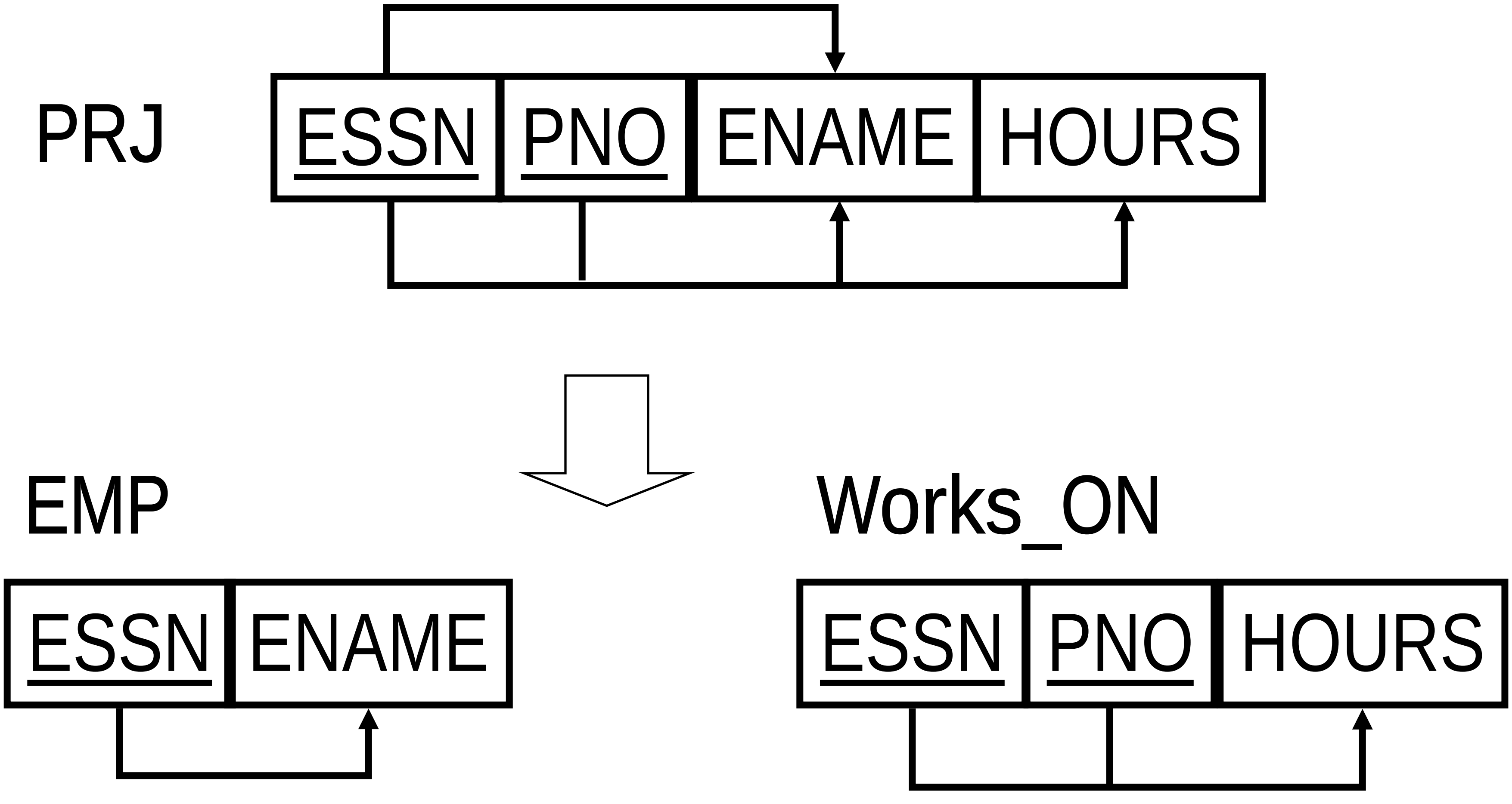
DEPARTMENT			
DID	DName	DMGR	DLOCATIONS
5	Research	101	HK
5	Research	101	Beijing
5	Research	101	Shanghai
2	Admin	105	London
1	Headquarters	108	Hyderabad

DEPARTMENT		
DID	DName	DMGR
5	Research	101
2	Admin	105
1	Headquarters	108

Location	
DID	DLOCATIONS
5	HK
5	Beijing
5	Shanghai
2	London
1	Hyderabad

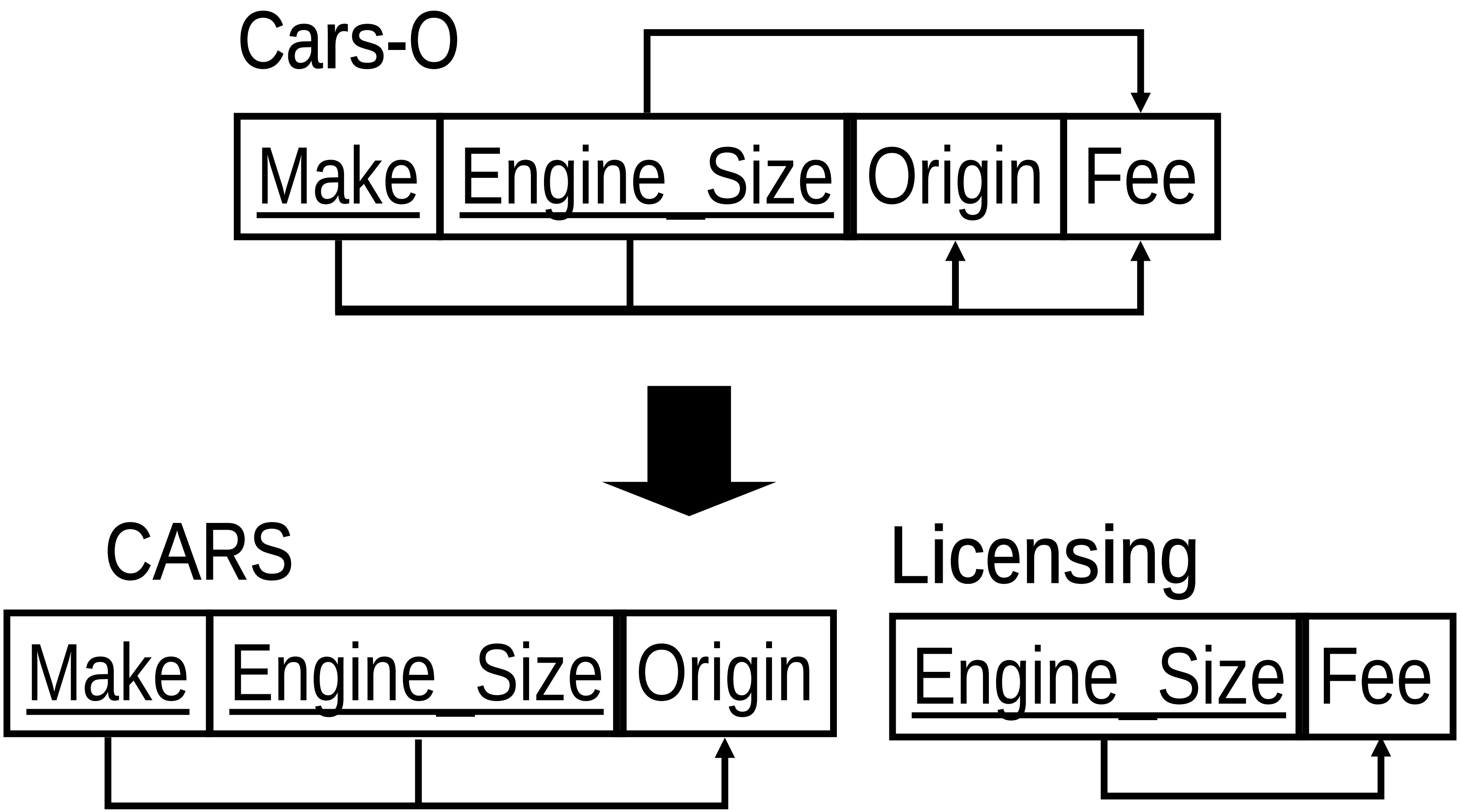
Second Normal Form (2NF)

A relation schema is in 2NF if every nonprime attribute A in R is fully functionally dependent on every key of R.



Second Normal Form (2NF)

Example:

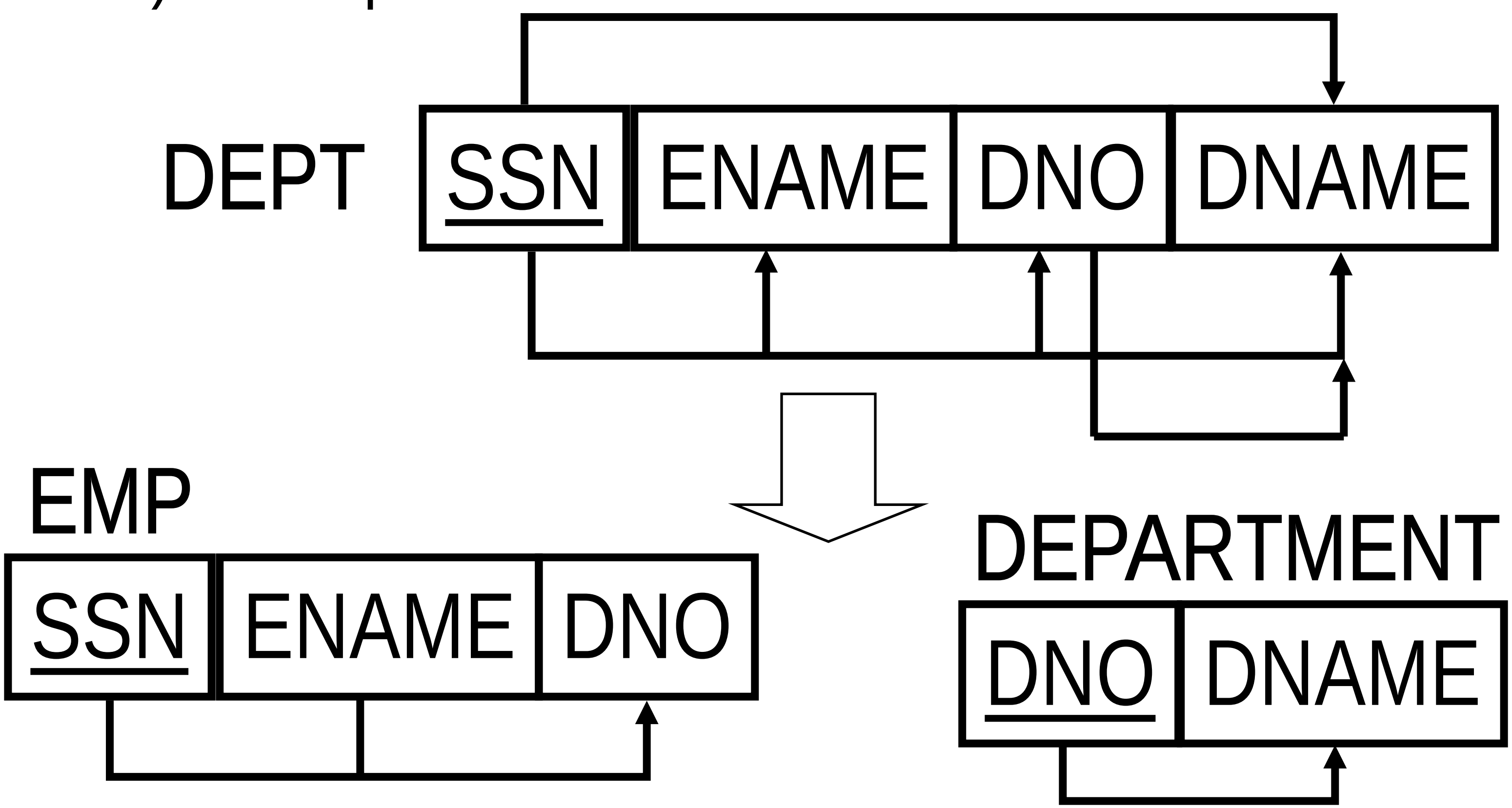


Third Normal Form (3NF)

A relation schema is in 3NF if for all non trivial dependencies in F^+ are of the form $X \rightarrow A$ with either:

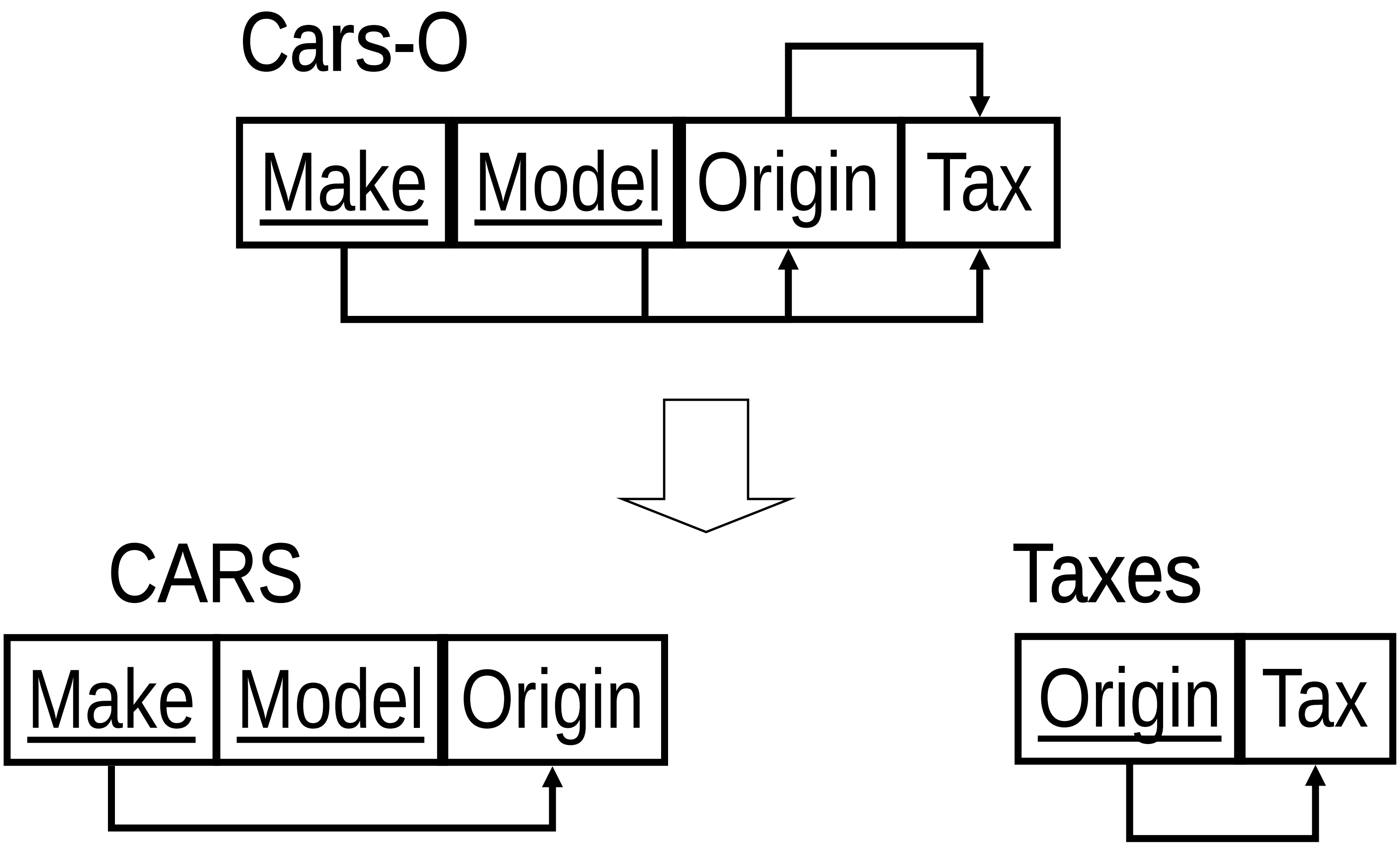
a). X is a superkey

b). A is a prime attribute



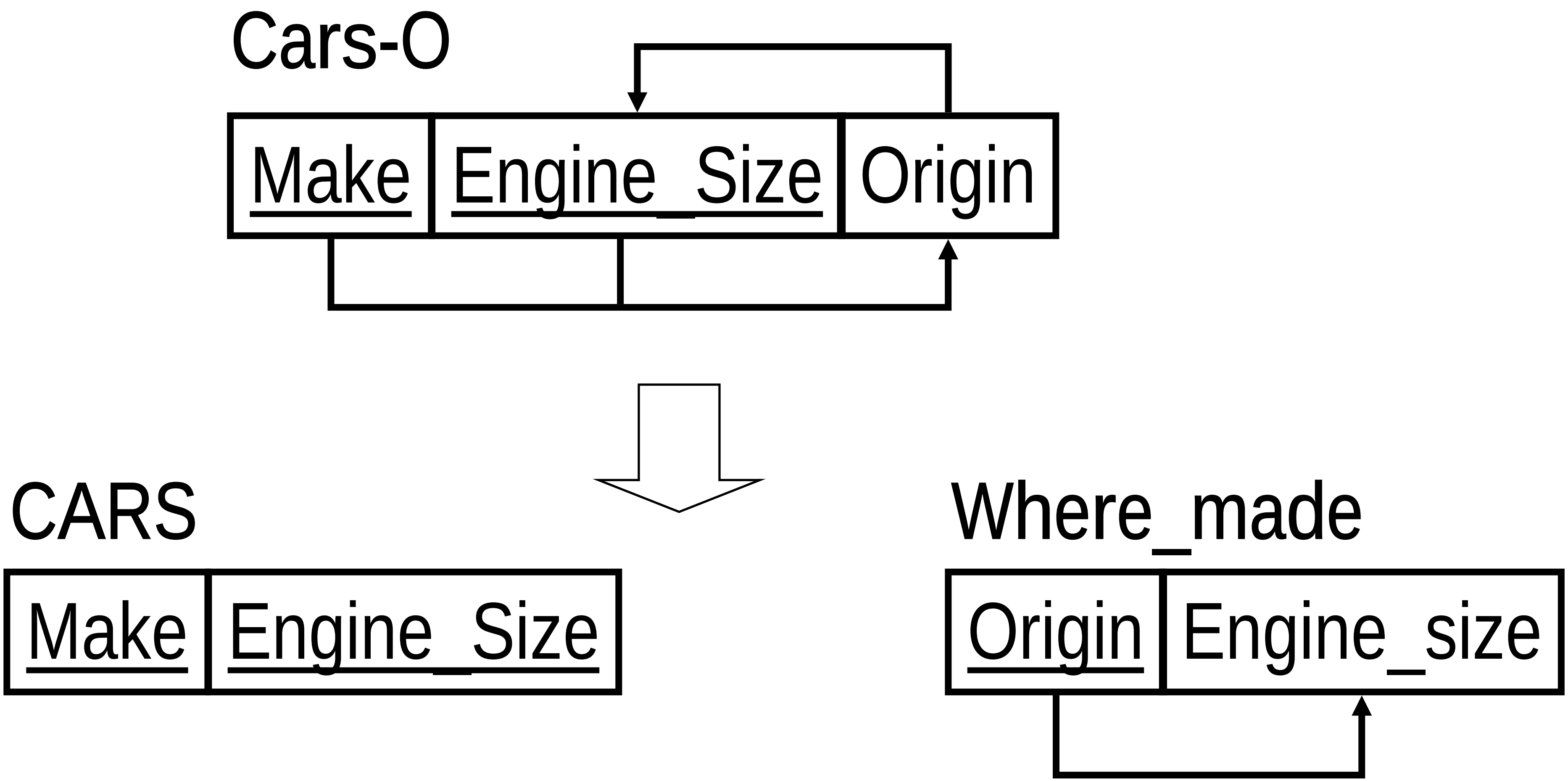
Third Normal Form (3NF)

Example:



Boyce-Codd Normal Form (BCNF)

A relation schema is in BCNF if for all non trivial dependencies in F^+ of the form $X \rightarrow A$; X is a superkey.



Summary

Normal Form	Test	Normalization (Remedy)
First (1NF)	Relation should have no non-atomic attributes or nested relations.	Form new relations for each non-atomic attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no non-key attributes should be functionally dependent on a part of 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 non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes.) That is, there should be no transitive dependency of a non-key attribute on the primary key.	Decompose and set up a relation that includes the non key attribute(s) that functionally determine(s) other non-key attribute(s).

Computing the closure of F F^+

- Closure F^+ of a set of FDs can be determined by using the inference rules IR1, IR2, and IR3 on F
- first, determine each set of attributes X that appears on left hand side (lhs) of some FD in F
- use IR1, IR2 and IR3 to determine the set of all attributes that are dependent on X
- this is called the <u>closure X^+ of X under F</u>

Algorithm for computing X^+

Algorithm: X^+ (closure of X under F)

$X^+ := X$;

while (changes to X^+) do

for each FD $Y \to Z$

begin

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

end

☞ does $F \models X \to Y$?

Covers and Equivalences of sets of FDs

- When are two sets of FDs equivalent?
  - ☞ We may have two sets of FDs F and G which may represent same constraints, but G might be much simpler than F; hence using G is to our advantage in enforcing the FDs in the database.
  - ☞ Given sets of FDs G and F, G and F are equivalent when $G^+ = F^+$

    G is said to cover F (and F is said to cover G) or G is a cover of F
  - ☞ G can cover F but F may not cover G; in this case $G^+ \neq F^+$
- <u>Given FD sets G and F, does F cover G?</u>
  - Calculate X^+ with respect to G for each FD $X \rightarrow Y$ in F
  - if X^+ includes Y for each X^+ then G covers F

Nonredundant Covers

- Given a set of FDs F, if a proper subset of F of F covers F, then F is redundant
  - ☞ we can remove some FD, say $X \rightarrow Y$ from F

<u>Algorithm:</u> Nonredundant Cover

G := F
For each FD $X \rightarrow Y$ in G do
    if $(X \rightarrow Y) \in \{ F — (X \rightarrow Y) \}^+$
        then F := $\{F — (X \rightarrow Y) \}$
G:= F;

Canonical (Minimal) Cover

- A set of FDs F_c is a <u>canonical (minimal) cover</u> if every FD in F_c satisfies:
  1. Each FD in F_c is simple (single attribute on the right hand side)
     - ☞ always possible using IR4 and IR5
  2. We cannot replace FD $X \rightarrow Y$ in F_c with an FD $Z \rightarrow A$, where $Z \subset X$ and still have a set of FDs that is equivalent to F_c
     - ☞ lhs of FDs do not have any <u>extraneous attributes.</u>
  3. No FD $X \rightarrow A$ is redundant
     - ☞ $\{F_c — (X \rightarrow A)\}^+ \neq F_c^+$

☞ There can be several canonical covers for a set of FDs

☞ can always find <u>at least one</u>

Additional Properties of Decompositions

- Normalization does not mean we have a good design
- also need to examine inter-relation properties

Definitions:

$R(A_1, A_2, \ldots, A_n)$ universal relation schema

$D = (R_1, R_2, \ldots, R_m)$ a decomposition of R

<u>Attribute preservation condition:</u>

$$\cup_{i=1,n} R_i = R$$

Need to consider inter relation properties: dependency preservation and lossless join.

Dependency Preservation Property

- Each FD $X \rightarrow Y$ should appear directly in some R_i or be inferred from the FDs in some R_i so as to allow efficient validation of the FDs
  - ☞ FDs are normally specified within a relation scheme
  - ☞ when do FDs go across relation schemes

$R(A, B, C, D)$ $\qquad A \rightarrow BCD; \quad BC \rightarrow AD; \quad D \rightarrow B$

$R_1(A, C, D)$ $\qquad A \rightarrow CD;$

$R_2(B, D)$ $\qquad D \rightarrow B$

Dependency Preservation Property

Definition:

Given a set of FDs F on R, the projection of F on R_i, denoted by $\underline{\pi F(R_i)}$ is the set of FDs $X \rightarrow Y$ in F^+ such that attributes in $X \cup Y$ are all contained in R_i.

- ☞ $D = (R_1, R_2, \ldots, R_m)$ is <u>dependency preserving with respect to F</u> if

$$((\pi_{R1}(F)) \cup (\pi_{R2}(F)) \cup \ldots \cup (\pi_{Rm}(F)))^+ = F^+$$

Lossless Join Property

STUDENT_COURSE			
SID	Name	Cnum	Sem
1	Jim	comp231	fall95
1	Jim	comp111	spring94
2	Alice	comp111	spring94
2	Alice	comp211	fall95

STUDENT		
SID	Name	Sem
1	Jim	fall95
1	Jim	spring94
2	Alice	spring94
2	Alice	fall95

COURSE	
Cnum	Sem
comp231	fall95
comp111	spring94
comp111	spring94
comp211	fall95

STUDENT_COURSE_SPURIOUS			
SID	Name	Cnum_	Sem
1	Jim	comp231	fall95
1	Jim	comp211	fall95
1	Jim	comp111	spring94
2	Alice	comp231	fall95
2	Alice	comp211	fall95
2	Alice	comp111	spring94

Lossless Join Property

- $D = (R_1, R_2, \ldots, R_m)$ has loss less join property with respect to F if for every relation instance r of R that satisfies F

$$\bowtie \pi_{R_i}(r) = r$$

Properties of Lossless Join Decomposition

<u>Property LJ1</u>

A decomposition $D = \{R_1, R_2\}$ has lossless join property with respect to set of FDs F iff either

- the FD $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ is in F^+ or
- the FD $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ is in F^+

☞ $(R_1 \cap R_2)$ are attributes common to both R_1 and R_2

☞ $R_1 - R_2$ are attributes in R_1 not in R_2.

Properties of Lossless Join Decomposition

<u>Property LJ2</u>

If $D = \{R_1, R_2, ---, R_m\}$ of R has the lossless join property with respect to F and $D_1 = \{Q_1, Q_2, \ldots , Q_k\}$ of R_i, has the lossless join property with respect to $\pi_F(R_i)$, then $D = \{R_1, R_2, \ldots , R_{i-1}, Q_1, Q_2, \ldots , Q_k, R_{i+1}, \ldots , R_m\}$ has the lossless join property with respect to F

☞ can replace schema R_i by its loss less decomposition $\{Q_1, Q_2, \ldots , Q_k\}$

Lossless Join and Dependency Preserving 3NF algorithm

1. Find a minimal set (cover) of FDs G equivalent to F
2. For each X of an FD $X \rightarrow A$ in G

   Create a relation schema R_i in D with the attributes $\{X \cup A_1 \cup A_2 \cup \ldots \cup A_k\}$ where the A_j's are all the attributes appearing in an FD in G with X as left hand side
3. If any attributes in R are not placed in any R_i, create another relation in D for these attributes

4 If none of them contain a candidate key K or R, create one more relation schema that contains attributes in K.

- ☞ Step 2 assures dependency preserving property
- ☞ Step 4 assures lossless join property

Lossless Join and Dependency Preserving 3NF algorithm

Problems:

Must find a minimal cover G for F

No efficient algorithm for finding a minimal cover

Several minimal covers can exist for F; the result of the algorithm can be different depending on which is chosen

Lossless Join Decomposition into BCNF

1. set D:={R}
2. While there is a relation schema Q in D that is not in BCNF do
   - choose a Q in D that is not in BCNF
   - find a FD $X \rightarrow Y$ that violates BCNF
   - replace Q in D by two relation schemas (Q - Y) and $(X \cup Y)$

Nulls and Decomposition

- theory of lossless join decomposition assumes that nulls are not allowed for join attributes
- allowing nulls causes problems of
  - <u>missing tuples</u>

    employee(Emp#, name, … Dno)

    department(Dno, …)
  - <u>dangling tuples</u>

    employee(emp#, …)

    works_on(emp#, dept#)

    department(dept#,…)

Problems with decomposition

- Need all FDs to get proper decomposition
- decomposition algorithms are not deterministic
    - algorithms are not popular in practice
    - much of database design is still intuitive and heuristic