An Introduction to Database Systems

chapter 12. Further Normalization I : 1NF, 2NF, 3NF, BCNF

spark@dblab.sogang.ac.kr

12.1 Introduction

☐ Running example

S(S#, SNAME, STATUS, CITY)
PRIMARY KEY (S#)
P(P#, PNAME, COLOR, WEIGHT, CITY)
PRIMARY KEY (P#)
SP(S#, P#, QTY)
PRIMATY KEY (S#, P#)
FOREIGN KEY (S#) REFERENCES S
FOREIGN KEY (P#) REFERENCES P

spark@dblab.sogang.ac.kr

12. 1 Introduction

- What happens if the design is changed in some way ?

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

S#	P#	QTY
S1	P1	300
S1	P2	300
S1	P3	300
S1	P4	300
S1	P5	300
S1	P6	300
S2	P1	300
S2	P2	300
S3	P2	300
S4	P2	300
S4	P4	300
S4	P5	300

SCP

redundancy

S#	CITY	P#	QTY
S1	London	P1	300
S1	London	P2	300
S1	London	P3	300
S1	London	P4	300
S1	London	P5	300
S1	London	P6	300
S2	Paris	P1	300
S2	Paris	P2	300
S3	Paris	P2	300
S4	London	P2	300
S4	London	P4	300
S4	London	P5	300

spark@dblab.sogang.ac.kr

12. 1 Introduction

☐ Redundancy in SCP

- **every SCP tuple for supplier S1 tells us S1 is located in London,**
- **every SCP tuple for supplier S2 tells us S2 is located in Paris, and so on.**

in turn, leads to several further problems

(ex) after an update, supplier S1 might be shown as being located in London by one tuple and in Amsterdam by another

spark@dblab.sogang.ac.kr

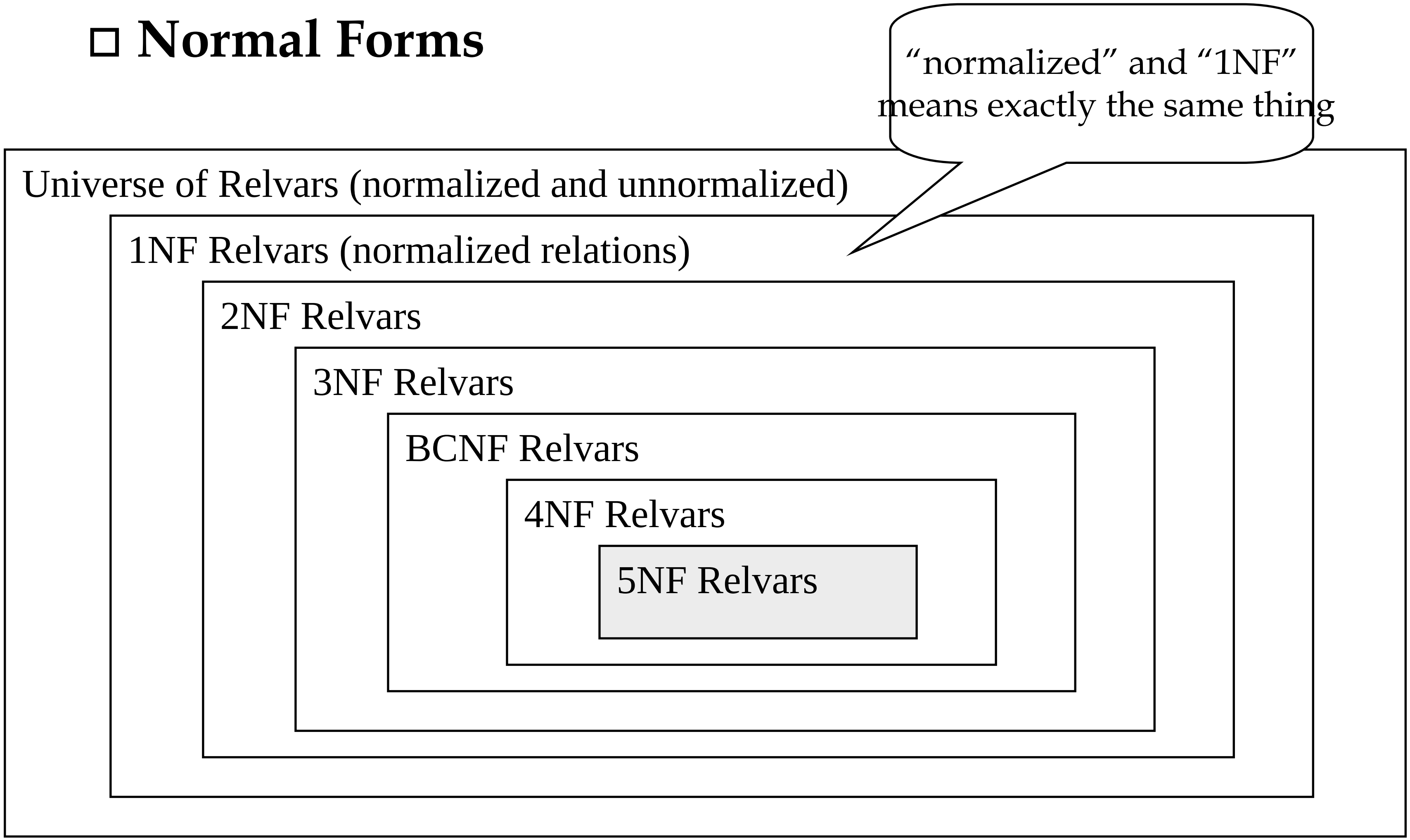
12. 1 Introduction

- **Good Design Principle**
  - **"one fact in one place"**
  - **i.e. avoid redundancy**
- **The subject of further normalization is essentially just a formalization of simple ideas like this one**
  - **a formalization, however, that does have very practical application in the area of database design**
- **Normalization**
  - **Given a relation, even though it is normalized, how the relations containing certain undesirable properties, can be converted t a more desirable form.**
- **Normal Forms**
  - **if it satisfies a certain specified set of constraints**
  - **ex) 1NF iff their (relvar) legal values are normalized relations.**

spark@dblab.sogang.ac.kr

12. 1 Introduction

□ Normal Forms



spark@dblab.sogang.ac.kr

12. 1 Introduction

□ Normalization Procedure

- the successive reduction of a given collection of relations to some more desirable form
- the procedure is reversible (ex, 2NF $\rightleftarrows$ 3NF)
  - no information is lost (nonloss or information-preserving)

□ Structure of chapter

- nonloss decomposition
- the crucial importance of FDs
- alternative decomposition

  (choosing the “best” decomposition of a given relation)

spark@dblab.sogang.ac.kr

12. 2 Nonloss Decomposition and Functional Dependencies

□ Nonloss(lossless decomposition)

- the procedure involves breaking down or decomposing a given relation into other relation and that the decomposition is required to be reversible
- the question of whether a given decomposition is nonloss is intimately bound up with the concept of functional dependence

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

nonloss

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

lossy

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

We cannot tell which supplier has which city

spark@dblab.sogang.ac.kr

12. 2Nonloss Decomposition and Functional Dependencies

- **What is it that makes the first decomposition nonloss and the other lossy**
  - the decomposition
    - a process of projection
  - figure (a)
    - “If we join relations SST and SC back together again, we get back the original relation S”
  - figure (b)
    - If we join SST and STC, we do not get back the original relation S
- **reversibility**
  - the original relation is equal to the join of its projections
  - projection : decomposition operator in the normalization procedure
  - join(natural join) : recomposition operator

spark@dblab.sogang.ac.kr

12.2 Nonloss Decomposition and Functional Dependencies

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

FDs

S# → STATUS
S# → CITY

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

S# → STATUS
S# → CITY

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

S# → STATUS

S# → CITY is lost

spark@dblab.sogang.ac.kr

12.2 Nonloss Decomposition and Functional Dependencies

- If R1 and R2 are projections of some relation R, and R1 and R2 include all of the attributes of R, what conditions have to be satisfied in order to guarantee that joining R1 and R2 takes us back to original R ?
  - Functional Dependencies

- ## Heath's Theorem :
  - Let R(A,B,C) be a relation, where A, B, and C are sets of attributes, If R satisfies the FD $A \rightarrow B$, then R is equal to the join of its projections on {A,B} and {A,C}.

spark@dblab.sogang.ac.kr

More on FDs

- **Left-irreducible FDs**
  - **a FD is said to be left-irreducible if its left-hand side is "not too big"**
  - **In SCP,**
    - **{S#, P#} → CITY ≠ S# → CITY**
    - **CITY is irreducibly dependent on S#, but not {S#,P#}**

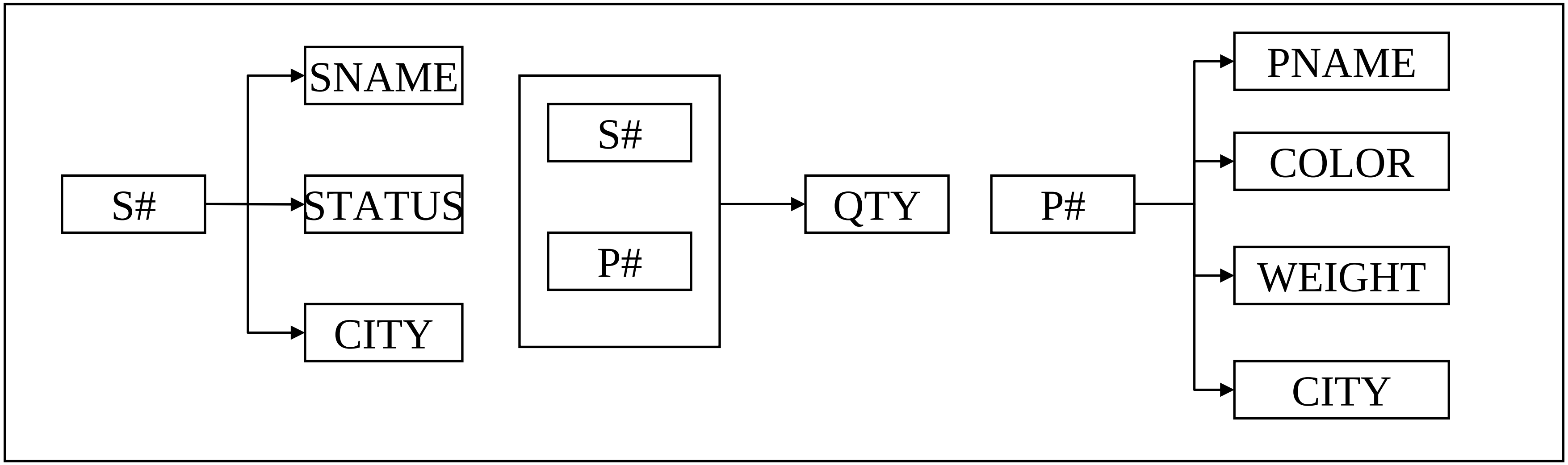
      (fully)

spark@dblab.sogang.ac.kr

More on FDs

□ FD-diagrams

- **Let R be a relation and let I be some irreducible set of FDs that apply to R**



• An arrow out of a candidate key(actually primary key)- for one value of each candidate key, there is always one value of everything else. (Those arrow can never be eliminated)
• It is if there are any other arrows that difficulties arise(The normalization procedure can be characterized as a procedure for eliminating arrows that are not arrows out of candidate keys)

spark@dblab.sogang.ac.kr

More on FDs

- **FDs are a semantic notion :**

  recognizing the FDs is part of the process of understanding what the data means(ex. FD S# $\rightarrow$ CITY)

  - **There is a constraint in the real world that the database represents, namely that each suppliers is located in precisely one city.**
  - **Since it is part of the semantics of the situation, that constraint must somehow be observed in the database**
  - **The way to ensure that is is so observed is to specify it in the database definition, so that the DBMS enforce it.**
  - **The way to specify it in the database definition is to declare the FD.**

spark@dblab.sogang.ac.kr

12. 3 First, Second and Third Normal Forms

□ Assumption for simplicity

- each relation has exactly one candidate key

□ Third Normal Form(informal definition)

A relation is in 3NF iff the nonkey attributes are

(a) mutually independent, and

(b) irreducibly dependent on the primary key

- A nonkey attribute is any attribute that does not participate in the primary key of the relation
- Two or more attributes are mutually independent if none of them is functionally dependent on any combination of the others

spark@dblab.sogang.ac.kr

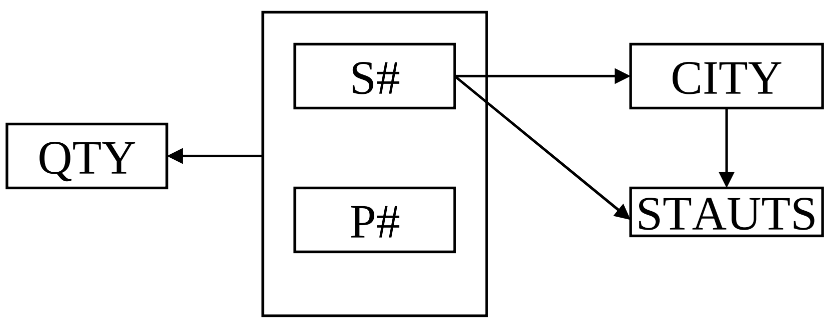
12.3 First, Second and Third Normal Forms

□ Normalization Procedure

- **First Normal Form**
  - A relation is in 1NF iff all underlying domains contain scalar values only.
  - Any normalized relations is in 1NF
  - FIRST(S#, STATUS, CITY, P#, QTY)

    PRIMARY KEY(S#, P#)

  FD diagram



spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

□ Sample tabulation of FIRST

FIRST

S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

Anomalies

- (Ex) supplier-city redundancy : FD S# $\rightarrow$ CITY)

Insert anomaly

- We cannot insert the fact that a particular supplier is located in a particular city until that supplier supplies at least one part.
- Ex) insertion of tuple <S5, , Athens, , > causes the violation of entity integrity rule
  - No primary key value may be null

Solution : Unbundling procedure "Place logically separate information into separate relations"

Delete anomaly

- If we delete the only FIRST tuple for a particular supplier, we destroy not only the shipment connecting that supplier to some part but also the information that supplier is located in a particular city.
- Ex) deletion of tuple <S3, 10, Paris, P2, 200> causes to lose the information that S3 is located in Paris.
- The real problem : FIRST contains too much information all bundled (delete a tuple $\Rightarrow$ delete too much)

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

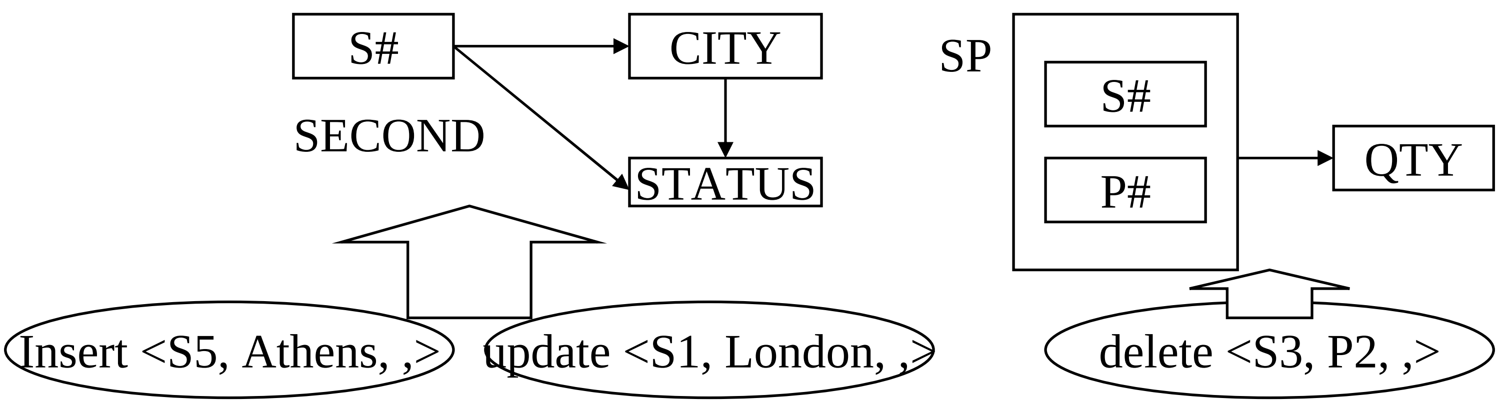
□ Update Anomaly

- **The city value for a given supplier appears in FIRST many times in general ⇒ redundancy**
- **ex) <S1, London > ⇒ <S1, Amsterdam>**
  - **search all tuples (S1)**
  - **inconsistency**

□ Solutions

- **decomposition**

FIRST ⇒ SECOND(S#, STATUS, CITY), SP(S#, P#, QTY)



spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

SECOND

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

- The effect of the decomposition of FIRST in SECOND and SP
  - has been to eliminate the dependencies that were not irreducible
- Mixing the two kinds of information in the same relation
  - was that caused the problems in the first place
- A relation is in 2NF iff
  - 1NF
  - every non-key attribute is irreducibly dependent on the primary key
    - ex) SECOND, SP : 2NF
    - ex) FIRST : 1NF, not 2NF

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

- The first step in the normalization procedure
  - is to take projections to eliminate non-irreducible functional dependencies
  - Given R,

    R(A, B, C, D)

      primary key (A,B)

      $A \rightarrow D$
  - replacing R by its two projections R1 and R2

    R1(A,D)

      primary key (A)

    R2(A, B, C)

      primary key (A, B)

      foreign key (A) references R1

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

- **SECOND relations**
  - lack of mutual independence among its nonkey attribute (FD CITY $\rightarrow$ STATUS)
  - transitive dependency
    - causes anomalies
    - If A $\rightarrow$ B and B $\rightarrow$ C, then A $\rightarrow$ C holds
    - ex) S# $\rightarrow$ CITY $\rightarrow$ STATUS

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

- **Anomaly of SECOND relation (CITY-STATUS redundancy)**
  - SECOND(S#, STATUS, CITY)
- **Insert anomaly**
  - we cannot insert the fact that a particular city has a particular status
  - ex)
    - Note state that Supplier in Rome must have a status of 50 until we have some supplier located in that city
- **Delete anomaly**
  - If we delete the only SECOND tuple for a particular city, we destroy not only the information for that supplier concerned but also the information that that city has that particular status.
  - Ex) Deletion <S5, 30, Athens > causes to loss (Athens : 30)
  - the problem is bundling
    - solution : unbundling (supplier information, city information)

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

Update anomaly

- the status for a given city appears in SECOND many times in general.
- Ex) update < London, 20 > $\Rightarrow$ <London, 30>
  - search all tuples
  - inconsistency

Solution

- SECOND(S#, CITY, STATUS) $\Rightarrow_{\text{decomposition}}$ SC(S#, CITY),
  CS(CITY, STATUS)

$$\boxed{\text{S\#}} \longrightarrow \boxed{\text{CITY}} \qquad \boxed{\text{CITY}} \longrightarrow \boxed{\text{STATUS}}$$

The effect of the decomposition is to eliminate the transitive dependencies

12.3 First, Second and Third Normal Forms

□ Sample tabulation of SC and CS

SC

S#	CITY
S1	London
S2	Paris
S3	Paris
S4	London
S5	Athens

CS

CITY	STATUS
Athens	30
London	20
Paris	10
Rome	50

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

Third Normal Form

- only one candidate key (primary key)
- A relation is 3NF iff
  - 2NF
  - Every nonkey attribute is nontransitively(no mutual dependency) dependent on the primary key.
- SC, CS : 3NF
- SECOND : 2NF, not 3NF

spark@dblab.sogang.ac.kr

12.3 First, Second and Third Normal Forms

- The second step in the normalization procedure
  - is to take projections to eliminate transitive dependencies
  - Given a R

    R(A, B, C)

    primary key (A)

    $B \rightarrow C$
  - replacing R by its two projections R1 and R2

    R1 (B, C)

    primary key (B)

    R2(A, B)

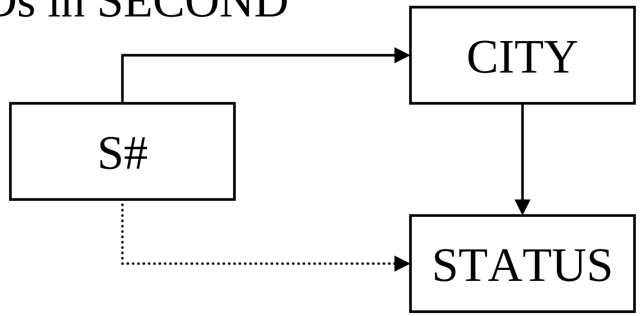
    primary key(A)

    foreign key (B) references R1
  - normalization :
    - a matter of data semantics, a matter of the data value

spark@dblab.sogang.ac.kr

12.4 Dependency Preservation

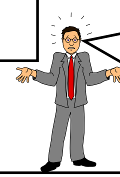
FDs in SECOND



Two projections are independent (updates can be made to either one without regard for the other)
The join of the two projections after update will always be a valid SECOND

(A) SC(S#, CITY), CS(CITY, STATUS)

(B) SC(S#, CITY), SS(S#, STATUS)



Two projections are not independent (updates to either of the two projection must be monitored to ensure that FD CITY → STATUS is not violated)
ex) <S3, Paris > → <S3, London>
<London, 20>

B is less satisfactory than A
(still not possible to insert the information that a particular city has a particular status unless some supplier is located in that city

spark@dblab.sogang.ac.kr

12.4 Dependency Preservation

□ Basic Problem

- in (B), FD CITY $\rightarrow$ STATUS : inter-relation constraint
  - maintained by procedural application code
- in (A), FD S# $\rightarrow$ STATUS : transitive (inter-relation constraint)
  - enforced automatically if the two intra-relation constraints S# $\rightarrow$ CITY and CITY $\rightarrow$ STATUS are enforced
  - it is simple (primary key constraint)
- the concept of independent projection :
  - a guideline for choosing a particular decomposition when there is more than one possibility

spark@dblab.sogang.ac.kr

12.4 Dependency Problem

□ Rissanen's theorem

- the projection R1 and R2 of R are independent iff
  - every FD in R is a logical consequence of those in R1 and R2,
  - the common attribute of R1 and R2 form a candidate key for at least one of the pair
- in (A),
  - common attribute CITY : primary key for CS
  - every FD in SECOND either appears in one of the projections or is a logical consequence of those that do
  - two projections are independent
- in (B),
  - common attribute S# is a candidate key for both
  - CITY $\rightarrow$ STATUS cannot be deduced from the FDs in those projection
  - two projections are not independent
- in another possibility, SECOND $\Rightarrow$ SS(S#, STATUS), CS(CITY, STATUS)
  - nonloss
  - not a valid decomposition

spark@dblab.sogang.ac.kr

12.4 Dependency Preservation

□ Atomic

- A relation that cannot be decomposed into independent projections is said to be *atomic*

□ Dependency Preservation

- the idea that the normalization procedure should decompose relations into projections that are independent
  - $R \rightarrow R_1, R_2, \ldots., R_n$ by normalization procedure
  - FD S for R, FD $S_1, S_2, \ldots.., S_n$ for $R_1, R_2, \ldots, R_n$
  - each FD in $S_i \Rightarrow$ attributes of R_i
    - enforcing $(S_1, S_2, \ldots, S_n)$: equivalent to enforcing S of R
    - dependency preserving
  - $S' = S_1 \cup S_2 \ldots S_n$, $S' \approx$ equivalent to S^+ ?

spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

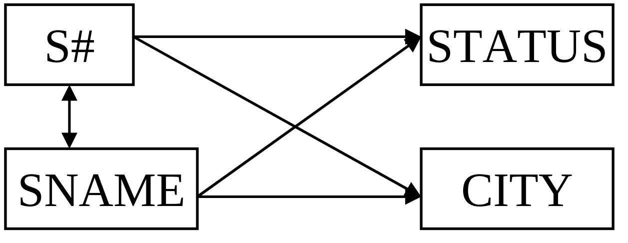
- Definition of 3NF did not adequately deal with
  - the case of relation that had two(or more) candidate keys such that
    - the two candidate keys were composite, and
    - they overlapped(i.e., had at least one attribute in common)
- Boyce/Codd normal form
  - A relation is in BCNF iff
    - every nontrivial, left-irreducible FD has a candidate key as its determinant
    - A relation is in BCNF iff the only determinant are candidate keys
      - stronger than 3NF (more restrictive)
  - ex)
    - FIRST, SECOND : not 3NF, not BCNF
      - FIRST : 3 determinants (S#, CITY, {S#, P#}) , but only {S#,P#} is the candidate key
      - SECOND : 2 determinants (S#, CITY), but s# is candidate key
    - SP, SC, CS : 3NF, BCNF
      - in each case the candidate key is the only determinant in the relation.

spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

(1) two disjoint(non-overlapping) candidate keys

- **ex) S(S#, SNAME, STATUS, CITY)**
  - **S# , SNAME : unique**
  - **STATUS, CITY : mutually independent**
  - **BCNF**
    - **only determinants are candidate keys**



S(S#, SNAME, STATUS, CITY)
candidate key (S#)
candidate key(SNAME)

spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

(2) the candidate keys overlap

- **ex) SSP(S#, SNAME, P#, QTY)**
  - **SNAME : unique**
  - **candidate keys ; (S#, P#), (SNAME, P#)**
  - **determinants : S#, SNAME, (S#, P#), (SNAME, P#)**
  - **3NF, but not BCNF**

SSP

S#	SNAME	P#	QTY
S1	Smith	P1	300
S1	Smith	P2	200
S1	Smith	P3	400
S1	Smith	P4	200
…	…	…	…

redundancy

$\Downarrow$

anomaly

Solution :

SSP ⇒ SS(S#, SNAME), SP(S#, P#, QTY)(or SP(SNAME, P#, QTY))

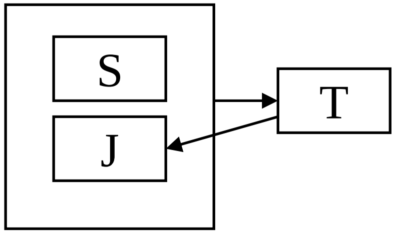
spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

- **SS-SP design is better than SSP design ?**
  - **Formalized common sense**
  - **mechanize the principles**
- **The Relation SJT(S, J, T) for student, subject, and teacher**

S	J	T
Smith	Math	Prof. White
Smith	Physics	Prof. Green
Jones	Math	Prof. White
Jones	Physics	Prof. Brown

Meaning :
Student S is taught subject J by teacher T



- **Semantic constraints**
  - **For each subject, each student of that subject is taught by only one teacher**
  - **Each teacher teaches only one subject**
  - **Each subject is taught by several teachers**

spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

- **In SJT**
  - two overlapping candidate keys
    - {S, J}, {S, T}
  - 3NF, but not BCNF
  - update anomaly (I/ D/ U)
    - (D) If we delete the information that Jones is studying physics, we cannot do so without at the same time losing the information that Prof. Brown teaches physics.
    - Because T is a determinant but not a candidate key
- **Solution**
  - SJT $\Rightarrow$ ST(S,T), TJ(T,J) : BCNF

spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

- **Decomposition**
  - **solve certain problems, introduce different ones**
    - **two projections ST and TJ are not independent in Rissanen's sense SJ $\rightarrow$ T cannot be deduced from T $\rightarrow$ J**
    - **two projections cannot be independently updated**
    - **ex)**

    insert < Smith, Brown > : rejection

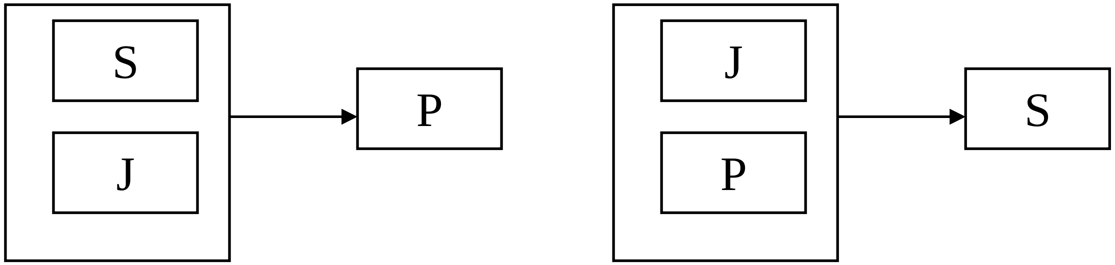
    - **two objectives**
      - **decomposing a relation into BCNF components**
      - **decomposing it into independent components**

      can occasionally be in conflict

spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

- Ex) EXAM(S, J, P) for S(student), J(Subject), P(Position)
  - the meaning of EXAM tuples {S:s, J;j, P:p}
    - student s was examined in Subject j and achieved position p in the class list
  - semantic constraints
    - There is no ties : that is, no two student obtained the same position in the same subject
  - Functional dependencies



Determinants {S, J}, {J,P} are candidate keys.
BCNF : update anomalies do not occur

Overlapping candidate keys do not necessarily lead to problems of the kind

spark@dblab.sogang.ac.kr

12.5 Boyce/Codd Normal Form

- **BCNF**
  - eliminate certain problem cases that could occur under the old 3NF
  - no reference to the concepts of 1NF, 2NF, primary key, or transitive dependence

spark@dblab.sogang.ac.kr

12.6 A Note on Relation-Valued Attributes

- Relvars can have relation-valued attributes
- From the point of view of database design, it tends to be asymmetric

→ Practical problems

<table>
<tr><th>S#</th><th>PQ</th></tr>
<tr><td>S1</td><td><table><tr><th>P#</th><th>QTY</th></tr><tr><td>P1
P2
…
P6</td><td>300
200
…
100</td></tr></table></td></tr>
<tr><td>S2</td><td><table><tr><th>P#</th><th>QTY</th></tr><tr><td>P1
P2</td><td>300
200</td></tr></table></td></tr>
<tr><td>…</td><td>…</td></tr>
<tr><td>S5</td><td><table><tr><th>P#</th><th>QTY</th></tr><tr><td></td><td></td></tr></table></td></tr>
</table>

spark@dblab.sogang.ac.kr

12.6 A Note on Relation-Valued Attributes

- **The symmetric queries**
  1. **Get S# for suppliers who supply part P1**
  2. **Get P# for parts supplied by supplier S1**

→ **Different formulation**
  1. **(SPQ WHERE P# ('P1') IN PQ { P# }) { S# }**
  2. **((SPQ WHERE S# = S# ('S1')) { PQ }) { P# }**

ex) two update operations

1. Create a new shipment for supplier S6, part P5, quantity 500
2. Create a new shipment for supplier S2, part P5, quantity 500

spark@dblab.sogang.ac.kr

12.6 A Note on Relation-Valued Attributes

- **relvar SP**
  - There is no qualitative difference between these two updates

- **relvar SPQ**
  - Two updates differ in kind significantly

```
1.  INSERT  INTO  SPQ  RELATION
          { TUPLE {  S#  S#  ( 'S6' ) ,
                          PQ  RELATION  { TUPLE { P# ( 'P5' ),
                                                     QTY QTY ( 500 ) } } } } ;

2.  UPDATE  SPQ  WHERE  S# = S#  ( 'S2' )
      INSERT  INTO  PQ  REALTION  {  TUPLE { P# ( 'P5' ),
                                               QTY  QTY ( 500 ) } } ;
```


spark@dblab.sogang.ac.kr

12.6 A Note on Relation-Valued Attributes

- **Relvars without relation-valued attributes are usually to be prefered**

  ← Simpler logical structure

- **Cases where a relation-valued attribute does make sense**

ex) catalog relvar RVK

```
VAR  RVK  BASE  RELATION
        { RVNAME  NAME,  CK  RELATION  { ATTNAME NAME } }
          KEY  { RVNAME, CK } ;
```


spark@dblab.sogang.ac.kr

12.6 A Note on Relation-Valued Attributes

<table>
<tr><th>RVNAME</th><th>CK</th></tr>
<tr><td>S</td><td><table><tr><th>ATTNAME</th></tr><tr><td>S#</td></tr></table></td></tr>
<tr><td>SP</td><td><table><tr><th>ATTNAME</th></tr><tr><td>S#</td></tr><tr><td>P#</td></tr></table></td></tr>
<tr><td>MARRIAGE</td><td><table><tr><th>ATTNAME</th></tr><tr><td>HUSBAND</td></tr><tr><td>DATE</td></tr></table></td></tr>
<tr><td>MARRIAGE</td><td><table><tr><th>ATTNAME</th></tr><tr><td>DATE</td></tr><tr><td>WIFE</td></tr></table></td></tr>
<tr><td>MARRIAGE</td><td><table><tr><th>ATTNAME</th></tr><tr><td>WIFE</td></tr><tr><td>HUSBAND</td></tr></table></td></tr>
</table>

spark@dblab.sogang.ac.kr