Database Management Systems

Winter 2002

CMPUT 391: Database Design Theory

Dr. Osmar R. Zaïane

University of Alberta

Chapter 15 of Textbook

Course Content

- Introduction
- **Database Design Theory**
- Query Processing and Optimisation
- Concurrency Control
- Data Base Recovery and Security
- Object-Oriented Databases
- Inverted Index for IR
- Spatial Data Management
- XML
- Data Warehousing
- Data Mining
- Parallel and Distributed Databases

Objectives of Lecture 2

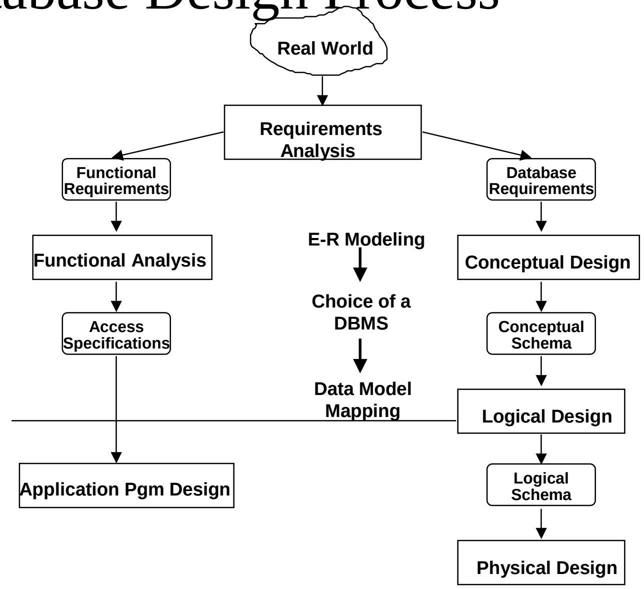
Database Design Theory

- Understand some limitations of Entity Relationship Model
- Introduce Functional Dependencies in Relational Database Design
- Introduce Decomposition and Normalization

Database Design Theory

- Database Design Process
- Redundancy Anomalies
- Functional Dependencies
- Armstrong Axioms and Derived Rules
- Normal Forms
- Decomposition of Relations

Database Design Process



Choices of DBMS

- Technical Factors
  - data model, user interfaces
  - programming languages,
  - application development tools
  - storage structures, access methods
- Economic Factors
  - software, hardware, database
  - acquisition, maintenance
  - personnel, training, operation
- Political Factors

Logical Database Design

- System independent phase
  - obtain a desirable database scheme in the database model of the chosen database management system
- System dependent phase
  - adjust the database scheme obtained in the previous phase to conform to the chose database management system
  - DDL statements

Physical Database Design

- Purpose
  - to specify the appropriate file structures and indexes
- Criteria
  - efficiency
- Approach
  - analyzing the database queries and transactions, including expected frequency
  - specifying the general user requirements
- Guideline
  - speeding natural join operations
  - separate read-only and update transactions
  - index files for search and hashing for random access
  - focus on attributes used most frequently

Implementation

- Coding
  - DDL for database scheme
  - SDL for physical scheme
  - develop application programs
- Testing
- Operation and Maintenance

Bad Database Design

- Pitfalls in Relational Database Design
  - Repetition of information
  - Inability to represent certain information
  - Loss of information

Consider the following relation schemes:

Branch = (branch-name, assets, branch-city)
Borrow = (branch-name, loan-number, customer-name, amount)
Deposit = (branch-name, account-number, customer-name, amount)

Repetition of Information

Consider an alternative design with the single scheme below

Lending = (branch-name, assets, branch-city, loan-number, customer-name, amount)

branch-name	assets	branch-city	loan-number	customer-name	amount
Downtown	9000	Edmonton	17	Jones	1000
Downtown	9000	Edmonton	93	Smith	2000
Downtown	9000	Edmonton	93	Hays	2900
Redwood	21000	Edmonton	23	Jackson	1200
Redwood	21000	Edmonton	23	Smith	2000
SUB	17000	Edmonton	19	Hays	2900
SUB	17000	Edmonton	19	Turner	500
SUB	17000	Edmonton	19	Brooks	2200

What if a customer wishes to open an account but not a loan ?

Repetition of Information

Consider an alternative design

Branch-Cust = (branch-name, assets, branch-city, customer-name)
Cust-Loan = (customer-name, loan-number, amount)

branch-name	assets	branch-city	customer-name
Downtown	9000	Edmonton	Jones
Downtown	9000	Edmonton	Smith
Downtown	9000	Edmonton	Hays
Redwood	21000	Edmonton	Jackson
Redwood	21000	Edmonton	Smith
SUB	17000	Edmonton	Hays
SUB	17000	Edmonton	Turner
SUB	17000	Edmonton	Brooks

customer	loan	amount
Jones	17	1000
Smith	93	2000
Hays	93	2900
Jackson	23	1200
Smith	23	2000
Hays	19	2900
Turner	19	500
Brooks	19	2200

What will happen if we do a join ?

Database Design Theory

- Database Design Process
- Redundancy Anomalies
- Functional Dependencies
- Armstrong Axioms and Derived Rules
- Normal Forms
- Decomposition of Relations

The Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - ***redundant storage, insert/delete/update anomalies***
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: <u>*decomposition*</u> (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Database Design Theory

- Database Design Process
- Redundancy Anomalies
- Functional Dependencies
- Armstrong Axioms and Derived Rules
- Normal Forms
- Decomposition of Relations

Functional Dependencies (FDs)

- A <u>functional dependency</u> $X \rightarrow Y$ holds over relation R if, for every allowable instance *r* of R:
  - $t1 \in r$, $t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that $K \rightarrow R$
  - However, $K \rightarrow R$ does not require K to be *minimal*!

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (<u>ssn</u>, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)
- <u>*Notation*</u>: We will denote this relation schema by listing the attributes: SNLRWH
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  - *ssn* is the key: S $\longrightarrow$ SNLRWH
  - *rating* determines *hrly_wages*: R $\longrightarrow$ W

Example (Contd.)

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- Problems due to R$\longrightarrow$W :
  - <u>*Update anomaly*</u>: Can we change W in just the 1st tuple of SNLRWH?
  - <u>*Insertion anomaly*</u>: What if we want to insert an employee and don't know the hourly wage for his rating?
  - <u>*Deletion anomaly*</u>: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

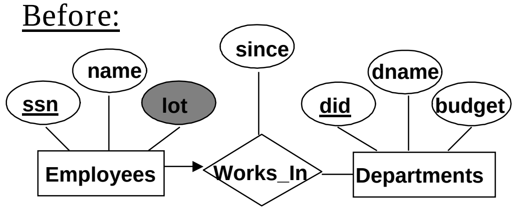
R	W
8	10
5	7

Wages

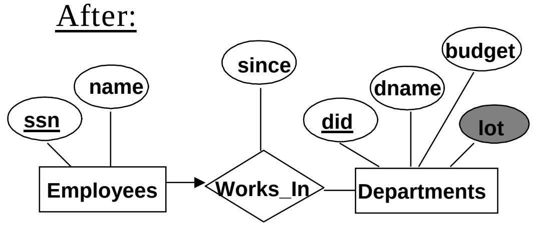
Refining an ER Diagram

- 1st diagram translated:
  Workers(S,N,L,D,C)
  Departments(D,M,B)
  - Lots associated with workers.
- Suppose all workers in a dept are assigned the same lot:
  D $\longrightarrow$ L
- Redundancy; fixed by:
  Workers2(S,N,D,C)
  Dept_Lots(D,L)
  Departments(D,M,B)
- Can fine-tune this:
  Workers2(S,N,D,C)
  Departments(D,M,B,L)

<u>Before:</u>



<u>After:</u>



Database Design Theory

- Database Design Process
- Redundancy Anomalies
- Functional Dependencies
- Armstrong Axioms and Derived Rules
- Normal Forms
- Decomposition of Relations

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn* $\longrightarrow$ *did*, *did* $\longrightarrow$ *lot* implies *ssn* $\longrightarrow$ *lot*
- An FD *f* is ***implied by*** a set of FDs *F* if *f* holds whenever all FDs in *F* hold.
  - F^+ = *closure of F* is the set of all FDs that are implied by *F*.
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - ***Reflexivity***: If $Y \subseteq X$, then $X \longrightarrow Y$
  - ***Augmentation***: If $X \longrightarrow Y$, then $XZ \longrightarrow YZ$ for any Z
  - ***Transitivity***: If $X \longrightarrow Y$ and $Y \longrightarrow Z$, then $X \longrightarrow Z$
- These are *sound* and *complete* inference rules for FDs!

Reflexivity

- If $Y \subseteq X$, then $X \rightarrow Y$

- $R=(\overbrace{A,B,\underbrace{C,D}_{Y}}^{X},E)$

$t_1 = (a_1,b_1,c_1,d_1,e_1)$
$t_2 = (a_2,b_2,c_2,d_2,e_2)$
$\pi_X(t_1) = \pi_X(t_2) \Rightarrow$
$a_1 = a_2, b_1 = b_2, \underbrace{c_1 = c_2, d_1 = d_2}$

$\pi_Y(t_1) = \pi_Y(t_2) \Leftarrow$

Augmentation

- If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z

- $R=(\overbrace{A,B}^{X},\underbrace{C,D}_{Y},\overbrace{E}^{Z})$

$t_1 = (a_1,b_1,c_1,d_1,e_1)$
$t_2 = (a_2,b_2,c_2,d_2,e_2)$
$\pi_{XZ}(t_1) = \pi_{XZ}(t_2) \Rightarrow$
$a_1 = a_2, b_1 = b_2, e_1 = e_2$
Since $X \longrightarrow Y$ and $e_1 = e_2$
then $c_1 = c_2, d_1 = d_2, e_1 = e_2$
$\pi_{YZ}(t_1) = \pi_{YZ}(t_2)$

Transitivity

- If $X \rightarrow Y$, and $Y \rightarrow Z$ then $X \rightarrow Z$

- $R=(\overbrace{A,B}^{X},\underbrace{C,D}_{Y},\overbrace{E}^{Z})$

$t_1 = (a_1,b_1,c_1,d_1,e_1)$
$t_2 = (a_2,b_2,c_2,d_2,e_2)$
assume $X \longrightarrow Y$ and $Y \longrightarrow Z$
$\pi_X(t_1) = \pi_X(t_2) \Rightarrow$
$a_1 = a_2, b_1 = b_2$
Since $X \longrightarrow Y$ then $c_1 = c_2, d_1 = d_2$
$\Rightarrow \pi_Y(t_1) = \pi_Y(t_2)$
Since $Y \longrightarrow Y$ then $e_1 = e_2$
$\Rightarrow \pi_Z(t_1) = \pi_Z(t_2)$

Reasoning About FDs (Contd.)

- Couple of additional rules (that follow from Armstrong Axioms):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Example: Contracts(*cid*,*sid*,*jid*,*did*,*pid*,*qty*,*value*), and:
  - C is the key: $C \rightarrow CSJDPQV$
  - Project purchases each part using single contract: $JP \rightarrow C$
  - Dept purchases at most one part from a supplier: $SD \rightarrow P$
- $JP \rightarrow C$, $C \rightarrow CSJDPQV$ imply $JP \rightarrow CSJDPQV$
- $SD \rightarrow P$ implies $SDJ \rightarrow JP$
- $SDJ \rightarrow JP$, $JP \rightarrow CSJDPQV$ imply $SDJ \rightarrow CSJDPQV$

Closure of a Set of Functional Dependencies

- It is not sufficient to consider the given set of functional dependencies
- We need to consider ALL functional dependencies that hold.
- Given F, a set of functional dependencies, the set of all functional dependencies logically implied by F are called the closure of F denoted by F^+

Reasoning About FDs (Contd.)

- Computing the closure of a set of F of FDs can be expensive. (Size of closure F^+ is exponential in # attrs!)
- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F. An efficient check:
  - Compute *attribute closure* of X (denoted X^+) wrt F:
    - Set of all attributes A such that $X \rightarrow A$ is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X^+
- Does $F = \{A \rightarrow B,\ B \rightarrow C,\ C\,D \rightarrow E\}$ imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Computing the Attribute Closure

- The attribute closure X^+ of a set of attributes with respect to a given set of functional dependencies F is the set of all attributes A such that $X \rightarrow A$ holds.
- To check whether an FD $X \rightarrow Y$ holds wrt F, we just have to check whether $Y \subseteq X^+$ (no need to compute F^+)
- Algorithm for Attribute Closure:

closure := X;
while (changes in *closure*) **do**
  foreach functional dependency $U \rightarrow V$ **do**
    if $U \subseteq$ closure **then** *closure* := *closure* $\cup$ V;

Database Design Theory

- Database Design Process
- Redundancy Anomalies
- Functional Dependencies
- Armstrong Axioms and Derived Rules
- $\boxed{\text{Normal Forms}}$
- Decomposition of Relations

Normal Forms

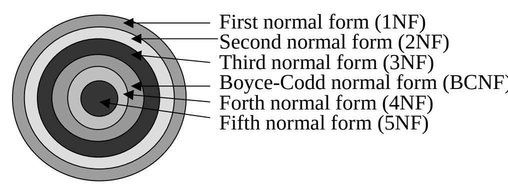
- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given $A \longrightarrow B$: Several tuples could have the same A value, and if so, they'll all have the same B value!

Desired Normal Forms

- The normalization process was first introduced by Codd in 1972. It takes a relation schema through a series of tests and verifies whether it satisfies certain normal forms.
- Initially, Codd introduced 3 normal forms 1NF, 2NF and 3NF but later Boyce and Codd introduced a stronger definition for 3NF called Boyce-Codd Normal Form (BCNF).
- There are also 4NF and 5NF based on Multivalued Dependencies.

Normal Form Tests

- 1NF: Relation should have no non-atomic attributes or nested relations
- 2NF: Relation where the primary key contains multiple attributes and no nonkey attribute should be FD on a a part of the primary key.
- 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.
- A relation in 3NF is also in 2NF and a relation in 2NF is also in 1NF.



1NF Example

StuID	Activity	Fee
100	Skiing	200
100	Golf	100
150	Swimming	65
175	Squash	50
175	Swimming	65
200	Swimming	65
200	Golf	100

- Activity relation is in 1NF (each attribute has one single value by tuple)
- Key= StuID+Activity
- Deletion and Insertion anomalies
- Relation contains 2 themes
- Fee is dependent on part of the key (Activity)
- Split the relation into 2 relations with one theme each.
- 2NF: a non-key attribute can't be dependent on part of the key but must be dependent on the whole key

2NF Example

StuID	Activity
100	Skiing
100	Golf
150	Swimming
175	Squash
175	Swimming
200	Swimming
200	Golf

Activity	Fee
Skiing	200
Golf	100
Swimming	65
Squash	50

- No non-key attribute is dependent on part of a key
- Note that in this case the keys are just one attribute for both relations ➔ automatically in 2NF

2NF Example 2 and 3NF

StuID	Residence	Fee
100	Lister	$4907
150	Pembina	$4587
200	Lister	$4907
250	HUB	$3600
300	Lister	$4907

- Key = StuID ➔ 2NF
- StuID $\rightarrow$ (Residence, Fee)
- StuID $\rightarrow$ Residence but also Residence $\rightarrow$ Fee (transitive dependency)
- Delete StuID 150, add (Fac. St-Jean, $2923) ➔ modification anomalies.
- No non-key attribute is dependent on non-key attribute/s (transitive dependency).
- 3NF is in 2NF+ no transitive dependencies

StuID	Residence

Residence	Fee

Problems with 3NF (Example)

StuID	Major	Faculty
100	Math	Pavol
150	Physics	Tico
200	Math	Pavol
250	Math	Calvert
300	Physics	Popovic
300	Biology	Wong

- Key = StuID + Major
- Candidate key = StuID+Faculty
- Faculty $\rightarrow$ Major
- Student can have many majors and student can have many advisors ➔ StuID $\not\rightarrow$ Major and StuID $\not\rightarrow$ Faculty
- 1NF and 2NF (all attrib part of key), 3NF (no transitive dependencies)
- Delete StuID 300, add (Dahl advises statistics) ➔ modification anomalies.
- Determinant (Faculty) is not part of a key ➔ not BCNF

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs *F* is in BCNF if, for all X $\rightarrow$ A in F^+
  - A $\in$ X (called a *trivial* FD), or
  - X contains a key for R.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

Third Normal Form (3NF)

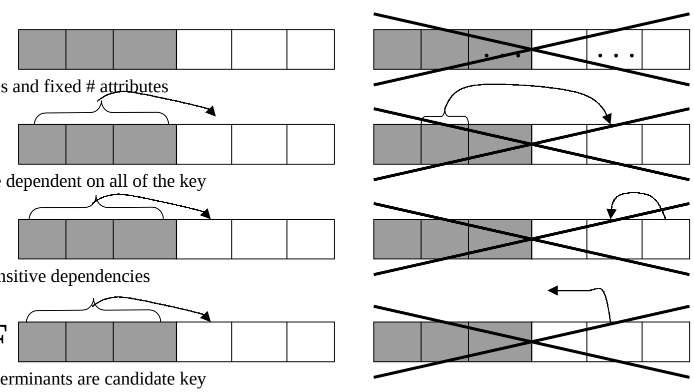
- Relation R with FDs *F* is in 3NF if, for all X $\rightarrow$ A in F^+
  - A $\in$ X (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key for R.
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

What Does 3NF Achieve?

- If 3NF violated by X $\rightarrow$A, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.
    - There is a chain of FDs K $\longrightarrow$ X $\longrightarrow$ A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
- But: even if reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC, S $\longrightarrow$ C, C $\longrightarrow$ S is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

Normal Form Conditions Revised

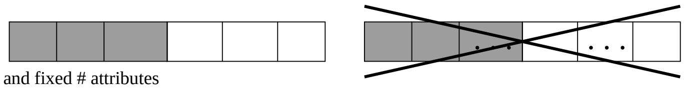
- 1NF
  No nested tables and fixed # attributes
- 2NF
  All non-key are dependent on all of the key
- 3NF
  2NF and no transitive dependencies
- BCNF
  3NF and all determinants are candidate key



1NF and 2NF Revised

1NF: a relation in which the intersection of each row and column contains one and only one value.
i.e. Tables should have atomic values only.

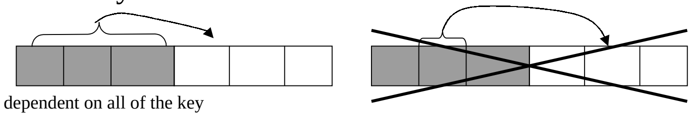
1NF



No nested tables and fixed # attributes

2NF: a relation in 1NF and every non primary key attribute is fully functionally dependent on the primary key.
i.e. There are no non-key attributes with partial key dependencies in any table.

2NF

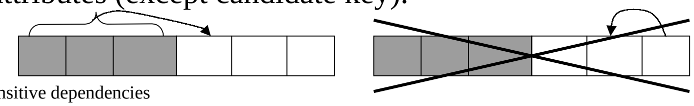


All non-key are dependent on all of the key

3NF and BCNF Revised

3NF: a relation in 2NF and in which no non-primary key attribute is transitively dependent on the primary key.
i.e. There are no non-key attributes with dependencies on other non-key attributes (except candidate key).

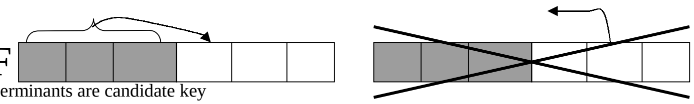
3NF



2NF and no transitive dependencies

BCNF: a relation in 3NF and in which there are no dependencies of part of the compound key on another attribute.
i.e. Every determinant is a candidate key.

BCNF



3NF and all determinants are candidate key

Database Design Theory

- Database Design Process
- Redundancy Anomalies
- Functional Dependencies
- Armstrong Axioms and Derived Rules
- Normal Forms
- Decomposition of Relations

Decomposition of a Relation Scheme

- Suppose that relation R contains attributes *A1 ... An.* A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- E.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs S $\rightarrow$SNLRWH and R $\rightarrow$W
  - Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and RW
- The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Problems with Decompositions

- There are three potential problems to consider:
  - ❖ Some queries become more expensive.
    - e.g., How much did sailor Joe earn? (salary = W*H)
  - ❖ Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - Fortunately, not in the SNLRWH example.
  - ❖ Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, not in the SNLRWH example.
- ***Tradeoff***: Must consider these issues vs. redundancy.

Lossless Join Decompositions

- Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance r that satisfies F:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)*

More on Lossless Join

- The decomposition of R into R_1 and R_2 is lossless-join wrt F iff the closure of F contains:
  - $R_1 \cap R_2 \rightarrow R_1$, or
  - $R_1 \cap R_2 \rightarrow R_2$
- The attributes common to R_1 and R_2 must contain a key for either R_1 or R_2.

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Dependency Preserving Decomposition

- Consider the contract relation schema CSJDPQV, C is key, JP $\rightarrow$ C (project purchases a part using one contract) and SD $\rightarrow$ P (department purchases only one part from a suplier).
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP $\rightarrow$ C requires a join!
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem (3).)*
- *Projection of set of FDs F*: If R is decomposed into X, ... projection of F onto X (denoted F_X) is the set of FDs U $\rightarrow$ V in F^+ (*closure of F*) such that U, V are in X.

Dependency Preserving Decompositions

- Decomposition of R into X and Y is *dependency preserving* if (F_X union F_Y) $^+$ = F^+
  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- Important to consider F^+, not F, in this definition:
  - ABC, A$\longrightarrow$ B, B$\longrightarrow$ C, C$\longrightarrow$ A, decomposed into AB and BC.
  - Is this dependency preserving? Is C$\longrightarrow$ A preserved?????
- Dependency preserving does not imply lossless join:
  - ABC, A$\longrightarrow$ B, decomposed into AB and BC.
- And vice-versa! (Example?)

Decomposition into BCNF

- Consider relation R with FDs F. If X$\rightarrow$ Y violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C, JP $\rightarrow$C, SD$\longrightarrow$ P, J$\longrightarrow$ S
  - To deal with SD$\longrightarrow$ P, decompose into SDP, CSJDQV.
  - To deal with J$\longrightarrow$ S, decompose CSJDQV into JS and CJDQV
- In general, several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g., CSZ, CS $\rightarrow$ Z, Z $\rightarrow$C
  - Can't decompose while preserving 1st FD; not in BCNF.
- Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP $\rightarrow$C, SD $\rightarrow$ P and J $\rightarrow$S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (*Redundancy!*)

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  - If $X \rightarrow Y$ is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' $JP \rightarrow C$. What if we also have $J \rightarrow C$?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

Minimal Cover for a Set of FDs

- *Minimal cover* G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.
- e.g., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ has the following minimal cover:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ and $EF \rightarrow H$
- M.C. $\rightarrow$ Lossless-Join, Dep. Pres. Decomp!!! (in book)

Multi-value Dependency

StuID	Major	Activity
100	Math	Skiing
100	Physics	Skiing
100	Math	Golf
100	Physics	Golf
200	Physics	Swimming
200	Biology	Swimming

- Key= StuID+Major+Activity
- 1NF (obvious) 2NF (all atrib key) 3NF (no transitive dependency) BCNF (no nonkey determinant) ➔
  StuID $\not\rightarrow$ Major, StuID $\not\rightarrow$ Activity

- We talk about multi-value dependencies
- StuID $\rightarrow\rightarrow$ Major and StuID $\rightarrow\rightarrow$ Activity
- Major and Activity are independent
- Anomalies: add student 100 signs up for squash, remove student 100 and swimming.

4NF and 5NF

StuID	Major
100	Math
100	Physics
200	Physics
200	Biology

StuID	Activity
100	Skiing
100	Golf
200	Swimming

- Now suppose only students Majoring in PhysEd can sign up for Decathlon
- Create another relation for the restrictions

Major	Activity
PhysEd	Decathlon

Inference Rules

- Reflexivity for FDs
  If $Y \subseteq X$ then $X \rightarrow Y$.
- Augmentation rule for FDs
  If $X \rightarrow Y$ then $XW \rightarrow Y$.
- Transitivity rule for FDs
  If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$.

Rules for both FDs and MVDs

- If $X \rightarrow Y$ then $X \rightarrow\rightarrow Y$.
- If $X \rightarrow\rightarrow Y$ and there exits $W \subseteq R$ such that $W \cap Y = \emptyset$ and $W \rightarrow Z$, then $X \rightarrow Z$.

Replication
Coalescence

- Complementation rule for MVDs
  If $X \rightarrow\rightarrow Y$ then $X \rightarrow\rightarrow (R - XY)$
- Augmentation for MVDs
  If $X \rightarrow\rightarrow Y$ and $V \subseteq W$, $W \subseteq R$ then $WX \rightarrow\rightarrow VY$.
- Transitivity rule for MVDs
  If $X \rightarrow\rightarrow Y$ and $Y \rightarrow\rightarrow Z$ then $X \rightarrow\rightarrow (Z - Y)$.

Reduced Covering

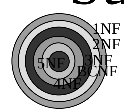
- Let M be a set of MVDs on R. Then $X \rightarrow\rightarrow W$ in M^+ is said to be
  - **trivial** if $XW = R$ or $W \subseteq X$,
  - **left-reducible** if there exists an $X' \subset X$ such that $X' \rightarrow\rightarrow W$ in M^+,
  - **right-reducible** if there exists a $W' \subset W$ such that $X \rightarrow\rightarrow W'$ in M^+,
  - **transferable** if there exists an $X' \subset X$ such that $X' \rightarrow\rightarrow (X - X')W$ in M^+.
- $X \rightarrow\rightarrow W$ is **reduced** if it is non-trivial, left-reduced, right-reduced, and non-transferable.
- M^* is then defined as the set of all reduced MVDs in M^+.

A relation scheme R is in Fourth Normal Form (4NF) with respect to a set M of FDs and MVDs if for every non-trivial MVD $X \rightarrow\rightarrow W$ in M^+ that holds in R, X is a key of R.

Examples

- Faculty = { Prof, Course, GraduateStudent }
  Prof $\rightarrow\rightarrow$ Course | GraduateStudent
  Thus, { (Prof, Course); (Prof, GraduateStudent)} is a 4NF decomposition of Faculty.
- Bank = { Customer, Account, Balance, Loan, Amount}
  Customer $\rightarrow\rightarrow$ Account, Balance | Loan, Amount
  Thus, {(Customer, Loan, Amount); (Customer, Account, Balance)} is a 4NF decomposition of Bank.
- Employee (Name, Project, Dependent)
  Name $\rightarrow\rightarrow$ Project | Dependent
  Thus, {(Name, Project); (Name, Dependent)} is a 4NF decomposition of Employee.

Summary of Schema Refinement



- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.