Schema Refinement

- The normal form satisfied by a relation is a measure of the redundancy in the relation

- A relation with redundancy can be refined by *decomposing* it with smaller relations

- Although decomposition can eliminate redundancy, it can lead to problems of its own and should be used with caution

Problems Caused by Redundancy

Update anomalies: If one copy of data is updated, an inconsistency is created unless all copies are similarly updated

Insertion anomalies: It may not be possible to store some information unless some other information is stored as well

Deletion anomalies: It may not be possible to delete some information without losing some other information as well

- Consider a relation Hourly_Emps

Hourly_Emps(*ssn*, *name*, *lot*, *rating*, *hourly_wages*, *hours_worked)*

- For example, we will refer to the Hourly_Emps schema as *SNLRWH (W denotes the hourly wages attribute)*

- The *key* for Hourly_Emps is *ssn*

- Suppose *hourly_wages* attribute is determined by the *rating* attribute. That is, for a given *rating* value, there is only one permissible *hourly_wages* value

- This IC (Integrity Constraint) is an example of a *functional dependency*

- It leads to possible redundancy in the relation Hourly_Emps, as illustrated in Figure 15.1

- If the same value appears in the *rating* column of two tuples, the IC tells us that the same value must appear in the *hourly_wages* column as well

ssn	*name*	*lot*	*rating*	*hourly_wages*	*hours_worked*
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

Figure 15.1 An Instance of the Hourly_Emps Relation

- This redundancy <u>leads to potential inconsistency</u>

- For example, the *hourly_wages* in the first tuple could be updated without making a similar change in the second tuple, which is an example of an *update anomaly*

- Also, we cannot insert a tuple for an employee unless we know the hourly wage for the employee's rating value, which is an example of an *insertion anomaly*

- If we delete all tuples with a given rating value (e.g., we delete the tuples for Smethurst and Guldu) we lose the association between that *rating* value and its *hourly_wage* value *(deletion anomaly)*

Example

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

- An **update anomaly**. Employee 519 is shown as having different addresses on different records

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424	Dr. Newsome	29-Mar-2007	?

- An **insertion anomaly**. Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

- A **deletion anomaly**. All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any courses

Null Values

- Let us consider whether the use of *null* values can address some of these problems

- Clearly, *null* values cannot help eliminate update anomalies

- For example, we cannot record the hourly_wage for a rating unless there is an employee with that rating, because we cannot store a null value in the *ssn* field, which is a primary key field

- Similarly, to deal with the deletion anomaly example, we might consider storing a tuple with *null* values in all fields except *rating* and *hourly_wages* if the last tuple with a given rating would otherwise be deleted

- However, this solution will not work because it requires the *ssn* value to be null, and primary key fields cannot be null

- Thus, null values do not provide a general solution to the problems of redundancy

Decompositions

- Redundancy arises when a relational schema forces an association between attributes that is not natural

- Functional dependencies can be used to identify such situations and to suggest refinements to the schema

- Problems arising from redundancy can be solved by replacing a relation with a collection of 'smaller' relations

- Each of the smaller relations contains a (strict) subset of the attributes of the original relation

- ***decomposition*** is dividing the larger relation into the smaller relations

- We can deal with the redundancy in Hourly_Emps by decomposing it into two relations:

  Hourly_Emps2(*ssn*, *name*, *lot*, *rating*, *hours_worked)*
  Wages(*rating*, *hourly_wages)*

- The instances of these relations corresponding to the instance of Hourly_Emps relation in Figure 15.1 is shown in Figure 15.2

ssn	*name*	*lot*	*rating*	*hours_worked*
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

rating	*hourly_wages*
8	10
5	7

Figure 15.2 Instances of Hourly_Emps2 and Wages

- Note that we can easily record the hourly wage for any rating simply by adding a tuple to Wages, even if no employee with that rating appears in the current instance of Hourly_Emps

- Changing the wage associated with a rating involves updating a single Wages tuple. This is more efficient than updating several tuples (as in the original design), and it also eliminates the potential for inconsistency

- Notice that the insertion and deletion anomalies have also been eliminated

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

Functional Dependencies

- A **functional dependency** (FD) is a constraint between two sets of attributes in a relation from a database

- Let <u>R be a relation schema</u> and let X and Y be nonempty sets of attributes in R

- We say that an <u>instance r</u> of R satisfies the FD $X \rightarrow Y$ if the following holds for every pair of tuples t_1 and t_2 in r:

  If $t1.X = t2.X$, then $t1.Y = t2.Y$

- An FD $X \rightarrow Y$ essentially says that if two tuples agree on the values in attributes X, they must also agree on the values in attributes Y

Note: $X \rightarrow Y$ is read as *X functionally determines Y*, or simply as *X determines Y*

▪Figure 15.3 illustrates the meaning of the FD $AB \rightarrow C$ by showing an instance that satisfies this dependency

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

Figure 15.3 An Instance that Satisfies $AB \rightarrow C$

▪Recall that a *legal* instance of a relation must satisfy all specified ICs, including all specified FDs

- A primary key constraint is a special case of an FD. The attributes in the key play the role of *X*, and the set of all attributes in the relation plays the role of *Y*

Reasoning About Functional Dependencies

- As an example, consider:

Workers(<u>*ssn*</u>, *name*, *lot*, *did*, *since)*

- We know that $ssn \rightarrow did$ holds, since *ssn* is the key, and FD $did \rightarrow lot$ is given to hold

- Thus, the FD $ssn \rightarrow lot$ also holds on Workers

Closure of a Set of FDs

- The set of all FDs implied by a given set (*F*) of FDs is called the **closure of F** and is denoted as F^+

- Armstrong's Axioms, can be applied repeatedly to infer all FDs implied by a set (*F*) of FDs

Properties of Functional Dependencies

- We use *X*, *Y*, and *Z* to denote sets of attributes over a relation schema *R*:

- **Reflexivity:** If $X \supseteq Y$, then $X \to Y$.
- **Augmentation:** If $X \to Y$, then $XZ \to YZ$ for any Z.
- **Transitivity:** If $X \to Y$ and $Y \to Z$, then $X \to Z$.
- **Union:** If $X \to Y$ and $X \to Z$, then $X \to YZ$.
- **Decomposition:** If $X \to YZ$, then $X \to Y$ and $X \to Z$.

Example: Suppose we are given a Relation schema R with attributes A, B, C, D, E, F, and the FDs $A \rightarrow BC$, $B \rightarrow E$, $CD \rightarrow EF$. Show that the FD $AD \rightarrow F$ holds for R and is thus a member of the closure of the given set:

Ans:

1. $A \rightarrow BC$ (given)
2. $A \rightarrow C$ (1, decomposition)
3. $AD \rightarrow CD$ (2, augmentation)
4. $CD \rightarrow EF$ (given)
5. $AD \rightarrow EF$ (3 and 4, transitivity)
6. $AD \rightarrow F$ (5, decomposition)

Attribute Closure

- If we want to check whether a given dependency, say $X \rightarrow Y$ is in the closure of a set (*F*) of FDs, we can do so efficiently without computing F^+

- We first compute the **attribute closure** X^+ with respect to *F*, which is the set of attributes *A* such that $X \rightarrow A$ can be inferred using the Armstrong Axioms

- The algorithm for computing the attribute closure of a set *X* of attributes is shown in Figure 15.6

$closure = X;$

repeat until there is no change: {

if there is an FD $U \rightarrow V$ in F such that $U \subseteq closure$,

then set $closure = closure \cup V$

}

Figure 15.6 Computing the Attribute Closure of Attribute Set X

- This algorithm starts with attribute *X* and stops as soon as there is no change in the *closure*
- By varying the starting attribute and the order in which the algorithm considers FDs, we can obtain all candidate keys

Note: Using Attribute Closure algorithm we can find out all candidate keys of a relation

Example: Suppose we are given a relation schema R with attributes A, B, C, D, E, F and FDs: $A \rightarrow BC$, $E \rightarrow CF$, $B \rightarrow E$, $CD \rightarrow EF$. Compute the closure $\{A,B\}^+$ of the set of attributes $\{A,B\}$ under this set of FDs

Ans:

1. Initialize the *closure* to $\{A,B\}$
2. Go through inner loop 4 times, once for each of the given FDs

   i) FD $A \rightarrow BC$, here A is a subset of *closure*, so add B and C to *closure* (*closure* $=\{A,B,C\}$)

   ii) FD $E \rightarrow CF$, do not add C and F to *closure*, because E is not a subset of *closure*

iii) FD B → E, add E to *closure* (*closure* ={ A,B,C,E})
iv) FD CD → EF, do not add F to *closure*
now *closure*={ A,B,C,E}

3. Go through inner loop 4 times, once for each of the given FDs

i) FD A → BC, no change in *closure*
ii) FD E → CF, *closure* ={ A,B,C,E,F}
iii) FD B → E, no change in *closure*
iv) FD CD → EF, no change in *closure*

4. Go through inner loop 4 times, once for each of the given FDs. Closure does not change so the process terminates.
so closure of $\{A,B\}^+ = \{A,B,C,E,F\}$

Normalization

- The normalization process, as first proposed by Codd, in which a series of tests are conducted on a relation to "certify" whether it satisfies a certain ***normal form***

- Initially, Codd proposed three normal forms: first, second, and third normal form

- Boyce-Codd normal form (BCNF)-was proposed later by Boyce and Codd

- All these normal forms are based on the ***functional dependencies*** among the attributes of a relation

- Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of **multivalued dependencies** and **join dependencies**, respectively

- ***Normalization of data*** is a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies

- Relation which does not satisfy the normal form test is decomposed into smaller relations

▪Another point is that the database designers *need not* normalize to the highest possible normal form

▪Every relation in BCNF is also in 3NF, every relation in 3NF is also in 2NF, and every relation in 2NF is in 1NF

Single valued attribute: Attribute which has a single value
Example: age

Multi valued attribute: Attribute which has a set of values
Example: color

Composite attribute: These attributes can be divided into smaller subparts, which represent more basic attributes

Example: Address

StreetAddress City State Zip

Number Street ApartmentNumber

First Normal Form (1NF)

- It disallow multivalued attributes, composite attributes, and their combinations
- It states that
  - *Domain of an attribute must include only atomic (simple, indivisible) values* (i.e. *the value of any attribute in a tuple must be a single value)*

- Example1: Consider the DEPARTMENT relation shown in Figure 5.1

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	{Boston,Sugarland,Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

Fig 5.1

- This is not in 1NF because DLOCATION is not an atomic attribute. So convert this attribute into single valued attribute as shown in Fig 5.2

DNAME	**DNUMBER**	DMGRSSN	**DLOCATION**
Research	5	333445555	Boston
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Fig 5.2

- Redundancy exists in this relation (Fig 5.2), so decompose this relation into two relations: DEPARTMENT and DEPT_LOCATIONS as shown in Fig 5.3

DNAME	**DNUMBER**	DMGRSSN
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

Fig 5.3a

DNUMBER	**DLOCATION**
1	Houston
4	Stafford
5	Boston
5	Sugarland
5	Houston

Fig 5.3b

- Example2: Consider EMP_PROJ relation shown in Figure 5.4a (relation which contains another relation)
- Convert it in to relation shown in Figure 5.4b
- This is not in 1NF because PNUMBER and HOURS are not an atomic attributes
- Convert relation in Fig 5.4b to the relation in Fig 5.5

SSN	ENAME	PROJS	
		PNUMBER	HOURS

Fig 5.4a

<u>SSN</u>	ENAME	PNUMBER	HOURS
123456789	Smith	1 2	32.5 7.5
666884444	Narayan	3	40.0
453453453	Joyce	1 2	20.0 20.0
999887777	Zelaya	30 10	30.0 10.0

Fig 5.4b

SSN	ENAME	PNUMBER	HOURS
123456789	Smith	1	32.5
123456789	Smith	2	7.5
666884444	Narayan	3	40.0
453453453	Joyce	1	20.0
453453453	Joyce	2	20.0
999887777	Zelaya	30	30.0
999887777	Zelaya	10	10.0

Fig 5.5

- Redundancy exists in this relation (Fig 5.5), so decompose this relation into two relations: EMP_PROJ1 and EMP_PROJ2 as shown in Fig 5.6

SSN	ENAME
123456789	Smith
666884444	Narayan
453453453	Joyce
999887777	Zelaya

SSN	PNUMBER	HOURS
123456789	1	20.0
453453453	1	32.5
123456789	2	7.5
453453453	2	20.0
666884444	3	40.0
999887777	10	10.0
999887777	30	30.0

Fig 5.6

Second Normal Form (2NF)

- It is based on the concept of *full functional dependency*

- A functional dependency $X \rightarrow Y$ is a ***full functional dependency*** if removal of any attribute *A* from *X* means that the dependency does not hold any more;
  That is, for any attribute $A \in X$, $(X - \{A\})$ does not functionally determine Y

- 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 $A \in X$, $(X - \{A\}) \rightarrow Y$

▪In Figure 5.7, {SSN, PNUMBER} → HOURS is a full dependency (neither SSN → HOURS nor PNUMBER → HOURS holds)

▪However, the dependency {SSN, PNUMBER} → ENAME is partial because SSN → ENAME holds

SSN	**PNUMBER**	**HOURS**	**ENAME**	**PNAME**	**PLOCATION**
123456789	1	32.5	Smith	ProductX	Boston
123456789	2	7.5	Smith	ProductY	Sugarland
666884444	3	40.0	Narayan	ProductZ	Houston
453453453	1	20.0	Joyce	ProductX	Boston
453453453	2	20.0	Joyce	ProductY	Sugarland
333445555	2	10.0	Franklin	ProductY	Sugarland

Fig 5.7

Prime attribute: An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R

Nonprime attribute: An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key

- The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key

- If the primary key contains a single attribute, the test need not be applied at all

- A relation schema R is in 2NF if every *nonprime attribute A* in R is *fully functionally dependent* on the primary key of R

- The EMP_PROJ relation in Fig 5.7 is in 1NF but it is not in 2NF. The nonprime attribute ENAME violates 2NF because of FD2, as do the nonprime attributes PNAME and PLOCATION because of FD3

FD1 {SSN,PNUMBER} → HOURS
FD2 SSN → ENAME
FD3 PNUMBER → {PNAME, PLOCATION}

- If a relation schema is not in 2NF, it can be "second normalized" or "2NF normalized" into a number of 2NF relations in which *nonprime attributes* are associated only with the *part of the primary key* on which they are fully functionally dependent

- The functional dependencies FD1, FD2, and FD3 in Fig 5.7 hence lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure 5.8, each of which is in 2NF

<u>SSN</u>	**<u>PNUMBER</u>**	**HOURS**

<u>SSN</u>	**ENAME**

<u>PNUMBER</u>	**PNAME**	**PLOCATION**

Fig 5.8

Third Normal Form (3NF)

- It is based on the concept of *transitive dependency*

- A functional dependency X → Y in a relation schema R is a *transitive dependency* if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, and both X → Z and Z → Y hold

- The dependency SSN → DMGRSSN is transitive through DNUMBER in EMP_DEPT of Fig 5.9,because both the dependencies
  SSN → DNUMBER and DNUMBER → DMGRSSN hold *and* DNUMBER is neither a key itself nor a subset of the key of EMP_DEPT

ENMAE	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN
Smith	123456789	1965-01-09	Houston	5	Research	333445555
Wong	333445555	1955-12-08	Dallas	5	Research	333445555
Alicia	999887777	1968-07-19	Spring	4	Administration	987654321
Jennifer	987654321	1941-06-20	Boston	4	Administration	987654321
Narayan	666884444	1962-09-15	Humble	5	Research	333445555

Fig 5.9

- According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key

Example1: The relation schema EMP_DEPT in Fig 5.9 is in 2NF, since no partial dependencies on a key exist

- However, EMP_DEPT is not in 3NF because of the transitive dependency of DMGRSSN (and also DNAME) on SSN via DNUMBER

▪We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2 shown in Fig 5.10

ENMAE	<u>SSN</u>	BDATE	ADDRESS	DNUMBER

<u>DNUMBER</u>	DNAME	DMGRSSN

Fig 5.10

Example2: The relation in Fig 5.10.1 is in 2NF, since no partial dependencies on a key exist

▪However it is not in 3NF because of the transitive dependency of DNAME on EMPNO via DNUMBER

▪We can normalize this relation by decomposing it into the two 3NF relations as shown in Fig 5.10.2

EMPNO	ENAME	DNUMBER	DNAME
1	Kevin	201	R&D
2	Jones	224	IT
3	Jake	201	R&D

Fig 5.10.1

EMPNO	ENAME	DNUMBER
1	Kevin	201
2	Jones	224
3	Jake	201

DNUMBER	DNAME
201	R&D
224	IT

Fig 5.10.2

Boyce-Codd Normal Form (BCNF)

- 3NF does not deal satisfactorily with the case of a relation with overlapping candidate keys i.e. composite candidate keys with at least one attribute in common

- BCNF is based on the concept of a *determinant*

- A determinant is any attribute (simple or composite) on which some other attribute is fully functionally dependent

- A relation is in BCNF if, and only if, every determinant is a candidate key

- Consider the following relation and determinants

  R(<u>a, b</u>, c, d)

  a, c $\rightarrow$ b, d

  a, d $\rightarrow$ b

- First determinant is a candidate key. {a, c} determine all non key attributes {b, d} of R

- Second determinant is not a candidate key. {a, d} does not determine all non key attributes of R (it does not determine c)

- This relation is not in BCNF

- <u>Example:</u>

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

Fig 5.11

- In this relation FDs are

$$\{STUDENT, COURSE\} \rightarrow INSTRUCTOR$$
$$\{STUDENT, INSTRUCTOR\} \rightarrow COURSE$$
$$\{INSTRUCTOR, COURSE\} \not\rightarrow STUDENT$$

- This relation is not in BCNF, because {INSTRUCTOR,COURSE} is not a candidate key in third FD
- Decompose the relation into two relations as shown below

STUDENT	**COURSE**
Narayan	Database
Smith	Database
Smith	Operating Systems
Smith	Theory
Wallace	Database
Wallace	Operating Systems
Wong	Database
Zelaya	Database

COURSE	**INSTRUCTOR**
Database	Mark
Database	Navathe
Operating Systems	Ammar
Theory	Schulman
Operating Systems	Ahamad
Database	Omiecinski

▪ Example:

Manager	Project	Branch
Brown	Mars	Chicago
Green	Jupiter	Birmingham
Green	Mars	Birmingham
Hoskins	Saturn	Birmingham
Hoskins	Venus	Birmingham

▪ {Manager, Project} $\longrightarrow$ {Branch}
{Project, Branch} $\longrightarrow$ {Manager}
{Manager, Branch} $\not\longrightarrow$ {Project}

▪ Third FD is not a candidate key. So relation is not in BCNF

▪ Decompose relation in to two relations with each two attributes with one common attribute

Lossless Join Decomposition

- A decomposition is *lossless* if we can recover original relation from the decomposed relations

$R(\underline{A},B,C)$

Decompose

$R1(\underline{A},B)$ $R2(\underline{A},\underline{C})$

Recover

$R^1(A,B,C)$ should be the same as $R(A,B,C)$

R^1 is in general larger than R.
Must ensure $R^1 = R$

- Sometimes the same set of data is reproduced:

Example1

Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

Name	Price
Word	100
Oracle	1000
Access	100

Name	Category
Word	WP
Oracle	DB
Access	DB

- (Word, 100) + (Word, WP) → (Word, 100, WP)
- (Oracle, 1000) + (Oracle, DB) → (Oracle, 1000, DB)
- (Access, 100) + (Access, DB) → (Access, 100, DB)

- Sometimes it's not:

Example2

Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

Category	Name
WP	Word
DB	Oracle
DB	Access

Category	Price
WP	100
DB	1000
DB	100

- (Word, WP) + (100, WP) → (Word, 100, WP)
- (Oracle, DB) + (1000, DB) → (Oracle, 1000, DB)
- (Oracle, DB) + (100, DB) → **(Oracle, 100, DB)**
- (Access, DB) + (1000, DB) → **(Access, 1000, DB)**
- (Access, DB) + (100, DB) → (Access, 100, DB)

Ensuring lossless decomposition

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$ $\quad$ $R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ or $A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p$
Then the decomposition is lossless

- In Example1 name → price, so first decomposition was *lossless*

- In Example2 category $\not\rightarrow$ name and category $\not\rightarrow$ price, and so second decomposition was *lossy*

Dependency-Preserving Decomposition

- If R is decomposed into X, Y and Z, and we enforce the FDs that hold individually on X, on Y and on Z, then all FDs that were given to hold on R must also hold

- To define dependency-preserving decompositions precisely, we have to introduce the concept of a projection of FDs

- Let R be a relation schema that is decomposed into two schemas with attribute sets X and Y, and let F be a set of FDs over R

- The **projection of F on X** is the set of FDs in the closure F^+ (not just F) that involve only attributes in X

- We will denote the projection of F on attributes X as F_X

- Note that a dependency $U \rightarrow V$ in F^+ is in F_X only if all the attributes in U and V are in X

- The decomposition of relation schema R with FDs F into schemas with attribute sets X and Y is **dependency-preserving** if $(F_X \cup F_Y)^+ = F^+$

<u>Example:</u> Suppose that a relation R with attributes ABC is decomposed into relations with attributes AB and BC. The set F of FDs over R includes $A \rightarrow B$, $B \rightarrow C$, and $C \rightarrow A$. Is

this decomposition dependency-preserving? Is $C \rightarrow A$ preserved?

Ans: given set of FDs $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

$$F^+ = F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$$

So $F^+ = \{A \rightarrow B, B \rightarrow C, C \rightarrow A, A \rightarrow C, B \rightarrow A, C \rightarrow B\}$

$$F_{AB} = \{A \rightarrow B, B \rightarrow A\}$$

$$F_{BC} = \{B \rightarrow C, C \rightarrow B\}$$

$F_{AB} \cup F_{BC} = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}$

$(F_{AB} \cup F_{BC})^+ = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B, C \rightarrow A, A \rightarrow C\}$

$(F_{AB} \cup F_{BC})^+ = F^+$

So decomposition is dependency preserved

$C \rightarrow A$ is also preserved, because $(F_{AB} \cup F_{BC})^+$ contains $C \rightarrow A$

Multi Valued Dependencies (MVD)

- The multivalued dependency $X \rightarrow\rightarrow Y$ is said to hold over R if, in every legal instance r of R, each X value is associated with a set of Y values and this set is independent of the values in the other attributes

- Relation shown in Fig 5.12 has two MVDs:

  ENAME $\rightarrow\rightarrow$ PNAME and

  ENAME $\rightarrow\rightarrow$ DNAME

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	Jogn

Fig 5.12 EMP

- An MVD X →→ Y in R is called a trivial MVD if (a) Y is a subset of X, or (b) X U Y = R

- An MVD that satisfies neither (a) nor (b) is called a nontrivial MVD

- Example: AB →→ B trivial MVD
  CD → D trivial FD

Fourth Normal Form (4NF)

- A relation is in 4NF if it is in BCNF and contains no MVDs

- BCNF to 4NF involves the removal of the MVDs from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s)

Example1:The EMP relation of Fig 5.12 is not in 4NF because it contains MVDs ENAME $\rightarrow\rightarrow$ PNAME and ENAME $\rightarrow\rightarrow$ DNAME

- We decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS shown in Fig 5.13

ENAME	PNAME
Smith	X
Smith	Y

EMP_PROJECTS

ENAME	DNAME
Smith	John
Smith	Anna

EMP_DEPENDENTS

Fig 5.13

Example2

Branch_Staff_Client relation

Branch_No	*SName*	*CName*
B3	Ann Beech	Aline Stewart
B3	David Ford	Aline Stewart
B3	Ann Beech	Mike Richie
B3	David Ford	Mike Richie

Branch_Staff relation

Branch_No	*SName*
B3	Ann Beech
B3	David Ford

Branch_Client relation

Branch_No	*CName*
B3	Aline Stewart
B3	Mike Richie