MODULE – 3
RELATIONAL DATABASE DESIGN

Data Dependency/Functional Dependency

A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the determinant and the other attribute is called the determined. For each value of the determinant there is associated one and only one value of the determined.

If A is the determinant and B is the determined then we say that *A functionally determines B* and graphically represent this as A -> B. The symbols A & B can also be expressed as *B is functionally determined by* A.

Since for each value of A there is associated one and only one value of B.

The following table illustrates A ⇒ B:

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Data Dependency/Functional Dependency

A	B
1	1
2	4
3	9
4	16
2	4
3	11
7	9

In this table ***A does not functionally determine B.***

Since for A = 3 there is associated more than one value of B.

Functional dependency can also be defined as follows:
An attribute in a relational model is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent.

Data Dependency/Functional Dependency

A functional dependency denoted (FD) is denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subset of relation R specifies a constraint on the tuples that can form a relation state of r of R. The constraint is that, for any two tuple t1 and t2 in r that have t1[X] = t2[X], we must have t1[Y] = t2[Y].

It means that the values of the Y of a tuple in r depend upon or determined by, the value of X component. Alternatively, the values of X component of a tuple uniquely (or Functionally) determine the values of the Y component.

In other words, there is a functional dependency from X to Y or that Y is functionally dependent on X.

Thus, X functionally determines Y in a relation schema R if and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y value.

Data Dependency/Functional Dependency

For example, in the relation schema PROJECT of the employees, the following functional dependencies should hold-

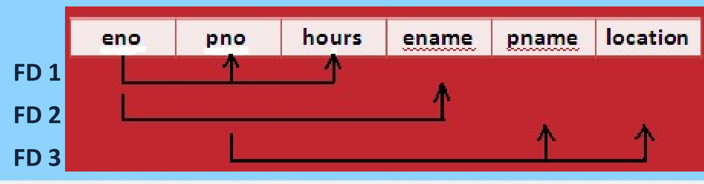
1. eno → ename
2. pno → pname, location
3. Eno, pno → hours

These functional dependencies specifies that

1. The value of an employee's number (eno) uniquely determines the employee name (ename).
2. The value of project number (pno) uniquely determines that project name (pname) and project location (location).
3. A combination of eno and pno values uniquely determines the number of hours the employee works on the project per week.

Alternatively, we can say that ename is functionally determined by eno and so on.

PROJECT SCHEMA



Data Dependency/Functional Dependency

Types of Functional Dependencies

There are many types of functional dependencies that depending on different criteria-

1. Full Functional Dependency
2. Partial Dependency
3. Transitive Dependency
4. Trivial and Non-trivial Dependency

- **Full Functional Dependency:**
  For a relation schema R and a FD, A → B, B is fully functionally dependent on A if there is no C, where C is proper subset of A, such that C → B.
  The Dependency A → B is left reduced, that is, there is no extraneous attributes in left side in the dependency.

- **Partial Dependency:**
  A functional dependency that holds in a relation is partial when removing one of the determining attributes gives a functional dependency that holds in the relation.
  E.g. if {A,B} → {C} but also {A} → {C} then {C} is partially functionally dependent on {A,B}.

Data Dependency/Functional Dependency

- **Partial Dependency:**
  Partial Dependency is a form of Functional dependency that holds on a set of attributes. It is about the complete dependency of a right hand side attribute on one of the left hand side attributes. In a functional dependency XY →Z, if Z (RHS attribute) can be uniquely identified by one of the LHS attributes, then the functional dependency is partial dependency.

  Example:
  Let us assume a relation R with attributes A, B, C, and D. Also, assume that the set of functional dependencies F that hold on R as follows;
  F = {A → B, D → C}.
  From set of attributes F, we can derive the primary key. For R, the key can be (A,D), a composite primary key. That means, AD → BC, AD can uniquely identify B and C. But, for this case A and D is not required to identify B or C uniquely. To identify B, attribute A is enough. Likewise, to identify C, attribute D is enough. The functional dependencies AD → B or AD → C are called as Partial functional dependencies.

Data Dependency/Functional Dependency

- **Transitive Dependency:** The functional dependency follows the mathematical property of *transitivity*, which states that if A=B and B=C, then A=C. Because ItemNo determines CategoryID, which in turn determines CategoryName and CategoryManager, the relation contains a transitive dependency.

  ItemNo → Title, Price, CategoryID
  CategoryId → CategoryName, CategoryManager

  Transitive dependencies occur when there is an indirect relationship that causes a functional dependency.

  Examples: For example, "A -> C" is a transitive dependency when it is true only because both "A -> B" and "B -> C" are true.

Data Dependency/Functional Dependency

- **Trivial and Non-trivial Dependency:**
  Trivial: If an FD X → Y holds where Y subset of X, then it is called a trivial FD. Trivial FDs are always hold.
  Symbolically: A → B is trivial functional dependency if B is a subset of A.
  The following dependencies are also trivial:
  A → A
  AB → A
  AB → B
  B → B

  For example:
  Consider a table with two columns Student_id and Student_Name.
  {Student_Id, Student_Name} → Student_Id
  is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined. Also,
  Student_Id→Student_Id
  Student_Name→Student_Name
  are trivial dependencies too.

Data Dependency/Functional Dependency

- **Trivial and Non-trivial Dependency:**
  Non-trivial: If an FD X → Y holds where Y is not subset of X, then it is called non-trivial FD.
  For example:
  An employee table with three attributes:
  emp_id, emp_name, emp_address.

  The following functional dependencies are non-trivial:

  emp_id→ emp_name (emp_name is not a subset of emp_id)
  emp_id → emp_address (emp_address is not a subset of emp_id)

  On the other hand, the following dependencies are trivial:

  emp_id, emp_name → emp_name
  emp_name is a subset of {emp_id, emp_name}

  Completely non-trivial: If an FD X → Y holds where x intersect Y = Φ, is said to be completely non-trivial FD.

Data Dependency/Functional Dependency

Conclusion of Functional Dependency

Functional Dependencies:

- Data dependencies are constraints imposed on data in database.
- They are part of the scheme definition.
- FDs allow us to formally define keys.
- A conjecture (It has to be proven) is that a set of functional dependencies and one join dependency are enough to express the dependency structure of a relational database scheme.

Motivation:

Functional dependencies help in accomplishing the following two goals:

(a) controlling redundancy and
(b) enhancing data reliability.

Data Dependency/Functional Dependency

Problematic Issue:

Representing the set of all FDs for a relation R.

Solution:

- Find a basic set of FDs.
- Use axioms for inferring.
- Represent the set of all FDs as the set of FDs that can be inferred from the basic set of FDs.

Axioms:

An inference axiom is a rule that states if a relation satisfies certain FDs then it must satisfy certain other FDs.

Data Dependency/Functional Dependency

Armstrong's axioms are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong.

FD manipulations:
Soundness -- no incorrect FD's are generated
Completeness -- all FD's can be generated

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z & W to represent any subset of and, for short, the union of two sets of attributes and by instead of the usual X U Y.

F1. Reflexivity X → X
F2. Augmentation If (Z ⊆ W; X → Y) then XW → YZ
F3. Additivity If { (X → Y) (X → Z)} then X → YZ
F4. Projectivity If (X → YZ) then X → Y
F5. Transitivity If (X → Y) and (Y → Z) then (X → Z)
F6. Pseudotransitivity If (X → Y) and (YZ → W) then XZ → W

Data Dependency/Functional Dependency

Axiom Name	Axiom	Example
Reflexivity	if a is set of attributes, b ⊆ a, then a →b	SSN,Name → SSN
Augmentation	if a→ b holds and c is a set of attributes, then ca→cb	SSN → Name then SSN,Phone → Name, Phone
Transitivity	if a →b holds and b→c holds, then a→ c holds	SSN →Zip and Zip→City then SSN →City
Union or Additivity *	if a → b and a → c holds then a→ bc holds	SSN→Name and SSN→Zip then SSN→Name,Zip
Decomposition or Projectivity*	if a → bc holds then a → b and a → c holds	SSN→Name,Zip then SSN→Name and SSN→Zip
Pseudotransitivity*	if a → b and cb → d hold then ac → d holds	Address → Project and Date→Amount then Address,Date → Amount
(NOTE)	ab→ c does NOT imply a → b and b → c	

Data Dependency/Functional Dependency

Prove or disprove the following inference rules for functional dependencies-

i. $\{W \rightarrow Y, X \rightarrow Z\} \Rightarrow \{WX \rightarrow Y\}$
ii. $\{X \rightarrow Y\}$ and $Y \supseteq Z \Rightarrow \{X \rightarrow Z\}$
iii. $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow \{X \rightarrow YZ\}$
iv. $\{X \rightarrow Z, Y \rightarrow Z\} \Rightarrow \{X \rightarrow Y\}$
v. $\{XY \rightarrow Z, Z \rightarrow W\} \Rightarrow \{X \rightarrow W\}$

i. $\{W \rightarrow Y, X \rightarrow Z\} \Rightarrow \{WX \rightarrow Y\}$

Given

$W \rightarrow Y$ --------- (a)

$X \rightarrow Z$ --------- (b)

by augmenting in (a) by X

$WX \rightarrow XY$ ------- (c)

by decomposing in (c)

$WX \rightarrow Y$ Hence, it is proved.

Data Dependency/Functional Dependency

Prove or disprove the following inference rules for functional dependencies-

ii. $\{X \rightarrow Y\}$ and $Y \supseteq Z \Rightarrow \{X \rightarrow Z\}$

Given

$X \rightarrow Y$ --------- (a)

$Y \supseteq Z$ and that two tuples t1 and t2 exist in some relation instance r of R such that t1[Y] = t2[Y]. Then t1[Z] = t2[Z] because $Y \supseteq Z$; hence $Y \rightarrow Z$ must hold.

by transitivity $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$. Hence, it is proved.

iii. $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow \{X \rightarrow YZ\}$

Given

$X \rightarrow Y$ --------- (a)

$Y \rightarrow Z$ --------- (b)

by augmentation rule on (b) by Y

$Y \rightarrow YZ$ --------- (c)

and by transitivity rule on (a) and (c)

$X \rightarrow Y$ and $Y \rightarrow YZ \Rightarrow \{X \rightarrow YZ\}$

Hence, it is proved.

Data Dependency/Functional Dependency

Prove or disprove the following inference rules for functional dependencies-

iv. {X → Z, Y →Z} ⇒ {X →Y}

Given X → Z i.e. X ⊒Z and that any two tuples t1 and t2 of relation R such that t1[X] = t2[X] then t1[Z] = t2[Z].

Also given Y → Z i.e. Y ⊒Z and that any two tuples t1 and t2 of relation R such that t1[Y] = t2[Y] then t1[Z] = t2[Z].
This implies that X ⊒ Y; i.e. X →Y. Hence, it is proved.

iv. {XY → Z, Z →W} ⇒ {X →W}
Given
XY →Z --------- (a)
Z →W --------- (b)
by transitivity rule on (a) and (c)
XY →Z and Z →W ⇒ {XY → W} -------- (c)
by decomposition rule on (c)
X → W, Y → W
Hence, it is proved.

Data Dependency/Functional Dependency

Closure of Functional Dependencies

Suppose F is a Set of functional dependencies for a given relation R. Then,

Closure of F:

It is a set of all functional dependencies that include F as well as all dependencies that are inferred from F. It is denoted by

F+

X → Y is inferred from F specified in R if X → Y holds in every legal relation state r of R.
By applying the following 6 inference rules, we can inferred FDs of F.

1. Reflexivity X → X
2. Augmentation If (Z ⊆ W; X → Y) then XW → YZ
3. Additivity If { (X → Y) (X → Z)} then X → YZ
4. Projectivity If (X → YZ) then X → Y
5. Transitivity If (X → Y) and (Y → Z) then (X → Z)
6. Pseudotransitivity If (X → Y) and (YZ → W) then XZ → W

Data Dependency/Functional Dependency

Example for Closure of F:

Suppose we are given a relation scheme *R*=(*A*,*B*,*C*,*G*,*H*,*I*), and the set of functional dependencies:

$$A \rightarrow B$$
$$A \rightarrow C$$
$$CG \rightarrow H$$
$$CG \rightarrow I$$
$$B \rightarrow H$$

Applying the rules to the scheme and set *F* mentioned above, we can derive the following:

1. $A \rightarrow H$, as we saw by the transitivity rule.
2. $CG \rightarrow HI$ by the union rule.
3. $AG \rightarrow I$ by several steps:
   i. Note that $A \rightarrow C$ holds.
   ii. Then $AG \rightarrow CG$, by the augmentation rule.
   iii. Now by transitivity, $AG \rightarrow I$.

(You might notice that this is actually pseudotransivity if done in one step.)

Data Dependency/Functional Dependency

Example for Closure of F:

Assume that there are 4 attributes A, B, C, D, and that F = {A → B, B → C}. Then, F + includes all the following:

FDs: A → A, A → B, A → C, B → B, B → C, C → C, D → D, AB → A, AB → B, AB → C, AC → A, AC → B, AC → C, AD → A, AD → B, AD → C, AD → D, BC → B, BC → C, BD → B, BD → C, BD → D, CD → C, CD → D, ABC → A, ABC → B, ABC → C, ABD → A, ABD → B, ABD → C, ABD → D, BCD → B, BCD → C, BCD → D, ABCD → A, ABCD → B, ABCD → C, ABCD → D.

Data Dependency/Functional Dependency

Example for Closure of F:

Assume that there are 4 attributes A, B, C, D, and that F = {A → B, B → C}.
To compute F+,
we first get:

A+ = AB+ = AC+ = ABC+ = {A, B, C}
B+ = BC+ = {B, C}
C+ = {C}
D+ = {D}
AD+ = {A, D}
BC+ = {B, C}
BD+ = BCD+ = {B, C, D}
ABD+ = ABCD+ = {A, B, C, D}
ACD+ = {A, C, D}

It is easy to generate the FDs in F + from the closures of the above attribute sets.

Data Dependency/Functional Dependency

Closure of Attribute

Define the closure of α under F (denoted by α+) as the set of attributes that are functionally determined by α under F:

$\alpha \rightarrow \beta$ is in F+ $\Leftrightarrow \beta \subseteq \alpha+$

- Algorithm to compute α+, the closure of α under F

```
result := α;
while (changes to result) do
    for each β → γ in F do
        begin
        if β ⊆ result then result := result ∪ γ;
        end
```

Example:

R = (A, B, C, G, H, I) and F = {A → B A → C CG → H CG → I B → H}

- (AG+)
  1. result = AG
  2. result = ABCG (A → C and A ⊆ AGB)
  3. result = ABCGH (CG → H and CG ⊆ AGBC)
  4. result = ABCGHI (CG → I and CG ⊆ AGBCH)

Data Dependency/Functional Dependency

Finding Candidate Key

Let F be a set of FDs, and R a relation.

A candidate key is a set X of attributes in R such that

- X + includes all the attributes in R.
- There is no proper subset Y of X such that Y + includes all the attributes in R.

Note: A proper subset Y is a subset of X such that Y != X (i.e., X has at least one element not in Y).

Example.

Consider a table R(A, B, C, D), and that F = {A → B, B → C}.

A is not a candidate key, because A + = {A, B, C} which does not include D.
ABD is not a candidate key even though ABD+ = {A, B, C, D}.
This is because AD+ = {A, B, C, D}, namely, there is a proper subset AD of ABD such that AD+ includes all the attributes. AD is a candidate key.

Data Dependency/Functional Dependency

Finding Candidate Key

Example.

Consider a table R(A, B, C, D, E, F), and that F = {A → C, C → D, D → B, E → F}.
Find all the possible candidate key.

Given A → C, A determines C
C → D, C determines D
D → B, D determines B
E → F E determines F

Now, the easiest way is to find which attributes are not determined. In this example, A and E are not determined. Then, find out the closure of (AE)+.

(AE)+ = $\underline{A}$E
= A$\underline{C}$E
= AC$\underline{D}$E
= ACD$\underline{B}$E
= ACDB$\underline{E}$F

Closure of AE has all the attributes. Thus, AE is a candidate key. In this way, we can find out more candidate keys for this problem.

Normalization

While designing a database out of an entity–relationship model, the main problem existing in that "raw" database is redundancy. Redundancy is storing the same data item in more one place. A redundancy creates several problems like the following:

- Extra storage space: storing the same data in many places takes large amount of disk space.
- Entering same data more than once during data insertion.
- Deleting data from more than one place during deletion.
- Modifying data in more than one place.
- Anomalies may occur in the database if insertion, deletion, modification etc are no done properly. It creates inconsistency and unreliability in the database.

To solve this problem, the "raw" database needs to be normalized. This is a step by step process of removing different kinds of redundancy and anomaly at each step. At each step a specific rule is followed to remove specific kind of impurity in order to give the database a slim and clean look.

Normalization

Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purpose,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e. data is logically stored.

Normalization

Un-Normalized Form (UNF)

If a table contains non-atomic values at each row, it is said to be in UNF. An **atomic value** is something that can not be further decomposed. A **non-atomic value**, as the name suggests, can be further decomposed and simplified. Consider the following table:

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI
		Feb	1200		
		Mar	850		
E02	BB	Jan	2200	B02	UTI
		Feb	2500		
E03	CC	Jan	1700	B01	SBI
		Feb	1800		
		Mar	1850		
		Apr	1725		

In the sample table above, there are multiple occurrences of rows under each key Emp-Id. Although considered to be the primary key, Emp-Id cannot give us the unique identification facility for any single row. Further, each primary key points to a variable length record (3 for E01, 2 for E02 and 4 for E03).

Normalization

Problems without Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

Update anomalies − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

Deletion anomalies − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

Insert anomalies − We tried to insert data in a record that does not exist at all.
Normalization is a method to remove all these anomalies and bring the database to a consistent state.

Normalization

Normalization Rule

Normalization rule are divided into following normal form.

- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF
- Fourth Normal Form
- Fifth Normal Form (PJNF)

First Normal Form (1NF)

A relation is in first normal form if it meets the definition of a relation:

- Each attribute (column) value must be a single value only.
- All values for a given attribute (column) must be of the same type.
- Each attribute (column) name must be unique.
- The order of attributes (columns) is insignificant
- No two tuples (rows) in a relation can be identical.
- The order of the tuples (rows) is insignificant.

If you have a *key* defined for the relation, then you can meet the *unique row* requirement.

Normalization

First Normal Form (1NF)

A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values. The above table in UNF can be processed to create the following table in 1NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI
E01	AA	Feb	1200	B01	SBI
E01	AA	Mar	850	B01	SBI
E02	BB	Jan	2200	B02	UTI
E02	BB	Feb	2500	B02	UTI
E03	CC	Jan	1700	B01	SBI
E03	CC	Feb	1800	B01	SBI
E03	CC	Mar	1850	B01	SBI
E03	CC	Apr	1725	B01	SBI

As you can see now, each row contains unique combination of values. Unlike in UNF, this relation contains only atomic values, i.e. the rows can not be further decomposed, so the relation is now in 1NF.

Normalization

Un-Normal Form Table

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
			09/08/2013	23.93
			09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
			09/08/2013	24.14
			09/09/2013	24.33

Table in First Normal Form

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

n can be removed into another related relation, it would come to 2NF

Normalization

Second Normal Form (2NF)

A relation is said to be in 2NF f if it is already in 1NF and each and every attribute fully depends on the primary key of the relation. Speaking inversely, if a table has some attributes which is not dependant on the primary key of that table, then it is not in 2NF.

Let us explain. Emp-Id is the primary key of the above relation. Emp-Name, Month, Sales and Bank-Name all depend upon Emp-Id. But the attribute Bank-Name depends on Bank-Id, which is not the primary key of the table. So the table is in 1NF, but not in 2NF. If this position can be removed into another related relation, it would come to 2NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id
E01	AA	JAN	1000	B01
E01	AA	FEB	1200	B01
E01	AA	MAR	850	B01
E02	BB	JAN	2200	B02
E02	BB	FEB	2500	B02
E03	CC	JAN	1700	B01
E03	CC	FEB	1800	B01
E03	CC	MAR	1850	B01
E03	CC	APR	1726	B01

Bank-Id	Bank-Name
B01	SBI
B02	UTI

After removing the portion into another relation we store lesser amount of data in two relations without any loss information. There is also a significant reduction in redundancy.

Normalization

The following example relation *is not* in 2NF:
STOCKS (Company, Symbol, Headquarters, Date, Close_Price)

To start the normalization process, list the functional dependencies (FD):

FD1: Symbol, Date → Company, Headquarters, Close Price

FD2: Symbol → Company, Headquarters

Consider that Symbol, Date → Close Price. So we might use Symbol, Date as our key.
However we also see that: Symbol → Headquarters

- This violates the rule for 2NF in that a *part of our key*. key determines a non-key attribute.
- Another name for this is a *Partial key dependency*. Symbol is only a "part" of the key and it determines a non-key attribute.
- Also, consider the insertion and deletion anomalies.

One Solution:

Split this up into two new relations:
COMPANY (Company, Symbol, Headquarters)
STOCK_PRICES (Symbol, Date, Close_Price)

Normalization

- At this point we have two new relations in our relational model. The original "STOCKS" relation we started with is removed form the model.
- Sample data and functional dependencies for the two new relations:
- COMPANY Relation:

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

FD1: Symbol, Date → Company, Headquarters, Close Price

FD2: Symbol → Company, Headquarters

Company	Symbol	Headquarters
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

Symbol	Date	Close Price
MSFT	09/07/2013	23.96
MSFT	09/08/2013	23.93
MSFT	09/09/2013	24.01
ORCL	09/07/2013	24.27
ORCL	09/08/2013	24.14
ORCL	09/09/2013	24.33

t can be removed into another related relation, it would come to 2NF.

Normalization

Third Normal Form (3NF)

A relation is in third normal form (3NF) if it is in second normal form and it contains no *transitive dependencies*.

Consider relation R containing attributes A, B and C. R(A, B, C)

If A → B and B → C then A → C

Transitive Dependency: Three attributes with the above dependencies.

Example: At CUNY:

Course_Code → Course_Number, Section

Course_Number, Section → Classroom, Professor

Consider one of the new relations we created in the STOCKS example for 2nd normal form:

The functional dependencies we can see are:

FD1: Symbol → Company

FD2: Company → Headquarters

so therefore: Symbol → Headquarters

This is a transitive dependency.

What happens if we remove Oracle?

We loose information about 2 different facts.

Company	Symbol	Headquarters
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

Normalization

The solution again is to split this relation up into two new relations:

STOCK_SYMBOLS(Company, Symbol)

COMPANY_HEADQUARTERS(Company, Headquarters)

This gives us the following sample data and FD for the new relations

FD1: Symbol → Company

Company	Symbol
Microsoft	MSFT
Oracle	ORCL

FD2: Company → Headquarters

Company	Headquarters
Microsoft	Redmond, WA
Oracle	Redwood Shores, CA

Normalization

Boyce-Codd Normal Form (BCNF)

- A relation is in BCNF if every determinant is a candidate key.
- Recall that not all determinants are keys.
- Those determinants that are keys we initially call *candidate keys*.
- Eventually, we select a single candidate key to be *the key* for the relation.
- Consider the following example:
  - Funds consist of one or more Investment Types.
  - Funds are managed by one or more Managers
  - Investment Types can have one more Managers
  - Managers only manage one type of investment.

Relation: FUNDS (FundID, InvestmentType, Manager)

FD1: FundID, InvestmentType → Manager

FD2: FundID, Manager → InvestmentType

FD3: Manager → InvestmentType

FundID	InvestmentType	Manager
99	Common Stock	Smith
99	Municipal Bonds	Jones
33	Common Stock	Green
22	Growth Stocks	Brown
11	Common Stock	Smith

Normalization

- In this case, the combination FundID and InvestmentType form a *candidate key* because we can use FundID,InvestmentType to uniquely identify a tuple in the relation.
- Similarly, the combination FundID and Manager also form a *candidate key* because we can use FundID, Manager to uniquely identify a tuple.
- Manager by itself is not a candidate key because we cannot use Manager alone to uniquely identify a tuple in the relation.
- Is this relation FUNDS(FundID, InvestmentType, Manager) in 1NF, 2NF or 3NF ? Given we pick FundID, InvestmentType as the *Primary Key:*
  - ✓ 1NF for sure.
  - ✓ 2NF because all of the non-key attributes (Manager) is dependant on all of the key.
  - ✓ 3NF because there are no transitive dependencies.

- However consider what happens if we delete the tuple with FundID 22. We loose the fact that Brown manages the InvestmentType “Growth Stocks.”

Normalization

- Therefore, while FUNDS relation is in 1NF, 2NF and 3NF, it is in BCNF because not all determinants (Manager in FD3) are candidate keys.
- The following are steps to normalize a relation into BCNF:
  - ✓ List all of the determinants.
  - ✓ See if each determinant can act as a key (candidate keys).
  - ✓ For any determinant that is *not* a candidate key, create a new relation from the functional dependency. Retain the determinant in the original relation.
- For our example:FUNDS (FundID, InvestmentType, Manager)
- The determinants are:FundID, InvestmentType FundID, Manager Manager
- Which determinants can act as keys?

  FundID, InvestmentType *YES*

  FundID, Manager *YES*

  Manager *NO*
- Create a new relation from the functional dependency:

  MANAGERS(<u>Manager</u>, InvestmentType),

  FUND_MANAGERS(<u>FundID</u>, <u>Manager</u>)

In this last step, we have retained the determinant “Manager” in the original relation MANAGERS.Each of the new relations should be checked to ensure they meet the definitions of 1NF, 2NF, 3NF and BCNF

Normalization

- For our example:FUNDS (FundID, InvestmentType, Manager)
- The determinants are:FundID, InvestmentType FundID, Manager Manager
- Which determinants can act as keys?

  FundID, InvestmentType *YES*

  FundID, Manager *YES*

  Manager *NO*
- Create a new relation from the functional dependency:

  MANAGERS(<u>Manager</u>, InvestmentType),

  FUND_MANAGERS(<u>FundID</u>, <u>Manager</u>)

In this last step, we have retained the determinant “Manager” in the original relation MANAGERS. Each of the new relations should be checked to ensure they meet the definitions of 1NF, 2NF, 3NF and BCNF

FundID	Manager
99	Smith
99	Jones
33	Green
22	Brown
11	Smith

InvestmentType	Manager
Common Stock	Smith
Municipal Bonds	Jones
Common Stock	Green
Growth Stocks	Brown
Common Stock	Smith

Normalization

Fourth Normal Form (4NF)

A relation is in fourth normal form if it is in BCNF and it contains no *multivalued dependencies*.

Multivalued Dependency: A type of functional dependency where the determinant can determine more than one value.

More formally, there are 3 criteria:

- There must be at least 3 attributes in the relation. call them A, B, and C, for example.
- Given A, one can determine multiple values of B.
- Given A, one can determine multiple values of C.
- B and C are independent of one another.

Book example:

Student has one or more majors.

Student participates in one or more activities.

FD1: StudentID →→ Major
FD2: StudentID →→ Activities

Normalization

A few characteristics:

- No regular functional dependencies
- All three attributes taken together form the key.
- Later two attributes are independent of one another.
- Insertion anomaly: Cannot add a stock fund without adding a bond fund (NULL Value). Must always maintain the combinations to preserve the meaning.

Stock Fund and Bond Fund form a multivalued dependency on Portfolio ID.

PortfolioID →→ Stock Fund
PortfolioID →→ Bond Fund

Portfolio ID	Stock Fund	Bond Fund
999	Janus Fund	Municipal Bonds
999	Janus Fund	Dreyfus Short-Intermediate Municipal Bond Fund
999	Scudder Global Fund	Municipal Bonds
999	Scudder Global Fund	Dreyfus Short-Intermediate Municipal Bond Fund
888	Kaufmann Fund	T. Rowe Price Emerging Markets Bond Fund

Normalization

Resolution: Split into two tables with the common key:

Portfolio ID	Stock Fund
999	Janus Fund
999	Scudder Global Fund
888	Kaufmann Fund

Portfolio ID	Bond Fund
999	Municipal Bonds
999	Dreyfus Short-Intermediate Municipal Bond Fund
888	T. Rowe Price Emerging Markets Bond Fund

Normalization

Fifth Normal Form (5NF)

- Also called “Projection Join” Normal form.
- There are certain conditions under which after decomposing a relation, it cannot be reassembled back into its original form.

Canonical Cover (Fc) of FDs

Canonical Cover (Fc)

Canonical cover (Fc) is a minimal set of FDs equivalent to F.

1. No FDs in Fc contains extraneous attributes.
2. Each LHS (Left Hand Side) of FD in Fc is unique.

Algorithm

i/p → F
o/p → Fc

repeat

1. Use the decompose rule to replace
   a1 → b1 & a1→ b2
   by a1 → b1b2
2. Find FD a → b which has an extraneous attribute 'A' and remove A from a → b
3. Remove redundant FD

Until F does not change

Canonical Cover (Fc) of FDs

Steps to find out Canonical Cover (Fc)

1. Singleton RHS (Right Hand Side)
2. Removal of extraneous attribute
3. Removal of Redundant FD

Consider two sets of FDs

F = { A→B, AB →C, D→AC, D→E}

G= { A→BC, D→AB}

Which one is true?

a) F covers G
b) G covers F
c) F & G are equal
d) None

Let us consider first set of FDs F = { A→B, AB →C, D→AC, D→E}

1. Here given D→AC apply decompose rule D→A, D→C
2. After this FDs are F = { A→B, AB →C, D→A, D→C, D→E}
   Only one FD is a partial FD AB →C where B is extraneous. Because, A+ = AB & B+ = B.

Canonical Cover (Fc) of FDs

3. Now find out redundant FDs
   F = { A→B, A →C, D→A, D→C, D→E}
   redundant FD can be find out by finding closure of each attribute set of FD, without any help of that FD.
   for example:
   A→B For this FD calculate A+ and try to derive B from A without help of A→B is it possible? No, then it is non-redundant.
   A+ = AC
   AB→C For this FD calculate AB+ and try to derive C from AB without help of AB→C is it possible? No, then it is non-redundant.
   A+ = AB
   D→A For this FD calculate D+ and try to derive A from D without help of D→A is it possible? No, then it is non-redundant.
   D+ = DCE
   D→C For this FD calculate D+ and try to derive C from D without help of D→C is it possible? **Yes, then it is redundant.**
   D+ = DAEBC
   D→E For this FD calculate D+ and try to derive E from D without help of D→E is it possible? No, then it is non-redundant.
   D+ = DACB

Canonical Cover (Fc) of FDs

Hence Fc for given FD set F = { A→B, A →C, D→A, D→C, D→E} is
Fc = { A→B, A →C, D→A, D→E}
Similarly, we'll consider G = { A→BC, D→AB}

1. Here given A→BC, D→AB apply decompose rule A→B, A →C, D→A, D→B
2. After this FDs are G = {A→B, A →C, D→A, D→B}
   Here, No FD has extraneous attribute. Follow third step
3. Now find out redundant FDs
   G = {A→B, A →C, D→A, D→B}
   A→B For this FD calculate A+ and try to derive B from A without help of A→B is it possible? No, then it is non-redundant.
   A+ = AC
   AB→C For this FD calculate AB+ and try to derive C from AB without help of AB→C is it possible? No, then it is non-redundant.
   A+ = AB
   D→A For this FD calculate D+ and try to derive A from D without help of D→A is it possible? No, then it is non-redundant.
   D+ = DB
   D→ B For this FD calculate D+ and try to derive C from D without help of D→B is it possible? **Yes, then it is redundant.**
   D+ = DABC

Canonical Cover (Fc) of FDs

Hence Gc for given FD set G = { A→B, A →C, D→A, D→B} is

Gc = { A→B, A →C, D→A}

Now, Compare Fc and Gc

Fc = { A→B, A →C, D→A, D→E}

Gc = { A→B, A →C, D→A}

a) F covers G
b) G covers F
c) F & G are equal
d) None

Answer is d) None