COMP1140: Database and Information Management

Lecture Note – Week 5

Dr Suhuai Luo

School of EEC

University of Newcastle

Notice

- Assignment 1 is due now (10 am Tue, 22/8)
- Assignment 2 starts now.

Last lecture

- Relational Model
- Mapping from EER model to relational model (i.e. first step of Logical Database Design)
- Note: 7 logical steps:
  - *Get relations from EER*
  - Relation normalisation
  - Validate relations against user transaction
  - *Check integrity constrains*
  - Review logical data model with user
  - Merge logical models into global model
  - Check for future growth

This week

- Normalization
  - Introduction
  - Redundancy
  - Anomalies
  - Lossless join decomposition
- Functional dependencies
- Normal Forms – definitions & how to get them
- About assignment 2
- Ref: chapter 14 & 15

Normalization - Introduction

- EER Modelling is a subjective process.
- This may result in many different relation schema (after mapping)
- How can we validate whether the relations created are good?
  - There are *minimal* number of attributes necessary to support the data requirements of the enterprise;
  - attributes with a close logical relationship are found in the same relation;
  - *minimal* redundancy with each attribute represented only once, with the important exception of attributes that form all or part of foreign keys. - Check for the existence of redundancies!

Normalization – Introduction (contd.)

- Redundancy?
  - Unnecessary repetition of data in relations

- Example

Lecturer

<u>lecID</u>	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Normalization – Introduction (contd.)

- Redundancy causes anomalies and consistency issues

- Types of anomalies:
  - Insertion Anomaly
  - Deletion Anomaly
  - Modification Anomaly

Normalization – Introduction (contd.)

- Insertion Anomaly
  - inconsistency
  - circular dependency
- Example

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Normalization – Introduction (contd.)

- Deletion Anomaly
  - Example:
  - Deleting lecturer -> lose dept info

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Normalization – Introduction (contd.)

- Modification Anomaly
  - Example:
  - phone changes ->dept info update

lecID	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Normalization – Introduction (contd.)

- How can redundancies be avoid ?
  - Decompose relations to smaller relations without redundancies
    - Example:
      - LecturerInfo(<u>lecID</u>, name, salary, deptCode)
      - Dept(<u>deptCode</u>, dname, deptPhone, building)
    - No insertion, deletion or modification anomalies!!!!

- Normalization:
  - A formal process to minimize redundancies in relations

Considerations when decomposing relations

- Loss-less join property:
  - No loss of data from original relation
  - Can obtain original relation by joining decomposed relation
- Dependency-preserving property:
  - No loss of dependencies (i.e. functional dependencies*) during decomposition

<u>lecID</u>	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Example of a lossy join

- Decomposing S to S_1 and S_2

S

S	P	D
S1	P1	D1
S2	P2	D2
S3	P1	D3

→

S_1

S	P
S1	P1
S2	P2
S3	P1

S_2

P	D
P1	D1
P2	D2
P1	D3

Joining S_1 and S_2 on P will result in spurious tuples: data is lost!
Not preserving the **lossless-join** property!!!

Functional Dependency

- Consider the lecturer relation again:

Lecturer

<u>lecID</u>	name	salary	deptCode	dname	deptPhone	building
L023	Paul Lee	12	PHYS	Dept. of Physics	X14090	Physics
L012	Mary Smith	12	DCIT	School of Design, Comm and IT	X54500	ICT
L021	Peter Wang	10	DCIT	School of Design, Comm and IT	X54500	ICT

Whenever the same deptCode appear, same dept information appear – redundancy!!!

Functional Dependency (contd.)

- Formally, we can define a functional dependency as follows:

- A functional dependency, denoted by $X \rightarrow Y$, where X and Y are sets of attributes in relation R, specifies the following constraint:

  Let t_1 and t_2 be tuples of relation R for **any** given instance

  whenever $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$

  where $t_i[X]$ represents the values for X in tuple t_i

Functional Dependency (contd.)

- Graphically, if X→Y in relation R, then

R

	...	X	...	Y	...
	...				
t_1 →		a		b	
	...				
t_2 →		a		b	
	...				

Whenever X values are equal Y values are equal

Functional Dependency (contd.)

- Points to note:
  - Functional dependency X→Y does not necessarily mean Y→X
    - E.g. Lecturer relation

deptCode → building

However,

building $\not\rightarrow$ deptCode

Many departments may share the same building!

Functional Dependency (contd.)

- Functional dependency must hold for **any** instance (i.e. all instances)
- You cannot determine a functional dependency by considering one instance alone
- Semantics (meaning) of attributes in the domain (i.e. enterprise) needs to be understood
- Practically, considering instances provides only hints to the existence of functional dependencies

Functional Dependency (contd.)

- Terminology:
  - Functional dependency X → Y in relation **R**, we say,
    - X functionally determines Y
    - Y is functionally dependent on X
  - X is also called the **determinant** (i.e. left-hand side of the functional dependency)

Functional Dependency (contd.)

- Example
  - Lecturer(lecID, name, salary, deptCode, dname, deptPhone, building)
  - Functional dependencies
    - lecID → name, salary, deptCode
    - deptCode → dname, deptPhone, building

Functional Dependency (contd.)

- **Full functionally dependency**: A functional dependency X → Y is fully functional dependent if Y is functionally dependent on X, but not on any proper subset of X.

- Example:
  - lecID → deptCode is a full functional dependency
  - lecID, name → deptCode is a valid functional dependency but not a full functional dependency

- **Partially dependency**: if the dependency holds for part of X

Functional Dependency (contd.)

- **Transitive dependency**: If X, Y, and Z are sets of attributes in relation R, and X → Y, Y → Z, then X → Z.
  - We say that Z is transitively dependent on X.

- Example:
    - lecID → name, salary, deptCode
    - deptCode → dname, deptPhone, building
  - Then by transitive property
    - lecID → dname, deptPhone, building

Functional Dependency (contd.)

- Observation:
  - A functional dependency results in a redundancy unless the determinant of the functional dependency contains a key!

  - Explanation:
    - If X → Y exists in relation R and if X does not contain a key, then there can be multiple distinct tuples in R with the same values for X which also mean the same Y values repeat for these tuples.
    - However, if X contains a key, each tuple's X-value is unique. Therefore no redundancy based on the functional dependency.

Normal Forms

- Unnormalized Form: a table that contains one or more repeating groups
- There are many normal forms allowing varying degrees of redundancy

- We will study the following
  - 1st Normal Form
  - 2nd Normal Form
  - 3rd Normal Form
  - Boyce-Codd Normal Form

1st Normal Form

- A relation is in 1st normal form if each attribute value is a single, atomic value from its domain
- That is, an attribute value cannot be multiple or composite value
- By definition of relational model (i.e. domain constraint), every relation is in 1st normal form

1st Normal Form (contd.)

- Example: unnormalized relation

Student

stdID	name	Course	cName	semester	grade
S001	Paul Lee	INFT2040	Database Sys.	2	A
		INFT2031	Networks	3	B+
S002	Mary Smith	INFT1001	Found. of IT	1	A
		INFT2040	Database Sys.	2	A-

- Example: Relation in 1st Normal Form

Student

stdID	name	course	cName	semester	grade
S001	Paul Lee	INFT2040	Database Sys.	2	A
S001	Paul Lee	INFT2031	Networks	3	B+
S002	Mary Smith	INFT1001	Found. of IT	1	A
S002	Mary Smith	INFT2040	Database Sys.	2	A-

2nd Normal Form

- A relation R is in 2nd normal form if
  - R is in 1st normal form, and
  - Every non-candidate key attribute is fully functionally dependent on a candidate key

2nd Normal Form (contd.)

- Example

Student (<u>stdID, course, semester,</u> name, cName, grade)

Functional dependencies:

FD1: stdID → name (partial dependency)

FD2: course → cName (partial dependecy)

FD3: stdID, course, semester → grade

<u>stdID</u>	name	<u>course</u>	cName	<u>semester</u>	grade
S001	Paul Lee	INFT2040	Database Sys.	2	A
S001	Paul Lee	INFT2031	Networks	3	B+
S002	Mary Smith	INFT1001	Found. of IT	1	A
S002	Mary Smith	INFT2040	Database Sys.	2	A-

Lossless join decomposition

- Theorem

  Relation R_1 and R_2 *is* a lossless-join decomposition of relation S, *if* $R_1 \cap R_2$ (i.e. common attributes for R_1 and R_2) contains a key for either R_1 or R_2.

- Accordingly, if X → Y is causing an anomaly in relation R, we can decompose as follows:
  - Relation1: R-Y
  - Relation2: XY

  Relation1 $\cap$ Relation2 = {X} and X is key for Relation2

2^{nd} Normal Form (contd.)

- Example: Decomposing the Student relation
  - Step1: based on FD1

    Student1(stdID, name)

    Student2(stdID, course, semester, cName, grade)

    Not in 2^{nd} Normal Form Because of FD2

  - Step2: based on FD2

    Student1(stdID, name)

    Student2(course, cName)

    Student3(stdID, course, semester, grade)

    Decomposed relations in 2^{nd} Normal Form

Decomposing based on FD1 & FD2

- Student (stdID, course, semester, name, cName, grade)

FD1: stdID → name (partial dependency)

Step 0: find R, X, Y, in the definition:

R = stdID, course, semester, name, cName, grade

Now we want ***to fix a partial dependency FD1***, so

X = stdID, Y = name, XY = stdID, name

Step 1: get relation1: r1 (R-Y= stdID, course, semester, ~~name~~, cName, grade)

Step 2: get relation2: r2 (XY= stdID, name)

Now, based on r1, r2, and we want to eliminate FD2: course → cName:

Keep r2, convert r1 into 2 relations in a same way as on FD1:

X = course, Y = cName, XY = course, cName

Step 1: get relation1: r3 (R-Y= stdID, course, semester, ~~name~~, ~~cName~~, grade)

Step 2: get relation2: r4 (XY= course, cName)

Final result is r2, r3, r4: Student1(stdID, name), Student2(course, cName), Student3(stdID, course, semester, grade)

3rd Normal Form

- A relation R is in 3rd Normal Form if
  - R is in 2nd Normal Form, and
  - No non-candidate-key attribute is transitively dependent on a candidate key

3rd Normal Form (contd.)

- Example: Lecturer relation

  LecturerInfo(<u>lecID</u>, name, salary, deptCode, dname, deptPhone, building)

  Functional Dependencies:

  FD1: lecID → name, salary, deptCode (Primary Key)

  FD2: deptCode → dname, deptPhone, building (Transitive dependency)

- This relation is in 2nd Normal Form but not in 3rd Normal Form because of FD2.

3rd Normal Form (contd.)

- Decomposing Lecturer relation based on FD2:

  LecturerInfo(<u>lecID</u>, name, salary, deptCode, dname, deptPhone, building)

  - Method:
    - Form a new relation out of the transitive dependency
    - Keep the PK relation
  - Result:
    - Lecturer1(<u>deptCode</u>, dname, deptPhone, Building)
    - Lecturer2(<u>lecID</u>, name, salary, deptCode)
  - Decomposed relations are in 3rd Normal Form

Boyce-Codd Normal Form

- It is possible to have redundancy based on functional dependencies even when relations are in 3rd Normal Form (rare but possible!)

- Example:

  ClientInterview(<u>clientNo, interviewDate</u>, interviewTime, staffNo, roomNo)

  FD1: clientNo, interviewDate → interviewTime, staffNo, roomNo (Primary key)

  FD2: staffNo, interviewDate, interviewTime → clientNo (Candidate Key)

  FD3: roomNo, interviewDate, interviewTime → staffNo, clientNo (Candidate Key)

  FD4: staffNo, interviewDate → roomNo

Boyce-Codd Normal Form (contd.)

- A relation R is in BCNF if and only if
  - Every functional dependency, X → Y in R, X is a candidate key

- Because of FD1 – FD4, the relation is in 3rd Normal Form
- Because of FD4, the relation is not in BCNF!

Boyce-Codd Normal Form (contd.)

- Decomposing method:
  - Within original relation, for X->Y, where X is not a candidate key, keep the original relation but remove the Y; and form a new relation with X & Y, with X as PK
- Decomposing based on FD4:
  - ClientInterview1(<u>staffNo, interviewDate</u>, roomNo)
  - ClientInterview2(<u>clientNo, interviewDate</u>, staffNo, interviewTime)

The above relations are in BCNF!

However, **FD3 (**roomNo, interviewDate, interviewTime → staffNo, clientNo) **is lost**!!! (not preserving the dependency!)

- A dependency preserving decomposition to BCNF is not always possible!!! (Therefore may stop at 3NF)

- We can always get a lossless join and dependency preserving decomposition to 3rd Normal Form

Summary

- Redundancy
- Anomalies
- Lossless join decomposition
- Functional dependencies
- Normal Forms

Summary (cont'd) – 3NF or BCNF?

- Goal for a relational database normalization:
  - BCNF (no redundant information)
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept:
  - 3NF (possible repetition of information)
  - Lossless join
  - Dependency preservation
- Q: what's the relationship between BCNF and 2nd & 3rd Norm?

Lab This Week

- Review concepts of redundancy, anomaly, lossless join, function dependency, and normal forms.
- Based on given data sheet, work out EER, EER mapping, normalise to BCNF
- Assignment 2 planning

Assignment 2

- Starts now
- Due: week 8
- Make sure work on the EER & Relations revised
- <u>The specification</u>