Review of Last Lecture

1. Mapping of Regular Entity Types: table
2. Mapping of Multivalued attributes: new table + FK
3. Mapping of Weak Entity Types: table + FK
4. Mapping 1:1: merged, FK or new table + FKs
5. Mapping 1 to N: FK approach
6. Mapping N to M: new table + FKs
7. Mapping ternary and n-ary: new table + FKs

Note: If the relationship has simple attributes, they should be included in the new table or in the FK table.

Today's Plan

- Why redundancy is trouble
  - Update anomaly and NULL
- Functional Dependency
  - How to use key and FD to decide whether a design is good
- Decomposition
- Three main goals of schema refinement
  - How to check whether a decomposition is lossless
  - How check for FD preserving
  - How to check for minimal redundancy
- Different normal forms and BCNF

NORMALIZATION

CHAPTER 08

Collin Roberts

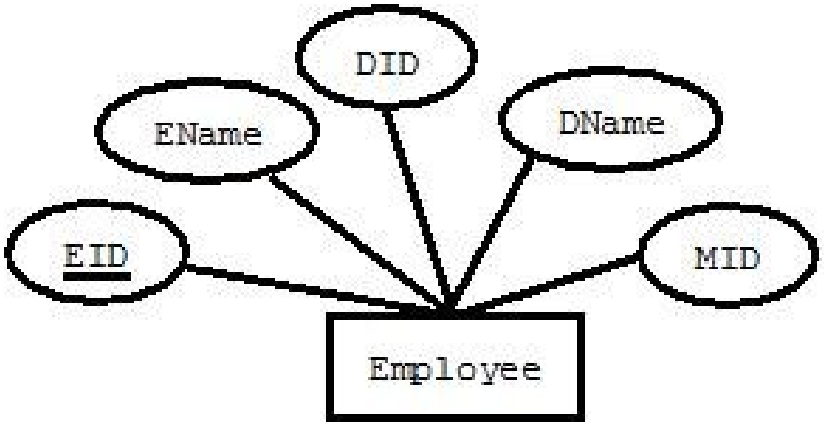
University of Waterloo

Recommended Readings

- Textbook Chapters 10 and 11

Motivation for Normalization

- Interview: “We need a DB to store employee info along with their department and manager … …”
- ERD



- Mapping

  Employee(EID, EName, DID, MID, DName)

- Is this good?

What is Wrong?

EMP(ENAME, SSN, BDATE, ADDRESS, DNUM, DNAME, DMGRSSN)

The description of the department (the name and the manager of the department) is repeated for every employee that works in that department.

Redundancy!

The department is described redundantly.
This leads to update anomalies

What is Wrong?

- Modification anomaly: what if Karen becomes the new manager in Marketing?
- Deletion anomaly: what if Jack resigns?
- Insertion anomaly: what if we want to create a new department called Public Relations?
- Insertion anomaly: what if Mike is hired but we have yet to decide what department to put him in?

ENAME	SSN	BDATE	ADDRESS	DNUM	DNAME	DMGRSSN
John	A1	…	…	1	Marketing	A2
Kate	A2	…	…	1	Marketing	A2
Karen	A3	…	…	1	Marketing	A2
Jack	A4	…	…	2	Sales	A4
NULL	NULL	NULL	NULL	3	PR	NULL
Mike	A5	…	…	NULL	NULL	NULL

Update Anomaly

Insertion anomalies:

if you insert an employee

need to know which department he/she works

need to know the descriptive information for that department.

if you want to insert a department, you can't...until there is at least one employee.

Deletion anomalies: if you delete an employee, is that dept. gone? was this the last employee in that dept.?

Modification anomalies: change DNAME, for example, need to change it everywhere! don't know where they are

Using Null

- May not store information about a department with no employees
- May waste space
- May make it hard to specify & understand joins
- May make it hard to aggregate (count, sum, …)
- May have different meanings:
  - attribute *does not apply* to this tuple
  - attribute value is *unknown*
  - value is *known but absent* (not yet recorded)
- Make it hard to interpret query answers

NULL values also cause problems

Schema Refinement Techniques

- Decomposition starts with an ER diagram, and uses the FDs to guide the schema design and decomposition, e.g., to replace a relation R(ABCD) with, two relations, say R1(AB) and R2(BCD) using the project operator.

- Synthesis is another refinement technique which takes all attributes over the original relation R, a set of FDs over these attributes, and constructs a *good* schema

Functional Dependency (Formally)

- Basis for normalization
- Another kind of *integrity constraints*: generalization of the notion of a key
- Let R be a relation schema, $\alpha \subseteq R$ *and* $\beta \subseteq R$. The functional dependency

  $$\alpha \rightarrow \beta$$

  holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

  $$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$
- If the value of the first attribute(s), A, is known, then the value of the second attribute, B, is known (i.e., determined, nailed down)
- If a relation R is legal under a set F of FDs, we say R satisfies F

Functional Dependency (Informally)

- If attributes α and β come from the same relation R, and the value of α determines the value of β, we say β is functionally depends on α, denoted as

$$\alpha \rightarrow \beta$$

- In other words, if we know the value for α, we know the value for β.
- The dependency could be among more than two attributes, on both sides!

FD: Example

- Examples of functional dependencies:

employee-number → employee-name

course-number → course-title

- Examples that are NOT functional dependencies

employee-name → employee-number X

course-number → book X

course-number → car-color X

Basic Rules of FD

- Note (A, B) is sometimes written as (AB)
- A -> A
  - Example: Name -> Name
  - This is called a trivial FD
- If A -> B, then (AC) -> B
  - Example: SID -> SName, so (SID, GPA) -> SName
  - This is also called a trivial FD
- If A -> B and X -> B, then (AX) -> B.
  - Example: SIN -> SName and Email -> SName, so (SIN, Email) -> SName
- If A -> B and B -> C, then A -> C
  - Example: SSN -> DNO and DNO -> DName, so SSN -> DName

Good Design?

Employee(SIN,Fname,Lname,Birthday,DNO,Salary,Phone, Address)

- What are the CKs?

SIN

- What are the FDs?

SIN -> Fname,Lname,Birthday,DNO,Salary,Phone, Address

Is this a good design?

Good Design?

Student(SID,E-mail,Sname,Program,GPA,Standing,StartYear,Address)

- What are the CKs?

SID

E-mail

Is this a good design?

- What are the FDs?

SID -> E-mail,Sname,Program,GPA,Standing, StartYear,Address

E-mail -> SID, Sname,Program,GPA,Standing,StartYear,Address

Good Design?

Employee(ENAME,SIN,BDATE,ADDRESS,DNUM,DNAME,MGRSIN)

- What are the CKs?

SIN

Is this a good design?

- What are the FDs?

SIN -> ENAME,BDATE,ADDRESS,DNUM,DNAME,MGRSIN

DNUM -> DNAME, MGRSIN

MGRSIN - > DNUM, DNAME

Good Design?

Std_Course(SID,E-mail,Sname, Cnum, Cname,Pre-req,Grade)

- What are the CKs?

(SID, Cnum)

(E-mail, Cnum)

- What are the FDs?

(SID, Cnum) - > Everything

(E-mail, Cnum) -> Everything

Cnum -> Cname, Pre-req

SID -> Email, Sname

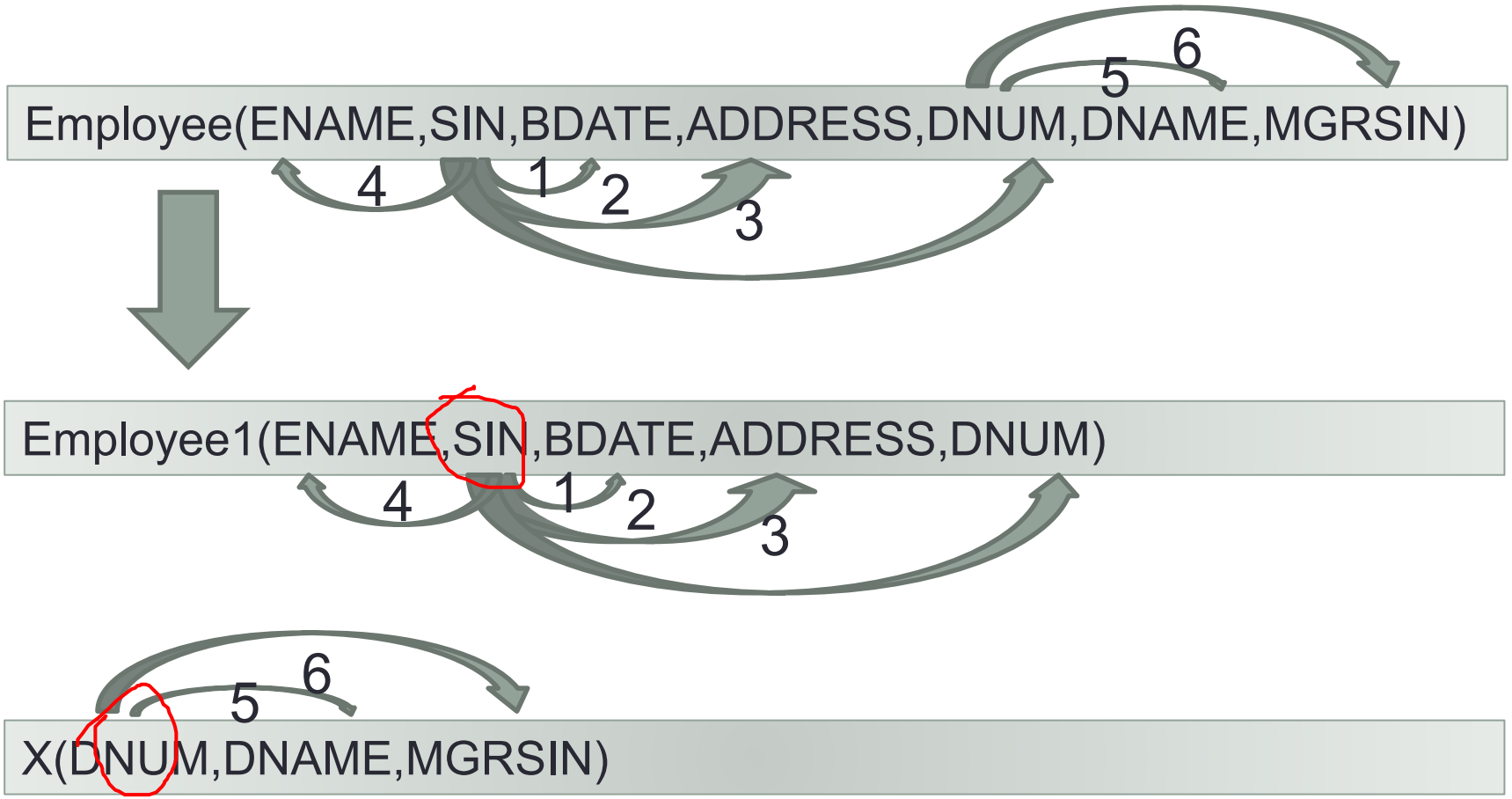
E-mail -> SIDm, Sname

Is this a good design?

Do you see a pattern?

- In all good designs, all FDs are based on CKs (left-hand side of the FD is always a CK)
- In all bad designs, not all FDs are based on CKs (left-hand side of the FD is NOT always a CK)
- Normalization approach
  - Break up a table until all FDs are based on CKs.
    - Say, T(ABCDEFG) becomes T1(ACD) and T2(BEFG)
  - The next slide presents a complete example

How to fix a bad design by decomposition



What would be a good name for table X?

Goal of Decomposition

We have three goals:

lossless decomposition
(don't throw any information away)
(be able to reconstruct the original relation)

dependency preservation
all of the non-trivial FDs each end up in just one relation (not split across two or more relations)

Boyce-Codd normal form (BCNF) - no redundancy beyond foreign keys; all FDs implied by keys

Candidate Key: Review

Employee(SSN, NAME, EID)

If the attribute EID is unique for each employee, then either SSN or EID could be a key. Thus, they are both *candidate keys*. However, only one of them may be designated the *primary key*.

Normalization considers all candidate keys of a relation, not just the primary key.

Keys and Key Attributes

Assign(SSN, PROJECT, S_DATE)

Here the attributes SSN and PROJECT taken together is the *key*. SSN is a *key attribute*; so is PROJECT. S_DATE is a *non-key attribute*.

EMPLOYEE (SSN, NAME, SALARY, JOB_DESC)

Here the attribute SSN is the *key*. SSN is a *key attribute*. All other attributes are *non-key attributes*.

Another way to view the key: there is a FD from the key to every attribute in the relation

Lossless vs. Lossy Decomposition

When R is decomposed into R_1 and R_2

Check to see if $(R_1 * R_2) = R$

if it is a lossy decomposition, then $R_1 * R_2$ gives you TOO MANY tuples.

Example: Lossy Decomposition

original

Employee(SS-number, name, project, p-title)

1	smith	p1	accounting
2	jones	p1	accounting
3	smith	p2	billing

decomposition:

Employee (SS-number, name)

1	smith
2	jones
3	smith

Project (project, p-title, name)

p1	account	smith
p1	account	jones
p2	billing	smith

now if we join them with natural join, what happens?

you get at least one extra tuple!!!

1	smith	p2	billing

Test for Lossless Decomposition

Let R_1 and R_2 form a decomposition of R.

R_1 and R_2 are both sets of attributes from R.

The decomposition is lossless if ...

> the attributes in common are a key for *at least one* of the relations R_1 and R_2

Example I: Lossless Test

Employee(SS-number, name, project, p-title)

decomposition: Employee (SS-number, name)
Project (project, p-title, name)

Which attribute is in common? name (of the employee)

Is name a key for either of these two tables?

NO! We have a problem.

Example II: Lossless Test

Employee(SS-number, name, project, p-title)

decomposition: Employee (SS-number, name, project)

Project (project, p-title)

Which attribute is in common? project

Is project a key for either of these two tables?

Yes!

Example III: Lossless Test

Employee(SS-number, name, project, p-title)

decomposition: Employee (SS-number, name)
Project (project, p-title)

Which attribute is in common? None

We have a problem (unless the original Employee relation did not mean to associate an employee with a project).

Lossless Exercise

- Three possible decompositions for relation TEACH. Which one(s) is(are) lossless?
  1. {student, instructor} and {student, course}
  2. {course, instructor } and {course, student}
  3. {instructor, course } and {instructor, student}

TEACH

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

Dependency Preserving

Suppose F is the original set of FDs and G is set of FDs after decomposition

The decomposition is dependency preserving if $F^+ \equiv G^+$. That is, the two closures must be equivalent.

*We will discuss the concept of closure (F^+) in the next lecture

Simple Way

- One simple way to check whether a decomposition is FD preserving:
  - After the decomposition, for each FD ($A \rightarrow B$), check whether both A and B are in the same relation
    - If yes, definitely yes
    - If no, likely no
      - Based on those FDs that are preserved, can you deduce the ones that were destroyed?

Decomposition: Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - Can be decomposed in two different ways
- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing R_1 **join** R_2)

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.

Normalization and Normal Form

- **Normalization**: The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations which are in “certain” normal form

- **Normal form**: Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

Assumptions

- During normalization, it is always assumed the following information is given:
  - Relations with attributes, PKs, CKs, FKs and
  - FDs!
- Where do FDs come from? From the semantics of the problem!
  - It is the responsibility of the user and DB developer to figure out together what the FDs are!

Levels of Normal Form

- 2NF, 3NF, BCNF based on keys and FDs of a relation schema
- 4NF based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs
  - Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation)
- In this class, we only focus on BCNF as it is one of the most common NFs used in practice.

Important Normal Form

1NF - all attribute values (domain values) are atomic
(part of the definition of the relational model)
2NF - all non-key attributes must depend on a **whole** candidate key
(no partial dependencies)
3NF – table is in 2NF and all non-key attributes must depend on **only** a candidate key (no transitive dependencies)

BCNF - every determinant is a superkey, X --> A, X is a superkey

BCNF >> 3NF >> 2NF >> 1NF

Normalization Made Easy (Ronald Fagin)

Every attribute must depend
upon the key, ← definition of key
the whole key, ← 2NF
and
nothing but the key. ← BCNF (and 3NF)

In-class Exercise: R in BCNF?

Given: R = { S, B, D, C } (or just SBDC)

1 Key = { SBD } F = { S→C }

- ?

2 Key = { SBD, CBD } F = { S→C }

- ?

3 Key = { SBD } F = { SBD→C }

- ?

BCNF?

Employee(SIN,Fname,Lname,Birthday,DNO,Salary,Phone, Address)

- What are the CKs?

SIN

- What are the FDs?

SIN -> Fname, Lname,
SIN -> SIN,Fname,Lname,Birthday,DNO,Salary,Phone, Address

BCNF?

Student(SID,E-mail,Sname,Program,GPA,Standing,StartYear,Address)

- What are the CKs?

SID

E-mail

- What are the FDs?

SID->E-mail,Sname,Program,GPA,Standing,StartYear,Address

E-mail->SID, Sname,Program,GPA,Standing,StartYear,Address

BCNF?

Employee(ENAME,SIN,BDATE,ADDRESS,DNUM,DNAME,MGRSIN)

- What are the CKs?

SIN

- What are the FDs?

SIN->ENAME,BDATE,ADDRESS,DNUM,DNAME,MGRSIN

DNUM->DNAME,MGRSIN

BCNF?

Std_Course(SID,E-mail,Sname, Cnum, Cname,Pre-req,Grade)

- What are the CKs?

(SID, Cnum)

(E-mail, Cnum)

- What are the FDs?

(SID, Cnum)-> E-mail,Sname,Cname,Pre-req,Grade

(E-mail, Cnum)-> SID,Sname,Cname,Pre-req,Grade

Cnum->Cname,Pre-req

SID->,E-mail,Sname

E-mail->SID, Sname

Questions About Decomposition

- When to decompose
- How to come up with a correct decomposition

Answer: BCNF

- A relation *R* is in Boyce-Codd Normal Form if
  - For every non-trivial X → Y, X is a super key
  - That is, all FDs follow from "key → other attributes"
  - Non-trivial: Y is not a member of X
- When to decompose
  - As long as some relation is not in BCNF
- How to come up with a correct decomposition
  - Always decompose on a BCNF violation

☞Then it is guaranteed to be a correct decomposition!

Summary of Today's Lecture

- Why redundancy is trouble
  - Update anomaly and NULL
- Functional Dependency
  - How to use key and FD to decide whether a design is good
- Decomposition
- Three main goals of schema refinement
  - How to check whether a decomposition is lossless
  - How check for FD preserving
  - How to check for minimal redundancy
- Different normal forms and BCNF