CS322: Database Systems

Relational Database Design

Dr. Manas Khatua
Assistant Professor
Dept. of CSE
IIT Jodhpur
E-mail: manaskhatua@iitj.ac.in

Introduction

- Conceptual data model (e.g. ER Model) teaches
  - entity types,
  - relationship types, and
  - their respective attributes
- Each relation schema consists of a number of attributes
- the relational database schema consists of a number of relation schemas
- Common understanding from the previous:
  - attributes are grouped to form a relation schema
- How to measure the quality / goodness of the design?

Cont…

- two levels to understand the goodness of a relation schema
  - logical (or conceptual) level
    - how users interpret the relation schemas and the meaning of their attributes
    - enables users to understand clearly the meaning of the data in the relations
  - implementation (or physical storage) level
    - how the tuples in a base relation are stored and updated
    - enables users for systematic storing, updating, and accessing the data in the relations

Cont…

- database design is performed using two approaches:
  - bottom-up (design by Synthesis)
    - considers the basic relationships among individual attributes as the starting point
    - it suffers from a large number of binary relationships among attributes as the starting point.
  - top-down (design by Analysis)
    - starts with a number of groupings of attributes into relations that exist together naturally
    - The relations are then analyzed individually and collectively, leading to further decomposition
- implicit goals of the design activity
  - information preservation
    - in terms of maintaining all concepts, including attribute types, entity types, and relationship types as well as generalization/specialization relationships
  - minimum redundancy
    - minimizing redundant storage of the same information
    - reducing the need for multiple updates to maintain consistency

Redundant Information

- Suppose we combine *instructor* and *department* into *inst_dept* relationship

  inst_dept = (*Instructor ID*, *name*, *salary*, *dept_name*, *building*, *budget*)

- Result is possible repetition of information

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Informal Design Guidelines

- Four informal guidelines
  - Making sure that the semantics of the attributes is clear in the schema
  - Reducing the redundant information in tuples
  - Reducing the NULL values in tuples
  - Disallowing the possibility of generating spurious tuples

- A simplified COMPANY relational database schema

EMPLOYEE = {Ename, Ssn, Bdate, Address, *Dnumber*}
DEPARTMENT = {Dname, Dnumber, *Dmgr_ssn*}
DEPT_LOCATIONS = {*Dnumber*, Dlocation}
PROJECT = {Pname, Pnumber, Plocation, *Dnum*}
WORKS_ON = {*Ssn*, *Pnumber*, Hours}

*italic font represents F.K.

Cont…

- Let a database design as follows:

- All attributes:

{ Ename, Ssn, Bdate, Address, Dnumber, Dname, Dlocation, Dmgr_ssn, Pnumber, Pname, Plocation, Hours }

- Database Schema

EMP_DEPT = {Ename, Ssn, Bdate, Address, Dnumber, Dname, Dlocation, Dmgr_ssn}

EMP_PROJ = {Ssn, Pnumber, Hours, Ename , Pname, Plocation}

- Observations:
  - there is nothing wrong logically with these two relations
  - They may be good as views
  - But, they cause problems when used as base relations

Database Anomalies

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston

- Insertion Anomaly
- Deletion Anomaly
- Modification Anomaly

Spurious Tuples

- Suppose that we used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ.

EMP_PROJ = {Ssn, Pnumber, Hours, Ename , Pname, Plocation}

EMP_LOCS = {Ename, Plocation}
EMP_PROJ1 = {Ssn, Pnumber, Hours, Pname, Plocation}

- Logically they are not incorrect. But, we cannot recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS.
- After performing NATURAL JOIN operation, the generated additional tuples that were not in EMP_PROJ are called **spurious tuples**
- **So,** avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Summary of Design Guidelines

1. Avoid redundant information which creates anomalies in tuple insertion, deletion, and modification

2. Avoid NULL values which yields waste of storage space and creates difficulty of performing selections, aggregation, and joins

3. Decompose a relation schema based upon the primary key, foreign key relationship to avoid the generation of invalid and spurious tuples

Functional Dependencies

- A functional dependency (FD) is a constraint between two sets of attributes from the database.

- Let *R* be a relation schema
$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

- 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,
$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider *r*(A,*B*) with the following instance of *r.*

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

- Hence, the main use of functional dependencies is to describe a relation schema *R* by specifying constraints on its attributes that must hold *at all times*.

Cont…

- *K* is a superkey for relation schema *R* if and only if
  - $K \rightarrow R$
- *K* is a candidate key for *R* if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

- One of the candidate keys is designated to be the primary key, and the others are called secondary keys.

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.

- Consider the schema:
  inst_dept (*ID*, *name*, *salary*, *dept_name*, *building*, *budget*).

  We expect these functional dependencies to hold:

  dept_name $\rightarrow$ *building*

  and, *ID* $\rightarrow$ *building*

  but would not expect the following to hold:

  dept_name $\rightarrow$ *salary*

Use of FD

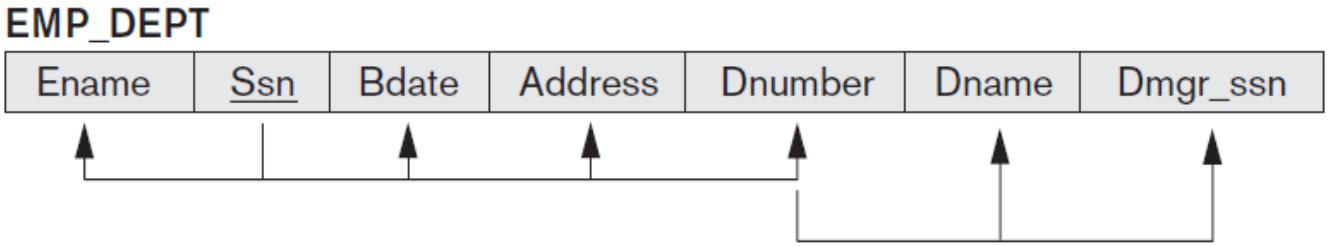
- We use FDs to:
  - test relations to see if they are legal under a given set of FDs.
    - If a relation *r* is legal under a set *F* of FDs, we say that *r* **satisfies** *F.*
  - specify constraints on the set of legal relations
    - We say that *F* **holds on** *R* if all legal relations on *R* satisfy the set of FDs *F.*

- Note:
  - A specific instance of a relation schema may satisfy a FD even if the FD does not hold on all legal instances.
  - For example,

  a specific instance of *instructor* may, by chance, satisfy
  name $\rightarrow$ *ID.*

Closure of a Set of FDs

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the *closure* of *F* by **F^+**.
- F^+ is a superset of *F*.



- F^+ = {Ssn $\rightarrow$ Ename, Bdate, Address, Dnumber
  Dnumber $\rightarrow$ Dname, Dmgr_ssn
  Ssn $\rightarrow$ Dname, Dmgr_ssn

Cont…

- We can find F^{+}, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**

- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).

- Additional rules:
  - If $\alpha \rightarrow \beta$ holds *and* $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds *and* $\gamma\,\beta \rightarrow \delta$ holds, then $\alpha\,\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

Example (Closure of a Set of FDs)

- $R = (A, B, C, G, H, I)$

$F = \{ A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H\}$

To test whether the decomposition satisfies the dependency preservation, we need to find the all FDs functionally determined by the given set of FDs.

- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

Example (Closure of an Attribute Set)

- $R = (A, B, C, G, H, I)$
  $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$

To test whether a set of attributes, say (AG), is a superkey, we need to find the set of attributes functionally determined by (AG).

- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. result = ABCGHI (CG $\rightarrow$ I and CG $\subseteq$ AGBCH)

- Is *AG* a candidate key?

  1. Is AG a super key?
     1. Does $AG \rightarrow R?$ == Is $(AG)^+ \supseteq$ R
  2. Is any subset of AG a super key?
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq$ R
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq$ R

Canonical Cover

- Sets of FDs may have redundant dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, \quad B \rightarrow C, A \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
    - E.g.: on LHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
- Intuitively, a canonical cover of F is a "minimal" set of FDs equivalent to F, having no redundant dependencies or redundant parts of dependencies.
- What is the use of it?

To minimize the number of FDs that need to be tested in case of an update in the relation, we may restrict F to a canonical cover Fc.

Extraneous Attributes

- Consider a set *F* of FDs and the FD $\alpha \rightarrow \beta$ in *F*.
  - Attribute A is **extraneous** in α if $A \in \alpha$ and *F* logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute *A* is **extraneous** in β if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies *F.*
- *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one
- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - *B* is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping *B* from $AB \rightarrow C$).
- Example: Given $F = \{AB \rightarrow C, AB \rightarrow CD\}$
  - *C* is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting *C*

Testing if an Attribute is Extraneous

- Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.

- To test if attribute $A \in \alpha$ is extraneous in α
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
  2. check that $(\{\alpha\} - A)^+$ contains β; if it does, *A* is extraneous in α

- To test if attribute $A \in \beta$ is extraneous in β
  1. compute α^+ using only the dependencies in
     $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
  2. check that α^+ contains *A;* if it does, *A* is extraneous in β

Canonical Cover

- A **canonical cover** for *F* is a set of dependencies F_c such that
  - *F* logically implies all dependencies in F_c, and
  - F_c logically implies all dependencies in *F*, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique.

- To compute a canonical cover for *F*:
  repeat
    Use the union rule to replace any dependencies in *F*
      $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$
    Find a functional dependency $\alpha \rightarrow \beta$ with an
      extraneous attribute either in α or in β
    /* Note: test for extraneous attributes done using F_c, not F*/
    If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
  until *F* does not change

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Example (Canonical Cover)

- $R = (A, B, C)$

  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$

- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

- A is extraneous in $AB \rightarrow C$
  - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
    - Yes: in fact, $B \rightarrow C$ is already present!
  - Set is now $\{A \rightarrow BC, B \rightarrow C\}$

- C is extraneous in $A \rightarrow BC$
  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
    - Yes*:* using transitivity on $A \rightarrow B$ *and* $B \rightarrow$ C.
      - Can use attribute closure of A in more complex cases

- The canonical cover is: $A \rightarrow B$
  $B \rightarrow C$

Thanks!