PSPT MGR GOVERNMENT ARTS AND SCIENCE COLLEGE
SIRKALI – PUTHUR

LECTURE NOTES
ON
DATABASE SYSTEMS
16SCCCS4

M.PRIYA,
ASSISTANT PROFESSOR,
DEPARTMENT OF COMPUTER SCIENCE,
PSPT MGR GOVERNMENT ARTS AND SCIENCE COLLEGE,
SIRKALI

CORE COURSE IV

DATABASE SYSTEMS

SEMESTER - IV

Unit I

Introduction: Database-System Applications- Purpose of Database Systems - View of Data Database Languages - Relational Databases - Database Design -Data Storage and Querying Transaction Management -Data Mining and Analysis - Database Architecture -Database Users and Administrators - History of Database Systems.

Unit II

Relational Model: Structure of Relational Databases -Database Schema - Keys – Schema Diagrams - Relational Query Languages - Relational Operations Fundamental Relational-Algebra Operations Additional Relational-Algebra Operations- Extended Relational-Algebra Operations - Null Values - Modification of the Database.

Unit III

SQL:Overview of the SQL Query - Language - SQL Data Definition - Basic Structure of SQL Queries - Additional Basic Operations - Set Operations - Null Values Aggregate Functions - Nested Subqueries - Modification of the Database -Join Expressions - Views - Transactions -Integrity Constraints - SQL Data Types and Schemas - Authorization

Unit IV

Relational Languages: The Tuple Relational Calculus - The Domain Relational Calculus Database Design and the E-R Model: Overview of the Design Process - The Entity-Relationship Model - Reduction to Relational Schemas - Entity-Relationship Design Issues - Extended E-R Features - Alternative Notations for Modeling Data - Other Aspects of Database Design

Unit V

Relational Database Design: Features of Good Relational Designs - Atomic Domains and First Normal Form - Decomposition Using Functional Dependencies - Functional-Dependency Theory - Decomposition Using Functional Dependencies - Decomposition Using Multivalued Dependencies-More Normal Forms - Database-Design Process

Text Book:

1. Database System Concepts, Sixth edition, Abraham Silberschatz, Henry F. Korth, S. Sudarshan, McGraw-Hill-2010.

Reference Books:

1. Database Systems: Models, Languages, Design and Application, Ramez Elmasri, Pearson Education 2014

Relational Database Design

In general, the goal of relational database design is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily. This is accomplished by designing schemas that are in an appropriate *normal form*. In this chapter, we introduce a formal approach to relational database design based on the notion of functional dependencies.

Features of Good Relational Designs

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

  (No connection to relationship set inst_dept)

- 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

A Combined Schema without Repetition

Consider combining relations

- *sec_class(sec_id, building, room_number)* and
- *section(course_id, sec_id, semester, year)*

into one relation

- *section(course_id, sec_id, semester, year, building, room_number)*

- No repetition in this case

Suppose we had started with *inst_dept.* How would we know to split up (decompose) it into *instructor* and *department*? Write a rule "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key" Denote as a functional dependency:

dept_name $\rightarrow$ *building, budget*

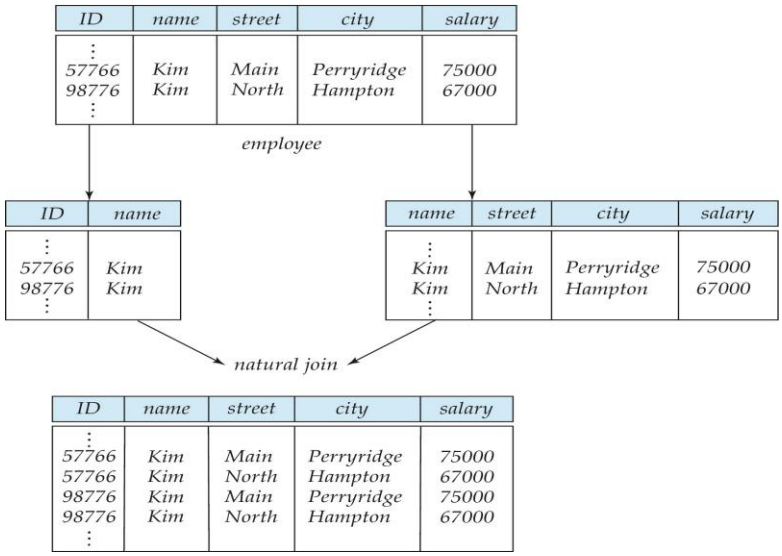
In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated. This indicates the need to decompose *inst_dept* . Not all decompositions are good. Suppose we decompose

employee(ID, name, street, city, salary) into

employee1 (*ID*, *name*)

employee2 (*name*, *street, city, salary*)

Clearly, we would like to avoid such decompositions. We shall refer to such decompositions as being lossy decompositions, and, conversely, to those that are not as lossless decompositions.



Loss of information via a bad decomposition.

Atomic Domains and First Normal Form

The E-R model allows entity sets and relationship sets to have attributes that have some degree of substructure. When we create tables from E-R designs that contain these types of attributes, we eliminate this substructure.

For composite attributes, we let each component be an attribute in its own right. Formultivalued attributes,we create one tuple for each item in amultivalued set. In the relationalmodel, we formalize this idea that attributes do not have any substructure. A domain is atomic if elements of the domain are considered to be indivisible units. We say that a relation schema *R* is in first normal form (1NF) if the domains of all attributes of *R* are atomic.

Aset of names is an example of a nonatomic value. For example, if the schema of a relation *employee* included an attribute *children* whose domain elements are sets of names, the schema would not be in first normal form. Composite attributes, such as an attribute *address* with component attributes *street*, *city*, *state*, and *zip* also have nonatomic domains.

Non-atomic values complicate storage and encourage redundant (repeated) storage of data.

Goal

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
    - each relation is in good form
    - the decomposition is a lossless-join decomposition
- Our theory is based on:
    1. functional dependencies
    2. multivalued dependencies

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*

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.

- *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 $\alpha \subset K$, $\alpha \rightarrow R$
- 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→ building and ID → building*
  - but would not expect the following to hold:
    - *dept_name → salary*

Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation *r* is legal under a set *F* of functional dependencies, 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 functional dependencies *F.*
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.

For example, a specific instance of *instructor* may, by chance, satisfy

$name \rightarrow ID.$

- *A* functional dependency is trivial if it is satisfied by all instances of a relation

Example*:*

- $ID, name \rightarrow ID$
- $name \rightarrow name$

In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Closure of a Set of Functional Dependencies

- 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*.

Boyce-Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

$\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)

α is a superkey for *R*

Example schema *not* in BCNF:

instr_dept (*ID, name, salary, dept_name, building, budget*) because *dept_name*$\rightarrow$ *building, budget* holds on *instr_dept,* but *dept_name* is not a superkey .

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF. We decompose *R* into:
    - (α U β)
    - (*R* - (β - α))

In our example,

α = *dept_name*

β = *building, budget*

and *inst_dept* is replaced by

(α U β) = (*dept_name, building, budget*)

(*R* - (β - α)) = (*ID, name, salary, dept_name*)

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation

- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving.*
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form.*

Third Normal Form

- A relation schema *R* is in third normal form (3NF) if for all:
    - $\alpha \rightarrow \beta$ in F^+

      at least one of the following holds:
    - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
    - α is a superkey for *R*
    - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R.*
- (NOTE*:* each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).

Goals of Normalization

- Let *R* be a relation scheme with a set *F* of functional dependencies.
- Decide whether a relation scheme *R* is in "good" form.
- In the case that a relation scheme *R* is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
    - each relation scheme is in good form
    - the decomposition is a lossless-join decomposition
    - Preferably, the decomposition should be dependency preserving.

Functional-Dependency Theory

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF
- We then develop algorithms to test if a decomposition is dependency-preserving.

Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - For e.g.: 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^+.

Closure of a Set of Functional Dependencies

- 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).

Example

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

$A \rightarrow C$

$CG \rightarrow H$

$CG \rightarrow I$

$B \rightarrow H\}$

- 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

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F:

$F^+ = F$

repeat

for each functional dependency f in F^+

apply reflexivity and augmentation rules on f

add the resulting functional dependencies to F^+

for each pair of functional dependencies $f_1$and f_2 in F^+

if f_1 and f_2 can be combined using transitivity

then add the resulting functional dependency to F^+

until F^+ does not change any further

Design Goals

- Goal for a relational database design is:
    - BCNF.
    - Lossless join.
    - Dependency preservation.
- If we cannot achieve this, we accept one of
    - Lack of dependency preservation
    - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.
    - Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.

Multivalued Dependencies

- Suppose we record names of children, and phone numbers for instructors:
    - *inst_child*(*ID*, *child_name*)
    - *inst_phone*(*ID*, *phone_number*)
- If we were to combine these schemas to get
    - *inst_info*(*ID*, *child_name*, *phone_number*)

Let *R* be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency

$$\alpha \rightarrow\rightarrow \beta$$

holds on *R* if in any legal relation *r(R),* for all pairs for tuples t_1 and t_2 in *r* such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in *r* such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:

- To test relations to determine whether they are legal under a given set of functional and multivalued dependencies
- To specify constraints on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.

- If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations *r′* that does satisfy the multivalued dependency by adding tuples to *r.*

Fourth Normal Form

- A relation schema *R* is in 4NF with respect to a set *D* of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:

  - $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema *R*

- If a relation is in 4NF it is in BCNF

4NF Decomposition Algorithm

result: = {*R*};
done := false;
compute D^+;
Let D_i denote the restriction of D^+ to R_i

while (not *done*)
if (there is a schema R_i in *result* that is not in 4NF) then
begin

let $\alpha \rightarrow\rightarrow \beta$ be a nontrivial multivalued dependency that holds

on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \phi$;

result := (*result* - R_i) $\cup$ (R_i - β) $\cup$ (α, β);

end

else *done*:= true;

Note: each R_i is in 4NF, and decomposition is lossless-join

Further Normal Forms

- Join dependencies generalize multivalued dependencies
  - lead to project-join normal form (PJNF) (also called fifth normal form)
- A class of even more general constraints, leads to a normal form called domain-key normal form.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables.
  - *R* could have been a single relation containing *all* attributes that are of interest (called universal relation).
  - Normalization breaks *R* into smaller relations.
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
    - Example: an *employee* entity with attributes
      department_name and *building*,
      and a functional dependency
      department_name $\rightarrow$ *building*
    - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Modeling Temporal Data

- Temporal data have an association time interval during which the data are *valid.*
- A snapshot is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
    - attributes, e.g., address of an instructor at different points in time
    - entities, e.g., time duration when a student entity exists
    - relationships, e.g., time during which an instructor was associated with a student as an advisor.
- But no accepted standard
- Adding a temporal component results in functional dependencies like

  ID $\rightarrow$ *street, city*

  not to hold, because the address varies over time

- A temporal functional dependency X → Y holds on schema *R* if the functional dependency X → Y holds on all snapshots for all legal instances r (*R*).