UCS310
Database Management System

Relational Design

Lecture-16

Date: 16 Feb 2023

Dr. Sumit Sharma
Assistant Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala

Recap

- Relation mapping
  - Simple, composite, multivalued attributes
  - Strong entity
  - Weak entity
  - 1:1, 1:M, M:N binary relations
  - Unary relation
  - Ternary relation
  - IS_A hierarchy (superclass, subclass)
- ER mapping procedure with an example
- Case study

Features of a Good Relational Design

- Suppose we combine ***instructor*** and ***department*** into ***in_dep***, which represents the natural join on the relations *instructor* and *department*

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

- There is a repetition of information
- Need to use null values (if we add a new department with no instructors)

A Combined Schema Without Repetition

Not all combined schemas result in the repetition of information

- 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

Design Guidelines for Relational Design

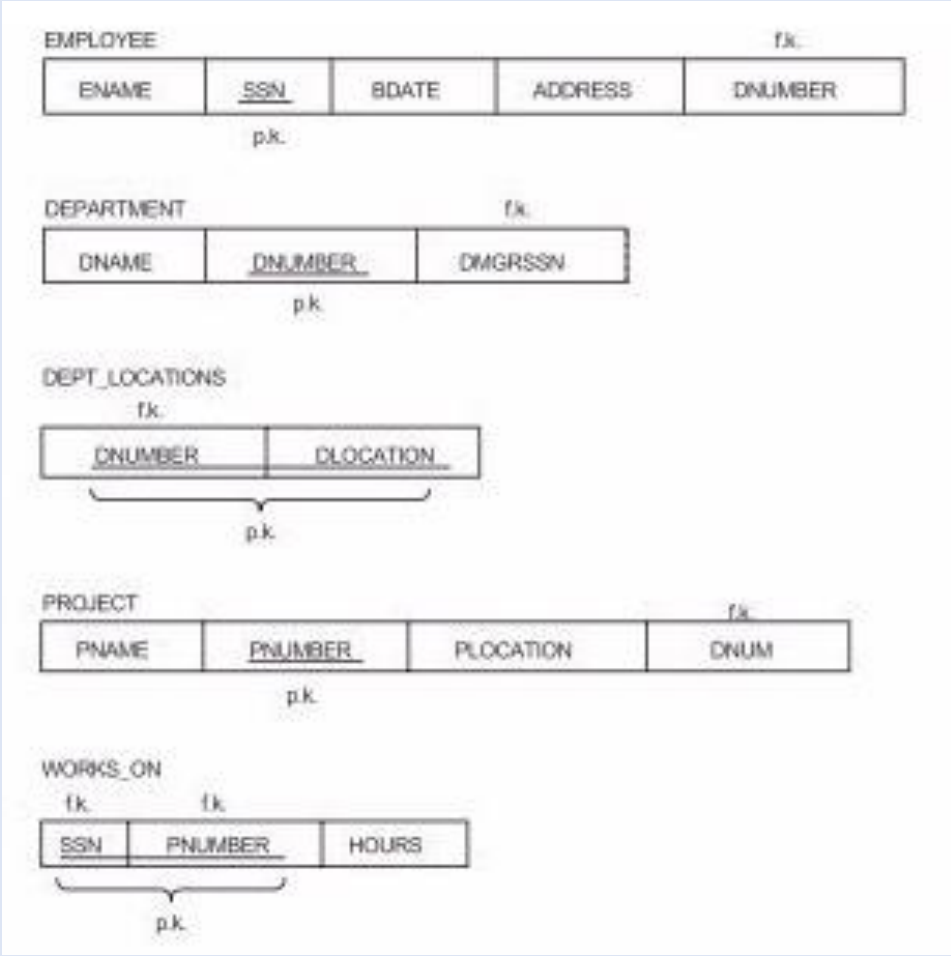
- What is relational database design?
  - The grouping of attributes to form "good" relation schemas
- Two levels of relation schemas
  - The logical "user view" level
  - The storage "base relation" level
- What are the criteria for "good" base relations?

Semantics of the Relation Attributes

GUIDELINE 1:

Each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes)

- Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities
- Entity and relationship attributes should be kept apart as much as possible.



Semantics of the Relation Attributes

GUIDELINE 1:

Each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes)

- Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities
- Entity and relationship attributes should be kept apart as much as possible.

Bottom Line:

Design a schema that can be explained relation by relation easily.

The semantics of attributes should be easy to interpret.

Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems
- Information is stored redundantly, wasting storage
- Problems with update anomalies
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

EXAMPLE OF AN UPDATE ANOMALY

Consider the relation:

EMP_PROJ (Emp#, Proj#, Ename, Pname, No_hours)

- **Update Anomaly:** Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

EXAMPLE OF AN UPDATE ANOMALY

- **Insert Anomaly:** Cannot insert a project unless an employee is assigned to

  Inversely - Cannot insert an employee unless an he/she is assigned to a project

- **Delete Anomaly:** When a project is deleted, it will result in deleting all the employees who work on that project. Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project

Guideline to Redundant Information in Tuples and Update Anomalies

- **GUIDELINE 2:**

  Design a schema that does not suffer from insertion, deletion and update anomalies.

  If there are any present, then note them so that applications can be made to take them into account

Null Values in Tuples

GUIDELINE 3:

Relations should be designed such that their tuples will have as few NULL values as possible

- Attributes that are NULL frequently could be placed in separate relations (with the primary key)

- Reasons for nulls:
  - attribute not applicable or invalid
  - attribute value unknown (may exist)
  - value known to exist, but unavailable

Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

Decomposition

- The only way to avoid the repetition-of-information problem in the i*n_dep* schema is to decompose it into two schemas – instructor and *department* schemas.
- Not all decompositions are good. Suppose we decompose

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

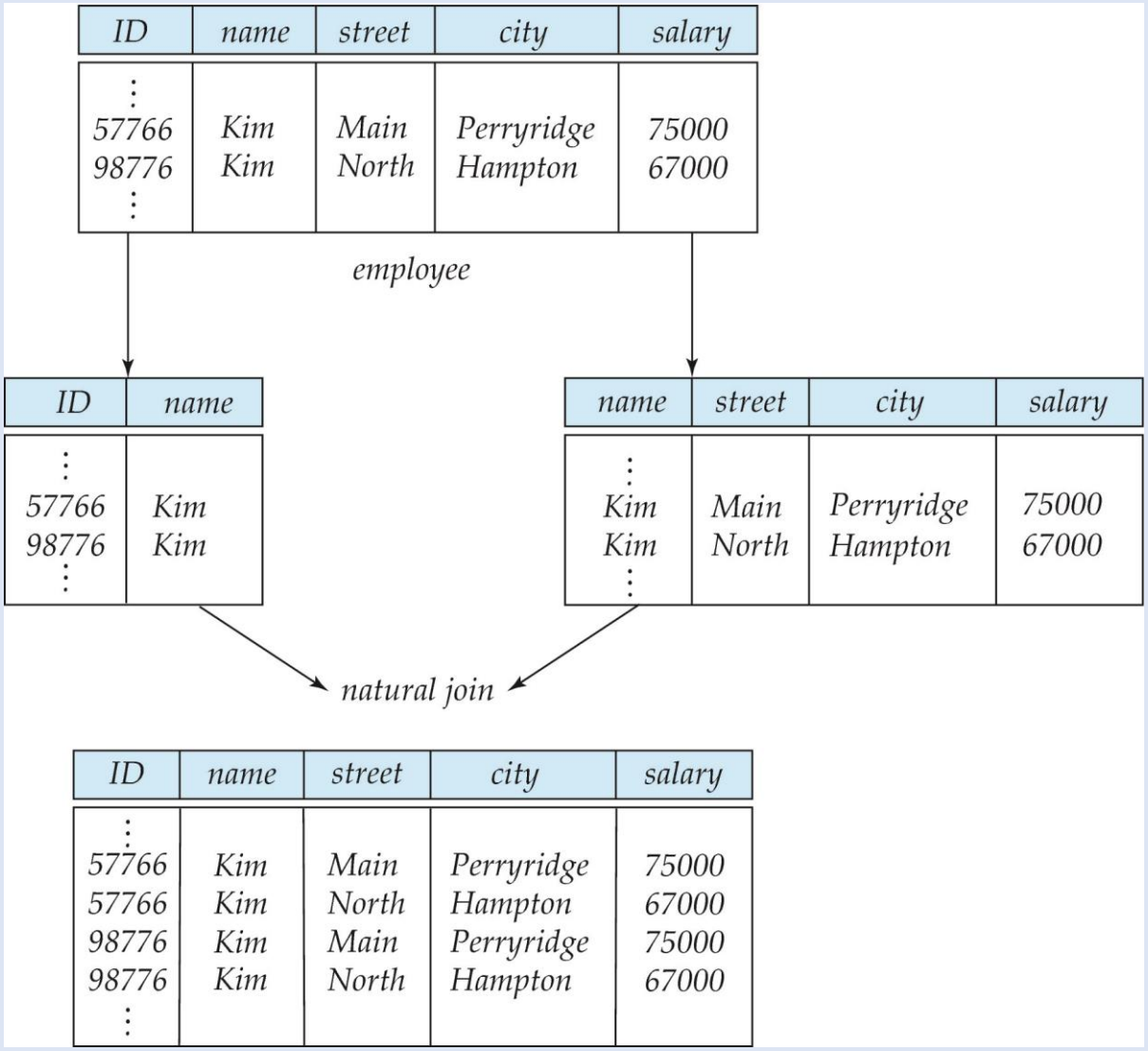
  into

  employee1 (*ID*, *name*)

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

  The problem arises when we have two employees with the same name
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.

A Lossy Decomposition



Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R . That is R = R_1 U R_2
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas R_1 U R_2

- Formally,

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

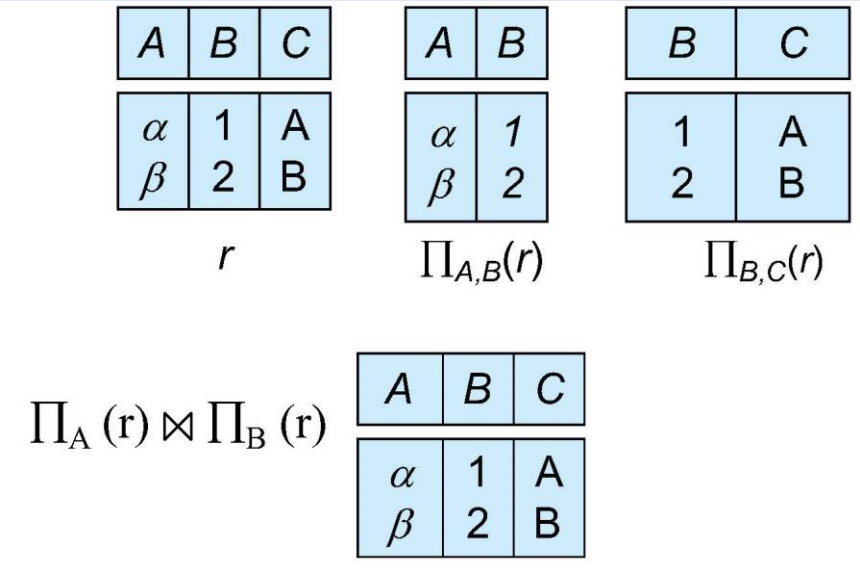
- And, conversely a decomposition is lossy if

$$r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

Example of Lossless Decomposition

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

  $R_1 = (A, B)$ $R_2 = (B, C)$



Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

- **GUIDELINE 4:**

The relations should be designed **to satisfy the lossless join condition**.

No spurious tuples should be generated by doing a natural-join of any relations.

Functional Dependencies

- Functional dependencies (FDs) are used to specify **formal measures of the "goodness" of relational designs**

- FDs and keys are used to define normal forms for relations
- FDs are constraints that are derived from the meaning and interrelationships of the data attributes

- A set of attributes X functionally determines a set of attributes Y
  - if the value of X determines a unique value for Y

Functional Dependencies

- If A is key then,
  - all the other attributes, i.e., B and C can be uniquely identified using A
  - Means,

    A -> BC

A	B	C
1		
2	a	b
3		
4		

Functional Dependencies

- X -> Y holds,
  - if whenever two tuples have the same value for X,
  - they must have the same value for Y

- For any two tuples t1 and t2 in any relation instance r(R):
  - If t1[X]=t2[X],
  - then t1[Y]=t2[Y]

- X -> Y in R specifies a constraint on all relation instances r(R)
- Written as X -> Y; can be displayed graphically on a relation schema as in Figures (denoted by the arrow:)
- FDs are derived from the real-world constraints on the attributes

Examples of FD constraints

- social security number determines employee name
  - SSN -> ENAME
- project number determines project name and location
  - PNUMBER -> {PNAME, PLOCATION}
- employee ssn and project number determines the hours per week that the employee works on the project
  - {SSN, PNUMBER} -> HOURS

Examples of FD constraints

- An FD is a property of the attributes in the schema R
- The constraint must hold on every relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R (since we never have two distinct tuples with t1[K]=t2[K])

Functional Dependencies (FDs)

- If A -> B,
  - then A need not be key always
  - For every value of A, we should be able to uniquely identifying B values

A	B	C
a	1	d
a	1	e
b	2	f
b	2	g

FDs tells which are the areas that can further decompose

FD Example

- Rule out the FD based on the tables

Eid	Ename
1	a
2	b
3	b

Eid -> Ename

Ename -> Eid

FD Example

- Rule out the FD based on the tables

Eid	Ename
1	a
2	b
3	b

Eid -> Ename

~~Ename -> Eid~~

A	B
1	1
1	2
2	2

A -> B

B -> A

FD Example

- Rule out the FD based on the tables

Eid	Ename
1	a
2	b
3	b

Eid -> Ename

~~Ename -> Eid~~

A	B
1	1
1	2
2	2

~~A -> B~~

~~B -> A~~

FD Example

- Rule out the FD based on the tables

A	B	C
1	1	4
1	2	4
2	1	3
2	2	3
2	4	3

A -> B

B -> C

B -> A

C -> B

C-> A

A -> C

FD Example

- Rule out the FD based on the tables

A	B	C
1	1	4
1	2	4
2	1	3
2	2	3
2	4	3

~~A -> B~~

~~B -> C~~

~~B -> A~~

~~C -> B~~

C-> A

A -> C

FD Example

- Rule out the FD based on the tables

X	Y	Z
1	4	3
1	5	3
4	6	3
3	2	3

XZ -> X

XY -> Z

Z -> Y

Y -> Z

XZ -> Y

FD Example

- Rule out the FD based on the tables

X	Y	Z
1	4	3
1	5	3
4	6	3
3	2	3

XZ -> X

XY -> Z

~~Z -> Y~~

Y -> Z

~~XZ -> Y~~

FD Example

A	B	C
1	1	1
1	1	0
2	3	2
2	3	2

- B does not functionally determine C
- A -> B is valid for the particular instance but may not hold for the entire database
- Therefore, A does not functionally determine B

Functional Dependencies (FDs)

- If A -> B,

	A	B
t1		
t2		
	If t1 and t2 agree here	Then they must agree here also
	If t1 and t2 disagree here	Then they may agree or Disagree here

A	B
1	a
1	a
2	a
3	b
1	a
1	a

A	B
1	a
2	a
3	b

Inference Rules for FDs

- Given a set of FDs F, we can infer additional FDs that hold whenever the FDs in F hold
- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset of* X, then X -> Y
  - IR2. (**Augmentation**) If X -> Y, then XZ -> YZ

    (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z

- IR1, IR2, IR3 form a sound and complete set of inference rules

Inference Rules for FDs

Some additional inference rules that are useful:

- (**Decomposition**) If X -> YZ, then X -> Y and X -> Z
- (**Union**) If X -> Y and X -> Z, then X -> YZ
- (**Psuedotransitivity**) If X -> Y and WY -> Z, then WX -> Z

The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Thanks!