Database Theory for Relational Databases:

Functional Dependencies
Decompositions
Normal Forms

CENG315 INFORMATION MANAGEMENT

Features of Good Relational Design

- *The table contains data about the instructors (ID, name, salary) and departments (dept_name, building, budget)*
- There is repetition of information
- There are anomalies
- Need to use null values (if we add a new department with no instructors) insertion anomaly
- If we delete 15151 Mozart we lose the information about Music department unintentionally deletion anomaly
- If we change the building of Physics department in the first record and do not change it in the rest of the file update anomaly

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

inst_dept

Decomposition as a Solution

- *Decomposition*

  Instructors(ID, name, salary, dept_name)

  Departments (dept_name, building, budget)

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

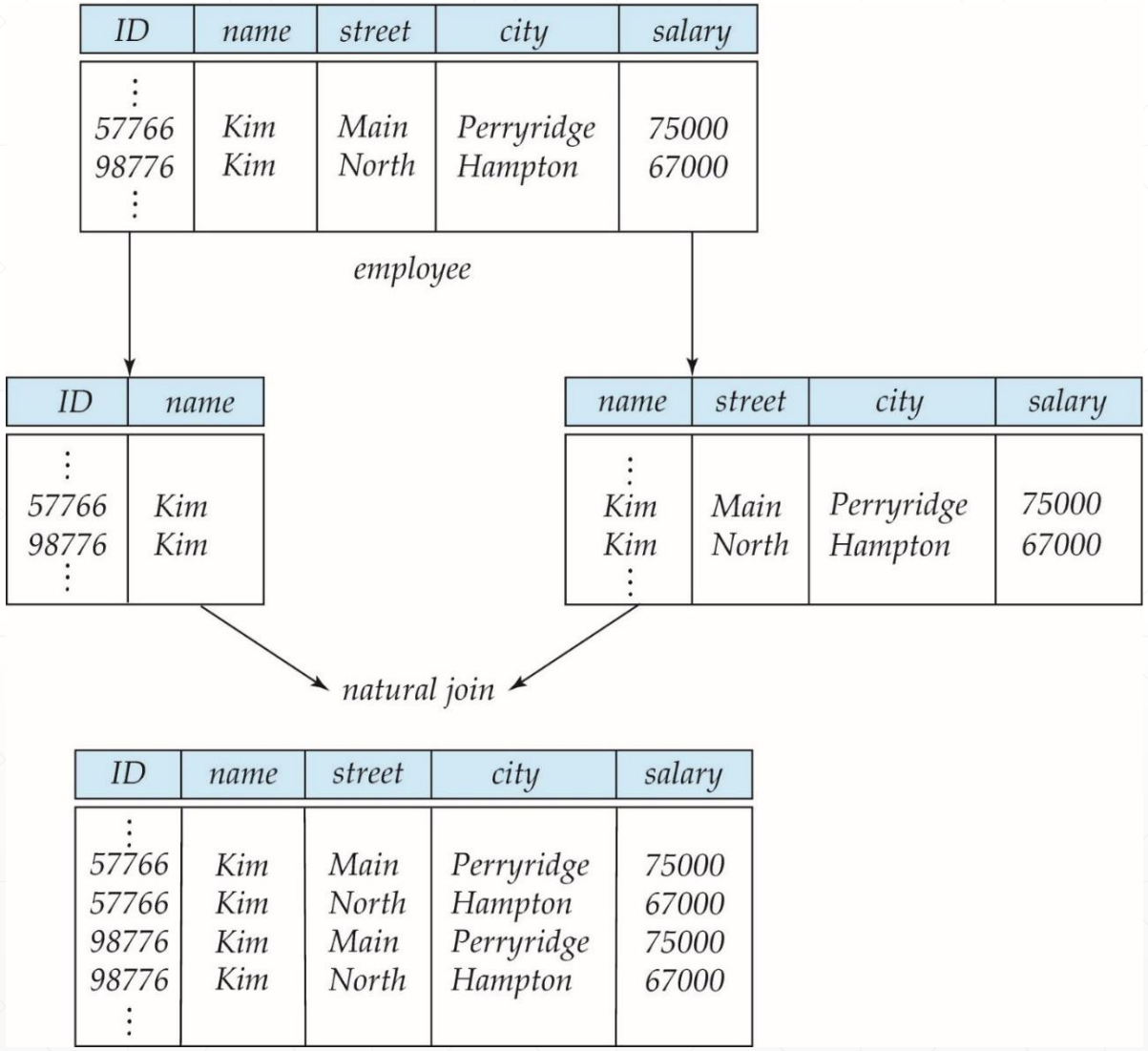
inst_dept

A Lossy Decomposition

- Not all decompositions are good.
- Suppose we decompose

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

  - *employee1* (*ID*, *name*)
  - *employee2* (*name*, *street, city, salary*)
- We cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.



Example of Lossless Decomposition

- Decomposition of $R = (A, B, C)$
  - $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

Relational Design by Decomposition

- “Mega relations” + properties of data
- The system decomposes those on based on the properties that are specified
- The final set of decomposed relations satisfy normal form
  - Normal forms are “good” relations
    - No anomalies
    - No lost information
- The properties themselves are defined either as
  a) Functional dependencies → Boyce-Code Normal Form (BCNF)
  b) Multivalued dependencies → Fourth Normal Form

Functional Dependencies

- Student (SSN, SName, address, HScode, HSname, HScity, GPA, priority)
- Suppose priority is determined by GPA
  - GPA > 3.8 $\rightarrow$ priority = 1
  - 3.3 < GPA $\leq$ 3.8 $\rightarrow$ priority = 2
  - GPA $\leq$ 3.3 $\rightarrow$ priority = 3

Functional Dependencies (Cont.)

- Student (SSN, SName, address, HScode, HSname, HScity, GPA, priority)
- Suppose priority is determined by GPA
  - GPA > 3.8 $\rightarrow$ priority = 1
  - 3.3 < GPA $\leq$ 3.8 $\rightarrow$ priority = 2
  - GPA $\leq$ 3.3 $\rightarrow$ priority = 3
- Two tuples with the same GPA have same priority.

Functional Dependencies (Cont.)

- Student (SSN, SName, address, HScode, HSname, HScity, GPA, priority)
- Suppose priority is determined by GPA
  - GPA > 3.8 $\rightarrow$ priority = 1
  - 3.3 < GPA $\leq$ 3.8 $\rightarrow$ priority = 2
  - GPA $\leq$ 3.3 $\rightarrow$ priority = 3
- Two tuples with the same GPA have same priority.
- $\forall$ t, u $\in$ student: t.GPA = u.GPA $\Rightarrow$ t.priority = u.priority
  - GPA $\rightarrow$ priority

Functional Dependencies (Cont.)

- Two attributes A and B in relation R:
- $\forall$ t, u $\in$ R, t.A = u.A $\Rightarrow$ t.B = u.B
- A $\rightarrow$ B

Functional Dependencies (Cont.)

- Functional dependencies do not always have to have one attribute on each side, they can have a set of attributes.
- $\forall$ t, u $\in$ R, t $[A_1, A_2,\ldots,A_n]$ = u $[A_1, A_2,\ldots,A_n] \rightarrow$ t $[B_1, B_2,\ldots,B_m]$ = u $[B_1, B_2,\ldots,B_m]$
- $A_1, A_2,\ldots,A_n \rightarrow B_1, B_2,\ldots,B_m$

Functional Dependencies (Cont.)

- Based on knowledge of real world
- All instances of the relation must adhere

Functional Dependencies (Cont.)

- $X \rightarrow Y$ is an assertion about a relation R that whenever two tuples of R agree on all the attributes of X, then they must also agree on all attributes in set Y.
  - Say "$X \rightarrow Y$ holds in R."
  - Convention: …, X, Y, Z represent sets of attributes; A, B, C,… represent single attributes.
  - Convention: No set formers in sets of attributes, just ABC, rather than $\{A,B,C\}$.

Rules for Functional Dependencies: Splitting Right Sides of FD's

- $X \rightarrow A_1A_2\ldots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2$,…, $X \rightarrow A_n$ hold for R.
- Example: $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$.
- There is no splitting rule for left sides.
- We'll generally express FD's with singleton right sides.

Example: FD's

Customers (name, addr, drinksLiked, manf, favDrink)

- Reasonable FD's to assert:
  1. name → addr favDrink
     - Note this FD is the same as name → addr and name → favDrink.
  2. drinksLiked → manf

Example: Possible Data

name	addr	drinksLiked	manf	favDrink
Janeway	Voyager	Coke	C.C.	Soda
Janeway	Voyager	IceTea	Lipton	Soda
Spock	Enterprise	Coke	C.C.	Coke

Because name → addr

Because name → favDrink

Because drinksLiked → manf

Rules for Functional Dependencies: Combining Rule

- The inverse of splitting rule.
- Customers (name, addr, drinksLiked, manf, favDrink)
  - name → addr and name → favDrink
    - name → addr favDrink

Rules for Functional Dependencies: Trivial-Dependency Rules

- Nontrivial functional dependency
  - $X \rightarrow Y$ if $Y \not\subseteq X$
- Trivial functional dependency
  - $X \rightarrow Y$ if $Y \subseteq X$
- If $X \rightarrow Y$ then $X \rightarrow X \cup Y$
- If $X \rightarrow Y$ then $X \rightarrow X \cap Y$ (is also implied by the splitting rule)

Rules for Functional Dependencies: Transitive Rule

- Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, surely $A \rightarrow C$ holds, even if we don't say so.

Closure of Attributes

- The set of all functional dependencies logically implied by X is the closure of X, denoted X^+.
- Given relation, a set of FDs, a set of attributes X
- Find Y such that $X \rightarrow Y$ ($Y = \{B_1, B_2, \ldots, B_n\}$)
  - X^+ $\{A_1, A_2, \ldots, A_n\}^+$

The Algorithm to Find Closure of Attributes

- Input: A set of attributes $X = \{A_1, A_2, \ldots, A_n\}$ and a set of FD's of relation R.
- Output: The closure X^+ ($\{A_1, A_2, \ldots, A_n\}^+$)
  - If necessary, split the FD's of R, so each FD in R has a single attribute on the right.
  - Start with the set itself.
  - Repeat until there is no change.
    - If $X \rightarrow Y$ and X is in the set, then add Y to the set.
- We have effectively doing is applying the combining and transitive rules to the attributes to the set until there is no change.

Example: Finding Closure of Attributes

- Given FDs: A → B and B → D
- Closure of A:
  - A^+ = {A, }
  - A^+ = {A, B, }
  - A^+ = {A, B, D}
- Closure of B:
  - B^+ = {B, }
  - B^+ = {B, D}

Keys of Relations

- *K* is a *superkey* for relation R if *K* functionally determines all attributes of R.
- *K* is a *key* for R if *K* is a superkey, but no proper subset of *K* is a superkey.

Example: Superkey

Customers (name, addr, drinksLiked, manf, favDrink)

- name → addr favDrink
- drinksLiked → manf
- {name, drinksLiked} is a superkey because together these attributes determine all the other attributes.

Example: Key

- {name, drinksLiked} is a key because neither {name} nor {drinksLiked} is a superkey.
  - name doesn't → manf; drinksLiked doesn't → addr.
- There are no other keys, but lots of superkeys.
  - Any superset of {name, drinksLiked}.

Closure and Keys

- Is X a superkey for R?
  - Compute X^+, if X^+ = all attributes then X is a superkey.

- How can we find all keys given a set of FDs?
  - Consider every subset of attributes (begin with single attributes)

Review: Functional Dependencies

- Suppose $X \rightarrow$ all attributes
  - X is a superkey
- Trivial functional dependency
  - $X \rightarrow Y$ if $Y \subseteq X$
- Nontrivial functional dependency
  - $X \rightarrow Y$ if $Y \not\subseteq X$

Review: Functional Dependencies (Cont.)

- Splitting rule
  - X → ABC
    - X → A
    - X → B
    - X → C
- Can we also split the left-hand side?
  - NO
  - HSname HScity → HScode
    - ~~HSname → Hscode~~
    - ~~HScity → Hscode~~

Review: Functional Dependencies (Cont.)

- Combining rule
  - $A \rightarrow B, \quad A \rightarrow C, \quad A \rightarrow D$
    - $A \rightarrow BCD$
- Transitive rule
  - $A \rightarrow B, \quad B \rightarrow C$
    - $A \rightarrow C$

Review: Functional Dependencies (Cont.)

- Closure Example
  - Student (SSN, SName, address, HScode, HSname, HScity, GPA, priority)
    - SSN → SName address GPA
    - GPA → priority
    - HScode → HSname HScity
  - $\{SSN, HScode\}^+ = ?$

Review: Functional Dependencies (Cont.)

- Closure Example
  - Student (SSN, SName, address, HScode, HSname, HScity, GPA, priority)
    - SSN → SName address GPA
    - GPA → priority
    - HScode → HSname HScity
  - $\{SSN, HScode\}^+ = \{SSN, HScode, SName, address, GPA, priority, HSname, HScity\}$
    - {SSN, HScode} → all attributes
      - Superkey

Relational Schema Design

- Problems such as redundancy that occur when we try to cram too much into a single relation are called *anomalies*.
- The principal kinds of anomalies that we encounter are:
  - *Redundancy:* Information may be repeated unnecessarily in several tuples.
  - *Update anomalies:* We may change information in one tuple but leave the same information unchanged in another.
  - *Deletion anomalies:* If a set of values becomes empty, we may lose other information as a side effect.
- Goal of relational schema design is to avoid anomalies.

Example of Bad Design

- Customers (name, addr, drinksLiked, manf, favDrink)

name	addr	drinksLiked	manf	favDrink
Janeway	Voyager	Coke	C.C.	Soda
Janeway	???	IceTea	Lipton	???
Spock	Enterprise	Coke	???	Coke

Data is redundant, because each of the ???'s can be figured out by using the FD's name → addr favDrink and drinksLiked → manf.

This Bad Design Also Exhibits Anomalies

- Customers (name, addr, drinksLiked, manf, favDrink)

name	addr	drinksLiked	manf	favDrink
Janeway	Voyager	Coke	C.C.	Soda
Janeway	Voyager	IceTea	Lipton	Soda
Spock	Enterprise	Coke	C.C.	Coke

- Update anomaly: If Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Coke, we lose track of the fact that Coca-Cola manufactures Coke.
- Insertion anomaly: You cannot add new drink without a person information.

Boyce-Codd Normal Form

- The goal of decomposition is to replace a relation by several that do not exhibit anomalies.
- The condition under which the anomalies discussed can be guaranteed not to exist is called *Boyce-Codd Normal Form*, or *BCNF*.
- We say a relation *R* is in *BCNF* if whenever $X \rightarrow Y$ is a nontrivial FD that holds in *R*, *X* is a superkey.
  - Remember: *Nontrivial* means *Y* is not contained in *X*.
  - Remember: A *superkey* is any superset of a key (not necessarily a proper superset).

Example

Customers (name, addr, drinksLiked, manf, favDrink)

FD's: name → addr favBeer, drinksLiked → manf

- Only key is {name, drinksLiked}.
- In each FD, the left side is *not* a superkey.
- Any one of these FD's shows *Customers* is not in BCNF.

Another Example

Customers (name, manf, manfAddr)

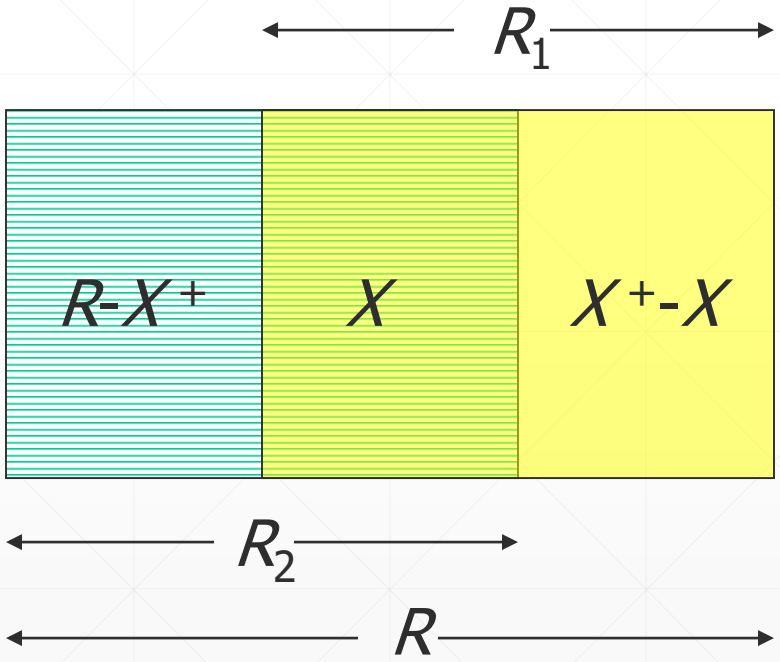
FD's: name → manf, manf → manfAddr

- Only key is {name} .
- name → manf does not violate BCNF, but manf → manfAddr does.

Decompose *R* Using $X \rightarrow Y$

- Replace *R* by relations with schemas:
  1. $R_1 = X^+$.
  2. $R_2 = R - (X^+ - X)$.
- *Project* given FD's *F* onto the two new relations.

Decomposition Picture



Example: BCNF Decomposition

Customers (name, addr, drinksLiked, manf, favDrink)

F = name → addr, name → favDrink, drinksLiked → manf

- Pick BCNF violation name → addr.
- Close the left side: {name}$^+$ = {name, addr, favDrink}.
- Decomposed relations:
  1. Customers1 (name, addr, favDrink)
  2. Customers2 (name, drinksLiked, manf)

Example (Cont.)

- We are not done; we need to check Customers1 and Customers2 for BCNF.
- Projecting FD's is easy here.
- For Customers1(name, addr, favDrink)
  - Relevant FD's are name → addr and name → favDrink.
  - Thus, {name} is the only key and Customers1 is in BCNF.

Example (Cont.)

- For Customers2 (name, drinksLiked, manf), the only FD is drinksLiked → manf, and the only key is {name, drinksLiked}.
  - Violation of BCNF.
- drinksLiked$^+$ = {drinksLiked, manf}, so we decompose *Customers2* into:
  1. Customers3 (drinksLiked, manf)
  2. Customers4 (name, drinksLiked)

Example (Cont.)

- The resulting decomposition of *Customers* :
  1. Customers1 (name, addr, favDrink)
  2. Customers3 (drinksLiked, manf)
  3. Customers4 (name, drinksLiked)
- Notice: *Customers1* tells us about Customers, *Customers3* tells us about drinks, and *Customers4* tells us the relationship between Customers and the drinks they like.

Third Normal Form -- Motivation

- There is one structure of FD's that causes trouble when we decompose.
- $AB \rightarrow C$ and $C \rightarrow B$.
  - Example: A = street address, B = city, C = zip code.
- There are two keys, $\{A,B\}$ and $\{A,C\}$.
- $C \rightarrow B$ is a BCNF violation, so we must decompose into BC, AC.

We Cannot Enforce FD's

- The problem is that if we use *AC* and *BC* as our database schema, we cannot enforce the FD $AB \rightarrow C$ by checking FD's in these decomposed relations.
- Example with *A* = street, *B* = city, and *C* = zip on the next slide

An Unenforceable FD

street (A)	zip (C)
545 Tech Sq.	02138
545 Tech Sq.	02139

city (B)	zip (C)
Cambridge	02138
Cambridge	02139

$AB \rightarrow C$ and $C \rightarrow B$

Join tuples with equal zip codes.

street (A)	city (B)	zip (C)
545 Tech Sq.	Cambridge	02138
545 Tech Sq.	Cambridge	02139

Although no FD's were violated in the decomposed relations, FD street city → zip is violated by the database as a whole.

3NF Let's Us Avoid This Problem

- 3rd Normal Form (3NF) modifies the BCNF condition, so we do not have to decompose in this problem situation.
- An attribute is *prime* if it is a member of any key.
- $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime.

Example: 3NF

- In our problem situation with FD's $AB \rightarrow C$ and $C \rightarrow B$, we have keys AB and AC.
- Thus A, B, and C are each prime.
- Although $C \rightarrow B$ violates BCNF, it does not violate 3NF.

What 3NF and BCNF Give You?

- There are two important properties of a decomposition:
  1. *Lossless Join*: It should be possible to project the original relations onto the decomposed schema, and then reconstruct the original.
  2. *Dependency Preservation*: It should be possible to check in the projected relations whether all the given FD's are satisfied.

What 3NF and BCNF Give You? (Cont.)

- We can get (1) with a BCNF decomposition.
- We can get both (1) and (2) with a 3NF decomposition.
- But we can’t always get (1) and (2) with a BCNF decomposition.
  - street-city-zip is an example.

What 3NF and BCNF Give You? (Cont.)

- There are some situations where BCNF is not dependency preserving
- Solution: Define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems)
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

What 3NF and BCNF Give You? (Cont.)

- Disadvantages to 3NF.
  - We may have to use null values to represent some of the possible meaningful relationships among data items.
  - There is the problem of repetition of information.
- Advantages to 3NF over BCNF: It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation.
- Relations in 3NF may have some redundancy, but that is deemed an acceptable trade-off in cases where there is no dependency-preserving decomposition into BCNF.

References

- Jeffrey D. Ullman and Jennifer Widom, “A First Course in Database Systems”, 3rd Ed., 2007.
  - http://infolab.stanford.edu/~ullman/fcdb/aut07/slides/fds.pdf
- A. Silberschatz, HF. Korth, S. Sudarshan, Database System Concepts, 7th Ed., McGraw-Hill, 2019.
  - Chapter 7, https://www.db-book.com/db7/slides-dir/PPTX-dir/ch7.pptx (modified)
- Stanford DB Class