Discussion 5

Normalization
EECS 484

Logistics

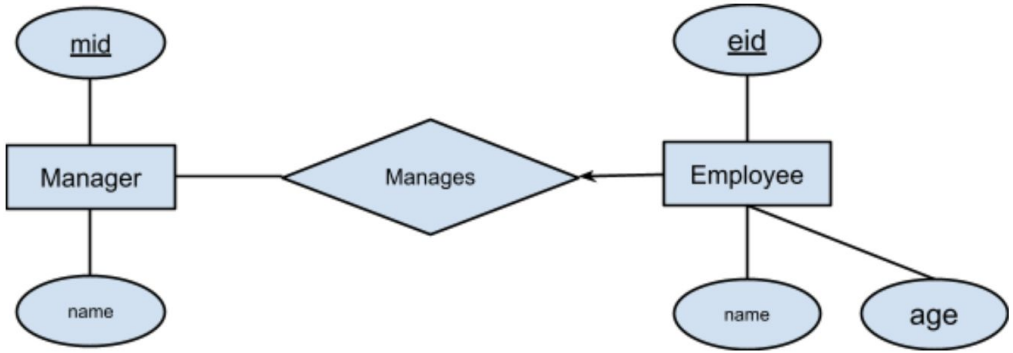
- **Homework 2** due **Today** at 11:45 PM ET
- **Homework 3** released, due **Oct 9th** at 11:45 PM ET
    - **Assign questions to corresponding pages** on Gradescope to indicate where your responses are located
- **Project 2** released, due **Oct 22nd** at 11:45 PM ET

Logistics

- **Midterm: Oct 10th**, 6:15–8:15 PM ET, at **CCCB** on Central Campus
  - Sample Exams available on Canvas
  - Covers everything up to and including Lecture 13: Midterm Review
    - Lectures, discussions, projects, and homework
  - Please bring pens/pencils, an eraser, and your Mcard (or a valid government-issued ID) with you
  - One letter size (8.5x11) cheat sheet is allowed, **handwritten** on **one side**
    - It should have your name on it and you will need to hand it in with your exam
    - It is an honor code violation for you to reproduce someone else's cheat sheet
  - More detailed info and room assignments can be found on piazza @444

Recap of ER to Relational Terminology

- In Approach 1, an entity and its relationship translate to different tables
- In Approach 2, the tables are combined
- They apply when handling arrows in ER diagrams
- A translation may result in use of Approach 1, Approach 2, or both, depending on the key/participation constraints and preference among the two approaches



Normalization

Normalization Theory

- There are a lot of different ways we can represent data in a relation
  - Want to reduce redundancy and maximize integrity
- How do we make good tables
  - Start with a nice ER diagram and make tables
    - Great if starting from scratch
  - Take existing tables and decompose them into smaller tables
    - Will do this using functional dependencies and normal forms

Functional Dependency

- A → B
  - A functionally determines B
  - “If you know A, then you know B”
    - No other options
- F is a set of FDs for a relation
- F+ = closure of F = set of all FDs we can derive from F
- Example
  - F = ID→ Name, ID→Title, Title→Manager
    F+ also includes, by transitivity, ID→Manager
    - Notice, ID allows me to functionally determine all columns

ID	Name	Title	Manager
1	John	Clerk	Alice
2	Jane	Clerk	Alice
3	Alice	Manager	Bob
4	Bob	Owner	Bob

Armstrong’s Axioms

- Rules of inference for functional dependencies
  - Repeatedly applying them allows us to generate F+
- Reflexivity
  - If $y \subseteq x$ then $x \rightarrow y$
  - Trivial self dependence
- Augmentation
  - If $x \rightarrow y$ then $xz \rightarrow yz$ for any z
- Transitivity
  - If $x \rightarrow y$ and $y \rightarrow z$ then $x \rightarrow z$
- Derived axioms
  - Union: If $x \rightarrow y$ and $x \rightarrow z$ then $x \rightarrow yz$
  - Decomposition: if $x \rightarrow yz$ then $x \rightarrow y$ and $x \rightarrow z$

Relation Decomposition

- Let’s split up larger relations into smaller relations
  - More relations means that data is more spread out
    - Will need joins to connect them back
    - **Lossless join** = can reconstruct data of original relation from decomposed relations
    - **Dependency preserving** = do we still have all the same dependencies
    - Lossless join is required while dependency preserving is nice to have
  - Notationally:
    - Decompose X into Y and Z
    - Lossless join: $(Y \cap Z \rightarrow Y)$ or $(Y \cap Z \rightarrow Z)$
      - Attributes common to Y and Z contains a key for Y or Z
    - Dependency preserving: $F+ = (F_Y \cup F_Z)+$
      - F_Y is the set of FDs in F+ that only involve attributes in Y (same for F_Z)
      - Original dependencies are preserved in the decomposed relations
- We’ll use normal forms to help us decompose

Third Normal Form (3NF)

- Builds off First and Second Normal Forms
  - 1NF = Each column contains a single value and each row must be unique
  - 2NF = 1NF + Attributes not a part of the primary key are fully dependent on primary key
  - 3NF = 2NF + No transitive functional dependencies
- Relation R is in 3NF form if and only if for all dependencies of the form $X \rightarrow A$ in F+ **(where X is a subset of attributes of R and A is a single attribute of R)**, the following hold true:
  - $A \subseteq X$ (trivial dependence) or
  - X is a super key or
  - A is some part of a minimal key in R
- Superkey is a set of attributes that can unique identify a row in a table
  - Can be larger than candidate key (non-minimal)
- Can always decompose R into a collection of 3NF relations that satisfies the lossless join and dependency preserving properties

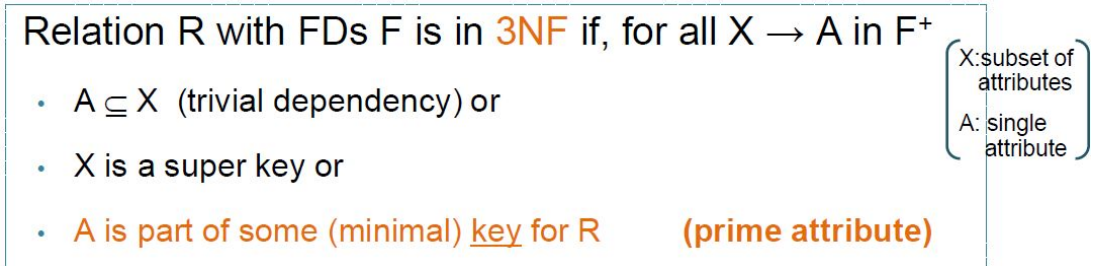
Boyce-Codd Normal Form (BCNF)

- Relation R is in BCNF form if and only if for all dependencies of the form $X \rightarrow A$ in F+ (where X is a subset of attributes of R and A is a single attribute of R)
  - $A \subseteq X$ (trivial dependence) or
  - X is a super key
- Missing the “part of a minimal key” category from 3NF
- Stronger than 3NF
  - If relation is BCNF then it is 3NF
    - Not the other way around though
- To decompose R into BCNF, start with unnormalized relation
  - If $X \rightarrow Y$ violates BCNF decompose into R-Y and XY
    - Repeat for all $X \rightarrow Y$
  - Satisfies lossless join property, but not always dependency preserving

Examples

Question 1

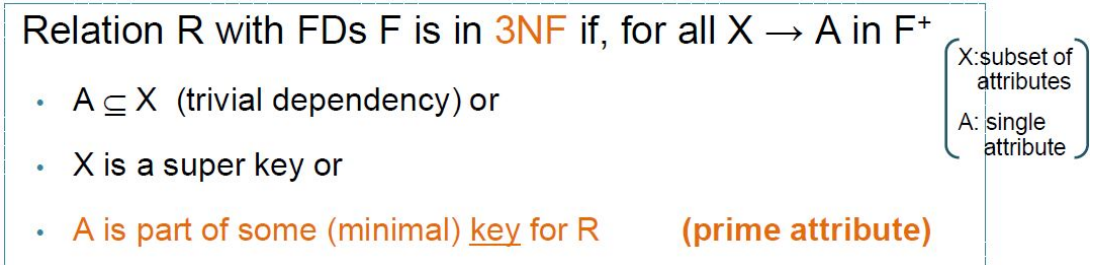
- Consider a relation R=(A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - BE→D
  - BE→AC
  - C→E



- What are candidate keys here?
  - Remember a candidate key is a **minimal** set of columns that allow us to **uniquely** define the relation
- Is R 3NF?
  - If so, is R BCNF?

Question 1

- Consider a relation R=(A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - BE→D
  - BE→AC
  - C→E



- What are candidate keys here?
  - Remember a candidate key is a minimal set of columns that allow us to uniquely define the relation Keys founds = {A}

    Repetitively union A→B, A→C and A→D to get A→BCD

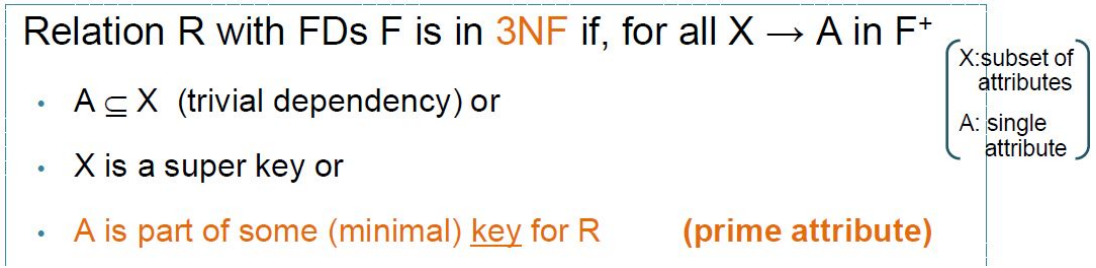
    Augment A on both sides to get A→ABCD

    Use transitivity on A→C and C→E to get A→E

    Union A→ABCD and A→E to get A→ABCDE

Question 1

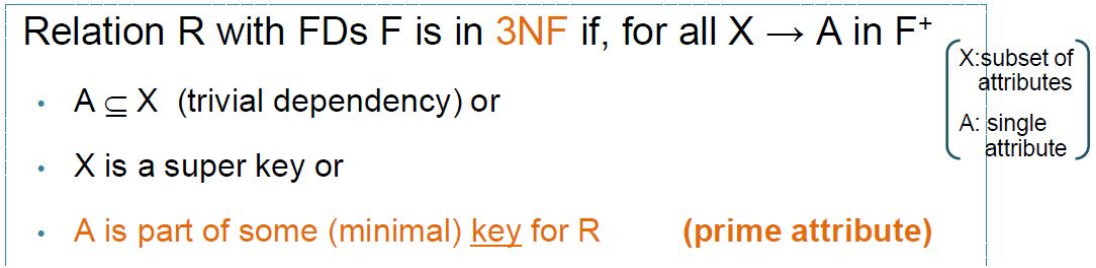
- Consider a relation R=(A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - BE→D
  - BE→AC
  - C→E



- What are candidate keys here?
  - Remember a candidate key is a minimal set of columns that allow us to uniquely define the relation Keys founds = {A, BE}
    Union BE→D and BE→AC to get BE→ACD
    Augment BE on both sides to get BE→ABCDE

Question 1

- Consider a relation R=(A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - BE→D
  - BE→AC
  - C→E



- What are candidate keys here?
  - Remember a candidate key is a minimal set of columns that allow us to uniquely define the relation Keys founds = {A, BE, BC}

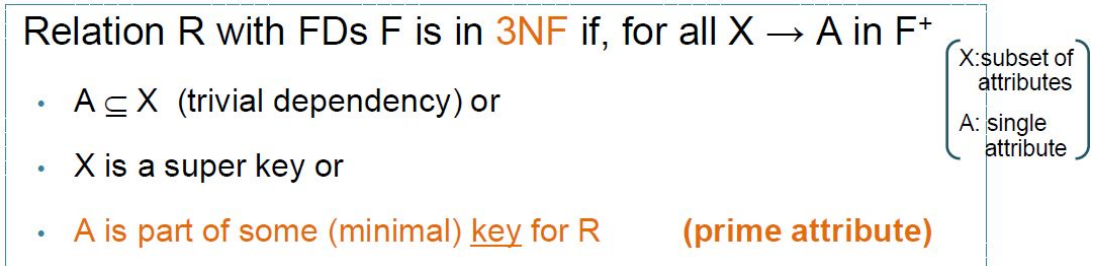
    Augment B on both sides of C→E to get BC→BE

    Since we already know BE is a key

    Use transitivity on BC→BE and BE→ABCDE to get BC→ABCDE

Question 1

- Consider a relation R=(A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - C→E
  - BE→D
  - BE→AC



- FDs in F^+ = {A→B, A→C, A→D, C→E, BE→D, BE→A, BE→C, ……}
- Candidate keys = {A, BC, BE}
- Is R 3NF?
  - Yes

Question 1

- Consider a relation R=(A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - C→E
  - BE→D
  - BE→AC

Rel. R with FDs F is in BCNF if, for all X → A in F^+

- $A \subseteq X$ (trivial FD), or
- X is a super key

X:subset of attributes

A: single attribute

- FDs in F^+ = {A→B, A→C, A→D, C→E, BE→D, BE→A, BE→C, ……}
- Candidate keys = {A, BC, BE}
- Is R BCNF?
  - No, C→E violates both constraints

Question 2

- Relation R = (A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - C→E
  - BE→D
  - BE→AC
- Decompose into BCNF

High-Level Algorithm

Input: a relation R with FDs F

1. Identify if any FDs violate BCNF (How?)
   - If X → Y violates BCNF, decompose R into **R - Y and XY**
2. Repeat for every X → Y that violates BCNF.

Output: a collection of relations that are in BCNF

Question 2

- Relation R = (A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - C→E
  - BE→D
  - BE→AC
- Decompose into BCNF
  - We know C→E violates constraint
  - R1={A,B,C,D} R2={C,E}
  - Is this lossless?
    - Yes, R1∩R2=C which is a key for R2 (C→E)

High-Level Algorithm

Input: a relation R with FDs F

1. Identify if any FDs violate BCNF (How?)
   - If $X \rightarrow Y$ violates BCNF, decompose R into **R - Y and XY**
2. Repeat for every $X \rightarrow Y$ that violates BCNF.

Output: a collection of relations that are in BCNF

Relation R, FDs F; Decomposed to X, Y

- Test: lossless-join w.r.t. F if and only if F^+ contains:

$$X \cap Y \rightarrow X, \quad \text{or} \quad X \cap Y \rightarrow Y$$

i.e. attributes common to X and Y contain a key for either X or Y

Question 2

- Relation R = (A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - C→E
  - BE→D
  - BE→AC
- Decompose into BCNF
  - R1={A,B,C,D} R2={C,E}
  - Is this dependency preserving?
    - F1 contains A→B, A→C, A→D, F2 contains C→E (F1+ = F+ that is restricted R1)
    - (F1 ∪ F2) contains A→B, A→C, A→D, C→E
    - (F1 ∪ F2)+ contains A→B, A→C, A→D, C→E, A→E
    - Not equal to original F+, so not dependency preserving
  - Corrected Solution: (Derived FDs that need to be added: BC → D; BC → A)
    - F1 contains A→B, A→C, A→D, BC → D, BC → A; F2 contains C→E
    - (F1 ∪ F2) contains A→B, A→C, A→D, BC → D, BC → A, C→E
    - (F1 ∪ F2)+ still does not contain BE → D or BE → AC
    - Therefore, it is not dependency preserving!

Informally: We don't want the original FDs to span two tables.

R has a dependency-preserving decomposition to X, Y

if $F^+ = (F_x \cup F_y)^+$

Question 2

- Relation R = (A,B,C,D,E)
  - Dependencies as follows
  - A→B
  - A→C
  - A→D
  - C→E
  - BE→D
  - BE→AC

Rel. R with FDs F is in BCNF if, for all X → A in F^+

- $A \subseteq X$ (trivial FD), or
- X is a super key

(X:subset of attributes; A: single attribute)

- Decompose into BCNF
  - R1={A,B,C,D} R2={C,E}
  - R1 and R2 are both in BCNF form
    - You can check this
    - Note: No way of testing if collection of relations are in BCNF, only one relation at a time
  - Since R1 and R2 don't preserve dependencies, we could leave R as a 3NF relation

HW2 University Query 1:

Consider a database consisting of the following five tables. Attributes that are underlined have been designated the primary key of their respective tables, and fields with identical names in different tables can be safely assumed to be foreign keys. The Project Name and Course Name fields are specified as UNIQUE, and all fields are required *except* for Major, which may be NULL (undeclared).

Students(SID, Name, Major)

Projects(PID, P_Name)

Courses(CID, C_Name)

Members(PID, SID)

Enrollments(CID, SID)

1. Write a query that returns CID of CS-heavy courses, which is defined as the following: strictly fewer than 10 non-CS majors are enrolled (including courses in which 0 non-CS majors are enrolled). A student whose major is CS will have the VARCHAR2 value 'CS' for their Major field, while a non-CS student will have something else. Remember that the Major field *can* be NULL. The results should be sorted in **descending** order by CID.

Good luck on the exam!