Introduction

Main question

- How do we design “good” tables for a relational database?
  - Typically we start with ER and convert it into tables
  - Still, different people come up with different ER, and thus different tables. Which one is better? What design should we choose?
- Relational design theory
  - A theory on how to identify and create a good table design or a “normal form”
  - Several definitions of “normal forms” exist
  - We learn the most popular normal form, Boyce-Codd Normal Form (BCNF)

Warning

- The most difficult and theoretical part of the course. Pay attention!

Motivation & Intuition

(StudentClass(sid, name, addr, dept, cnum, title, unit) slide)

- Q: Is it a good table design?

- REDUNDANCY: The same information mentioned multiple times. Redundancy leads to potential anomaly.
  1. UPDATE ANOMALY: Only some information may be updated
2. INSERTION ANOMALY: Some information cannot be represented
   – Q: What if a student does not take any class?

3. DELETION ANOMALY: Deletion of some information may delete others
   – Q: What if the only class that a student takes is cancelled?

• Q: Is there a better design? What tables would you use?

• Q: Any way to arrive at such table design more systematically?
  – Q: Where is the redundancy from?
    (Slide on “guessing” missing info)

– FUNCTIONAL DEPENDENCY: Some attributes are “determined” by other attrs
  * e.g., sid → (name, addr), (dept, cnum) → (title, unit)
  * When there is a functional dependency, we may have redundancy.
    - e.g., (301, James, 11 West) is stored redundantly. So is (CS, 143, database, 04).

– DECOMPOSITION: When there is a FD, no need to store multiple instances of this relationship. Store it once in a separate table
  * (Intuitive normalization of StudentClass table)
    StudentClass(sid, name, addr, dept, cnum, title, unit)
    FDs: sid→(name, addr), (dept, cnum)→(title, unit)
    1. sid → (name, addr): no need to store it multiple time. separate it out
2. \((\text{dept, cnum}) \rightarrow (\text{title, unit})\). separate it out

- Basic idea of table “normalization”
  - Whenever there is a FD, the table may be “bad” (not in normal form)
  - We use FDs to “split” or “decompose” table and remove redundancy
  - We learn FUNCTIONAL DEPENDENCY and DECOMPOSITION to formalize this.

### Functional Dependency

#### Overview

- The fundamental tool for normalization theory
- May seem dry and irrelevant, but bear with me. Extremely useful
- Things to learn
  - FD, trivial FD, logical implication, closure, FD and key, projected FD

#### Functional dependency \(X \rightarrow Y\)

- Notation: \(u[X]\) - values for the attributes \(X\) of tuple \(u\)
  e.g., Assuming \(u = (\text{sid: 100, name: James, addr: Wilshire})\), \(u[\text{sid, name}] = (100, \text{James})\)
- FUNCTIONAL DEPENDENCY \(X \rightarrow Y\)
  - For any \(u_1, u_2 \in R\), if \(u_1[X] = u_2[X]\), then \(u_1[Y] = u_2[Y]\)
  - More informally, \(X \rightarrow Y\) means that “no two tuples in \(R\) can have the same \(X\) values but different \(Y\) values”

\(\langle\text{e.g., StudentClass(sid, name, addr, dept, cnum, title, unit)}\rangle\)
  - \(Q:\) sid \(\rightarrow\) name?

  - \(Q:\) dept, cnum \(\rightarrow\) title, unit?

  - \(Q:\) dept, cnum \(\rightarrow\) sid?
- Whether a FD is true or not depends on real-world semantics

\[\begin{array}{ccc}
A & B & C \\
\hline
a_1 & b_1 & c_1 \\
a_1 & b_2 & c_2 \\
a_2 & b_1 & c_3 \\
\end{array}\]

Q: AB \rightarrow C. Is this okay?

Replace \(c_3\) to \(c_1\).

\[\begin{array}{ccc}
A & B & C \\
\hline
a_1 & b_1 & c_1 \\
a_1 & b_2 & c_2 \\
a_2 & b_1 & c_1 \\
\end{array}\]

Q: AB \rightarrow C. Is this okay?

NOTE: AB \rightarrow C does not mean no duplicate C values.

Replace \(b_2\) to \(b_1\)

\[\begin{array}{ccc}
A & B & C \\
\hline
a_1 & b_1 & c_1 \\
a_1 & b_1 & c_2 \\
a_2 & b_1 & c_3 \\
\end{array}\]

Q: AB \rightarrow C. Is this okay?

- **TRIVIAL** functional dependency: \(X \rightarrow Y\) when \(Y \subseteq X\)
  - It is always true regardless of real world semantics
    (diagram)

- **NON-TRIVIAL** FD: \(X \rightarrow Y\) when \(Y \not\subseteq X\)
  (diagram)

- **COMPLETELY NON-TRIVIAL** FD: \(X \rightarrow Y\) with no overlap between \(X\) and \(Y\)
  (diagram)

We will focus on completely non-trivial functional dependency.

**Implication and Closure**

- **LOGICAL IMPLICATION**
ex) \( R(A, B, C, G, H, I) \)
\( F: A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \) (set of functional dependencies)

- Q: Is \( A \rightarrow H \) true under \( F \)?

\( F \) LOGICALLY IMPLIES \( A \rightarrow H \)

(\langle canonical database method to prove \( A \rightarrow H \rangle \)

\begin{array}{c|c|c|c|c|c|c}
A & B & C & G & H & I \\
\hline
a_1 & b_1 & c_1 & g_1 & h_1 & i_1 \\
\hline
\end{array}

If ? = h_1, then \( A \rightarrow H \)

* Q: \( AG \rightarrow I \)?

- CLOSURE OF FD \( F \): \( F^+ \)

\( F^+ \): the set of all FD’s that are logically implied by \( F \).

- CLOSURE OF ATTRIBUTE SET \( X \): \( X^+ \)

\( X^+ \): the set of all attrs that are functionally determined by \( X \)

- Q: What attribute values do we know given (sid, dept, cnum)?

- CLOSURE \( X^+ \) COMPUTATION ALGORITHM

(\langle \( X^+ \) computation algorithm slide \rangle)

Start with \( X^+ = X \)
Repeat until no change in \( X^+ \)
  If there is \( Y \rightarrow Z \) and \( Y \subset X^+ \), add \( Z \) to \( X^+ \)

(\langle example \rangle)

\( R(A, B, C, G, H, I) \) and \( A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \)
- Q: \{A\}⁺?

- Q: \{A, G\}⁺?

- **FUNCTIONAL DEPENDENCY AND KEY**

  - Key determines a tuple and functional dependency determines other attributes. Any formal relationship?
  - Q: In previous example, is \( (A, B) \) a key of \( R \)?
    \[ R(A, B, C, G, H, I) \text{ and } A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \]

  - \( X \) is a KEY of \( R \) if and only if
    1. \( X \rightarrow \) all attributes of \( R \) (i.e., \( X⁺ = R \))
    2. No subset of \( X \) satisfies 1 (i.e., \( X \) is minimal)

- **PROJECTING FD**

  \( R(A, B, C, D) : A \rightarrow B, B \rightarrow A, A \rightarrow C \)

  - Q: What FDs hold for \( R'(B, C, D) \) which is a projection of \( R \)?

  - In order to find FD’s after projection, we first need to compute \( F⁺ \) and pick the FDs from \( F⁺ \) with only the attributes in the projection.

**Decomposition**

- (Remind the decomposition idea of StudentClass table)

- Splitting table \( R(A₁, \ldots, Aₙ) \) into two tables, \( R₁(A₁, \ldots, Aᵳ) \) and \( R₂(Aⱼ, \ldots, Aₙ) \)

  - \( \{A₁, \ldots, Aᵳ\} = \{A₁, \ldots, Aᵳ\} \cup \{Aⱼ, \ldots, Aₙ\} \)
  - (Conceptual diagram for \( R(X, Y, Z) \rightarrow R₁(X, Y) \) and \( R₂(Y, Z) \))
• Q: When we decompose, what should we watch out for?

LOSSLESS-JOIN DECOMPOSITION

• $R = R_1 \bowtie R_2$

• Intuitively, we should not lose any information by decomposing $R$

• Can reconstruct the original table from the decomposed tables

• Q: When is decomposition lossless?

\[
\begin{array}{|c|c|c|}
\hline
\text{cnum} & \text{sid} & \text{name} \\
\hline
143 & 1 & \text{James} \\
143 & 2 & \text{Elaine} \\
325 & 3 & \text{Susan} \\
\hline
\end{array}
\]

– Q: Decompose into $S_1(\text{cnum}, \text{sid})$, $S_2(\text{cnum}, \text{name})$. Lossless?

– Q: Decompose into $S_1(\text{cnum}, \text{sid})$, $S_2(\text{sid}, \text{name})$. Lossless?

• DECOMPOSITION $R(X, Y, Z) \Rightarrow R_1(X, Y), R_2(X, Z)$ IS LOSSLESS IF $X \rightarrow Y$ OR $X \rightarrow Z$

– That is, the shared attributes are the key of one of the decomposed tables

– We can use FDs to check whether a decomposition is lossless

\textbf{Example:} StudentClass(sid, name, addr, dept, cnum, title, unit)
sid → (name, addr), (dept, cnum) → (title, unit)

* Q: Decomposition into $R_1$(sid, name, addr), $R_2$(sid, dept, cnum, title, unit). Lossless?

**Boyce-Codd Normal Form (BCNF)**

FD, key & redundancy

- **Example:** StudentClass(sid, name, addr, dept, cnum, title, unit)
  - Q: sid → (name, addr). Does it cause redundancy?

  - After decomposition, Student(sid, name, addr)
  - Q: sid → (name, addr). Does it still cause redundancy?

  * Q: Why does the same FD cause redundancy in one case, but not in the other?

- In general, FD $X \rightarrow Y$ leads to redundancy if $X$ DOES NOT CONTAIN A KEY.

**BCNF definition**

- $R$ is in BCNF with regard to $F$, iff for every non-trivial $X \rightarrow Y$, $X$ contains a key
- “Good” table design (no redundancy due to FD)
- Q: Class(dept, cnum, title, unit). dept, cnum→title, unit.
  - Q: Intuitively, is it a good table design? Any redundancy? Any better design?

  - Q: Is it in BCNF?

- Q: Employee(name, dept, manager). name→dept, dept→manager.
– Q: What is the English interpretation of the two dependencies?

– Q: Intuitively, is it a good table design? Any redundancy? Better design?

– Q: Is it in BCNF?

• Remarks: Most times, BCNF tells us when a design is “bad” (due to redundancy from functional dependency.

BCNF normalization algorithm

• Decomposing tables until all tables are in BCNF
  – For each FD $X \rightarrow Y$ that violates the condition, separate those attributes into another table to remove redundancy.
  – We also have to make sure that this decomposition is lossless.

• Algorithm
  For any R in the schema
  If non-trivial $X \rightarrow Y$ holds on R, and if X does not have a key
  1. Compute $X^+$ ($X^+$: closure of X)
  2. Decompose R into $R_1(X^+)$ and $R_2(X, Z)$ // X is common attributes where Z is all attributes in R except $X^+$

Repeat until no more decomposition

• Example: ClassInstructor(dept, cnum, title, unit, instructor, office, fax)
  instructor $\rightarrow$ office, office $\rightarrow$ fax
  (dept, cnum) $\rightarrow$ (title, unit), (dept, cnum) $\rightarrow$ instructor.

– Q: What is the English interpretation of the two dependencies?

– Q: Intuitively, is it a good table design? Any redundancy? Better design?

– Q: Is it in BCNF?
– Q: Normalize it into BCNF using the algorithm.

NOTE: The algorithm guarantees lossless join decomposition, because after the decomposition based on \( X \to Y \), \( X \) becomes the key of one of the decomposed table.

- **Example:** \( R(A, B, C, G, H, I) \), \( A \to B, A \to C, G \to I, B \to H \). Convert to BCNF.

- Q: Does the algorithm lead to a unique set of relations?

\( \langle \text{e.g., } R(A, B, C), A \to C, B \to C \rangle \)

Q: What if we start with \( A \to C \)?

Q: What if we start with \( B \to C \)?

- Q: \( R_1(A, B), R_2(B, C, D) \) with \( A \to B, B \to A, A \to C \). Are \( R_1 \) and \( R_2 \) in BCNF?

NOTE: We have to check all implied FD’s for BCNF, not just the given ones.

**Good Table Design in Practice**

- Normalization splits tables to reduce redundancy.
- However, splitting tables has negative performance implication

**Example:** Instructor: name, office, phone, fax
name \( \to \) office, office \( \to \) (phone, fax)

(design 1) Instructor(name, office, phone, fax)
(design 2) Instructor(name, office), Office(office, phone, fax)
Q: Retrieve (name, office, phone) from Instructor. Which design is better?

- As a rule of thumb, start with normalized tables and merge them if performance is not good enough

Things to Remember

- Functional dependency $X \rightarrow Y$
  - Trivial functional dependency
  - Logical implication
  - Closure
- Decomposition
  - Lossless join decomposition
- Boyce-Codd Normal Form (BCNF)
- BCNF decomposition algorithm