CSE 180, Database Systems I
Shel Finkelstein, UC Santa Cruz
Lecture 9

Chapter 7: Relational DB Design Theory (Functional Dependencies, etc.)
Sections 7.1 - 7.5, 7.8
Part 1

Database System Concepts, 7th Ed.

Important Notices

- Lab3 solution has been posted on Piazza under Resources→Lab3.
  - Deadline for questions about Lab2 grade is **Friday, Feb 28**.
- Lab4 assignment has been posted on Piazza under Resources→Lab4. See Piazza announcement about Lab4. Please read it soon!
  - Lab4 is due **Sunday, March 8**, by 11:59pm (2 week assignment).
  - Your solution should be submitted via Canvas as a zip file.
    - Late Lab Assignments will not be accepted.
    - Be sure that you submit the correct file!
  - Lab4 deals with material from this Lecture, Lecture 8.
    - Additional Piazza announcements describe some differences between PostgreSQL Stored Functions and the PSM standard.
  - Lab4 will be discussed at Lab Sections.
  - Load Data for Lab4 has now been posted; you'll need to use that Load Data to complete Lab4.

Important Notices

- Midterm been discussed in class and returned in class.
  - Deadline for questions about Midterm grade is this **Wednesday, March 4**.
  - Please talk to to your TA first, and then to me if you disagree with grade.
- No Office Hour for me on Wed, Feb 26, 3:00-4:00pm; instead, my Office Hour this week will be Mon, Feb 24, 3:00-4:00pm in E2-249B.
  - Please come to my Office Hours to pick up Midterm if you haven't already picked it up
- Gradiance #4 is due **Thursday, March 5** by 11:59pm.
- Attend Lectures, Lab Sections, Office Hours and LSS Tutoring.
  - See Piazza notices about LSS Tutoring with Jeshwanth Bheemanpally.

Database Schema Design

- So far, we have learned database query languages:
  - SQL, Relational Algebra
- How can you tell whether a given database schema is “good” or “bad”?
- Design theory:
  - A set of design principles that allows one to decide what constitutes a “good” or “bad” database schema design.
  - A set of algorithms for modifying a “bad” design to a “better” one.

Example

- If we know that rank determines the salary scale, which is a better design? Why?

- Employees(eid, name, addr, rank, salary_scale)

OR

- Employees2(eid, name, addr, rank)

  Salary_Table(rank, salary_scale)

Lots of Duplicate Information

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Lots of duplicate information
  - Employees who have the same rank have the same salary scale.

Update Anomaly

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Update anomaly
  - If one copy of salary scale is changed, then all copies of that salary scale (of the same rank) have to be changed.

Insertion Anomaly

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Insertion anomaly
  - How can we store a new rank and salary scale information if currently, no employee has that rank?
  - Use NULLS?

Deletion Anomaly

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Deletion anomaly
  - If Hugh is deleted, how can we retain the rank and salary scale information?
  - Is using NULL a good choice?
    - (Why not?)

So What Would Be a Good Schema Design for this Example?

- salary_scale is dependent only on rank
  - Hence associating employee information such as name, addr with salary_scale causes redundancy.
- Based on the constraints given, we would like to refine the schema so that such redundancies **cannot** occur.
- Note however, that sometimes database designers **may choose** to live with redundancy in order to improve query performance.
  - Ultimately, a good design is depends on the query workload.
  - But understanding anomalies and how to deal with them is still important.

Functional Dependencies

- The information that rank determines salary_scale is a type of integrity constraint known as a *Functional Dependency (FD).*

- Functional Dependencies can help us detect anomalies that may exist in a given schema.

- The FD "rank → salary_scale" suggests that

  Employees(eid, name, addr, rank, salary_scale)

  should be *decomposed* into two relations:

  Employees2(eid, name, addr, rank)

  Salary_Table(rank, salary_scale).

Meaning of an FD

- We have seen a kind of Functional Dependency before.
- Keys:
  - Emp(ssn, name, addr)
  - If two tuples agree on the ssn value, then they must also agree on the name and address values. (ssn → name, addr).

- Let r(R) be a relation schema. A *Functional Dependency (FD)* is an integrity constraint of the form:

  $X \rightarrow Y$ (read as "X *determines Y* or X *functionally determines Y*")

  where X and Y are non-empty subsets of attributes R.
- A relation instance of r(R) *satisfies* the FD X → Y if

  for every pair of tuples t and t' in r, if t[X] = t'[X], then t[*Y*] = t'[*Y*]

Denotes the X value(s) of tuple t, i.e., project t on the attributes in X.

Illustration of the Semantics of an FD

- Relation schema r(R) with the FD $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ where $\{ A_1, \ldots, A_m, B_1, \ldots, B_n \} \subseteq R$

	$A_1\ A_2\ \ldots\ A_m$	$B_1\ \ldots\ B_n$	the rest of the attributes in R, if any	
t	xxxxxxxxxxxxxxx	yyyyyyyyyy	zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz	
t'	xxxxxxxxxxxxxxx	yyyyyyyyyy	wwwwwwwwwwwwwwwwwwwwwww	
	vvvvvvvvvvvvvv	yyyyyyyyyy	uuuuuuuuuuuuuuuuuuuuuuuuuuuuu	OK
	xxxxxxxxxxxxxxx	vvvvvvvvvv	uuuuuuuuuuuuuuuuuuuuuuuuuuuuu	
				VIOLATION!

The actual values do not matter, but they cannot be the same if instance is a set.

More on Meaning of an FD

- Relation r(R) satisfies $X \rightarrow Y$
  - Pick any two tuples t and t' of an instance of r. If t and t' agree on the X attributes, then they must also agree on the *Y* attributes.
  - The above must hold for *every possible instance* of r.

- An FD is a statement about *all possible legal instances* of a schema. We cannot just look at an instance (or even at a set of instances) to determine which FDs hold.
  - Looking at an instance may enable us to determine that some FDs are not satisfied.

Functional-Dependency Theory Roadmap

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We'll define Normal Forms that may help us avoid Redundancy and Anomalies.
- Then we'll define Decompositions of relations into multiple relations (similar to our Employee/Rank example), and we'll explain what it means for a decomposition to be "Lossless".
- Silberschatz textbook includes algorithms to decompose relations losslessly into "Normal Forms".
  - We'll discuss one simple approach.
- But we will briefly mention what it means for a decomposition to be Dependency-Preserving.
- And we'll end by explaining that 2 out of 3 ain't bad.
  - We'd like 3 nice properties, but we might only be able to get 2.
  - Anomaly avoidance, Lossless Decomposition, Dependency-Preservation

Reasoning about FDs

R(A,B,C,D,E)

Suppose A → C and C → E. Is it also true that A → E ?

In other words, suppose that an instance r satisfies A → C and C → E. Must r also satisfy A → E ?

YES

Proof: ?

Implication of FDs

- We say that a set $\mathcal{F}$ of FDs *implies* an FD F if for every instance r that satisfies $\mathcal{F}$, it must also be true that r satisfies F.
- Notation: $\mathcal{F} \models F$
- Note that just finding some instance(s) of r that both satisfy $\mathcal{F}$ and also satisfy F is not sufficient to prove that $\mathcal{F} \models F$.
- How can we determine whether or not $\mathcal{F}$ implies F?

Armstrong's Axioms

- Use Armstrong's Axioms to determine whether or not $\mathcal{F} \models F$.

- Let X, Y and Z denote sets of attributes over a relation schema r(R).

  - **Reflexivity**: If $Y \subseteq X$, then $X \rightarrow Y$

    Example: ssn, name $\rightarrow$ name

    - FDs in this category are called trivial FDs.
  - **Augmentation**: If $X \rightarrow Y$, then for any attributes Z, $XZ \rightarrow YZ$.

    Example: ssn, name, addr $\rightarrow$ name addr
  - **Transitivity**: If $X \rightarrow Y$, and $Y \rightarrow Z$, then $X \rightarrow Z$.

    Example: If ssn $\rightarrow$ rank, and rank $\rightarrow$ sal_scale, then ssn $\rightarrow$ sal_scale.

Union and Decomposition Rules

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

- Union and Decomposition rules are not essential. In other words, they can be derived using Armstrong's axioms.

- Derivation of the Union rule:

  (to fill in)

Union and Decomposition Rules

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

- Union and Decomposition rules are not essential. In other words, they can be derived using Armstrong's axioms.

- Derivation of the Union rule:

  Since $X \rightarrow Z$, we get $XY \rightarrow YZ$ (augmentation)

  Since $X \rightarrow Y$, we get $X \rightarrow XY$ (augmentation)

  Therefore, $X \rightarrow YZ$ (transitivity)

Additional Rules

- Derivation of the Decomposition rule:
  (to fill in)

Additional Rules

- Derivation of the Decomposition rule:

  X → YZ (given)

  YZ → Y (reflexivity)

  YZ → Z (reflexivity)

  Therefore, X → Y and X → Z (transitivity).

- We use the notation $\mathcal{F} \vdash$ F to mean that *F can be derived from* $\mathcal{F}$ *using Armstrong's axioms*.
  - What was the meaning of $\mathcal{F} \models$ F ($\mathcal{F}$ implies F)?

Pseudo-Transitivity Rule

- **Pseudo-Transitivity**: If $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$.

- Can you derive this rule using Armstrong's axioms?

- Derivation of the Pseudo-Transitivity rule:

  (to fill in)

Pseudo-Transitivity Rule

- **Pseudo-Transitivity**: If X $\rightarrow$ Y and WY $\rightarrow$ Z, then XW $\rightarrow$ Z.

- Can you derive this rule using Armstrong's axioms?

- Derivation of the Pseudo-Transitivity rule:

  X -> Y and WY -> Z

  XW -> WY (augmentation)

  WY -> Z (given)

  Therefore XW -> Z (transitivity)

Completeness of Armstrong's Axioms

- **Completeness**: If a set $\mathcal{F}$ of FDs implies F, then F can be derived from $\mathcal{F}$ by applying Armstrong's axioms.
  - If $\mathcal{F} \models F$, then $\mathcal{F} \vdash F$.

    semantic — syntactic
  - If $\mathcal{F}$ implies F, then F can be derived from $\mathcal{F}$ using Armstrong's axioms.

For those familiar with Mathematical Logic:

- $\mathcal{F} \models F$ is "model-theoretic"
- $\mathcal{F} \vdash F$ is "proof-theoretic"

Soundness of Armstrong's Axioms

- **Soundness**: If F can be derived from a set of FDs $\mathcal{F}$ through Armstrong's axioms, then $\mathcal{F}$ implies F.
  - If $\mathcal{F} \vdash$ F, then $\mathcal{F} \models$ F.
    - That is, if $\mathcal{F}$ can be derived from F using Armstrong's axioms, then $\mathcal{F}$ implies F.
  - Handwaving proof: If one can generate F from $\mathcal{F}$ using Armstrong's axioms, then surely $\mathcal{F}$ implies F. (Why?)

- With Completeness and Soundness, we know that

  $\mathcal{F} \vdash$ F if and only if $\mathcal{F} \models$ F

  In other words, the FDs that can be derived from $\mathcal{F}$ using Armstrong's axioms are precisely *all* the FDs that must hold under $\mathcal{F}$ (that is, all the axioms that $\mathcal{F}$ implies).
  - Great! But how can we decide whether or not $\mathcal{F}$ implies F?

Closure of a Set of FDs $\mathcal{F}$

- Let $\mathcal{F}^+$ denote the set of all FDs implied by a given set $\mathcal{F}$ of FDs.
  - Also called the closure of $\mathcal{F}$.
- To decide whether $\mathcal{F}$ implies F, first compute $\mathcal{F}^+$, then see whether F is a member of $\mathcal{F}^+$.
- Example: Compute $\mathcal{F}^+$ for the set { A → B, B → C} of FDs.
- Trivial FDs
  - A → A, B → B, C → C, AB → A, AB → B, BC → B, BC → C, AC → A, AC → C, ABC → A, ABC → B, ABC → C, ABC → AB , ABC → AC , ABC → BC, ABC → ABC
- Transitivity (non-trivial FDs)
  - AC → B AC → A (trivial), A → B (given), so AC → B (transitivity).
  - AB → C AB → B (trivial), B → C (given), so AB → C (transitivity).
  - A → C A → B (given), B → C (given), so A → C (transitivity).

Expensive and tedious! Let's find a better way.

Attribute Closure Algorithm

- Let X be a set of attributes and $\mathcal{F}$ be a set of FDs. The *attribute closure X^+ with respect to $\mathcal{F}$* is the set of all attributes A such that $X \rightarrow A$ can be derived from $\mathcal{F}$.
  - That is, all the attributes A such that $\mathcal{F} \vdash X \rightarrow A$

Input: A set X of attributes and a set $\mathcal{F}$ of FDs.

Output: X^+

```
Closure = X;                    // initialize Closure to equal the set X
repeat until no change in Closure {
        if there is an FD  U → V in F such that U ⊆ Closure,
        then Closure = Closure ∪ V;
}
return Closure;
```

- If $A \in$ Closure (that is, if $A \in X^+$) , then $X \rightarrow A$.
  - More strongly, $\mathcal{F} \vdash X \rightarrow A$ if and only $A \in X^+$

FD Example 1 using Attribute Closure

- $\mathcal{F}$ = { A → B, B → C }.
- Question: Does A → C?
- Compute A^+

- Closure = { A }
- Closure = { A, B } (due to A → B)
- Closure = { A, B, C } (due to B → C)
- Closure = { A, B, C }
  - no change, stop
- Therefore A^+ = {A, B, C }
- Since C ∈ A^+, answer YES.

FD Example 2 using Attribute Closure

- $\mathcal{F}$ = { AB → E, B → AC, BE → C }
- Question: Does BC → E?
- Compute BC^+

- Closure = { B, C }
- Closure = { A, B, C } (due to B → AC)
- Closure = { A, B, C, E } (due to AB → E)
- Closure = { A, B, C, E } (due to BE → C)
  - No change, so stop.
- Therefore BC^+ = {A,B,C,E}
- Since E ∈ BC^+, answer YES.

A Better Algorithm for FDs

It's much easier to compute Attribute Closure X^+, rather than FD Closure $\mathcal{F}^+$

- To determine if an FD $X \rightarrow Y$ is implied by $\mathcal{F}$, compute X^+ and check if $Y \subseteq X^+$.
- So computing the Attribute Closure X^+ is much less expensive (and less tedious) to compute than is the FD Closure $\mathcal{F}^+$.

Correctness of Algorithm

- Is it correct?

  Prove that the algorithm indeed computes X^+.

  - Show that for any attribute $A \in X^+$, it is the case that
    $X \rightarrow A$ is derivable from $\mathcal{F}$.
  - Show if $X \rightarrow A$ is derivable from $\mathcal{F}$, then it must be that
    $A \in X^+$.

Correctness of the Attribute Closure Algorithm

If $A \in X^+$, then $\mathcal{F} \vdash X \rightarrow A$

- This is true because Armstrong's Axioms are "sound".

If $\mathcal{F} \vdash X \rightarrow A$, then $A \in X^+$

- Proof by contradiction. Won't go through proof details.
- Please let me know if you'd like to see the proof.

Using Attribute Closure Algorithm to Find All Superkeys/Keys for Relation R, given Functional Dependencies $\mathcal{F}$

Attribute Closure algorithm can be modified to **find all superkeys and all candidate keys** for R, given Functional Dependencies $\mathcal{F}$.

- How?
  - Compute the closure of a single attribute in attr(R). Then compute the closure of every 2 attribute set, 3 attribute set, and so on.
  - If the closure of a set of attributes contains all attributes of relation R, then it is a *superkey* for R.
  - If no proper subset of those attributes has a closure that contains all attributes of the relation, then it is a *key* for R.

Using Attribute Closure Algorithm to Determine if a Set of Attributes X is a SuperKey/Key for Relation R, given Functional Dependencies $\mathcal{F}$

Attribute Closure algorithm can be modified to determine **if a set of attributes X is a superkey/key** for R, given Functional Dependencies $\mathcal{F}$.

- How?
  - Compute the attribute closure of X^+.
  - If X^+ = attr(R), then X is a *superkey* for R.
  - If no proper subset of X has a closure that contains all attributes of the relation, then X is a *key* for R.

Practice Homework: Design Theory

1. Let R(A,B,C,D,E) be a relation schema and let $\mathcal{F}$ = { AB $\rightarrow$ E, B $\rightarrow$ AC, BE $\rightarrow$ C } be a set of FDs that hold over R.
   a. Prove that $\mathcal{F} \models$ B $\rightarrow$ E using Armstrong's axioms.
   b. Compute the closure of B. That is, compute B^+.
   c. Give a key for R. Justify why your answer is a key for R.
   d. Show an example relation that satisfies $\mathcal{F}$.
   e. Show an example relation that does not satisfy $\mathcal{F}$.

2. Let R(A,B,C,D,E) be a relation schema and let $\mathcal{F}$ = { A $\rightarrow$ C, B $\rightarrow$ AE, B $\rightarrow$ D, BD $\rightarrow$ C } be a set of FDs that hold over R.
   a. Show that B $\rightarrow$ CD using Armstrong's axioms.
   b. Show a relation of R such that R satisfies $\mathcal{F}$ but R does not satisfy A $\rightarrow$ D.
   c. Is AB a key for R?