

Lecture 04: Functional Dependencies

15-445/645 Database Systems (Fall 2017)

Carnegie Mellon University

Prof. Andy Pavlo

Database Design

- Metrics for a good database design (Good database schema)
 1. Data Integrity (no loss of data) ← This lecture.
 2. Good performance
- Integrity vs performance is a common tradeoff in databases

Redundancy

- Issues with example database $Student(student_id, course_id, room, grade, name, address)$
 - Duplicate columns
 - Duplicate entries (Obama appears twice) → redundancy.
 - Update anomalies: Changes to room numbers means all records need to be updated
 - Insert anomalies: May be impossible to add student to DB if they're not enrolled in a course
 - Delete anomalies: If all students are deleted, we may lose the room number for a course
 - The changes done to better this database is called **decomposition** (ex) $\left\{ \begin{array}{l} Student(student_id, name, address) \\ Rooms(course_id, room) \\ Courses(student_id, course_id, grade) \end{array} \right.$
Obviously (looks better. (to humans))
- Functional Dependencies** → Why it's better. "reasoning".

- A functional dependency is a form of constraint
 - Basic idea: A value of a variable depends on the value of another variable
 - Example: $sid \rightarrow name$ because name depends on sid ("student ID implies name")
 - You can check if an FD is violated by an instance, but you can't prove a FD using just an instance
 - Two FDs $X \rightarrow Y$ and $X \rightarrow Z$ can be written $X \rightarrow (YZ) \rightarrow Y \cup Z$
 - But $XY \rightarrow Z$ is not the same as $X \rightarrow Z$ and $X \rightarrow Y$
 - Defining FDS in SQL
- schema → blueprint.
instance → realization.
Any FD can be violated with one more tuple.
- Def. $t_1[x] = t_2[x] \Rightarrow t_1[y] = t_2[y]$.
implies

```
CREATE ASSERTION student-name
CHECK (NOT EXISTS
(SELECT * FROM students AS s1,
students AS s2
WHERE s1.sid = s2.sid
AND s1.name <> s2.name))
```

$sid \rightarrow name$

- Issues with FDs in SQL
 - Performance: Need to validate FD across entire table when inserting or updating a tuple
 - **No major DBMS supports SQL-92 assertions**
- **FDs are important because they allow us to decide if our database design is correct**

could try for optimizing. but it is hard.

Closures and Canonical Covers

Given a set of FDs f_1, \dots, f_n we define the **Closure F^+** as the set of all implied FDs

Given a closure, the attribute closure is: Given an attribute X , the closure X^+ is the set of all attributes such that $X \rightarrow A$ can be inferred using **Armstrong's axioms**

Why are closures important

- Checking closure at runtime is expensive
- We want **minimal set of FDs that is enough to ensure correctness**

The minimal set of all FDs is called **the canonical cover**

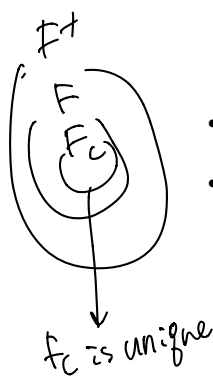
A canonical cover F_c must have the following properties

1. The RHS of every FD is a single attribute
2. The closure of F_c is identical to the closure of F $F_c^+ = F^+$
3. The F_c is minimal (deleting any attribute from LHS or RHS of an FD violates property #2)

Why do canonical covers matter

- the canonical cover is the minimum number of assertions needed to assure database integrity and correctness
- They allow us to find the **super key**

- implies
- Reflexivity $X \supseteq Y \Rightarrow X \rightarrow Y$
 - Augmentation $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
 - Transitivity $(X \rightarrow Y) \wedge (Y \rightarrow Z) \Rightarrow X \rightarrow Z$
 - Union $(X \rightarrow Y) \wedge (X \rightarrow Z) \Rightarrow X \rightarrow YZ$
 - Decomposition $X \rightarrow YZ \Rightarrow (X \rightarrow Y) \wedge (X \rightarrow Z)$
 - Pseudo-transitivity $(X \rightarrow Y) \wedge (YW \rightarrow Z) \Rightarrow XW \rightarrow Z$



Super and Candidate Keys

• **Super key:** set of attributes where no distinct tuples have the same values for these attributes

- Allow us to determine whether we can decompose a table into multiple sub-tables
- **Allow us to ensure that we are able to recreate the original relation through joins**

• **Candidate key:** set of attributes that uniquely identify a tuple according to a key constraint

• A candidate key is a super key, but not all super keys are candidates



Schema Decompositions

- **Objective:** Split a single relation R into a set of relations R_1, \dots, R_n
- **Goals (in order of importance)**
 1. (MANDATORY) Lossless joins: Want to be able to construct original relation by joining smaller ones using a natural join
 2. Dependency preservation: Minimize cost of global integrity constraints based on FD's. *Very expensive*
 3. Redundancy avoidance: avoid unnecessary data duplication
- A schema preserves dependencies if its original FD's do not span multiple tables

Lossless Joins

- Motivation: Avoid information loss.
- Goal: No noise introduced when reconstituting universal relation via joins.
- Test: At each decomposition $R=(R_1 \cup R_2)$, check whether $(R_1 \cap R_2) \rightarrow R_1$ or $(R_1 \cap R_2) \rightarrow R_2$.

Dependency Preservation

- Motivation: Efficient FD assertions.
- Goal: No global integrity constraints that require joins of more than one table with itself.
- Test: $R=(R_1 \cup \dots \cup R_n)$ is dependency preserving if closure of FD's covered by each $R_i =$ closure of FD's covered by $R=F$.

Redundancy Avoidance

- Motivation: Avoid update, delete anomalies.
- Goal: Avoid update anomalies, wasted space.
- Test: For an $X \rightarrow Y$ covered by R_n , X should be a super key of R_n .