Relational Database Design
Functional Dependencies

Functional Dependencies (FD) - Definition

Let R be a relation scheme and X, Y be sets of attributes in R.

A functional dependency from X to Y exists if and only if:

- For every instance of |R| of R, if two tuples in |R| agree on the values of the attributes in X, then they agree on the values of the attributes in Y

We write $X \rightarrow Y$ and say that X determines Y

Example on Student (sid, name, supervisor_id, specialization):

- {supervisor_id} $\rightarrow$ {specialization} means
  - If two student records have the same supervisor (e.g., Dimitris), then their specialization (e.g., Databases) must be the same
  - On the other hand, if the supervisors of 2 students are different, we do not care about their specializations (they may be the same or different).

Sometimes, I omit the brackets for simplicity:

- supervisor_id $\rightarrow$ specialization

Trivial FDs

A functional dependency $X \rightarrow Y$ is trivial if Y is a subset of X

- {name, supervisor_id} $\rightarrow$ {name}
  - If two records have the same values on both the name and supervisor_id attributes, then they obviously have the same name.
  - Trivial dependencies hold for all relation instances

A functional dependency $X \rightarrow Y$ is non-trivial if $Y \cap X = \emptyset$

- {supervisor_id} $\rightarrow$ {specialization}
  - Non-trivial FDs are given implicitly in the form of constraints when designing a database.
  - For instance, the specialization of a students must be the same as that of the supervisor.
  - They constrain the set of legal relation instances. For instance, if I try to insert two students under the same supervisor with different specializations, the insertion will be rejected by the DBMS

Functional Dependencies and Keys

A FD is a generalization of the notion of a *key.*

For Student (sid, name, supervisor_id, specialization),
we write:

{sid} $\rightarrow$ {name, supervisor_id, specialization}

- The sid determines all attributes (i.e., the entire record)
- If two tuples in the relation student have the same sid, then they must have the same values on all attributes.
- In other words they must be the same tuple (since the relational model does not allow duplicate records)

Superkeys and Candidate Keys

A set of attributes that determine the entire tuple is a **superkey**

- {sid, name} is a superkey for the student table.
- Also {sid, name, supervisor_id} etc.

A minimal set of attributes that determines the entire tuple is a **candidate key**

- {sid, name} is not a candidate key because I can remove the name.
- sid is a candidate key – so is HKID (provided that it is stored in the table).

If there are multiple candidate keys, the DB designer chooses designates one as the **primary key.**

Reasoning about Functional Dependencies

It is sometimes possible to infer new functional dependencies from a set of given functional dependencies

- independently from any particular instance of the relation scheme or of any additional knowledge

Example:

From

{sid} $\rightarrow$ {first_name} and

{sid} $\rightarrow${last_name}

We can infer

{sid} $\rightarrow$ {first_name, last_name}

Armstrong's Axioms

Be X, Y, Z be subset of the relation scheme of a relation R

Reflexivity:

If $Y \subseteq X$, then $X \to Y$ (trivial FDs)

- {name, supervisor_id}→{name}

Augmentation:

If $X \to Y$, then $X \cup Z \to Y \cup Z$

- if {supervisor_id} →{spesialization} ,
- then {supervisor_id, name}→{spesialization, name}

Transitivity:

If $X \to Y$ and $Y \to Z$, then $X \to Z$

- if {supervisor_id} →{spesialization} and {spesialization} →{lab}, then {supervisor_id}→{lab}

Properties of Armstrong's Axioms

Armstrong's axioms are sound (i.e., correct) and complete (i.e., they can produce all possible FDs)

Example: Transitivity
Let X, Y, Z be subsets of the relation R

If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Additional Rules based on Armstrong's axioms

Armstrong's axioms can be used to produce additional rules that are not basic, but useful:

Weak Augmentation rule: Let X, Y, Z be subsets of the relation R

$$\text{If } X \rightarrow Y \text{ , then } X \cup Z \rightarrow Y$$

Proof of soundness for Weak Augmentation

If $X \rightarrow Y$

(1) Then by Augmentation $X \cup Z \rightarrow Y \cup Z$

(2) And by Reflexivity $Y \cup Z \rightarrow Y$ because $Y \subset Y \cup Z$

(3) Then by Transitivity of (1) and (2) we have $X \cup Z \rightarrow Y$

Other useful rules:

If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y \cup Z$ **(union)**

If $X \rightarrow Y \cup Z$, then $X \rightarrow Y$ and $X \rightarrow Z$ **(decomposition)**

If $X \rightarrow Y$ and $ZY \rightarrow W$, then $ZX \rightarrow W$ **(pseudotransitivity)**

Closure of a Set of Functional Dependencies

For a set F of functional dependencies, we call the closure of F, noted F+, the set of all the functional dependencies that can be derived from F (by the application of Armstrong's axioms).

- Intuitively, F+ is equivalent to F, but it contains some additional FDs that are only implicit in F.

Consider the relation scheme R(A,B,C,D) with

F = {{A} →{B},{B,C} →{D}}

F+ = {

{A} →{A}, {B}→{B}, {C}→{C}, {D}→{D}, {A,B}→{A,B}, […],

{A}→{B}, {A,B}→{B}, {A,D}→{B,D}, **{A,C}→{B,C}**,
{A,C,D}→{B,C,D}, {A} →{A,B},

{A,D}→{A,B,D}, {A,C}→{A,B,C}, {A,C,D}→{A,B,C,D},

{B,C} →{D}, […], **{A,C} →{D}**, […]}

Finding Keys

Example: Consider the relation scheme R(A,B,C,D) with functional dependencies $\{A\}\rightarrow\{C\}$ and $\{B\}\rightarrow\{D\}$.

Is {A,B} a candidate key?

For {A,B} to be a candidate key, it must

- determine all attributes (i.e., be a superkey)
- be minimal

{A,B} is a superkey because:

- $\{A\}\rightarrow\{C\} \Rightarrow \{A,B\}\rightarrow\{A,B,C\}$ *(augmentation by AB)*
- $\{B\}\rightarrow\{D\} \Rightarrow \{A,B,C\}\rightarrow\{A,B,C,D\}$ *(augmentation by A,B,C)*
- We obtain $\{A,B\}\rightarrow\{A,B,C,D\}$ *(transitivity)*

Closure of a Set of Attributes

For a set X of attributes, we call the **closure** of X *(with respect to a set of functional dependencies F)*, noted X+, the maximum set of attributes such that $X \rightarrow X+$ *(as a consequence of F)*

Consider the relation scheme R(A,B,C,D) with functional dependencies $\{A\} \rightarrow \{C\}$ and $\{B\} \rightarrow \{D\}$.

- {A}+ = {A,C}
- {B}+ = {B,D}
- {C}+={C}
- {D}+={D}
- {A,B}+ = {A,B,C,D}

Redundancy of FDs

Sets of functional dependencies may have redundant dependencies that can be inferred from the others

- $\{A\}\rightarrow\{C\}$ is redundant in: $\{\{A\}\rightarrow\{B\}, \{B\}\rightarrow\{C\}, \{A\}\rightarrow \{C\}\}$

Parts of a functional dependency may be redundant

- Example of extraneous/redundant attribute on RHS:

$\{\{A\}\rightarrow\{B\}, \{B\}\rightarrow\{C\}, \{A\}\rightarrow\{C,D\}\}$ can be simplified to

$\{\{A\}\rightarrow\{B\}, \{B\}\rightarrow\{C\}, \{A\}\rightarrow\{D\}\}$

(because $\{A\}\rightarrow\{C\}$ is inferred from $\{A\} \rightarrow \{B\}, \{B\}\rightarrow\{C\}$)

- Example of extraneous/redundant attribute on LHS:

$\{\{A\}\rightarrow\{B\}, \{B\}\rightarrow\{C\}, \{A,C\}\rightarrow\{D\}\}$ can be simplified to

$\{\{A\}\rightarrow\{B\}, \{B\}\rightarrow\{C\}, \{A\}\rightarrow\{D\}\}$

(because of $\{A\}\rightarrow\{C\}$)

Canonical Cover

A *canonical cover* for F is a set of dependencies F_c such that

- F and $F_{c,}$ are equivalent
- F_c contains no redundancy
- Each left side of functional dependency in F_c is unique.
    - For instance, if we have two FD $X \rightarrow Y$, $X \rightarrow Z$, we convert them to $X \rightarrow Y \cup Z$.

Algorithm for canonical cover of F:

repeat

Use the union rule to replace any dependencies in F

$X_1 \rightarrow Y_1$ and $X_1 \rightarrow Y_2$ with $X_1 \rightarrow Y_1 Y_2$

Find a functional dependency $X \rightarrow Y$ with an extraneous attribute either in X or in Y

If an extraneous attribute is found, delete it from $X \rightarrow Y$

until F does not change

Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Example of Computing a Canonical Cover

$R = (A, B, C)$

$$
\begin{aligned}
F = \{A &\rightarrow BC \\
B &\rightarrow C \\
A &\rightarrow B \\
AB &\rightarrow C\}
\end{aligned}
$$

Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$

- Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

A is extraneous in $AB \rightarrow C$ because of $B \rightarrow C.$

- Set is now $\{A \rightarrow BC, B \rightarrow C\}$

C is extraneous in $A \rightarrow BC$ because of $A \rightarrow B$ and $B \rightarrow$ C.

The canonical cover is:

$$
\begin{aligned}
A &\rightarrow B \\
B &\rightarrow C
\end{aligned}
$$

Pitfalls in Relational Database Design

Functional dependencies can be used to refine ER diagrams or independently (i.e., by performing repetitive decompositions on a "universal" relation that contains all attributes).

Relational database design requires that we find a “good” collection of relation schemas. A bad design may lead to

- Repetition of Information.
- Inability to represent certain information.

Design Goals:

- Avoid redundant data
- Ensure that relationships among attributes are represented
- Facilitate the checking of updates for violation of database integrity constraints.

Example of Bad Design

Consider the relation schema: *Lending-schema* = (*branch-name, branch-city, assets, customer-name, loan-number, amount)* where:

{branch-name}$\rightarrow${branch-city, assets}

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

Bad Design

- Wastes space. Data for *branch-name, branch-city,* assets are repeated for each loan that a branch makes
- Complicates updating, introducing possibility of inconsistency of *assets* value
- Difficult to store information about a branch if no loans exist. Can use null values, but they are difficult to handle.

Usefulness of FDs

Use functional dependencies to decide whether a particular relation R is in "**good**" form.

In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that

- each relation is in good form
- the decomposition is a lossless-join decomposition
- if possible, preserve dependencies

In our example the problem occurs because there FDs ({branch-name}$\rightarrow${branch-city, assets}) where the LHS is not a key

Solution: decompose the relation schema *Lending-schema* into:

- *Branch-schema = (branch-name, branch-city,assets)*
- *Loan-info-schema = (customer-name, loan-number, branch-name, amount)*

Contoh

$R = (A,B,C,G,H,I)$

$F = \{A \rightarrow B,\ A \rightarrow C,\ CG \rightarrow H,\ CG \rightarrow I,\ B \rightarrow H\}$

Contoh

R = (A,B,C,G,H,I)

F = {$A \rightarrow B$, $A \rightarrow C$, $CG \rightarrow H$, $CG \rightarrow I$, $B \rightarrow H$}

Beberapa anggota F^+ :

- $A \rightarrow H$
  - Dengan menerapkan rule transitivity dari $A \rightarrow B$ dan $B \rightarrow H$
- $AG \rightarrow I$
  - Dengan menerapkan rule augmentation pada $A \rightarrow C$ berupa penambahan G, sehingga didapat $AG \rightarrow CG$, kemudian menerapkan transitivity dengan $CG \rightarrow I$
- $CG \rightarrow HI$
  - Didapat dari $CG \rightarrow H$ dan $CG \rightarrow I$ (union rule). Union rule diperoleh dari
    - Definisi functional dependency
    - Augmentation pada $CG \rightarrow I$ untuk mendapat $CG \rightarrow CGI$, augmentation $CG \rightarrow H$ untuk mendapat $CGI \rightarrow HI$, dan kemudian dilakukan transitivity