Database Technology

Topic 5: Functional Dependencies and Normalization

Olaf Hartig

olaf.hartig@liu.se

LINKÖPING UNIVERSITY

Motivation

- How can we be sure that the translation of an EER diagram into a relational schema results in a good database design?
- Given a deployed database, how can we be sure that it is well-designed?
- **What *is* a good database design?**
  - Informal measures
  - Formal measure: *normal forms*
    - Definition based on functional dependencies

Informal Measures

Example of Bad Design

EMP_DEPT

Ename	<u>Ssn</u>	Bdate	Address	Dnumber	Dname	Dmgr_ssn

- Every tuple contains employee data and department data
- Redundancy
  - Dname and Dmgr_ssn repeated for every employee in a department
- Potential for too many NULL values
  - Employees not in any department need to pad tuples with NULLs
- Update anomalies
  - Deleting the last employee in a department will result in deleting the department
  - Changing the department name or manager requires many tuples to be updated
  - Inserting employees requires checking for consistency of its department name and manager

Informal Measures

- Easy-to-explain meaning for each relation schema
  - Each relation schema should be about only one type of entities or relationships
  - Natural result of good ER design
- Minimal redundant information in tuples
  - Avoids update anomalies
  - Avoids wasted space
- Minimal number of NULL values in tuples
  - Avoids inefficient use of space
  - Avoids costly outer joins
  - Avoids ambiguous interpretation (e.g., unknown vs. does not apply)

Quiz

EMP_PROJ

<u>Ssn</u>	<u>Pnumber</u>	Hours	Ename	Pname	Plocation

The above relation schema representing the relationship between employees and the projects which they work on ...

A. … is an example of good design

B. … does not allow for an employee to work a different number of hours on each project that employee is assigned to

C. … uses exactly one tuple to record an employee’s name

D. … cannot be used in a straightforward manner to record the name and location of a project that has no employees assigned

Foundations of Formal Measures

Functional Dependencies (FDs) – Idea

- Assume that no two actors have the same name
- Each actor has a unique year and city of birth
- Thus, given an actor's name, there is only one possible value for birth year and city
  - name → yearOfBirth
  - name → cityOfBirth

Actor		
name	**yearOfBirth**	**cityOfBirth**
Ben Affleck	1972	Berkeley
Alan Arkin	1934	New York
Tommy Lee Jones	1946	San Saba
John Wells	1957	Alexandria
Steven Spielberg	1946	Cincinnati
Daniel Day-Lewis	1957	Greenwich

- However, given a birth year, we do not have a unique corresponding name or city
  - ~~yearOfBirth → name~~
  - ~~yearOfBirth → city~~
- Cannot tell from the example whether city determines name or birth year

Functional Dependencies (FDs) – Definition

- Constraint between two sets of attributes from a relation

Let R be a relational schema with the attributes $A_1, A_2, \ldots, A_n$ and let X and Y be subsets of $\{A_1, A_2, \ldots, A_n\}$.

Then, the functional dependency $X \rightarrow Y$ specifies the following constraint on *any* valid relation state r of R.

For *any* two tuples t_1 and t_2 in state r we have that:

$$\text{if } t_1[X] = t_2[X], \text{ then } t_1[Y] = t_2[Y] .$$

  - where $t[X]$ is the sequence of values that the tuple t has for the attributes in set X
  - We say "X determines Y" or "Y depends on X"

Running Example

- Consider the following relation schema

R(PID, PersonName, Country, Continent,
ContinentArea, NumberVisitsCountry)

- Functional dependencies?

FD1: PID → PersonName

FD2: PID, Country → NumberVisitsCountry

FD3: Country → Continent

FD4: Continent → ContinentArea

Identifying Functional Dependencies

- Property of the semantics (the meaning) of the attributes
- Recognized and recorded as part of database design
- Given an arbitrary relation state,
  - we cannot determine which FDs hold
  - we can observe that an FD does not hold
    if there are tuples that violate the FD

Trivial Functional Dependencies

- Some dependencies must always hold
  - {name, yearOfBirth} → {name, yearOfBirth}
  - {name, yearOfBirth} → {name}
  - {name, yearOfBirth} → {yearOfBirth}
- Formally:
  - Let R be a relation schema, and
  - let X and Y be subsets of attributes in R.
  - If Y is a subset of X, then $X \rightarrow Y$ holds trivially.

Implication for FDs

- Let R be a relational schema and let Σ be a set of FDs for R
- **Definition:** Σ is said to logically imply an FD $X \rightarrow Y$ if this FD holds in *all instances* of R that satisfy all FDs in Σ
  - Example: Σ = { FD3, FD4 } with FD3: Country → Continent and FD4: Continent → ContinentArea

    Then, Σ logically implies FD5: Country → ContinentArea
- **Definition:** The closure of Σ, denoted by Σ^+, is the set of all FDs that are logically implied by Σ
- Clearly, Σ is a subset of Σ^+. However, what else is in Σ^+?

Reasoning About FDs

- Logical implications can be derived by using inference rules called **Armstrong's rules**:
  - *Reflexivity:* If Y is a subset of X, then $X \rightarrow Y$
  - *Augmentation:* If $X \rightarrow Y$, then $XZ \rightarrow YZ$
    (we use XY as a short form for $X \cup Y$)
  - *Transitivity:* If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- These three rules are *sound*
  - i.e., given a set Σ of FDs, any FD that can be derived by applying these rules repeatedly is in Σ^+

- These three rules are *complete*
  - i.e., given a set Σ of FDs, by applying these rules repeatedly, we will eventually find every FD that is in Σ^+

Reasoning About FDs (cont'd)

- Logical implications can be derived by using inference rules called **Armstrong's rules**:
  - *Reflexivity:* If Y is a subset of X, then $X \rightarrow Y$
  - *Augmentation:* If $X \rightarrow Y$, then $XZ \rightarrow YZ$
    (we use XY as a short form for $X \cup Y$)
  - *Transitivity:* If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- Additional rules can be derived:
  - *Decomposition:* If $X \rightarrow YZ$, then $X \rightarrow Y$
  - *Union:* If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Pseudo-transitivity:* If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Running Example (cont'd)

- Recall *R*(PID, PersonName, Country, Continent, ContinentArea, NumberVisitsCountry) with:

  FD1: PID → PersonName

  FD2: PID, Country → NumberVisitsCountry

  FD3: Country → Continent

  FD4: Continent → ContinentArea

- Show that we also have *FD':* PID,Country → NumberVisitsCountry, Continent, ContinentArea, PersonName
  - *FD5:* Country → ContinentArea (transitive rule with FD3 and FD4)
  - *FD6:* Country → Continent, ContinentArea (union rule with FD3 and FD5)
  - *FD7:* PID, Country → PID, Continent, ContinentArea (augmentation of FD6)
  - *FD8:* PID, Country → Continent, ContinentArea (decomposition of FD7)
  - *FD9:* PID, Country → PersonName (augmentation + decomposition FD1)
  - Finally, *FD'* by union rule with FD2, FD8, and FD9

Revisiting Keys

- Given a relation schema R with attributes $A_1, A_2, \ldots, A_n$ and X a subset of these attributes
- X is a **superkey** of R if $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ or, $X \rightarrow \{A_1, A_2, \ldots, A_n\} \setminus X$
  - Often written as $X \rightarrow R$

- Given a set of FDs, how can we easily test whether $X \rightarrow R$?

> Let Σ be a set of FDs over the attributes of a relation R and let X be a subsets of these attributes.
>
> The **attribute closure** of X w.r.t. Σ is the maximum set of attributes functionally determined by X.

  - If the attribute closure of X contains all attributes, we have $X \rightarrow R$
  - The attribute closure can be computed in polynomial time ...

Computing (Super)Keys

function *ComputeAttrClosure*(X, Σ)
begin
    $X^+ := X$;
    while Σ contains an FD $Y \rightarrow Z$ such that
        (i) Y is a subset of X^+, and
        (ii) Z is not a subset of X^+ **do**
      $X^+ := X^+ \cup Z$;
    end while
    return X^+;
end

- Example: Recall R(PID, PersonName, Country, Continent, ContinentArea, NumberVisitsCountry) with:

  FD1: PID → PersonName

  FD2: PID, Country → NumberVisitsCountry

  FD3: Country → Continent

  FD4: Continent → ContinentArea

- The attribute closure of X = { PID, Country } w.r.t. FD1–FD4 is { PID, Country, PersonName, NumberVisitsCountry, Continent, ContinentArea }

Revisiting Keys (cont'd)

- Given a relation schema R with attributes $A_1, A_2, \ldots, A_n$ and X a subset of these attributes
- X is a **superkey** of R if $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ or, $X \rightarrow \{A_1, A_2, \ldots, A_n\} \setminus X$
  - Often written as $X \rightarrow R$
  - Can be tested easily by computing the attribute closure of X
- However, not every superkey is a candidate key
- To determine that X is a **candidate key** of R, we also need to show that no proper subset of X determines R
  - i.e., there does not exist a Y such that $Y \subsetneq X$ and $Y \rightarrow R$
- Hence, identifying *all* candidate keys is a matter of testing increasingly smaller subsets of $\{ A_1, A_2, \ldots, A_n \}$

Normal Forms and Normalization

Overview

- 1NF, 2NF, 3NF, BCNF (4NF, 5NF)
- Relation in higher normal form also satisfies the conditions of every lower normal form
- The higher the normal form, the less the redundancy
- 3NF and BCNF are our formal measure of good database design
  - Reduce redundancy
  - Reduce update anomalies
- *Normalization*: process of turning a set of relations that are in lower normal forms into relations that are in higher normal forms
  - by successively decomposing lower normal form relations

First Normal Form (1NF)

- Relational schema is in 1NF if it does not allow for non-atomic values

This is how we dealt with multi-valued attributes in the EER-to-RM translation.

R_{non1NF}

ID	**Name**	**LivesIn**
100	Pettersson	{Stockholm, Linköping}
101	Andersson	{Linköping}
102	Svensson	{Ystad, Hjo, Berlin}

Decompose

$R1_{1NF}$

ID	**Name**
100	Pettersson
101	Andersson
102	Svensson

$R2_{1NF}$

ID	***LivesIn***
100	*Stockholm*
100	*Linköping*
101	*Linköping*
102	*Ystad*
102	*Hjo*
102	*Berlin*

Preliminaries for Next Definitions

- 2NF, 3NF, and BCNF are defined in terms of
  FDs, candidate keys, and (non)prime attributes
  - **Prime attribute**: every attribute that is part of *some* candidate key
  - **Non-prime attribute**: every attribute that is not prime

Second Normal Form (2NF)

- Relation schema *R* is in 2NF if it is in 1NF and it does not have any **non-prime attributes** that are functionally dependent on a **part of a candidate key**

$\mathbf{R}_{\mathbf{non2NF}}$

EmpID	***Dept***	**Work%**	**EmpName**
100	*Dev*	50	Baker
100	*Support*	50	Baker
200	*Dev*	80	Miller

FD1: EmpID → EmpName

FD2: { EmpID, Dept } → { Work%, EmpName }

- Why do we want to avoid such a functional dependency in *R*?
  - Part of a candidate key can have repeated values
  - Then, so will have the non-prime attributes that depend on the part
  - Hence, redundancy and, thus, waste of space and update anomalies

Decomposition into 2NF

- Let *X* → *Y* be the FD that violates 2NF in a relation schema *R*
- Create a new relation schema *R1* with the attributes in *X* and in *Y*
- Remove attributes *Y* from *R* to form another new relation schema *R2*
- New relational database schema consists now of *R1* and *R2*

R_{non2NF}

EmpID	***Dept***	**Work%**	**EmpName**
100	*Dev*	50	Baker
100	*Support*	50	Baker
200	*Dev*	80	Miller

FD1: EmpID → EmpName

FD2: { EmpID, Dept } → { Work%, EmpName }

Decompose

$R1_{2NF}$

EmpID	**EmpName**
100	Baker
200	Miller

with FD1

$R2_{2NF}$

EmpID	***Dept***	**Work%**
100	*Dev*	50
100	*Support*	50
200	*Dev*	80

with FD3: {EmpID, Dept}→ {Work%}

Third Normal Form (3NF)

- Relation schema *R* is in 3NF if it is in 1NF and it does not have any **non-prime attributes** that are functionally dependent on a set of attributes that is **not a superkey**

$\mathbf{R}_{\mathbf{non3NF}}$

<u>ID</u>	**Name**	**Zip**	**City**
<u>100</u>	Andersson	58214	Linköping
<u>101</u>	Björk	10223	Stockholm
<u>102</u>	Carlsson	58214	Linköping

FD1: Zip → City

FD2: ID → { Name, Zip, City }

- Why do we want to avoid such a functional dependency in *R*?
  - Set of attributes that is not a candidate key can have repeated values
  - Then, so will have the non-prime attributes that depend on it
  - Hence, redundancy and, thus, waste of space and update anomalies

Decomposition into 3NF

- Let *X* → *Y* be the FD that violates 3NF in a relation schema *R*
- Create a new relation schema *R1* with the attributes in *X* and in *Y*
- Remove attributes *Y* from *R* to form another new relation schema *R2*
- New relational database schema consists now of *R1* and *R2*

R_{non3NF}

ID	**Name**	**Zip**	**City**
100	Andersson	58214	Linköping
101	Björk	10223	Stockholm
102	Carlsson	58214	Linköping

FD1: Zip → City

FD2: ID → { Name, Zip, City }

Decompose

$R1_{3NF}$

Zip	**City**
58214	Linköping
10223	Stockholm

with FD1

$R2_{3NF}$

ID	**Name**	**Zip**
100	Andersson	58214
101	Björk	10223
102	Carlsson	58214

with FD3: ID → {Name, ZIP}

Boyce-Codd Normal Form (BCNF)

- Relation schema *R* is in BCNF if it is in 1NF and
  for **every FD *X*→*Y*** on *R* we have that ***X* is a superkey**

- Example
  - Let *R*(<u>*A*</u>,<u>*B*</u>,*C*,*D*) be a relation schema with *AB*→*CD* and *C*→*B*
  - *AB* is a candidate key, and so is *AC* !
  - *R* is in 3NF (*D* is the only non-prime attribute and AB is a cand. key)
  - *R* is not in BCNF (because *C* is not a candidate key)

Decomposition into BCNF

- Let $X \rightarrow Y$ be the FD that violates BCNF in a relation schema R
- Create a new relation schema $R1$ with the attributes in X and in Y
- Remove attributes Y from R to form another new relation schema $R2$
- New relational database schema consists now of $R1$ and $R2$
- We may have to find a new primary key for $R1$ and for $R2$

- Example
  - Let $R(\underline{A},\underline{B},C,D)$ be a relation schema with $AB \rightarrow CD$ and $C \rightarrow B$
  - AB is a candidate key, and so is AC !
  - R is in 3NF (D is the only non-prime attribute and AB is a cand. key)
  - R is not in BCNF (because C is not a candidate key)
  - By using $C \rightarrow B$, we decompose R into
    $R1(\underline{C},B)$ with $C \rightarrow B$ and
    $R2(\underline{A},\underline{C},D)$ with $AC \rightarrow D$

Example

Running Example (cont'd)

- Recall *R*(PID, Country, PersonName, Continent, ContinentArea, NumberVisitsCountry) with:

  FD1: PID → PersonName

  FD2: PID, Country → NumberVisitsCountry

  FD3: Country → Continent

  FD4: Continent → ContinentArea

> 2NF: Non-prime attributes must not be functionally dependent on a part of a candidate key.

- Is *R* in 2NF?
  - No, non-prime attribute PersonName depends on PID (part of a candidate key)
  - Decompose *R* into *R1*(PID, PersonName) and *R2*(PID, Country, Continent, ContinentArea, NumberVisitsCountry)
  - *R1* is in 2NF, but *R2* is not because non-prime attributes Continent and ContinentArea depend on a part of a candidate key (Country)
  - Decompose *R2* into *R2X*(Country, Continent, ContinentArea) and *R2Y*(PID, Country, NumberVisitsCountry)
  - Now, with *R1*, *R2X*, and *R2Y* we have decomposed *R* into 2NF relations

Running Example (cont'd)

- Now we have *R1*(<u>PID</u>, PersonName),
  R2X(<u>Country</u>, Continent, ContinentArea), and
  R2Y(<u>PID</u>, <u>Country</u>, NumberVisitsCountry) with:

  FD1: PID → PersonName

  FD2: PID, Country → NumberVisitsCountry

  FD3: Country → Continent

  FD4: Continent → ContinentArea

> 3NF: Non-prime attributes must not be functionally dependent on a set of attributes that is not a superkey.

- Are *R1*, *R2X*, and *R2Y* in 3NF, respectively?
  - *R1* and *R2Y* are
  - *R2X* is not because of FD4
  - Decompose *R2X* into *R2XA*(<u>Country</u>, Continent) and *R2XB*(<u>Continent</u>, ContinentArea)
  - Now, with *R1*, *R2XA*, *R2XB*, and *R2Y* we have 3NF relations only

Running Example (cont'd)

- Now we have *R1*(<u>PID</u>, PersonName),
  R2XA(<u>Country</u>, Continent),
  R2XB(<u>Continent</u>, ContinentArea), and
  R2Y(<u>PID</u>, <u>Country</u>, NumberVisitsCountry) with:

  FD1: PID → PersonName

  FD2: PID, Country → NumberVisitsCountry

  FD3: Country → Continent

  FD4: Continent → ContinentArea

> BCNF: For every FD *X*→*Y* it holds that *X* is a superkey of the relation that the FD is associated with.

- Are *R1*, *R2XA*, *R2XB*, and *R2Y* in BCNF, respectively?
  - Yes.

Desirable Properties of Normalization

Desirable Properties

- Keep all the attributes from the initial schema
- **Non-additive join property** (also called *lossless join property)*
  - It must be possible that if we join the smaller relations produced by normalization, then we recover the initial relation without generating spurious tuples
  - Our normalization procedure has this property
- Example for a decomposition that does not have the property
  - Consider *R*(Student, Assignment, Mark)
  - Decomposition into *R1*(Student, Mark) and
    R2(Assignment, Mark)
  - There are instances of *R* for which joining their decomposed *R1* and *R2* (by *R1*.Mark=*R2*.Mark) result in another instance of *R* containing additional ("spurious") tuples that were not in the initial instance of *R*

Desirable Properties (cont'd)

- **Dependency preservation:** Each FD of the initial schema is preserved as an FD of at least one of the smaller relations produced by normalization
- Example: Consider *R*(<u>Proj</u>, Dept, Div) with *FD1:* Proj → Dept
  FD2: Dept → Div
  FD3: Proj → Div
  - *R* is not in BCNF (why?)
  - Two alternative decompositions into BCNF relations:

    D1: *R1*(<u>Proj</u>, Dept) with *FD1* and *R2*(<u>Dept</u>, Div) with *FD2*

    D2: *R1*(<u>Proj</u>, Dept) with *FD1* and *R3*(<u>Proj</u>, Div) with *FD3*

    (Notice that, by our normalization approach, we would not do D2 because FD3 does not violate the BCNF condition)
  - *D2* does not preserve *FD2*! D1 does because in *D1* it can be reconstructed by applying the transitivity rule to *FD1* and *FD3*.
- Our normalization procedure can guarantee dependency preservation only up to 3NF (that is, there are some cases in which the normalization into BCNF is not dependency preserving); same holds for any other procedure