Database Technology

Topic 6: 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	Ssn	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 or other special treatment in queries
  - Avoids ambiguous interpretation (e.g., unknown vs. does not apply)

Quiz

EMP_PROJ

Ssn	Pnumber	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

LINKÖPING UNIVERSITY

Functional Dependencies (FDs) – Idea

- Assume that no two actors have the same name
- Each actor has a unique city of birth in some country
- Thus, given an actor's name, there is only one possible value for birth city and country
  - name → cityOfBirth
  - name → countryOfBirth
- However, given a country of birth, we do not have a unique corresponding name
  - ~~countryOfBirth → name~~
- Cannot tell from the example whether city determines name or country

Actor

name	cityOfBirth	countryOfBirth
Ben Affleck	Berkeley	USA
Alan Arkin	New York	USA
Tommy Lee Jones	San Saba	USA
John Wells	Alexandria	USA
Steven Spielberg	Cincinnati	USA
Daniel Day-Lewis	Greenwich	UK

Preliminary Definition

Let *R* be a relational schema with the attributes $A_1, A_2, \ldots, A_n$, let *X* be a subset of $\{A_1, A_2, \ldots, A_n\}$, and let *t* be a tuple for *R*.
Then, we write ***t*[*X*]** to denote the sequence of values that *t* has for the attributes in set *X*.

- Example:
  - Assume *X* = {cityOfBirth} and *Y* = {name, cityOfBirth}
  - Let *t* be the first tuple in the example table
  - Then, *t*[*X*] is the tuple ('Berkley')

    and *t*[*Y*] is the tuple ('Ben Afflect', 'Berkley')

Actor

name	cityOfBirth	countryOfBirth
Ben Affleck	Berkeley	USA
Alan Arkin	New York	USA
Tommy Lee Jones	San Saba	USA
John Wells	Alexandria	USA
Steven Spielberg	Cincinnati	USA
Daniel Day-Lewis	Greenwich	UK

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* → *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:
if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

  - We say "*X* determines *Y*" or "*Y* depends on *X*"

Trivial Functional Dependencies

- Some dependencies must always hold
- Examples:
  - {name, cityOfBirth} → {name, cityOfBirth}
  - {name, cityOfBirth} → {name}
  - {name, cityOfBirth} → {cityOfBirth}
- 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* → *Y* holds trivially.

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

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

Implication and Closure

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

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

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* → *Y*
  - *Augmentation:* If *X* → *Y*, then *XZ* → *YZ*
    (we use *XY* as a short form for *X* U *Y*)
  - *Transitivity:* If *X* → *Y* and *Y* → *Z*, then *X* → *Z*
- These three rules are *sound*
  - i.e., given a set *F* of FDs, any FD that can be derived by applying these rules repeatedly is in F^+
- These three rules are *complete*
  - i.e., given a set *F* of FDs, by applying these rules repeatedly, we will eventually find every FD that is in F^+

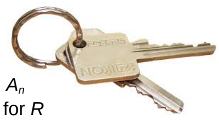
Reasoning About FDs (cont'd)

- Logical implications can be derived by using inference rules called **Armstrong's rules**:
  - *Reflexivity:* If *Y* is a subset of *X*, then *X* → *Y*
  - *Augmentation:* If *X* → *Y*, then *XZ* → *YZ*
    (we use *XY* as a short form for *X* U *Y*)
  - *Transitivity:* If *X* → *Y* and *Y* → *Z*, then *X* → *Z*
- Additional rules can be derived:
  - *Decomposition:* If *X* → *YZ*, then *X* → *Y*
  - *Union:* If *X* → *Y* and *X* → *Z*, then *X* → *YZ*
  - *Pseudo-transitivity:* If *X* → *Y* and *WY* → *Z*, then *WX* → *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 $A_1, A_2, \ldots, A_n$
  X a subset of these attributes, and *F* is a set of FDs for *R*
- *X* is a **superkey** of *R* if *X* → $\{A_1, A_2, \ldots, A_n\}$ is in F^+
  - Often written as *X* → *R*
- Given a set *F* of FDs, how can we easily test whether *X* → *R* is in F^+?

  Let *F* 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. *F* is the maximum set of attributes functionally determined by *X*.

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

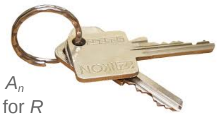
Picture source: https://pixabay.com/en/key-keychain-house-keys-door-key-2744636/

Computing (Super)Keys

function *ComputeAttrClosure*(X, F)
begin
  $X^+ := X$;
  while F 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 $A_1, A_2, \ldots, A_n$, X a subset of these attributes, and F is a set of FDs for R
- X is a **superkey** of R if $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ is in F^+
  - 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 { $A_1, A_2, \ldots, A_n$ }

Picture source: https://pixabay.com/en/key-keychain-house-keys-door-key-2744636/

Normal Forms

Overview

- (1NF, 2NF,) 3NF, BCNF (4NF, 5NF)
  - BCNF: Boyce-Codd Normal Form
- 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

Boyce-Codd Normal Form (BCNF)

- Relation schema R with a set F of functional dependencies is in BCNF if for **every** non-trivial **FD $X \rightarrow Y$ in F^+** we have that **X is a superkey**
  - Note that it is sufficient to check the FDs in F
- Example relation that is not in BCNF:

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

FD1: Zip → City
FD2: ID → { Name, Zip, City }

- Why do we want to avoid FDs whose left-hand-side is not a superkey?
  - Set of attributes that is not a superkey can have repeated values
  - So may have the attributes that depend on it
  - Hence, redundancy and, thus, waste of space and update anomalies

Quiz (Running Example)

- Relation schema R with a set F of functional dependencies is in BCNF if for **every** non-trivial **FD $X \rightarrow Y$ in F^+** we have that **X is a superkey**
- Recall R(PID, Country, PersonName, Continent, ContinentArea, NumberVisitsCountry) with:
  - *FD1:* PID → PersonName
  - *FD2:* PID, Country → NumberVisitsCountry
  - *FD3:* Country → Continent
  - *FD4:* Continent → ContinentArea
- Is R in BCNF?
  Yes / No
- What can we do about it? ► Decompose R

Desirable Properties of Decompositions

Attribute Preservation

- Of course, keep all the attributes from the initial schema !
- Formally:
  - Suppose attr(R) denotes the set of attributes in a relation schema R
  - Then, given a relation schema R, a set of relation schemas $R_1, \ldots, R_n$ is an attribute-preserving decomposition of R if

$$\text{attr}(R) = \bigcup_{i=1 \ldots n} \text{attr}(R_i)$$

Dependency Preservation

- Idea: every FD of the initial schema can be recovered based on the FDs of the schemas in the decomposition
- 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*
  - *D2* does not preserve *FD2*!
  - *D1* preserves *FD3* because in *D1*, *FD3* can be reconstructed by applying the transitivity rule to *FD1* and *FD2*

Dependency Preservation (formally)

- Let R be a relation schema with a set F of FDs
- Let $R_1, R_2, \ldots, R_n$ be a decomposition of R
- For every R_i we call the set of all FDs in F^+ that mention only attributes from R_i the restriction of F to R_i
- Then, the decomposition is dependency preserving if for the restrictions $F_1, F_2, \ldots, F_n$ of F to $R_1, R_2, \ldots, R_n$ it holds that

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

Picture source: https://pixabay.com/en/search-to-find-internet-1013910/

Non-Additive Join Property

- Also called *lossless join property*
- It must be possible that if we join the relations $R_1, \ldots, R_n$, then we recover the initial relation R without generating additional tuples (also called "spurious tuples")
- 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

Student	Assignment	Mark
Alice	A1	100
Bob	A1	80
Bob	A2	100

BCNF Decomposition Algorithm

Decomposition Step

- Let *X* → *Y* be the FD that violates BCNF in a relation schema *R*
- Replace *R* by two new relation schemas *R1* and *R2* constructed as follows
- Create *R1* with all the attributes in *X* and in *Y*
- Create *R2* from *R* by removing all attributes that are in *Y* and not in *X*
- Example: recall the example relation that was not in BCNF

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

Zip	City
58214	Linköping
10223	Stockholm

with FD1

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

with FD3: ID→ {Name, ZIP}

- Note that *R1* or *R2* may still not be in BCNF

Algorithm

function *DecomposeBCNF*(*R*, *F*)
begin
Result := *R*;
while there is a relation schema R_i in *Result* for which
the restriction of F^+ to R_i contains a non-trivial
FD *X* → *Y* that violates the BCNF condition
do
Decompose R_i into R_i1 and R_i2 as on the previous slide;
Replace R_i in *Result* by R_i1 and R_i2;
end while

return *Result*;
end

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
- *R* is not in BCNF (*FD1*, *FD3*, and *FD4* violate the BCNF condition)
- By using *FD1*, we decompose *R* into
  - *R1*(PID, PersonName) with *FD1*, and
  - *R2*(PID, Country, Continent, ContinentArea, NumberVisitsCountry) with *FD2*, *FD3*, and *FD4* (and others)
- Now, *R1* is in BCNF, but *R2* is not because of *FD3* and *FD4* (and others)
  - Hence, we need to decompose *R2* further ...

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
- Given *R2*(PID, Country, Continent, ContinentArea, NumberVisitsCountry) with *FD2*, *FD3*, and *FD4* (and others)
- We may decompose *R2* by using *FD3*, in which case we would end up with a BCNF decomposition of *R* that is not dependency preserving

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
- Given *R2*(PID, Country, Continent, ContinentArea, NumberVisitsCountry) with *FD2*, *FD3*, and *FD4* (and others)
- Let's use *FD4* instead, which gives us
  - *R2X*(Continent, ContinentArea) with *FD4*, and
  - *R2Y*(PID, Country, Continent, NumberVisitsCountry) with *FD2* and *FD3* (and others)
- *R2X* is in BCNF, but *R2Y* still is not because of *FD3*

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
- Given *R2Y*(PID, Country, Continent, NumberVisitsCountry) with *FD2* and *FD3* (and others)
- Since *FD3* violates BCNF for *R2Y* we use it to decompose *R2Y* into
  - *R2YA*(Country, Continent) with *FD3*, and
  - *R2YB*(PID, Country, NumberVisitsCountry) with *FD2*
- Finally, *R2YA* and *R2YB* are also in BCNF
- Hence, the result of decomposing *R* consists of *R1*, *R2X*, *R2YA,* and *R2YB*

Properties of the Algorithm

- Results depend on the FDs chosen for the decomposition steps
- Any resulting decomposition has the non-additive join property (lossless)
- Finding a dependency-preserving decomposition is not guaranteed,
  - even if one exists and may be found by choosing other (BCNF-violating) FDs for the decomposition steps

- For some cases, there does not exist any decomposition into BCNF relations that is lossless and dependency preserving
  - Example: *R*(*A*, *B*, *C*) with *FD1*: $AB \rightarrow C$ and *FD2*: $C \rightarrow B$
  - For 3NF, there always exists a decomposition that is lossless and dependency preserving (but our algorithm is not guaranteed to find it)

www.liu.se