COMP3311 24T1 Database Systems

week 8 - 1

Outline

- **Announcements**
- Recap
- Closures
- Normal Forms
- Normalisation

Announcement 1

- **Quiz 5**
  - Due on this Friday (5th April)
  - Just do it!

Announcement 2

- **Provisional marks for assignment 1**
  - available now
  - let us know if there's any issues
  - tutors are helping to mark your assignments manually

Announcement 3

- **Assignment - 2**
  - specifications will be released later this week
  - due date: **Week 10 Friday (19th April) 10:00pm (We will release the official notice thru Webcms... may need to further defer it)**
  - practicing pyscopg2 (actually still mainly about SQL queries)
  - will be using Python
- **Submit your special consideration requests to the portal earlier!**
  - received some approvals from the school for assignment 1 yesterday ...
- **Submit your ELP extension request 24h before the due date!**

Don’t worry about your final exam preparation. Assignment 2 will have much less questions. It won’t take you too long to finish.

Announcement 4

- **Help Session**
  - Thursday session is **officially** shifted to 11:00 - 13:00 (G05) next week
  - but that's our last physical lecture ...
  - That session is hosted by Kenneth
    - a very helpful, nice and experienced tutor!

Outline

- Announcements
- **Recap**
- Closures
- Normal Forms
- Normalisation

Recap

Redundancies

- 3 types - anomalies, triggers vs design

Functional Dependencies

- definition
- inference rules

- **F1. Reflexivity** e.g. $X \rightarrow X$
- **F2. Augmentation** e.g. $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- **F3. Transitivity** e.g. $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$
- **F4. Additivity** e.g. $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$
- **F5. Projectivity** e.g. $X \rightarrow YZ \Rightarrow X \rightarrow Y, X \rightarrow Z$
- **F6. Pseudotransitivity** e.g. $X \rightarrow Y, YZ \rightarrow W \Rightarrow XZ \rightarrow W$

Outline

- Announcements
- Recap
- **Closures**
- Normal Forms
- Normalisation

Closures

Given a set F of fds, how many new ***fds*** can we derive?

- For a finite set of attributes, there must be a finite set of derivable fds.

The largest collection of dependencies that can be derived from F is called the closure of F and is denoted $\mathbf{F^+}$.

Closures allow us to answer two interesting questions:

- is a particular dependency $X \rightarrow Y$ derivable from F?
- are two sets of dependencies F and G equivalent?

Closures - cont

For the question "is $X \rightarrow Y$ derivable from F?" ...

- compute the closure $\mathbf{F^+}$; check whether $\mathbf{X \rightarrow Y \in F^+}$

For the question "are F and G equivalent?" ...

- compute closures $\mathbf{F^+}$ and $\mathbf{G^+}$; check whether they're equal

Unfortunately, closures can be very large, e.g.

$R = ABC, \quad F = \{ AB \rightarrow C, \; C \rightarrow B \}$

$F^+ = \{ A \rightarrow A, \; AB \rightarrow A, \; AC \rightarrow A, \; AB \rightarrow B, \; BC \rightarrow B, \; ABC \rightarrow B,$
$C \rightarrow C, \; AC \rightarrow C, \; BC \rightarrow C, \; ABC \rightarrow C, \; AB \rightarrow AB, \; \ldots\ldots,$
$AB \rightarrow ABC, \; AB \rightarrow ABC, \; C \rightarrow B, \; C \rightarrow BC, \; AC \rightarrow B, \; AC \rightarrow AB \}$

Closures - cont

Algorithms based on F^+ rapidly become **infeasible**.

To solve this problem ...

- use closures based on sets of attributes rather than sets of fds.

Given a set X of attributes and a set F of fds, the closure of X (denoted X^+) is

- the largest set of attributes that can be derived from X using F

Determining X^+ from $\{X \rightarrow Y, Y \rightarrow Z\}$ ... XYZ = X+

For computation, $|X^+|$ is bounded by the number of attributes.

Closures - cont

Algorithm for computing attribute closure:

Input: F (set of FDs), X (starting attributes)
Output: X+ (attribute closure)

```
Closure = X
while (not done) {
  OldClosure = Closure
  for each A → B such that A ⊂ Closure
    add B to Closure
  if (Closure == OldClosure) done = true
}
```

Closures - cont

For the question "is $X \rightarrow Y$ derivable from F?" ...

- compute the closure X^+, check whether $Y \subset X^+$

For the question "are F and G equivalent?" ...

- for each dependency in G, check whether derivable from F
- for each dependency in F, check whether derivable from G
- if true for all, then $F \Rightarrow G$ and $G \Rightarrow F$ which implies $F^+ = G^+$

E

Attribute Closure Example

Let's consider a database for a simple online bookstore with the following attributes:

- B: Book_ID
- T: Title
- A: Author
- P: Publisher
- Y: Year

Set 1:

- B → T, A, P, Y (A Book_ID determines the Title, Author, Publisher, and Year)
- T, A → B (The Title and Author together determine the Book_ID)
- P, Y → T (The Publisher and Year together determine the Title)

Set 2:

- B → T, A (A Book_ID determines the Title and Author)
- B → P, Y (A Book_ID also determines the Publisher and Year)
- T, A → B (The Title and Author together determine the Book_ID)
- P, Y → T (The Publisher and Year together determine the Title)

Are Set 1 and Set 2 equal?

E

Attribute Closure Example - cont

Let's consider a database for a simple online bookstore with the following attributes:

Set 1:

- B → T, A, P, Y
- T, A → B
- P, Y → T

Set 2:

- B → T, A
- B → P, Y
- T, A → B
- P, Y → T

- **for each dependency in G, check whether derivable from F**
- **for each dependency in F, check whether derivable from G**
- **if true for all, then F ⇒ G and G ⇒ F which implies F+ = G+**

For Set 1:

- B^+ = BTAPY
- $\{TA\}^+$ = TAB
- $\{PY\}^+$ = PYT

For Set 2:

- B^+ = BTAPY
- {TA}+ = TAB
- $\{PY\}^+$ = PYT

Set 1 and Set 2 are equal

Closures - Determining Keys

For the question "what are the keys of R implied by F?" ...

- find subsets $K \subset R$ such that $K^+ = R$

E

Determine Keys Example

Example: determine primary keys for each of the following:

1. R= ABCDEFG, FD = { A → B, C → D, E → FG }
   - A? A+ = AB, no
   - AC? {AC}+ = ABCD, no
   - ACE? {ACE}+ = ABCDEFG, YES

2. R= ABCD, FD = { A → B, B → C, C → D }
   - A

3. R= ABC, FD = { A → B, B → C, C → A }
   - A or B or C

Closures – Minimal Covers

For a given application, we can define many different sets of fds with the same closure (e.g. F and G where $F^+ = G^+$)

Which one is best to "model" the application?

- any model has to be **complete** (i.e. capture entire semantics)
- models should be as **small** as possible (we use them to check DB validity after update; less checking is better)

Can we derive the **smallest complete** set of fds?

Minimal Covers - Definition

Minimal cover $\mathbf{F_c}$ for a set F of fds:

- F_c is equivalent to F
- all fds have the form $X \rightarrow A$ (where A is a single attribute)
- it is not possible to make F_c smaller
  - either by deleting an fd
  - or by deleting an attribute from an fd

An fd d is redundant if $(F-\{d\})^+ = F^+$

An attribute a is redundant if $(F-\{d\} \cup \{d'\})^+ = F^+$
(where d' is the same as d but with attribute a removed)

Minimal Covers – Calculation

Algorithm for computing minimal cover:

Inputs: set F of fds
Output: minimal cover F_c of F s.t. $F_c = F$

Step 1: put $f \in F_c$ into canonical form
Step 2: eliminate redundant attributes from $f \in F_c$
Step 3: eliminate redundant fds from F_c

Step 1: put fds into canonical form

for each $f \in F_c$ like $X \rightarrow \{A1,\ldots,An\}$
  remove $X \rightarrow \{A1,\ldots,An\}$ from F_c
  add $X \rightarrow A1, \ldots X \rightarrow An$ to F_c
end

- **F1. Reflexivity** e.g. $X \rightarrow X$
- **F2. Augmentation** e.g. $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- **F3. Transitivity** e.g. $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$
- **F4. Additivity** e.g. $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$
- **F5. Projectivity** e.g. $X \rightarrow YZ \Rightarrow X \rightarrow Y, X \rightarrow Z$
- **F6. Pseudotransitivity** e.g. $X \rightarrow Y, YZ \rightarrow W \Rightarrow XZ \rightarrow W$

Minimal Covers – Calculation (cont)

Step 2: eliminate redundant attributes

for each $f \in F_c$ like $X \rightarrow A$
  for each b in X
    $f' = (X-\{b\}) \rightarrow A$; $G = F_c - \{f\} \cup \{f'\}$
    if $(G^+ == F_c^+)$ $F_c = G$
  end
end

Step 3: eliminate redundant functional dependencies

for each $f \in F_c$
  $G = F_c - \{f\}$
  if $(G^+ == F_c^+)$ $F_c = G$
end

E

Minimal Covers Example

Example: compute minimal cover

$R = ABC, F = \{ A \rightarrow BC,\ \ B \rightarrow C,\ \ A \rightarrow B,\ \ AB \rightarrow C \}$

- canonical fds: $A \rightarrow B,\ \ A \rightarrow C,\ \ B \rightarrow C,\ \ AB \rightarrow C$
- redundant attrs: $A \rightarrow B,\ \ A \rightarrow C,\ \ B \rightarrow C,\ \ A\cancel{B} \rightarrow C$
- redundant fds: $A \rightarrow B,\ \ \cancel{A \rightarrow C},\ \ B \rightarrow C$

Result: $F_C = \{B \rightarrow C,\ \ A \rightarrow B\}$

Outline

- Announcements
- Recap
- Closures
- **Normal Forms**
- Normalisation

Normalisation

Normalisation: branch of relational theory providing design insights.

The goals of normalisation:

- be able to characterise the level of redundancy in a relational schema
- provide mechanisms for transforming schemas to remove redundancy

Normalisation draws heavily on the theory of functional dependencies.

Normalisation algorithms reduce the amount of redundancy in a schema

- by decomposition (break schema into connected pieces)

Normal Forms

Normalisation theory defines six normal forms (NFs).

- First,Second,Third Normal Forms (1NF,2NF,3NF) (Codd 1972)
- Boyce-Codd Normal Form (BCNF) (1974)
- Fourth Normal Form (4NF) (Zaniolo 1976, Fagin 1977)
- Fifth Normal Form (5NF) (Fagin 1979)

We say that "a schema is in xNF", which ...

- tells us something about the level of redundancy in the schema

1NF allows most redundancy; 5NF allows least redundancy.
For most practical purposes, **BCNF (or 3NF)** are acceptable NFs.

Normal Forms

1NF

- all attributes have atomic values
  we assume this as part of relational model,
  so every relation schema is in 1NF

2NF

- all non-key attributes fully depend on key
  (i.e. no partial dependencies)
  avoids much redundancy

3NF

BCNF

- **no attributes depends on non-key attrs**
  (i.e. no transitive dependencies)
  avoids most remaining redundancy

Normal Forms (cont)

In practice, BCNF and 3NF are the most important.

Boyce-Codd Normal Form (BCNF):

- eliminates all redundancy due to functional dependencies
- but may not preserve original functional dependencies

Third Normal Form (3NF):

- eliminates most (but not all) redundancy due to fds
- guaranteed to preserve all functional dependencies

Boyce-Codd Normal Form

A relation schema R is in BCNF w.r.t a set F of functional dependencies iff:

for all fds $X \rightarrow Y$ in F^+

- either $X \rightarrow Y$ is trivial (i.e. $Y \subset X$)
- or X is a superkey (i.e. non-strict superset of attributes in key)

A DB schema is in BCNF if all of its relation schemas are in BCNF.

Observations:

- any two-attribute relation is in BCNF
- any relation with key K, other attributes Y, and $K \rightarrow Y$ is in BCNF

E

Boyce-Codd Normal Form Example

Example: schema in BCNF

$R = ABCD,\ F = \{ A \rightarrow B, A \rightarrow C, A \rightarrow D \}$

Why?

- either $X \rightarrow Y$ is trivial (i.e. $Y \subset X$)
- or X is a superkey (i.e. non-strict superset of attributes in key)

key(R) = A, all fds have key on RHS

Example: schema not in BCNF

$R = ABCD,\ F = \{ A \rightarrow BCD, D \rightarrow B, BC \rightarrow AD \}$

Why?

if key(R) = A, $D \rightarrow B$ does not have key on LHS

if key(R) = BC, $D \rightarrow B$ does not have key on LHS

Third Normal Form

A relation schema R is in 3NF w.r.t a set F of functional dependencies iff:
for all fds $X \rightarrow Y$ in F^+

- either $X \rightarrow Y$ is trivial (i.e. $Y \subset X$)
- or X is a superkey
- or **Y is a single attribute from a key**

A DB schema is in 3NF if all relation schemas are in 3NF.

The extra condition represents a slight weakening of BCNF requirements.

Third Normal Form (cont)

If we transform a schema into 3NF, we are guaranteed:

- lossless join decomposition
- the new schema preserves all of the fds from the original schema

However, we are not guaranteed:

- no update anomalies due to fd-based redundancy

Whether to use BCNF or 3NF depends on overall design considerations.

E

3rd Normal Form Example

Example: schema in 3NF

Why?

R = ABCDE, F = { B → ACDE, E → B }, key(R) = B

in E → B, E is not a key, but B is

- either X → Y is trivial (i.e. Y ⊂ X)
- or X is a superkey
- or Y is a single attribute from a key

Example: schema not in 3NF

Why?

R = ABCDE, F = { B → ACDE, E → D }, key(R) = B

in E → D, E is not a key, neither is D

Outline

- Announcements
- Recap
- Closures
- Normal Forms
- **Normalisation**

Normalisation

Normalisation aims to put a schema into xNF

- by ensuring that all relations in the schema are in xNF

How normalisation works ...

- decide which normal form xNF is "acceptable"
  - i.e. how much redundancy are we willing to tolerate?
- check whether each relation in schema is in xNF
- if a relation is not in xNF
  - **partition into sub-relations** where each is "closer to" xNF
- repeat until all relations in schema are in xNF

Relation Decomposition

The standard transformation technique to remove redundancy:

- decompose relation R into relations S and T

We accomplish decomposition by

- selecting (overlapping) subsets of attributes
- forming new relations based on attribute subsets

Properties: $R = S \cup T$, $S \cap T \neq \varnothing$ and $r(R) = s(S) \bowtie t(T)$

May require several decompositions to achieve acceptable NF.

Normalisation algorithms tell us how to choose S and T.

Schema (Re)Design

Consider the following relation for BankLoans:

branchName	branchCity	assets	custName	loanNo	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-15	1500
Mianus	Horseneck	400000	Jones	L-93	500
Round Hill	Horseneck	8000000	Turner	L-11	900
North Town	Rye	3700000	Hayes	L-16	1300

This schema has all of the update anomalies mentioned earlier.

Schema (Re)Design (cont)

To improve the design, decompose the BankLoans relation.

branchName	branchCity	assets	custName	loanNo	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-15	1500
Mianus	Horseneck	400000	Jones	L-93	500
Round Hill	Horseneck	8000000	Turner	L-11	900
North Town	Rye	3700000	Hayes	L-16	1300

The following decomposition is not helpful:

Branch(branchName, branchCity, assets)
CustLoan(custName, loanNo, amount)

because we lose information (which branch is a loan held at?)

Another possible decomposition:

BranchCust(branchName, branchCity, assets, custName)
CustLoan(custName, loanNo, amount)

Schema (Re)Design (cont)

Another possible decomposition:

BranchCust(branchName, branchCity, assets, custName)
CustLoan(custName, loanNo, amount)

branchName	branchCity	assets	custName
Downtown	Brooklyn	9000000	Jones
Redwood	Palo Alto	2100000	Smith
Perryridge	Horseneck	1700000	Hayes
Downtown	Brooklyn	9000000	Jackson
Mianus	Horseneck	400000	Jones
Round Hill	Horseneck	8000000	Turner
North Town	Rye	3700000	Hayes

custName	loanNo	amount
Jones	L-17	1000
Smith	L-23	2000
Hayes	L-15	1500
Jackson	L-15	1500
Jones	L-93	500
Turner	L-11	900
Hayes	L-16	1300

Schema (Re)Design (cont)

branchName	branchCity	assets	custName
Downtown	Brooklyn	9000000	Jones
Redwood	Palo Alto	2100000	Smith
Perryridge	Horseneck	1700000	Hayes
Downtown	Brooklyn	9000000	Jackson
Mianus	Horseneck	400000	Jones
Round Hill	Horseneck	8000000	Turner
North Town	Rye	3700000	Hayes

custName	loanNo	amount
Jones	L-17	1000
Smith	L-23	2000
Hayes	L-15	1500
Jackson	L-15	1500
Jones	L-93	500
Turner	L-11	900
Hayes	L-16	1300

branchName	branchCity	assets	custName	loanNo	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-15	1500
Mianus	Horseneck	400000	Jones	L-93	500
Round Hill	Horseneck	8000000	Turner	L-11	900
North Town	Rye	3700000	Hayes	L-16	1300

Different from the original table !

Now consider the result of
(BranchCust ⋈ CustLoan)

branchName	branchCity	assets	custName	loanNo	amount
Downtown	Brooklyn	9000000	Jones	L-17	1000
Downtown	Brooklyn	9000000	Jones	L-93	500
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Perryridge	Horseneck	1700000	Hayes	L-16	1300
Downtown	Brooklyn	9000000	Jackson	L-15	1500
Mianus	Horseneck	400000	Jones	L-93	500
Mianus	Horseneck	400000	Jones	L-17	1000
Round Hill	Horseneck	8000000	Turner	L-11	900
North Town	Rye	3700000	Hayes	L-16	1300
North Town	Rye	3700000	Hayes	L-15	1500

Schema (Re)Design (cont)

This is clearly not a successful decomposition.

The fact that we ended up with extra tuples was symptomatic of **losing some critical "connection" information** during the decomposition.

Such a decomposition is called **a lossy decomposition**.

In a good decomposition, we should be able to reconstruct the original relation exactly:

- if R is decomposed into S and T, then $S \bowtie T = R$
- Such a decomposition is called **lossless join decomposition**.

BCNF Normalisation Algorithm

The following algorithm converts an arbitrary schema to BCNF:

```
Inputs: schema R, set F of fds
Output: set Res of BCNF schemas

Res = {R};
while (any schema S ∈ Res is not in BCNF) {
   choose any fd X → Y on S that violates BCNF
   Res = (Res-S) ∪ (S-Y) ∪ XY
}
```

The last step means: make a table from XY; drop Y from table S

The "choose any" step means that the algorithm is non-deterministic

E

BCNF Normalisation Example

Recall the BankLoans schema:

BankLoans(branchName, branchCity, assets, custName, loanNo, amount)

Rename to simplify ...

B = branchName, C = branchCity, A = assets, N = CustName, L = loanNo, M = amount

So ... R = BCANLM, F = { B → CA, L → MN }, key(R) = BL,
R is not in BCNF, because B → CA is not a whole key

- either X → Y is trivial (i.e. Y ⊂ X)
- or X is a superkey (i.e. non-strict superset of attributes in key)

E

BCNF Normalisation Example (cont)

- either X → Y is trivial (i.e. Y ⊂ X)
- or X is a superkey (i.e. non-strict superset of attributes in key)

R = BCANLM, F = { B → CA, L → MN }, key(R) = BL,
R is not in BCNF, because B → CA is not a whole key

Decompose into

- S = BCA, FS = { B → CA } key(S) = B (for B → CA)
- T = BNLM, FT = { L → NM }, key(T) = BL (removing CA from R)

S = BCA is in BCNF, only one fd and it has key on LHS
T = BLNM is not in BCNF, because L → NM is not a whole key

Decompose into

- U = LNM, FU = { L → NM }, key(U) = L, which is BCNF, (for L → NM)
- V = BL, FV = {}, key(V) = BL, which is BCNF, (removing NM from T)

Result:

- S = (branchName, branchCity, assets) = Branches
- U = (loanNo, custName, amount) = Loans
- V = (branchName, loanNo) = BranchOfLoan

3NF Normalisation Algorithm

The following algorithm converts an arbitrary schema to 3NF:

Inputs: schema R, set F of fds
Output: set Ri of 3NF schemas

let F_c be **a reduced minimal cover** for F
Res = {}
for each fd $X \rightarrow Y$ in F_c {
  if (no schema $S \in$ Res contains XY) {
    Res = Res $\cup$ XY
  }
}
if (no schema $S \in$ Res contains a key for R) {
  K = any candidate key for R
  Res = Res $\cup$ K
}

E

3NF Normalisation Example

Recall the BankLoans schema:

BankLoans(branchName, branchCity, assets, custName, loanNo, amount)

Rename to simplify ...

B = branchName, C = branchCity, A = assets, N = CustName, L = loanNo, M = amount
R = BCANLM, F = { B → CA, L → MN }, key(R) = BL

Compute minimal cover = { B → C, B → A, L → M, L → N }

Reduce minimal cover = { B → CA, L → MN }

Convert into relations: S = BCA, T = LNM

No relation has key BL, so add new table containing key U = BL

Result is S = BCA, T = LNM, U = BL ... same as BCNF

3NF Normalisation Algorithm

To achieve a "good" database design:

- identify attributes, entities, relationships → ER design
- map ER design to relational schema
- identify constraints (including keys and functional dependencies)
- apply BCNF/3NF algorithms to produce normalised schema

Note: may subsequently need to "denormalise" if the design yields inadequate performance.

Thank you!