COMP9311 Week 9 Lecture

Relational Design Theory

Relational Design Theory

2/55

Previously we studied ER design and ER-to-relational mapping.

We claimed that this allows us to produce "good" schemas.

Can we make a stronger/formal statement on what makes a schema good?

The study of relational design theory

- examines some foundational notions of "schema goodness"
- provides methods to transform schemas to make them better

... Relational Design Theory

3/55

Functional dependencies

- are constraints between attributes within a relation
- have implications for "good" relational schema design

What we study here:

- basic theory and definition of functional dependencies
- methodology for improving schema designs (normalisation)

The aim of studying this:

- improve understanding of relationships among data
- gain enough formalism to assist practical database design

Relational Design and Redundancy

4/55

A good relational database design:

- must capture all necessary attributes/associations
- do this with a minimal amount of stored information

Minimal stored information ⇒ no redundant data.

In database design, redundancy is generally a "bad thing":

- causes problems maintaining consistency after updates

But ... redundancy may give performance improvements

- e.g. avoid a join to collect bits of data together

5/55

... Relational Design and Redundancy

Consider the following relation defining bank accounts/branches:

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

We need to be careful updating this data, otherwise we may introduce inconsistencies.

... Relational Design and Redundancy

6/55

Insertion anomaly:

- when we insert a new record, we need to check that branch data is consistent with existing tuples

Update anomaly:

- if a branch changes address, we need to update all tuples referring to that branch

Deletion anomaly:

- if we remove information about the last account at a branch, all of the branch information disappears

(Insertion/update anomalies are avoidable, but need extra DBMS work)

... Relational Design and Redundancy

7/55

Insertion anomaly example (insert account A-306 at Round Hill):

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
A-306	800	1111111	Round Hill	Horseneck	8000800

8/55

... Relational Design and Redundancy

Update anomaly example (update Round Hill branch address):

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Palo Alto	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

... Relational Design and Redundancy

Deletion anomaly example (remove account A-101):

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Where is the Downtown branch located? What are its assets?

Database Design (revisited)

To avoid these kinds of update problems:

- need a schema with "minimal overlap" between tables
- each table contains a "coherent" collection of data values

One way to acheive this:

- start with a "universal relation" U (all relevant attrs)
- decompose relation U into several smaller relations R_i
- where each R_i has "minimal overlap" with other R_j
- but there is sufficient overlap to reconstruct original table

Typically, each R_i contains info about one entity (e.g. branch, customer, ...)

... Database Design (revisited)

But we already have a design procedure (ER-then-relational)

- and it appears to produce schemas without redundancy

Why do we need another design procedure?

1. ER design does not guarantee minimal redundancy
   - dependency theory allows us to check designs for residual problems
2. the new procedure gives (semi)automated design
   - determine all of the attributes in the problem domain
   - collect them all together in a "universal relation"
   - provide information about how attributes are related
   - apply normalisation to decompose into non-redundant relations

---

Notation/Terminology

12/55

Most texts adopt the following terminology:

Relation schemas	upper-case letters, denoting set of all attributes (e.g. R, S, P, Q)
Relation instances	lower-case letter corresponding to schema (e.g. r(R), s(S), p(P), q(Q))
Tuples	lower-case letters (e.g. t, t', t_1, u, v)
Attributes	upper-case letters from start of alphabet (e.g. A, B, C, D)
Sets of attributes	simple concatenation of attribute names (e.g. X=ABCD, Y=EFG)
Attributes in tuples	tuple[attrSet] (e.g. t[ABCD], t[X])

Functional Dependency

13/55

A relation instance r(R) satisfies a dependency X → Y if

- for any t, u ∈ r, t[X] = u[X] ⇒ t[Y] = u[Y]

In other words, if two tuples in R agree in their values for the set of attributes X, then they must also agree in their values for the set of attributes Y.

We say that "Y is functionally dependent on X".

Attribute sets X and Y may overlap; trivially true that X → X.

Notes:

- X → Y can also be read as "X determines Y"
- the single arrow → denotes functional dependency
- the double arrow ⇒ denotes logical implication

... Functional Dependency

14/55

Consider the following instance r(R) of the relation schema R(ABCDE):

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_1	c_2	d_2	e_1
a_3	b_2	c_1	d_1	e_1
a_4	b_2	c_2	d_2	e_1
a_5	b_3	c_3	d_1	e_1

What kind of dependencies can we observe among the attributes in r(R)?

... Functional Dependency

15/55

Since A values are unique, the definition of fd gives:

- A → B, A → C, A → D, A → E
- A → BC, A → CD, ... A → BCDE
- can be summarised as A → BCDE

Since all E values are the same, it follows that:

- A → E, B → E, C → E, D → E
- in general, cannot be summarised as ABCD → E

... Functional Dependency

16/55

Other observations:

- combinations of BC are unique, therefore BC → ADE
- combinations of BD are unique, therefore BD → ACE
- if C values match, so do D values, therefore C → D
- however, D values don't determine C values, so !(D → C)

We could derive many other dependencies, e.g. AE → BC, ...

In practice, choose a minimal set of fds (basis)

- from which all other fds can be derived
- which captures useful problem-domain information

Exercise: Functional Dependencies (1)

17/55

Real estate agents conduct visits to rental properties

- need to record which property, who went, when, results
- each property is assigned a unique code (P#, e.g. PG4)
- each staff member has a staff number (S#, e.g. SG43)
- staff members use company cars to conduct visits
- a visit occurs at a specific time on a given day
- notes are made on the state of the property after each visit

The company stores all of the associated data in a spreadsheet.

... Exercise: Functional Dependencies (1)

Describe any functional dependencies that exist in this data:

P#	When	Address	Notes	S#	Name	CarReg
PG4	03/06 15:15	55 High St	Bathroom leak	SG44	Rob	ABK754
PG1	04/06 11:10	47 High St	All ok	SG44	Rob	ABK754
PG4	03/07 12:30	55 High St	All ok	SG43	Dave	ATS123
PG1	05/07 15:00	47 High St	Broken window	SG44	Rob	ABK754
PG1	05/07 15:00	47 High St	Leaking tap	SG44	Rob	ABK754
PG2	13/07 12:00	12 High St	All ok	SG42	Peter	ATS123
PG1	10/08 09:00	47 High St	Window fixed	SG42	Peter	ATS123
PG3	11/08 14:00	99 High St	All ok	SG41	John	AAA001
PG4	13/08 10:00	55 High St	All ok	SG44	Rob	ABK754
PG3	05/09 11:15	99 High St	Bathroom leak	SG42	Peter	ATS123

State assumptions used in determining the fds.

Exercise: Functional Dependencies (2)

What functional dependencies exist in the following table:

A	B	C	D
1	a	6	x
2	b	7	y
3	c	7	z
4	d	6	x
5	a	6	y
6	b	7	z
7	c	7	x
8	d	6	y

How is this case different to the previous one?

Functional Dependency

The above discussion was about dependency within a relation instance r(R).

Much more important for design is the notion of dependency across all possible instances of the relation (i.e. a schema-based dependency).

This is a simple generalisation of the previous definition:

- for any t, u ∈ any r(R), t[X] = u[X] ⇒ t[Y] = u[Y]

Useful because such dependencies reflect the semantics of the problem domain.

... Functional Dependency

Can we generalise some ideas about functional dependency?

E.g. are there dependencies that hold for any relation?

- yes, but they're generally trivial, e.g. Y ⊂ X ⇒ X → Y

E.g. do some dependencies suggest the existence of others?

- yes, rules of inference allow us to derive dependencies
- allows us to reason about sets of functional dependencies

Inference Rules

Armstrong's rules are complete, general rules of inference on fds.

F1. Reflexivity e.g. $X \rightarrow X$

- a formal statement of trivial dependencies; useful for derivations

F2. Augmentation e.g. $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

- if a dependency holds, then we can freely expand its left hand side

F3. Transitivity e.g. $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

- the "most powerful" inference rule; useful in multi-step derivations

---

... Inference Rules

Armstrong's rules are complete, but other useful rules exist:

F4. Additivity e.g. $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$

- useful for constructing new right hand sides of fds (also called union)

F5. Projectivity e.g. $X \rightarrow YZ \Rightarrow X \rightarrow Y, X \rightarrow Z$

- useful for reducing right hand sides of fds (also called decomposition)

F6. Pseudotransitivity e.g. $X \rightarrow Y, YZ \rightarrow W \Rightarrow XZ \rightarrow W$

- shorthand for a common transitivity derivation

... Inference Rules

Example: determining validity of $AB \rightarrow GH$, given:

$$R = ABCDEFGHIJ$$

$$F = \{ AB \rightarrow E, \quad AG \rightarrow J, \quad BE \rightarrow I, \quad E \rightarrow G, \quad GI \rightarrow H \}$$

Derivation:

1.	$AB \rightarrow E$	(given)
2.	$AB \rightarrow AB$	(using F1)
3.	$AB \rightarrow B$	(using F5 on 2)
4.	$AB \rightarrow BE$	(using F4 on 1,3)
5.	$BE \rightarrow I$	(given)
6.	$AB \rightarrow I$	(using F3 on 4,5)
7.	$E \rightarrow G$	(given)

8. AB → G (using F3 on 1,7)
9. AB → GI (using F4 on 6,8)
10. GI → H (given)
11. AB → H (using F3 on 6,8)
12. AB → GH (using F4 on 8,11)

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 fds.

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

Closures allow us to answer two interesting questions:

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

... Closures

For the question "is X → Y derivable from F?" ...

- compute the closure F^+; check whether X → Y ∈ F^+

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

- compute closures F^+ and G^+; check whether they're equal

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

R = ABC, F = { AB → C, C → B }
F^+ = { A → A, AB → A, AC → A, AB → B, BC → B, ABC → B,
C → C, AC → C, BC → C, ABC → C, AB → AB, ... ,
AB → ABC, AB → ABC, C → B, C → BC, AC → B, AC → AB }

... Closures

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 largest set of attributes that can be derived from X using F, is called the closure of X (denoted X^+).

We can prove (using additivity) that (X → Y) ∈ F^+ iff Y ⊂ X^+.

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

... Closures

For the question "is X → 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 ⇒ G and G ⇒ F which implies $F^+ = G^+$

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

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

Exercise: Computing Keys

Determine possible primary keys for each of the following:

1. FD = { A → B, C → D, E → FG }
2. FD = { A → B, B → C, C → D }
3. FD = { A → B, B → C, C → A }
4. FD = { ABH → C, A → D, C → E F → A, E → F, BGH → E }

Normalization

Normalization: branch of relational theory providing design insights.

The goals of normalization:

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

Normalization draws heavily on the theory of functional dependencies.

Normal Forms

Normalization 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)

NF hierarachy: 5NF ⇒ 4NF ⇒ BCNF ⇒ 3NF ⇒ 2NF ⇒ 1NF

1NF allows most redundancy; 5NF allows least redundancy.

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

---

... Normal Forms

33/55

The normalization process:

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

... Normal Forms

34/55

In practice, BCNF and 3NF are the most important.
(these are the "acceptable normal forms" for relational design)

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

Relation Decomposition

35/55

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

May require several decompositions to achieve acceptable NF.

Normalization algorithms tell us how to choose S and T.

Schema 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 Design

To improve the design, decompose the BankLoans relation.

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 Design

The BranchCust relation instance:

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

... Schema Design

The CustLoan relation instance:

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 Design

Now consider the result of (BranchCust Join 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 Design

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 Join(S,T) = R

Such a decomposition is called lossless join decomposition.

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

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

Observations:

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

... Boyce-Codd Normal Form

If we transform a schema into BCNF, we are guaranteed:

- no update anomalies due to fd-based redundancy
- lossless join decomposition

However, we are not guaranteed:

- the new schema preserves all fds from the original schema

This may be a problem if the fds contain significant semantic information about the problem domain.

If we need to preserve dependencies, use 3NF.

BCNF Decomposition

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

... BCNF Decomposition

Example (the BankLoans schema):

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

Has functional dependencies F

- branchName → assets,branchCity
- loanNo → amount,branchName

The key for BankLoans is branchName, custName, loanNo

... BCNF Decomposition

Applying the BCNF algorithm:

- check BankLoans relation ... it is not in BCNF
  (branchName → assets, branchCity violates BCNF criteria; LHS is not a key)
- to fix ... decompose BankLoans into

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

- check Branch relation ... it is in BCNF
  (the only nontrivial fds have LHS=branchName, which is a key)

(continued)

... BCNF Decomposition

Applying the BCNF algorithm (cont):

- check LoanInfo relation ... it is not in BCNF
  (loanNo → amount, branchName violates BCNF criteria; LHS is not a key)
- to fix ... decompose LoanInfo into

  Loan(branchName, loanNo, amount)
  Borrower(custName, loanNo)

- check Loan ... it is in BCNF
- check Borrower ... it is in BCNF

Exercise: BCNF Decomposition (1)

Consider the schema R and set of fds F

R = ABCDEFGH

F = { ABH → C, A → DE, BGH → F, F → ADH, BH → GE }

Produce a BCNF decomposition of R.

Exercise: BCNF Decomposition (2)

Recall the following table (based on e.g. a spreadsheet):

P#	When	Address	Notes	S#	Name	CarReg
PG4	03/06 15:15	55 High St	Bathroom leak	SG44	Rob	ABK754
PG1	04/06 11:10	47 High St	All ok	SG44	Rob	ABK754
PG4	03/07 12:30	55 High St	All ok	SG43	Dave	ATS123
PG1	05/07 15:00	47 High St	Broken window	SG44	Rob	ABK754
PG1	05/07 15:00	47 High St	Leaking tap	SG44	Rob	ABK754

PG2 | 13/07 12:00 | 12 High St | All ok | SG42 | Peter | ATS123
...

Recall the functional dependencies identified previously,

Use these to convert the table to a BCNF schema.

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

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.

... Third Normal Form

The following algorithm converts an arbitrary schema to 3NF:

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

let Fc be a minimal cover for F
Res = {}
for each fd X → Y in Fc {
    if (no schema S ∈ Res contains XY) {
        Res = Res ∪ XY
    }
}
if (no schema S ∈ Res contains a candidate key for R) {
    K = any candidate key for R
    Res = Res ∪ K
}
```

... Third Normal Form

Critical step is producing minimal cover F_c for F

A set F of fds is minimal if

- every fd X → Y is simple
  (Y is a single attribute)
- every fd X → Y is left-reduced
  (no Z ⊂ X such that Z → A could replace X → A in F and preserve F^+)
- every fd X → Y is necessary
  (no X → Y can be removed without changing F^+)

Algorithm: right-reduce, left-reduce, eliminate redundant fds

---

Exercise: 3NF Decomposition (1)

54/55

Consider the schema R and set of fds F

R = ABCDEFGH

F = F_c = { ABH → C, A → D, C → E, F → A, E → F, BGH → E }

Produce a 3NF decomposition of R.

Database Design Methodology

55/55

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 normalized schema

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

Produced: 19 Sep 2012