NORMAL FORMS

CS121: Relational Databases

Fall 2018 – Lecture 18

Equivalent Schemas

- Many different schemas can represent a set of data
  - Which one is best?
  - What does “best” even mean?
- Main goals:
  - Representation must be complete
  - Data should not be unnecessarily redundant
  - Should be easy to manipulate the information
  - Should be easy to enforce [most] constraints

Normal Forms

- A "good" pattern for database schemas to follow is called a <u>normal form</u>
- Several different normal forms, with different constraints
- Normal forms can be formally specified
  - Can test a schema against a normal form
  - Can transform a schema into a normal form
- Goal:
  - Design schemas that satisfy a particular normal form
  - If a schema isn't "good," transform it into an appropriate normal form

Example Schema Design

- Schema for representing loans and borrowers:
  - *customer* relation stores customer details, including a *cust_id* primary-key attribute
  - *loan*(*<u>loan_id</u>*, *amount*)
  - *borrower*(*cust_id*, *loan_id*)
- Many-to-many mapping
  - A customer can have multiple loans
  - A loan can be owned by multiple customers

loan_id	amount
…	…
L-100	10000
…	…

loan

cust_id	loan_id
…	…
23-652	L-100
15-202	L-100
23-521	L-100
…	…

borrower

Larger Schema?

- Could replace *loan* and *borrower* relations with a larger, combined relation

  bor_loan(<u>*cust_id*</u>, <u>*loan_id*</u>, *amount*)

cust_id	loan_id	amount
…	…	…
23-652	L-100	10000
15-202	L-100	10000
23-521	L-100	10000
…	…	…

bor_loan

- Rationale:
  - Eliminates a join when retrieving loan amounts
- Problem: mapping between customers and loans is many-to-many
  - Multiple redundant copies of *amount* to keep in sync!

Repeated Values

- How do we *know* that this is a problem?
  - "Because we see values that appear multiple times"
  - *This isn't a good enough reason!!!*
  - Could easily have different loans with the same amount
- A repeated value doesn't *automatically* indicate a problem…

cust_id	loan_id	amount
…	…	…
23-652	L-100	10000
19-065	L-205	10000
15-202	L-100	10000
23-521	L-100	10000
20-419	L-205	10000
…	…	…

bor_loan

Back to the Enterprise

- What are the rules of the enterprise that we are modeling?
  - "Every loan must have only one amount."
- In other words:
  - Every loan ID corresponds to exactly one amount.
  - If there were a schema (*loan_id*, *amount*) then *loan_id* can be a primary key.
- Specified as a functional dependency
  - *loan_id* → *amount*
  - *loan_id* functionally determines *amount*

Repeated Values v2.0

- *bor_loan* relation has both *loan_id* and *amount* attributes

  bor_loan(<u>*cust_id*</u>, <u>*loan_id*</u>, *amount*)

- But, *loan_id* → *amount*, and *loan_id* by itself <u>can't</u> be a primary key in *bor_loan*
  - Need to support many-to-many mappings between customers and loans
  - Combination of *cust_id* and *loan_id* must be a primary key, so a particular *loan_id* value can appear multiple times
- In rows with the same *loan_id* value, *amount* will have to be repeated.

Functional Dependencies

- Functional dependencies are *very* important in schema analysis
  - Have a *lot* to do with keys!
  - "Good" schema designs are guided by functional dependencies
  - Frequently helpful to identify them during schema design
- Can formally define functional dependencies, and reason about them
- Can also specify constraints on schemas using functional dependencies

Another Example Schema

- A "large" schema for employee information

  employee(<u>emp_id</u>, emp_name, phone, title, salary, start_date)

emp_id	emp_name	phone	title	salary	start_date
…	…	…	…	…	…
123-45-6789	Jeff	555-1234	CTO	120000	1996-03-15
314-15-9265	Mary	555-3141	CFO	120000	1997-08-02
987-65-4321	Helen	555-9876	Developer	90000	1996-05-23
101-01-0101	Marcus	555-1010	Tester	70000	1995-11-04
…	…	…	…		…

employee

- Employee ID is unique, but other attributes could have duplicate values

Smaller Schemas?

- Could represent this with two smaller schemas:
  emp_ids(*<u>emp_id</u>*, *emp_name*)
  emp_details(*emp_name*, *phone*, *title*, *salary*, *start_date*)

emp_id	emp_name
…	…
123-45-6789	Jeff
314-15-9265	Mary
987-65-4321	Helen
101-01-0101	Marcus
…	…

emp_ids

emp_name	phone	title	salary	start_date
…	…	…	…	…
Jeff	555-1234	CTO	120000	1996-03-15
Mary	555-3141	CFO	120000	1997-08-02
Helen	555-9876	Developer	90000	1996-05-23
Marcus	555-1010	Tester	70000	1995-11-04
…	…	…		…

emp_details

- Generate original employee data with a join:
  emp_ids ⋈ *emp_details*
- Any problems with this?

emp_name is not unique!

- Joins using *emp_name* can generate invalid tuples!

emp_id	emp_name
…	…
314-15-9265	Mary
161-80-3398	Mary
…	…

emp_ids

emp_name	phone	title	salary	start_date
…	…	…	…	…
Mary	555-3141	CFO	120000	1997-08-02
Mary	555-1618	Gofer	25000	1998-01-07
…	…	…		…

emp_details

emp_ids ⋈ *emp_details*

emp_id	emp_name	phone	title	salary	start_date
…	…	…	…	…	…
314-15-9265	Mary	555-1618	Gofer	25000	1998-01-07
314-15-9265	Mary	555-3141	CFO	120000	1997-08-02
161-80-3398	Mary	555-3141	CFO	120000	1997-08-02
161-80-3398	Mary	555-1618	Gofer	25000	1998-01-07
…	…	…	…		…

Bad Decompositions

- This decomposition is clearly broken
  - It can't represent the information correctly!
- Problem: enterprise needs to support different employees with the same name
- <u>Lossy decompositions</u> cannot accurately represent all facts about an enterprise
- <u>Lossless decompositions</u> can accurately represent all facts
- "Good" schema designs avoid lossy decompositions

First Normal Form

- A schema is in <u>first normal form</u> (1NF) if all attribute domains are atomic
  - An atomic domain has values that are indivisible units
- E-R model supports non-atomic attributes
  - Multivalued attributes
  - Composite attributes
- Relational model specifies atomic domains for attributes
  - Schemas are automatically in 1NF
  - Mapping from E-R model to relational model changes composite/multivalued attributes into an atomic form

1NF Example

- E-R diagram for magazine subscribers
  - *address* is composite
  - *email_addr* is multivalued

subscriber

<u>sub_id</u>
{ email_addr }
address
    street
    city
    state
    zip_code

- Converts to a 1NF schema:

  subscriber(*<u>sub_id</u>*, *street*, *city*, *state*, *zip_code*)

  sub_emails(*<u>sub_id</u>*, *<u>email_addr</u>*)
  - The conversion rules we have discussed, *automatically* convert E-R schemas into 1NF

1NF and Non-Atomic Attributes

- Many, but not all, SQL DBs have non-atomic types
  - Some offer support for composite attributes
  - Some offer support for multivalued attributes
  - These are SQL extensions – not portable
- As long as you steer clear of using non-atomic attributes in primary/foreign keys, can sometimes be quite useful
  - Will likely encounter them very rarely in practice, though
  - Biggest reason: DB support for list/vector column-types isn't terribly widespread, or always very easy to use

1NF and Non-Atomic Attributes (2)

- Composite types:
  - e.g. defining an "address" composite type
  - Can definitely be useful for making a schema clearer, as long as they aren't used in a key!
- Multivalued types:
  - e.g. arrays, lists, sets, vectors
  - Can sometimes be useful for storing pre-computed values that aren't expected to change frequently
  - If you are regularly issuing queries that search through or change these values, you may need to revise your schema!
    - Should probably factor non-atomic data out into a separate table

Other Normal Forms

- Other normal forms relate to functional dependencies
- Analysis of functional dependencies shows if a schema needs decomposed
- Keys are functional dependencies too!
- Formally define functional dependencies, and reason about them
- Define normal forms in terms of functional dependencies

Schemas and Constraints

- Keys and functional dependencies are <u>constraints</u> that a database must satisfy
  - <u>Legal</u> relations satisfy the required constraints
  - Relation doesn't contain any tuples that violate the specified constraints
- More terminology:
  - Relation schema *R*, relation *r*(*R*)
  - A set of functional dependencies *F*
  - Relation *r* <u>satisfies</u> *F* if *r* is legal
  - When we say "*F* <u>holds</u> on *R*", specifies the set of relations with *R* as their schema, that are legal with respect to *F*

Functional Dependencies

- Formal definition of a functional dependency:
  - Given a relation schema R with attribute-sets $\alpha, \beta \subseteq R$
  - The functional dependency $\alpha \rightarrow \beta$ holds on $r(R)$ if
    $\langle\ \forall\ t_1, t_2 \in r : t_1[\alpha] = t_2[\alpha] : t_1[\beta] = t_2[\beta]\ \rangle$
- In other words:
  - For all pairs of tuples t_1 and t_2 in r,
    if $t_1[\alpha] = t_2[\alpha]$ then $t_1[\beta] = t_2[\beta]$
  - α functionally determines β

Dependencies and Superkeys

- Given relation schema R, a subset K of R can be a superkey
  - In a relation $r(R)$, no two tuples can share the same values for attributes in K
- Can also say: K is a superkey if $K \rightarrow R$
  - The functional dependency $K \rightarrow R$ holds if
    $\langle \forall\ t_1, t_2 \in r(R) : t_1[K] = t_2[K] : t_1[R] = t_2[R] \rangle$
  - $t_1[R] = t_2[R]$ (or $t_1 = t_2$) means t_1 and t_2 are the same tuple
  - The superkey K functionally determines the whole relation R
- Functional dependencies are a more general form of constraint than superkeys are.

The *bor_loan* Relation

- *bor_loan*(<u>*cust_id*</u>, <u>*loan_id*</u>, *amount*)
  - Functional dependency: *loan_id* → *amount*
  - "Every loan has exactly one amount."
  - Every tuple in *bor_loan* with a given *loan_id* value must have the same *amount* value
- *bor_loan* also has a primary key
  - Specifies another functional dependency
  - *cust_id*, *loan_id* → *cust_id*, *loan_id*, *amount*
  - This is not a functional dependency *specifically required* by what the enterprise needs to model
    - Can be inferred from other functional dependencies in the schema

Trivial Dependencies

- A trivial functional dependency is satisfied by *all* relation values!
  - For a relation R containing attributes A and B,
    $A \rightarrow A$ is a trivial dependency
    $\langle \forall\ t_1, t_2 \in r : t_1[A] = t_2[A] : t_1[A] = t_2[A] \rangle$
    - Well, duh!
  - $AB \rightarrow A$ is also a trivial dependency
    - If $t_1[AB] = t_2[AB]$, then of course $t_1[A] = t_2[A]$ too!
- In general: $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Closure

- Given a set of functional dependencies, we can infer other dependencies
  - Given relation schema $R(A, B, C)$
  - If $A \rightarrow B$ and $B \rightarrow C$, holds on R, then $A \rightarrow C$ also holds on R
- Given a set of functional dependencies F
  - F^+ denotes the closure of F
  - F^+ includes F, and all dependencies that can be inferred from F. ($F \subseteq F^+$)

Boyce-Codd Normal Form

- Eliminates all redundancy that can be discovered using functional dependencies
- Given:
  - Relation schema *R*
  - Set of functional dependencies *F*
- *R* is in BCNF with respect to *F* if:
  - For all functional dependencies $\alpha \rightarrow \beta$ in F^+, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
    - $\alpha \rightarrow \beta$ is a trivial dependency
    - α is a superkey for *R*
- A database design is in BCNF if all schemas in the design are in BCNF

BCNF Examples

- The *bor_loan* schema isn't in BCNF

  bor_loan(<u>*cust_id*</u>, <u>*loan_id*</u>, *amount*)

  - *loan_id* → *amount* holds on *bor_loan*
  - This is not a trivial dependency, and *loan_id* isn't a superkey for *bor_loan*

- The *borrower* and *loan* schemas <u>are</u> in BCNF

  borrower(<u>*cust_id*</u>, <u>*loan_id*</u>)

  - No nontrivial dependencies hold

  loan(<u>*loan_id*</u>, *amount*)

  - *loan_id* → *amount* holds on *loan*
  - *loan_id* is the primary key of *loan*

BCNF Decomposition

- If *R* is a schema not in BCNF:
  - There is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for *R*
- Replace *R* with two schemas:

  $(\alpha \cup \beta)$

  $(R - (\beta - \alpha))$

    - (stated this way in case α and β overlap; usually they don't)
- *The new schemas might also not be in BCNF!*
  - Repeat this decomposition process until all schemas are in BCNF

Undoing the Damage

- For *bor_loan*, α = *loan_id*, β = *amount*

  R = (*cust_id*, *loan_id*, *amount*)

  ($\alpha \cup \beta$) = (*loan_id*, *amount*)

  (*R* – (β – α)) = (*cust_id*, *loan_id*)
- Rules successfully decompose *bor_loan* back into *loan* and *borrower* schemas

Review

- Normal forms are guidelines for what makes a database design "good"
  - Can formally specify them
  - Can transform schemas into normal forms
- Functional dependencies specify constraints between attributes in a schema
  - A more general kind of constraint than key constraints
- Covered 1NF and BCNF
  - 1NF requires all attributes to be atomic
  - BCNF uses functional dependencies to eliminate redundant data

Next Time!

- A big question to explore:
  - Given a set of functional dependencies *F*, we need to know what dependencies can be inferred from it!
    - i.e. given *F*, how to compute F^+
  - BCNF needs this information, as do other normal forms
- Does Boyce-Codd Normal Form have drawbacks?
  - (yes.)
  - Motivates the development of 3rd Normal Form