FDs and Normal Forms

CISC637, Lecture #10

Ben Carterette

Database Design

- Database design is the process of going from *requirements* to *relational schema*
  - Schema should capture as many requirements as possible
  - Should also avoid redundancy and data inconsistency
  - Should also enable efficient storage and querying
  - These goals are not always mutually attainable
- There are many possible valid schema
  - One giant table that stores all data: valid design that can capture all requirements, but will have tons of redundancy
  - …
  - Lots of small tables that store bits of data: valid design that has no redundancy but may not capture all requirements
- FDs and normal forms are tools for evaluating a possible design

Design Example #1

- Change to university requirements:
  - Each instructor is an advisor for at most one department
  - Students can have majors in more than one department
  - Students can have multiple advisors, but at most one advisor per department
- FDs to capture these requirements:
  - i_ID → dept_name
    - given instructor ID, there is at most one possible department they advise for
  - s_ID, dept_name → i_ID
    - given student ID and department, there is at most one possible instructor advising

Possible Relational Schema

- Possible relational schema #1:
  - Student(<u>s_ID</u>, name, tot_cred)
  - Instructor(<u>i_ID</u>, name, salary, dept_name)
  - DeptAdvisor(<u>s_ID</u>, <u>dept_name</u>, i_ID)
- Possible relational schema #2:
  - Student(<u>s_ID</u>, name, tot_cred)
  - StudentDept(<u>s_ID</u>, <u>dept_name</u>)
  - Instructor(<u>i_ID</u>, name, salary, dept_name)
  - Advisor(<u>s_ID</u>, <u>i_ID</u>)
- Both have pros and cons

Design Example #2

- New university requirement
  - All students must have exactly one advisor
  - All instructors must advise exactly one student
  - Student and instructor must be in the same department
- FDs:
  - s_ID → name, tot_cred
  - i_ID → name, salary
  - s_ID → i_ID
  - i_ID → s_ID
  - s_ID, i_ID → dept
- Two possible schema:
  - StudentInstructor(s_ID, s_name, tot_cred, i_ID, i_name, salary, dept)
  - StudentInstructor(i_ID, i_name, salary, s_ID, s_name, tot_cred, dept)
  - 3NF? Yes to both! BCNF? Yes to both! Good design? Probably not.

FDs and Normal Forms

- FDs support a sort of logical system
  - Each one is a sentence that needs to be true on all possible database instances
    - (they can be checked for truth in the data)
  - There are *rules of inference* that can be applied to find FDs other than the ones specifically derived from requirements
- The system is **closed**
  - FDs inferred using rules are true, even if not specified
  - All FDs that are true can be derived using the rules
- The **closure** of a set of FDs is the set of all FDs that can be found by recursively applying inference rules
  - We will use F to refer to a given set of FDs
  - F^+ to refer to its closure

Transitivity

- If A → B and B → C, then A → C
- Example:
  - New university requirement
    - All students must have exactly one advisor
    - All instructors must advise exactly one student
    - Student and instructor in same department
  - FDs:
    1. s_ID → student name, tot_cred
    2. i_ID → instructor name, salary
    3. s_ID → i_ID
    4. i_ID → s_ID
    5. s_ID, i_ID → dept

---

6. #3 and #2: s_ID → i_ID, inst name, salary
7. #4 and #1: i_ID → s_ID, student name, tot_cred
8. #6 and #7: s_ID → i_ID, inst name, salary, student name, tot_cred

7

Augmentation

- If A → B, then {A, C} → {B, C}
- Example:
  - FDs:
    1. s_ID → student name, tot_cred
    2. i_ID → instructor name, salary
    3. s_ID → i_ID
    4. i_ID → s_ID
    5. s_ID, i_ID → dept

---

6. augment #1: s_ID, i_ID → student name, tot_cred, i_ID
7. augment #2: i_ID, s_ID → instructor name, salary s_ID

8

Reflexivity & Armstrong's Axioms

- If A is a set of attributes and B is a subset of A, then A → B
  - Trivial FDs
  - But a specific rule is necessary for the inference rules to be sound and complete

- Armstrong's axioms:
  1. Reflexivity
  2. Augmentation
  3. Transitivity

  - Applying these rules recursively to a set F will:
    - give every possible FD that is true (completeness)
    - will never give an FD that is not true (soundness)

Armstrong's Axioms

- Armstrong's axioms:
  1. Reflexivity

     If A is a set of fields and B is a subset of A, then A → B
  2. Augmentation

     If A → B, then {A, C} → {B, C}
  3. Transitivity

     If A → B and B → C, then A → C

  - Applying these rules recursively to a set of FDs will:
    - give every possible FD that is true (completeness)
    - will never give an FD that is not true (soundness)

Union

- If A → B and A → C, then A → {B, C}
  - Not one of Armstrong's rules, but follows from them
- Example:
  - FDs:
    1. s_ID → student name, tot_cred
    2. i_ID → instructor name, salary
    3. s_ID → i_ID
    4. i_ID → s_ID
    5. s_ID, i_ID → dept

    ---

    6. union of #1 and #3: s_ID → student name, tot_cred, i_ID
    7. union of #2 and #4: i_ID → inst name, salary, s_ID

Decomposition

- If A → {B, C}, then A → B and A → C
  - Also not one of Armstrong's rules
- Example:
  - FDs:
    1. s_ID → student name, tot_cred
    2. i_ID → instructor name, salary
    3. s_ID → i_ID
    4. i_ID → s_ID
    5. s_ID, i_ID → dept

    ---

    6. decompose #1: s_ID → student name, s_ID → tot_cred

FD Closure Example

- A *contract* is an agreement that a *supplier* will supply *qty* items of a certain *part* to a certain *project* associated with a certain *dept*
  - Contracts are identified by a unique ID
  - When a part is needed for a project, the entire quantity of that part is purchased with a single contract
  - A department purchases at most one part from a supplier
- F = the set of FDs from requirements:
  - contract → supplier, project, dept, part, qty, value
  - project, part → contract
  - supplier, dept → part
- Schema (based on first candidate key FD):
  - Contracts(<u>contract</u>, supplier, project, dept, part, qty, value)
- Apply Armstrong's rules to get F^+

BCNF/3NF Definitions Redux

- A relation R is in BCNF iff, for all X → A in F^+,
  - A $\subseteq$ X (a trivial FD), or
  - X is a superkey for R
- A relation R is in 3NF iff, for all X → A in F^+,
  - A $\subseteq$ X (a trivial FD), or
  - X is a superkey for R, or
  - A is part of some candidate key for R

Attribute Closure

- Armstrong's inference rules are computationally inefficient
  - We could be applying them for a long time before we've completed F^+
- But we usually don't really care much about F^+ itself
  - What we care about is:
    - Is there some FD that is not a key? (violates BCNF)
    - Is there some FD that has fields on the right that are not part of any candidate key? (violates 3NF)
- The **attribute closure** of a field A is the set of all fields that are functionally determined by A after recursive inference
  - If A → all fields, then A is a superkey
  - If every field produces a superkey (or nothing), then BCNF

Attribute Closure

- An efficient algorithm exists to compute attribute closure

given a field (or set of fields) A and a set of functional dependencies F:

```
let A⁺ = A
do:
  for each FD f ∈ F:
    let B = left side of f
    let C = right side of f
    if B ⊆ A⁺:
      let A⁺ = A⁺ ∪ C
until A⁺ is unchanged
```


Student <= Advisor => Instructor

- FDs:
  1. s_ID → student name, tot_cred
  2. i_ID → instructor name, salary
  3. s_ID → i_ID
  4. i_ID → s_ID
  5. s_ID, i_ID → dept
- Possible schema:
  - StudentInstructor(<u>sid</u>, sname, tot_cred, iid, iname, salary, dept)
  - Is this in BCNF?
- Is s_ID a superkey on StudentInstructor?
  - Start by putting s_ID in A^+
  - For each FD, if left side is in current version of A^+, put right side in A^+
    - After first loop over FDs, A^+ = {s_ID, student name, tot_cred, i_ID, dept}
    - After second loop, A^+ = {s_ID, student name, tot_cred, i_ID, dept, instructor name, salary}
  - Since A^+ contains all fields, s_ID is a superkey
  - Therefore FDs 1 and 3 do not violate BCNF

Contracts Example

- FDs:
  - contract → supplier, project, dept, part, qty, value
  - project, part → contract
  - supplier, dept → part
- Is {project, part} a superkey?
  - Yes
- Is {supplier, dept} a superkey?
  - No—violates BCNF
  - Verify using attribute closure algorithm
- Candidate key {supplier, dept, project} cannot be found using attribute closure
  - But it can be verified to be a candidate key

Student <= Advisor => Instructor

- FDs:
  1. s_ID → student name, tot_cred
  2. i_ID → instructor name, salary
  3. s_ID → i_ID
  4. i_ID → s_ID
  5. s_ID, i_ID → dept

- Alternate possible schema:
  – Student(s_ID, s_name, tot_cred)
  – Instructor(i_ID, i_name, salary)
  – Advisor(s_ID, i_ID, dept)
  – Are these in BCNF?
    - Yes: s_ID (from FD 1) is a superkey for Student
      i_ID (from FD 2) is a superkey for Instructor
      {s_ID, i_ID} (from FD 5) is a superkey for Advisor
  – Only problem is that we have to do extra work to make sure every instructor is an advisor and every student has an advisee
    - It happened automatically in the previous design

Formal Database Design Process

- Write out requirements as clearly as possible
- Draw an E-R diagram showing all important entities and relationships
- Use requirements and E-R diagram to list functional dependencies
- Convert E-R diagram to relational schema, preserving as many FDs as possible by keeping related attributes together
- Use attribute closure to test each relation for BCNF
  – If a relation is not in BCNF, either decompose it into BCNF relations …
  – … or test for 3NF