CS 133: Databases

Fall 2018
Lec 21 – 11/20
Database Design
Prof. Beth Trushkowsky

Goals for Today

• Understand the issues that can occur from a poorly designed relational schema

• Learn about the goals and process of schema refinement

• Discuss the role of functional dependencies in discovering and fixing issues in a design

Review: Database Design

• Requirements Analysis
  – user needs; what must database do?

• Conceptual Design
  – high level description (often done w/ER model)

• Logical Design
  – translate ER into DBMS data model

• Schema Refinement
  – consistency, normalization

• Physical Design - indexes, disk layout

• Security Design - who accesses what

Participation Constraints in SQL

• We can capture participation constraints involving one entity set in a binary relationship, but little else

CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11) NOT NULL,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  ON DELETE NO ACTION)
Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```sql
CREATE TABLE Dep_Policy (
    pname CHAR(20),
    age INTEGER,
    cost REAL,
    ssn CHAR(11) NOT NULL,
    PRIMARY KEY (pname, ssn),
    FOREIGN KEY (ssn) REFERENCES Employees,  
    ON DELETE CASCADE
)
```

ISA Hierarchies to Relations

- **Three relations**
  Employees(ssn, name, lot)
  Contract_emps(ssn, contractid)
  Hourly_Emps(ssn, hourly_wages, rating, hours_worked)

- **Alternative (assuming covering)**
  Contract_emps(ssn, name, lot, contractid)
  Hourly_Emps(ssn, name, lot, hourly_wages, rating, hours_worked)

Table and Column constraints (SQLite)

- **Column definition**
  - `column-name` type-name column-definition

- **Table constraint**
  - PRIMARY KEY column-definition
  - UNIQUE
  - CHECK `expr` column-definition
  - FOREIGN KEY `column-name` foreign-key-clause

https://www.sqlite.org/lang_createtable.html
Examples: Create Table

```sql
CREATE TABLE Friends (  
    friend1    VARCHAR(40),  
    friend2    VARCHAR(40),  
    PRIMARY KEY(friend1,friend2),  
    CONSTRAINT notSame CHECK (friend1 <> friend2)
);
```

```sql
CREATE TABLE Bar_Owns(  
    name       VARCHAR(40) PRIMARY KEY,  
    address    VARCHAR(40) NOT NULL,  
    phone      CHAR(12) DEFAULT "555-555-5555",  
    owner      CHAR(11) NOT NULL,  
    FOREIGN KEY owner REFERENCES Drinkers(ssn)  
        ON DELETE NO ACTION
);
```

Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS

Example, SQL:1999 syntax:

```sql
CREATE TRIGGER youngSailorUpdate
AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
    INSERT INTO YoungSailors(sid, name, age, rating)
        SELECT sid, name, age, rating
    FROM NewSailors N
    WHERE N.age <= 18
```

Combos: Entities and Relationships

- For one-to-many relationship, combining entity set and relationship set into one relation helped us capture participation constraint

- What about combining Bars and Sells as Bar_Sells?

```
Bars --| address    type   |-- Beers
     | bar_name    |     |
     |             |     |
Sells --| price  | beer_name|
```

Schema Refinement

- Start with initial relational schema, either from scratch or from E/R modeling

- Schema refinement objective: could there be issues caused by data redundancy?

- Next: why redundancy is “bad”
Example: Hourly_Emps

- Consider a relation obtained from Hourly_Emps:
  Hourly_Emps (ssn, name, lot, rating, wage_per_hr, hrs_per_wk)

  Note on notation: can denote a relation schema by listing its attributes, e.g., SNLRWH
  \( \rightarrow \) the set of attributes \( \{S, N, L, R, W, H\} \)

- Assume we know, from application semantics:
  - ssn uniquely identifies an employee (is a key)
  - An employee’s rating determines their wage_per_hr

Decomposing a Relation

- Redundancy can be removed by “chopping” the relation into pieces.

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

We’ll see how a type of integrity constraint, called functional dependencies, is used to drive this “chopping” process

Redundancy Problems

Hourly_Emps (instance)

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

- Update anomaly: What if we change W in this tuple only?
- Deletion anomaly: What if we delete all employees with rating 5?
- Insertion anomaly: What if we want to insert an employee and don’t know the hourly wage their rating? (or we get it wrong?)

Taming Schema Redundancy

- Integrity constraints, in particular functional dependencies, can be used to identify schemas with problems and to suggest refinements

- Main refinement technique: decomposition
  - E.g., replacing ABCD (via projection) with:
    - AB and BCD, or
    - ACD and ABD, or
    - Etc.

- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?
Keys (Review)

- A set of fields is a candidate key (shortened as just key) for a relation if:
  1. No two distinct tuples can have the same values in all candidate key fields, and
  2. This is not true for any subset of the key’s attributes.

Q. Consider relation $R(a, b, c)$.
For a fixed setting of $a$ and $b$ values, how many different $c$ values could there be?

- A candidate key is minimal.
  If $AB$ is a candidate key, then neither $A$ nor $B$ is a key on its own.

- A superkey is not necessarily minimal (although it could be)
  - If $AB$ is a candidate key, then $ABC$, $ABD$, and even $AB$ are superkeys.

Functional Dependencies (cntd)

- Where do FDs come from?
  - Real-world integrity constraints and semantics

- Keys redefined as FDs with set of attributes $K$ and relation $R$:
  1. if $K \rightarrow$ all (other) attributes of $R$
     $K$ is a “super key”
  2. And if no proper subset of $K$ satisfies the above condition, then
     $K$ is minimal (and thus a candidate key)

Functional Dependencies

- Let $X$ and $Y$ be sets of attributes in a relation $R$
- A functional dependency (FD) has the form $X \rightarrow Y$

- If two tuples in $R$ have same values for all attributes in $X$, then they must also have same values for all attributes in $Y$

  ![Diagram showing two tuples $t1$ and $t2$ and their corresponding attributes.
  - $t1$ and $t2$ have the same values for $X$.
  - $t1$ and $t2$ have the same values for $Y$.
  - Can read $\rightarrow$ as “determines”]

Exercise 3: Constructing FDs

- What functional dependencies do you think would make sense for this application?

  ![Database schema diagram:
  - Bars
  - Sells
  - Beers
  - Relationships:
    - Bar_name $\rightarrow$ address
    - beer_name $\rightarrow$ type
    - bar_name, beer_name $\rightarrow$ price]
Reasoning About FDs

- Given some FDs, can usually infer additional FDs that are true.
  
  E.g., College use case:

  - `profId → profName` and `profId → dept`
  
  implies:

  - `profId → profName, dept`

- However, `building, roomNum → profName` does NOT imply `building → profName` or `roomNum → profName`.

- A particular FD `f` is implied by a set of FDs `F` if `f` holds whenever all FDs in `F` hold.

Closure and Rules of Inference

- `F^+` = closure of `F` is the set of all FDs that are implied by `F` (includes "trivial dependencies": RHS ⊆ LHS).

- Armstrong’s Axioms (X, Y, Z are sets of attributes):
  - Reflexivity: If `Y ⊆ X`, then `X → Y`.
  - Augmentation: If `X → Y`, then `XZ → YZ` for any `Z`.
  - Transitivity: If `X → Y` and `Y → Z`, then `X → Z`.

- Some additional rules (that follow from AA):
  - Union: If `X → Y` and `X → Z`, then `X → YZ`.
  - Decomposition: If `X → YZ`, then `X → Y` and `X → Z`.

Example: Using Inference Rules

- Suppose relation `R` has three attributes `A, B, C` and these FDs:
  - `A → B`  
  - `B → C`  

- Using reflexivity
  - `A → A, AB → A, etc.`

- Using transitivity
  - `A → C`  

- Using augmentation
  - `AC → BC, AB → AC, AB → BC`  

Attribute Closure

- If we just want to check if a given FD `X → Y` is in `F^+`, then:

  1) Compute the attribute closure of `X` (denoted `X^+`) with respect to `F`:
     - `X^+` = Set of all attributes `A` such that `X → A` is in `F^+`.
       - initialize `X^+ := X`  
       - Repeat until no change to `X^+`:
         - if `U → V` in `F` such that `U` is in `X^+`, then add `V` to `X^+`.

  2) Check if `Y` is in `X^+`.

Q. How can attribute closure be used to determine if a set of attributes is a key for a relation?
Exercise 4

- **Contracts**\((cid,sid,jid,did,pid,qty,value)\), and:
  - C is the primary key: \(C \rightarrow CSJDPQV\)
  - Project purchases each part using single contract: \(JP \rightarrow C\)
  - Dept purchases at most 1 part from a supplier: \(SD \rightarrow P\)

- **Show that SDJ is a superkey for Contracts**
  - \(JP \rightarrow C, C \rightarrow CSJDPQV\) \(\implies\) \(JP \rightarrow CSJDPQV\) (by transitivity) (shows that JP is a superkey)
  - \(SD \rightarrow P\) \(\implies\) \(SDJ \rightarrow JP\) (by augmentation)
  - \(SDJ \rightarrow JP, JP \rightarrow CSJDPQV\) \(\implies\) \(SDJ \rightarrow CSJDPQV\) (by transitivity) thus SDJ is a superkey