CS122D: Beyond SQL Data Management — Lecture #3 —

Mike Carey
UC Irvine
mjcarey@ics.uci.edu
Announcements

• HW1 will be available late this evening!
  • Get PostgreSQL installed quickly (try it w/Hoofers data)
  • You’ll be wrangling ShopALot hypothetical business data
  • Do HW individually and compare w/brainstorming buddy!
  • Goal is to refresh/enhance everyone’s “SQL query chops”

• Be sure to keep watching the wiki and Piazza pages
  • http://www.ics.uci.edu/~cs122d
  • http://piazza.com/uci/spring2021/cs122d/home

• Quizzes (from discussions) will close on Fridays at 8PM

• Today:  *Conceptual Data Modeling (E-R)*
  • The third (of four) “preparatory” (background) lectures
A Brief Acknowledgement

• Several of the NoSQL database vendors have been kind enough to provide us with access to their training materials

• I will be reusing some of that material (especially their nice professional graphics) for use in our course slides

• So: Today I want to acknowledge DataStax (Cassandra)!
  • When I took their online course “DS220: Cassandra Data Modeling Methodology” I was both pleased and amazed to discover that they favor many of the the same E-R diagramming conventions that I use when teaching CS122A
  • Today’s data modeling examples will thus be drawn from their DS220 materials, hopefully lowering the risk of boredom for any alumni of my CS122A (😊)
Database Design Process (CS122A)

1. Requirements Gathering (interviews)
2. Conceptual Design (using E-R model)
3. Platform Choice (which DBMS?)
4. Logical Design (for target data model)
5. Physical Design (for target DBMS, workload)
6. Implement (and test, of course 😊)

(Note: Expect backtracking, iteration, and other incremental adjustments.)
E-R Modeling Process

- What are the **entities** and **relationships** in the application at hand?
- What information about them – **attributes** – do we want/need to have in the application’s database?
- What other integrity constraints and/or business rules should be considered/enforced?

➢ Use an E-R diagram to document (and discuss) the answers!
Entities and Entity Sets

- An entity is a real-world object, distinguishable from other objects, from the viewpoint of the target application, and is described by a set of attributes
  - E.g., Video and Actor are entities in the diagram above
- An entity set is a collection of similar (!) entities, e.g., all Videos
  - Each entity set has a key attribute (singular or composite), e.g., Video.id
  - Each entity’s key value will be different (unique!) within a given entity set
- Each attribute has a domain (~data type) that needs to be noted somewhere
  - E.g., Video.description and Video.title are strings
A relationship is an association between two (or more) entities, and a relationship may also have attributes:

- E.g., features (a.k.a. appears in) is a relationship in the diagram above.
- (Notice how DataStax handles a common relationship-naming dilemma! 😊)

A relationship set is a collection of similar (!) relationships.

A given entity set can participate in different relationship sets:

- May even have different roles in the same relationship set, e.g., manager vs. managee for employees in a manager relationship set for an HR application.

We may be sloppy sometimes in our use of the words entity and relationship (vs. entity set and relationship set)...

M. Carey, Spring 2021: CS122D
Cardinality Constraints

- Relationship sets’ connections to the related entity sets are labeled with **cardinality constraints** that specify important “how many” information
  - The relationship in the diagram above is labeled as m-to-n (“many-to-many”)
  - In this example, a given Video entity can be related to (*i.e.*, "feature") multiple Actors
  - Also, a given Actor entity can be related to (*i.e.*, “appear in”) multiple Videos
- These constraints come from the “world”, *i.e.*, they are inherent in the application domain the application is supposed to manage data for
  - Other possibilities above would have been be 1-to-n, m-to-1, and 1-to-1
  - **Q:** What would be the meanings for the “world” for other cardinality options?
Signs of Weakness

• A *weak entity* is an entity that is not allowed to exist in the database without being connected through an identifying relationship (*a.k.a.* a weak relationship) to a strong entity

• For example:
IsA ("is a") Hierarchies

- Entities can be related via an IsA hierarchy to model entity specialization, e.g., for Videos

- IsA’s annotations indicate:
  - Whether or not the subtypes (think “subentity sets”) are disjoint from one another
  - Whether or not the subtypes are covering for the supertype

**Q:** Which entities have which attributes? (Trailer? Movie?)
More Complex Attributes

- A composite attribute has further internal (record-like) structure
  - Essentially think “attributes can have attributes”
  - E.g., Actor.name in the diagram above

- A multivalued attribute can hold multiple values (non-1NF)
  - E.g., Video.tags or Video.genres in the diagram above
Full Conceptual Video Schema
Full Conceptual Video Schema (I)
Full Conceptual Video Schema (II)
Full Conceptual Video Schema (III)
Logical Design for SQL (I)

- A simple **entity set** can be represented by a corresponding SQL table

```
CREATE TABLE User ( id INTEGER, password VARCHAR, email VARCHAR, registration_date DATETIME, name_first VARCHAR, name_last VARCHAR, PRIMARY KEY (id) )
```

(Also notice the **composite attributes**; would name-qualify them in general, *e.g.*, User.address.zipcode.)
Logical Design for SQL (II)

- An entity set with set-valued attributes will need to be represented by several SQL tables (to be 1NF-compliant)

```
CREATE TABLE Video ( id INTEGER, title VARCHAR, description VARCHAR, url VARCHAR, ... , PRIMARY KEY (id) )
CREATE TABLE Video_tags ( video_id INTEGER, tag VARCHAR, ... , PRIMARY KEY (video_id, tag), FOREIGN KEY (video_id) REFERENCES Video )
CREATE TABLE Video_genres ( video_id INTEGER, genre VARCHAR, ... , PRIMARY KEY (video_id, genre) , FOREIGN KEY (video_id) REFERENCES Video )
```
An **m-to-n relationship set** can be represented by a SQL table with an appropriately chosen set of primary (and foreign) key columns.

```sql
CREATE TABLE Appears_in (
  video_id INTEGER,
  actor_first_name VARCHAR,
  actor_last_name VARCHAR,
  character_name VARCHAR,
  PRIMARY KEY (video_id, actor_first_name, actor_last_name),
  FOREIGN KEY (video_id) REFERENCES Video,
  FOREIGN KEY (actor_first_name, actor_last_name) REFERENCES Actor )
```

(Also notice the presence and treatment of the **relationship attribute** in this example.)
Logical Design for SQL (IV)

• An 1-to-n relationship set can be “optimized” – if desired – by combining the two SQL tables that would otherwise have the same primary key

```
CREATE TABLE Actor (actor_first_name VARCHAR, actor_last_name VARCHAR,
    character_name VARCHAR, ... , appears_in_video_id INTEGER,
    PRIMARY KEY (actor_first_name, actor_last_name),
    FOREIGN KEY (appears_in_video_id ) REFERENCES Video )
```

(Again, also pay attention to the treatment of the relationship attribute in this example.)
Logical Design for SQL (V)

- An **weak entity** with its **weak relationship** can be handled similarly, but the weak entity table’s key must include the key of the strong entity as well.

```sql
CREATE TABLE Encoding (
    video_id INTEGER, height INTEGER, width INTEGER, encoding VARCHAR, ...
    PRIMARY KEY (video_id, encoding),
    FOREIGN KEY (video_id) REFERENCES Video )
```

(Again, pay careful attention to the treatment of keys in this example.)
Logical Design for SQL (VI)
Logical Design for SQL (VI)

• An IsA hierarchy in an E-R schema can be “tabularized” for SQL in one of at least three ways:
  • Delta tables
    • Root table defines the PK and the top-level fields
    • “Sub-tables” have the same PK (also an FK to its parent) and the newly-appearing fields
    • Join queries are needed to materialize all instances of a non-root entity set
  • Mashup table
    • The “one table to rule them all” – has all fields from everywhere in the entity hierarchy
    • Must also tag rows with information about their particular entity type(s)
  • Union tables
    • Every table has the same PK plus all fields (newly-appearing and inherited) for its entity type
    • Each entity in the hierarchy therefore lives (fully) in a table unique to its entity type/subtype
    • Union-all queries are needed to materialize all instances of a non-leaf entity type
    • (Comes up short w.r.t. PK enforcement)

(We’ll talk later on in the quarter about NoSQL IsA mapping options...)
• Delta tables

CREATE TABLE Video ( id INTEGER PRIMARY KEY, title VARCHAR, description VARCHAR, ... )
CREATE TABLE Full_Video ( id INTEGER PRIMARY KEY, FOREIGN KEY (id) REFERENCES (Video) )
CREATE TABLE Trailer ( id INTEGER PRIMARY KEY, trailer_for INTEGER, FOREIGN KEY (id) REFERENCES (Video), FOREIGN KEY (trailer_for) REFERENCES (Full_Video) )
CREATE TABLE TV_Show ( id INTEGER PRIMARY KEY, season INTEGER, episode INTEGER, FOREIGN KEY (id) REFERENCES (Full_Video) )
CREATE TABLE Movie ( id INTEGER PRIMARY KEY, rating VARCHAR, FOREIGN KEY (id) REFERENCES (Full_Video) )

• Mashup table


CREATE TABLE Video ( id INTEGER PRIMARY KEY, title VARCHAR, description VARCHAR, ... , trailer_for INTEGER, season INTEGER, episode INTEGER, rating VARCHAR, kind VIDEO_KIND, FOREIGN KEY (trailer_for) REFERENCES (Video) )

• Union tables

CREATE TABLE Full_Video ( id INTEGER PRIMARY KEY, title VARCHAR, description VARCHAR, ... )
... 
CREATE TABLE Movie ( id INTEGER PRIMARY KEY, title VARCHAR, description VARCHAR, ... , rating VARCHAR)
Physical Design for SQL

- Create indexes to support the application’s anticipated query needs!

```
CREATE TABLE User ( id INTEGER, password VARCHAR, email VARCHAR, registration_date DATETIME, name_first VARCHAR, name_last VARCHAR, PRIMARY KEY (id) )

CREATE INDEX User_email_idx ON User (email)
CREATE INDEX User_reg_date_idx ON User (registration_date)
...
```
Questions...?