Introduction to Data Management

Database Design

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle
Announcements

- HW 3 out

- Azure credits have been issued ($75)
  - Go to this link (posted on message board too):
    - https://portal.azure.com/#blade/Microsoft_Azure_EducationMenuBlade/overview
    - Enter your @uw.edu email in sign-in

- Make sure the setup is working by yesterday

- HW 1 grades will be back shortly
- HW 2 grades back by Friday
Recap – Relational Model

- SQL is parsed by the DBMS and translated into an RA plan that is more directly executable.
- Both query types work on the assumption that you are using relational data.
- The relational model specifies mechanics of how data can be organized:
  - No prescription of how data should be organized.
Goals for Today

- With some application in mind, we can use an entity relationship (ER) diagram to conceptualize and communicate.
- And with an ER diagram, we can use SQL to realize the model.
Outline

- Introduce Database Design
- ER Diagrams
- ER-to-SQL conversion along the way
- Integrity constraints along the way
Database Design

- Communication is Key
- Other people are involved in the design process
- Non-computer scientists have to interact with the data too
Database Design

**Database Design** or **Logical Design** or **Relational Schema Design** is the process of organizing data into a database model. This is done by considering *what data needs to be stored* and the *interrelationship of the data*. 

[Diagram showing the process from arbitrary data to a database model]
The Database Design Process

Conceptual Model

Relational Model
→ + Schema
→ + Constraints

Conceptual Schema
→ + Normalization

Physical Schema
→ + Partitioning
→ + Indexing
The Database Design Process

Conceptual Model

Relational Model
→ + Schema
→ + Constraints

Conceptual Schema
→ + Normalization

Physical Schema
→ + Partitioning
→ + Indexing
The Database Design Process

Conceptual Model

Relational Model
→ + Schema
→ + Constraints

Conceptual Schema
→ + Normalization

Physical Schema
→ + Partitioning
→ + Indexing
The Database Design Process

Conceptual Model

Relational Model
→ + Schema
→ + Constraints

Conceptual Schema
→ + Normalization

Physical Schema
→ + Partitioning
→ + Indexing

Next Unit
Humans are visual creatures so a visual model serves us best.
These are all the blocks we will learn about

<table>
<thead>
<tr>
<th>Entity set</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Entity set" /></td>
<td><img src="image2" alt="Attribute" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Relationship" /></td>
<td><img src="image4" alt="Subclass" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weak Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5" alt="Weak Entity" /></td>
</tr>
</tbody>
</table>
Entity Sets

- An “entity set” is like a class
- An attribute is like a field
- An “entity” is like a object
  - Corresponds to a row
Entity Sets

- An “entity set” is like a class
- An attribute is like a field
- An “entity” is like a object
  - Corresponds to a row

Underline indicates the attribute is part of the primary key
**Entity Sets**

- An "entity set" is like a **class**
- An **attribute** is like a **field**
- An "**entity**" is like a **object**
  - Corresponds to a row

---

![ER Diagram for Person]

- Underline indicates the attribute is part of the primary key
- Every entity set should have a primary key
Entity Sets

- An “entity set” is like a class
- An attribute is like a field
- An “entity” is like a object
  - Corresponds to a row

```
CREATE TABLE Person (  
  ssn INT PRIMARY KEY,  
  name TEXT,  
  address TEXT);  
```
If $A$ and $B$ are sets, then a relation $R$ is a subset of $A \times B$. 

\[ R \subseteq A \times B \]
**Relationship**

If $A$ and $B$ are sets, then a relation $R$ is a subset of $A \times B$

---

**Diagram:**

- **Product**:
  - Beyblade, ...
  - Trolls, ...

- **Company**:
  - Hasbro, ...
  - Nyform, ...

**Relationships**:

- **Product** connects to **Company** via the "makes" relation.
- **Product** has attributes: price, name.
- **Company** has attributes: name, address, ceo.
If $A$ and $B$ are sets, then a relation $R$ is a subset of $A \times B$.
Relation Multiplicity

- **One-to-one**
- **Many-to-one**
- **Many-to-many**

July 15, 2019
Relation Multiplicity

- **One-to-one**
- **Many-to-one**
- **Many-to-many**

Product
- Beyblade, ...
- Trolls, ...

Company
- Hasbro, ...
- Nyform, ...

Product makes Company
Relation Multiplicity

- **One-to-one**

- **Many-to-one**

- **Many-to-many**

CREATE TABLE Product (  
    name VARCHAR(100) PRIMARY KEY,  
    ...);  
CREATE TABLE Company (  
    name VARCHAR(100) PRIMARY KEY,  
    ...);  
CREATE TABLE Makes (  
    cname VARCHAR(100) UNIQUE REFERENCES Company,  
    pname VARCHAR(100) UNIQUE REFERENCES Product,  
    ...);
Relation Multiplicity

- One-to-one
- Many-to-one
- Many-to-many

Product | Company
---|---
Beyblade, … | Hasbro, …
Trolls, … | Nyform, …

Product makes Company
Relation Multiplicity

- One-to-one
- Many-to-one
- Many-to-many

CREATE TABLE Product (  
    name VARCHAR(100) PRIMARY KEY,  
    ...);  
CREATE TABLE Company (  
    name VARCHAR(100) PRIMARY KEY,  
    ...);  
CREATE TABLE Makes (  
    cname VARCHAR(100) UNIQUE REFERENCES Company,  
    pname VARCHAR(100) UNIQUE REFERENCES Product,  
    PRIMARY KEY (cname, pname),  
    ...);
Relation Multiplicity

- **One-to-one**
- **Many-to-one**
- **Many-to-many**

Product
- Beyblade, ...
- Trolls, ...

Company
- Hasbro, ...
- Nyform, ...

Product makes Company
Relation Multiplicity

- **One-to-one**
- **Many-to-one**
- **Many-to-many**

Product

- Beyblade, ...
- Trolls, ...

Company

- Hasbro, ...
- Nyform, ...

Product makes Company
Relation Multiplicity

- One-to-one
- **Many-to-one**
- Many-to-many

Do I need a Makes table?

![ER Diagram](image-url)
 Relation Multiplicity

- One-to-one
- **Many-to-one**
- Many-to-many

Do I need a Makes table?

Key observation: In this many-to-one relationship, each company can make many products, but each product can only be made by a one company.
Relation Multiplicity

- **One-to-one**
- **Many-to-one**
- **Many-to-many**

Do I need a Makes table?

Key observation: In this many-to-one relationship, each company can make many products, but each product can only be made by a one company.

If we allow products to be made by multiple companies, we would have a many-to-many relationship.
Relation Multiplicity

- **One-to-one**
- **Many-to-one**
- **Many-to-many**

CREATE TABLE Company (
  name VARCHAR(100) PRIMARY KEY,
  ...
);

CREATE TABLE Product (
  name VARCHAR(100) PRIMARY KEY,
  cname VARCHAR(100) REFERENCES Company (name),
  ...
);
Relation Multiplicity

- One-to-one
- **Many-to-one**
- Many-to-many

CREATE TABLE Company (  
  name VARCHAR(100) PRIMARY KEY,  
  ...);  
CREATE TABLE Product (  
  name VARCHAR(100) PRIMARY KEY,  
  cname VARCHAR(100)  
  REFERENCES Company  
  ...);  

Foreign key alone is able to encode the Makes relationship.
• Relations can have attributes too!
Exactly-One Reference

- Rounded arrow means the relationship is not optional (exactly one vs. at most one)

CREATE TABLE Company (  
  name VARCHAR(100) PRIMARY KEY,  
  ...);  
CREATE TABLE Product (  
  name VARCHAR(100) PRIMARY KEY,  
  cname VARCHAR(100) NOT NULL  
  REFERENCES Company  
  ...);
Multi-Way Relations

- Product
- Company
- Person

purchase
Definition of a relation generalizes!

**Relationship**

If $A$ and $B$ are sets, then a relation $R$ is a subset of $A \times B$
Definition of a relation generalizes!

**Relationship**

If $A$, $B$, and $C$ are sets, then a relation $R$ is a subset of $A \times B \times C$
CREATE TABLE Product (  
  name VARCHAR(100) PRIMARY KEY,
  ...);
CREATE TABLE Company (  
  name VARCHAR(100) PRIMARY KEY,
  ...);
CREATE TABLE Person (  
  ssn INT PRIMARY KEY,
  ...);
CREATE TABLE Purchase (  
  cname VARCHAR(100) REFERENCES Company,
  pname VARCHAR(100) REFERENCES Product,
  ssn INT REFERENCES Person,
  PRIMARY KEY (cname, pname, ssn),
  ...);
I want purchases to be such that a person will only buy each product from a single company.

How would you draw it? Remember that the arrows read like an implication/function. Discuss!
I want purchases to be such that a person will only buy each product from a single company.
Do I need a Purchase table?
Multi-Way Relations

Do I need a Purchase table?
Probably a good idea

Product

Company

purchase

Person
Now do I need a Purchase table?

- Product
- Company
- Person

purchase
Now do I need a Purchase table?
Nah
Rules of Thumb in Database Design

Design Principles (common sense):
- Pick the right entities
- Don’t over complicate things
- Follow the application spec
A weak entity set has a key that is from another entity set

University(size, name)
Team(sport, name, uname)
Subclassing

- Distinguish special entities in an entity set
- Mimics heuristics in object oriented programming

Subclasses are mutually exclusive
Subclassing

Implicitly inherits superclass attributes and key
Subclassing

Company(...)  Makes(...)  Product(price, name)

Toy(name, age)
Candy(name, isChocolate)
Normal arrows are shorthand versions of \((<=1)\)
Rounded arrows are shorthand versions of \((=1)\)

Each product can be made by, at most, 3 companies
Other Constraints

- **CHECK** (condition)
  - Single attribute
  - Single tuples

```sql
CREATE TABLE User (
    uid INT PRIMARY KEY,
    firstName TEXT,
    lastName TEXT,
    age INT CHECK (age > 12 AND age < 120),
    email TEXT,
    phone TEXT,
    CHECK (email IS NOT NULL OR phone IS NOT NULL)
);```
ON UPDATE/ON DELETE

- **NO ACTION** $\rightarrow$ (default) error out
- **CASCADE** $\rightarrow$ update/delete referencers
- **SET NULL** $\rightarrow$ set referencers’ field to NULL
- **SET DEFAULT** $\rightarrow$ set referencers’ field to default
  
  - Assumes default was set, e.g.

```sql
CREATE TABLE Table (
    id INT DEFAULT 42 REFERENCES OtherTable,
    ...
);
```
CREATE TABLE Company (  
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (  
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100)  
      REFERENCES Company  
      ON UPDATE CASCADE  
      ON DELETE SET NULL);

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>Hasbro</td>
<td>Beyblade</td>
</tr>
<tr>
<td>Nyform</td>
<td>Troll</td>
</tr>
</tbody>
</table>
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100)
    REFERENCES Company
    ON UPDATE CASCADE
    ON DELETE SET NULL);

UPDATE Company
    SET name = 'lmao'
WHERE name = 'Hasbro';
CREATE TABLE Company (  
    name VARCHAR(100) PRIMARY KEY);
CREATE TABLE Product (  
    name VARCHAR(100) PRIMARY KEY,  
    cname VARCHAR(100)  
    REFERENCES Company  
    ON UPDATE CASCADE  
    ON DELETE SET NULL);

UPDATE Company  
    SET name = 'lmao'  
    WHERE name = 'Hasbro';
Referential Constraint Maintenance

CREATE TABLE Company (  
   name VARCHAR(100) PRIMARY KEY);

CREATE TABLE Product (  
   name VARCHAR(100) PRIMARY KEY,  
   cname VARCHAR(100)  
      REFERENCES Company  
      ON UPDATE CASCADE  
      ON DELETE SET NULL);

DELETE FROM Company  
WHERE name = 'lmao';

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>lmao</td>
<td>Beyblade</td>
</tr>
<tr>
<td>Nyform</td>
<td>Troll</td>
</tr>
</tbody>
</table>
CREATE TABLE Company (  
    name VARCHAR(100) PRIMARY KEY);  
CREATE TABLE Product (  
    name VARCHAR(100) PRIMARY KEY,  
    cname VARCHAR(100)  
        REFERENCES Company  
        ON UPDATE CASCADE  
        ON DELETE SET NULL);  

DELETE FROM Company  
WHERE name = 'lmao';

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>Nyform</td>
<td>Beyblade</td>
</tr>
<tr>
<td></td>
<td>Troll</td>
</tr>
</tbody>
</table>
Assertions

- Hard to support
- Usually impractical
- Usually not supported
  - Simulated with triggers

```
CREATE ASSERTION myAssert CHECK
  (NOT EXISTS (SELECT Product.name
                FROM Product, Purchase
                WHERE Product.name = Purchase.prodName
                GROUP BY Product.name
                HAVING count(*) > 200));
```
Triggers

Triggers activate on a specified event

CREATE TRIGGER LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT AS
    IF (ROWCOUNT_BIG() = 0) RETURN;
    IF EXISTS (SELECT *
        FROM Purchasing.PurchaseOrderHeader AS p
        JOIN inserted AS i
        ON p.PurchaseOrderID = i.PurchaseOrderID
        JOIN Purchasing.Vendor AS v
        ON v.BusinessEntityID = p.VendorID
        WHERE v.CreditRating = 5
    )
    BEGIN
        RAISERROR ('A vendor''s credit rating is too low to accept new purchase orders.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN
    END;
GO

= you don’t need to study this for the class
Takeaways

- ER diagrams can sketch out high-level designs
- Certain rules of thumb for ER-to-SQL conversions help preserve design semantics
- SQL allows you to make rules specific to your application