Database Systems
CSE 414

Lecture 17: E/R Diagrams (4.1-6) and Constraints (7.1-2)

Announcements

• HW5 due on Thursday
  – (was Tuesday before)
• WQ6 due on Sunday

Database Design

What it is:
• Starting from scratch, design the database schema: relation, attributes, keys, foreign keys, constraints etc

Why it’s hard:
• The database will be in operation for years.
• Updating the schema in production is very hard:
  – schema change modifications are expensive (why?)
  – making the change without introducing any bugs is hard
  – this part is, by far, the most important consideration in practice

Database Design Process

Conceptual Model:

Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details
Physical Schema

Entity / Relationship Diagrams

• Entity set = a class
  – An entity = an object
• Attribute
• Relationship
What is a Relation(ship)?

- A mathematical definition:
  - if A, B are sets, then a relation R is a subset of $A \times B$

- $A=\{1,2,3\}$, $B=\{a,b,c,d\}$,
  $A \times B = \{(1,a),(1,b), \ldots, (3,d)\}$
  $R = \{(1,a), (1,c), (3,b)\}$

- **makes** is a subset of $Product \times Company$:

Multiplicity of E/R Relations

- one-one:
- many-one
- many-many

Multi-way Relationships

How do we model a purchase relationship between buyers, products and stores?

- Can still model as a mathematical set (Q. how?)
- A. As a set of triples $\subseteq Person \times Product \times Store$
Q: What does the arrow mean?

A: A given person buys a given product from at most one store

[Arrow pointing to E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E]

Q: What does the arrow mean?

A: A given person buys a given product from at most one store AND every store sells to every person at most one product

Converting Multi-way Relationships to Binary

Arrows go in which direction?

Make sure you understand why!

3. Design Principles

What's wrong?

Moral: be faithful to the specifications of the app!

Design Principles: What's Wrong?
Design Principles: What’s Wrong?

Moral: don’t complicate life more than it already is.

From E/R Diagrams to Relational Schema

- Entity set → relation
- Relationship → relation

Entity Set to Relation

Product

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Camera</td>
<td>99.99</td>
</tr>
<tr>
<td>Pokemn19</td>
<td>Toy</td>
<td>29.99</td>
</tr>
</tbody>
</table>

N-N Relationships to Relations

Represent this in relations

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>cust-ID</th>
<th>date</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>4/10/2011</td>
<td>UPS</td>
<td></td>
</tr>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>4/9/2011</td>
<td>FEDEX</td>
<td></td>
</tr>
</tbody>
</table>

N-1 Relationships to Relations

Represent this in relations

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>cust-ID</th>
<th>date</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
N-1 Relationships to Relations

Orders

prod-ID cust-ID date name address

Shipment -> Shipping-Co

Prod

cust-ID date name

Orders\( (\text{prod-ID, cust-ID, date, ship_co, ship_date}) \)

Shipping-Co\( (\text{name, address}) \)

Remember: many-one relationship becomes FK not relation

Multi-way Relationships to Relations

Product

name address prod-ID price

Purchase

prod-ID price

Store

name ssn

Person

name ssn

Purchase\( (\text{prod-ID, ssn, name}) \)

Modeling Subclasses

Some objects in a class may be special
- define a new class
- better: define a subclass

Products

Software products

Educational products

So --- we define subclasses in E/R

Subclasses

Product

name category price

Software Product

Lisa

platforms

Educational Product

Lisa

Age Group

Subclasses to Relations (one option)

Product

name category price

Sw.Product

Name platform

Lisa

Gizmo unix

Ed.Product

Name Age Group

Gizmo toddler

Toy retired

Other ways to convert are possible...
Modeling Union Types with Subclasses

Say: each piece of furniture is owned either by a person or by a company

Solution 1. Acceptable but imperfect (What's wrong?)

Solution 2: better, more laborious

Weak Entity Sets

Entity sets are weak when their key comes from other classes to which they are related.

What Are the Keys of R ?

Ex: NFL Game DB
Integrity Constraints Motivation

An integrity constraint is a condition specified on a database schema that restricts the data stored in an instance of the database.

- ICs help prevent entry of incorrect information
- How? DBMS enforces integrity constraints
  - Allows only legal database instances (i.e., those that satisfy all constraints) to exist
  - Ensures that all necessary checks are always performed and avoids duplicating the verification logic in each application

Constraints in E/R Diagrams

Finding constraints is part of the modeling process. Commonly used constraints:

- **Keys**: social security number uniquely identifies a person.
- **Single-value constraints**: can have only one genetic father
- **Referential integrity constraints**: if you work for a company, it must exist in the database.
- **Other constraints**: peoples’ ages are between 0 and 150. Some values should not be NULL

Keys in E/R Diagrams

No formal way to specify multiple keys in E/R diagrams

Single Value Constraints

Other Constraints

Which one is FK?

Q: What does this mean?
A: A Company entity cannot be connected by relationship to more than 99 Product entities
Constraints in SQL

Constraints in SQL:
• Keys, foreign keys
• Attribute-level constraints
• Tuple-level constraints
• Global constraints: assertions
• The more complex the constraint, the harder it is to check and to enforce...
  – (Still, performance is secondary to correctness.)

Key Constraints

Product(name, category)

CREATE TABLE Product (name CHAR(30) PRIMARY KEY, category VARCHAR(20))

OR:

CREATE TABLE Product (name CHAR(30), category VARCHAR(20), PRIMARY KEY (name))

Keys with Multiple Attributes

Product(name, category, price)

CREATE TABLE Product (name CHAR(30), category VARCHAR(20), price INT, PRIMARY KEY (name, category))

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>10</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>20</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Photo</td>
<td>30</td>
</tr>
</tbody>
</table>

Other Keys

CREATE TABLE Product (productID CHAR(10), name CHAR(30), category VARCHAR(20), price INT, PRIMARY KEY (productID), UNIQUE (name, category))

There is at most one PRIMARY KEY; there can be many UNIQUE

Foreign Key Constraints

CREATE TABLE Purchase (prodName CHAR(30) REFERENCES Product(name), date DATETIME)

prodName is a foreign key to Product(name)
name must be a key in Product (i.e., PK or UNIQUE)

Foreign Key Constraints

• Example with multi-attribute primary key
CREATE TABLE Purchase (prodName CHAR(30), category VARCHAR(20), date DATETIME, FOREIGN KEY (prodName, category) REFERENCES Product(name, category))

• (name, category) must be a KEY in Product
What happens when data changes?

Types of updates:
- In Purchase: insert/update
- In Product: delete/update

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ProName</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
</tbody>
</table>

What happens when data changes?

SQL has three options for maintaining referential integrity on changes:
- **NO ACTION** reject bad modifications (default)
- **CASCADE** after delete/update do delete/update
- **SET NULL** set foreign-key field to NULL
- **SET DEFAULT** set FK field to default value

```
CREATE TABLE Product (productID CHAR(10), name CHAR(30), category VARCHAR(20), price INT CHECK (price > 0), PRIMARY KEY (productID))
```

Constraints on Attributes and Tuples

- Constraints on attributes:
  - **NOT NULL** -- obvious meaning...
  - **CHECK** condition -- any condition!
- Constraints on tuples
  - **CHECK** condition

```
CREATE TABLE Purchase (prodName CHAR(30), category VARCHAR(20), date DATETIME, FOREIGN KEY (prodName, category) REFERENCES Product(name, category) ON UPDATE CASCADE ON DELETE SET NULL)
```

Constraints on Attributes and Tuples

```
CREATE TABLE Product (productID CHAR(10), name CHAR(30), category VARCHAR(20) CHECK (category in ('toy','gadget','apparel')), price INT CHECK (price > 0), PRIMARY KEY (productID))
```
Constraints on Attributes and Tuples

```sql
CREATE TABLE Product (
    productID CHAR(10),
    name CHAR(30) NOT NULL,
    category VARCHAR(20) CHECK (category in ('toy', 'gadget', 'apparel')),
    price INT CHECK (price > 0),
    PRIMARY KEY (productID))
```

Constraints on Attributes and Tuples

```sql
CREATE TABLE R (
    A int NOT NULL,
    B int CHECK (B > 50 and B < 100),
    C varchar(20),
    D int,
    CHECK (C >= 'd' or D > 0))
```

Constraints on Attributes and Tuples

```sql
CREATE TABLE Purchase (
    prodName CHAR(30) CHECK (prodName IN (SELECT Product.name FROM Product, Purchase WHERE Product.name = Purchase.prodName GROUP BY Product.name HAVING count(*) > 200)),
    date DATETIME NOT NULL)
```

General Assertions

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

But most DBMSs do not implement assertions because it is hard to support them efficiently. Instead, they provide triggers.