Constraints

- Constraint: a property that we want our data to satisfy
- Enforce by taking actions:
  - Forbid an update
  - Or perform compensating updates

- Two approaches:
  - Declarative constraints
  - Triggers
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 enforce
Keys

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

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>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>40</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))
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 (primary key or unique)
CREATE TABLE Purchase (  
    prodName CHAR(30),  
    category VARCHAR(20),  
    date DATETIME,  
    store VARCHAR(30),  
    FOREIGN KEY (prodName, category)  
    REFERENCES Product(name, category)  
)
What happens during updates?

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td></td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purchase</th>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
<td></td>
</tr>
</tbody>
</table>
What happens during updates?

- SQL has three policies for maintaining referential integrity:
  - **Reject** violating modifications (default)
  - **Cascade**: after a delete/update do a delete/update
  - **Set-null** set foreign-key field to NULL

```sql
CREATE TABLE Purchase (  
    prodName CHAR(30)  
    REFERENCES Product(name)  
    ON DELETE SET NULL  
    ON UPDATE CASCADE)
```
Constraints on Attributes and Tuples

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

```sql
CREATE TABLE Purchase (... 
  store VARCHAR(30) NOT NULL, ...)
```

```sql
CREATE TABLE Product(... 
  price INT CHECK(price>0 and price <999))
```

- Constraints on tuples
  - CHECK condition

```sql
... CHECK(price*quantity < 1000)...
```
Constraint Examples

CREATE TABLE Purchase (  
    prodName CHAR(30)  
    CHECK (prodName IN  
            SELECT Product.name  
            FROM Product),  
    date DATETIME NOT NULL)

Often not implemented in DBMSs!
CREATE ASSERTION myAssert CHECK
NOT EXISTS(
    SELECT Product.name
    FROM Product, Purchase
    WHERE Product.name = Purchase.prodName
    GROUP BY Product.name
    HAVING count(*) > 200)
Semantic Optimization with Constraints

Product(name, price)
Purchase(buyer, seller, prodName, store)

```
SELECT Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
```

```
SELECT Purchase.store
FROM Purchase
```
Semantic Optimization with Constraints

Product(name, price)
Purchase(buyer, seller, prodName, store)

\[
\text{SELECT Purchase.store} \\
\text{FROM Product, Purchase} \\
\text{WHERE Product.name = Purchase.prodName}
\]

Yes, if Purchase.prodName is a foreign key, and not null
Database Design

- Why do we need it?
  - Need a way to model real world entities in terms of relations
  - Not easy to go from real-world entities to a database schema

- Consider issues such as:
  - What entities to model
  - How entities are related
  - What constraints exist in the domain
  - How to achieve good designs

- Several formalisms exists
  - We discuss E/R diagrams
Database Design Process

**Today**
- Data Modeling

**Next lecture**
- Refinement
- SQL Tables
- Files
- Conceptual Schema
- Physical Schema

- E/R diagrams
- Relations
Conceptual Schema Design

Conceptual Model:

Relational Model: plus FDs (FD = Functional Dependency)

Normalization: Eliminates anomalies
Entity / Relationship Diagrams

- Product
- address
- buys

entity set

All entities in the same entity set have the same attributes

Association between 2 or more entities

A relationship may have attributes too!
Keys in E/R Diagrams

- Every entity set must have a key

Diagram:
- Product
  - name
  - price
  - category
What is a Relationship?

- If A, B are sets, then a relationship R is a subset of A × B

- A={1,2,3}, B={a,b,c,d},
  - A × B = {(1,a),(1,b), . . . , (3,d)}
  - R = {(1,a), (1,c), (3,b)}

- “makes” is a subset of Product × Company:
Multiplicity of E/R Relations

- **one-one:**
  - Diagram showing a one-to-one relationship between two entities.

- **one-many:**
  - Diagram showing a one-to-many relationship between two entities.

- **many-many:**
  - Diagram showing a many-to-many relationship between two entities.
What does this mean?
Multi-way Relationships

How do we model a work relationship between employees, departments and locations?
Key Constraints in Multi-way Relationships

Q: What does the arrow mean?

A: A given employee works at a single department and location
Q: Every employee works in at least one department.
**Participation Constraints**

**Q:** Every employee works in at least one department, and every department has at least one employee.
Participation Constraints

Q: What does this mean?

A: previous + every department has exactly one manager
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:** a person has only one (biological) mother.
  - **Referential integrity constraints:** if you work for a company, it must exist in the database.
  - **Other constraints:** peoples’ ages are between 0 and 150.
Keys in E/R Diagrams

No formal way to specify multiple keys
Single Value Constraints

makes vs makes
Referential Integrity Constraints

Each product made by at most one company.
Some products made by no company

Each product made by exactly one company.
Other Constraints

What does this mean?
Design Principles

What’s wrong?

Be faithful to the specifications of the app!
Design Principles

What’s wrong?

Product — Purchase — Store

- date
- personAddr
- personName

Pick the right kind of entities!
Design Principles

What’s wrong?

Don’t complicate your design more than necessary!
From E/R Diagrams to Relational Schema

- Entity set → relation
- Relationship → relation
Entity Set to Relation

Product(name, category, price)

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadgets</td>
<td>$19.99</td>
</tr>
</tbody>
</table>
Relationships to Relations

Watch out for attribute name conflicts

**Makes**\((\text{product-name}, \text{product-category}, \text{companyName}, \text{year})\)

<table>
<thead>
<tr>
<th>ProductName</th>
<th>ProductCategory</th>
<th>CompanyName</th>
<th>startYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td>1963</td>
</tr>
</tbody>
</table>
Modeling Subclasses

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

So --- we define subclasses in E/R
Subclasses

Product

- name
- price
- category

Software Product

Educational Product

isa

platforms

ageGroup
Subclasses to Relations

Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>

Educational Product

<table>
<thead>
<tr>
<th>Name</th>
<th>ageGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>toddler</td>
</tr>
<tr>
<td>Camera</td>
<td>adult</td>
</tr>
</tbody>
</table>

Software Product

<table>
<thead>
<tr>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
</tr>
</tbody>
</table>
E/R Inheritance

Entity sets overlap

No need for multiple inheritance
Modeling Union Types With Subclasses

Say:
	each piece of furniture is owned either by a person, or by a company
Modeling Union Types with Subclasses

- Solution 1: acceptable, but imperfect *(why?)*
Modeling Union Types with Subclasses

- Solution 2: better, more laborious

In fact there is no formal way to represent disjoint vs overlapping subclasses.
Entity sets are weak when their key comes from classes to which they’re related
Handling Weak Entity Sets

Employee(ssn, name, dept)
Dependents(ssn, name, age)

No need to represent policy separately

We should have an arrow here, otherwise Dependents key would have NULL value!