DS 1300 - Introduction to SQL
Part 2 – Multi-table Queries

By Michael Hahsler
based on slides for CS145 Introduction to Databases (Stanford)
What you will learn about in this section

1. Foreign key constraints
2. Joins: basics
3. Joins: SQL semantics
4. Activities: Multi-table queries
Foreign Key Constraints

• Suppose we have the following schema:

```
Students(sid: text, name: text, gpa: real)
Enrolled(student_id: text, cid: text, grade: real)
```

• And we want to impose the following constraint:

  ‘Only existing students may enroll in courses’ i.e. a student must appear in the Students table to enroll in a class

<table>
<thead>
<tr>
<th>Students</th>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>101</td>
<td>Bob</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
</tr>
</tbody>
</table>

We say that student_id is a **foreign key** that refers to Students.
Declaring Foreign Keys

Students(sid: text, name: text, gpa: real)
Enrolled(student_id: text, cid: text, grade: text)

CREATE TABLE Enrolled(
    student_id CHAR(20),
    cid CHAR(20),
    grade CHAR(10),
    PRIMARY KEY (student_id, cid),
    FOREIGN KEY (student_id) REFERENCES Students
)

Primary key  -  Foreign key
Foreign Keys and Update Operations

Students(sid: text, name: text, gpa: real)
Enrolled(student_id: text, cid: text, grade: text)

• What if we insert a tuple into Enrolled, but no corresponding student?
  − INSERT is rejected (foreign keys are constraints)!

• What if we delete a student?
  1. Disallow the delete
  2. Remove all of the courses for that student
  3. SQL allows a third via NULL (not yet covered)

SQLite: Enable foreign keys with
PRAGMA foreign_keys = ON;

DB Browser: check “Foreign Keys” in “Edit Pragma”
Keys and Foreign Keys

**Company**

<table>
<thead>
<tr>
<th>CName</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

**Product**

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

What is a foreign key vs. a key here?
Keys and Foreign Keys

- This example uses **natural keys**.
- Often **surrogate keys** are used instead:

  Company(CName, StockPrice, Country)
  Product(PName, Price, Category, Manufacturer)

- Why?
- Why do we use SMU IDs and Social Security Numbers?
Joins

Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

This will need information from both tables...
Joins

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

SELECT PName, Price
FROM Product, Company
WHERE Manufacturer = CName
AND Country = 'Japan'
AND Price <= 200

A join between tables returns all unique combinations of their tuples which meet some specified join condition
Several equivalent ways to write a basic join in SQL:

```
SELECT PName, Price
FROM Product, Company
WHERE
  Manufacturer = CName
  AND Country='Japan'
  AND Price <= 200
```

```
SELECT PName, Price
FROM Product
JOIN Company ON Manufacturer = Cname
WHERE
  Price <= 200
  AND Country='Japan'
```

Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
### Joins

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>PName</td>
<td>Cname</td>
</tr>
<tr>
<td>Price</td>
<td>Stock</td>
</tr>
<tr>
<td>Category</td>
<td>Country</td>
</tr>
<tr>
<td>Manuf</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gizmo</th>
<th>$19</th>
<th>Gadgets</th>
<th>GWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Powergizmo</td>
<td>$29</td>
<td>Gadgets</td>
<td>GWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**SELECT** PName, Price
**FROM** Product
**JOIN** Company **ON** Manufacturer = Cname
**WHERE** Price <= 200 AND Country='Japan'

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
</tr>
</tbody>
</table>
### SQL Semantics Example

**SELECT** R.A 
**FROM** R, S 
**WHERE** R.A = S.B
An Example of SQL Semantics

SELECT R.A
FROM R, S
WHERE R.A = S.B
Tuple Variable Ambiguity in Multi-Table

Person(name, address, worksfor)
Company(name, address)

SELECT DISTINCT name, address
FROM Person, Company
WHERE worksfor = name

Which “address” does this refer to?
Which “name”s??
Tuple Variable Ambiguity in Multi-Table

Person(name, address, worksfor)
Company(name, address)

SELECT DISTINCT Person.name, Person.address
FROM Person, Company
WHERE Person.worksfor = Company.name

SELECT DISTINCT p.name, p.address
FROM Person p, Company c
WHERE p.worksfor = c.name

Both equivalent ways to resolve variable ambiguity
A Note on Semantics

• “semantics” is not equal to “execution order”

• The preceding slides show what a join means

• Not actually how the DBMS executes it under the covers
Activities

1. Create the product/company database from the slide set. Add the following relation

   \[
   \text{Purchase}(\text{id}, \text{product}, \text{buyer}).
   \]

   with the appropriate foreign key constraints and add some data.

2. Find all countries that manufacture some product in the ‘Gadgets’ category (shows each country only once).

3. Find all products that are manufactured in the US sorted by price.

4. For a given buyer, in how many different countries are the products she purchases manufactured?