COMP 430
Intro. to Database Systems

Multi-table SQL

Get clickers today!

Slides use ideas from Chris Ré and Chris Jermaine.
The need for multiple tables

Using a single table leads to repeating data
• Provides the opportunity for inconsistency
• Requires more storage space & I/O time

<table>
<thead>
<tr>
<th>Product</th>
<th>p_name</th>
<th>price</th>
<th>manufacturer</th>
<th>address</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
<td></td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
<td></td>
</tr>
<tr>
<td>Widget</td>
<td>19.99</td>
<td>WidgetsRUs</td>
<td>20 Main St.</td>
<td>New York</td>
<td>NY</td>
<td></td>
</tr>
<tr>
<td>HyperWidget</td>
<td>203.99</td>
<td>Hyper</td>
<td>1 Mission Dr.</td>
<td>San Francisco</td>
<td>CA</td>
<td></td>
</tr>
</tbody>
</table>

Product (p_name, price, manufacturer, address, city, state)
Using multiple tables

### Product

<table>
<thead>
<tr>
<th>p_name</th>
<th>price</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Widget</td>
<td>19.99</td>
<td>WidgetsRUs</td>
</tr>
<tr>
<td>HyperWidget</td>
<td>203.99</td>
<td>Hyper</td>
</tr>
</tbody>
</table>

### Company

<table>
<thead>
<tr>
<th>c_name</th>
<th>address</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
<tr>
<td>WidgetsRUs</td>
<td>20 Main St.</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>Hyper</td>
<td>1 Mission Dr.</td>
<td>San Francisco</td>
<td>CA</td>
</tr>
</tbody>
</table>

Product (**p_name**, price, manufacturer)  

Company (**c_name**, address, city, state)  

Later in course: Deciding what fields belong in what tables.
Foreign keys & referential integrity

<table>
<thead>
<tr>
<th>Product</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>price</td>
<td>manufacturer</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Widget</td>
<td>19.99</td>
<td>WidgetsRUs</td>
</tr>
<tr>
<td>HyperWidget</td>
<td>203.99</td>
<td>Hyper</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c_name</td>
<td>address</td>
<td>city</td>
<td>state</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
<tr>
<td>WidgetsRUs</td>
<td>20 Main St.</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>Hyper</td>
<td>1 Mission Dr.</td>
<td>San Francisco</td>
<td>CA</td>
</tr>
</tbody>
</table>

Product’s Manufacturer is a *foreign key*.

- Foreign keys always refer to primary keys.
- We want to enforce *referential integrity* – each Manufacturer value is an existing c_name.
Creating a table with a foreign key

CREATE TABLE Product (  
  p_name VARCHAR(50),  
  price DECIMAL(6,2),  
  manufacturer VARCHAR(50),  
  PRIMARY KEY (p_name),  
  FOREIGN KEY (manufacturer) REFERENCES Company (c_name)
);

CREATE TABLE Company (  
  c_name VARCHAR(50),  
  address VARCHAR(50),  
  city VARCHAR(50),  
  state CHAR(2),  
  PRIMARY KEY (c_name)
);
Foreign key represents a dependence

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>c_name</td>
</tr>
<tr>
<td>price</td>
<td>address</td>
</tr>
<tr>
<td>manufacturer</td>
<td>city</td>
</tr>
<tr>
<td>manufacturer</td>
<td>state</td>
</tr>
</tbody>
</table>

Gizmo  | GizmoWorks | 123 Gizmo St. | Houston | TX |

Product conceptually dependent on Company to elaborate details.
Product data dependent on Company data being entered.

CREATE TABLE Product (  
  ....  
  FOREIGN KEY (manufacturer) REFERENCES Company (c_name)  
);

CREATE TABLE Company ( ...);

Product def’n dependent on Company def’n.
Foreign keys vs. pointers

Foreign keys relate tables, or equivalently, sets of attributes.

- Repeats data to connect individual records.
- One relationship between tables vs. many pointers between table records.

Avoid data pointers because

- Difficult to maintain pointers when data moves, esp. with concurrency.
- Want to relate with query results, in addition to tables.
In what ways do we relate tables?

Explore design issues & common patterns later.

Two basic building blocks:
- One-to-many relationships
- Many-to-many relationships
In what ways do we relate tables?

Explore design issues & common patterns later.

Two basic building blocks:
  • One-to-many relationships
  • Many-to-many relationships
One-to-many & many-to-one relationships

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>c_name</td>
</tr>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Widget</td>
<td>WidgetsRUs</td>
</tr>
<tr>
<td>HyperWidget</td>
<td>Hyper</td>
</tr>
</tbody>
</table>

Product (p_name, price, manufacturer)

Company (c_name, address, city, state)

Each Company can have many Products.
Each Product is made by exactly one Company.
Many-to-many relationships

### Student

<table>
<thead>
<tr>
<th>s_id</th>
<th>first_name</th>
<th>last_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>S01</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>S02</td>
<td>Mary</td>
<td>Wallace</td>
</tr>
<tr>
<td>S03</td>
<td>Sue</td>
<td>Roper</td>
</tr>
<tr>
<td>S04</td>
<td>Mark</td>
<td>Jones</td>
</tr>
</tbody>
</table>

### Enrollment

<table>
<thead>
<tr>
<th>s_id</th>
<th>crn</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>S01</td>
<td>01234</td>
<td>A</td>
</tr>
<tr>
<td>S01</td>
<td>46117</td>
<td>C</td>
</tr>
<tr>
<td>S02</td>
<td>01234</td>
<td>B</td>
</tr>
</tbody>
</table>

### Course

<table>
<thead>
<tr>
<th>crn</th>
<th>dept</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>01234</td>
<td>COMP</td>
<td>140</td>
</tr>
<tr>
<td>15134</td>
<td>COMP</td>
<td>160</td>
</tr>
<tr>
<td>46117</td>
<td>ELEC</td>
<td>220</td>
</tr>
</tbody>
</table>

Each Student can be enrolled in many Courses.
Each Course has many Students.
What is Student’s primary key?

A. s_id  
B. s_id, first_name, last_name  
C. first_name, Last_name  
D. last_name

Student

<table>
<thead>
<tr>
<th>s_id</th>
<th>first_name</th>
<th>last_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>S01</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>S02</td>
<td>Mary</td>
<td>Wallace</td>
</tr>
<tr>
<td>S03</td>
<td>Sue</td>
<td>Roper</td>
</tr>
<tr>
<td>S04</td>
<td>Mark</td>
<td>Jones</td>
</tr>
</tbody>
</table>
What is Course’s primary key?

A. crn
B. crn, dept, number
C. dept, number
D. number

Course

<table>
<thead>
<tr>
<th>crn</th>
<th>dept</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>01234</td>
<td>COMP</td>
<td>140</td>
</tr>
<tr>
<td>15134</td>
<td>COMP</td>
<td>160</td>
</tr>
<tr>
<td>46117</td>
<td>ELEC</td>
<td>220</td>
</tr>
</tbody>
</table>

Response Counter

0% 25% 25% 25%
What is Enrollment’s primary key?

A. s_id
B. s_id, crn  
C. s_id, crn, grade  
D. crn  
E. grade

**Enrollment**

<table>
<thead>
<tr>
<th>s_id</th>
<th>crn</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>S01</td>
<td>01234</td>
<td>A</td>
</tr>
<tr>
<td>S01</td>
<td>46117</td>
<td>C</td>
</tr>
<tr>
<td>S02</td>
<td>01234</td>
<td>B</td>
</tr>
</tbody>
</table>
What is a foreign key in Student?

A. s_id  
B. first_name, last_name  
C. N/A

Student (s_id, FirstName, last_name)  
Enrollment (s_id, crn, grade)  
Course (crn, dept, number)
What is a foreign key in Course?

A. crn
B. dept, number
C. N/A

Student (s_id, first_name, last_name)
Enrollment (s_id, crn, grade)
Course (crn, dept, number)
What is a foreign key in Enrollment?

A. s_id
B. crn
C. the pair sid, crn is a single foreign key
D. s_id and crn are each foreign keys

D. s_id and crn are each foreign keys
Creating a table with a foreign key

```
CREATE TABLE Student (
s_id CHAR(10),
first_name VARCHAR(50),
last_name VARCHAR(50),
PRIMARY KEY (s_id)
);

CREATE TABLE Course (
crn CHAR(10),
department CHAR(4),
number CHAR(3),
PRIMARY KEY (crn)
);

CREATE TABLE Enrollment (
s_id CHAR(10),
crn CHAR(10),
grade CHAR(2),
PRIMARY KEY (s_id, crn),
FOREIGN KEY (s_id) REFERENCES Student (s_id),
FOREIGN KEY (crn) REFERENCES Course (crn)
);
```
Another representation – ER diagram

Style: Crow’s foot
Conceptual level: Physical
A more abstract ER diagram

Style: Chen
Conceptual level: Logical
### Joining tables

**Product**

<table>
<thead>
<tr>
<th>p_name</th>
<th>price</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Widget</td>
<td>19.99</td>
<td>WidgetsRUs</td>
</tr>
<tr>
<td>HyperWidget</td>
<td>203.99</td>
<td>Hyper</td>
</tr>
</tbody>
</table>

**Company**

<table>
<thead>
<tr>
<th>c_name</th>
<th>address</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
<tr>
<td>WidgetsRUs</td>
<td>20 Main St.</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>Hyper</td>
<td>1 Mission Dr.</td>
<td>San Francisco</td>
<td>CA</td>
</tr>
</tbody>
</table>

**SQL Query**

```sql
SELECT * 
FROM Product, Company 
WHERE manufacturer = c_name;
```
Joining tables

<table>
<thead>
<tr>
<th>p_name</th>
<th>price</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Widget</td>
<td>19.99</td>
<td>WidgetsRUs</td>
</tr>
<tr>
<td>HyperWidget</td>
<td>203.99</td>
<td>Hyper</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c_name</th>
<th>address</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
<tr>
<td>WidgetsRUs</td>
<td>20 Main St.</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>Hyper</td>
<td>1 Mission Dr.</td>
<td>San Francisco</td>
<td>CA</td>
</tr>
</tbody>
</table>

This is simplest & most common kind of join – an *inner join*. See other kinds later.
# Joins – forgetting the join condition

```sql
SELECT *
FROM Product, Company;
```

<table>
<thead>
<tr>
<th>p_name</th>
<th>price</th>
<th>manufacturer</th>
<th>c_name</th>
<th>address</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
<td>WidgetsRUs</td>
<td>20 Main St.</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
<td>Hyper</td>
<td>1 Mission Dr.</td>
<td>San Francisco</td>
<td>CA</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
<td>WidgetsRUs</td>
<td>20 Main St.</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
<td>Hyper</td>
<td>1 Mission Dr.</td>
<td>San Francisco</td>
<td>CA</td>
</tr>
<tr>
<td>Widget</td>
<td>19.99</td>
<td>WidgetsRUs</td>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
</tbody>
</table>

All combinations of records! – Cross-product of tables.
Selection & Projection

```
SELECT p_name
FROM   Product, Company
WHERE  manufacturer = c_name AND state = 'TX';
```

<table>
<thead>
<tr>
<th>p_name</th>
<th>price</th>
<th>manufacturer</th>
<th>address</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>39.99</td>
<td>GizmoWorks</td>
<td>123 Gizmo St.</td>
<td>Houston</td>
<td>TX</td>
</tr>
</tbody>
</table>

**Selection**

<table>
<thead>
<tr>
<th>p_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
</tr>
<tr>
<td>Powergizmo</td>
</tr>
</tbody>
</table>

**Projection**

<table>
<thead>
<tr>
<th>p_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
</tr>
<tr>
<td>Powergizmo</td>
</tr>
</tbody>
</table>
Attribute name conflicts

Person (name, address, works_for)
Company (name, address)

```
SELECT  name, address
FROM    Person, Company
WHERE   works_for = name;
```

```
SELECT  Person.name, Person.address
FROM    Person, Company
WHERE   Person.works_for = Company.name;
```

```
SELECT  p.name, p.address
FROM    Person p, Company c
WHERE   p.works_for = c.name;
```
Semantics, multisets, sets
Semantics – set notation

SELECT [DISTINCT] T₁.a₁, T₁.a₂, ..., Tₙ.aₘ
FROM T₁, T₂, ..., Tₙ
WHERE Conditions(T₁.a'₁, T₁.a'₂, ..., Tₙ.a'ᵣ);

\{(T₁.a₁, T₁.a₂, ..., Tₙ.aₘ) | Conditions(T₁.a'₁, T₁.a'₂, ..., Tₙ.a'ᵣ)\}

Multisets by default. Sets with DISTINCT.
Semantics – order of steps

SELECT [DISTINCT] $T_1.a_1, T_1.a_2, ..., T_n.a_m$
FROM $T_1, T_2, ..., T_n$
WHERE Conditions($T_1.a'_1, T_1.a'_2, ..., T_n.a'_p$);

Answer = {}
for row_1 in $T_1$ do
  for row_2 in $T_2$ do
    ...
    for row_n in $T_n$ do
      if Conditions(row_1.a'_1, row_1.a'_2, ..., row_n.a'_p)
      then Answer = Answer $\cup$ (row_1.a_1, row_1.a_2, ..., row_n.a_m)
return Answer

Multiset union by default.
Set union with DISTINCT.

1. Cross-product
2. Selection
3. Projection
Activity: Writing multi-table queries

03a-queries.ipynb
What is result of query?

A. $R \times (S \cup T)$
B. $R \cup S \cup T$
C. $R \cap (S \cup T)$
D. None of the above

[Correct Answer: D. None of the above]

Schemas: $R(a)$ $S(a)$ $T(a)$

SQL Query:

```sql
SELECT DISTINCT R.a
FROM R, S, T
WHERE R.a = S.a OR R.a = T.a;
```
What is result of query if S is empty?

A. R
B. T
C. $R \cap T$
D. $\emptyset$

Schemas: R(a) S(a) T(a)

```
SELECT DISTINCT R.a
FROM   R, S, T
WHERE  R.a = S.a OR R.a = T.a;
```
Query summary

Schemas: R(a) S(a) T(a)

SELECT DISTINCT R.a
FROM R, S, T
WHERE R.a = S.a OR R.a = T.a;

• If S = ∅, then ∅
• If T = ∅, then ∅
• Else R ∩ (S ∪ T)
Two ways to understand multisets

<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, a)</td>
<td>2</td>
</tr>
<tr>
<td>(1, b)</td>
<td>1</td>
</tr>
<tr>
<td>(2, c)</td>
<td>3</td>
</tr>
<tr>
<td>(1, d)</td>
<td>2</td>
</tr>
</tbody>
</table>
Multiset union

\[ \mathcal{Z} = \mathcal{X} \cup \mathcal{Y} \]

\[ \lambda(\mathcal{Z}) = \lambda(\mathcal{X}) + \lambda(\mathcal{Y}) \]

<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(\mathcal{X}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, a)</td>
<td>2</td>
</tr>
<tr>
<td>(1, b)</td>
<td>0</td>
</tr>
<tr>
<td>(2, c)</td>
<td>3</td>
</tr>
<tr>
<td>(1, d)</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(\mathcal{Y}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, a)</td>
<td>5</td>
</tr>
<tr>
<td>(1, b)</td>
<td>1</td>
</tr>
<tr>
<td>(2, c)</td>
<td>2</td>
</tr>
<tr>
<td>(1, d)</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(\mathcal{Z}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, a)</td>
<td>7</td>
</tr>
<tr>
<td>(1, b)</td>
<td>1</td>
</tr>
<tr>
<td>(2, c)</td>
<td>5</td>
</tr>
<tr>
<td>(1, d)</td>
<td>2</td>
</tr>
</tbody>
</table>
Multiset intersection

\[
\begin{align*}
\text{Tuple} & \quad \lambda(X) \\
(1, a) & \quad 2 \\
(1, b) & \quad 0 \\
(2, c) & \quad 3 \\
(1, d) & \quad 0
\end{align*}
\]

\[
\begin{align*}
\text{Tuple} & \quad \lambda(Y) \\
(1, a) & \quad 5 \\
(1, b) & \quad 1 \\
(2, c) & \quad 2 \\
(1, d) & \quad 2
\end{align*}
\]

\[
\begin{align*}
\text{Tuple} & \quad \lambda(Z) \\
(1, a) & \quad 2 \\
(1, b) & \quad 0 \\
(2, c) & \quad 2 \\
(1, d) & \quad 0
\end{align*}
\]

\[
\sum \cap = \min \left( \lambda(X), \lambda(Y) \right)
\]
Multiset difference

\[ \lambda(Z) = \lambda(X) - \lambda(Y) \]
Set & multiset union

\[
\begin{align*}
\text{SELECT } & \text{ R.a} \\
\text{FROM } & \text{ R, S} \\
\text{WHERE } & \text{ R.a = S.a} \\
\text{UNION} \\
\text{SELECT } & \text{ R.a} \\
\text{FROM } & \text{ R, T} \\
\text{WHERE } & \text{ R.a = T.a;}
\end{align*}
\]
Set & multiset intersection

\[
\text{SELECT R.a FROM R, S WHERE R.a = S.a}
\]

\[
\text{INTERSECTION}
\]

\[
\text{SELECT R.a FROM R, T WHERE R.a = T.a;}
\]

\[
\text{SELECT R.a FROM R, S WHERE R.a = S.a}
\]

\[
\text{INTERSECTION ALL}
\]

\[
\text{SELECT R.a FROM R, T WHERE R.a = T.a;}
\]
Set & multiset difference

<table>
<thead>
<tr>
<th>SQL Expression</th>
<th>SQL Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT R.a</td>
<td>SELECT R.a</td>
</tr>
<tr>
<td>FROM R, S</td>
<td>FROM R, S</td>
</tr>
<tr>
<td>WHERE R.a = S.a</td>
<td>WHERE R.a = S.a</td>
</tr>
<tr>
<td><strong>EXCEPT</strong></td>
<td><strong>EXCEPT ALL</strong></td>
</tr>
<tr>
<td>SELECT R.a</td>
<td>SELECT R.a</td>
</tr>
<tr>
<td>FROM R, T</td>
<td>FROM R, T</td>
</tr>
<tr>
<td>WHERE R.a = T.a;</td>
<td>WHERE R.a = T.a;</td>
</tr>
</tbody>
</table>
Activity – Sets & multisets

03b-sets-multisets.ipynb
A final tricky example

Company (c_name, hq_loc)
Product (p_name, manufacturer, factory_loc)

Goal: Find HQs of companies that manufacture in both U.S. and China.

Suggested solution:

SELECT hq_loc
FROM Company, Product
WHERE manufacturer = c_name
    AND factory_loc = ‘U.S.’
INTERSECT
SELECT hq_loc
FROM Company, Product
WHERE manufacturer = c_name
    AND factory_loc = ‘China’;
Exercise: Compute results

```
SELECT hq_loc
FROM Company, Product
WHERE manufacturer = c_name
  AND factory_loc = 'U.S.'
INTERSECT
SELECT hq_loc
FROM Company, Product
WHERE manufacturer = c_name
  AND factory_loc = 'China';
```
One solution: set membership + subquery

```
SELECT DISTINCT hq_loc
FROM Company, Product
WHERE manufacturer = c_name
  AND c_name IN (  
    SELECT manufacturer
    FROM Product
    WHERE factory_loc = 'U.S.'
  )
  AND c_name IN (  
    SELECT manufacturer
    FROM Product
    WHERE factory_loc = 'China'
  );
```

More subqueries later.