Lecture 2 (cont’d) & Lecture 3: Advanced SQL – Part I
Announcements!

1. You should be Jupyter notebook Ninjas!

2. Welcome Ting!
   - New TA-Office hours on website (room to be announced)

3. Project groups finalized!
   - If you do not have a group talk with us ASAP!

4. Problem Set #1 released
Lecture 2 (cont’d) & Lecture 3: Advanced SQL – Part I
Today’s Lecture

1. Recap from Lecture 2 & Multi-table queries
   • ACTIVITY: Multi-table queries

2. Set operators & nested queries
   • ACTIVITY: Set operator subtleties
Lecture 2 (cont’d):
Introduction to SQL
3. Multi-table queries
What you will learn about in this section

1. Primary keys and Foreign keys recap
2. Joins: SQL semantics
3. ACTIVITY: Multi-table queries
Keys and Foreign Keys

A **key** is a *minimal subset of attributes* that acts as a unique identifier for tuples in a relation.

If two tuples agree on the values of the key, then they must be the **same tuple**!
Keys and Foreign Keys

What is a foreign key vs. a key here?

A foreign key is an attribute (or collection of attributes) in one table that uniquely identifies a row of another table.

The foreign key is defined in a second table, but it refers to the primary key in the first table.

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>
Declaring Foreign Keys

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

CREATE TABLE Product(
  pname VARCHAR(100),
  price FLOAT,
  category VARCHAR(100),
  manufacturer VARCHAR(100),
  PRIMARY KEY (pname, manufacturer),
  FOREIGN KEY (manufacturer) REFERENCES Company(cname)
)
Declaring Foreign Keys

CREATE TABLE Company(
    cname VARCHAR(100),
    stockprice FLOAT,
    country VARCHAR(100),
    PRIMARY KEY (cname),
    FOREIGN KEY (cname) REFERENCES Product(pname, manufacturer)
)

CREATE TABLE Product(
    pname VARCHAR(100),
    price FLOAT,
    category VARCHAR(100),
    manufacturer VARCHAR(100),
    PRIMARY KEY (pname, manufacturer)
)

Can we do this? What would be the problem?
### Declaring Foreign Keys

```
CREATE TABLE Company(
    cname VARCHAR(100),
    stockprice FLOAT,
    country VARCHAR(100),
    PRIMARY KEY (cname),
    FOREIGN KEY (cname) REFERENCES Product(pname, manufacturer)
)
```

```
CREATE TABLE Product(
    pname VARCHAR(100),
    price FLOAT,
    category VARCHAR(100),
    manufacturer VARCHAR(100),
    PRIMARY KEY (pname, manufacturer)
)
```

Can we do this? What would be the problem?

We can have products without a registered company! Bad design! We’ll see more next week.
If the **primary key** is a set of columns (a **composite key**), then the **foreign key** also must be a set of columns that corresponds to the **composite key**.
SELECT PName, Price 
FROM Product, Company 
WHERE Manufacturer = CName 
AND Country='Japan' 
AND Price <= 200
An example of SQL semantics

```
SELECT R.A
FROM R, S
WHERE R.A = S.B
```

Output

```
A
3
3
```

Cross Product

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
```

Apply Selections / Conditions

```
A | B | C
---|---|---
3 | 3 | 4
3 | 3 | 5
```

Apply Projection

```
A
3
3
```
Note the **semantics** of a join

1. Take **cross product**:
   \[ X = R \times S \]

2. Apply **selections / conditions**:
   \[ Y = \{(r, s) \in X \mid r.A = = r.B\} \]

3. Apply **projections** to get final output:
   \[ Z = (y.A,) \text{ for } y \in Y \]

Recall: Cross product \((A \times B)\) is the set of all unique tuples in \(A,B\)

Ex: \{a,b,c\} \times \{1,2\} = \{(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)\}

= Filtering!

= Returning only *some* attributes

Remembering this order is critical to understanding the output of certain queries (see later on...)
Note: we say “semantics” not “execution order”

• The preceding slides show *what a join means*

• Not actually how the DBMS executes it under the covers
A Subtlety about Joins

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

Find all countries that manufacture some product in the ‘Gadgets’ category.

SELECT Country
FROM Product, Company
WHERE Manufacturer=CName AND Category='Gadgets'
A subtlety about Joins

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PName</strong></td>
<td><strong>Cname</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>GWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>Hitachi</td>
</tr>
<tr>
<td>MultiTouch</td>
<td></td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td><strong>Stock</strong></td>
</tr>
<tr>
<td>$19</td>
<td>25</td>
</tr>
<tr>
<td>$29</td>
<td>65</td>
</tr>
<tr>
<td>$149</td>
<td>15</td>
</tr>
<tr>
<td>$203</td>
<td></td>
</tr>
<tr>
<td><strong>Category</strong></td>
<td><strong>Country</strong></td>
</tr>
<tr>
<td>Gadgets</td>
<td>USA</td>
</tr>
<tr>
<td>Gadgets</td>
<td>Japan</td>
</tr>
<tr>
<td>Photography</td>
<td></td>
</tr>
<tr>
<td>Household</td>
<td></td>
</tr>
</tbody>
</table>

SELECT Country
FROM  Product, Company
WHERE  Manufacturer=Cname
AND    Category='Gadgets'

What is the problem?
What’s the solution?
ACTIVITY: Lecture-2-3.ipynb
An Unintuitive Query

```sql
SELECT DISTINCT R.A
FROM R, S, T
WHERE R.A=S.A OR R.A=T.A
```

What does it compute?
An Unintuitive Query

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```

Computes \( R \cap (S \cup T) \)

But what if \( S = \emptyset \)?

Go back to the semantics!
An Unintuitive Query

\[ \text{SELECT DISTINCT R.A} \]
\[ \text{FROM R, S, T} \]
\[ \text{WHERE R.A=S.A OR R.A=T.A} \]

• Recall the semantics!
  1. Take cross-product
  2. Apply selections / conditions
  3. Apply projection

• If \( S = {} \), then the cross product of \( R, S, T = {} \), and the query result = {}!

Must consider semantics here.
Are there more explicit way to do set operations like this?
Lecture 3: Advanced SQL – Part I
1. Set Operators & Nested Queries
What you will learn about in this section

1. Multiset operators in SQL

2. Nested queries

3. ACTIVITY: Set operator subtleties
An Unintuitive Query

```
SELECT DISTINCT R.A
FROM    R, S, T
WHERE   R.A=S.A OR R.A=T.A
```

What does it compute?

Computes \( R \cap (S \cup T) \)

But what if \( S = \phi \)?

Go back to the semantics!
An Unintuitive Query

```
SELECT DISTINCT R.A 
FROM   R, S, T 
WHERE  R.A=S.A OR R.A=T.A
```

• Recall the semantics!
  1. Take cross-product
  2. Apply selections / conditions
  3. Apply projection

• If S = {}, then the cross product of R, S, T = {}, and the query result = {}!

Must consider semantics here. Are there more explicit way to do set operations like this?
What does this look like in Python?

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```

• Semantics:
  1. Take cross-product
  2. Apply selections / conditions
  3. Apply projection

Joins / cross-products are just nested for loops (in simplest implementation)!

If-then statements!
What does this look like in Python?

```python
output = {}
for r in R:
    for s in S:
        for t in T:
            if r['A'] == s['A'] or r['A'] == t['A']:
                output.add(r['A'])
return list(output)
```

Can you see now what happens if $S = []$?
Multiset Operations
Recall Multisets

Equivalent Representations of a Multiset

Multiset $X$

<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>

Note: In a set all counts are \{0,1\}.
Generalizing Set Operations to Multiset Operations

\[ \text{Multiset } X \]

<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>0</td>
</tr>
<tr>
<td>(2, c)</td>
<td>3</td>
</tr>
<tr>
<td>(1, d)</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \cap \]

\[ \text{Multiset } Y \]

<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(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>

\[ \text{Multiset } Z \]

<table>
<thead>
<tr>
<th>Tuple</th>
<th>( \lambda(Z) )</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>2</td>
</tr>
<tr>
<td>(1, d)</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \lambda(Z) = \min(\lambda(X), \lambda(Y)) \]

For sets, this is intersection
Generalizing Set Operations to Multiset Operations

For sets, this is union

\[ \lambda(Z) = \max(\lambda(X), \lambda(Y)) \]
Multiset Operations in SQL
Explicit Set Operators: INTERSECT

\[
\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}
\]

SELECT R.A
FROM R, S
WHERE R.A=S.A
INTERSECT
SELECT R.A
FROM R, T
WHERE R.A=T.A
UNION

\[
\begin{align*}
\text{SELECT} & \quad R.A \\
\text{FROM} & \quad R, S \\
\text{WHERE} & \quad R.A = S.A \\
\text{UNION} & \quad \\
\text{SELECT} & \quad R.A \\
\text{FROM} & \quad R, T \\
\text{WHERE} & \quad R.A = T.A
\end{align*}
\]

\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}

Why aren’t there duplicates?

What if we want duplicates?
UNION ALL

```
SELECT R.A
FROM R, S
WHERE R.A = S.A
UNION ALL
SELECT R.A
FROM R, T
WHERE R.A = T.A
```

\{r. A | r. A = s. A\} \cup \{r. A | r. A = t. A\}

ALL indicates the Multiset disjoint union operation
Generalizing Set Operations to Multiset Operations

For sets, this is disjoint union

$$\lambda(Z) = \lambda(X) + \lambda(Y)$$
EXCEPT

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

\[
\text{EXCEPT} \\
\text{SELECT } R.A \\
\text{FROM } R, T \\
\text{WHERE } R.A=T.A
\]

\[
\{r.A \mid r.A = s.A\}\backslash\{r.A\mid r.A = t.A\}
\]

What is the multiset version?

\[
\lambda(Z) = \lambda(X) - \lambda(Y)
\]

For elements that are in X
INTERSECT: Still some subtle problems...

Company(name, hq_city)
Product(pname, maker, factory_loc)

SELECT hq_city
FROM Company, Product
WHERE maker = name
AND factory_loc = 'US'
INTERSECT
SELECT hq_city
FROM Company, Product
WHERE maker = name
AND factory_loc = 'China'

“What if two companies have HQ in US: BUT one has factory in China (but not US) and vice versa? **What goes wrong?**
INTERSECT: Remember the semantics!

Example: `C JOIN P` on `maker = name`

<table>
<thead>
<tr>
<th></th>
<th>C.name</th>
<th>C.hq_city</th>
<th>P.pname</th>
<th>P.maker</th>
<th>P.factory_loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Co.</td>
<td>Seattle</td>
<td>X</td>
<td>X Co.</td>
<td>U.S.</td>
<td></td>
</tr>
<tr>
<td>Y Inc.</td>
<td>Seattle</td>
<td>X</td>
<td>Y Inc.</td>
<td>China</td>
<td></td>
</tr>
</tbody>
</table>

```
Company(name, hq_city) AS C
Product(pname, maker, factory_loc) AS P

SELECT hq_city
FROM Company, Product
WHERE maker = name
AND factory_loc = 'US'
INTERSECT
SELECT hq_city
FROM Company, Product
WHERE maker = name
AND factory_loc = 'China'
```
INTERSECT: Remember the semantics!

Company(name, hq_city) AS C
Product(pname, maker, factory_loc) AS P

Example: C JOIN P on maker = name

<table>
<thead>
<tr>
<th>C.name</th>
<th>C.hq_city</th>
<th>P.pname</th>
<th>P.maker</th>
<th>P.factory_loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Co.</td>
<td>Seattle</td>
<td>X</td>
<td>X Co.</td>
<td>U.S.</td>
</tr>
<tr>
<td>Y Inc.</td>
<td>Seattle</td>
<td>X</td>
<td>Y Inc.</td>
<td>China</td>
</tr>
</tbody>
</table>

X Co has a factory in the US (but not China)
Y Inc. has a factor in China (but not US)
But Seattle is returned by the query!

We did the INTERSECT on the wrong attributes!
One Solution: **Nested Queries**

```
Company(name, hq_city)
Product(pname, maker, factory_loc)

SELECT DISTINCT hq_city
FROM Company, Product
WHERE maker = name
AND name IN (
    SELECT maker
    FROM Product
    WHERE factory_loc = 'US'
) AND name IN (
    SELECT maker
    FROM Product
    WHERE factory_loc = 'China'
)
```

“Headquarters of companies which make gizmos in US AND China”

Note: If we hadn’t used DISTINCT here, how many copies of each hq_city would have been returned?
High-level note on nested queries

• We can do nested queries because SQL is *compositional*:
  
  • Everything (inputs / outputs) is represented as multisets - the output of one query can thus be used as the input to another (nesting)!
  
• This is *extremely* powerful!
Nested queries: Sub-queries Returning Relations

```
SELECT c.city
FROM Company c
WHERE c.name IN (
    SELECT pr.maker
    FROM Purchase p, Product pr
    WHERE p.product = pr.name
    AND p.buyer = 'Joe Blow'
)
```

“Cities where one can find companies that manufacture products bought by Joe Blow”

Another example:

- Company(name, city)
- Product(name, maker)
- Purchase(id, product, buyer)
Nested Queries

Is this query equivalent?

```
SELECT c.city
FROM Company c,
     Product pr,
     Purchase p
WHERE c.name = pr.maker
  AND pr.name = p.product
  AND p.buyer = 'Joe Blow'
```

Beware of duplicates!
Nested Queries

\[
\begin{align*}
&\text{SELECT DISTINCT c.city} \\
&\text{FROM Company c, Product pr, Purchase p} \\
&\text{WHERE c.name = pr.maker} \\
&\quad \text{AND pr.name = p.product} \\
&\quad \text{AND p.buyer = 'Joe Blow'}
\end{align*}
\]

\[
\begin{align*}
&\text{SELECT DISTINCT c.city} \\
&\text{FROM Company c} \\
&\text{WHERE c.name \ IN (} \\
&\quad \text{SELECT pr.maker} \\
&\quad \text{FROM Purchase p, Product pr} \\
&\quad \text{WHERE p.product = pr.name} \\
&\quad \quad \text{AND p.buyer = 'Joe Blow'})
\end{align*}
\]

Now they are equivalent
Subqueries Returning Relations

You can also use operations of the form:

- \( s > \text{ALL} \ R \)
- \( s < \text{ANY} \ R \)
- \( \text{EXISTS} \ R \)

Ex:

Product(name, price, category, maker)

```
SELECT name
FROM Product
WHERE price > ALL(
    SELECT price
    FROM Product
    WHERE maker = 'Gizmo-Works')
```

Find products that are more expensive than all those produced by "Gizmo-Works".

ANY and ALL not supported by SQLite.
Subqueries Returning Relations

You can also use operations of the form:

- \( s > \text{ALL } R \)
- \( s < \text{ANY } R \)
- \( \text{EXISTS } R \)

Ex: \( \text{Product(name, price, category, maker)} \)

```
SELECT p1.name
FROM Product p1
WHERE p1.maker = 'Gizmo-Works'
     AND \( \text{EXISTS}(\)
         SELECT p2.name
         FROM Product p2
         WHERE p2.maker <> 'Gizmo-Works'
             AND p1.name = p2.name)
```

Find `copycat` products, i.e. products made by competitors with the same names as products made by “Gizmo-Works”

\( <> \) means \(!=\)
Nested queries as alternatives to INTERSECT and EXCEPT

\[
(\text{SELECT } R.A, R.B \\
\text{ FROM } R) \ \text{INTERSECT} \\
(\text{SELECT } S.A, S.B \\
\text{ FROM } S)
\]

\[
\text{SELECT } R.A, R.B \\
\text{ FROM } R \\
\text{WHERE EXISTS(} \\
(\text{SELECT } * \\
\text{ FROM } S \\
\text{ WHERE R.A=S.A AND R.B=S.B})
\]

\[
(\text{SELECT } R.A, R.B \\
\text{ FROM } R) \ \text{EXCEPT} \\
(\text{SELECT } S.A, S.B \\
\text{ FROM } S)
\]

\[
\text{SELECT } R.A, R.B \\
\text{ FROM } R \\
\text{WHERE NOT EXISTS(} \\
(\text{SELECT } * \\
\text{ FROM } S \\
\text{ WHERE R.A=S.A AND R.B=S.B})
\]

INTERSECT and EXCEPT not in some DBMSs!

If R, S have no duplicates, then can write without sub-queries (HOW?)
Correlated Queries

SELECT DISTINCT title
FROM Movie AS m
WHERE year <> ANY(
    SELECT year
    FROM Movie
    WHERE title = m.title)

Find movies whose title appears more than once.

Note the scoping of the variables!

Note also: this can still be expressed as single SFW query...
Complex Correlated Query

```
SELECT DISTINCT x.name, x.maker
FROM Product AS x
WHERE x.price > ALL(
    SELECT y.price
    FROM Product AS y
    WHERE x.maker = y.maker
    AND y.year < 1972)
```

Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972.

Can be very powerful (also much harder to optimize)
Activity-3-1.ipynb
Basic SQL Summary

• SQL provides a high-level declarative language for manipulating data (DML)

• The workhorse is the SFW block

• Set operators are powerful but have some subtleties

• Powerful, nested queries also allowed.