Announcements

- Webquiz 2 is due on Sunday
  - Webquiz 3 will go out
- Homework 2 is due on Wednesday
  - Homework 3 uses Microsoft Azure Cloud services
    - (no more sqlite!)
  - Look for instructions on setting up your Azure Account next week.
    - Even if you already have a Microsoft Account you will create a new one for this class
Lecture Goals

- Today we will learn how to write (even) more powerful SQL queries

- Reading: Ch. 6.3
```
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON
Product.name = Purchase.prodName
```

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

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
</tr>
<tr>
<td>Camera</td>
</tr>
</tbody>
</table>
SELECT Product.name, COUNT(*)
FROM Product JOIN Purchase ON
Product.name = Purchase.prodName
GROUP BY Product.name

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

What Changes?

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
Projecting Columns with Grouping

**SQL Query 1:**
```
SELECT product, max(quantity)
FROM Purchase
GROUP BY product
```

**SQL Query 2:**
```
SELECT product, quantity
FROM Purchase
GROUP BY product
```

**Table:**

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.5</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

**Message:**
Can’t project a non-grouped / non-aggregated column!
Subqueries

- A subquery is a SQL query nested inside a larger query.
- Such inner-outer queries are called nested queries.
- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause

- Rule of thumb: avoid writing nested queries when possible.
  - But sometimes it's impossible, as we will see.
SQL Subqueries

Just because you can use them doesn’t mean you should.
Subqueries…

• Can appear as computed values in a SELECT clause

• Can appear in FROM clauses and aliased using a tuple variable that represents the tuples in the result of the subquery

• Can return a single constant to be compared with another value in a WHERE clause
• Can return relations to be used in WHERE clauses
1. Subqueries in SELECT

Product (pname, price, cid)  
Company (cid, cname, city)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city
    FROM Company Y
    WHERE Y.cid=X.cid) as City
FROM   Product X
```

What happens if the subquery returns more than one city?
We get a runtime error
... as usual, SQLite simply ignores the extra values
1. Subqueries in SELECT

Product \((\text{pname}, \text{price}, \text{cid})\)
Company \((\text{cid}, \text{cname}, \text{city})\)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city 
  FROM Company Y 
  WHERE Y.cid=X.cid) as City 
FROM Product X
```

**Correlated Subquery:** a sub-query that uses values from the outer query. In this case the inner query has to be executed for every row of outer query.
Product \((p\text{name}, \text{price}, \text{cid})\)
Company \((\text{cid}, \text{c\text{name}}, \text{city})\)

1. Subqueries in SELECT

Whenever possible, don’t use a nested queries:

\[
\text{SELECT } X.p\text{name}, \ (\text{SELECT } Y.\text{city} \\
\text{FROM } \text{Company } Y \\
\text{WHERE } Y.\text{cid}=X.\text{cid}) \text{ as City} \\
\text{FROM } \text{Product } X
\]

CSE 344 - Winter 2017 12
Product (pname, price, cid)
Company (cid, cname, city)

1. Subqueries in SELECT

Whenever possible, don’t use a nested queries:

```
SELECT X.pname, (SELECT Y.city
                FROM Company Y
                WHERE Y.cid=X.cid) as City
FROM   Product X
```

```
SELECT X.pname, Y.city
FROM   Product X, Company Y
WHERE  X.cid=Y.cid
```

We have “unnested” the query
Product \((\text{pname}, \text{price}, \text{cid})\)
Company \((\text{cid}, \text{cname}, \text{city})\)

1. Subqueries in SELECT

Compute the number of products made by each company

\[
\begin{align*}
\text{SELECT DISTINCT C.cname, (SELECT count(*) FROM Product P WHERE P.cid=C.cid)} \\
\text{FROM Company C}
\end{align*}
\]
1. Subqueries in SELECT

Compute the number of products made by each company

```
SELECT DISTINCT C.cname, (SELECT count(*)
  FROM Product P
  WHERE P.cid=C.cid)
FROM Company C
```

Better: we can unnest using a
GROUP BY

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```
Product (\textit{pname}, price, cid)
Company (\textit{cid}, cname, city)

1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
FROM Product P
WHERE P.cid=C.cid)
FROM Company C
```

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```
Product \((p\text{name}, \ price, \ cid)\)
Company \((cid, \ c\text{name}, \ c\text{ity})\)

1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
  FROM Product P
  WHERE P.cid=C.cid)
FROM Company C
```

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```

```
SELECT C.cname, count(p\text{name})
FROM Company C LEFT OUTER JOIN Product P
ON C.cid=P.cid
GROUP BY C.cname
```

No! Different results if a company has no products.
Product \((\text{pname, price, cid})\)
Company \((\text{cid, cname, city})\)

2. Subqueries in FROM

Find all products whose prices is > 20 and < 500

\[
\text{SELECT X.pname} \\
\text{FROM (SELECT *} \\
\text{\quad \text{FROM Product AS Y}} \\
\text{\quad \text{WHERE price > 20) as X}} \\
\text{\quad \text{WHERE X.price < 500}}
\]

Side note: This is not a correlated subquery. (why?)

Try unnest this query!
2. Subqueries in FROM

Use the result of the inner query as a new table in the FROM clause.

- We will see that sometimes we really need a subquery
  - will see most compelling examples next lecture
  - in that case, we can put it in the FROM clause
Product (\texttt{pname}, \texttt{price}, \texttt{cid})
Company (\texttt{cid}, \texttt{cname}, \texttt{city})

3. Subqueries in WHERE

Find all companies that make \textbf{some} products with price < 100

Existential quantifiers: there exists an \textit{x} such that \( P(x) \)

\[
\exists x \in X \ P(x)
\]

Useful Keyword: \textbf{EXISTS}, \textbf{IN}, \textbf{ANY}, \textbf{ALL}

Using \textbf{EXISTS}:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT *
               FROM Product P
               WHERE C.cid = P.cid and P.price < 100)
\end{verbatim}
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using **IN**

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
FROM Product P
WHERE P.price < 100)
```
3. Subqueries in WHERE

Find all companies that make **some** products with price < 200

Existential quantifiers

Using **ANY**:  

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
                 FROM Product P
                 WHERE P.cid = C.cid)
```

Not supported in sqlite
Product \((p\text{name}, \ price, \ cid)\)
Company \((c\text{id}, \ c\text{name}, \ c\text{ity})\)

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Now let's unnest it:

```
SELECT DISTINCT  C.cname
FROM    Company C, Product P
WHERE   C.cid = P.cid and P.price < 200
```

Existential quantifiers are easy!
3. Subqueries in WHERE

Find all companies where all their products have price < 100

Same as:

Find all companies that make only products with price < 100

Universal quantifiers: for all \( x \), \( P(x) \) holds

\[ \forall x \in X \ P(x) \]
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

Find all companies where all their products have price < 100

Step 1: Find the other companies: i.e. with some product >= 100

```
SELECT DISTINCT  C.cname
FROM   Company C
WHERE  C.cid IN (SELECT P.cid
                 FROM   Product P
                 WHERE  P.price >= 100)
```

Step 2: Find all companies where all their products have price < 100

```
SELECT DISTINCT  C.cname
FROM   Company C
WHERE  C.cid NOT IN (SELECT P.cid
                      FROM   Product P
                      WHERE  P.price >= 100)
```
Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies where all their products have price < 100

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
    FROM Product P
    WHERE P.cid = C.cid and P.price >= 100)
```
3. Subqueries in WHERE

Find all companies where all their products have price < 100

Using **ALL**:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 100 >= ALL (
    SELECT price
    FROM Product P
    WHERE P.cid = C.cid
)
```
Question for Database Theory
Fans and their Friends

• Can we unnest the *universal quantifier* query?

• We need to first discuss the concept of
  *monotonicity*
Product \((p\text{name}, \ p\text{rice}, \ c\text{id})\)
Company \((c\text{id}, \ c\text{name}, \ c\text{ity})\)

**Monotone Queries**

- **Definition:** A query \(Q\) is **monotone** if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

- **Similar to Monotone Functions**

---

**Figure 1.** A monotonically increasing function. It is strictly increasing on the left and right while just **monotonic** (unchanging) in the middle.

**Figure 2.** A monotonically decreasing function

**Figure 3.** A function that is not monotonic
Product \((pname, \ price, \ cid)\)
Company \((cid, \ cname, \ city)\)

**Monotone Queries**

- Definition: A query \(Q\) is **monotone** if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>(pname)</td>
<td>(cid)</td>
</tr>
<tr>
<td>Gizmo</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>(pname)</td>
<td>(cid)</td>
</tr>
<tr>
<td>Gizmo</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
<tr>
<td>iPad</td>
<td>c001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>(pname)</td>
<td>(cid)</td>
</tr>
<tr>
<td>Gizmo</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
<tr>
<td>iPad</td>
<td>c001</td>
</tr>
</tbody>
</table>
Monotone Queries

- **Theorem:** If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.
Monotone Queries

- **Theorem:** If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

- **Proof.** We use the nested loop semantics: if we insert a tuple in a relation $R_i$, this will not remove any tuples from the answer.

```plaintext
SELECT $a_1, a_2, \ldots, a_k$
FROM $R_1$ AS $x_1, R_2$ AS $x_2, \ldots, R_n$ AS $x_n$
WHERE Conditions
```

```plaintext
for $x_1$ in $R_1$ do
  for $x_2$ in $R_2$ do
    \ldots
    for $x_n$ in $R_n$ do
      if Conditions
        output ($a_1, \ldots, a_k$)
```
Monotone Queries

- The query:

Find all companies where all their products have price < 100

is not monotone

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>999.99</td>
<td>c001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c001</td>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
</tbody>
</table>

Consequence: we cannot write it as a SELECT-FROM-WHERE query without nested subqueries
Queries that must be nested

- Queries with universal quantifiers or with negation

- Queries that use aggregates in certain ways
  - \( \text{sum}(...) \) and \( \text{count}(*) \) are NOT monotone, because they do not satisfy set containment
  - \( \text{select count}(* \text{ from R}) \) is not monotone!

That is, cannot be SFW queries
SQLite SELECT

https://sqlite.org/lang_select.html