Announcements

- Should now have seats for all registered

Outline

- Inner joins (6.2, review)
- Outer joins (6.3.8)
- Aggregations (6.4.3 – 6.4.6)

UNIQUE

- PRIMARY KEY adds implicit "NOT NULL" constraint while UNIQUE does not
  - you would have to add this explicitly for UNIQUE:
    
    ```
    CREATE TABLE Company(
      name VARCHAR(20) NOT NULL, ...
    UNIQUE (name));
    ```
  - You almost always want to do this (in real schemas)
    - SQL Server behaves strangely with NULL & UNIQUE
    - otherwise, think through NULL for every query
    - you can remove the NOT NULL constraint later

(INNER) Joins

```
SELECT a1, a2, ..., an
FROM R1, R2, ..., Rm
WHERE Cond
```

(Nested loop semantics)

```python
for t1 in R1:
  for t2 in R2:
    ...
    for tm in Rm:
      if Cond(t1.a1, t1.a2, ...):
        output(t1.a1, t1.a2, ..., tm.an)
```

(INNER) joins

```
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget' AND manufacturer = cname
```

Company(cname, country)
Product(pname, price, category, manufacturer) — manufacturer is foreign key
**Inner joins**

```sql
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget' AND
    manufacturer = cname
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>category</td>
</tr>
<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>

Not output because country != 'USA'
(also cname != manufacturer)

```sql
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget' AND
    manufacturer = cname
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>category</td>
</tr>
<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>

Not output because country != 'USA'

```sql
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget' AND
    manufacturer = cname
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>category</td>
</tr>
<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>

Not output because category != 'gadget' (and ...)

```sql
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget' AND
    manufacturer = cname
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>category</td>
</tr>
<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>

Not output because category != 'gadget'
### (Inner) joins

**Original SQL Query:**
```
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget' AND manufacturer = cname
```

**Result:**
```
pname | category | manufacturer
-------|----------|-----------------
Gizmo  | gadget   | GizmoWorks      
Camera | Photo    | Hitachi         
OneClick| Photo    | Hitachi         
```

**Notes:**
- Not output because category != 'gadget'

**Restricting category = 'gadget':**
```
SELECT DISTINCT cname
FROM Product JOIN Company ON
     country = 'USA' AND category = 'gadget' AND manufacturer = cname
```

**Result:**
```
pcname | category | manufacturer  
-------|----------|--------------
Gizmo   | gadget   | GizmoWorks   
Camera  | Photo    | Hitachi      
OneClick| Photo    | Hitachi      
```

**Restricting country = 'USA':**
```
SELECT DISTINCT cname
FROM Product (where category = 'gadget') JOIN Company (where country = 'USA')
```

**Result:**
```
pcname | category | manufacturer  
-------|----------|--------------
Gizmo   | gadget   | GizmoWorks   
```

**Alternative Syntax:**
```
SELECT DISTINCT cname
FROM Product JOIN Company ON
     country = 'USA' AND category = 'gadget' AND manufacturer = cname
```

**Notes:**
- Emphasizes that the predicate is part of the join.
- Now only one combination to consider
- (Query optimizers do this too.)
Self-Joins and Tuple Variables

- **Ex:** find companies that manufacture both products in the 'gadgets' category and in the 'photo' category
- Just joining Company with Product is insufficient; need to join Company with Product with Product
- When a relation occurs twice in the FROM clause we call it a **self-join**; in that case every column name in Product is ambiguous (why?)
  - are you referring to the tuple in the 2nd or 3rd loop?

Name Conflicts

- When a name is ambiguous, qualify it:
  
  ```sql
  WHERE Company.name = Product.name AND ...
  ```
- For self-join, we need to distinguish tables:
  
  ```sql
  FROM Product x, Product y, Company
  ```
- These new names are called “tuple variables”
  - can think of as name for the variable of each loop
  - can also write “Company AS C” etc.
  - can make SQL query shorter: C.name vs. Company.name

**Self-joins**

```sql
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
AND x.category = 'gadget'
AND y.category = 'photo'
AND x.manufacturer = cname
AND y.manufacturer = cname;
```

**Self-joins**

```sql
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
AND x.category = 'gadget'
AND y.category = 'photo'
AND x.manufacturer = cname
AND y.manufacturer = cname;
```

**Not output because y.category != 'photo'**

**Self-joins**

```sql
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
AND x.category = 'gadget'
AND y.category = 'photo'
AND x.manufacturer = cname
AND y.manufacturer = cname;
```

**Self-joins**

```sql
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
AND x.category = 'gadget'
AND y.category = 'photo'
AND x.manufacturer = cname
AND y.manufacturer = cname;
```
Self-joins

SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
AND x.category = 'gadget'
AND y.category = 'photo'
AND x.manufacturer = cname
AND y.manufacturer = cname;

Not output because y.manufacturer != cname

Outer joins

Product(name, category)
Purchase(prodName, store) -- prodName is foreign key

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

Or equivalently:

SELECT Product.name, ..., Purchase.store
FROM Product JOIN Purchase ON
Product.name = Purchase.prodName

But some Products may not be not listed. Why?
### Outer Joins

- **Left outer join:**
  - Include the left tuple even if there's no match
- **Right outer join:**
  - Include the right tuple even if there's no match
- **Full outer join:**
  - Include both left and right tuples even if there's no match

- (Also something called a **UNION JOIN**, though it's rarely used.)
- (Actually, all of these are used much more rarely than inner joins.)
Outer Joins Example

See lec04-sql-outer-joins.sql...

Aggregation in SQL

```sql
>sqlite3 lecture04
sqlite> create table Purchase(
    pid int primary key,
    product text,
    price float,
    quantity int,
    month varchar(15));
sqlite> -- download data.txt
sqlite> .import lec04-data.txt Purchase
```

Comment about SQLite

- One cannot load NULL values such that they are actually loaded as null values
- So we need to use two steps:
  - Load null values using some type of special value
  - Update the special values to actual null values
```sql
update Purchase
    set price = null
where price = 'null'
```

Simple Aggregations

Five basic aggregate operations in SQL

```sql
select count(*) from Purchase
select sum(quantity) from Purchase
select avg(price) from Purchase
select max(quantity) from Purchase
select min(quantity) from Purchase
```

Aggregates and NULL Values

Null values are not used in aggregates
```sql
insert into Purchase
    values(12, 'gadget', NULL, NULL, 'april')
```

Let's try the following
```sql
select count(*) from Purchase
select count(quantity) from Purchase
select sum(quantity) from Purchase
select count(quantity) from Purchase
where quantity is not null;
-- "is not null" is redundant
```

Aggregates and NULL Values

Null values are not used in aggregates
```sql
insert into Purchase
    values(12, 'gadget', NULL, NULL, 'april')
```

Let's try the following
```sql
select count(*) from Purchase
    -- NULL is counted in count(*)
select count(quantity) from Purchase
    -- NULL is ignored in count(quantity)
select sum(quantity) from Purchase
select sum(quantity)
from Purchase
where quantity is not null;
```

Other DBMSs have other ways of importing data
Counting Duplicates

COUNT applies to duplicates, unless otherwise stated:

```
SELECT Count(product)  
FROM Purchase         
WHERE price > 4.99    
```

same as Count(*) if no nulls

We probably want:

```
SELECT Count(DISTINCT product)  
FROM Purchase       
WHERE price > 4.99   
```

More Examples

```
SELECT Sum(price * quantity)  
FROM Purchase          
WHERE product = 'bagel'  
```

Simple Aggregations

```
SELECT Sum(price * quantity)  
FROM Purchase                  
WHERE product = 'bagel'   
```

```
SELECT Sum(price * quantity) / sum(quantity)  
FROM Purchase                  
WHERE product = 'bagel'   
```

More Examples

```
SELECT sum(price * quantity) / sum(quantity)  
FROM Purchase                  
WHERE product = 'bagel'   
```

What happens if there are NULLs in price or quantity?

Lesson: disallow NULLs unless you need to handle them.