CSE 344

JUNE 22\textsuperscript{ND}

INTRODUCTION TO JOINS

(2.1-2.3 & 6.1-6.2)
REVIEW

• Data model gives languages for
  • describing schema (what data is allowed in the DB)
  • writing queries (asking questions & updating data)

• Relational data model
  • database is a collection of tables
  • schema describes each table
    • name of table and columns
    • types of all columns
  • query language (SQL for now)
    • insert, remove, and print rows of table
    • more to come…
ADMINISTRIVIA

• Should have access to your gitlab repository
• HW1 starter code is there
  • Fill in the .sql files
  • Use test script to check that it works
    • (we will test more thoroughly)
  • Commit, tag, and push to gitlab to turn it in
DEMO 1

• What operations should we expect SQLite (or any DBMS) to support just on what we know right now?
  • create table
  • insert into
  • select
  • delete from
• What sorts of inputs do these functions need to have?
  • create table: table name, schema
  • insert into: table name, tuple
  • select: table name, attributes
  • delete from: table name, condition
**DEMO 1**

- **Common Syntax**
  - `CREATE TABLE [tablename]`  
    `([att1] [type1],`  
    `[att2] [type2]...)`;
  - `INSERT INTO [tablename] VALUES ([val1],[val2]...)`;
  - `SELECT [att1],[att2],... FROM [tablename]`  
    `WHERE [condition]`;
  - `DELETE FROM [tablename]`  
    `WHERE [condition]`
DEMO 1
DISCUSSION

• Two other operations we want to support
  • ALTER TABLE: Adds a new attribute to the table
  • UPDATE: Change the attribute for a particular tuple in the table (rather than insert/delete)

• Common Syntax
  • ALTER TABLE [tablename] ADD [attname] [atttype]
  • UPDATE [tablename] SET [attname]=[value]
DISCUSSION

• Two other operations we want to support
  • ALTER TABLE: Adds a new attribute to the table
  • UPDATE: Change the attribute for a particular tuple in the table (rather than insert/delete)

• Common Syntax
  • ALTER TABLE [tablename] ADD [attname] [atttype]
  • UPDATE [tablename] SET [attname]= [value]
    WHERE [condition]
DEMO 2
DISCUSSION

Tables are NOT ordered

- they are sets or multisets (bags)

Tables are FLAT

- No nested attributes

Tables DO NOT prescribe how they are stored on disk

- This is called physical data independence

All three allow DBMSs to be more efficient.

(Last one also simplifies application development.)
DISCUSSION

• Tables may not be ordered, but data can be returned in an order with the ORDER BY modifier
DISCUSSION

• Tables may not be ordered, but data can be returned in an order with the ORDER BY modifier

• Whew, today’s been a lot of coding... I know what you’re thinking...
THEORY BREAK
THEORY BREAK

• We can think of accessing information through queries as some combination of functions
THEORY BREAK

• We can think of accessing information through queries as some combination of functions
  • Consider a table of UW students (with all relevant info):
• We can think of accessing information through queries as some combination of functions

  • Consider a table of UW students (with all relevant info):
    • How would we need to get the birth year of all CSE students from California?
THEORY BREAK

• We can think of accessing information through queries as some combination of functions
  • Consider a table of UW students (with all relevant info):
    • How would we need to get the birth year of all CSE students from California?
    • *Think of the file as a set of tuples*
THEORY BREAK

• We can think of accessing information through queries as some combination of functions
  • Consider a table of UW students (with all relevant info):
    • How would we need to get the birth year of all CSE students from California?
    • *Think of the file as a set of tuples*
    • Find the set of CSE students and the set of students from California; Find the intersection of these sets, return just the year from the birthday values of this set
THEORY BREAK

• We can think of accessing information through queries as some combination of functions
  • Consider a table of UW students (with all relevant info):
    • How would we need to get the birth year of all CSE students from California?
    • *Think of the file as a set of tuples*
    • Find the set of CSE students and the set of students from California; Find the intersection of these sets, return just the year from the birthday values of this set
    • *What does this return?*
• We can think of accessing information through queries as some combination of functions
  • Consider a table of UW students (with all relevant info):
    • How would we need to get the birth year of all CSE students from California?
    • *Think of the file as a set of tuples*
    • Find the set of CSE students and the set of students from California; Find the intersection of these sets, return just the year from the birthday values of this set
    • *What does this return?*
    • Years, but with many duplicates. Even though sets don’t allow duplicates, the objects are unique.
• If we only want to return unique elements, we can use the DISTINCT modifier
  • Even if we hide some attributes from the output, the data is all still there.
  • When we select a subset of the attributes, this function is called a projection
    • projections usually produce duplicate values
    • takes work to remove them, so DBMSs usually leave them
    • except on disk, DBMSs work with multisets not sets
THEORY BREAK

• This was all for a single table.
• Data models specify how our data are stored and how the data are related
• Need to utilize these relations, or the database was pointless
• This involves a JOIN
THEORY BREAK

• This was all for a single table.
• Data models specify how our data are stored and how the data are related
• Need to utilize these relations, or the database was pointless
• This involves a JOIN
  • 1NF makes us split up data that belongs together, so query language must make it easy to put them back together whenever necessary
  • we do this with joins
JOIN: INTRO

• The JOIN is the way we use the relationships between tables in a query
  • Example: if we want all of the products and their relevant company information, we need to join those two tables.
  • The result of the join is all of the relevant information from both tables
    • Join occurs based on the join condition.
  • By default, join produces every combination of tuples from the two tables as a row
    • join condition allows you to restrict to the combinations that make sense
    • DBMSs are very good at joining efficiently
Retrieve all Japanese products that cost < $150

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>
Retrieve all Japanese products that cost < $150

```
SELECT pname, price
FROM   Product, Company
WHERE  ...
```
JOINS IN SQL

Given the following tables:

Product(pname, price, category, manufacturer)
Company(cname, country)

Write a SQL query to retrieve all Japanese products that cost less than $150.

```sql
SELECT pname, price
FROM Product, Company
WHERE manufacturer=cname AND country='Japan' AND price < 150
```
Retrieve all Japanese products that cost < $150

```sql
SELECT pname, price
FROM Product
JOIN Company
WHERE manufacturer=cname AND country='Japan' AND price < 150
```
Retrieve all Japanese products that cost less than $150.

```
SELECT pname, price
FROM Product
JOIN Company ON manufacturer = cname
WHERE country = 'Japan' AND price < 150
```
JOINS IN SQL

Product(pname, price, category, manufacturer)
Company(cname, country)

Retrieve all Japanese products that cost < $150

\[
\begin{align*}
\text{SELECT} & \quad P.pname, P.price \\
\text{FROM} & \quad \text{Product P, Company C} \\
\text{WHERE} & \quad P.manufacturer = C.cname \text{ AND } \\
& \quad C.country = 'Japan' \text{ AND } P.price < 150
\end{align*}
\]

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>

P and C are called “tuple variables”
JOINS IN SQL

Retrieve all USA companies that manufacture "gadget" products

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>
JOINS IN SQL

Product(pname, price, category, manufacturer)
Company(cname, country)

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>

Retrieve all USA companies that manufacture "gadget" products

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

Why DISTINCT?
JOINS IN SQL

The standard join in SQL is called an inner join

• Each row in the result must come from both tables in the join

Sometimes we want to include rows from only one of the two table: outer join
Employee(id, name)
Sales(employeeID, productID)

INNER JOIN

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Jack</td>
</tr>
<tr>
<td>3</td>
<td>Jill</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>employeeID</th>
<th>productID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>344</td>
</tr>
<tr>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>2</td>
<td>544</td>
</tr>
</tbody>
</table>

Retrieve employees and their sales
Employee(id, name)
Sales(employeeID, productID)

INNER JOIN

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Jack</td>
</tr>
<tr>
<td>3</td>
<td>Jill</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>employeeID</th>
<th>productID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>344</td>
</tr>
<tr>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>2</td>
<td>544</td>
</tr>
</tbody>
</table>

Retrieve employees and their sales

```
SELECT * 
FROM   Employee E, Sales S 
WHERE  E.id = S.employeeID 
```
INNER JOIN

**Employee**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Jack</td>
</tr>
<tr>
<td>3</td>
<td>Jill</td>
</tr>
</tbody>
</table>

**Sales**

<table>
<thead>
<tr>
<th>employeeID</th>
<th>productID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>344</td>
</tr>
<tr>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>2</td>
<td>544</td>
</tr>
</tbody>
</table>

Retrieve employees and their sales

```sql
SELECT * 
FROM Employee E, Sales S 
WHERE E.id = S.employeeID
```
Employee(id, name)
Sales(employeeID, productID)

INNER JOIN

Employee

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Jack</td>
</tr>
<tr>
<td>3</td>
<td>Jill</td>
</tr>
</tbody>
</table>

Sales

<table>
<thead>
<tr>
<th>employeeID</th>
<th>productID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>344</td>
</tr>
<tr>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>2</td>
<td>544</td>
</tr>
</tbody>
</table>

Jill is missing

Retrieve employees and their sales

```
SELECT *
FROM   Employee E, Sales S
WHERE  E.id = S.employeeID
```
Employee(id, name)
Sales(employeeID, productID)

INNER JOIN

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Jack</td>
</tr>
<tr>
<td>3</td>
<td>Jill</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>employeeID</th>
<th>productID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>344</td>
</tr>
<tr>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>2</td>
<td>544</td>
</tr>
</tbody>
</table>

Retrieve employees and their sales

SELECT *
FROM Employee E
INNER JOIN Sales S
ON E.id = S.employeeID

Jill is missing

Jill is missing
Employee(id, name)
Sales(employeeID, productID)

OUTER JOIN

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Jack</td>
</tr>
<tr>
<td>3</td>
<td>Jill</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>employeeID</th>
<th>productID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>344</td>
</tr>
<tr>
<td>1</td>
<td>355</td>
</tr>
<tr>
<td>2</td>
<td>544</td>
</tr>
</tbody>
</table>

Jill is present

Retrieve employees and their sales

```
SELECT * 
FROM Employee E 
LEFT OUTER JOIN Sales S 
ON E.id = S.employeeID
```
INNER) JOINS

Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company

SELECT DISTINCT cname
FROM Product, Company
WHERE country='USA' AND category = 'gadget'
AND manufacturer = cname
(INNER) JOINs

```
SELECT DISTINCT cname
FROM    Product, Company
WHERE   country='USA' AND category = 'gadget'
        AND manufacturer = cname
```
### (INNER) JOINS

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

#### Product

<table>
<thead>
<tr>
<th>pname</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Hitachi</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

#### Company

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>
(INNER) JOINS

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

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

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

<table>
<thead>
<tr>
<th>Pname</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Hitachi</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cname</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>
(INNER) JOINS

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

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

SELECT DISTINCT cname
FROM Product
JOIN Company ON country = 'USA' AND category = 'gadget'
    AND manufacturer = cname
(INNER) JOINS

```sql
SELECT  x1.a1, x2.a2, ... xm.am
FROM    R1 as x1, R2 as x2, ... Rm as xm
WHERE   Cond
```

for x1 in R1:
    for x2 in R2:
        ...
        for xm in Rm:
            if Cond(x1, x2...):
                output(x1.a1, x2.a2, ... xm.am)

This is called nested loop semantics since we are interpreting what a join means using a nested loop
ANOTHER EXAMPLE

Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company

Retrieve all USA companies that manufacture products in both ‘gadget’ and ‘photography’ categories
ANOTHER EXAMPLE

Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company

Retrieve all USA companies that manufacture products in both ‘gadget’ and ‘photography’ categories

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

Does this work?
ANOTHER EXAMPLE

Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company

Retrieve all USA companies that manufacture products in both ‘gadget’ and ‘photography’ categories

SELECT DISTINCT z.cname
FROM Product x, Company z
WHERE z.country = 'USA'
    AND x.manufacturer = z.cname
    AND (x.category = 'gadget'
        OR x.category = 'photography');

What about this?
ANOTHER EXAMPLE

Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company

Retrieve all USA companies that manufacture products in both ‘gadget’ and ‘photography’ categories

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

Need to include Product twice!
SELF-JOINS AND TUPLE VARIABLES

Find USA companies that manufacture both products in the ‘gadgets’ and ‘photo’ category

Joining Product with Company is insufficient: need to join Product, with Product, and with Company

When a relation occurs twice in the FROM clause we call it a self-join; in that case we must use tuple variables (why?)
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 = z.cname
    AND y.manufacturer = z.cname;
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 = z.cname
    AND y.manufacturer = z.cname;

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cname</td>
</tr>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>Hitachi</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>
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 = z.cname
AND y.manufacturer = z.cname;
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 = z.cname
    AND y.manufacturer = z.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 = z.cname
    AND y.manufacturer = z.cname;
```
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 = z.cname
AND y.manufacturer = z.cname;
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 = z.cname
    AND y.manufacturer = z.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 = z.cname
    AND y.manufacturer = z.cname;
```
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 = z.cname
  AND y.manufacturer = z.cname;
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 = z.cname
       AND y.manufacturer = z.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
```

We want to include products that are never sold, but some are not listed! Why?
OUTER JOINS

Product(name, category)
Purchase(prodName, store)

-- prodName is foreign key

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

<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>

### Purchase

<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>
### Database Query

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

---

**Product**

<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>

**Purchase**

<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>
### SQL Query

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

### Product Table

<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>

### Purchase Table

<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

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName

<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>

<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>
</tbody>
</table>
```sql
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName
```
### Product

<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>

### Purchase

<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>
</tbody>
</table>
```
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName
```
```sql
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName
```
### Product

<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>

### Purchase

<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>

```sql
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName
```
```
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```
### Product

<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>

### Purchase

<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>
<tr>
<td>OneClick</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```sql
SELECT Product.name, Purchase.store
FROM Product FULL OUTER JOIN Purchase ON
Product.name = Purchase.prodName
```
OUTER JOINS

\[
\text{tableA (LEFT/RIGHT/FULL) OUTER JOIN tableB ON p}
\]

Left outer join:
- Include tuples from `tableA` even if no match

Right outer join:
- Include tuples from `tableB` even if no match

Full outer join:
- Include tuples from both even if no match

In all cases:
- Patch tuples without matches using NULL