CSC 261/461 – Database Systems
Lecture 3

Fall 2017
1. Start forming project team.

2. Set up Jupyter Notebook.
   – I have posted the directions on Piazza.

3. Project 1 (part 1) will be released soon.

4. We will post a problem set this week.
   – Useful for Quiz 2
Today’s Lecture

1. Single Table Query
2. Multi Table Query
3. Aggregation & GROUP BY
4. Advanced SQL
SINGLE-TABLE QUERIES
What you will learn about in this section

1. The SFW query

2. Other useful operators: LIKE, DISTINCT, ORDER BY

3. ACTIVITY: Single-table queries
SQL Query

- Basic form (there are many many more bells and whistles)

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Call this a **SFW** query.
**Selection** is the operation of filtering a relation’s tuples on some condition.

```sql
SELECT * 
FROM  Product 
WHERE Category = 'Gadgets'
```

<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>

**Simple SQL Query: Selection**

The SQL query selects all columns (*) from the `Product` table where the `Category` is 'Gadgets'. The result is a table containing only the products in the Gadgets category.
**Projection** is the operation of producing an output table with tuples that have a subset of their prior attributes.

Simple SQL Query: Projection

<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>

**SELECT** Pname, Price, Manufacturer
**FROM** Product
**WHERE** Category = ‘Gadgets’
Notation

Input schema

```sql
SELECT Pname, Price, Manufacturer
FROM Product
WHERE Category = 'Gadgets'
```

Output schema

```
Answer(PName, Price, Manufacturer)
```
A Few Details

• SQL commands are case insensitive:
  – Same: SELECT, Select, select
  – Same: Product, product

• Values are not:
  – Different: ‘Seattle’, ‘seattle’

• Use single quotes for constants:
  – ‘abc’ - yes
  – “abc” - no
LIKE: Simple String Pattern Matching

- \texttt{LIKE} \ p: pattern matching on strings
- \ p \ may \ contain \ two \ special \ symbols:
  - \% = any sequence of characters
  - \_ = any single character

\begin{verbatim}
SELECT * FROM Products WHERE PName LIKE '%%gizmo%%'
\end{verbatim}
DISTINCT: Eliminating Duplicates

SELECT DISTINCT Category
FROM Product

Versus

SELECT Category
FROM Product

Gadgets
Photography
Household

Gadgets
Photography
Household
ORDER BY: Sorting the Results

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Category='gizmo' AND Price > 50
ORDER BY Price, PName
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.
3. MULTI-TABLE QUERIES
What you will learn about in this section

1. Foreign key constraints

2. Joins: basics

3. Joins: SQL semantics

4. ACTIVITY: Multi-table queries
Foreign Key constraints

• Suppose we have the following schema:

  Students(sid: string, name: string, gpa: float)
  Enrolled(student_id: string, cid: string, grade: string)

• And we want to impose the following constraint:

  ‘Only bona fide students may enroll in courses’ i.e. a student must appear in the Students table to enroll in a class

<table>
<thead>
<tr>
<th>Students</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
<td>gpa</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Bob</td>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
<td>3.8</td>
<td></td>
</tr>
</tbody>
</table>

| Enrolled | | |
|----------|---|---|---|
| student_id | cid | grade |
| 123 | 564 | A |
| 123 | 537 | A+ |

We say that student_id is a **foreign key** that refers to Students
Declaring Foreign Keys

Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)

CREATE TABLE Enrolled(
    student_id CHAR(20),
    cid CHAR(20),
    grade CHAR(10),
    PRIMARY KEY (student_id, cid),
    FOREIGN KEY (student_id) REFERENCES Students(sid)
)
Foreign Keys and update operations

Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)

• What if we insert a tuple into Enrolled, but no corresponding student?
  – INSERT is rejected (foreign keys are constraints)!
What is a foreign key vs. a key here?

<table>
<thead>
<tr>
<th>Company</th>
<th>CName</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
<td></td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
<td></td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<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>
<td></td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
<td></td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
<td></td>
</tr>
</tbody>
</table>
Ex: Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer = CName
    AND Country='Japan'
    AND Price <= 200
```
Joins

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer = CName
AND Country='Japan'
AND Price <= 200
```

A join between tables returns all unique combinations of their tuples which meet some specified join condition.
Joins

Several equivalent ways to write a basic join in SQL:

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer = CName
    AND Country='Japan'
    AND Price <= 200
```

```
SELECT PName, Price
FROM Product
JOIN Company ON Manufacturer = Cname
    AND Country='Japan'
WHERE Price <= 200
```

A few more later on...
### Joins

**Product**

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19</td>
<td>Gadgets</td>
<td>GWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29</td>
<td>Gadgets</td>
<td>GWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**Company**

<table>
<thead>
<tr>
<th>Cname</th>
<th>Stock</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWorks</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>

**Query**

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer = CName
    AND Country='Japan'
    AND Price <= 200
```
Tuple Variable Ambiguity in Multi-Table

```
Person(name, address, worksfor)
Company(name, address)
```

```
SELECT DISTINCT name, address
FROM       Person, Company
WHERE      worksfor = name
```

Which “address” does this refer to?

Which “name”s??
Tuple Variable Ambiguity in Multi-Table

Person(name, address, worksfor)
Company(name, address)

SELECT DISTINCT Person.name, Person.address
FROM Person, Company
WHERE Person.worksfor = Company.name

SELECT DISTINCT p.name, p.address
FROM Person p, Company c
WHERE p.worksfor = c.name

Both equivalent ways to resolve variable ambiguity
Acknowledgement

• Some of the slides in this presentation are taken from the slides provided by the authors.
• Many of these slides are taken from cs145 course offered by Stanford University.
• Thanks to YouTube, especially to Dr. Daniel Soper for his useful videos.