

# CSC 261/461 – Database Systems

## Lecture 3

Spring 2018

# Announcements

1. Start forming project teams.
  - Maximum Size of a team: 4
  - It's a team project so you are not allowed to work alone!
2. Project 1 (milestone 1) will be released soon (this weekend).
3. We will post a problem set this week.
  - Useful for Quiz 2
4. (Optional) workshops starting from Tomorrow

# 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>
```

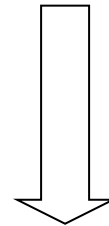
Call this a **SFW** query.

# Simple SQL Query: Selection

**Selection** is the operation of filtering a relation's tuples on some condition

PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

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



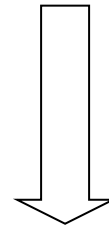
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks

# Simple SQL Query: Projection

**Projection** is the operation of producing an output table with tuples that have a subset of their prior attributes

PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

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



PName	Price	Manufacturer
Gizmo	\$19.99	GizmoWorks
Powergizmo	\$29.99	GizmoWorks

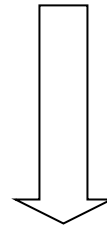


# Notation

Input schema

Product(PName, Price, Category, Manufacturer)

```
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
- **Table or relation names** may be case insensitive:
  - Depends on OS. For Unix, it's case sensitive.
  - Different: Product, product
- **Values** are not:
  - Different: 'Seattle', 'seattle'
- Use single quotes for constants:
  - 'abc' - yes
  - "abc" - no

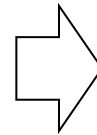
# LIKE: Simple String Pattern Matching

```
SELECT *  
FROM Products  
WHERE PName LIKE '%gizmo%'
```

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

# DISTINCT: Eliminating Duplicates

```
SELECT DISTINCT Category  
FROM Product
```



Category
Gadgets
Photography
Household

Versus

```
SELECT Category  
FROM Product
```



Category
Gadgets
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

Students			Enrolled		
sid	name	gpa	student_id	cid	grade
101	Bob	3.2	123	564	A
123	Mary	3.8	123	537	A+

Two red arrows originate from the 'student\_id' column of the 'Enrolled' table. One arrow points to the row where student\_id is 123 and cid is 564. The other arrow points to the row where student\_id is 123 and cid is 537. Both of these rows have a corresponding entry in the 'Students' table with sid 123 (Mary).

student\_id alone is not  
a key- what is?

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)!

# Keys and Foreign Keys

## Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

## Company:

Primary Key: CName

Foreign Key: None

## Product:

Primary Key: PName

Foreign Key: Manufacturer references CName

## Product

<u>PName</u>	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

What is a  
foreign key  
vs. a primary  
key here?

# Joins

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

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
```

*Note: we will often omit attribute types in schema definitions for brevity, but assume attributes are always atomic types*

# Joins

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

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

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

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

PName	Price	Category	Manuf
Gizmo	\$19	Gadgets	GWorks
Powergizmo	\$29	Gadgets	GWorks
SingleTouch	\$149	Photography	Canon
MultiTouch	\$203	Household	Hitachi

Company

Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan



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

PName	Price
SingleTouch	\$149.99

# 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)
```

Both equivalent  
ways to resolve  
variable  
ambiguity

```
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
```

# 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.