

# Lectures 2, 3 & 4: Introduction to SQL

Copyright: These slides are the modified version of the slides used in CS145 Introduction to Databases course at Stanford by Dr. Peter Bailis

# Announcements!

1. If you still have Jupyter trouble, let us know!

# Lecture 2: SQL Part I

# Today's Lecture

1. SQL introduction & schema definitions
  - ACTIVITY: Table creation
2. Basic single-table queries
  - ACTIVITY: Single-table queries!
3. Multi-table queries
  - ACTIVITY: Multi-table queries!

# 1. SQL Introduction & Definitions

# What you will learn about in this section

1. What is SQL?
2. Basic schema definitions
3. Keys & constraints intro
4. ACTIVITY: CREATE TABLE statements

# SQL Motivation

- Dark times 5 years ago.
  - Are databases dead?
- Now, as before: everyone sells SQL
  - Pig, Hive, Impala
- “Not-Yet-SQL?”



# Basic SQL



# SQL Introduction

- SQL is a standard language for querying and manipulating data
- SQL is a **very high-level** programming language
  - This works because it is optimized well!
- Many standards out there:
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ....
  - Vendors support various subsets

SQL stands for  
Structured Query Language

*NB:* Probably the world's most successful **parallel**  
programming language (multicore?)

# SQL is a...

- Data Definition Language (DDL)
  - Define relational *schemata*
  - Create/alter/delete tables and their attributes
- Data Manipulation Language (DML)
  - Insert/delete/modify tuples in tables
  - Query one or more tables – discussed next!

# Tables in SQL

## Product

| PName       | Price    | Manufacturer |
|-------------|----------|--------------|
| Gizmo       | \$19.99  | GizmoWorks   |
| Powergizmo  | \$29.99  | GizmoWorks   |
| SingleTouch | \$149.99 | Canon        |
| MultiTouch  | \$203.99 | Hitachi      |

A relation or table is a *multiset* of tuples having the attributes specified by the schema

Let's break this definition down

# Tables in SQL

## Product

| PName       | Price    | Manufacturer |
|-------------|----------|--------------|
| Gizmo       | \$19.99  | GizmoWorks   |
| Powergizmo  | \$29.99  | GizmoWorks   |
| SingleTouch | \$149.99 | Canon        |
| MultiTouch  | \$203.99 | Hitachi      |

A multiset is an unordered list (or: a set with multiple duplicate instances allowed)

List: [1, 1, 2, 3]

Set: {1, 2, 3}

Multiset: {1, 1, 2, 3}

i.e. no *next()*, etc. methods!

# Tables in SQL

## Product

| PName       | Price    | Manufacturer |
|-------------|----------|--------------|
| Gizmo       | \$19.99  | GizmoWorks   |
| Powergizmo  | \$29.99  | GizmoWorks   |
| SingleTouch | \$149.99 | Canon        |
| MultiTouch  | \$203.99 | Hitachi      |

An attribute (or column) is a typed data entry present in each tuple in the relation

*NB: Attributes must have an **atomic** type in standard SQL, i.e. not a list, set, etc.*

# Tables in SQL

## Product

| PName       | Price    | Manufacturer |
|-------------|----------|--------------|
| Gizmo       | \$19.99  | GizmoWorks   |
| Powergizmo  | \$29.99  | GizmoWorks   |
| SingleTouch | \$149.99 | Canon        |
| MultiTouch  | \$203.99 | Hitachi      |

Also referred to sometimes as a record

A tuple or row is a single entry in the table having the attributes specified by the schema

# Tables in SQL

## Product

| PName       | Price    | Manufacturer |
|-------------|----------|--------------|
| Gizmo       | \$19.99  | GizmoWorks   |
| Powergizmo  | \$29.99  | GizmoWorks   |
| SingleTouch | \$149.99 | Canon        |
| MultiTouch  | \$203.99 | Hitachi      |

The number of tuples is the cardinality of the relation

The number of attributes is the arity of the relation

# Data Types in SQL

- Atomic types:
  - Characters: CHAR(20), VARCHAR(50)
  - Numbers: INT, BIGINT, SMALLINT, FLOAT
  - Others: MONEY, DATETIME, ...
- Every attribute must have an atomic type
  - Hence tables are flat



# Table Schemas

- The **schema** of a table is the table name, its attributes, and their types:

```
Product(Pname: string, Price: float, Category:  
string, Manufacturer: string)
```

- A **key** is an attribute whose values are unique; we underline a key

```
Product(Pname: string, Price: float, Category:  
string, Manufacturer: string)
```

# Key constraints

A key is a minimal subset of attributes that acts as a unique identifier for tuples in a relation

- A key is an implicit constraint on which tuples can be in the relation
  - i.e. if two tuples agree on the values of the key, then they must be the same tuple!

```
Students(sid:string, name:string, gpa: float)
```

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?

## NULL and NOT NULL

- To say “don’t know the value” we use **NULL**
  - NULL has (sometimes painful) semantics, more detail later

```
Students(sid:string, name:string, gpa: float)
```

| sid | name | gpa  |
|-----|------|------|
| 123 | Bob  | 3.9  |
| 143 | Jim  | NULL |

*Say, Jim just enrolled in his first class.*

In SQL, we may constrain a column to be NOT NULL, e.g., “name” in this table

# General Constraints

- We can actually specify arbitrary assertions
  - E.g. *“There cannot be 25 people in the DB class”*
- In practice, we don’t specify many such constraints. Why?
  - Performance!

Whenever we do something ugly (or avoid doing something convenient) it’s for the sake of performance

# Summary of Schema Information

- Schema and Constraints are how databases understand the semantics (meaning) of data
- They are also useful for optimization
- SQL supports general constraints:
  - Keys and foreign keys are most important
  - We'll give you a chance to write the others

ACTIVITY: [Activity-2-1.ipynb](#)

## 2. 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'
```



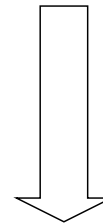
| PName      | Price   | Category | Manufacturer |
|------------|---------|----------|--------------|
| Gizmo      | \$19.99 | Gadgets  | GizmoWorks   |
| Powergizmo | \$29.99 | Gadgets  | GizmoWorks   |

# 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'
```



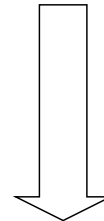
| PName      | Price   | Manufacturer |
|------------|---------|--------------|
| Gizmo      | \$19.99 | GizmoWorks   |
| Powergizmo | \$29.99 | GizmoWorks   |

# 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
  - Same: 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.



ACTIVITY: [Activity-2-2.ipynb](#)