DS 1300 - Introduction to SQL
Part 1 - Single-Table Queries
Overview

1. SQL introduction & schema definitions

2. Basic single-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. Activities: CREATE TABLE statements
Basic SQL
SQL Introduction

• SQL is a standard language for querying and manipulating data.

• SQL is a **high-level, declarative** programming language.

• SQL execution is highly optimized and parallelized.

• Many standards out there:
  – Standardized in 1986/87
  – ANSI SQL/ SQL-86, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), SQL:2011
  – Vendors support various subsets (e.g., SQLite implements most of the SQL-92 standard)
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
Tables in SQL

<table>
<thead>
<tr>
<th>Product</th>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td></td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td></td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td></td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td></td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

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

This is where the name "relational" databases comes from.
**Tables in SQL**

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

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

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

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

Sometimes also referred to as a **record**

<table>
<thead>
<tr>
<th>Product</th>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td></td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td></td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td></td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td></td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Data Types in SQL

• Atomic types:
  – Text: CHAR(20), VARCHAR(50)
  – Numbers: INT, BIGINT, SMALLINT, FLOAT
  – Others: MONEY, DATETIME, ...

• Every attribute must have an atomic type
Table Schemas

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

  Product(Pname: *text*, Price: *real*, Category: *text*, Manufacturer: *text*)

• A **key** is an attribute (combination) that identifies a tuple uniquely.

  Product(Pname: *text*, Price: *real*, Category: *text*, Manufacturer: *text*)
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:  text,  
             name:  text,  
             gpa:   real) |

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?
   (key candidates and primary key)
NULL and NOT NULL

• To say “don’t know the value” we use **NULL**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bob</td>
<td>3.9</td>
</tr>
<tr>
<td>143</td>
<td>Jim</td>
<td>NULL</td>
</tr>
</tbody>
</table>

*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
Activities

• SQLite data types:
  http://www.tutorialspoint.com/sqlite

• DB Browser
  – Create a database
  – Create a “Product” table
  – Add the shown data

<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>
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. Activities: 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.

```
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>

```
SELECT *
FROM   Product
WHERE  Category = 'Gadgets'
```
Projection is the operation of producing an output table with tuples that have a subset of their prior attributes.

```
SELECT Pname, Price, Manufacturer
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>
### Notation

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

Output schema:
```
Answer(PName, Price, Manufacturer)
```

Product(PName, Price, Category, Manufacturer)

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 text constants:
  – ‘abc’ - yes
  – “abc” - no
DISTINCT: Eliminating Duplicates

```
SELECT DISTINCT Category
FROM Product
```

Versus

```
SELECT Category
FROM Product
```

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Photography</td>
</tr>
<tr>
<td>Household</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Photography</td>
</tr>
<tr>
<td>Household</td>
</tr>
</tbody>
</table>
COUNT

COUNT is an aggregation function that returns the number of elements.

Example: Find the number of products with a price of $20 or more.

```
SELECT COUNT(*) FROM product WHERE price >= 20
```

Syntax: COUNT([ALL | DISTINCT] expression)
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.

Text is ordered alphabetically.
LIMIT Clause

Used to limit the data amount returned by the SELECT statement.

**Example:** Find the 5 most expensive products

```
SELECT * FROM product
ORDER BY price DESC
LIMIT 5
```

**Syntax:** LIMIT [no of rows] OFFSET [row num]

**Note:** LIMIT is not standard SQL (e.g., MS SQL Server uses SELECT TOP)
Operators

Some of the operators supported by SQL are:

- `=`, `==` equal
- `!=`, `<>` not equal
- `<`, `<=` less than (or equal)
- `>`, `>=` greater than (or equal)
- `+`, `-`, `/`, `*` arithmetic operators
- `AND`, `OR`, `NOT` logic operators
- `IS NULL`, `IS NOT NULL` checks for NULL values

Example: Find products and their price + 8% sales tax for gadgets that cost at least $100

```sql
SELECT pname, price * 1.08 AS Price_with_tax
FROM product,
WHERE category = 'Gadgets' AND price >= 100
```
IN and BETWEEN

The IN operator allows you to specify multiple values in a WHERE clause.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2
```
LIKE: Simple String Pattern Matching

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

```sql
SELECT * FROM Products WHERE PName LIKE '%gizmo%'
```
CASE Statement


Example:

SELECT name, CASE WHEN price > 200 THEN 'Yes' ELSE 'No' END AS expensive FROM Product

Product

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>50</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>299</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>89</td>
</tr>
</tbody>
</table>
Activities

• SQLite Operators
• Expressions
• Where clauses
• And & Or clauses

(http://www.tutorialspoint.com/sqlite/)

1. Find all the gadgets and sort them by price.
2. What is the most expensive gadget?
3. How many gadgets are in the database?
4. How many gadgets are less than $20?
5. How much does it cost to buy all gadgets?
6. What happens if the manufacturer GizmoWorks changes its name? This is why we need multiple tables!