Introduction to SQL

Zhifeng Wang

Department of Statistics
Florida State University
Data

Bank of America
How do they store data?
Database

- **Database**: collection of *organized* data.
- **Database Management System (DBMS)**: a software that is designed to allow the *definition, creation, querying, update* of database.
Relational Database

Relational Database: A digital database whose organization is based on the relational model of data.

- **Table**: a *structured list* of data of a specific type.

<table>
<thead>
<tr>
<th>prod_id</th>
<th>vend_id</th>
<th>prod_name</th>
<th>prod_price</th>
<th>prod_desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNBG01</td>
<td>DLL01</td>
<td>Fish bean bag toy</td>
<td>3.49</td>
<td>Fish bean bag toy. complete...</td>
</tr>
<tr>
<td>BNBG02</td>
<td>DLL01</td>
<td>Bird bean bag toy</td>
<td>3.49</td>
<td>Bird bean bag toy. eggs are...</td>
</tr>
<tr>
<td>BNBG03</td>
<td>DLL01</td>
<td>Rabbit bean bag toy</td>
<td>3.49</td>
<td>Rabbit bean bag toy. comes...</td>
</tr>
<tr>
<td>BR01</td>
<td>BRS01</td>
<td>8 inch teddy bear</td>
<td>5.99</td>
<td>8 inch teddy bear. comes with...</td>
</tr>
<tr>
<td>BR02</td>
<td>BRS01</td>
<td>12 inch teddy bear</td>
<td>8.99</td>
<td>12 inch teddy bear. comes ...</td>
</tr>
<tr>
<td>BR03</td>
<td>BRS01</td>
<td>18 inch teddy bear</td>
<td>11.99</td>
<td>18 inch teddy bear. comes ...</td>
</tr>
<tr>
<td>RGAN01</td>
<td>DLL01</td>
<td>Raccoon Ann</td>
<td>4.99</td>
<td>18 inch Raccoon Ann doll</td>
</tr>
<tr>
<td>RYL01</td>
<td>FNG01</td>
<td>King doll</td>
<td>9.49</td>
<td>12 inch king doll with roval ...</td>
</tr>
<tr>
<td>RYL02</td>
<td>FNG01</td>
<td>Queen doll</td>
<td>9.49</td>
<td>12 inch queen doll with roval ...</td>
</tr>
</tbody>
</table>
Basic SQL in Action

**Relational Database**

- **Row**: a *record* in a table.
- **Column**: a single field in a table.
- **Datatype**: a type of allowed data (for *every column*).
- **Primary Key**: a column (or set of columns) whose values uniquely identify every row in a table.
- **Relationship Diagram**: shows the relationships of tables stored in a database.
*All the examples will be based on the following database.

Figure: Sample tables relationship diagram: part of an order system used by a distributor of toys
Understanding the Sample Tables

- **Vendors**: stores the vendors whose products are sold.
- **Products**: contains the product catalog, one product per row.
- **Customers**: stores all customer (shop) information.
- **Orders**: stores customer orders (but not order details).
- **OrderItems**: stores the actual items in each order, one row per item per order.
Structured Query Language (SQL): programming language designed for managing data in DBMS.

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce.

- Almost every major DBMS supports SQL.
- SQL is easy to learn. The statements are all made up of descriptive English words.
- Despite its apparent simplicity, SQL is actually a very powerful language.
MySQL is an open-source relational database management system (RDBMS).

MySQL was owned by the Swedish company MySQL AB, now owned by Oracle Corporation.

MySQL is used in many high-profile, large-scale websites, including Google (though not for searches), Facebook, Amazon and YouTube.
They are **free**!

- **MySQL Community Edition**: the **basic** DBMS.
  https://dev.mysql.com/downloads/

- **MySQL Workbench**: a visual development environment for MySQL.
  https://dev.mysql.com/downloads/workbench/

- **MySQL Installer (for Windows)**: **all-in-one package**.
  https://dev.mysql.com/downloads/windows/installer/
Outline

SQL in Action

- Retrieving Data
- Sorting Retrieved Data
- Filtering Data
- Summarizing Data
- Grouping Data
- Working with Subqueries
- Joining Tables

*All the examples are from the book “SQL in 10 Minutes, 4th”.*
Retrieving Data – The SELECT Statement

- SQL statements are made up of plain English terms. These terms are called **keywords** (reserved words).
- We use the **SELECT** statement to retrieve one or more columns of data from a table.

```
SELECT column_name
FROM table_name;
```

1. The displayed data was **unsorted**.
2. Multiple SQL statements must be separated by **semicolons**.
3. SQL statements are **case-insensitive**.
4. All **extra white space** within a SQL statement will be ignored.
To explicitly sort data retrieved using a `SELECT` statement, the `ORDER BY` clause is used.

The `ORDER BY` Clause

```sql
SELECT column_name
FROM table_name
ORDER BY column_name;
```

1. The `ORDER BY` clause should be the last clause.
2. It is perfectly legal to sort data by a column that is not retrieved.
3. Ascending order is the default. Sort descending? Use `DESC` keyword.
Within a `SELECT` statement, data is filtered by specifying `search criteria` in the `WHERE` clause.

**The WHERE Clause**

```
SELECT column_name
FROM table_name
WHERE search_criteria;
```

- Make sure that `ORDER BY` comes after the `WHERE` clause when using both.
Filtering Data – WHERE Clause Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equality</td>
</tr>
<tr>
<td>! =</td>
<td>Non-equality</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>Between two specified values</td>
</tr>
<tr>
<td>IS NULL</td>
<td>Is a NULL value</td>
</tr>
</tbody>
</table>

**Figure:** WHERE Clause Operators

- Not all of these operators are supported by all DBMSs.
- For example, Microsoft Access supports `<>` instead of `!=`. 
Filtering Data – More Advanced Operators

For a greater degree of filter control, SQL lets you specify multiple conditions.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>matches all the specified conditions</td>
</tr>
<tr>
<td>OR</td>
<td>matches either of the specified conditions</td>
</tr>
<tr>
<td>IN</td>
<td>matches either of a list of values</td>
</tr>
<tr>
<td>NOT</td>
<td>negates a condition</td>
</tr>
</tbody>
</table>

Table: Advanced Operators

1. **WHERE** clauses can contain any number of **AND** and **OR** operators.
2. **AND** operator has precedence over **OR**.
Filtering Data – Wildcard Filtering Using LIKE

- All the previous operators we studied filter against known values.
- How could we search for all products that contained “bean bag” within the product name?
- The answer is “LIKE + wildcard”.

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>matches any number of occurrences of any character</td>
</tr>
<tr>
<td>_</td>
<td>matches a single character</td>
</tr>
</tbody>
</table>

Table: Wildcards

1. % wildcard does NOT match NULL.
2. More advanced pattern match (e.g., REGEXP):
Summarizing Data

- It is often necessary to summarize data without actually retrieving it all.
- For example,
  - Determining the number of rows in a table.
  - Obtaining the sum of a set of rows in a table.
  - Finding the highest, lowest, and average values in a column.
- **AVG**, **MAX**, **MIN**, **SUM** generally apply to numeric columns.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG()</td>
<td>Returns a column’s average value</td>
</tr>
<tr>
<td>COUNT()</td>
<td>Returns the number of rows in a column</td>
</tr>
<tr>
<td>MAX()</td>
<td>Returns a column’s highest value</td>
</tr>
<tr>
<td>MIN()</td>
<td>Returns a column’s lowest value</td>
</tr>
<tr>
<td>SUM()</td>
<td>Returns the sum of a column’s values</td>
</tr>
</tbody>
</table>

**Figure:** SQL Aggregate Functions
Grouping Data – The GROUP BY and HAVING Clauses

Grouping Data

- We already know how to get the number of products offered by a specific vendor.
- But what if we want to return the number of products offered by each vendor? 
- Answer: GROUP BY.

Filtering Groups

- In addition grouping data, SQL also allows you to filter which groups to include and which to exclude.
- SQL provides another clause for this purpose: HAVING.
- HAVING v.s. WHERE:
  - WHERE filters before data is grouped, and HAVING filters after data is grouped.
  - HAVING supports all WHEREs operators.
# Recap

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Columns or expressions to be returned</td>
<td>Yes</td>
</tr>
<tr>
<td>FROM</td>
<td>Table to retrieve data from</td>
<td>Only if selecting data from a table</td>
</tr>
<tr>
<td>WHERE</td>
<td>Row-level filtering</td>
<td>No</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>Group specification</td>
<td>Only if calculating aggregates by group</td>
</tr>
<tr>
<td>HAVING</td>
<td>Group-level filtering</td>
<td>No</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Output sort order</td>
<td>No</td>
</tr>
</tbody>
</table>

**Figure:** SELECT Clauses and Their Sequence
Multiple Tables

- All the `SELECT` statements we have seen so far are simple queries: single statements retrieving data from individual tables.

- Next, we will work on multiple tables, and discuss
  1. Working with Subqueries
  2. Joining Tables
SQL also enables you to create subqueries: queries that are embedded into other queries.

**Question**: List all the customers who ordered item 'RGAN01'.

Let's go back to the *relationship diagram*.

**Possible steps:**

1. Retrieve the order numbers of all orders containing item 'RGAN01'.
2. Retrieve the customer ID of all the customers who have orders listed in the order numbers returned in the previous step.
3. Retrieve the customer information for all the customer IDs returned in the previous step.
Why Use Joins?

- Breaking data into multiple tables enables more efficient storage, easier manipulation, and greater scalability.
- But if data is stored in **multiple tables**, how can you retrieve that data with a **single** SELECT statement?
- Answer: **JOIN**.
- Using a special syntax, multiple tables can be joined so that a **single set of output** is returned.
Different Join Types

- Equi-join (Inner Join)
- Cross Join
- Self-join
- (Left) Outer Join
Want More?

More **advanced** applications (e.g., creating table, updating/deleting data, combining queries, using views and cursors) are waiting for you!

Thank you!