Statistical Inference

Week 5 – Part 1 –Relational Database

CS 457 - L1 Data Science

Zeehasham Rasheed

Relational Database Management Systems

Commercial & **Open Source** RDBMS/SQL Systems

Oracle Database & SQL:

https://www.oracle.com/database/index.html

https://docs.oracle.com/en/database/

IBM DB2:

https://www.ibm.com/analytics/us/en/db2/

Microsoft SQL Server:

https://www.microsoft.com/en-us/sql-server/

MySQL:

https://www.mysql.com/

PostgreSQL:

https://www.postgresql.org/

MariaDB, SQLite, Informix, Apache Derby etc.

SQL-like System for Big Data

https://hive.apache.org/

Tutorials

https://www.tutorialspoint.com/sql/sql-overview.htm

Relational Database (Structured)

Review of Attributes of the Relational Data Model.

- The **relational model** is credited to E. F. Codd (1970!) at IBM.
  - https://researcher.watson.ibm.com/researcher/files/us-dchamber/SQL-encyclopedia-entry.pdf

- **Basic Definitions**:
  - **Data Structure**: Data are organized into **tables** (often called **relations**, not to be confused with **relationships**) with **rows** and **columns**. (like dataframes)
  - **Data Manipulation**: The structured query language (**SQL**) is used to manipulate data in the tables; informed by **set operations** (subset, intersect, union, …)
  - **Data Integrity**: The **DBMS** enforces business and technical rules to maintain **data integrity**
    - **Example**: Do not delete a **CUSTOMER** table row when there is a corresponding outstanding row in the **CUSTOMER_ORDERS** table.

Physical Layer Abstraction

- **SQL Applications specify what, not how**
  - leaves HW/SW *implementation* to system engineers

- **Query *optimization* engine**
  - Specific to & optimized for HW/OS infrastructure…
    - especially Oracle, for example:
    - https://www.oracle.com/database/technologies/exadata.html

- **Physical layer can change without modifying SQL applications**
- **Create indexes to support queries**
- **In-Memory databases increasingly common**
    - **Massive memory systems (20+ TB!), helpful for Big Data analytics**

Data Definition

Relational DBMS requires a **schema, predefined** at the start

- **Metadata that describes data types and structures**
  - CREATE TABLE(Column1 Datatype1, Column2 Datatype 2, …)

Defines Item Names and *computational* types (map them to NOIR)

- **Primary Key: used for basic indexing and fast lookup**
- **Foreign Key: for indexing links to other tables**

Triggers to respond to Insert, Update , & Delete

- **Transactions require atomic operations**
  - **Atomic means "Must fully complete or fully fail"**
    - **No intermediate partial states!**

Available Features

- **SQL + RDBMS implementations support:**
  - **Inserting/Storing transactions**
  - **Alter tables…**
  - **Drop tables…**
  - **Security and Access Controls**
    - **User access**
    - **Data redaction**
      - **Also called data making**
      - **For privacy, test, development**

Data Types in SQL

The following table lists some of the typical general data types in SQL.

Think about how these map to the NOIR analytical data types we discussed earlier.

Data Type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER or INT	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 10
DECIMAL(p,s), or NUMBER or NUMERIC or **FLOAT**	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values

Data Structure and Format

- **Relational Data Structure**:
  - A **table** is a **two-dimensional** representation of data (**like a .csv file**)
  - Each **column** is named and represents a "**field"**(of some **data type**)
  - Each **row** is a "**record"**.
  - A **table** can have any number of rows
    - (within physical storage limitations).

- **Shorthand Notation** – to express the structure of a table:
  - TABLE_NAME(PrimaryKeyField1, Field2, Field3, ...)
  - **example:**
    EMPLOYEE(EmpID, EmpName, Department, DateHired)

Keys and Relations

Relational Keys:

- **Primary Key**: column (or combination of columns) that **uniquely** identifies a row.
- **Composite Key**: a primary key that includes more than one column.
- **Foreign Key**: a field that **links** one table to another table. A table can have an unlimited number of foreign keys linking to other tables.

Properties of Tables (Relations)

- Each table in a database has a **unique name**.
- Each entry in a table at the intersection of a row/column can only store a **single value** (principle of **atomicity**). No multivalued fields are allowed in a table.
- Each **row** is **unique**. (no duplicates)
- Each column has a **unique field name**.
- **Sequence** of rows as well as columns is insignificant and can be interchanged without changing the data.

ACID Property

Transaction – a unit in a database that can be retrieved or updated.

In RDBs, all transactions must be ACID:

- **Atomic** – **All** of the work in a **transaction** completes (commits), or **none** of it completes (**why is this required for transactions**?)
- **Consistent** – A transaction transforms the database from one **consistent** state to another **consistent** state. Consistency is defined in terms of constraints.
- **Isolated** – The results of any changes made during a transaction are *not visible until the transaction has committed.*
- **Durable** – The results of a committed transaction *survive failures (commit or rollback)*

Think about how these requirements might affect performance and scalability

End of Part 1

Statistical Inference

Week 5 – Part 2 – SQL Syntax and Examples

CS 457 - L1 Data Science

Zeehasham Rasheed

SQL Syntax

SQL CREATE DATABASE Statement:

```
CREATE DATABASE database_name;
```

SQL CREATE TABLE Statement:

```
CREATE TABLE table_name( column1 datatype, column2
datatype, column3 datatype, ..... columnN datatype,
PRIMARY KEY( one or more columns ) );
```

SQL INSERT INTO Statement (enter a record):

```
INSERT INTO table_name( column1, column2....columnN)
VALUES ( value1, value2....valueN);
```

SQL COUNT() function:

The SQL COUNT() function returns the number of rows in a table satisfying the criteria specified in the WHERE clause. It sets the number of rows or non NULL column values.

SQL SUM() function:

The SQL AGGREGATE SUM() function returns the sum of all selected column.

SQL AVG() function:

The SQL AVG() function calculates arithmetic mean of a column of numeric type. It returns the mean of all non NULL values.

SQL MAX() function:

The aggregate function SQL MAX() is used to find the maximum value or highest value of a certain column or expression. This function is useful to determine the largest of all selected values of a column.

SQL MIN() function:

The aggregate function SQL MIN() is used to find the minimum value or lowest value of a column or expression. This function is useful to determine the smallest of all selected values of a column.

Resources

- **RDBMS/SQL Resources**
  - IDMA Chapter 6
    - https://info.jblearning.com/jams_data_links
  - Tutorial
    - https://www.tutorialspoint.com/sql/sql-overview.htm
  - PostgreSQL: https://www.postgresql.org/download/
  - MySQL: https://dev.mysql.com/downloads/mysql/
  - Oracle: https://livesql.oracle.com

Joins in SQL

- Single-table queries are straightforward.

- To find all 18 years old students, we can write:

```
SELECT *
  FROM Students S
 WHERE S.age=18
```

Joining Relations

- Chaining/Joining relations together is the basic **inference method** in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality
 FROM Students S, Mortality M
 WHERE S.Race = M.Race
```

S

Name	Race
Socrates	Man
Thor	God
Barney	Dinosaur
Blarney stone	Stone

M

Race	Mortality
Man	Mortal
God	Immortal
Dinosaur	Mortal
Stone	Non-living

Joining Relations Result

- Chaining/Joining relations together is the basic **inference method** in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality
 FROM Students S, Mortality M
 WHERE S.Race=M.Race
```

Name	Mortality
Socrates	Mortal
Thor	Immortal
Barney	Mortal
Blarney stone	Non-living

Join Types

- **JOIN or INNER JOIN (commonly used)**
- **LEFT JOIN**
- **RIGHT JOIN**
- **OUTER JOIN**

- Out of scope for this class
- Read Book Chapter 6 for details

Connecting to Python and R

- **Connecting your Python/R environment with PostgreSQL (or any Database of your company)**
  - Directly connect to your database table and perform Data Exploration, Machine Learning and Visualization in Python/R
  - Python library called **psycopg2**
  - R package called **postgres**

End of Part 2