Data Science 100
Databases Part 2
(The SQL)

Database Management Systems
A database management system (DBMS) is a software system that stores, manages, and facilitates access to one or more databases.

- **Relational** database management systems
  - SQL

Logically organize data in relations (tables)
Structured Query Language (SQL) to define, manipulate and compute on data.

Physical Data Independence

<table>
<thead>
<tr>
<th>Name</th>
<th>Prod</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>iPod</td>
<td>$200.00</td>
</tr>
<tr>
<td>Joey</td>
<td>Bike</td>
<td>$333.99</td>
</tr>
<tr>
<td>Alice</td>
<td>Car</td>
<td>$999.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
</tbody>
</table>

Optimized Data Structures

<table>
<thead>
<tr>
<th>Optimized Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>B+Trees</td>
</tr>
</tbody>
</table>

Conceptual SQL Evaluation

Try Queries Here
https://sqlfiddle.com/#!17/67109/12

Try Queries Here

How do you interact with a database?

- What is the DBMS?
  - Server
  - Software
  - A library

Answer: It can be all of these.
Interacting with a DBMS

**Query**

```sql
SELECT * FROM sales WHERE price > 100.0
```

**Response**

```
<table>
<thead>
<tr>
<th>Customer</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>$200.00</td>
</tr>
<tr>
<td>Joe</td>
<td>$333.99</td>
</tr>
</tbody>
</table>
```

Many systems will connect to a DBMS concurrently.

**Python Analysis**

```
# This looks like a star... How do we do analysis? Join!!?
```

**Connections between table**

**Data in the Organization**

* A little bit of buzzword bingo!

**Multidimensional Data Model**

<table>
<thead>
<tr>
<th>Sales Fact Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

**Time**

<table>
<thead>
<tr>
<th>TimeID</th>
<th>Date</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/30/16</td>
<td>Wed.</td>
</tr>
<tr>
<td>2</td>
<td>3/31/16</td>
<td>Thu.</td>
</tr>
<tr>
<td>3</td>
<td>4/1/16</td>
<td>Fri.</td>
</tr>
</tbody>
</table>

**Dimensions**

- Normalized Representation
- Fact Table
- Minimizes redundant info.
- Reduces data errors
- Dimensions
- Easy to manage and summarize
- Rename: Galaxy 1 → Phablet

**Joins!**

Bringing tables together for decades.
Join Queries

\[
\text{SELECT [DISTINCT] column expression list; FROM table_1, ..., table_n AS alias; [WHERE predicate];} \\
[GROUP BY column list]; [HAVING predicate]; [ORDER BY column list];
\]

1. FROM: compute outer product of tables.
2. WHERE: Check conditions, discard tuples that fail.
3. SELECT: Specify desired fields in output.

➢ Note: likely a terribly inefficient strategy!
➢ Query optimizer will find more efficient plans.

Return Sailors (S) and the dates of their Reservations (R)

\[
\text{SELECT S.sname, R.day} \\
\text{FROM Reserves AS R, Sailors AS S} \\
\text{WHERE S.sid = R.sid}
\]

\[
R1 \bowtie S1
\]

The Outer-Product (\(\times\))

\[
R1 \times S1: \text{Each row of } R1 \text{ paired with each row of } S1
\]

About Range Variables

➢ Needed when ambiguity could arise.
➢ e.g., same table used multiple times in FROM (“self-join”)

\[
\text{SELECT * FROM Sailors AS S1, Sailors AS S2} \\
\text{WHERE S1.age > S2.age}
\]

Inner/Natural Joins

\[
\text{SELECT s.sid, s.sname, r.bid} \\
\text{FROM Sailors s, Reserves r} \\
\text{WHERE s.sid = r.sid AND s.age > 20;}
\]

\[
\text{SELECT s.sid, s.sname, r.bid} \\
\text{FROM Sailors s, INNER JOIN Reserves r} \\
\text{ON s.sid = r.sid AND s.age > 20;}
\]

\[
\text{SELECT s.sid, s.sname, r.bid} \\
\text{FROM Sailors s, NATURAL JOIN Reserves r} \\
\text{WHERE s.age > 20;}
\]

➢ “NATURAL” means equi-join for each pair of attributes with the same name.
Left Join

Returns all matched rows, and preserves all unmatched rows from the table on the left of the join clause.
(Use nulls in fields of non-matching tuples)

```
SELECT s.sid, s.sname, r.bid
FROM Sailors2 s LEFT JOIN Reserves2 r
ON s.sid = r.sid;
```

Returns all sailors & bid for boat in any of their reservations.
Note: If there is a sailor without a boat reservation then the sailor is matched with the NULL bid.

Right Join

- Right join returns all matched rows, and preserves all unmatched rows from the table on the right of the join clause.

```
SELECT r.sid, b.bid, b.bname
FROM Reserves2 r RIGHT JOIN Boats2 b
ON r.bid = b.bid;
```

- Returns all boats & information on which ones are reserved.
- No match for b.bid if r.sid is NULL!

Full Outer Join

- Full Outer Join returns all (matched or unmatched) rows from the tables on both sides of the join clause.

```
SELECT r.sid, b.bid, b.bname
FROM Reserves2 r FULL JOIN Boats2 b
ON r.bid = b.bid;
```

- If no boat for a sailor? → b.bid IS NULL AND b.bname IS NULL!
- If no sailor for a boat? → r.sid IS NULL!
**Brief Detour: Null Values**

- Field values are sometimes unknown.
- SQL provides a special value `NULL` for such situations.
- Every data type can be `NULL`.
- The presence of `NULL` complicates many issues. E.g.:
  - Selection predicates (WHERE)
  - Aggregation
- But `NULLs` are common after outer joins.

**NULL in the WHERE clause**

- Consider a tuple where `rating` is `NULL`.
- `INSERT INTO sailors VALUES (11, 'Jack Sparrow', NULL, 35);`
- If we run the following query
  ```sql
  SELECT * FROM sailors WHERE rating > 8;
  ```
  Jack Sparrow will not be included in the output.

**NULL in comparators**

What entries are in the output of all these queries?

- `SELECT rating = NULL FROM sailors;`
- `SELECT rating < NULL FROM sailors;`
- `SELECT rating >= NULL FROM sailors;`
- `SELECT * FROM sailors WHERE rating = NULL;`

Rule: `(x op NULL)` evaluates to `NULL`.

**Explicit NULL Checks**

- To check if a value is `NULL` you must use explicit `NULL` checks.
  ```sql
  SELECT * FROM sailors WHERE rating IS NULL;
  SELECT * FROM sailors WHERE rating IS NOT NULL;
  ```

**NULL in Boolean Logic**

Three-valued logic:

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>F</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not</td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>And</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Or</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

- `SELECT * FROM sailors WHERE rating > 8 AND TRUE;`
- `SELECT * FROM sailors WHERE rating > 8 OR TRUE;`
- `SELECT * FROM sailors WHERE NOT (rating > 8);`

**NULL and Aggregation**

- `SELECT count(rating) FROM sailors;`  
  4
- `SELECT sum(rating) FROM sailors;`  
  27
- `SELECT avg(rating) FROM sailors;`  
  ??
- `SELECT count(*) FROM sailors;`  
  ??

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Popeye</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>OliveOyl</td>
<td>11</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Garfield</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>Jack</td>
<td>55</td>
<td>38</td>
</tr>
</tbody>
</table>
NULL and Aggregation

```sql
SELECT count(rating) FROM sailors;
```

```
4
```

```sql
SELECT sum(rating) FROM sailors;
```

```
27
```

```sql
SELECT avg(rating) FROM sailors;
```

```
(10+11+1+5) / 4 = 6.75
```

```sql
SELECT count(*) FROM sailors;
```

```
5
```

NULLs: Summary

- NULL op NULL is NULL
- WHERE NULL: do not send to output
- Boolean connectives: 3-valued logic
- Aggregates ignore NULL-valued inputs

SQL JOINs

```sql
http://sqlfiddle.com/#!17/f35aa/7
```

HTML table:

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Popeye</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>OliveOyl</td>
<td>11</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Garfield</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Bobs</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>Jack Sparow</td>
<td>(null)</td>
<td>35</td>
</tr>
</tbody>
</table>

SQL Query Demo

Returning to Notebook