Structured Query Language -Salt in a Food

Ruturaj Nalawade
Data Science Engineer
@Infostretch

Agenda

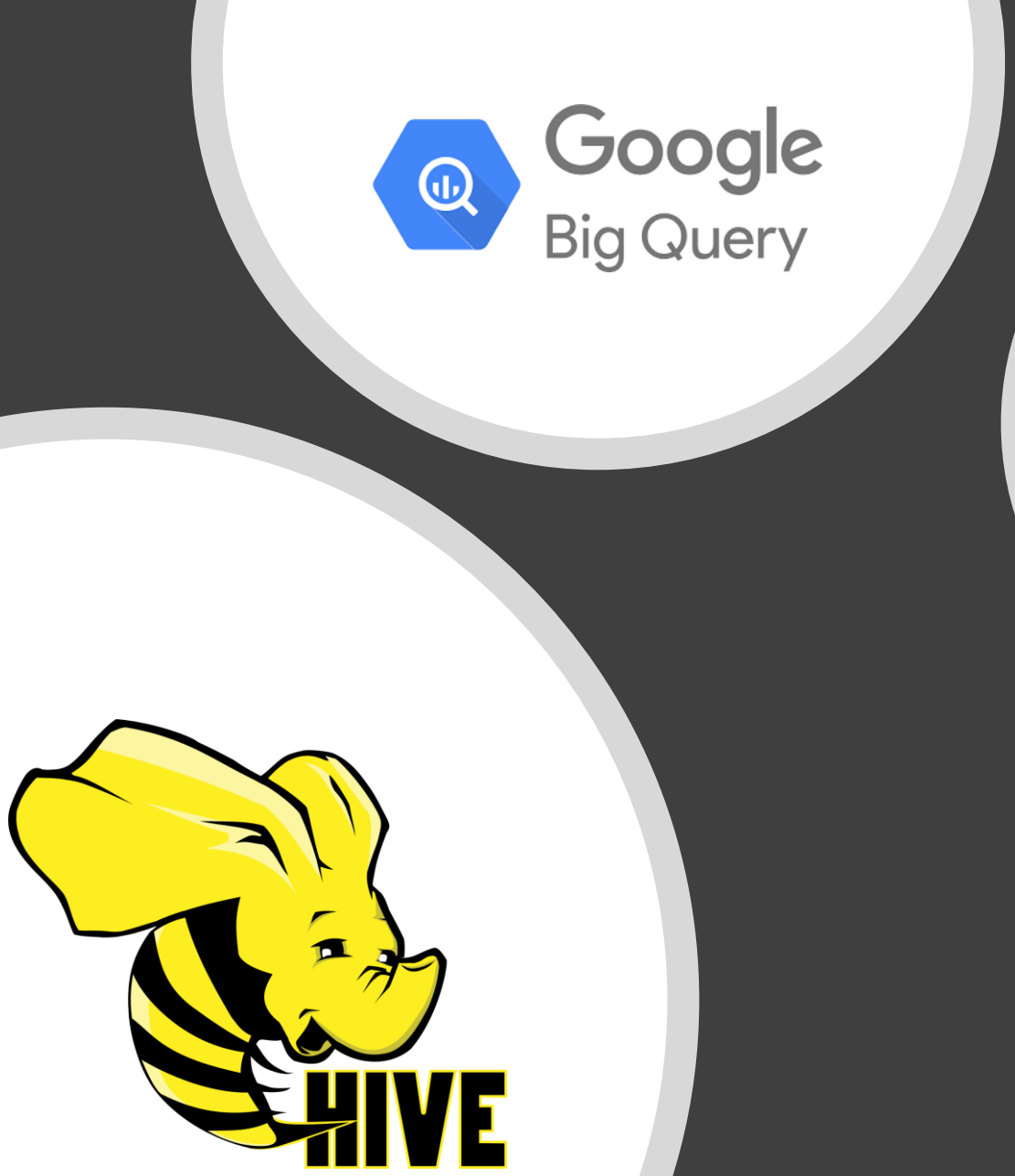
- Need of SQL
- Revisiting Basic SQL Concepts
  - Data Definition Commands
  - Data Manipulation Commands
  - Operators
  - Indexes
  - MySQL functions
- Basic SQL Queries & Analysis

Why you should know SQL?

- Data Mining
- Data Manipulation
- Combine Data from Multiple Sources
- Manage Large Pools of Data
- Servers and Databases

Why ML guys should know SQL?

- Data Exploration, Data Cleansing
  - Filling in missing data (imputing values)
  - Detecting and removing outliers
  - Smoothing
    - removing noise by averaging values together
  - Filtering, sampling
    - keeping only selected representative values
  - Feature extraction
    - e.g. in a photo database, which people are wearing glasses? which have more than one person? which are outdoors?


Google
Big Query


Spark SQL


Amazon Athena

HIVE

& many more


amazon
REDSHIFT


KSQL

Revisiting SQL

SQL Data Defination Commands	
Command OR Option	**Description**
CREATE SCHEMA | DATABASE	Creates a database schema
CREATE TABLE	Creates new table with given name
NOT NULL	Ensures that a column cannot have NULL value
UNIQUE	Ensures that all values in a column are different
PRIMERY KEY	Uniquely identifies each row/record in a database table
FOREIGN KEY	Uniquely identifies a row/record in any of the given database table
DEFAULT	Provide a default value for a column
CHECK	Ensures that all the values in a column satisfies certain conditions
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates dynamic subset of data from one or more tables
ALTER TABLE	Modifies table's definition
CREATE TABLE AS	Creates new table based on a query
DROP TABLE	Permanently deletes a table (thus the data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes the view

Revisiting SQL (continued)

SQL Data Manipulation Commands	
Command OR Option	**Description**
INSERT	Insert row(s) into a table
SELECT	Select attributes or rows in one or more tables or views
WHERE	Restricts the selection of rows based on conditional expression
GROUP BY	Groups selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on the attributes
UPDATE	Modifies attribute's values in one or more table's rows
DELETE	Deletes one or more rows from table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to original values

Revisiting SQL (continued)

SQL Data Manipulation Commands (Cont.)	
Comparison Operators	**Description**
= , < , > , <= , >= , <>	Used in conditional expression
Logical Operators	
AND / OR / NOT	Used in conditional expression
Special Operators	Used in conditional expression
BETWEEN	Checks whether the value is within range
IS NULL	Checks whether the value is null
LIKE	Checks whether the value mathces given string pattern
IN	Checks whether value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limit values to unique values
Aggregate Functions	Used with SELECT to return mathematical summary
COUNT	Returns no of rows with NON-NULL values for a given column
MIN	Returns minimium attribute value found in a given column
MAX	Returns maximum attribute value found in a given column
SUM	Returns sum of all values for a given column
AVG	Returns average of all values for a given column

Revisiting SQL (continued)

Common SQL Data Types		
DATA TYPE	**FORMAT**	**COMMENTS**
NUMERIC		
	INTEGER | INT	Stores 4 byte long integer value
	BIGINT	Stores 8 byte long integer value
	SMALLINT	Stores 2 byte long integer value
	BOOLEAN | TINYINT	Single byte integer value, Zero:False , NonZero:True
	DECIMAL | NUMERIC (M,D)	Stores exact numeric data values, where M is the length and D is the precision
	FLOAT | DOUBLE (M,D)	Stores approximate numeric data values, where M is the length and D is the precision
STRING		
	CHAR	Stores fixed length character data
	VARCHAR	Stores variable length character data
DATE		
	DATE	Stores date ('0000-00-00')
	TIME	Stores time ('00:00:00')
	DATETIME | TIMESTAMP	Stores date and time ('0000-00-00 00:00:00')

MySQL Functions :

- String:
  - CONCAT, LOWER, UPPER, LENGTH, CHAR_LENGTH, ASCII, FIELD, FORMAT, INSERT, REVERSE, STRCMP, SUBSTRING, TRIM….etc.
- Numeric Functions:
  - ABS, SIN, COS…, AVG, CEIL, DEGREES, DIV, FLOOR, LOG, MAX, MIN, MOD, POW, ROUND, SUM…etc.
- Date Functions:
  - DATE, NOW, DATEDIFF, DATE_ADD, DATE_SUB, DATE_FORMAT, HOUR, MINUTE_MONTH, WEEK, YEAR, TIMESTAMP, TO_DAYS, FROM_DAYS…etc.
- Advanced Functions:
  - CAST, COALESCE, CASE, IF, ISNULL, IFNULL, CONVERT …etc.

Indexing in MySQL:

- **REALLY** important to speed up query processing time.
- When primary key is declared, DBMS automatically creates unique index
- Multiple-Column index (col1, col2, col3):
  - (col1)
  - (col1, col2)
  - (col1, col2, col3)

DB Schema:

"student" data table

Column Name	Data Type
id*	Integer
name	Text

"term_gpa" data table

Column Name	Data Type
id*	Integer
term*	Integer
gpa	Float

"degrees" data table

Column Name	Data Type
id*	Integer
term	Integer
degree*	Char(5)

* Indicates primary key of the table

Data Snapshot:

"student"

id	name
1	Edith Warton
2	Dorian Gray
3	Ophelia
4	Henry James
5	Harper Lee

"term_gpa"

id	term	gpa
1	2011	3.32
1	2012	3.51
2	2011	2.22
2	2013	1.70
3	2011	3.70
4	2011	3.10
4	2012	3.21
4	2013	3.30
5	2013	2.99

"degrees"

id	term	degree
1	2012	EconS
3	2011	MathS
3	2011	CompS
4	2012	EngLT

Q: Get the students data(name, gpa, degree)

Query:

```
SELECT s.id AS id , s.name AS
        name , t.gpa AS gpa ,
        d.degree AS degree ,
        t.term as term
 FROM student AS s
 JOIN term_gpa AS t
        ON s.id=t.id
 LEFT JOIN degrees AS d
        ON d.id=s.id  AND
        t.term=d.term ;
```

```
mysql> SELECT s.id AS id , s.name AS name , t
    ->  FROM student AS s  JOIN term_gpa AS t
    ->  LEFT JOIN degrees AS d ON d.id=s.id A
+----+--------------+------+--------+------+
| id | name         | gpa  | degree | term |
+----+--------------+------+--------+------+
|  1 | Edith Warton | 3.32 | NULL   | 2011 |
|  1 | Edith Warton | 3.51 | EconS  | 2012 |
|  2 | Dorian Gray  | 2.22 | NULL   | 2011 |
|  2 | Dorian Gray  | 1.70 | NULL   | 2013 |
|  3 | Ophelia      | 3.70 | CompS  | 2011 |
|  3 | Ophelia      | 3.70 | MathS  | 2011 |
|  4 | Henry James  | 3.10 | NULL   | 2011 |
|  4 | Henry James  | 3.21 | EngLT  | 2012 |
|  4 | Henry James  | 3.30 | NULL   | 2013 |
|  5 | Harper Lee   | 2.99 | NULL   | 2013 |
+----+--------------+------+--------+------+
10 rows in set (0.00 sec)
```

Q: Get the students data(name, gpa, degree) for term 2012

Query:

```
SELECT s.id AS id , s.name AS
      name , t.gpa    AS gpa ,
      d.degree AS  degree , t.term as
      term
FROM student AS s
JOIN term_gpa AS t
                  ON s.id=t.id
LEFT JOIN degrees AS d
      ON d.id=s.id AND
      t.term=d.term
WHERE  t.term=2012;
```

```
mysql> SELECT s.id AS id , s.name AS name , t.gpa AS gpa , d.degree A
    -> FROM student AS s
    -> JOIN term_gpa AS t ON s.id=t.id
    -> LEFT JOIN degrees AS d ON d.id=s.id AND t.term=d.term
    -> WHERE  t.term=2012;
+----+--------------+------+--------+------+
| id | name         | gpa  | degree | term |
+----+--------------+------+--------+------+
|  1 | Edith Warton | 3.51 | EconS  | 2012 |
|  4 | Henry James  | 3.21 | EngLT  | 2012 |
+----+--------------+------+--------+------+
2 rows in set (0.00 sec)

mysql>
```

Q: Get the students data(name, gpa, degree) for term 2012

```
SELECT s.id AS id , s.name AS name ,
    t.gpa    AS gpa , d.degree AS
    degree , t.term as term
FROM student AS s
JOIN term_gpa AS t
    ON s.id=t.id
LEFT JOIN degrees AS d
    ON d.id=s.id AND
    t.term=d.term
WHERE  t.term=2012;
```

```
SELECT s.id AS id , s.name AS name ,
    t.gpa    AS gpa , d.degree AS
    degree , t.term as term
FROM student AS s
JOIN term_gpa AS t
    ON s.id=t.id
LEFT JOIN degrees AS d
    ON d.id=s.id AND
    t.term=d.term
    AND  t.term=2012;
```

Q: Get the students data(name, gpa, degree) for term 2012

Query:

```
SELECT s.id AS id , s.name AS
    name , t.gpa AS gpa , d.degree
    AS  degree , t.term as term
FROM student AS s
JOIN term_gpa AS t
    ON s.id=t.id
LEFT JOIN degrees AS d
    ON d.id=s.id AND
    t.term=d.term
    AND  t.term=2012;
```

```
mysql> SELECT s.id AS id , s.name AS     name , t.g
    -> FROM student AS s
    -> JOIN term_gpa AS t
    -> ON s.id=t.id
    -> LEFT JOIN degrees AS d
    -> ON d.id=s.id AND t.term=d.term
    ->  AND  t.term=2012;
+----+--------------+------+--------+------+
| id | name         | gpa  | degree | term |
+----+--------------+------+--------+------+
|  1 | Edith Warton | 3.32 | NULL   | 2011 |
|  1 | Edith Warton | 3.51 | EconS  | 2012 |
|  2 | Dorian Gray  | 2.22 | NULL   | 2011 |
|  2 | Dorian Gray  | 1.70 | NULL   | 2013 |
|  3 | Ophelia      | 3.70 | NULL   | 2011 |
|  4 | Henry James  | 3.10 | NULL   | 2011 |
|  4 | Henry James  | 3.21 | EngLT  | 2012 |
|  4 | Henry James  | 3.30 | NULL   | 2013 |
|  5 | Harper Lee   | 2.99 | NULL   | 2013 |
+----+--------------+------+--------+------+
9 rows in set (0.00 sec)
```

Q: Get the data of students who have failed at least once.

Query:

```
SELECT s.id AS id , MAX( s.name ) AS name , AVG( t.gpa ) AS gpa
FROM student AS s JOIN term_gpa AS t ON s.id=t.id
WHERE t.gpa < 3
GROUP BY s.id
HAVING count(t.gpa) >= 1;
```

```
mysql> SELECT s.id AS id , MAX( s.name ) AS name , AVG( t.gpa ) AS gpa
    -> FROM student AS s JOIN term_gpa AS t ON s.id=t.id
    -> WHERE t.gpa < 3
    -> GROUP BY s.id
    -> HAVING count(t.gpa) >= 1;
+----+-------------+----------+
| id | name        | gpa      |
+----+-------------+----------+
|  2 | Dorian Gray | 1.960000 |
|  5 | Harper Lee  | 2.990000 |
+----+-------------+----------+
2 rows in set (0.00 sec)
```

Q: Find the second highest grades

Query 1:

```
SELECT MAX(gpa) AS GPA
FROM term_gpa AS t
WHERE t.gpa not in ( SELECT MAX(gpa)  FROM term_gpa );
```

Query \ No of Rows	50k	100k	3.5 Lac	12.5 Million
Q1	0.2 Sec	0.23 sec	0.3 Sec	81 Sec
Q2	69 Sec	300 Sec	? > 2+ Hrs	? > 3+ Hrs
Q3	0.2 Sec	0.2 Sec	0.27 Sec	41 Sec

Query 2:

```
SELECT t1.gpa AS GPA
FROM term_gpa AS t1
WHERE 2 = (
      SELECT COUNT ( DISTINCT ( gpa ) ) FROM term_gpa AS t2
                  WHERE t1.gpa <= t2.gpa
                   );
```

Query 3:

```
SELECT  GPA
FROM ( SELECT gpa FROM term_gpa AS t ORDER BY gpa DESC LIMIT 2 )
       AS temp_table
ORDER BY GPA
LIMIT 1;
```

Case example: Categorizing data

```
SELECT id,  name, avg_gpa, CASE
        WHEN avg_gpa >=3.5 THEN 2
        WHEN avg_gpa <3.5 AND avg_gpa
                        >=3.0 THEN 1
        ELSE 0
        END AS gpa_type
FROM (
        SELECT t.id as id, MAX(s.name) AS
        name, AVG( gpa ) AS avg_gpa
        FROM term_gpa as t
        JOIN student as s ON t.id = s.id
        GROUP BY id
        ) AS avg_gpa_table
```

```
ysql> SELECT id,  name,avg_gpa, CASE WHEN a
s name, AVG( gpa ) AS avg_gpa FROM term_gpa
+----+--------------+----------+----------+
| id | name         | avg_gpa  | gpa_type |
+----+--------------+----------+----------+
|  1 | Edith Warton | 3.415000 |        1 |
|  2 | Dorian Gray  | 1.960000 |        0 |
|  3 | Ophelia      | 3.700000 |        2 |
|  4 | Henry James  | 3.203333 |        1 |
|  5 | Harper Lee   | 2.990000 |        0 |
+----+--------------+----------+----------+
rows in set (0.00 sec)

ysql>
```

Pattern Matching:

LIKE Operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

WildCard Characters:

- [*charlist*] - Defines sets and ranges of characters to match
- [^*charlist*] or [!*charlist*] - Defines sets and ranges of characters NOT to match

Pattern Matching E.g.:

```
SELECT id , name
 FROM student AS s
 WHERE name LIKE 'Dorian%' ;
```

```
mysql> SELECT id , name
    ->  FROM student AS s
    ->  WHERE name LIKE 'Dorian%' ;
+----+-------------+
| id | name        |
+----+-------------+
|  2 | Dorian Gray |
+----+-------------+
1 row in set (0.00 sec)
```

Before Indexing	39 Sec
After Indexing	0.27 Sec

```
SELECT id , name
 FROM student AS s
 WHERE name LIKE '%Gray' ;
```

```
mysql> SELECT id , name
    ->  FROM student AS s
    ->  WHERE name LIKE '%Gray' ;
+----+-------------+
| id | name        |
+----+-------------+
|  2 | Dorian Gray |
+----+-------------+
1 row in set (0.01 sec)
```

Before Indexing	39 Sec
After Indexing	First run: 5.2 Sec
	Next runs: 1.2 Sec

Logical Processing Order

SELECT *select_list* [INTO *new_table*]
[FROM *table_source*]
[WHERE *search_condition*]
[GROUP BY *group_by_expression*]
[HAVING *search_condition*]
[ORDER BY *order_expression* [ASC | DESC]]

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE or WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. TOP/ LIMIT/ ROWNUM

References:

- https://dev.mysql.com/doc/refman/8.0/en/
- https://docs.microsoft.com/en-us/sql/t-sql/queries/
- https://bensresearch.com/
- Etc.

Connect me:

: in/ruturaj-p-Nalawade

: n.ruturaj@outlook.com