

SQL IN 45 MINUTES

UCSD DSC 96, Fall 2019

Colin Jemmott

Actually mostly stolen from [MODE](#).



WHY SQL?

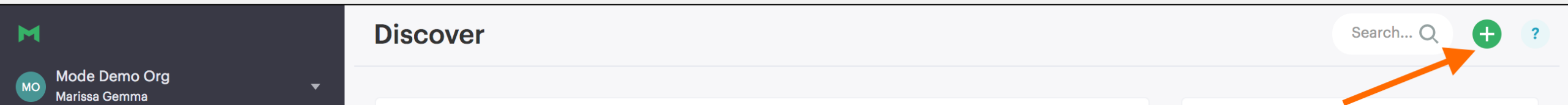
- It's semantically easy to understand and learn.
- Because it can be used to access large amounts of data directly where it's stored, analysts don't have to copy data into other applications.
- Compared to spreadsheet tools, data analysis done in SQL is easy to audit and replicate.

SQL is great for performing the types of aggregations that you might normally do with dataframes—sums, counts, minimums and maximums, etc.—but over much larger datasets.

M MODE

To follow along:

1. Go to <https://mode.com/>
2. Create an account
3. Click “new report” (see below)



CENSUS HOUSING UNITS

```
SELECT * FROM tutorial.us_housing_units
```

MO Personal > Untitled Report
Report ▾ Embed Share Schedule View

Untitled Report

Query 1 SQL

Run ☒ Limit 100 Format SQL View History...

```
1 SELECT *  
2 FROM tutorial.us_housing_units
```

Mode Public Warehouse ▾

Tables Definitions

Search public

TABLES

► tutorial

YOU CAN RUN A QUERY BY PRESSING ⌘ +
RETURN ON A MAC OR CTRL + RETURN ON A PC

SELECT AND FROM

```
SELECT year,  
       month,  
       west  
FROM tutorial.us_housing_units
```


WHAT ACTUALLY HAPPENS WHEN YOU RUN A QUERY?

- Query is sent to database
- Only results are sent back
- Database is not modified (with SELECT)

FORMATTING

- White space doesn't matter
- Capitalization doesn't matter
 - But there are strong conventions for `COMMANDS` to be all caps.

COLUMN NAMES

```
SELECT west AS "West Region"  
FROM tutorial.us_housing_units
```

```
SELECT west AS West_Region,  
       south AS South_Region  
FROM tutorial.us_housing_units
```

LIMIT

```
SELECT *  
  FROM tutorial.us_housing_units  
 LIMIT 15
```

WHERE

```
SELECT *  
  FROM tutorial.us_housing_units  
 WHERE month = 1
```

COMPARISON OPERATORS

```
SELECT *  
  FROM tutorial.us_housing_units  
 WHERE west > 30
```

Equal to	=
Not equal to	<> or !=
Greater than	>
Less than	<
Greater than or equal to	>=
Less than or equal to	<=

ARITHMETIC

```
SELECT year,  
       month,  
       west,  
       south,  
       (west + south)/2 AS south_west_avg  
FROM tutorial.us_housing_units
```

BILLBOARD TOP 100

```
SELECT * FROM tutorial.billboard_top_100_year_end
```


EXPLORING THE DATA

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 ORDER BY year DESC, year_rank
```

LIKE

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 WHERE "group" LIKE 'Snoop%'
```

- % is wildcard (_ for one char)
- LIKE is case sensitive
- Double quotes on group because that is a function in SQL!
- To ignore case when you're matching values, you can use the ILIKE command

IN

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 WHERE year_rank IN (1, 2, 3)
```

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 WHERE artist IN ('Taylor Swift', 'Usher', 'Ludacris')
```

BETWEEN

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 WHERE year_rank BETWEEN 5 AND 10
```

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 WHERE year_rank >= 5 AND year_rank <= 10
```

NULL

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 WHERE artist IS NULL
```

- `WHERE artist = NULL` will not work—you can't perform arithmetic on null values.

AND

OR

NOT

ORDER BY

```
SELECT *  
  FROM tutorial.billboard_top_100_year_end  
 WHERE year = 2013  
 ORDER BY year_rank DESC
```

APPLE STOCK PRICES

```
SELECT * FROM tutorial.aapl_historical_stock_price
```


COUNT

```
SELECT COUNT (*)  
FROM tutorial.aapl_historical_stock_price
```

- Non-distinct count
- Does not count nulls

SUM, MIN, MAX, AVG

```
SELECT SUM(volume)
FROM tutorial.aapl_historical_stock_price
```

```
SELECT MIN(volume) AS min_volume,
       MAX(volume) AS max_volume
FROM tutorial.aapl_historical_stock_price
```


GROUP BY

```
SELECT year,  
       COUNT(*) AS count  
FROM tutorial.aapl_historical_stock_price  
GROUP BY year
```

```
SELECT year,  
       month,  
       COUNT(*) AS count  
FROM tutorial.aapl_historical_stock_price  
GROUP BY year, month
```

HAVING

```
SELECT year,  
       month,  
       MAX(high) AS month_high  
FROM tutorial.aapl_historical_stock_price  
GROUP BY year, month  
HAVING MAX(high) > 400  
ORDER BY year, month
```



QUERY CLAUSE ORDER

1. SELECT
2. FROM
3. WHERE
4. GROUP BY
5. HAVING
6. ORDER BY

DISTINCT

```
SELECT DISTINCT month  
FROM tutorial.aapl_historical_stock_price
```

```
SELECT COUNT(DISTINCT month) AS unique_months  
FROM tutorial.aapl_historical_stock_price
```



Can be SLOOOOOW

CRUNCHBASE

```
SELECT * FROM benn.college_football_players
```

JOINS

```
SELECT companies.permalink AS companies_permalink,  
       companies.name AS companies_name,  
       acquisitions.company_permalink AS  
acquisitions_permalink,  
       acquisitions.acquired_at AS acquired_date  
FROM tutorial.crunchbase_companies companies  
JOIN tutorial.crunchbase_acquisitions acquisitions  
  ON companies.permalink = acquisitions.company_permalink
```


JOIN AS VENN DIAGRAM VS JOIN AS INNER PRODUCT

Confused about joins? Click [here](#).

MORE JOINS

- Inner, outer, left, right
- Joins using WHERE or ON
- FULL OUTER JOIN
- UNION
- Joins with comparison operators
- Joins on Multiple Keys
- Self joins



ADVANCED SQL TOPICS

- Data Types
- Date Format
- Data Wrangling with SQL
- Writing Subqueries in SQL
- Window Functions
- Performance Tuning SQL Queries

[link](#)