SQL SQL SQL

Queries

CREATE TABLE books(id INTEGER PRIMARY KEY, name TEXT, rating INTEGER);
INSERT INTO books VALUES (1, "The old man and the sea", 10);

```
SELECT
        COLUMN_NAME(S)
FROM
        TABLE_NAME
WHERE
        CONDITION
        LIKE, IN, BETWEEN
GROUP BY
        COLUMN_NAME(S)
HAVING
        AGGREGATE_CONDITION
ORDER BY
        COLUMN_NAME
LIMIT n
```

COUNT all rows

If you want to know all the values in the entire table use COUNT(*) you will get a single number.
SELECT COUNT(*)

COUNT DISTINCT values in a column

```
SELECT
        COUNT (DISTINCT COLUMN_NAME)
```

Conditions

```
WHERE FIRSTNAME     = 'BOB'     -- exact match
WHERE FIRSTNAME     != 'BOB'    -- everything excluding BOB
WHERE NOT FIRSTNAME  ='BOB'    -- everything excluding BOB

WHERE FIRSTNAME IN ('BOB', 'JASON')
WHERE FIRSTNAME = 'BOB' OR FIRSTNAME = 'JASON'   --  either condition is met
WHERE FIRSTNAME NOT IN ('BOB', 'JASON')   -- excludes both values
WHERE FIRSTNAME = 'BOB' AND LASTNAME = 'SMITH'  -- both conditions

WHERE GRADES  >= 90        -- greater than or equal to 90

WHERE SUBJECT IS NULL      -- returns values with missing values
WHERE SUBJECT NOT NULL     -- returns values with no missing values
```

Like Wildcads

LIKE operator is used in a WHERE clause to search for a specified pattern in a column. When you pass the LIKE operator in the '' upper and lower case matters.

% - represents zero, one, or multiple characters
_ - represents a single character

```
WHERE FIRSTNAME LIKE '_n%'
-- find values with an "n" in the second position
WHERE FIRSTNAME LIKE '[!BFL]%'
-- find everything exlcusing values that start with 'B', 'F' OR 'L'
```

GROUP BY

The GROUP BY function helps calculate summary values by the chosen column. It is often used with aggregate functions (COUNT, SUM, AVG, MAX, MIN).

```
SELECT subject, AVG(grades) AS avg_grades
FROM students
GROUP BY subject
HAVING avg_grades >= 10
```

When

```
SELECT COUNT(*),
   CASE
      WHEN heart_rate > ROUND(0.90 * (220-30))
THEN "above target"
      WHEN heart_rate > ROUND(0.50 * (220-30))
THEN "within target"
      ELSE "below target"
   END as "hr_zone"
FROM exercise_logs
```

Joins

Inner join, Left join

(Right join), Full Outer join

```
SELECT contact_name, company_name

FROM contacts

    LEFT JOIN companies
```

SQL SQL SQL
ON contacts.company_id = companies.company_id

AND companies.company_id=1001

Unline WHERE clause, it still shows all contacts, but only adds the company details where the id is 1001.

Aggregation

… WHERE teacher IS NULL

COUNT(teacher)

Individual values without NULL

SELECT SUM(standard_amt)/ SUM(standard_qty) AS standard_price

MIN(), MAX(), AVG()

GROUP BY

SELECT r.name, w.channel, COUNT(channel) num_events

FROM accounts a

JOIN web_events w

ON a.id = w.account_id

GROUP BY r.name, w.channel

ORDER BY num_events DESC;

Example: Which account used facebook most as a channel?

```
SELECT a.id, a.name, w.channel, COUNT(*)
use_of_channel
FROM accounts a
JOIN web_events w
ON a.id = w.account_id
WHERE w.channel = 'facebook'
GROUP BY a.id, a.name, w.channel
ORDER BY use_of_channel DESC
LIMIT 1;
```

HAVING

Used with GROUP BY

Like WHERE but it can use SUM(),COUNT(), …

DATE

YYYY-MM-DD hh-mm-ss

SELECT DATE_TRUNC('day',occurred_at)

… GROUP BY DATE_TRUNC('day',occurred_at)

SELECT DATE_PART('day',2017-04-01 12:15:01)

Each Tuesday in all weeks and years

Example: Find the sales in terms of total dollars for all orders in each year, ordered from greatest to least. Do you notice any trends in the yearly sales totals?

```
SELECT DATE_PART('year', occurred_at) ord
_year,  SUM(total_amt_usd) total_spent
FROM orders
GROUP BY 1
ORDER BY 2 DESC;
```

CASE WHEN - if

Similar to "if"

SELECT CASE WHEN abc OR cde THEN x
WHEN asd THEN y ELSE z END AS res

Subqueries

Avg number of events/ day for each channel.

```
SELECT channel, AVG(events) AS average_eve
nts
FROM (SELECT DATE_TRUNC('day',occurred_a
t) AS day, channel, COUNT(*) as events
      FROM web_events
      GROUP BY 1,2) sub
GROUP BY channel
ORDER BY 2 DESC;
```

WITH - Common Table Expression CTE

Cleaner & more common than subquery.

What is the lifetime AVG amount total_amt_usd, including only the companies that spent more per order, on average, than the average of all orders

```
WITH t1 AS (
   SELECT AVG(o.total_amt_usd) avg_all
   FROM orders o
   JOIN accounts a
   ON a.id = o.account_id),
t2 AS (
   SELECT o.account_id, AVG(o.total_amt_u
sd) avg_amt
   FROM orders o
   GROUP BY 1
   HAVING AVG(o.total_amt_usd) > (SELECT
* FROM t1))
SELECT AVG(avg_amt)
FROM t2;
```

SQL SQL SQL

Data Cleaning

Left & Right

you can pull the first three digits of a phone number using

LEFT(phone_number, 3) → 078

RIGHT(phone_number, 3) → 721

LENGTH(abc) counts characters

LOWER(abc), UPPER(abc)

STRPOS(phone, '4') counted from the left

CONCAT(first,' ',last) → Marvin Grass

first || ' ' || last → Marvin Grass

COALESCE replace no data, NULL

COALESCE(address, 'none')

Window functions

Over

OVER makes running totals (total until 1, 2, …)

Calculated the sum of quanties occurred until then

```
SELECT standard_amt_usd,
       DATE_TRUNC('year', occurred_at) as
year,
SUM(standard_amt_usd) OVER (PARTITION BY
DATE_TRUNC('year', occurred_at)
ORDER BY occurred_at) AS running_total
FROM orders
```

PARTITION BY

Each partition is COUNT,SUM,… separately.

ROW_NUMBER adds row numbers

ROW_NUMBER() OVER (PARTITION BY account_id ORDER BY occurred_at) AS row_num

RANK() will give same rows same rank

WINDOW

```
MIN(tot_usd) OVER acc_window AS min_amt,
MAX(tot_usd) OVER acc_window AS max_amt
FROM orders
WINDOW acc_window AS (PARTITION BY accoun
t_id ORDER BY DATE_TRUNC('year',occurred_
at))
```

Between WHERE and GROUP BY

LAG and LEAD

It returns the value from a previous row to the current row in the table.

Percentiles

NTILE(100) percentiles

NTILE(4) quartile

Advanced Joins

FULL OUTER JOIN

To get outer without inner join

```
  LEFT JOIN sales_reps
    ON accounts.sales_rep_id = sales_rep
s.id
   AND accounts.primary_poc < sales_reps.
name
```

SELF JOIN

One of the most common use cases for self JOINs is in cases where two events occurred, one after another.

UNION

The UNION operator is used to combine the result sets of 2 or more SELECT statements. Both same no. of columns, data types Appending.

UNION	removes duplicate rows
UNION ALL	all rows appended

```
SELECT *    FROM accounts
UNION ALL
SELECT *  FROM accounts
```

Performance Tuning

Making queries run more quickly

Aggregate data before joining

EXPLAIN SELECT….

To find out how long it will run