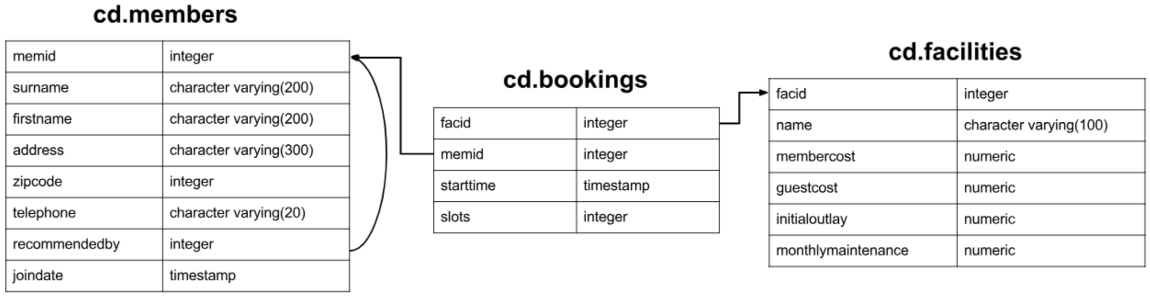
SQL Weekly Challenges

Week 1



cd.members

memid	integer
surname	character varying(200)
firstname	character varying(200)
address	character varying(300)
zipcode	integer
telephone	character varying(20)
recommendedby	integer
joindate	timestamp

cd.bookings

facid	integer
memid	integer
starttime	timestamp
slots	integer

cd.facilities

facid	integer
name	character varying(100)
membercost	numeric
guestcost	numeric
initialoutlay	numeric
monthlymaintenance	numeric

```
-- 1. How can you produce a list of facilities that charge a fee to members?
select * from cd.facilities where membercost > 0;

-- 2. How can you produce a list of all facilities with the word 'Tennis' in
their name?
select * from cd.facilities where name like '%Tennis%'

-- 3. How can you retrieve the details of facilities with ID 1 and 5? Try to
do it without using the OR operator.
select * from cd.facilities where facid in (1, 5);
```

lyft_drivers
index: int
start_date: datetime
end_date: datetime
yearly_salary: int

```
-- 4. Find all Lyft drivers who earn either equal to or less than 30k USD or
equal to or more than 70k USD. Output all details related to retrieved
records.
select * from lyft_drivers where yearly_salary <= 30000 or yearly_salary >=
70000;
```

oscar_nominees
year: int
category: varchar
nominee: varchar
movie: varchar
winner: bool
id: int

```
-- 5. Count the number of movies that Abigail Breslin was nominated for an
oscar.
select count(*) as n_movies_by_abi from oscar_nominees where nominee =
'Abigail Breslin';
```

Week 2

customers
id: int
first_name: varchar
last_name: varchar
city: varchar
address: varchar
phone_number: varchar

orders
id: int
cust_id: int
order_date: datetime
order_details: varchar
total_order_cost: int

```
-- 1. Find the total cost of each customer's orders.
-- Output customer's id, first name, and the total order cost.
-- Order records by customer's first name alphabetically.
select o.cust_id, c.first_name, sum(o.total_order_cost) as total_order_cost
from orders o inner join customers c on o.cust_id = c.id
group by o.cust_id, c.first_name order by c.first_name;
```

airbnb_hosts
host_id: int
nationality: varchar
gender: varchar
age: int

airbnb_units
host_id: int
unit_id: varchar
unit_type: varchar
n_beds: int
n_bedrooms: int
country: varchar
city: varchar

```
-- 2. Find the number of apartments per nationality that are owned by people
under 30 years old.
-- Output the nationality along with the number of apartments.
-- Sort records by the apartments count in descending order.
select h.nationality, count(distinct u.unit_id) as apartment_count from
airbnb_hosts h inner join airbnb_units u on h.host_id = u.host_id
where h.age < 30 and u.unit_type = 'Apartment' group by h.nationality order by
count(u.unit_id) desc;
```

hotel_reviews
hotel_address: varchar
additional_number_of_scoring: int
review_date: datetime
average_score: float
hotel_name: varchar
reviewer_nationality: varchar
negative_review: varchar
review_total_negative_word_counts: int
total_number_of_reviews: int
positive_review: varchar
review_total_positive_word_counts: int
total_number_of_reviews_reviewer_has_given: int
reviewer_score: float
tags: varchar
days_since_review: varchar
lat: float
lng: float

```
-- 3. Find the number of rows for each review score earned by 'Hotel Arena'.
-- Output the hotel name (which should be 'Hotel Arena'), review score along
with the
-- corresponding number of rows with that score for the specified hotel.
select hotel_name, reviewer_score, count(*) as count from hotel_reviews where
hotel_name = 'Hotel Arena' group by hotel_name, reviewer_score;
```

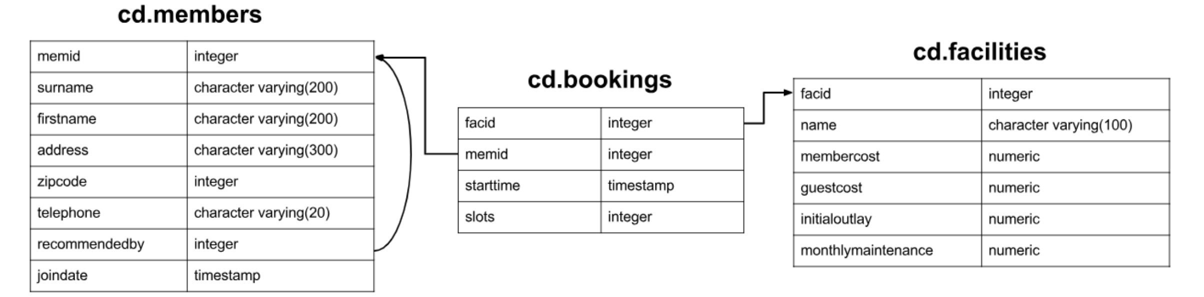
Week 3

```
-- 1. Find the total cost of each customer's orders.
-- Output customer's id, first name, and the total order cost.
-- Order records by customer's first name alphabetically.
select o.cust_id, c.first_name, sum(o.total_order_cost) as total_order_cost
from orders o inner join customers c on o.cust_id = c.id
group by o.cust_id, c.first_name order by c.first_name;
```

zillow_transactions
id: int
state: varchar
city: varchar
street_address: varchar
mkt_price: int

```
-- 2. Write a query that identifies cities with higher than average home
prices when compared to the national average. Output the city names.
SELECT city FROM zillow_transactions GROUP BY city
HAVING AVG(mkt_price) > (
    SELECT AVG(mkt_price)
    FROM zillow_transactions
)
ORDER BY AVG(mkt_price) DESC;

-- 3. Find the number of rows for each review score earned by 'Hotel Arena'.
-- Output the hotel name (which should be 'Hotel Arena'), review score along
with the
-- corresponding number of rows with that score for the specified hotel.
select hotel_name, reviewer_score, count(*) as count from hotel_reviews where
hotel_name = 'Hotel Arena' group by hotel_name, reviewer_score;
```

Week 4



cd.members

memid	integer
surname	character varying(200)
firstname	character varying(200)
address	character varying(300)
zipcode	integer
telephone	character varying(20)
recommendedby	integer
joindate	timestamp

cd.bookings

facid	integer
memid	integer
starttime	timestamp
slots	integer

cd.facilities

facid	integer
name	character varying(100)
membercost	numeric
guestcost	numeric
initialoutlay	numeric
monthlymaintenance	numeric

```
-- 1. How can you produce a list of facilities, with each labelled as 'cheap'
or 'expensive'
-- depending on if their monthly maintenance cost is more than $100?
-- Return the name and monthly maintenance of the facilities in question.
select name, case when monthlymaintenance > 100 then 'expensive' else 'cheap'
end as cost from cd.facilities;
```

```
-- 2. How can you produce a list of members who joined after the start of
September 2012?
-- Return the memid, surname, firstname, and joindate of the members in
question.
select memid, surname, firstname, joindate from cd.members where
date(joindate) >= date('2012-09-01');
```

```
-- 3. How can you produce a list of the start times for bookings by members
named 'David Farrell'?
select distinct starttime from cd.bookings b inner join cd.members m on
b.memid = m.memid where
m.surname = 'Farrell' and m.firstname = 'David';
```

Week 5

```
-- 1. Produce a count of the number of facilities that have a cost to guests of 10 or more.
select count(*) from cd.facilities where guestcost >= 10;

-- 2. Produce a count of the number of recommendations each member has made. Order by member ID.
select recommendedby, count(*) as count from cd.members where recommendedby is not null group by recommendedby order by recommendedby;

-- 3. Produce a list of facilities with more than 1000 slots booked. Produce an output table consisting of facility id and slots,
-- sorted by facility id.
select facid, sum(slots) as "Total Slots" from cd.bookings group by facid having sum(slots) > 1000 order by facid;
```

Week 6

```
-- 1. How can you produce a list of bookings on the day of 2012-09-14 which will cost the member (or guest) more than $30?
-- Remember that guests have different costs to members (the listed costs are per half-hour 'slot'), and the guest user is always ID 0.
--Include in your output the name of the facility, the name of the member formatted as a single column, and the cost.
-- Order by descending cost, and do not use any subqueries.
select m.firstname || ' ' || m.surname as member, f.name as facility,
case
when m.memid = 0 then b.slots * f.guestcost
else b.slots * f.membercost
end as cost from
cd.members m inner join cd.bookings b on m.memid = b.memid inner join cd.facilities f on b.facid = f.facid
where b.starttime >= '2012-09-14' and b.starttime < '2012-09-15' and
((m.memid = 0 and b.slots * f.guestcost > 30) or (m.memid != 0 and b.slots * f.membercost > 30))
order by cost desc;

-- 2. Produce a list of the total number of slots booked per facility in the month of September 2012.
-- Produce an output table consisting of facility id and slots, sorted by the number of slots.
select facid, sum(slots) as "Total Slots" from cd.bookings
where starttime >= '2012-09-01'
        and starttime < '2012-10-01'
group by facid order by sum(slots);
```

```
-- 3. How can you produce a list of all members who have used a tennis court?
-- Include in your output the name of the court, and the name of the member
formatted as a single column.
-- Ensure no duplicate data, and order by the member name followed by the
facility name.

select distinct m.firstname || ' ' || m.surname as member, f.name as facility
from
cd.members m inner join cd.bookings b on m.memid = b.memid inner join
cd.facilities f on b.facid = f.facid where f.name like '%Tennis Court%'
order by m.firstname || ' ' || m.surname, f.name;
```

Week 7

```
-- 1. Find the result of subtracting the timestamp '2012-07-30 01:00:00' from
the timestamp '2012-08-31 01:00:00'
select timestamp '2012-08-31 01:00:00' - timestamp '2012-07-30 01:00:00' as
interval;

-- 2. Return a count of bookings for each month, sorted by month
select date_trunc('month', starttime) as month, count(*) as count from
cd.bookings group by month order by month;
```

Week 8

```
-- 1. You'd like to get the first and last name of the last member(s) who
signed up - not just the date. How can you do that?
select firstname, surname, joindate from cd.members order by joindate desc
limit 1;

-- 2. How can you output a list of all members, including the individual who
recommended them (if any)?
-- Ensure that results are ordered by (surname, firstname).
select mem.firstname as memfname, mem.surname as memsname,
ref.firstname as recfname, ref.surname as recsname
from cd.members mem left outer join cd.members ref on mem.recommendedby =
ref.memid
order by mem.surname, mem.firstname;
```

Week 9

```
-- 1. Produce a list of facilities with more than 1000 slots booked. Produce
an output table consisting of facility id and slots,
-- sorted by facility id.
select facid, sum(slots) as "Total Slots" from cd.bookings group by facid
having sum(slots) > 1000 order by facid;

-- 2. Produce a list of facilities along with their total revenue.
-- The output table should consist of facility name and revenue, sorted by
revenue.
-- Remember that there's a different cost for guests and members!
select f.name, sum(
  case when b.memid = 0 then b.slots*f.guestcost
  else b.slots*f.membercost end
  ) as revenue from cd.bookings b inner join cd.facilities f on b.facid =
f.facid group by f.name order by revenue;
```

Week 10

```
-- 1. The telephone numbers in the database are very inconsistently formatted.
You'd like to print a list of member ids and numbers that have had
-- '-','(',')', and ' ' characters removed. Order by member id.
select memid, translate(telephone, '-() ', '') as telephone from cd.members
order by memid;

-- 2. Output the names of all members, formatted as 'Surname, Firstname'
select surname || ', ' ||firstname as name from cd.members;
```

Week 11

```
-- 1. The telephone numbers in the database are very inconsistently formatted.
You'd like to print a list of member ids and numbers that have had
-- '-','(',')', and ' ' characters removed. Order by member id.
select memid, translate(telephone, '-() ', '') as telephone from cd.members
order by memid;

-- 2. Produce a list of each member name, id, and their first booking after
September 1st 2012. Order by member ID.
select m.surname, m.firstname, m.memid, min(b.starttime) from cd.members m
inner join cd.bookings b
on m.memid = b.memid where b.starttime >= timestamp '2012-09-01' group by
m.surname, m.firstname, m.memid order by m.memid;
```

Week 12

```
-- 1. Return a count of bookings for each month, sorted by month
select date_trunc('month', starttime) as month, count(*) as count from
cd.bookings group by month order by month;

-- 2. Produce a list of each member name, id, and their first booking after
September 1st 2012. Order by member ID.
select m.surname, m.firstname, m.memid, min(b.starttime) from cd.members m
inner join cd.bookings b
on m.memid = b.memid where b.starttime >= timestamp '2012-09-01' group by
m.surname, m.firstname, m.memid order by m.memid;
```

Week 13

```
Table: Person

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| personId    | int     |
| lastName    | varchar |
| firstName   | varchar |
+-------------+---------+
personId is the primary key column for this table.
This table contains information about the ID of some persons and their first
and last names.

Table: Address

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| addressId   | int     |
| personId    | int     |
| city        | varchar |
| state       | varchar |
+-------------+---------+
addressId is the primary key column for this table.
Each row of this table contains information about the city and state of one
person with ID = PersonId.

-- 1. Write an SQL query to report the first name, last name, city, and state
of each person in the Person table.
-- If the address of a personId is not present in the Address table, report
null instead.
-- Return the result table in any order.
```

```
select firstName, lastName, city, state from Person p left outer join Address
a on p.personId = a.personId;

Table: Employee

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| salary      | int     |
| managerId   | int     |
+-------------+---------+
id is the primary key column for this table.
Each row of this table indicates the ID of an employee, their name, salary,
and the ID of their manager.

-- 2. Write an SQL query to find the employees who earn more than their
managers.
-- Return the result table in any order.
select a.name as Employee from Employee as a, Employee as b where a.Managerid
= b.Id and a.Salary > b.Salary;

Table: Customers

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
+-------------+---------+
In SQL, id is the primary key column for this table.
Each row of this table indicates the ID and name of a customer.


Table: Orders

+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| customerId  | int  |
+-------------+------+
In SQL, id is the primary key column for this table.
customerId is a foreign key (join key in Pandas) of the ID from the Customers
table.
Each row of this table indicates the ID of an order and the ID of the customer
who ordered it.
```

```
--3. Find all customers who never order anything.
-- Return the result table in any order.
select c.name as Customers from customers c where c.id not in (select distinct
customerId from orders);
```

Week 14

```
Table: Person

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| email       | varchar |
+-------------+---------+
id is the primary key column for this table.
Each row of this table contains an email. The emails will not contain
uppercase letters.

-- 1. Write an SQL query to report all the duplicate emails. Note that it's
guaranteed that the email field is not NULL.
-- Return the result table in any order.
select distinct Email from person group by email having count(email) > 1;

-- 2. Produce a list of each member name, id, and their first booking after
September 1st 2012. Order by member ID.
select m.surname, m.firstname, m.memid, min(b.starttime) from cd.members m
inner join cd.bookings b
on m.memid = b.memid where b.starttime >= timestamp '2012-09-01' group by
m.surname, m.firstname, m.memid order by m.memid;

Table: World

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| name        | varchar |
| continent   | varchar |
| area        | int     |
| population  | int     |
| gdp         | bigint  |
+-------------+---------+
In SQL, name is the primary key column for this table.
Each row of this table gives information about the name of a country, the
continent to which it belongs, its area, the population, and its GDP value.
A country is big if:
```

```
it has an area of at least three million (i.e., 3000000 km2), or
it has a population of at least twenty-five million (i.e., 25000000).

-- 3. Find the name, population, and area of the big countries.
-- Return the result table in any order.
select name, population, area from World where area >= 3000000 or population
>= 25000000;
```

Week 15

```
Table: Cinema

+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| id             | int      |
| movie          | varchar  |
| description    | varchar  |
| rating         | float    |
+----------------+----------+
id is the primary key for this table.
Each row contains information about the name of a movie, its genre, and its
rating.
rating is a 2 decimal places float in the range [0, 10]

-- 1. Write an SQL query to report the movies with an odd-numbered ID and a
description that is not "boring".
-- Return the result table ordered by rating in descending order.
select * from Cinema where (id % 2 <> 0) and (description <> 'boring') order
by rating desc;

Table: SalesPerson

+-----------------+---------+
| Column Name     | Type    |
+-----------------+---------+
| sales_id        | int     |
| name            | varchar |
| salary          | int     |
| commission_rate | int     |
| hire_date       | date    |
+-----------------+---------+
In SQL, sales_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a salesperson
alongside their salary, commission rate, and hire date.
```

```
Table: Company

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| com_id      | int     |
| name        | varchar |
| city        | varchar |
+-------------+---------+
In SQL, com_id is the primary key column for this table.
Each row of this table indicates the name and the ID of a company and the city
in which the company is located.


Table: Orders

+-------------+------+
| Column Name | Type |
+-------------+------+
| order_id    | int  |
| order_date  | date |
| com_id      | int  |
| sales_id    | int  |
| amount      | int  |
+-------------+------+
In SQL, order_id is the primary key column for this table.
com_id is a foreign key (join key in Pandas) to com_id from the Company table.
sales_id is a foreign key (join key in Pandas) to sales_id from the
SalesPerson table.
Each row of this table contains information about one order. This includes the
ID of the company, the ID of the salesperson, the date of the order, and the
amount paid.

-- 2. Find the names of all the salespersons who did not have any orders
related to the company with the name "RED".
-- Return the result table in any order.
select name from SalesPerson where sales_id not in
(select o.sales_id from orders o inner join company c on
o.com_id = c.com_id where c.name = "RED");
```

Week 16

```
Table: Views

+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| article_id    | int     |
| author_id     | int     |
| viewer_id     | int     |
| view_date     | date    |
+---------------+---------+
The table may have duplicate rows (In other words, there is no primary key for
this table in SQL).
Each row of this table indicates that some viewer viewed an article (written
by some author) on some date.
Note that equal author_id and viewer_id indicate the same person.

-- 1. Find all the authors that viewed at least one of their own articles.
-- Return the result table sorted by id in ascending order.
select distinct author_id as id from views where author_id = viewer_id order
by author_id;

Table: Product

+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
| unit_price   | int     |
+--------------+---------+
product_id is the primary key of this table.
Each row of this table indicates the name and the price of each product.
Table: Sales

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| seller_id   | int     |
| product_id  | int     |
| buyer_id    | int     |
| sale_date   | date    |
| quantity    | int     |
| price       | int     |
+-------------+---------+
This table has no primary key, it can have repeated rows.
product_id is a foreign key to the Product table.
```

```
Each row of this table contains some information about one sale.

-- 2. Write an SQL query that reports the products that were only sold in the
first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.
-- Return the result table in any order.
select p.product_id, p.product_name from product p inner join sales s on
p.product_id=s.product_id
group by s.product_id having min(s.sale_date) >= '2019-01-01' and
max(s.sale_date) <= '2019-03-31';
```

Week 17

```
Table: Products

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| product_id  | int     |
| store1      | int     |
| store2      | int     |
| store3      | int     |
+-------------+---------+
In SQL, product_id is the primary key for this table.
Each row in this table indicates the product's price in 3 different stores:
store1, store2, and store3.
If the product is not available in a store, the price will be null in that
store's column.

-- 1. Rearrange the Products table so that each row has (product_id, store,
price).
-- If a product is not available in a store, do not include a row with that
product_id and store combination in the result table.
SELECT product_id, 'store1' AS store, store1 AS price FROM Products WHERE
store1 IS NOT NULL
UNION
SELECT product_id, 'store2' AS store, store2 AS price FROM Products WHERE
store2 IS NOT NULL
UNION
SELECT product_id, 'store3' AS store, store3 AS price FROM Products WHERE
store3 IS NOT NULL;
```

```

Table: Logins

+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| user_id        | int      |
| time_stamp     | datetime |
+----------------+----------+
(user_id, time_stamp) is the primary key for this table.
Each row contains information about the login time for the user with ID user_id.

-- 2. Write an SQL query to report the latest login for all users in the year 2020. Do not include the users who did not login in 2020.
with cte as(
select user_id, time_stamp as last_stamp from Logins where YEAR(time_stamp) = 2020
), cte1 as(
    select user_id, last_stamp, rank() over(partition by user_id order by last_stamp desc) as r from cte
)
select user_id, last_stamp from cte1 where r = 1;
```

Week 18

```
Table: Users

+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| user_id        | int     |
| join_date      | date    |
| favorite_brand | varchar |
+----------------+---------+
user_id is the primary key of this table.
This table has the info of the users of an online shopping website where users can sell and buy items.


Table: Orders

+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| order_id      | int     |
| order_date    | date    |
| item_id       | int     |
| buyer_id      | int     |
| seller_id     | int     |
+---------------+---------+
order_id is the primary key of this table.
item_id is a foreign key to the Items table.
buyer_id and seller_id are foreign keys to the Users table.


Table: Items

+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| item_id       | int     |
| item_brand    | varchar |
+---------------+---------+
item_id is the primary key of this table.

-- 1. Write an SQL query to find for each user, the join date and the number
of orders they made as a buyer in 2019.
select u.user_id as buyer_id , u.join_date, IFNULL(count(o.order_id), 0) as
orders_in_2019
from users u
left join orders o
on u.user_id=o.buyer_id
and year(order_date)='2019'
group by u.user_id;

Table: Stocks

+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| stock_name    | varchar |
| operation     | enum    |
| operation_day | int     |
| price         | int     |
+---------------+---------+
(stock_name, operation_day) is the primary key for this table.
The operation column is an ENUM of type ('Sell', 'Buy')
Each row of this table indicates that the stock which has stock_name had an
operation on the day operation_day with the price.
It is guaranteed that each 'Sell' operation for a stock has a corresponding
'Buy' operation in a previous day. It is also guaranteed that each 'Buy'
operation for a stock has a corresponding 'Sell' operation in an upcoming day.
```

```
-- 2. Write an SQL query to report the Capital gain/loss for each stock.
-- The Capital gain/loss of a stock is the total gain or loss after buying and selling the stock one or many times.
select stock_name, sum(case
when operation = "Buy" then -price else price end) as capital_gain_loss
from Stocks group by stock_name;
```

Week 19

```
/*
1. Assume you're given tables with information on Snapchat users, including their ages and time spent sending and opening snaps.

Write a query to obtain a breakdown of the time spent sending vs. opening snaps as a percentage of total time spent on these activities grouped by age group. Round the percentage to 2 decimal places in the output.

Notes:

Calculate the following percentages:
time spent sending / (Time spent sending + Time spent opening)
Time spent opening / (Time spent sending + Time spent opening)
To avoid integer division in percentages, multiply by 100.0 and not 100.

activities Table
Column Name Type
activity_id integer
user_id integer
activity_type   string ('send', 'open', 'chat')
time_spent  float
activity_date   datetime

age_breakdown Table
Column Name Type
user_id integer
age_bucket  string ('21-25', '26-30', '31-25')

*/

WITH snaps_statistics AS (
  SELECT
    age.age_bucket,
    SUM(CASE WHEN activities.activity_type = 'send'
      THEN activities.time_spent ELSE 0 END) AS send_timespent,
    SUM(CASE WHEN activities.activity_type = 'open'
      THEN activities.time_spent ELSE 0 END) AS open_timespent,
    SUM(activities.time_spent) AS total_timespent
  FROM activities
```

```
 INNER JOIN age_breakdown AS age
    ON activities.user_id = age.user_id
  WHERE activities.activity_type IN ('send', 'open')
  GROUP BY age.age_bucket)

SELECT
  age_bucket,
  ROUND(100.0 * send_timespent / total_timespent, 2) AS send_perc,
  ROUND(100.0 * open_timespent / total_timespent, 2) AS open_perc
FROM snaps_statistics;

/*
2. Assume you're given two tables containing data about Facebook Pages and
their respective likes (as in "Like a Facebook Page").

Write a query to return the IDs of the Facebook pages that have zero likes.
The output should be sorted in ascending order based on the page IDs.

pages Table:
Column Name Type
page_id integer
page_name   varchar

page_likes Table:
Column Name Type
user_id integer
page_id integer
liked_date  datetime

*/
SELECT page_id from pages where page_id not in
(select distinct page_id from page_likes) order by page_id;
```

Week 20

```
/*
1. Given a table of tweet data over a specified time period, calculate the 3-
day rolling average of tweets for each user. Output the user ID, tweet date,
and rolling averages rounded to 2 decimal places.
Notes:
A rolling average, also known as a moving average or running mean is a time-
series technique that examines trends in data over a specified period of time.
In this case, we want to determine how the tweet count for each user changes
over a 3-day period.

tweets Table:
Column Name Type
```

```sql
user_id integer
tweet_date  timestamp
tweet_count integer

*/
SELECT
  user_id,
  tweet_date,
  ROUND(AVG(tweet_count) OVER (
    PARTITION BY user_id
    ORDER BY tweet_date
    ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
  ,2) AS rolling_avg_3d
FROM tweets;

/*
2. Assume you're given the tables containing completed trade orders and user
details in a Robinhood trading system.
Write a query to retrieve the top three cities that have the highest number of
completed trade orders listed in descending order. Output the city name and
the corresponding number of completed trade orders.

trades Table:
Column Name Type
order_id    integer
user_id integer
price   decimal
quantity    integer
status  string('Completed' ,'Cancelled')
timestamp   datetime

users Table:
Column Name Type
user_id integer
city    string
email   string
signup_date datetime

*/
select u.city, count(t.order_id) as total_orders from
trades t inner join users u on t.user_id = u.user_id
where t.status = 'Completed' group by u.city
order by count(t.order_id) desc limit 3;
```

Week 21

```
/*
Given the reviews table, write a query to retrieve the average star rating for
each product, grouped by month.
The output should display the month as a numerical value, product ID, and
average star rating rounded to two decimal places.
Sort the output first by month and then by product ID.

reviews Table:
Column Name Type
review_id   integer
user_id integer
submit_date datetime
product_id  integer
stars   integer (1-5)

*/
select EXTRACT(MONTH from submit_date) as mth, product_id as product,
ROUND(AVG(stars), 2) as avg_stars from reviews GROUP BY
EXTRACT(MONTH from submit_date), product_id ORDER BY
EXTRACT(MONTH from submit_date), product_id;

/*
Assume you're given a table with measurement values obtained from a Google
sensor over multiple days with measurements taken multiple times within each
day.
Write a query to calculate the sum of odd-numbered and even-numbered
measurements separately for a particular day and display the results in two
different columns. Refer to the Example Output below for the desired format.
Definition:
Within a day, measurements taken at 1st, 3rd, and 5th times are considered
odd-numbered measurements, and measurements taken at 2nd, 4th, and 6th times
are considered even-numbered measurements.

measurements Table:
Column Name Type
measurement_id  integer
measurement_value   decimal
measurement_time    datetime

*/
WITH ranked_measurements AS (
  SELECT
    CAST(measurement_time AS DATE) AS measurement_day,
    measurement_value,
    ROW_NUMBER() OVER (
      PARTITION BY CAST(measurement_time AS DATE)
      ORDER BY measurement_time) AS measurement_num
  FROM measurements
)
SELECT measurement_day,
SUM(measurement_value) FILTER (WHERE measurement_num % 2 != 0) as odd_sum,
SUM(measurement_value) FILTER (WHERE measurement_num % 2 = 0) as even_sum
FROM ranked_measurements group by measurement_day;
```

Week 22

```
/*
CVS Health is trying to better understand its pharmacy sales, and how well
different products are selling. Each drug can only be produced by one
manufacturer.

Write a query to find the top 3 most profitable drugs sold, and how much
profit they made. Assume that there are no ties in the profits. Display the
result from the highest to the lowest total profit.
Definition:
cogs stands for Cost of Goods Sold which is the direct cost associated with
producing the drug.
Total Profit = Total Sales - Cost of Goods Sold

pharmacy_sales Table:
Column Name Type
product_id  integer
units_sold  integer
total_sales decimal
cogs    decimal
manufacturer    varchar
drug    varchar

*/
SELECT drug, SUM(total_sales - cogs) as total_profit
FROM pharmacy_sales group by drug ORDER BY
SUM(total_sales - cogs) desc limit 3;

/*
CVS Health is analyzing its pharmacy sales data, and how well different
products are selling in the market. Each drug is exclusively manufactured by a
single manufacturer.

Write a query to identify the manufacturers associated with the drugs that
resulted in losses for CVS Health and calculate the total amount of losses
incurred.
```

```
Output the manufacturer's name, the number of drugs associated with losses,
and the total losses in absolute value. Display the results sorted in
descending order with the highest losses displayed at the top.
*/
SELECT manufacturer, count(DISTINCT drug),
SUM(cogs - total_sales) as total_loss
FROM pharmacy_sales
where cogs > total_sales
group by manufacturer
ORDER BY SUM(cogs - total_sales) desc;
```

Week 23

```
/*
Assume there are three Spotify tables: artists, songs, and global_song_rank,
which contain information about the artists, songs, and music charts,
respectively.
Write a query to find the top 5 artists whose songs appear most frequently in
the Top 10 of the global_song_rank table. Display the top 5 artist names in
ascending order, along with their song appearance ranking.
Assumptions:
If two or more artists have the same number of song appearances, they should
be assigned the same ranking, and the rank numbers should be continuous (i.e.
1, 2, 2, 3, 4, 5).
For instance, if both Ed Sheeran and Bad Bunny appear in the Top 10 five
times, they should both be ranked 1st and the next artist should be ranked
2nd.

artists Table:
Column Name Type
artist_id   integer
artist_name varchar

songs Table:
Column Name Type
song_id integer
artist_id   integer

global_song_rank Table:
Column Name Type
day integer (1-52)
song_id integer
rank    integer (1-1,000,000)

*/
```

```sql
WITH top_10_cte AS (
  SELECT
    artists.artist_name,
    DENSE_RANK() OVER (
      ORDER BY COUNT(songs.song_id) DESC) AS artist_rank
  FROM artists
  INNER JOIN songs
    ON artists.artist_id = songs.artist_id
  INNER JOIN global_song_rank AS ranking
    ON songs.song_id = ranking.song_id
  WHERE ranking.rank <= 10
  GROUP BY artists.artist_name
)

SELECT artist_name, artist_rank
FROM top_10_cte
WHERE artist_rank <= 5;

/*
New TikTok users sign up with their emails. They confirmed their signup by
replying to the text confirmation to activate their accounts. Users may
receive multiple text messages for account confirmation until they have
confirmed their new account.
A senior analyst is interested to know the activation rate of specified users
in the emails table. Write a query to find the activation rate. Round the
percentage to 2 decimal places.
Definitions:
emails table contain the information of user signup details.
texts table contains the users' activation information.
Assumptions:
The analyst is interested in the activation rate of specific users in the
emails table, which may not include all users that could potentially be found
in the texts table.
For example, user 123 in the emails table may not be in the texts table and
vice versa.

emails Table:
Column Name Type
email_id    integer
user_id integer
signup_date datetime

texts Table:
Column Name Type
text_id integer
email_id    integer
signup_action   varchar
*/
```

```
select ROUND(COUNT(texts.email_id)::DECIMAL / COUNT(distinct emails.email_id),
2) as activation_rate
from emails left join texts on emails.email_id = texts.email_id
and texts.signup_action = 'Confirmed';
```

Week 24

```
/*
Table: Products

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| product_id  | int     |
| store1      | int     |
| store2      | int     |
| store3      | int     |
+-------------+---------+
In SQL, product_id is the primary key for this table.
Each row in this table indicates the product's price in 3 different stores:
store1, store2, and store3.
If the product is not available in a store, the price will be null in that
store's column.

Rearrange the Products table so that each row has (product_id, store, price).
If a product is not available in a store, do not include a row with that
product_id and store combination in the result table.
*/
SELECT product_id, 'store1' AS store, store1 AS price FROM Products WHERE
store1 IS NOT NULL
UNION
SELECT product_id, 'store2' AS store, store2 AS price FROM Products WHERE
store2 IS NOT NULL
UNION
SELECT product_id, 'store3' AS store, store3 AS price FROM Products WHERE
store3 IS NOT NULL;



/*
Table: Sales

+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
(sale_id, year) is the primary key of this table.
product_id is a foreign key to Product table.
Each row of this table shows a sale on the product product_id in a certain year.
Note that the price is per unit.


Table: Product

+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
+--------------+---------+
product_id is the primary key of this table.
Each row of this table indicates the product name of each product.

Write an SQL query that reports the product_name, year, and price for each sale_id in the Sales table.
*/
select p.product_name, s.year, s.price from Sales s inner join Product p on s.product_id = p.product_id;
```

Week 25

```
/*
Table: Employees

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| employee_id | int     |
| name        | varchar |
| salary      | int     |
+-------------+---------+
In SQL, employee_id is the primary key for this table.
Each row of this table indicates the employee ID, employee name, and salary.

Calculate the bonus of each employee. The bonus of an employee is 100% of their salary if the ID of the employee is an odd number
and the employee name does not start with the character 'M'. The bonus of an employee is 0 otherwise.
*/
```

```
select employee_id,
CASE
WHEN (employee_id % 2) = 1 and name not like 'M%' then salary
ELSE 0
END as bonus
from Employees order by employee_id;

/*
Table: Employees

+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| employee_id | int      |
| name        | varchar  |
| manager_id  | int      |
| salary      | int      |
+-------------+----------+
employee_id is the primary key for this table.
This table contains information about the employees, their salary, and the ID
of their manager.
Some employees do not have a manager (manager_id is null).

Write an SQL query to report the IDs of the employees whose salary is strictly
less than $30000 and whose manager left the company.
When a manager leaves the company, their information is deleted from the
Employees table,
but the reports still have their manager_id set to the manager that left.
*/
select employee_id
from Employees
where salary < 30000 and manager_id not in
(select employee_id from Employees)
order by employee_id;
```

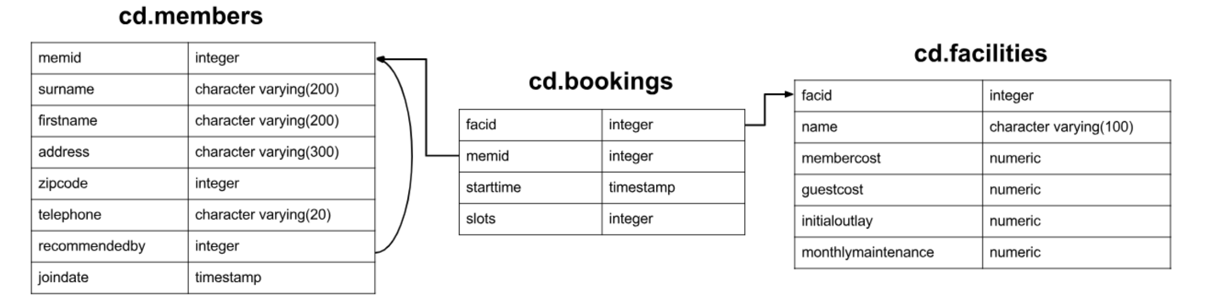
Week 26

```
/*
Table: Products

+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| product_id  | int     |
| low_fats    | enum    |
| recyclable  | enum    |
+-------------+---------+
In SQL, product_id is the primary key for this table.
```

```
low_fats is an ENUM of type ('Y', 'N') where 'Y' means this product is low fat
and 'N' means it is not.
recyclable is an ENUM of types ('Y', 'N') where 'Y' means this product is
recyclable and 'N' means it is not.

Find the ids of products that are both low fat and recyclable.
*/
select distinct product_id from Products where low_fats = 'Y' and recyclable =
'Y';



/*
Table: Stadium

+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| visit_date    | date    |
| people        | int     |
+---------------+---------+
visit_date is the primary key for this table.
Each row of this table contains the visit date and visit id to the stadium
with the number of people during the visit.
No two rows will have the same visit_date, and as the id increases, the dates
increase as well.

Write an SQL query to display the records with three or more rows with
consecutive id's,
and the number of people is greater than or equal to 100 for each.
*/



with cte as
(
SELECT id, visit_date, people,
row_number() over(order by id) as rn
FROM stadium
where people >= 100
), cte2 as
(
SELECT id, visit_date, people, id-rn as diff
from cte
)
select id, visit_date, people
from cte2
```

```
WHERE diff in (select diff from cte2
group by diff
having count(diff) >= 3)
```

Week 27



cd.members

memid	integer
surname	character varying(200)
firstname	character varying(200)
address	character varying(300)
zipcode	integer
telephone	character varying(20)
recommendedby	integer
joindate	timestamp

cd.bookings

facid	integer
memid	integer
starttime	timestamp
slots	integer

cd.facilities

facid	integer
name	character varying(100)
membercost	numeric
guestcost	numeric
initialoutlay	numeric
monthlymaintenance	numeric

```
-- 1. Produce a list of members (including guests), along with the number of hours they've booked in facilities,
-- rounded to the nearest ten hours. Rank them by this rounded figure, producing output of first name, surname, rounded hours, rank.
-- Sort by rank, surname, and first name.
WITH cte AS (
  SELECT
    m.firstname, m.surname, round(sum(b.slots)*0.5, -1) AS "hours" FROM
    cd.members m INNER JOIN cd.bookings b ON m.memid = b.memid
    GROUP BY m.firstname, m.surname
)
SELECT firstname, surname, hours,
rank() over( ORDER BY hours DESC) AS "rank"
FROM cte ORDER BY rank, surname, firstname;

-- 2. Produce a list of the total number of hours booked per facility, remembering that a slot lasts half an hour.
-- The output table should consist of the facility id, name, and hours booked, sorted by facility id.
-- Try formatting the hours to two decimal places.
select b.facid, f.name,
trim(to_char(sum(b.slots)/2.0, '9999999999999999D99')) as "Total Hours"
from cd.bookings b inner join cd.facilities f on b.facid = f.facid group by b.facid, f.name
order by b.facid;
```

Week 28

```
-- 1. Find the upward recommendation chain for member ID 27: that is, the member who recommended them,
-- and the member who recommended that member, and so on. Return member ID, first name, and surname. Order by descending member id.
with recursive recommenders(recommender) as (
    select recommendedby from cd.members where memid = 27
    union all
    select mems.recommendedby
        from recommenders recs
        inner join cd.members mems
            on mems.memid = recs.recommender
)
select recs.recommender, mems.firstname, mems.surname
    from recommenders recs
    inner join cd.members mems
        on recs.recommender = mems.memid
order by memid desc

-- 2. Find the downward recommendation chain for member ID 1: that is, the members they recommended,
-- the members those members recommended, and so on. Return member ID and name, and order by ascending member id.
with recursive recommendeds(memid) as (
    select memid from cd.members where recommendedby = 1
    union all
    select mems.memid
        from recommendeds recs
        inner join cd.members mems
            on mems.recommendedby = recs.memid
)
select recs.memid, mems.firstname, mems.surname
    from recommendeds recs
    inner join cd.members mems
        on recs.memid = mems.memid
order by memid
```