P11. ACME BOOKING ENGINE

#### In simple words, a booking engine is an application on hotel websites and social media pages to capture and process direct online reservations.

Read the following **relational model**:

* CUSTOMERS (id, lastname, firstname, address, zipcode, phonenumber, recommend\_id\*, registerdate)
* BOOKINGS (fac\_id\*, cust\_id\*, start\_datetime, nhours)
* FACILITIES (id, name, cust\_cost, guest\_cost, purchase\_cost, maintenance\_cost)

Meaning of the fields:

## CUSTOMERS:

* id: Customer’s id.
* lastname: Customer’s last name.
* firstname, Customer’s first name.
* address: Customer’s address.
* zipcode: Customer’s zip code.
* phonenumber: Customer’s contact telephone number.
* recommended\_id: Customer who recommended the service (if any).
* registerdate: Date when the customer joined the service.

## BOOKINGS:

* fac\_id: Facility id of the booking.
* cust\_id: Customer who made the booking.
* start\_datetime: Start date/time of the booking.
* nhours: Number of hours that the facility were booked.

## 

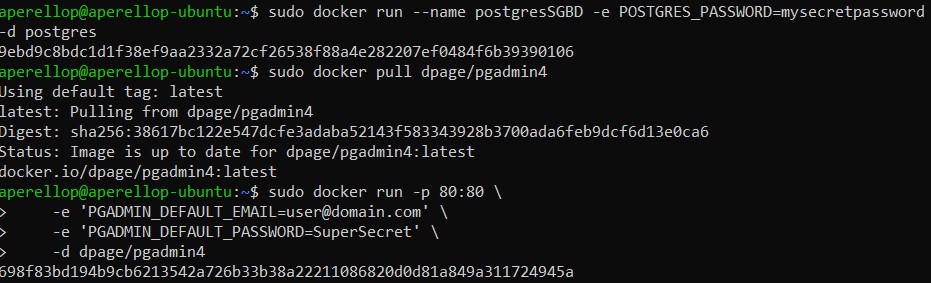
## FACILITIES:

* id: Id of the facility.
* name: Name of the facility.
* cust\_cost: Daily cost for customers. Data type must be money.
* guest\_cost: Daily cost for guests. Data type must be money.
* purchase\_cost: Purchase cost (to the enterprise) of the facility. Data type must be money.
* maintenance\_cost: Monthly maintenance cost of the facility. Data type must be money.

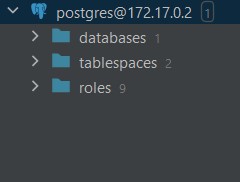
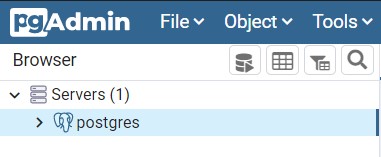
PART A

Describe the postgres **server** dockerization and the connection to a **client**. It can be psql, Pgadmin4, Beekeeper Studio or Datagrip.

First, we create a new Docker and open port 80 to then be able to access it from PgAdmin.



We create a server in PgAdmin and link it to the DataGrip.

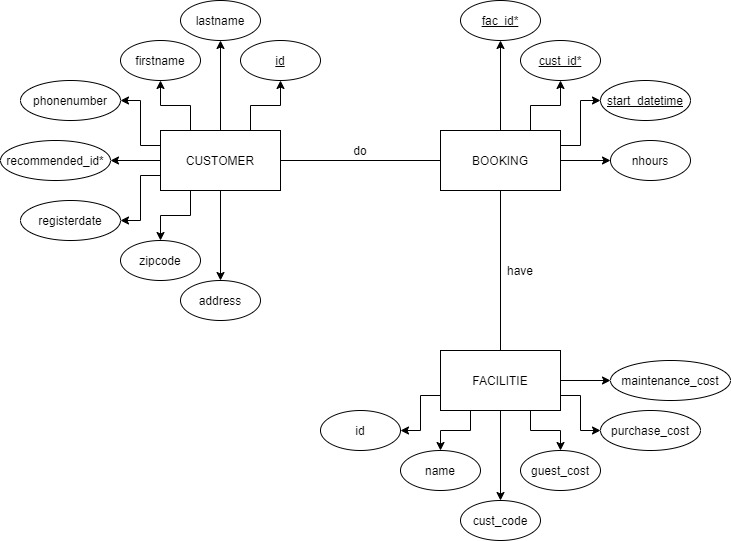


PART B

0.- What’s a booking? What’s a locator in a booking? Do we have a locator in this relational model?

A booking is the reservation code that user have and contains all the information of this. The locator is the fac\_id. We don’t have any locator in this relational model.

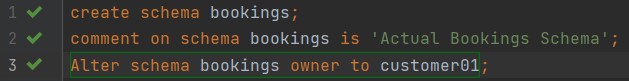
1.- Draw the entity relationship model of the relational model exposed above.



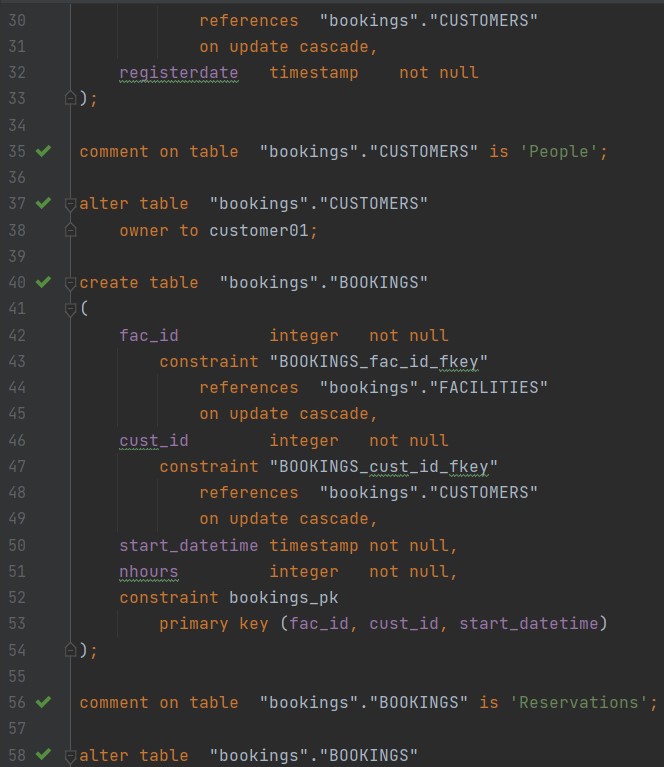
2.- Create a new database (name ‘bookings’) inside your PostgreSQL server for a new user with name ‘customer01’ (‘customer01’ must be the owner of ‘bookings’).



3.- Implement the relational model exposed above using PostgreSQL inside the database ‘bookings’. It’s up to you to choose the appropriate data types, integrity constraint, etc. Before beginning, check the difference between varchar and text data types here: <https://stackoverflow.com/questions/4848964/postgresql-difference-between-text-and-varchar-character-varying>

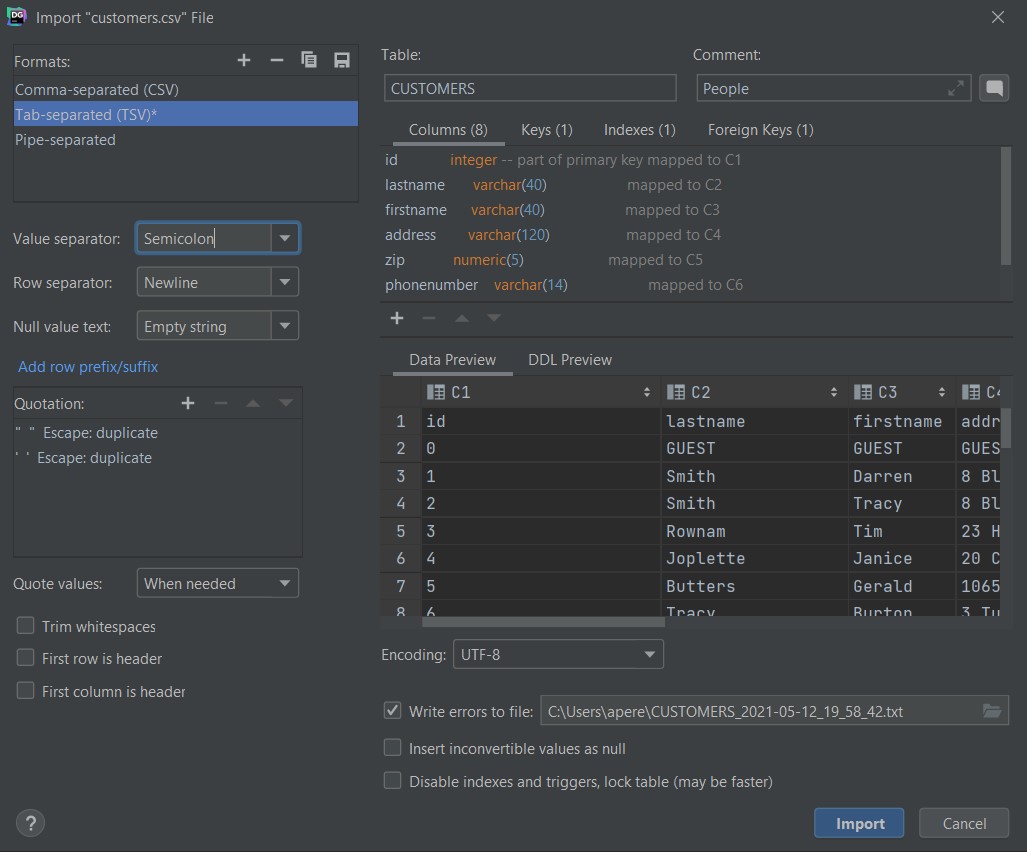




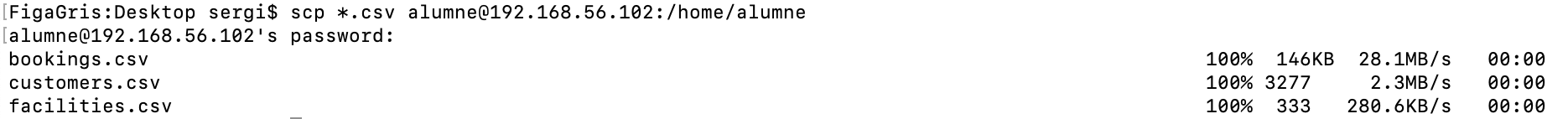


4.- Import the data of the following files inside the tables of the database:

* [bookings.csv](https://drive.google.com/file/d/18ZiRTxPiabvWZeoIkWiQO6_9u4kuqSV2/view?usp=sharing)
* [customers.csv](https://drive.google.com/file/d/15zJOJab3g7dGknUhch9QmHbsW-LPnxHz/view?usp=sharing)
* [facilities.csv](https://drive.google.com/file/d/1w2qBwgd28WvPsPcKQuXWKUlaXWS1yoWy/view?usp=sharing)



Clue: How to copy from Linux to your virtual machine.

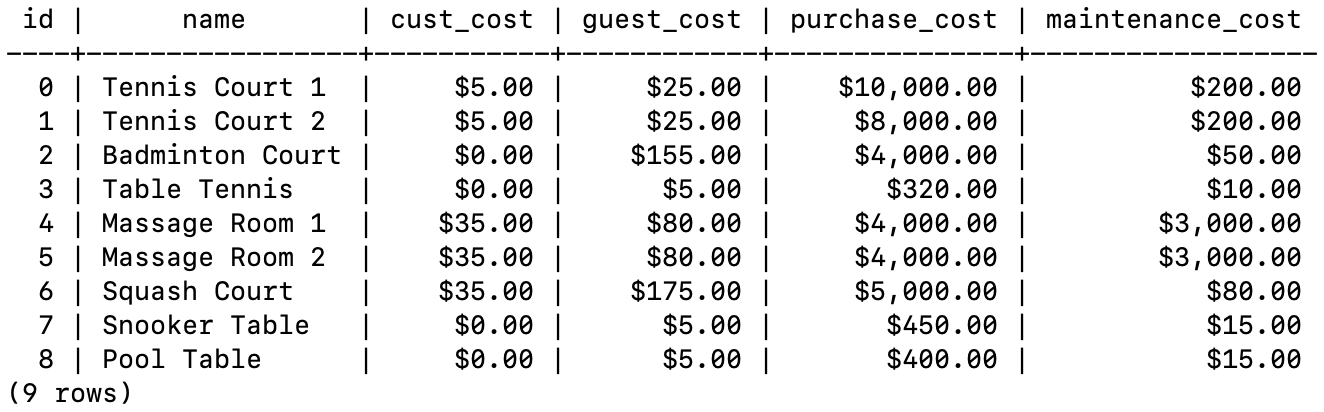


Note: *Before importing search on the Internet how to deal with nulls and quotes.*

5.- Show data of the facilities available.

SELECT \*

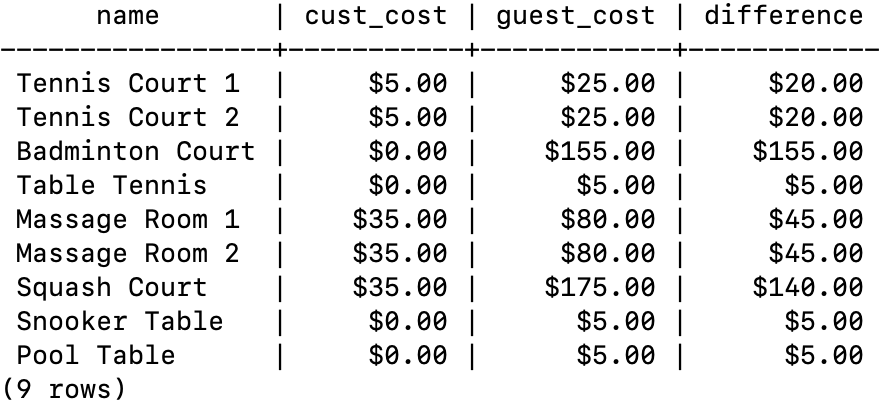
FROM bookings."FACILITIES";



6.- Show names of facilities and cost to customers and to guests. Show the difference between costs.

SELECT name, cust\_cost, guest\_cost, (guest\_cost - cust\_cost)

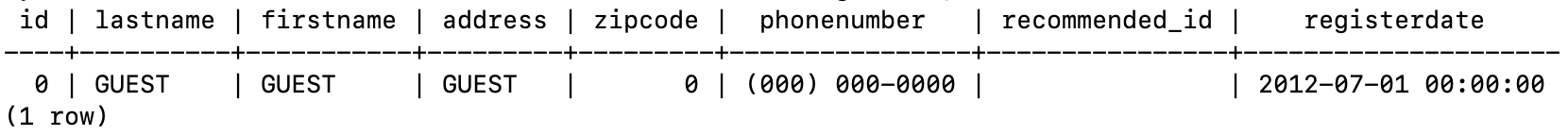
AS difference  
FROM bookings."FACILITIES"



7.- Show if there is a user (=row inside customers) for guests. Search what’s ‘ilike’ and use it.

SELECT \*

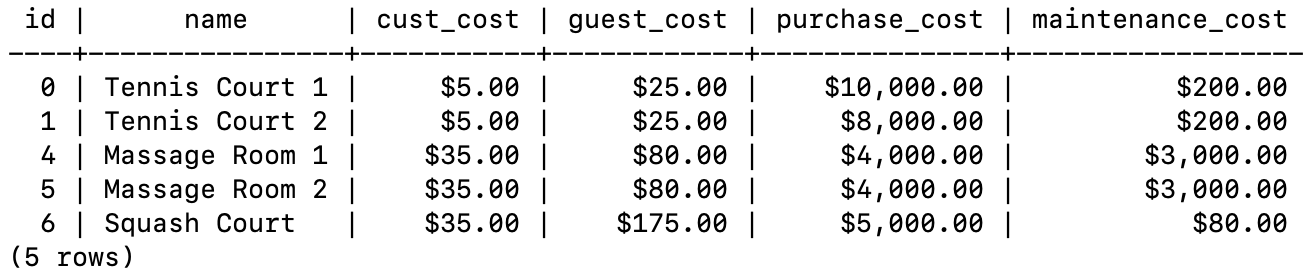
FROM bookings."CUSTOMERS"  
WHERE firstname ILIKE 'guest'  
 AND lastname ILIKE 'guest'  
 AND address ILIKE 'guest'



8.- Show the facilities that have a cost to customers. Before doing the exercise, read this [link](http://www.postgresqltutorial.com/postgresql-cast/).

SELECT \*

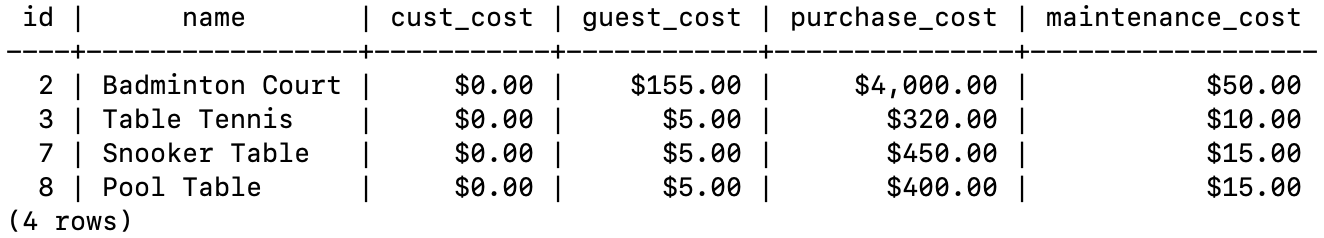
FROM bookings."FACILITIES"  
WHERE CAST (cust\_cost AS numeric) > 0;



9.- Show the free facilities to customers.

SELECT \*

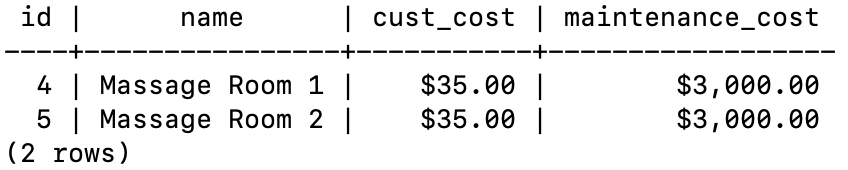
FROM bookings."FACILITIES"  
WHERE CAST (cust\_cost AS numeric) = 0;



10.- Show the facilities that have a cost to customers, and that the cost to customers is less than 1/50 of the monthly maintenance cost. Show the fields id, name, cust\_cost and maintenance\_cost.

SELECT \*

FROM bookings."FACILITIES"  
WHERE CAST (cust\_cost AS numeric) > 0   
 AND CAST (cust\_cost AS numeric) < (CAST (maintenance\_cost AS numeric) / 50 );

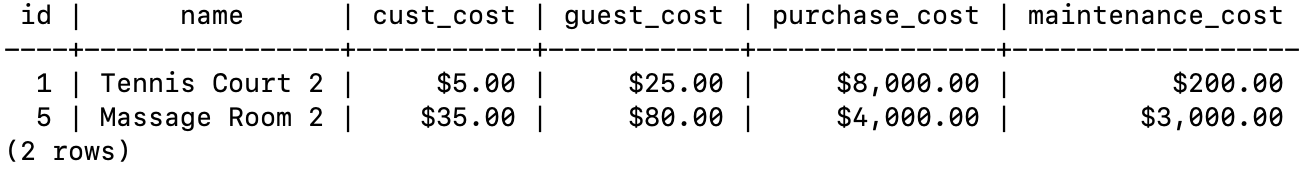


11.- Show facilities with ID 1 and 5. Do it without using the OR operator.

SELECT \*

FROM bookings."FACILITIES"  
WHERE id = 1  
UNION   
SELECT \*  
FROM bookings."FACILITIES"

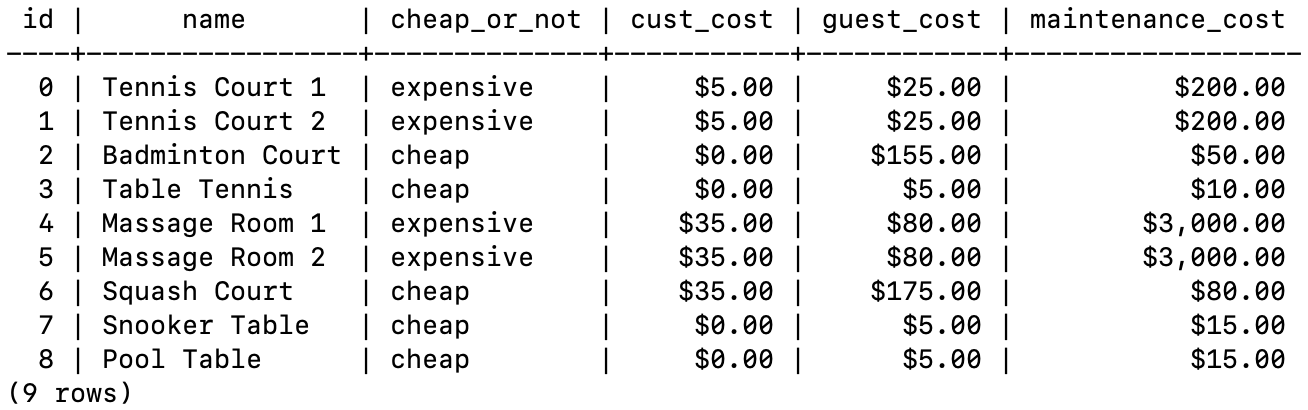
WHERE id = 5;



12.- Show facilities labelling them as 'cheap' or 'expensive' (name of the new column ‘cheap\_or\_not’) depending on if their monthly maintenance cost is more than $150. Show the fields id, name, cheap\_or\_not, cust\_cost, guest\_cost and maintenance\_cost.

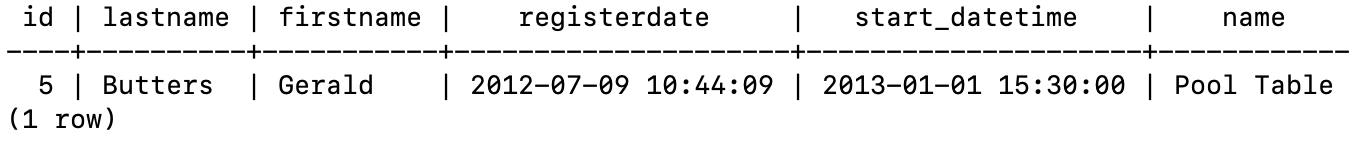
Clue: *Use CASE*.

SELECT name,  
 CASE  
 WHEN cast(maintenance\_cost as numeric) > 150   
 THEN  
 'expensive'  
 ELSE  
 'cheap'  
 END AS cheap\_or\_not, cust\_cost, guest\_cost, maintenance\_cost  
 FROM bookings."FACILITIES";



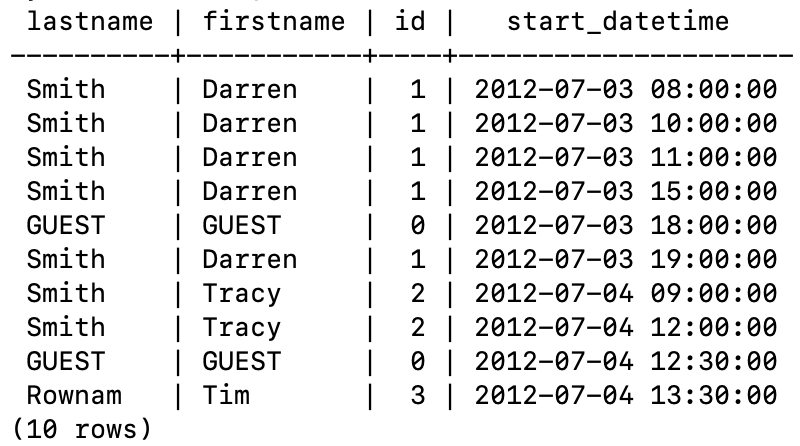
13.- Show customers who booked after the start of October 2012? Return the fields customer’s id, lastname, firstname, registerdate, start\_datetime of the bookings, and facility name.

SELECT c.id,c.lastname,c.firstname,to\_char(c.registerdate, 'YYYY-MM-DD HH:MI:SS'),  
 to\_char(b.start\_datetime, 'YYYY-MM-DD HH:MI:SS'), f.name  
FROM bookings."CUSTOMERS" AS c, bookings."BOOKINGS" AS b, bookings."FACILITIES" AS f  
WHERE *date*(start\_datetime) > '2012-10-01'  
 AND b.cust\_id=c.id  
 AND b.fac\_id=f.id;



14.- Show an ordered list of the first 10 customers who booked our facilities, order by booking start time, lastname, and firstname. It’s mandatory to use the LIMIT clause.

SELECT c.lastname, c.firstname, c.id, b.start\_datetime  
FROM bookings."BOOKINGS" as b, bookings."CUSTOMERS" as c  
WHERE b.cust\_id=c.id  
ORDER BY b.start\_datetime LIMIT 10;



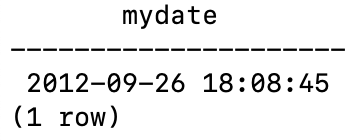
15.- Show a list of customer’s surnames and facility names (together). Order the results for that unique column (clue: union operator).

SELECT DISTINCT lastname AS field  
FROM bookings."CUSTOMERS"  
UNION   
SELECT name AS field  
FROM bookings."FACILITIES"   
ORDER BY field;



16.- Show the last date of the customer’s relation (field customers.registerdate).

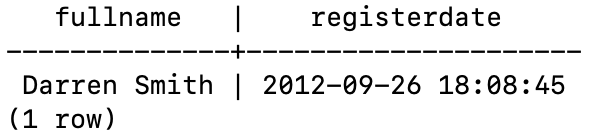
SELECT to\_char(registerdate, 'YYYY-MM-DD HH24:MI:SS') AS mydate  
FROM bookings."CUSTOMERS"  
 ORDER BY registerdate DESC LIMIT 1



17.- Show the full name (and date) of the customers who signed up in the date that you found in exercise 16. Obviously, you can not copy/paste that concrete date…

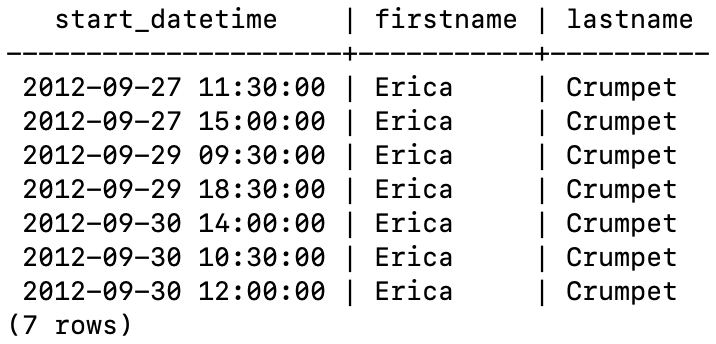
Clue: subquery.

SELECT concat\_ws(' ', c.firstname, c.lastname), to\_char(c.registerdate, 'YYYY-MM-DD HH24:MI:SS')  
FROM bookings."CUSTOMERS" AS c,  
 (SELECT registerdate AS mydate  
 FROM bookings."CUSTOMERS"  
 ORDER BY registerdate DESC LIMIT 1) AS b  
WHERE c.registerdate = b.mydate;



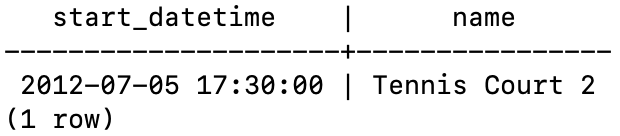
18.- Show start times for bookings by customers named 'E. Crumpet'.

SELECT to\_char(b.start\_datetime, 'YYYY-MM-DD HH24:MI:SS'), c.firstname, c.lastname  
FROM bookings."CUSTOMERS" AS c, bookings."BOOKINGS" AS b  
WHERE c.firstname LIKE 'E%' AND c.lastname LIKE 'Crumpet' AND b.cust\_id=c.id;



19.- Show start times for bookings for tennis courts on '5/7/2012'. Return a list of start time and facility name pairings, ordered by the time. It’s mandatory to use in the WHERE clause the function date\_trunc.

SELECT to\_char(b.start\_datetime, 'YYYY-MM-DD HH24:MI:SS'), f.name  
FROM bookings."BOOKINGS" AS b, bookings."FACILITIES" AS f  
WHERE *date*(b.start\_datetime) = '2012-07-05'  
 AND f.name LIKE 'Tennis%'  
 AND b.fac\_id=f.id;



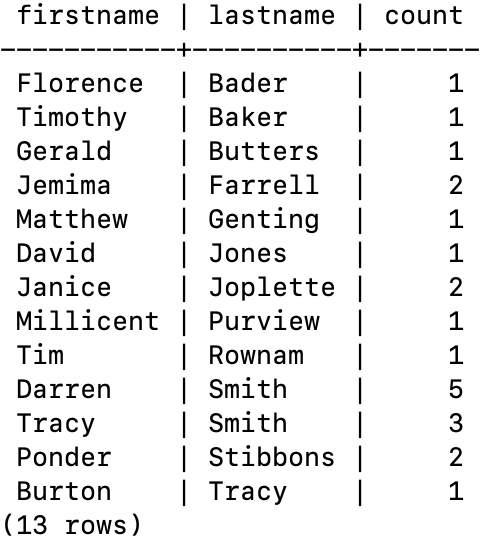
20.- Show all customers who have recommended another customer. Ensure that there are no duplicates in the list, and that results are ordered by lastname and firstname.

SELECT c.firstname, c.lastname  
FROM bookings."CUSTOMERS" AS c  
WHERE c.id IN (SELECT recommended\_id  
 FROM bookings."CUSTOMERS")  
 ORDER BY lastname;



21. Do the same query than before but showing how many customers where recommended by them.

SELECT c.firstname, c.lastname, COUNT(c.firstname)  
FROM bookings."CUSTOMERS" AS c, (SELECT recommended\_id AS rid   
 FROM bookings."CUSTOMERS") AS x  
 WHERE c.id = x.rid  
 GROUP BY c.firstname, c.lastname  
 ORDER BY c.lastname;



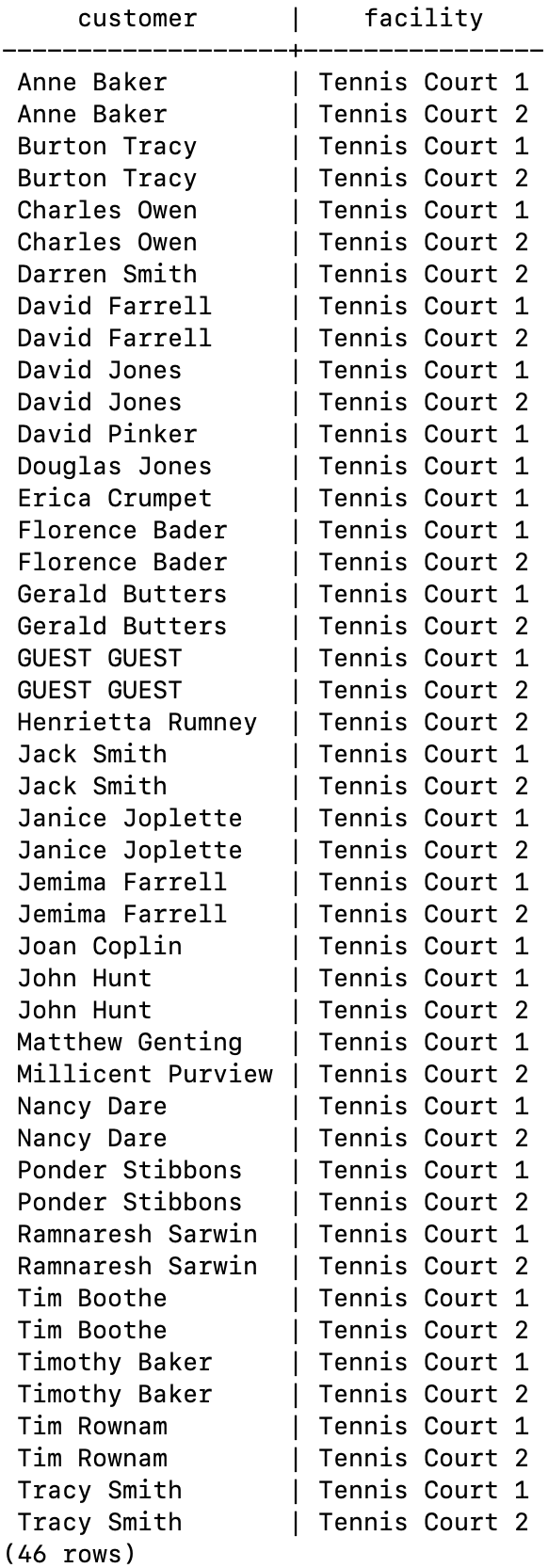
22.- Show customers’ full name, including the individual who recommended them (if any). Ensure that results are ordered by full name.

SELECT c.id, CONCAT\_WS(' ', c.firstname, c.lastname) AS customer, concat\_ws(' ', l.firstname, l.lastname) as recommend\_by  
FROM bookings."CUSTOMERS" AS c,  
 (SELECT DISTINCT c.id AS rid, c.firstname, c.lastname  
 FROM bookings."CUSTOMERS" AS c, (select recommended\_id AS rid from bookings."CUSTOMERS") AS x  
 WHERE c.id = x.rid) AS l  
WHERE c.recommended\_id=l.rid  
UNION   
SELECT c.id, CONCAT\_WS(' ', c.firstname, c.lastname) AS customer, '' AS recommend\_by  
FROM bookings."CUSTOMERS" AS c  
WHERE recommended\_id IS NULL  
ORDER BY customer;



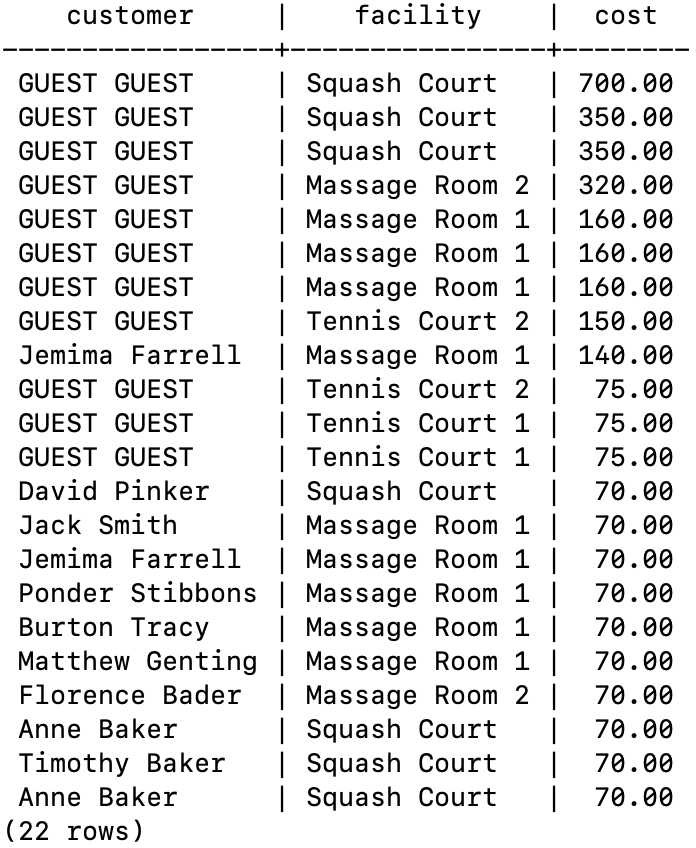
23.- Show full names of the customers who have used a tennis court. Don’t repeat data.

SELECT DISTINCT CONCAT\_WS(' ', c.firstname, c.lastname) AS customer, f.name AS facility  
FROM bookings."CUSTOMERS" AS c, bookings."FACILITIES" AS f, bookings."BOOKINGS" AS b  
WHERE f.name LIKE 'Tennis%'  
 AND f.id=b.fac\_id  
 AND b.cust\_id=c.id;



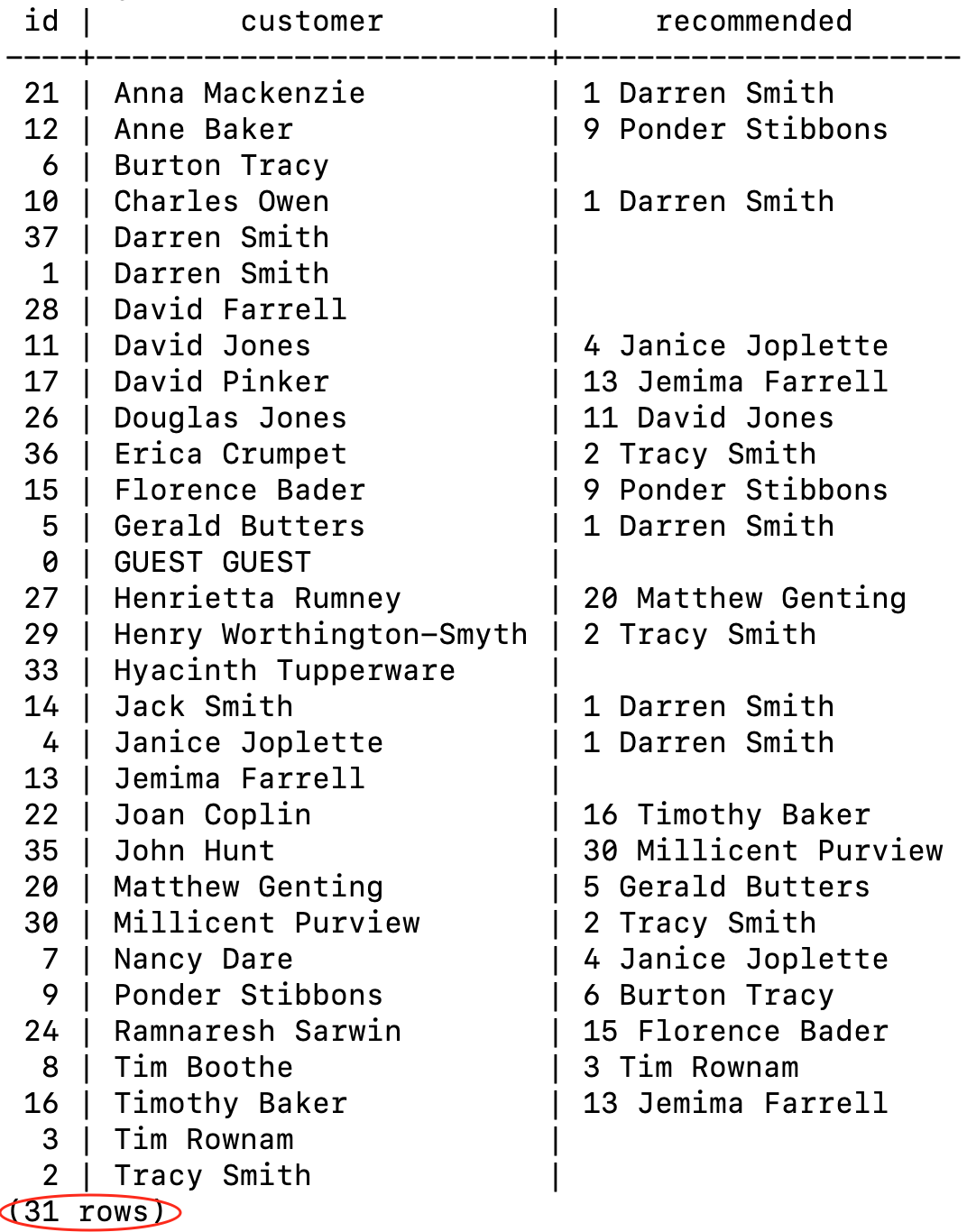
24.- Show bookings on the day of 2012-09-14 which cost to the customer (or guest) more than $30. Remember that guests have different costs than customers (the listed costs are per hour 'slot'), and the guest user always has ID 0. Include in your output the name of the facility, the name of the customer formatted as a single column, and the cost. Order by descending cost, and do not use any subqueries (clue: you must use case).

SELECT CONCAT\_WS(' ', c.firstname, c.lastname) AS customer, f.name AS facility,  
 CASE   
 WHEN c.id=0  
 THEN  
 trunc((cast(f.guest\_cost as numeric)\*b.nhours),2)  
 ELSE  
 trunc((cast(f.cust\_cost as numeric)\*b.nhours),2)  
 END AS cost  
  
FROM bookings."BOOKINGS" AS b, bookings."FACILITIES" AS f, bookings."CUSTOMERS" AS c  
WHERE *date*(b.start\_datetime)='2012-09-14'  
 AND  
 CASE   
 WHEN c.id=0  
 THEN  
 (cast(f.guest\_cost as numeric)\*b.nhours) > 30  
 ELSE  
 (cast(f.cust\_cost as numeric)\*b.nhours) > 30  
 END  
 AND b.fac\_id=f.id  
 AND c.id=b.cust\_id  
ORDER BY cost DESC



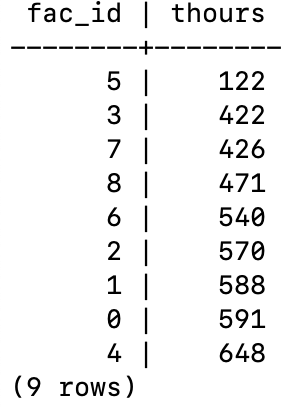
25.- Show a list of all customers, including the individuals who recommended them (if any), using a subquery. Ensure that there are no duplicates in the list, and that each firstname + lastname pairing is formatted as a column and ordered.

SELECT c.id, CONCAT\_WS(' ', c.firstname, c.lastname) AS customer, CONCAT\_WS(' ',l.rid, l.firstname, l.lastname) AS recommend\_by  
FROM bookings."CUSTOMERS" AS c,  
 (SELECT DISTINCT c.id AS rid, c.firstname, c.lastname  
 FROM bookings."CUSTOMERS" AS c, (SELECT recommended\_id AS rid  
 FROM bookings."CUSTOMERS") AS x  
 WHERE c.id = x.rid) AS l  
WHERE c.recommended\_id=l.rid  
UNION   
SELECT c.id, CONCAT\_WS(' ', c.firstname, c.lastname) AS customer, '' AS recommend\_by   
FROM bookings."CUSTOMERS" AS c  
WHERE recommended\_id IS NULL  
ORDER BY customer;



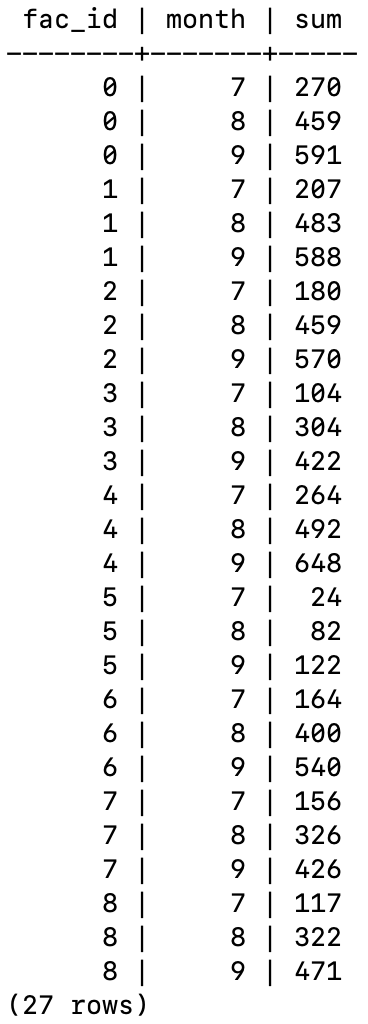
26.- Produce a list of the total number of hours booked per facility in the month of September 2012.

SELECT fac\_id, sum(nhours) AS thours   
FROM bookings."BOOKINGS"  
WHERE *date*(start\_datetime) > '2012-08-31'  
 AND *date*(start\_datetime) < '2012-10-1'  
GROUP BY fac\_id  
ORDER BY thours;



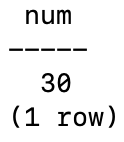
27.- Produce a list of the total number of hours booked per facility per month in the year of 2012. Order by facility id and month.

SELECT fac\_id, extract(MONTH FROM start\_datetime) AS month, sum(nhours) AS thours  
FROM bookings."BOOKINGS"  
WHERE extract(YEAR FROM start\_datetime)='2012'  
GROUP BY month, fac\_id  
ORDER BY fac\_id, month;



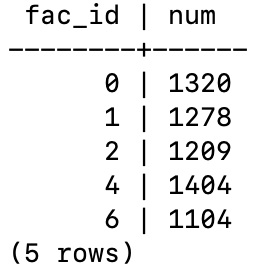
28.- Find the total number of customers who have made at least one booking.

SELECT count(\*) AS num FROM (SELECT DISTINCT cust\_id  
 FROM bookings."BOOKINGS") AS a;



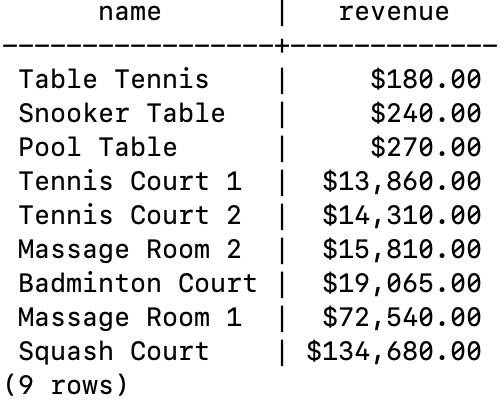
29.- SHOWf facilities with more than 1000 HOURS booked. Sort the results using facility id.

SELECT \*  
FROM (SELECT fac\_id, sum(nhours) as num  
 FROM bookings."BOOKINGS"   
 GROUP BY fac\_id) AS a  
WHERE a.num > 1000  
ORDER BY a.fac\_id;



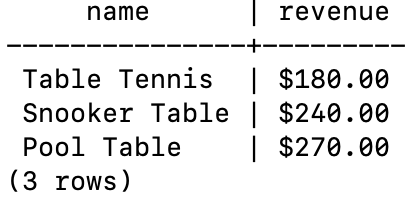
30.- Show facilities along with their total revenue. Remember that there's a different cost for guests and customers! Sort results by revenue.

SELECT a.name, cast(sum(cost) as money) AS revenue   
FROM (SELECT name, cust\_id,  
 CASE   
 WHEN cust\_id=0  
 THEN  
 (cast(f.guest\_cost as numeric)\*b.nhours)  
 ELSE  
 (cast(f.cust\_cost as numeric)\*b.nhours)  
 END AS cost  
 FROM bookings."BOOKINGS" AS b, bookings."FACILITIES" AS f  
 WHERE b.fac\_id=f.id) AS a  
GROUP BY name  
ORDER BY revenue;



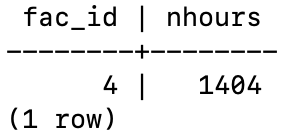
31.- Produce a list of facilities with a total revenue less than 1000. Remember that there's a different cost for guests and customers! Sort results by revenue.

SELECT \*  
FROM (SELECT a.name, cast(sum(cost) as money) AS revenue   
 FROM (SELECT name, cust\_id,  
 CASE  
 WHEN cust\_id=0 THEN  
 (cast(f.guest\_cost as numeric)\*b.nhours)  
 ELSE  
 (cast(f.cust\_cost as numeric)\*b.nhours)  
 END AS cost  
 FROM bookings."BOOKINGS" AS b, bookings."FACILITIES" AS f   
 WHERW b.fac\_id=f.id) AS a  
GROUP BY name  
ORDER BY revenue) AS x  
WHERE cast(x.revenue as numeric) < 1000;



32.- Output the facility id that has the highest number of hours booked.

SELECT fac\_id, sum(nhours) AS nhours  
FROM bookings."BOOKINGS"  
GROUP BY fac\_id  
ORDER BY nhours DESC LIMIT 1;



33.- Check the following query:

SELECT fac\_id, date\_part('month', start\_datetime), SUM(nhours)

FROM bookings

WHERE date\_part('year', start\_datetime) = 2012

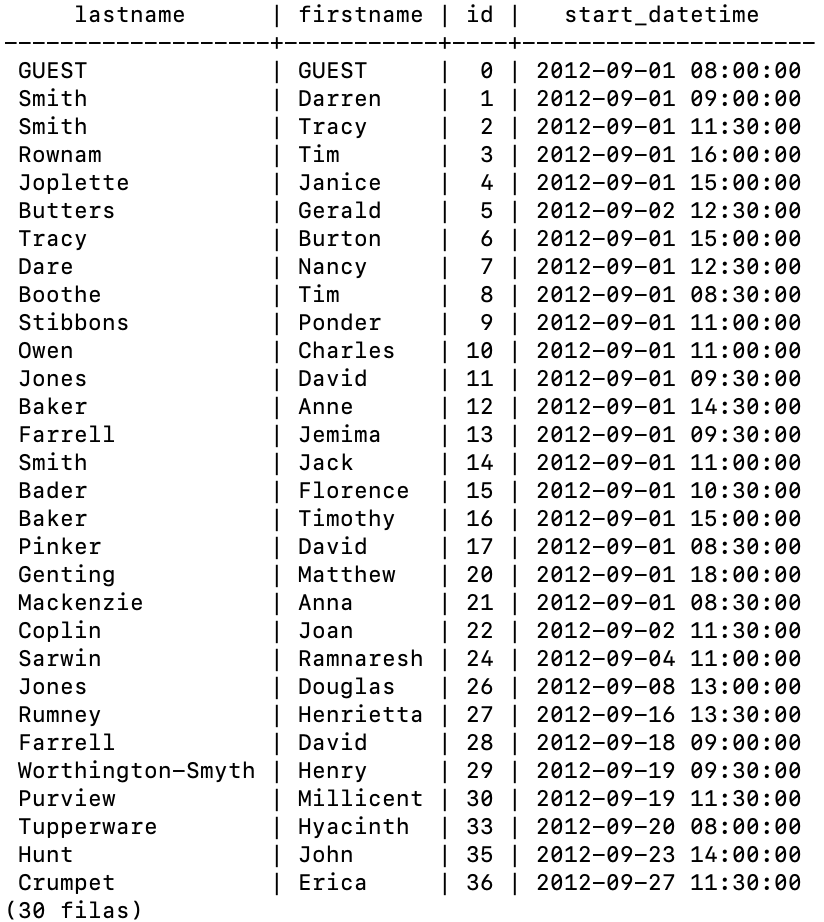
GROUP BY ROLLUP(fac\_id, date\_part('month', start\_datetime))

ORDER BY fac\_id, date\_part('month', start\_datetime);

Explain the clause GROUP BY ROLLUP and the meaning of the query. (You must look for information on the Internet).

34.- Show a list of each customer name, id, and their first booking since September 1st 2012. Order by customer id.

SELECT c.lastname, c.firstname, b.cust\_id AS id, b.start\_datetime  
FROM (SELECT a.cust\_id, min(a.start\_datetime) AS start\_datetime  
 FROM(SELECT \*  
 FROM bookings."BOOKINGS"  
 WHERE *date*(start\_datetime) >= '2012-09-01'  
 ORDER BY cust\_id, start\_datetime) AS a  
 GROUP BY a.cust\_id) AS b,  
 bookings."CUSTOMERS" AS c  
WHERE b.cust\_id=c.id;



35.- (Window function) Show a list of customer names, with each row containing the total customer count. Order by register date.

SELECT count(\*) OVER (), firstname, lastname   
FROM bookings."CUSTOMERS"



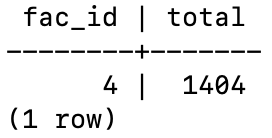
36.- (Window function) Show an increasing numbered list of customers, ordered by their date of registering. Remember that customers’ ids are not guaranteed to be sequential.

SELECT sum(0+1) OVER (ORDER BY id) AS id, firstname, lastname  
FROM bookings."CUSTOMERS";



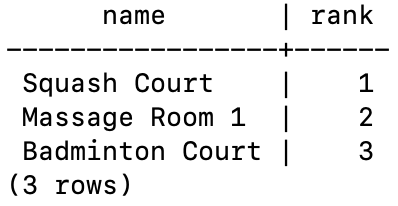
37.- (Windows function) DANGER: Show the facility id that has the highest number of hours booked. All tieing results must get an output.

SELECT DISTINCT fac\_id, sum(nhours) OVER (partition by fac\_id) AS total  
FROM bookings."BOOKINGS"  
ORDER BY total DESC  
LIMIT 1;



38.- (Window function) DANGER: Show a list of the top three revenue generating facilities (including ties (=”empates”)). Sort the results by rank and facility name.

SELECT x.name, sum(0+1) OVER (ORDER BY x.revenue DESC)   
FROM (SELECT a.name, CAST(sum(cost) AS money) AS revenue  
 FROM (SELECT name, cust\_id,  
 CASE   
 WHEN cust\_id=0   
 THEN  
 (cast(f.guest\_cost as numeric)\*b.nhours)  
 ELSE  
 (cast(f.cust\_cost as numeric)\*b.nhours)  
 END AS cost  
 FROM bookings."BOOKINGS" AS b, bookings."FACILITIES" AS f   
 WHERE b.fac\_id=f.id) AS a  
GROUP BY name  
ORDER BY revenue DESC) AS x LIMIT 3;



39.- We need to add it into the facilities table new records with the following values:

* id: 9, name: 'Spa', cust\_cost: 20, guest\_cost: 30, purchase\_cost: 100000, maintenance\_cost: 800
* id: 10, name: 'Spa 2', cust\_cost: 20, guest\_cost: 30, purchase\_cost: 100000, maintenance\_cost: 800.
* id: 11, name: 'Squash Court 2', cust\_cost: 3.5, guest\_cost: 17.5, purchase\_cost: 5000, maintenance\_cost: 80.

INSERT INTO bookings."FACILITIES" (id, name, cust\_cost, guest\_cost, purchase\_cost, maintenance\_cost) VALUES (9, 'Spa', '20', '30', '100000', '8000');  
INSERT INTO bookings."FACILITIES" (id, name, cust\_cost, guest\_cost, purchase\_cost, maintenance\_cost) VALUES (10, 'Spa 2', '20', '30', '100000', '8000');  
INSERT INTO bookings."FACILITIES" (id, name, cust\_cost, guest\_cost, purchase\_cost, maintenance\_cost) VALUES (11, 'Squash Court 2', '3.5', '17.5', '5000', '80');

40.- Let's add, again, a new ‘Spa 3’ to the facilities table. But this time, though, we want to automatically generate the value for the next id, rather than specifying it as a constant (clue: that’s an insert with a subquery). Use the following values for everything else:

INSERT INTO bookings."FACILITIES" (id, name, cust\_cost, guest\_cost, purchase\_cost, maintenance\_cost)  
VALUES ((select id+1 from bookings."FACILITIES" order by id desc limit 1), 'Squash Court 2', '3.5', '17.5', '5000', '80');

* name: 'Spa 3', cust\_cost: 20, guest\_cost: 30, purchase\_cost: 100000, maintenance\_cost: 800.

41.- We made a mistake when entering the data for the second tennis court. The initial purchase cost was 10000 rather than 8000: you need to alter the data to fix the error.

UPDATE bookings."FACILITIES"  
SET purchase\_cost = '10000'  
WHERE id = 1;

42.- We want to increase the price of the tennis courts for both customers and guests. Update the costs to be 6 for customers, and 30 for guests. Use only a single sentence.

UPDATE bookings."FACILITIES"  
SET cust\_cost = '$6.00',  
 guest\_cost = '$30.00'  
WHERE id = 1 or id = 0;

43.- We want to alter the price of the second tennis court so that it costs 10% more than the first one. Try to do this without using constant values for the prices, so that we can reuse the statement if we want to. Use only a single sentence.

UPDATE bookings."FACILITIES"  
SET cust\_cost = (SELECT cust\_cost\*1.1   
 FROM bookings."FACILITIES"   
 WHERE name='Tennis Court 2'),  
 guest\_cost = (SELECT guest\_cost\*1.1   
 FROM bookings."FACILITIES"   
 WHERE name='Tennis Court 2')  
WHERE id = 1;  
  
SELECT \*   
FROM bookings."FACILITIES";

44.- Using pgdump, dump all the database (with inserts) into a text files.

45.- Delete all bookings.

drop table bookings."BOOKINGS";

46.- We want to remove customer 37, who has never made a booking, from our database.

DELETE  
FROM bookings."CUSTOMERS"  
WHERE id = 37;  
  
SELECT \*   
FROM bookings."CUSTOMERS"   
WHERE id=37;

47.- How can we make that more general, to delete all customers who have never made a booking? Clue: Delete with subquery. To test the sentence inter customer 37 again.

DELETE  
FROM bookings."CUSTOMERS"  
WHERE id NOT IN (SELECT DISTINCT cust\_id   
 FROM bookings."BOOKINGS");