Alexander Powers

Prof. Guadalupe Canahuate

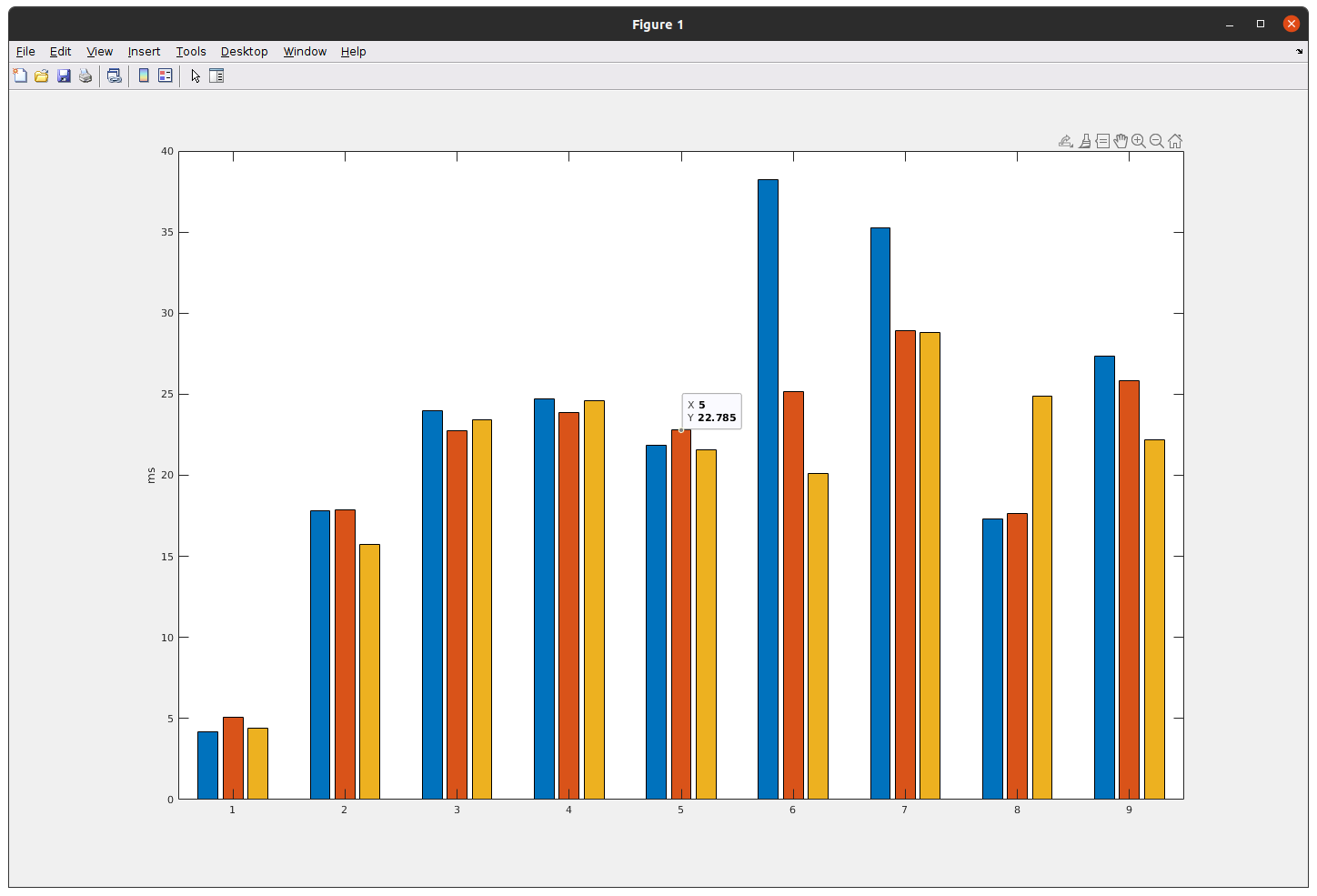
ECE:5995 Modern Databases

September 24, 2020

Homework One Report

This homework centered around the creation of eight queries of a relatively small dataset consisting of three tables, users, user\_stats, and reviews. The development of these queries used a range of PostgreSQL functionality including functions, and fairly complex SELECT statements to generate the queries of interest.

I made the decision to add my index to the userid attribute of the reviews table. My reasoning for this was to improve the performance of the join operation in queries six and seven. The bar chart below details the performance change of the queries with each type of index. Within each query, there are three bars for each of the three runs (left to right: no index, hash index, clustered b-tree index).



I saw the most drastic changes in queries six and seven. This was expected, as both the hash and clustered b-tree improve the performance of the INNER JOIN operation that is the cornerstone of those queries.

Appendix

*-- create user\_stats table*

CREATE TABLE IF NOT EXISTS homework.user\_stats (

userid character(20) NOT NULL,

ageGroup character(6), *-- groupX*

averageRating decimal,

reviewCount integer,

lastPosted date,

helpfulNum integer,

helpfulDenum integer,

helpfulnessRate decimal

);

*-- create rule to update the user\_stats table*

CREATE

OR REPLACE FUNCTION update\_user\_stats(

input\_userid character(20)

) RETURNS boolean AS $did\_insert$

DECLARE

did\_insert boolean := false;

the\_user\_id text;

the\_user\_age integer;

local\_ageGroup character(6); *-- groupX*

local\_averageRating decimal;

local\_reviewCount integer;

local\_lastPosted date;

local\_helpfulNum integer;

local\_helpfulDenum integer;

local\_helpfulnessRate decimal;

BEGIN

*-- determine if the user is already in the user\_stats table*

SELECT userid INTO the\_user\_id

FROM user\_stats u

WHERE u.userid = input\_userid

LIMIT 1;

*-- compute all of the attributes*

*-- compute ageGroup*

SELECT date\_part('year', age(dob)) INTO the\_user\_age

FROM users u

WHERE u.userid = input\_userid

LIMIT 1;

IF the\_user\_age < 30 THEN

local\_ageGroup := 'group1';

ELSEIF 30 <= the\_user\_age AND the\_user\_age <= 50 THEN

local\_ageGroup := 'group2';

ELSEIF the\_user\_age > 50 THEN

local\_ageGroup := 'group3';

ELSE

local\_ageGroup := NULL;

END IF;

*-- compute averageRating*

SELECT AVG(score) INTO local\_averageRating

FROM reviews r

WHERE r.userid = input\_userid;

*-- compute reviewCount*

SELECT COUNT(\*) INTO local\_reviewCount

FROM reviews r

WHERE r.userid = input\_userid;

*-- compute lastPosted*

SELECT dposted INTO local\_lastPosted

FROM reviews r

WHERE r.userid = input\_userid

ORDER BY dposted DESC

LIMIT 1;

*-- compute helpfulNumiewCount*

SELECT SUM(helpfulnessnumerator) INTO local\_helpfulNum

FROM reviews r

WHERE r.userid = input\_userid;

*-- compute helpfulDenum*

SELECT SUM(helpfulnessdenominator) INTO local\_helpfulDenum

FROM reviews r

WHERE r.userid = input\_userid;

if local\_helpfulDenum = 0 THEN

local\_helpfulnessRate := 0;

ELSE

local\_helpfulnessRate := local\_helpfulNum::NUMERIC / local\_helpfulDenum::NUMERIC;

END IF;

IF the\_user\_id IS NULL THEN *-- create new user\_stats*

INSERT INTO

user\_stats (userid, ageGroup, averageRating, reviewCount, lastPosted, helpfulNum, helpfulDenum, helpfulnessRate)

VALUES

(input\_userid, local\_ageGroup, local\_averageRating, local\_reviewCount, local\_lastPosted, local\_helpfulNum, local\_helpfulDenum, local\_helpfulnessRate) RETURNING userid INTO the\_user\_id;

did\_insert := true;

ELSE *-- update existing user\_stats*

UPDATE

user\_stats

SET ageGroup = local\_ageGroup, averageRating = local\_averageRating, reviewCount = local\_reviewCount, lastPosted = local\_lastPosted, helpfulNum = local\_helpfulNum, helpfulDenum = local\_helpfulDenum, helpfulnessRate = local\_helpfulnessRate

WHERE

userid = input\_userid;

END IF;

RETURN did\_insert;

END;

$did\_insert$ LANGUAGE plpgsql;

*-- populate the table*

*SELECT update\_user\_stats(userid) from users;*

Q1

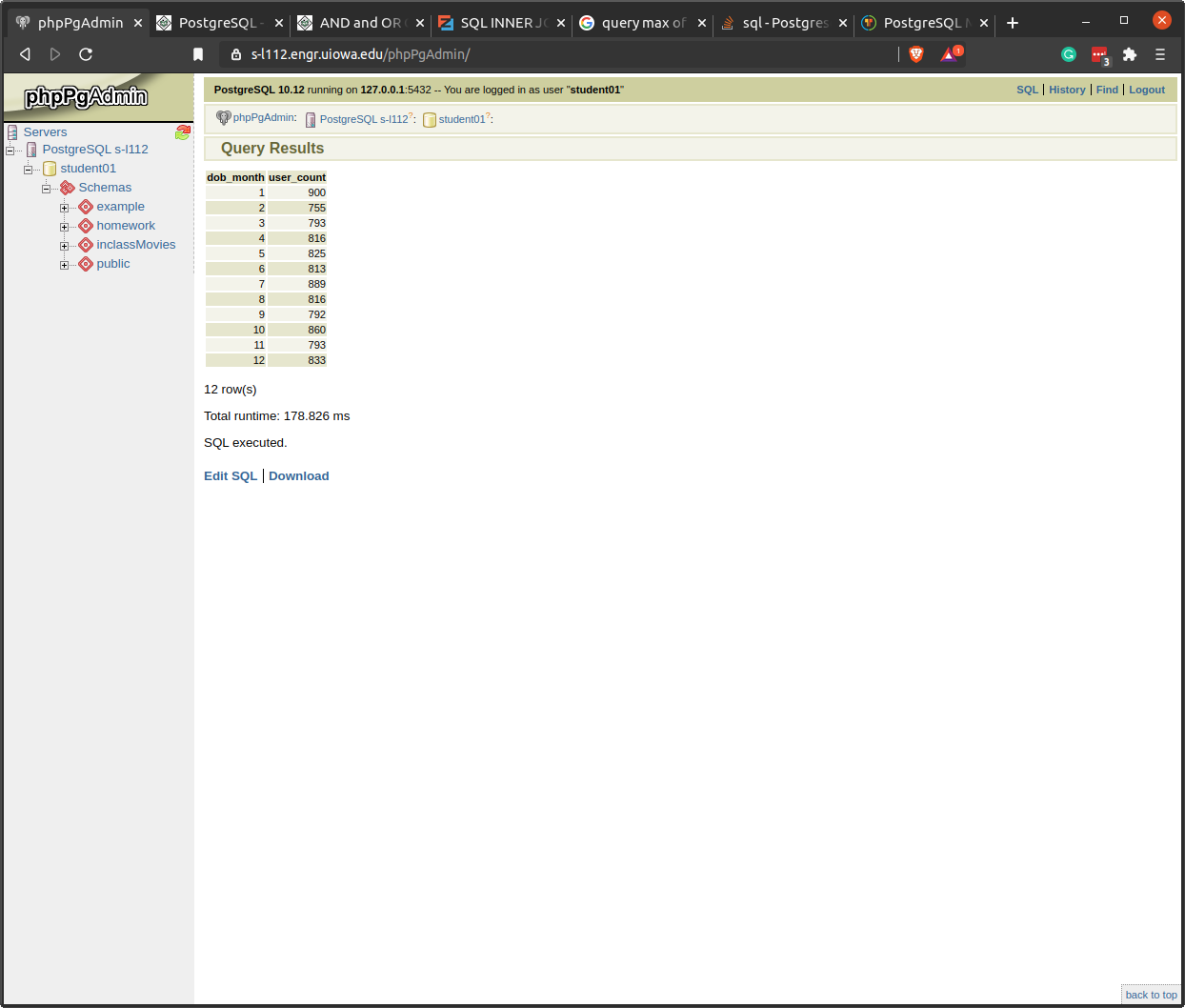
*-- The number of users' birthdays per month*

SELECT date\_part('month', dob) AS dob\_month, COUNT(userid) AS user\_count

FROM users

GROUP BY dob\_month

ORDER BY dob\_month ASC;



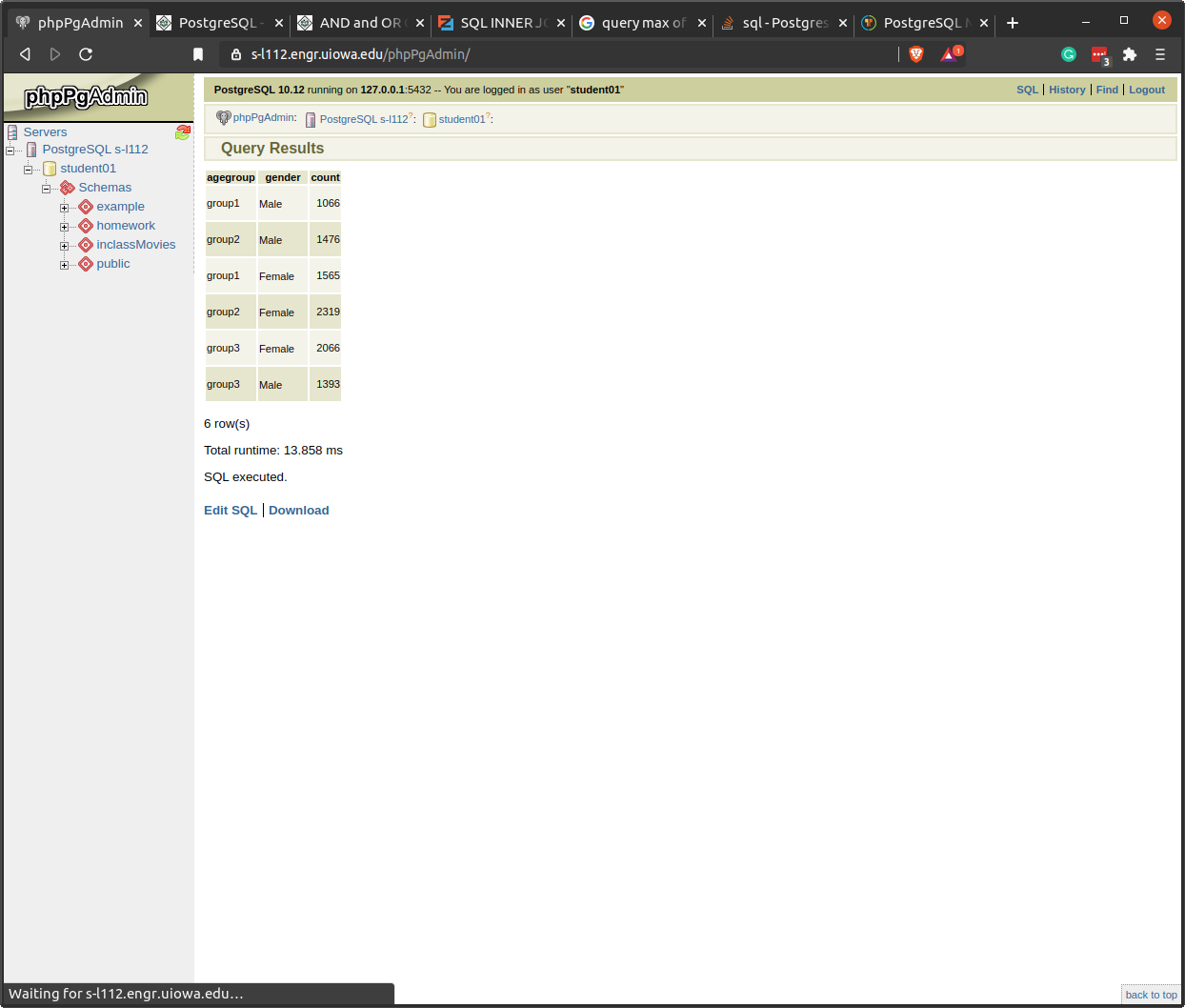
Q2

*-- The number of males/females users per age group.*

SELECT ageGroup, gender, count(\*)

FROM users INNER JOIN user\_stats ON (users.userid = user\_stats.userid)

GROUP BY ageGroup, gender;



Q3

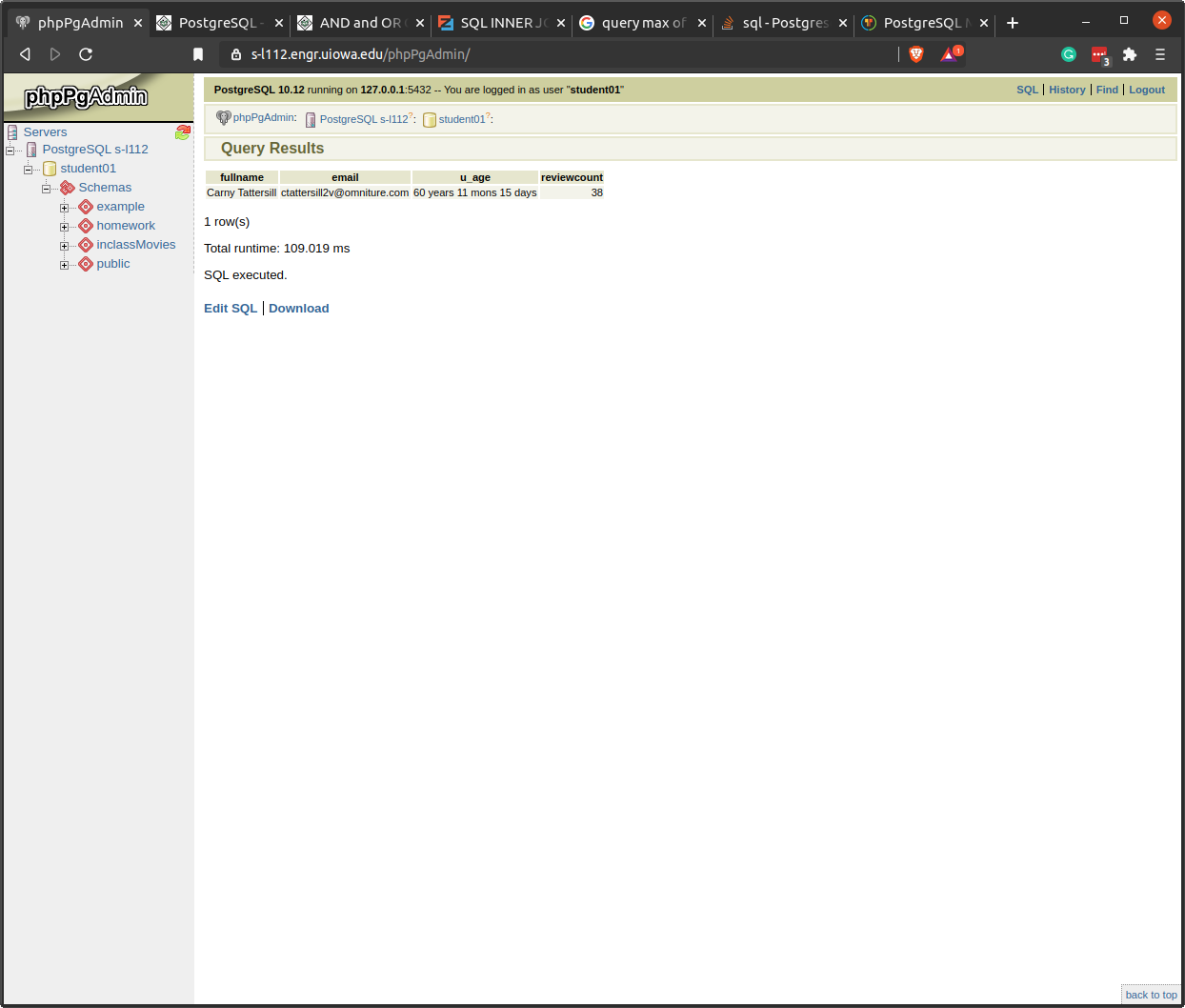
*-- Who had posted the most reviews? (Full name, email, gender, age, reviewcount)*

SELECT CONCAT(firstname, ' ', lastname) AS fullname, email, AGE(dob) as u\_age, reviewCount

FROM users INNER JOIN user\_stats ON (users.userid = user\_stats.userid)

ORDER BY reviewCount DESC

LIMIT 1;



Q4

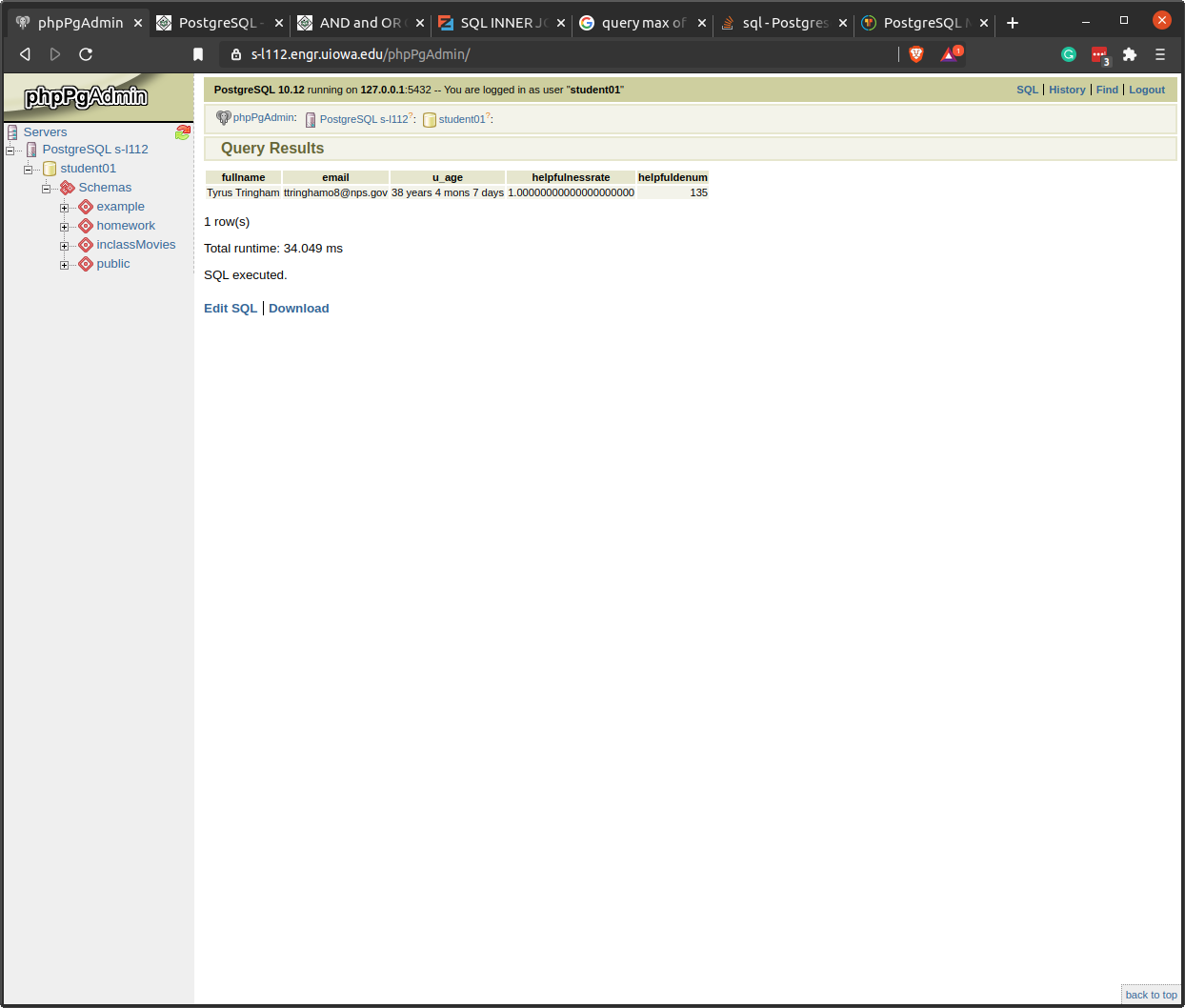
*--Who writes the most helpful reviews? (Full name, email, gender, age, helfulnessrate, and helpfuldenum) - break the ties using the helpfulDenum.*

SELECT CONCAT(firstname, ' ', lastname) AS fullname, email, AGE(dob) as u\_age, helpfulnessRate, helpfulDenum

FROM users INNER JOIN user\_stats ON (users.userid = user\_stats.userid)

ORDER BY helpfulnessrate DESC NULLS LAST, helpfulDenum DESC

LIMIT 1;



Q5

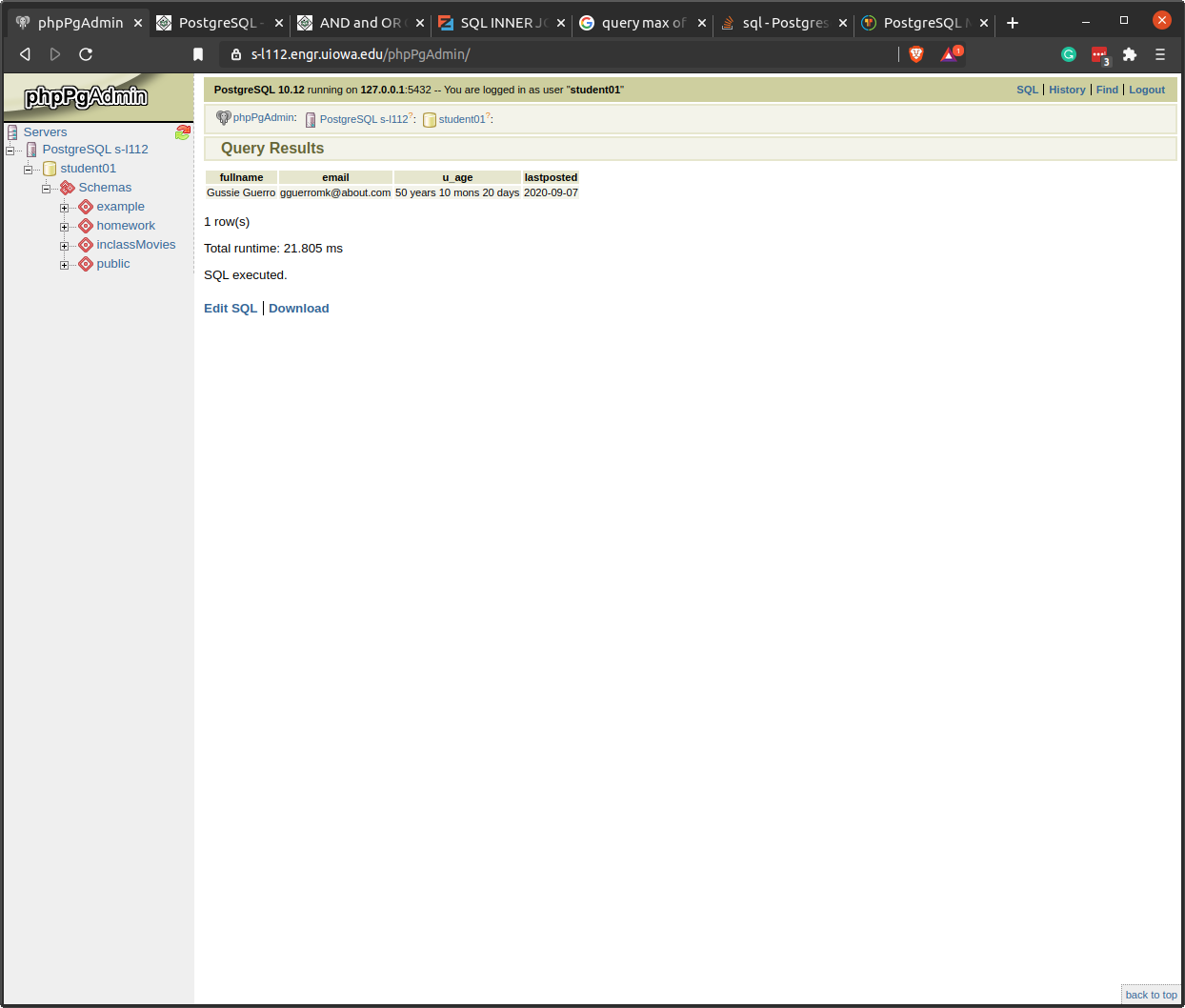
*-- Who posted last? (Full name, email, gender, age, and lastposted)*

SELECT CONCAT(firstname, ' ', lastname) AS fullname, email, AGE(dob) as u\_age, lastPosted

FROM users INNER JOIN user\_stats ON (users.userid = user\_stats.userid)

ORDER BY lastPosted DESC NULLS LAST

LIMIT 1;



Q6

*-- Find the review count and averageRating per ageGroup for each product.*

*-- Show productid, ageGroup, reviewCount, and averageRating.*

SELECT productid, ageGroup, SUM(reviewCount) AS review\_count, AVG(score) AS avg\_rating

FROM reviews INNER JOIN user\_stats ON (reviews.userid = user\_stats.userid)

WHERE

productid = 'B004JRKEH4' OR

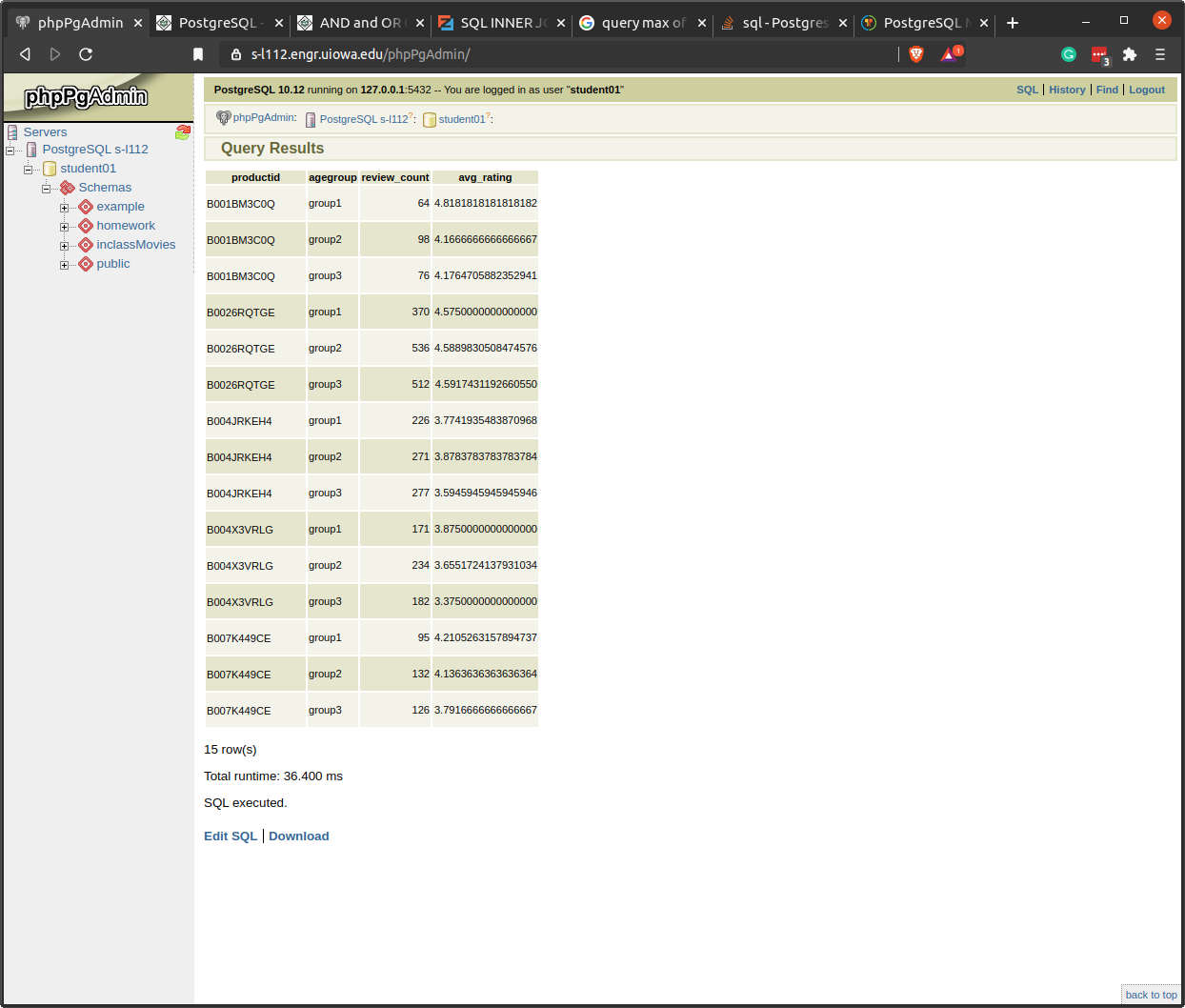
productid = 'B004X3VRLG' OR

productid = 'B0026RQTGE' OR

productid = 'B001BM3C0Q' OR

productid = 'B007K449CE'

GROUP BY productid, ageGroup;



Q7

*-- Find the review count and averageRating per gender for each product.*

*-- Show productid, gender, count, and average rating.*

SELECT productid, gender, SUM(reviewCount) AS review\_count, AVG(score) AS avg\_rating

FROM reviews

INNER JOIN user\_stats ON (reviews.userid = user\_stats.userid)

INNER JOIN users ON (reviews.userid = users.userid)

WHERE

productid = 'B004JRKEH4' OR

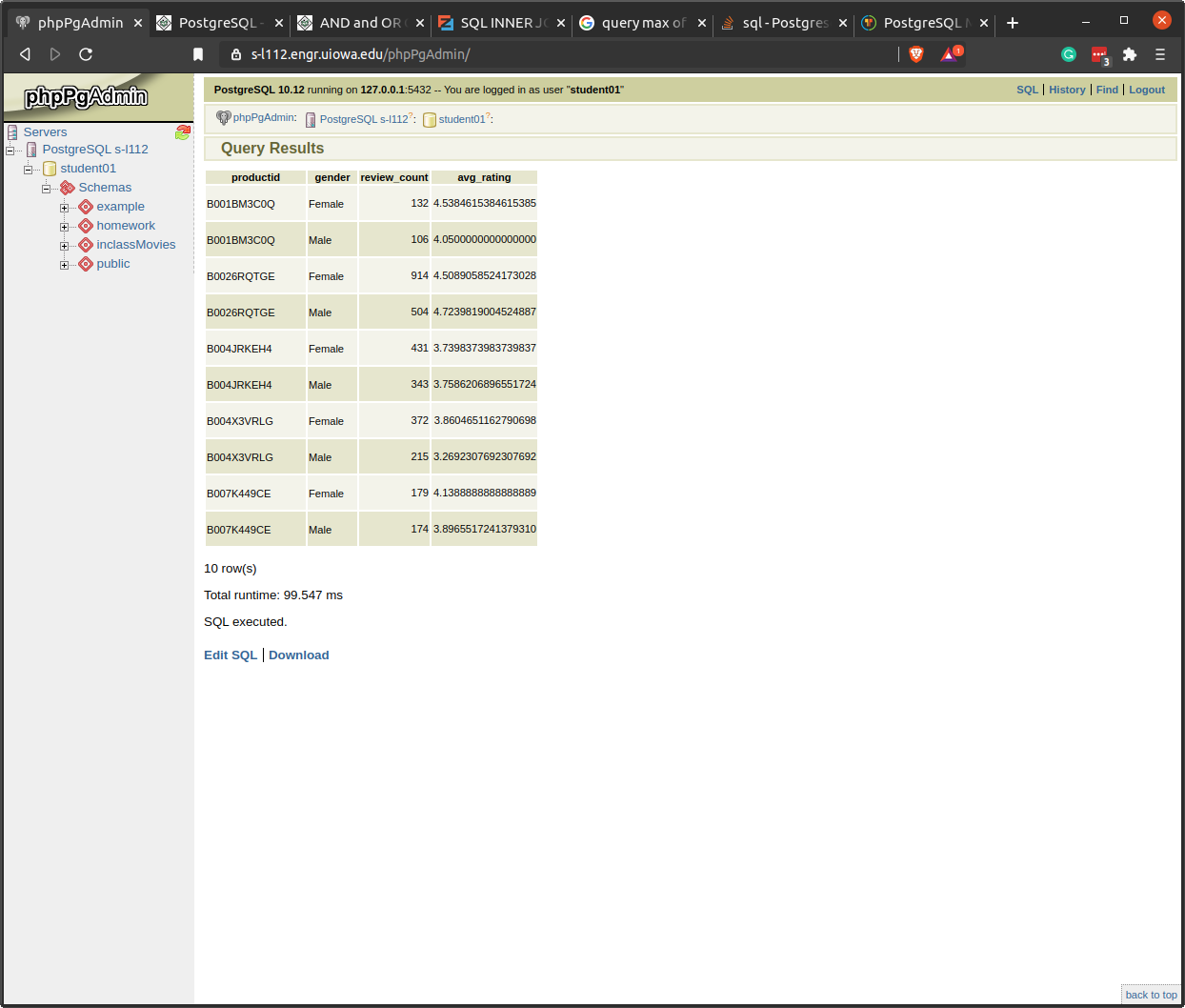
productid = 'B004X3VRLG' OR

productid = 'B0026RQTGE' OR

productid = 'B001BM3C0Q' OR

productid = 'B007K449CE'

GROUP BY productid, gender;



Q8

*-- Find the most helpful, highest rated review and the most helpful, lowest rated review for each product.*

*-- Show the full review (productid, userid, helpfulnessnumerator, helpfulnessdenominator, score, dposted, summary, review).*

*-- best score*

SELECT DISTINCT ON (productid) productid, userid, helpfulnessnumerator, helpfulnessdenominator, score, dposted, summary, review

FROM reviews r

WHERE

productid = 'B004JRKEH4' OR

productid = 'B004X3VRLG' OR

productid = 'B0026RQTGE' OR

productid = 'B001BM3C0Q' OR

productid = 'B007K449CE'

ORDER BY productid,

CASE

WHEN helpfulnessdenominator = 0 THEN 0

WHEN helpfulnessdenominator != 0 THEN helpfulnessnumerator::numeric/NULLIF(helpfulnessdenominator::numeric,0)

END DESC NULLS LAST, score DESC NULLS LAST, helpfulnessdenominator DESC NULLS LAST;

*-- worst score*

SELECT DISTINCT ON (productid) productid, userid, helpfulnessnumerator, helpfulnessdenominator, score, dposted, summary, review

FROM reviews r

WHERE

productid = 'B004JRKEH4' OR

productid = 'B004X3VRLG' OR

productid = 'B0026RQTGE' OR

productid = 'B001BM3C0Q' OR

productid = 'B007K449CE'

ORDER BY productid,

CASE

WHEN helpfulnessdenominator = 0 THEN 0

WHEN helpfulnessdenominator != 0 THEN helpfulnessnumerator::numeric/NULLIF(helpfulnessdenominator::numeric,0)

END DESC NULLS LAST, score ASC NULLS LAST, helpfulnessdenominator DESC NULLS LAST;