Kramer Johnson

CSPC 5021 02 HW4

11/2/20

a. We were able to get our team database into the AWS cloud with no issues.

1.

a.

```
/* Display top 20 shortstops by slugging percentage */
SELECT
   nameFirst AS 'First Name',
   nameLast  AS 'Last Name',
   Slugging
FROM (
            SELECT
               p.playerID,
               nameFirst,
               nameLast,
               POS,
               AB,
               b.yearId,
               ((H - 2B - 3B - HR) + (2B * 2) + (3B * 3) + (HR * 4)) / AB AS Slugging # Calculate slugging pct
            FROM people p
            JOIN batting b  ON p.playerID = b.playerID
            JOIN fielding f  ON p.playerID = f.playerID
            WHERE
                        POS = 'SS'          AND  # Only players who played as shortstop
                        b.yearID = '2019' AND  # Only data from 2019
                        f.G > 120                  # Only players who played as SS for more than 75% of games in 2019
            GROUP BY p.playerID
) t1
WHERE
   AB         > 300  AND # Players with at least 300 at bats
   Slugging > .400 # Slugging pct over .400
ORDER BY Slugging DESC
LIMIT 20;
```

b. The above SQL statement accurately retrieves the top shortstops by slugging percentage from 2019 (the last year we have data). The query ended up being more difficult than I expected and I may have overcomplicated it with the subquery. The squery is used to retrieve data about only players who played at least 120 games (3/4 of the season) as a shortstop in the year 2019. From this subquery, I pull the at bats (AB) and make sure they are over 300. I pull the slugging attribute and only get batters who had a percentage over .400. I order the results from the greatest slugging to lowest and display only the top 20.

	First Name	Last Name	Slugging
▶	Xander	Bogaerts	0.5554
	Trevor	Story	0.5544
	Javier	Baez	0.5311
	Marcus	Semien	0.5221
	Francisco	Lindor	0.5184
	Tim	Anderson	0.5080
	Trea	Turner	0.4971
	Jorge	Polanco	0.4849
	Corey	Seager	0.4826
	Ian	Desmond	0.4786
	Freddy	Galvis	0.4444
	Paul	DeJong	0.4443
	Didi	Gregorius	0.4414
	Nick	Ahmed	0.4371
	Starlin	Castro	0.4355
	Amed	Rosario	0.4318
	Dansby	Swanson	0.4224
	Jean	Segura	0.4201
	Willy	Adames	0.4181
	Jose	Iglesias	0.4067

c.

```
/* Display home run leading shortstops for last 10 years */
SELECT
   p.playerID,
   p.nameFirst       AS 'First Name',
   p.nameLast        AS 'Last Name',
   SUM(b.HR)         AS 'Home Runs',
   ROUND(AVG(b.HR))   AS 'Average HR'
FROM people p
            JOIN batting b  ON p.playerID = b.playerID
   JOIN (
            # Get table of shortstops in past 10 years
            SELECT playerID, POS
            FROM fielding
            WHERE

      yearID >= '2009' AND # only last 10 years
      POS     = 'SS'   AND # only shortstops
      G       > 120       # only players who played 75% of the season as shortstop
   GROUP BY playerID
   ) f ON p.playerID = f.playerID
WHERE
            b.yearID >= 2009
GROUP BY playerID
ORDER BY SUM(b.HR) DESC
LIMIT 20;
```

EX.

```
/* Display home run leading shortstops over the past 10 years that are under 30 */

SELECT
        p.playerID,
        p.nameFirst                 AS 'First Name',
  p.nameLast                AS 'Last Name',
  YEAR(CURDATE()) - p.birthYear     AS Age,
  SUM(b.HR)                 AS 'Total HR',
  ROUND(AVG(b.HR))              AS 'Avg HR'
FROM people p
        JOIN batting b ON p.playerID = b.playerID
  JOIN (
                # Get table of shortstops in past 10 years
                SELECT playerID, POS
                FROM fielding
                WHERE
                        yearID >= '2009' AND # only last 10 years
      POS   = 'SS'  AND # only shortstops
                        G     > 120     # only players who played 75% of the season as shortstop
                GROUP BY playerID
  ) f ON p.playerID = f.playerID
WHERE
        b.yearID >= 2009 AND            # Show results for past 10 years
  (YEAR(CURDATE()) - p.birthYear) < 30  # Only players under 30
GROUP BY playerID
ORDER BY SUM(b.HR) DESC
LIMIT 20;


/*
 * Stored procedure to get home run leading players since a
 * given year who are currently under a specified age
 *
 * @param maxAge    INT     the maximum age player to search for
 * @param sinceYear INT     the beginning year you want data for
 * @param position  CHAR(2) the position player you want to search for
 */
DELIMITER //
CREATE PROCEDURE filterPlayers(
        IN maxAge    INT,
  IN sinceYear INT,
  IN position  CHAR(2)

)
BEGIN
SELECT
        p.playerID,
        p.nameFirst                 AS 'First Name',
  p.nameLast                AS 'Last Name',
  YEAR(CURDATE()) - p.birthYear     AS Age,
  SUM(b.HR)                 AS 'Total HR',
  ROUND(AVG(b.HR))              AS 'Avg HR'
FROM people p
        JOIN batting b ON p.playerID = b.playerID
  JOIN (
        # Get table of shortstops in past 10 years
        SELECT playerID, POS
        FROM fielding
        WHERE
                yearID >= sinceYear AND # only since sinceYear
                POS     = position  AND # only players who played postion
                G     > 120         # only players who played 75% of the season as shortstop
        GROUP BY playerID
  ) f ON p.playerID = f.playerID
WHERE
        b.yearID >= sinceYear AND           # show results since sinceYear
       (YEAR(CURDATE()) - p.birthYear) < maxAge # only players under the max age
GROUP BY playerID
ORDER BY SUM(b.HR) DESC
LIMIT 20;

END //
DELIMITER ;
```

Sakila Queries:

```
-- Normal Join  Query #1  (MY PREFERENCE)
select a.actor_id, first_name, last_name, film_id
from actor a, film_actor fm
where a.actor_id = fm.actor_id;

-- SHORT CUT (NATURAL JOIN)
select customer_id, last_name, rental_id, rental_date
from customer natural join rental;

-- SHORT CUTS JOIN USING
select inv_number, p_code, p_descript, line_units, line_price
from invoice join line using (INV_NUMBER) join product using (P_CODE);

-- JOIN ON (HAVE TO USE ACTUAL JOIN COLUMNS)
select a.actor_id, a.first_name, a.last_name, f.film_id, f.title, f.description
from actor a join film_actor fm on a.actor_id = fm.actor_id
join film f on f.film_id = fm.film_id;
-- join ON
select store.manager_staff_id, staff.last_name, store.store_id
from store join staff on store.manager_staff_id = staff.staff_id
order by staff.last_name;


-- Left Outer Join (customers who don't have rentals will show up)
select rental_id, c.customer_id, first_name, last_name
from customer c left join rental r on c.customer_id = r.customer_id;

-- Right Outer Join (rentals that dont have customer will show up)
select rental_id, c.customer_id, first_name, last_name
from customer c right join rental r on c.customer_id = r.customer_id;

-- subqueries

select rental_id, rental.customer_id, last_name, first_name
from customer, rental
where customer.customer_id = rental.customer_id;

-- Subquery doesn't work
select v_code, v_name
from vendor
where v_code not in (select v_code from product);

-- subquery with where    AWESOME QUERY
select payment_id, amount from payment
where amount >= (select avg(amount) from payment);

-- subquery with where

select distinct c.customer_id, c.last_name, c.first_name
from customer c join rental using (customer_id)
join inventory using (inventory_id)
join film_actor using (film_id)
join actor using (actor_id)
where actor_id = (select actor_id from actor where last_name='SWANK');

-- Similar
select distinct c.customer_id, c.last_name, c.first_name
from customer c join rental using (customer_id)
join inventory using (inventory_id)
join film_actor using (film_id)
join actor a using (actor_id)
where a.last_name='SWANK';
```

```
-- in SubQueries
select distinct c.customer_id, c.last_name, c.first_name
from customer c join rental using (customer_id)
join inventory using (inventory_id)
join film_actor using (film_id)
join actor using (actor_id)
where actor_id in (select actor_id from actor
                                        where last_name like 'SW&'
                                        or last_name like '%WAN%');

-- in SubQueries  (SAME)
select distinct c.customer_id, c.last_name, c.first_name
from customer c join rental using (customer_id)
join inventory using (inventory_id)
join film_actor using (film_id)
join actor a using (actor_id)
where a.last_name like 'SW&'
                                        or a.last_name like '%WAN%';

-- subquery HAVING
select rental_id, sum(amount), AVG(amount)
from payment
group by rental_id
-- where sum(LINE_UNITS) > AVG(LINE_UNITS); -- Can't do this, that is why having
having sum(amount) > (SELECT AVG(amount) from payment);

-- subquery ALL AND ANY
select payment_id, amount
from payment
where amount > ALL(SELECT amount
                    from payment
                    where customer_id in (select customer_id
                                          from customer
                                          where address_id=2));


-- subquery ANY (DOESN'T Really make sense does it?)
select payment_id, amount
from payment
where amount > ANY(SELECT amount
                    from payment
                    where customer_id in (select customer_id
                                          from customer
                                          where address_id=2));


-- FROM SUBQUERIES

select distinct customer.customer_id, customer.last_name
from customer,
        (select rental.customer_id from rental natural join inventory
   where film_id=3) CP1,
   (select rental.customer_id from rental natural join inventory
   where film_id=7) CP2
where customer.customer_id=cp1.customer_id and cp1.customer_id=cp2.customer_id;

-- Attribute LIST SUBQUERIES

select payment_id, amount, (select avg(amount) from payment) as avgprice,
        amount-(select avg(amount) from payment) as diff
from payment;
```

```
-- correlated subquery  (Does outer first, then inner. This passes the first P_CODE from outer, and then
calcs the average for that product)
select rental_id, payment_id, amount
from payment p
where p.amount > (select avg(amount)
                                                    from payment pm
                                                    where pm.rental_id=p.rental_id);

-- exists query (correlated)   exists is only for subquerys

select customer_id, last_name, first_name
from customer
where exists (select customer_id from rental
                                        where rental.customer_id=
            customer.customer_id);

-- This doesn't work
select customer.customer_id, last_name, first_name
from customer, rental
where rental.customer_id=
customer.customer_id;

-- Date time queries
SELECT DAYOFMONTH('2001-11-10'), MONTH('2005-03-05');
SELECT ADDDATE('2008-01-02', 31);

select last_name, first_name, create_date, year(create_date) as YEARCREATE
from customer where year(create_date) > 2005;

-- Case SQL Statements
select lower(last_name) from customer;
select upper(last_name) from customer;
select last_name from customer where lower(last_name) like 'an%';

# these are bad naming conventions but I am keeping consistent with the original customer table
drop table CUSTOMER_2;
CREATE TABLE CUSTOMER_2 (
customer_id int,
last_name varchar(15),
first_name varchar(15),
active varchar(3),
email varchar(8)

);

INSERT INTO CUSTOMER_2 VALUES(345,'Terrell','Justine','615','322-9870');
INSERT INTO CUSTOMER_2 VALUES(347,'Olowski','Paul',615,'894-2180');
INSERT INTO CUSTOMER_2 VALUES(351,'Hernandez','Carlos','723','123-7654');
INSERT INTO CUSTOMER_2 VALUES(352,'McDowell','George','723','123-7768');
INSERT INTO CUSTOMER_2 VALUES(365,'Tirpin','Khaleed','723','123-9876');
INSERT INTO CUSTOMER_2 VALUES(368,'Lewis','Marie','734','332-1789');
INSERT INTO CUSTOMER_2 VALUES(369,'Dunne','Leona','713','894-1238');


-- Union Query
select last_name, first_name, active, email
from customer
union
select last_name, first_name, active, email from CUSTOMER_2;

-- Intersect Query (MYSQL DOES NOT SUPPORT)

select customer_id from customer
where active='1' and
```

```
customer_id in (SELECT DISTINCT  customer_id from rental);

-- minus alternative
select customer_id from customer
where active='1' and
customer_id not in (SELECT DISTINCT  customer_id from rental);

-- create view
create view pmt_stats as
select rental_id, sum(amount) as totcost, max(amount) as MaxAmt,
            MIN(amount) AS MinAmt, AVG(amount) AS AvgAmt
   FROM payment
   GROUP BY rental_id;

select * from pmt_stats;

-- updatable views

-- Triggers (Row level)


CREATE TABLE staff_audit (
   id INT AUTO_INCREMENT PRIMARY KEY,
   staff_id INT NOT NULL,
   last_name VARCHAR(50) NOT NULL,
   changedat DATETIME DEFAULT NULL,
   action VARCHAR(50) DEFAULT NULL
);

CREATE TRIGGER before_staff_update
   BEFORE UPDATE ON staff
   FOR EACH ROW
 INSERT INTO staff_audit
 SET action = 'update',
    staff_id = OLD.staff_id,
    last_name = OLD.last_name,
    changedat = NOW();

show triggers;
UPDATE staff
SET
   last_name = 'Phan'
WHERE
   staff_id = 1;
drop trigger before_staff_update;
-- Triggers (Stemployees_auditatement level)

-- Stored Procedure
```