Kramer Johnson

CSPC 5021 02 HW4

11/2/20

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

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

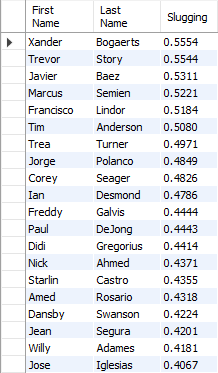
AB > 300 AND # Players with at least 300 at bats

Slugging > .400 # Slugging pct over .400

ORDER BY Slugging DESC

LIMIT 20;

1. 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.



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