**STAT 37820**

**HW 2**

**Siying Cao**

**UCID 10445633**

## Part 1. Database creation and modification

**Q2. Create two non-trivial queries, each should retrieve data from multiple tables in the database "School". Note: You may insert additional values in any tables to facilitate your queries.**

To facilitate my query, I insert more records into the tests and scores tables so that we end up with 19 unique tests, 11 unique students, and 57 records for scores. Note only a few students showed up in each test based on my construction. I only include the top five new records creation for illustration purpose.

**Code**

```
INSERT INTO tests VALUES
('2015-8-25', 'T', 30, 3, NULL),
('2015-8-29', 'T', 30, 4, NULL),
('2015-8-29', 'T', 30, 5, NULL),
('2015-8-27', 'Q', 15, 5, NULL),
('2015-8-30', 'T', 30, 5, NULL)

ALTER TABLE scores MODIFY COLUMN student_id INT UNSIGNED AFTER test_id;

INSERT INTO scores VALUES
(1, 1, 9),
(1, 3, 14),
(1, 9, 4),
(2, 3, 5),
(2, 9, 10)
```

Query #1. Which class has the highest and lowest average test scores?

```
SELECT classes.class_id, class_name, AVG(score) FROM classes, tests, scores
WHERE classes.class_id=tests.class_id
```

```
AND tests.test_id=scores.test_id
AND tests.type='T'
GROUP BY classes.class_id
HAVING COUNT(score) IS NOT NULL
ORDER BY -AVG(score);
```

Query #2. Find the name of the student who got the highest average score in Calculus tests (excluding quiz).

```
SELECT students.student_id, CONCAT(last_name, ',', first_name) AS name, AVG(score)
FROM students, scores, tests
WHERE students.student_id=scores.student_id
AND scores.test_id=tests.test_id
AND tests.type='T'
GROUP BY students.student_id
ORDER BY AVG(score);
```

## Part 2. More SQL exercises on Sakila database

**Q1. Show your code of finding the top five movies that earned the most money for the rental business. Also list the number of times the movies were rent out. Show results.**

**Code**

```
USE sakila;
SHOW TABLES;
SELECT film.film_id, title, SUM(amount) AS revenue,
COUNT(rental.rental_id) AS rent_times, film.rental_rate,
film.rental_rate*COUNT(rental.rental_id) AS direct_rev
FROM film, inventory, rental, payment
WHERE film.film_id=inventory.film_id
AND inventory.inventory_id=rental.inventory_id
AND rental.rental_id=payment.rental_id
GROUP BY film.film_id
ORDER BY -SUM(amount)
LIMIT 5;
```

**Output**

| film_id | title | revenue | rent_times | rental_rate | direct_rev |
|---|---|---|---|---|---|
| 879 | TELEGRAPH VOYAGE | 231.73 | 27 | 4.99 | 134.73 |
| 973 | WIFE TURN | 223.69 | 31 | 4.99 | 154.69 |
| 1000 | ZORRO ARK | 214.69 | 31 | 4.99 | 154.69 |
| 369 | GOODFELLAS SALUTE | 209.69 | 31 | 4.99 | 154.69 |
| 764 | SATURDAY LAMBS | 204.72 | 28 | 4.99 | 139.72 |

*Note: The idea is to find out the payment accruing to rentals associated with inventories of a given film. A little complication here arises as we have two variables indicating money earned on each rental. The first is rental_rate in FILM table while the other is payment in RENTAL table. It turns out that rental rate times the number of rentals for a given film rarely coincide with the total payment associated with the same film. This complication is a consequence of overdue charges, which I verified by combining the return and due date with the revenue data. For this reason, the code is written based on total payment rather than rental rate.*

**Q2. Create a nontrivial query using LEFT JOIN.**

```
# Query: Find out films that don't have actors
SELECT film.film_id, title, actor.actor_id, CONCAT(last_name, ',', first_name)
FROM (film LEFT JOIN film_actor ON film.film_id=film_actor.film_id), actor
WHERE actor.actor_id=film_actor.actor_id
ORDER BY film.title;
```

**Q3. Create a nontrivial query using correlated sub-query.**

```
# Query: How many rentals accrue to movies in the drama categories?
SELECT COUNT(rental.rental_id) FROM rental, inventory, category, film, film_category
WHERE film_category.film_id=film.film_id
AND film_category.category_id=(SELECT category_id FROM category WHERE
name='Drama')
AND film.film_id=inventory.film_id
AND inventory.inventory_id=rental.inventory_id;
```

## Part 3. Try SAS

**Q1. Create a small program by inputting data and run a few procedures, provide the code and selected output.**

**Code**

```
/* Input the first dataset*/
data dt1;
        infile datalines;
        input id $ x;
        datalines;
a 1.6
b 1.2
c 1.9
d 0.7
;
run;
/*Input the second dataset*/
data dt2;
        infile datalines;
        input id $ y z;
        datalines;
a 14 10
b 21 9
c 5  6
d 12 8
;
run;

/*Output to the right window*/
ods listings;

/*Procedure: sort the two datasets to prepare for merging*/
proc sort data=dt1;
        by id;
```

```
proc sort data=dt2;
        by id;
/*Create a new dataset by merging the two*/
data new;
        merge dt1 dt2;
        by id;
/*Procedure: Show the new dataset*/
proc print; run;

/*Procedure: Run simple linear regression*/
proc reg data=new;
        model z = x y;
run;
```

**Output (from screenshots)**

The SAS System 22:56 Monday, November 14, 2016 1

| Obs | id | y | z |
|---|---|---|---|
| 1 | a | 14 | 10 |
| 2 | b | 21 | 9 |
| 3 | c | 5 | 6 |
| 4 | d | 12 | 8 |

```
b position at 26.
```

The SAS System 22:56 Monday, November 14, 2016 2

The REG Procedure
Model: MODEL1
Dependent Variable: z

| | |
|---|---|
| Number of Observations Read | 4 |
| Number of Observations Used | 4 |

Analysis of Variance

| Source | DF | Sum of Squares | Mean Square | F Value | Pr > F |
|---|---|---|---|---|---|
| Model | 2 | 5.25640 | 2.62820 | 0.75 | 0.6319 |
| Error | 1 | 3.49360 | 3.49360 | | |
| Corrected Total | 3 | 8.75000 | | | |

| | | | |
|---|---|---|---|
| Root MSE | 1.86912 | R-Square | 0.6007 |
| Dependent Mean | 8.25000 | Adj R-Sq | -0.1978 |
| Coeff Var | 22.65597 | | |

The REG Procedure
Model: MODEL1
Dependent Variable: z

Parameter Estimates

| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > \|t\| |
|---|---|---|---|---|---|
| Intercept | 1 | 5.10975 | 4.84711 | 1.05 | 0.4832 |
| x | 1 | 0.29684 | 2.33626 | 0.13 | 0.9195 |
| y | 1 | 0.21073 | 0.18441 | 1.14 | 0.4577 |



| Obs | id | x | y | z |
|---|---|---|---|---|
| 1 | a | 1.6 | 14 | 10 |
| 2 | b | 1.2 | 21 | 9 |
| 3 | c | 1.9 | 5 | 6 |
| 4 | d | 0.7 | 12 | 8 |