Nicholas Capriotti

CS317 - Dr. Van Hilst.

9/12/2022

Exercise 1B

Continuing the work of assignment 1_a, this assignment is meant to continue the basic introduction to MySQL. This assignment includes an introduction to the 'join' function that will allow us to combine multiple data sets into one for the purpose of database management. The assignment will be an intro to the Left Join, right Join, and Inner Join, all of which can be used to compile multiple data sets into one table based on shared categories.

This is an extension of assignment 1_a, so the setup method is the exact same, however, make sure you are in the correct database by using 'USE sakila;'

First, we are asked to create a query that will return the number of films that have ever been rented, not the instances of a film being rented, but rather the actual number of unique films that have been rented out. The first query will have us pull our data from the 'rental' table where we have a rental, but it will only count each name one time, so a movie that is rented multiple times does not give us several "different" rentals. This tells us that there have been 4580 different movies rented out.

```
mysql> select count(distinct inventory_id) from rental;
+-----------------------------+
| count(distinct inventory_id) |
+-----------------------------+
|                        4580 |
+-----------------------------+
1 row in set (0.00 sec)
```

Next, we are asked to find the number of films that are currently rented out, which means they haven't been returned yet, this will be done with the addition of a "where" clause. A where clause allows us to add conditions to our query to help us narrow down our selection. Using this new clause narrows it down to 183 films that have been rented but not returned.

```
mysql> select count(rental_id) from rental where return_date is NULL;
+------------------+
| count(rental_id) |
+------------------+
|              183 |
+------------------+
```

The where clause works because it allows us to limit our entries to only the data entries that fit the criteria, in this case, by asking to find the count where the 'return_date' is NULL, we will select all rented movies that have no return date, meaning they have yet to be returned. This query returns a count of 183 movies that have yet to be returned. We use the rental_id rather than the inventory_id to avoid any double counting.

Next, we want to find all films that were rented out and not returned, however, we want to only find films that have not been returned by a specific date. This requires more filtering, and

we'll be looking at 2 columns specifically to find the information: “rental_date” and “return_date”. We are looking for any movie that has not been returned as of July 17, 2005.

```
mysql> select count(rental_id) from rental
    -> where rental_date < '2005-7-17'
    -> and(return_date > '2005-7-17'
    -> or return_date is NUll);
+------------------+
| count(rental_id) |
+------------------+
|              822 |
```

All we had to do was adjust the where clause to include 3 total conditions, we are going to check if the “return_date” is before, or after July 17 2005, or if the return date simply does not exist at all, we are making sure it would have been rented out before July 17, 2005. This query tells us that on July 17, 2005, there were 822 movies that were checked out, but not returned.

The next part asks us to answer a similar question: as of July 17, 2005, how many rentals were overdue. This means that the rental date must be before July 17, 2005 and the rental duration must have ended by that date, and that must also mean that as of July 17, 2005, there had been no return date entered into the database.

```
mysql> select count(r.rental_id)
    -> from rental r
    -> inner join rented_out ro
    -> on r.rental_id = ro.rental_id
    -> where r.rental_date < '2005-7-17'
    -> and r.return_date > '2005-7-17'
    -> and ro.due_date < '2005-7-17';
+--------------------+
| count(r.rental_id) |
+--------------------+
|                565 |
+--------------------+
```

We will use an inner join on this query to merge 2 data tables together so we can use both sets of information. In this query we want to join the “rented_out” table with the “rental” table, this allows us to have the due date for each rental. We also use variable names like r and ro for the “rental” and “rented_out” tables respectively; this is to save time and make our code neater in the future when we may be referencing the same table many times over. This query told us that as of July 17, 2005, there had been 565 movies overdue.

For the next part, we were tasked with finding a percentage value to represent the fraction of all DVDs that were overdue by the time they were returned. This will require us to perform operations on several sets of data that are in different tables within the database, we will use a nested query to do this. The nested query will allow us to access the different tables and gather the needed information without the need to perform each operation in its own command.

```
mysql> select
    ->     (
    ->         count(r.rental_id) * 100 / (
    ->             select
    ->                 count( r.rental_id)
    ->             from
    ->                 rental r
    ->                 inner join rented_out ro on r.rental_id = ro.rental_id
    ->         )
    ->     ) as percentage
    -> from
    ->     rental r
    ->     inner join rented_out ro on r.rental_id = ro.rental_id
    -> where
    ->     r.return_date > date_add(ro.due_date, interval 1 day);
+------------+
| percentage |
+------------+
|    45.3067 |
+------------+
```

Our query contains several parts, mainly, the 2 select statements give us the number of overdue rentals and the number of total rentals, the nested query allows us to have both select statements run in the same command, this allows us to work directly with the 2 counts give, (total rentals and overdue rentals) without the need to store that information anywhere, This means that we can run the query to get a percentage to print to the console. We are working with the assumption that we can find an overdue rental by using the “date-add()” function in our query to check if the return date is after the original due date, this would mean that the rental ran longer than was originally anticipated. The query tells us that approximately 45.3% of all rentals were overdue.

Next, we are asked to list films that were rented out as of July 17, 2005, however, this time, we want the names of the films and they are to be sorted by customer_id, and limited to 5 films. We will limit the number of results by setting a limit of 5 rows, and we can check that the order is not by customer_id. We will check for films rented during the selected time by using the same method as last time, this will give us a table with the rental id, rental date, inventory id, customer id (ascending), return date, staff id, and the last update, the results will only show 5 films.

```
mysql> select * from rental
    -> where rental_date < '2005-7-17'
    -> and(return_date > '2005-7-17'
    -> or return_date is NUll)
    -> order by customer_id
    -> limit 5;
+-----------+---------------------+--------------+-------------+---------------------+----------+---------------------+
| rental_id | rental_date         | inventory_id | customer_id | return_date         | staff_id | last_update         |
+-----------+---------------------+--------------+-------------+---------------------+----------+---------------------+
|      6163 | 2005-07-11 10:13:46 |         1330 |           1 | 2005-07-19 13:15:46 |        2 | 2006-02-15 21:30:53 |
|      5755 | 2005-07-10 12:38:56 |         2760 |           2 | 2005-07-19 17:02:56 |        1 | 2006-02-15 21:30:53 |
|      5118 | 2005-07-09 07:13:52 |          957 |           5 | 2005-07-18 05:18:52 |        2 | 2006-02-15 21:30:53 |
|      5721 | 2005-07-10 11:09:35 |         2348 |           5 | 2005-07-17 16:41:35 |        2 | 2006-02-15 21:30:53 |
|      6042 | 2005-07-11 03:17:04 |         2153 |           5 | 2005-07-19 07:08:04 |        1 | 2006-02-15 21:30:53 |
+-----------+---------------------+--------------+-------------+---------------------+----------+---------------------+
```

In this next part we start by copying a query from the MySQL official site, this will give us a list of overdue DVDs like some of our prior queries, Running the MySQL query gives us a table like the one shown below:

```
mysql> SELECT CONCAT(customer.last_name, ', ', customer.first_name) AS customer,
    -> address.phone, film.title
    -> FROM rental INNER JOIN customer ON rental.customer_id = customer.customer_id
    -> INNER JOIN address ON customer.address_id = address.address_id
    -> INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
    -> INNER JOIN film ON inventory.film_id = film.film_id
    -> WHERE rental.return_date IS NULL
    -> AND rental_date + INTERVAL film.rental_duration DAY < CURRENT_DATE()
    -> ORDER BY title
    -> LIMIT 5;
+----------------+--------------+------------------+
| customer       | phone        | title            |
+----------------+--------------+------------------+
| OLVERA, DWAYNE | 62127829280  | ACADEMY DINOSAUR |
| HUEY, BRANDON  | 99883471275  | ACE GOLDFINGER   |
| OWENS, CARMEN  | 272234298332 | AFFAIR PREJUDICE |
| HANNON, SETH   | 864392582257 | AFRICAN EGG      |
| COLE, TRACY    | 371490777743 | ALI FOREVER      |
+----------------+--------------+------------------+
```

This query uses multiple Inner Join functions to do multiple things: the first one joins to the address table to return customer phone numbers, there is a join to Inventory, this is probably used in the same way that we used it above where we wanted to limit the double counts, the third join is to the film table, this can give the title, rental duration, and due date. While there are only 3 visible columns, it is fascinating to see how much information sharing is needed to make things like this appear when your database has its information spread across several tables.

We are then asked to modify this query by first replacing the “current_date()” with the same date we have been working with: July 17, 2005. We also want to get rid of the limit which would stop the table from printing up to a set number of results, without a limit in place, we would get every result at once.

```
mysql> SELECT CONCAT(customer.last_name, ', ', customer.first_name) AS customer,
    -> address.phone, film.title
    -> FROM rental INNER JOIN customer ON rental.customer_id = customer.customer_id
    -> INNER JOIN address ON customer.address_id = address.address_id
    -> INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
    -> INNER JOIN film ON inventory.film_id = film.film_id
    -> WHERE rental.return_date IS NULL
    -> AND rental_date + INTERVAL film.rental_duration DAY < '2005-07-17'
    -> ORDER BY title;
```

This query returns an empty set, this is because we are asking for all movies that were overdue on July 17, 2005, but had not been returned at all. While the returns may be late, there is no instance of us seeing a film not be returned at all if it was overdue on that date. This is a fair result as it's fair to assume that it would be rare to see a film not be returned at all for several years. There is no empty table made, as it's not necessary, there is no need to waste computing time creating a table that will not be populated, so the empty set is returned.

This exercise was a good way to start the dive into the more complex functionalities of the MySQL system. There is a lot of functionality even though the commands are still relatively simple. It's all about being a good problem solver and knowing what information you need from each table, this means that there will be a huge emphasis on using the correct join functions. I think that with MySQL, there is less to be learned about the syntax, and more to be gained from the problem solving skills that are being introduced.