/\*Working with Temporal Data

Temporal data is the most involved when it comes to data generation and manipulation.\*/

SELECT UTC\_TIMESTAMP();

SELECT UTC\_DATE();

SELECT UTC\_TIME();

/\*If you are sitting at a computer in Zurich, Switzerland, and you open a session across the network to a MySQL server situated in New York, you may want to change the time zone setting for your session, which you can do via the following command:\*/

SELECT @@GLOBAL.time\_zone, @@SESSION.time\_zone;

/\*Generating Temporal Data - You can generate temporal data via any of the following means:

• Copying data from an existing date, datetime, or time column

• Executing a built-in function that returns a date, datetime, or time

• Building a string representation of the temporal data to be evaluated by the server

date YYYY-MM-DD

datetime YYYY-MM-DD HH:MI:SS

timestamp YYYY-MM-DD HH:MI:SS

time HHH:MI:SS\*/

use sakila;

UPDATE rental

SET return\_date = '2022-12-07 09:50:00'

WHERE rental\_id = 99999;

/\*String to Date Conversions

If the server is not expecting a datetime value or if you would like to represent the datetime using a nondefault format, you will need to tell the server to convert the string to a datetime using cast( ) function\*/

SELECT CAST('2022-12-07 09:50:00' AS DATETIME);

SELECT CAST('2022-12-07' AS DATE) date\_field,

CAST('09:50:00' AS TIME) time\_field;

/\*use a built-in function that allows you to provide a format string along with the date string str\_to\_date( )\*/

UPDATE rental

SET return\_date = STR\_TO\_DATE('december 7, 2022', '%M %d, %Y')

WHERE rental\_id = 99999;

SELECT CURRENT\_DATE(), CURRENT\_TIME(), CURRENT\_TIMESTAMP();

/\*Manipulating Temporal Data

Many of the built-in temporal functions take one date as an argument and return another date. MySQL’s date\_add() function Temporal functions that return dates\*/

SELECT DATE\_ADD(CURRENT\_DATE(), INTERVAL 5 DAY);

/\*second Number of seconds

minute Number of minutes

hour Number of hours

day Number of days

month Number of months

year Number of years

minute\_second Number of minutes and seconds, separated by “:”

hour\_second Number of hours, minutes, and seconds, separated by “:”

year\_month Number of years and months, separated by “-”\*/

UPDATE rental

SET return\_date = DATE\_ADD(return\_date, INTERVAL '3:27:11' HOUR\_SECOND)

WHERE rental\_id = 99999;

UPDATE employee

SET birth\_date = DATE\_ADD(birth\_date, INTERVAL '9-11' YEAR\_MONTH)

WHERE emp\_id = 4789;

SELECT LAST\_DAY('2022-12-07');

/\*Temporal functions that return strings

Most of the temporal functions that return string values are used to extract a portion of a date or time. For example, MySQL includes the dayname() function to determine which day of the week a certain date falls on\*/

SELECT DAYNAME('1995-11-18');

/\*Many such functions are included with MySQL for extracting information from date values, but I recommend that you use the extract() function instead, since it’s easier to remember a few variations of one function than to remember a dozen different functions.\*/

SELECT EXTRACT(DAY FROM '1995-11-18');

SELECT EXTRACT(WEEK FROM '1995-11-18');

/\*Temporal functions that return numbers\*/

SELECT DATEDIFF('1995-11-18', '2022-12-07');

SELECT DATEDIFF('2022-12-07', '1995-11-18');

/\*The datediff() function ignores the time of day in its arguments.\*/

/\*Conversion Functions\*/

-- String to Integer

SELECT CAST('1456328' AS SIGNED INTEGER);

/\*When converting a string to a number, the cast() function will attempt to convert the entire string from left to right; if any nonnumeric characters are found in the string, the conversion halts without an error\*/

SELECT CAST('999ABC111' AS UNSIGNED INTEGER);

/\*Exercise - 1

Write a query that returns the 17th through 25th characters of the string 'Please find the substring in this string'.\*/

SELECT SUBSTRING('Please find the substring in this string', 17, 9) AS Answer;

/\*Exercise - 2

Write a query that returns the absolute value and sign (−1, 0, or 1) of the number −25.76823. Also return the number rounded to the nearest hundredth.\*/

SELECT ABS(-25.76823);

SELECT SIGN(-25.76823);

SELECT ROUND(-25.76823, 2);

-- or

SELECT ABS(-25.76823) AS Absolute,

SIGN(-25.76823) AS just\_the\_sign,

ROUND(-25.76823, 2) AS rounded\_value;

/\*Exercise - 3

Write a query to return just the month portion of the current date.\*/

SELECT EXTRACT(MONTH FROM CURRENT\_DATE());

/\*Grouping and Aggregates

Sometimes you will want to find trends in your data that will require the database server to cook the data a bit before you can generate the results you are looking for.\*/

SELECT customer\_id

FROM rental;

SELECT customer\_id

FROM rental

GROUP BY customer\_id;

/\*To see how many films each customer rented, you can use an aggregate function in the select clause to count the number of rows in each group\*/

SELECT customer\_id, COUNT(\*) AS number\_of\_films

FROM rental

GROUP BY customer\_id;

/\*In order to determine which customers have rented the most films, simply add an order by clause\*/

SELECT customer\_id, COUNT(\*) AS number\_of\_films

FROM rental

GROUP BY customer\_id

ORDER BY 2 DESC;

/\*When grouping data, you may need to filter out undesired data from your result set based on groups of data rather than based on the raw data. Since the group by clause runs after the where clause has been evaluated, you cannot add filter conditions to your where clause for this purpose. you must put your group filter conditions in the having clause\*/

SELECT customer\_id, COUNT(\*)

FROM rental

GROUP BY customer\_id

HAVING COUNT(\*) >= 35;

SELECT customer\_id, COUNT(\*)

FROM rental

GROUP BY customer\_id

HAVING COUNT(\*) >= 35

ORDER BY 2 DESC;

/\*Aggregate functions perform a specific operation over all rows in a group.

max()

Returns the maximum value within a set

min()

Returns the minimum value within a set

avg()

Returns the average value across a set

sum()

Returns the sum of the values across a set

count()

Returns the number of values in a set\*/

SELECT MAX(amount) AS max\_amount

FROM payment;

SELECT MIN(amount) AS min\_amount

FROM payment;

SELECT AVG(amount) AS avg\_amount,

SUM(amount) AS total\_amount,

COUNT(\*) num\_of\_payments

FROM payment;

SELECT customer\_id,

MAX(amount) max\_amt,

MIN(amount) min\_amt,

AVG(amount) avg\_amt,

SUM(amount) tot\_amt,

COUNT(\*) num\_payments

FROM payment;

/\*While it may be obvious to you that you want the aggregate functions applied to each customer found in the payment table, this query fails because you have not explicitly specified how the data should be grouped. Therefore, you will need to add a group by clause to specify over which group of rows the aggregate functions should be applied\*/

SELECT customer\_id,

MAX(amount) max\_amt,

MIN(amount) min\_amt,

AVG(amount) avg\_amt,

SUM(amount) tot\_amt,

COUNT(\*) num\_payments

FROM payment

GROUP BY customer\_id;

/\*With the inclusion of the group by clause, the server knows to group together rows having the same value in the customer\_id column first and then to apply the five aggregate functions to each of the 599 groups. \*/

/\*Counting Distinct Values\*/

SELECT COUNT(customer\_id) num\_rows,

COUNT(DISTINCT customer\_id) num\_customers

FROM payment;

/\*Using Expressions

Along with using columns as arguments to aggregate functions, you can use expressions as well.\*/

SELECT MAX(DATEDIFF(return\_date, rental\_date))

FROM rental;

/\*How Nulls Are Handled while grouping\*/

CREATE TABLE number\_tbl (

val SMALLINT

);

INSERT INTO number\_tbl VALUES (1);

INSERT INTO number\_tbl VALUES (3);

INSERT INTO number\_tbl VALUES (5);

SELECT COUNT(\*) num\_rows,

COUNT(val) num\_vals,

SUM(val) total,

MAX(val) max\_val,

AVG(val) avg\_val

FROM number\_tbl;

/\*The results are as you would expect: both count(\*) and count(val) return the value 4, sum(val) returns the value 10, max(val) returns 5, and avg(val) returns 2.5.\*/

INSERT INTO number\_tbl VALUES (NULL);

SELECT COUNT(\*) num\_rows,

COUNT(val) num\_vals,

SUM(val) total,

MAX(val) max\_val,

AVG(val) avg\_val

FROM number\_tbl;

/\*Even with the addition of the null value to the table, the sum(), max(), and avg() functions all return the same values, indicating that they ignore any null values encountered. The count(\*) function now returns the value 5, which is valid since the number\_tbl table contains four rows, while the count(val) function still returns the value 4. The difference is that count(\*) counts the number of rows, whereas count(val) counts the number of values contained in the val column and ignores any null values encountered.\*/

/\*Single-Column Grouping\*/

SELECT actor\_id, COUNT(\*) as num\_of\_films

FROM film\_actor

GROUP BY actor\_id;

/\*Multicolumn Grouping

Expanding on the previous example, imagine that you want to find the total number of films for each film rating (G, PG, ...) for each actor.\*/

SELECT fa.actor\_id, f.rating, COUNT(\*) AS num\_of\_movies

FROM film\_actor AS fa

INNER JOIN film AS f USING (film\_id)

GROUP BY fa.actor\_id, f.rating

ORDER BY 1, 2;

-- or

SELECT fa.actor\_id, f.rating, COUNT(\*) AS num\_of\_movies

FROM film\_actor AS fa

INNER JOIN film AS f ON fa.film\_id = f.film\_id

GROUP BY fa.actor\_id, f.rating

ORDER BY 1, 2;

desc actor;

SELECT fa.actor\_id,

a.first\_name,

a.last\_name,

f.rating,

COUNT(\*) AS num\_of\_movies

FROM film\_actor AS fa

INNER JOIN film AS f ON fa.film\_id = f.film\_id

INNER JOIN actor AS a USING (actor\_id)

GROUP BY fa.actor\_id, f.rating

ORDER BY 1, 2;

/\*Grouping via Expressions\*/

SELECT EXTRACT(YEAR FROM rental\_date) AS year, COUNT(\*) how\_many

FROM rental

GROUP BY EXTRACT(YEAR FROM rental\_date);

/\*Generating Rollups\*/

SELECT fa.actor\_id, f.rating, COUNT(\*)

FROM film\_actor fa

INNER JOIN film f ON fa.film\_id = f.film\_id

GROUP BY fa.actor\_id, f.rating WITH ROLLUP

ORDER BY 1, 2;

/\*Group Filter Conditions

When grouping data, you also can apply filter conditions to the data after the groups have been generated. The having clause is where you should place these types of filter conditions.\*/

SELECT fa.actor\_id, f.rating, COUNT(\*) AS num\_of\_films

FROM film\_actor AS fa

INNER JOIN film AS f USING (film\_id)

WHERE f.rating IN ('G', 'PG')

GROUP BY 1, 2;

SELECT fa.actor\_id, f.rating, COUNT(\*) AS num\_of\_films

FROM film\_actor AS fa

INNER JOIN film AS f USING (film\_id)

WHERE f.rating IN ('G', 'PG')

GROUP BY 1, 2

HAVING num\_of\_films > 10;

/\*This query has two filter conditions: one in the where clause, which filters out any films rated something other than G or PG, and another in the having clause, which filters out any actors who appeared in more than 10 films. Thus, one of the filters acts on data before it is grouped, and the other filter acts on data after the groups have been created.\*/

/\*Exercise - 1

Construct a query that counts the number of rows in the payment table.\*/

desc payment;

SELECT COUNT(\*) AS number\_of\_rows

FROM payment;

/\*Exercise - 2

Modify your query from Exercise - 1 to count the number of payments made by each customer. Show the customer ID and the total amount paid for each customer.\*/

SELECT customer\_id, COUNT(payment\_id) AS num\_of\_payment, sum(amount) AS total\_paid

FROM payment

GROUP BY customer\_id;

SELECT customer\_id, COUNT(payment\_id) AS num\_of\_payments, sum(amount) AS total\_paid

FROM payment

GROUP BY customer\_id

ORDER BY 3 DESC;

/\*Exercise - 3

Modify your query from Exercise 8-2 to include only those customers who have made at least 40 payments.\*/

SELECT customer\_id, COUNT(payment\_id) AS num\_of\_payments, sum(amount) AS total\_paid

FROM payment

GROUP BY customer\_id

HAVING num\_of\_payments >= 40

ORDER BY 3 DESC;