Functions

Przemysław Rola

May 18, 2025

What is a Function?

- A set of instructions that processes data in a specific way.
- Functions can take one or more arguments, or operate without parameters.
- **Types of Functions:**
  - **Built-in:** Examples include aggregating, numeric, date, and string processing functions.
  - **User-defined:** Created by the administrator for the needs of the DBMS.
- Example of a string function: `CONCAT()`, which combines strings.
- Note: The `SELECT` keyword can call any function, e.g.:

```
1 SELECT CONCAT('This is', ' fun!') AS Function;
```

- Result: A table with a single column labeled "Function" and one row: "This is fun!".

String Functions

- **Examples of String Functions:**
  - `CHAR_LENGTH(arg)`: Returns the number of characters in `arg`.
  - `CONCAT_WS(sep, arg1, arg2, ...)`: Combines strings with separator `sep`.
  - `LCASE(str)`: Converts all characters to lowercase.
  - `UCASE(str)`: Converts all characters to uppercase.
  - `LEFT(str, n)`: Returns the first `n` characters of `str`.
  - `RIGHT(str, n)`: Returns the last `n` characters of `str`.
  - `REPLACE(str, substr1, substr2)`: Replaces `substr1` with `substr2`.
  - `SUBSTRING(str, m, n)`: Returns a substring of `n` characters from position `m`.
- Example: Extract a student's index number from their email:

```
1 SELECT
    SUBSTRING('s333333@student.uek.krakow.pl',
    2, 6) AS Index;
```

- Result: "333333".

Arithmetic Operators

- Among numeric functions, we start with arithmetic operators, which perform operations on numeric columns (not numeric strings!).
  - `+`: Addition operator.
  - `-`: Subtraction operator.
  - `*`: Multiplication operator.
  - `/`: Division operator.
  - `DIV`: Integer division operator.
  - `MOD` (or `%`): Modulo operator—returns the remainder of division.
- **Other Numeric Functions:**
  - `ROUND(arg, n)`: Rounds `arg` to `n` decimal places.
  - `FLOOR(arg)`: Returns the floor (integer part) of `arg`.
  - `CEIL(arg)`: Returns the ceiling of `arg` (smallest integer not less than `arg`).
- Example:

```
1 SELECT COUNT(*) FROM film ORDER BY
    rental_per_day, title;
```

Aggregate Functions - COUNT

- **Aggregate Functions:**
  - Allow simple calculations on large sets of rows.
  - SQL standard defines five: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.
  - Operate only on rows without `NULL` values in the processed column (or with at least one non-empty field when using `*`).
- `COUNT`: Returns the number of query results; works on all field types.
- Examples:

```
SELECT COUNT(*) AS p2 FROM rental WHERE
    staff_id = 2;
```

- To count distinct values, use `DISTINCT`:

```
SELECT COUNT(DISTINCT inventory_id) AS
    rentals FROM rental;
```

- Calculate multiple metrics or their ratio:

```
SELECT COUNT(inventory_id) / COUNT(DISTINCT
    customer_id) AS average FROM rental;
```

Aggregate Functions - SUM, AVG, MIN, MAX

- `SUM`, `AVG`, `MIN`, and `MAX` work only on numeric fields, returning sum, average, minimum, and maximum values.
- Examples:

```
SELECT SUM(amount) AS total FROM payment;
SELECT AVG(amount) AS average FROM payment;
SELECT MIN(amount) AS min FROM payment;
SELECT MAX(amount) AS max FROM payment;
```

- Combine multiple metrics:

```
SELECT MIN(amount) AS min, AVG(amount) AS avg,
    MAX(amount) AS max FROM payment;
```

- Limit the search scope:

```
SELECT COUNT(*) AS n, MIN(amount) AS min,
    AVG(amount) AS avg, MAX(amount) AS max FROM
    payment WHERE customer_id = 148;
```

Aggregate Functions and Grouping

- When used with `GROUP BY`, aggregate functions calculate metrics for individual groups. Examples:

```
1 SELECT staff_id, COUNT(*) AS n FROM payment
    GROUP BY staff_id;
2 SELECT customer_id, COUNT(*) AS n FROM payment
    GROUP BY customer_id;
3 SELECT customer_id, SUM(amount) AS total FROM
    payment GROUP BY customer_id;
```

- Sort results, e.g., by "total":

```
1 SELECT customer_id, SUM(amount) AS total FROM
    payment GROUP BY customer_id ORDER BY total
    DESC;
```

- Filter with `WHERE` or `HAVING`:

```
1 SELECT replacement_cost, AVG(rental_rate) AS
    avg FROM film GROUP BY replacement_cost
    HAVING MIN(rental_rate) > 0;
```

Task 1 - Questions

1. Calculate the average length of movie titles based on all films in the database. Display the result in a column labeled `average`.
2. Determine the number of films available for rental in each store (count each title only once). Display results in two columns: `store` (store ID) and `titles` (number of films).
3. Identify the top 10 most popular actors (those appearing in the most films), sorted descending by film count. Use columns: `actor` (actor ID) and `n` (number of films).
4. List the age rating category, along with the minimum, average, and maximum replacement cost for each, sorted by average. Use columns: `category`, `min`, `average`, and `max`.
5. List the IDs of 10 countries represented by the fewest cities (but more than 10) in the database. Display two columns: `country` (country ID) and `n` (number of cities). Is Poland among them?

Task 1 - Solutions (1-3)

1. Average length of movie titles:

```
SELECT AVG(CHAR_LENGTH(title)) AS average
    FROM film;
```

2. Number of films per store:

```
SELECT store_id AS store, COUNT(DISTINCT
    film_id) AS titles FROM inventory GROUP
    BY store;
```

3. Top 10 most popular actors:

```
SELECT actor_id AS actor, COUNT(film_id) AS
    n FROM film_actor GROUP BY actor ORDER BY
    n DESC LIMIT 10;
```

Task 1 - Solutions (4-5)

4. Age rating category replacement costs:

```
SELECT rating AS category,
    MIN(replacement_cost) AS min,
    AVG(replacement_cost) AS average,
    MAX(replacement_cost) AS max FROM film
    GROUP BY category ORDER BY average;
```

5. Countries with fewest cities (¿ 10):

```
SELECT country_id AS country, COUNT(city) AS
    n FROM city GROUP BY country HAVING n >
    10 ORDER BY n LIMIT 10;
```

Note: Poland is not included (represented by 8 cities).

Mathematical Functions

- **Selected Mathematical Functions:**
  - `ABS`: Absolute value.
  - `ACOS`: Arccosine.
  - `ASIN`: Arcsine.
  - `ATAN`: Arctangent.
  - `COS`: Cosine.
  - `EXP`: Exponential function (e^x).
  - `LN`: Natural logarithm.
  - `LOG10`: Base-10 logarithm.
  - `LOG2`: Base-2 logarithm.
  - `PI`: π.
  - `POW` (or `POWER`): Power function.
  - `RAND`: Random number in (0,1).
  - `SIGN`: Sign of a number.
  - `SIN`: Sine.
  - `LOG(a, x)`: Logarithm base a (natural if one argument).
  - `SQRT`: Square root.
  - `TAN`: Tangent.

Mathematical Functions - Example

- Generate a pseudorandom number from $\{0, 1, \ldots, 10\}$:

```
1 SELECT FLOOR(RAND() * 11) AS random;
```

Statistical Functions

- SQL provides variance and standard deviation (population and sample):

```
SELECT VAR_POP(amount) AS sigma FROM payment;
SELECT VAR_SAMP(amount) AS sigma FROM payment;
SELECT STDDEV_POP(amount) AS sigma FROM payment;
SELECT STDDEV_SAMP(amount) AS sigma FROM
    payment;
```

- Additional functions:
  - `VARIANCE`: Equivalent to `VAR_POP`.
  - `STD`, `STDDEV`: Equivalent to `STDDEV_POP`.
- In MySQL, compute mode/median manually, e.g.:

```
SELECT amount AS mode FROM payment GROUP BY
    amount ORDER BY COUNT(amount) DESC LIMIT 1;
SELECT amount AS median FROM payment ORDER BY
    amount LIMIT 8024, 1;
```

Task 2 - Questions

1. Calculate the average and standard deviation of actors' last name lengths, rounded to 4 decimal places, labeled `average` and `sigma`.
2. What is the range of the number of film copies available in store ID 1? Compute the minimum and maximum separately in columns `n_min` and `n_max` (two queries).
3. Calculate the average and standard deviation of film lengths for each age rating category, rounded to 2 decimal places, in columns `category`, `average`, and `sigma`.
4. Determine the mode of rental days for films with a rental rate of 4.99.
5. Calculate the average and standard deviation of replacement costs per age rating category, for films lasting 90–150 minutes, in columns `category`, `average`, and `sigma`.

Task 2 - Solutions (1-2)

1. Average and standard deviation of actors' last names:

```
SELECT ROUND(AVG(CHAR_LENGTH(last_name)), 4)
    AS average,
    ROUND(STDDEV_POP(CHAR_LENGTH(last_name)),
    4) AS sigma FROM actor;
```

2. Range of film copies in store ID 1:

```
SELECT store_id, COUNT(film_id) AS n_min
    FROM inventory WHERE store_id = 1 GROUP
    BY film_id ORDER BY n_min LIMIT 1;
```

```
SELECT store_id, COUNT(film_id) AS n_max
    FROM inventory WHERE store_id = 1 GROUP
    BY film_id ORDER BY n_max DESC LIMIT 1;
```

Task 2 - Solutions (3-5)

3. Average and standard deviation of film lengths per category:

```
SELECT rating AS category,
    ROUND(AVG(length), 2) AS average,
    ROUND(STDDEV_SAMP(length), 2) AS sigma
    FROM film GROUP BY category;
```

4. Mode of rental days for films at 4.99:

```
SELECT rental_duration AS mode FROM film
    WHERE rental_rate = 4.99 GROUP BY mode
    ORDER BY COUNT(mode) DESC LIMIT 1;
```

5. Average and standard deviation of replacement costs (90–150 min):

```
SELECT rating AS category,
    AVG(replacement_cost) AS average,
STDDEV_SAMP(replacement_cost) AS sigma
FROM film WHERE length BETWEEN 90 AND 150
GROUP BY category;
```

Dates and Times in SQL

- SQL types for date/time: `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`.
- Each has a valid range and "zero value" for invalid entries.
- Key rules:
  - Proper formatting is critical—incorrect formats may cause errors.
  - Dates use 'YYYY-MM-DD' (2-digit years as 1970–2069).
  - Auto-converts date/time to numeric types when needed.
  - Allows 'YYYY-00-00' for unknown day/month.
  - Time stored with microsecond precision.

Date and Time Ranges in SQL

- **Valid Ranges:**
  - `DATE`: '1000-01-01' to '9999-12-31'.
  - `DATETIME`: '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'.
  - `TIMESTAMP` (UTC): '1970-01-01 00:00:01.000000' to '2038-01-19 03:14:07.999999'.
  - `TIME`: '-838:59:59.000000' to '838:59:59.000000'.
  - `YEAR`: '1901' to '2155'.
- Use full formats, e.g., '13:13' becomes '13:13:00', '1313' becomes '00:13:13'.

MySQL Date and Time Functions - 1

- **Selected Functions:**
  - `CURDATE()`: Current date.
  - `CURTIME()`: Current time.
  - `NOW()`: Current date and time.
  - `DATE(expr)`: Extracts the date.
  - `TIME(expr)`: Extracts the time.
  - `DAY(date)`: Day of the month.
  - `DAYNAME(date)`: Name of the day.
  - `DAYOFWEEK(date)`: Day of the week (1 = Sunday).
  - `DAYOFYEAR(date)`: Day of the year.
  - `WEEKOFYEAR(date)`: Week of the year.
  - `MONTH(date)`: Month number.
  - `MONTHNAME(date)`: Month name.
  - `QUARTER(date)`: Quarter.
  - `YEAR(date)`: Year.
  - `HOUR(time)`: Hour.
  - `MINUTE(time)`: Minute.
  - `SECOND(time)`: Second.

MySQL Date and Time Functions - 2

- **More Functions:**
  - `MAKEDATE(year, day)`: Creates a date from day of the year.
  - `DATEDIFF(d1, d2)`: Days between `d1` and `d2`.
  - `TIMEDIFF(d1, d2)`: Time difference in time format.
  - `TIMESTAMPDIFF(unit, d1, d2)`: Time difference in specified unit.
  - `ADDDATE(date, days)`: Shifts date forward.
  - `ADDTIME(time, interval)`: Shifts time forward.
  - `SUBDATE(date, days)`: Shifts date backward.
  - `SUBTIME(time, interval)`: Shifts time backward.
- Examples:

```
1 SELECT DAYOFYEAR(NOW()) AS day,
    WEEKOFYEAR(NOW()) AS week, QUARTER(NOW()) AS
    quarter;
2 SELECT ADDDATE(NOW(), 14) AS date;
```

Formatting Dates and Times

- For Polish day/month names, set:

```
SET lc_time_names = 'pl_PL';
SELECT DAYNAME(NOW()) AS day;
```

- Use DATE_FORMAT to customize:

```
SELECT DATE_FORMAT(NOW(), '%W, %e %M %Y') AS
    today;
SELECT DATE_FORMAT(NOW(), '%d.%m.%y,
    %H:%i:%s:%f') AS now;
```

Formatting Dates and Times - 2

- Format specifiers:
  - `%a`: Abbreviated weekday.
  - `%b`: Abbreviated month.
  - `%c`: Month (1-12).
  - `%d`: Day (00-31).
  - `%e`: Day (0-31).
  - `%f`: Microseconds.
  - `%H`: Hour (00-23).
  - `%i`: Minutes.
  - `%M`: Month name.
  - `%m`: Month (01-12).
  - `%s`: Seconds.
  - `%W`: Weekday name.
  - `%Y`: Year (4 digits).
  - `%y`: Year (2 digits).

User-defined Functions

```
DELIMITER //

CREATE FUNCTION f(x DOUBLE)
RETURNS DOUBLE
DETERMINISTIC
BEGIN
    RETURN x * x - 4 * x;
END;
//

DELIMITER ;
```

1. DELIMITER // is used so the semicolons inside the function don't end the command prematurely.
2. DETERMINISTIC tells MySQL the function always returns the same output for the same input, which helps with optimization.

```
SELECT f(5);   -- returns 5
```

Task 3 - Questions

1. How many days did the shortest, average, and longest film rentals last? Display in columns `min`, `average`, and `max`.
2. Which day of the week had the most unreturned film rentals? Display the day name in `day` and count in `n`.
3. List all payments made on Wednesdays handled by staff ID 2, sorted ascending by payment date.
4. List total revenue by day of the week, in columns `day` (Sunday to Saturday) and `revenue`.
5. List revenue by staff ID and month, in columns `id` (staff ID), `month` (month name), and `revenue`, sorted by ID and month order.

Task 3 - Solutions (1-2)

1. Shortest, average, and longest rental durations:

   (Solution requires rental table data; example query:)

```
SELECT MIN(DATEDIFF(return_date,
    rental_date)) AS min,
AVG(DATEDIFF(return_date, rental_date)) AS
    average,
MAX(DATEDIFF(return_date, rental_date)) AS
    max
FROM rental WHERE return_date IS NOT NULL;
```

2. Day with most unreturned rentals:

```
SELECT DAYNAME(rental_date) AS day, COUNT(*)
    AS n FROM rental WHERE return_date IS
    NULL GROUP BY day ORDER BY n DESC LIMIT 1;
```

Task 3 - Solutions (3-4)

3. Payments on Wednesdays by staff ID 2:

```
SELECT * FROM payment WHERE
    DAYNAME(payment_date) = 'Wednesday' AND
    staff_id = 2
ORDER BY payment_date ASC;
```

4. Revenue by day of the week:

```
SELECT DAYNAME(payment_date) AS day,
    SUM(amount) AS revenue FROM payment
GROUP BY day
ORDER BY FIELD(day, 'Sunday', 'Monday',
    'Tuesday', 'Wednesday', 'Thursday',
    'Friday', 'Saturday');
```

Task 3 - Solutions (5)

1. Revenue by staff and month:

```sql
SELECT staff_id AS id,
    MONTHNAME(payment_date) AS month,
    SUM(amount) AS revenue
FROM payment
GROUP BY id, month
ORDER BY id, MONTH(payment_date);
```