15 UDFs

UDFs

-> Set of SQL statements that perform a specific task.
-> It takes input params, performs action and returns either a single value or a table.
-> Functions can't be used for DML (INSERT, UPDATE, DELETE)
-> We could create SP to group a set of SQL statements, but SPs can't be called from within the SQL statements.

Built-In Functions provided by DBMS`

-> Functions must return a value, for SPs it is optional
-> Functions can be called from procedures
-> Functions allow only SELECT but SPs allow DML statements
-> Function can be used in SELECT, WHERE, Having, ORDER BY, Group BY clauses.

;

```SQL
DELIMITER //

CREATE FUNCTION discount_v3(product VARCHAR(1000), sale_price INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    IF LEFT(product, 1) = 'A' THEN
        RETURN 0.5*sale_price; -- 50% off
    ELSEIF LEFT(product, 1) = 'K' THEN
        RETURN 0; -- 100% off (free)
    ELSE
        RETURN sale_price; -- No discount
    END IF;
END //

DELIMITER ;



select product, sale_price, discount_v3(product, sale_price) from bb_products order by product;
```

```sql
CREATE FUNCTION calculate_experience(join_date DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    RETURN TIMESTAMPDIFF(YEAR, join_date, CURDATE());
END;
```

Usage:

```sql
SELECT
    Programmer_Name,
    DOJ,
    calculate_experience(DOJ) AS Years_Experience
FROM programmers;
```

MySQL User-Defined Functions (UDFs): Comprehensive Notes

Introduction to User-Defined Functions

User-Defined Functions (UDFs) in MySQL allow developers to create their own custom functions to extend MySQL's functionality. Unlike built-in functions, UDFs are created by database users to meet specific requirements and can be reused across multiple queries.

Types of MySQL UDFs

1. Scalar Functions

Return a single value based on the input parameters.

```sql
CREATE FUNCTION calculate_experience(join_date DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    RETURN TIMESTAMPDIFF(YEAR, join_date, CURDATE());
END;
```

Usage:

```sql
SELECT
    Programmer_Name,
    DOJ,
    calculate_experience(DOJ) AS Years_Experience
FROM programmers;
```

2. Table Functions (Requires MySQL 8.0+)

Return a table result set, similar to a table or view.

```sql
CREATE FUNCTION get_language_programmers(lang VARCHAR(100))
RETURNS TABLE (
    programmer VARCHAR(100),
    experience_years INT
)
READS SQL DATA
BEGIN
    RETURN TABLE(
        SELECT
            Programmer_Name,
            TIMESTAMPDIFF(YEAR, DOJ, CURDATE()) AS experience_years
        FROM programmers
        WHERE Primary_Language = lang OR Secondary_Language = lang
    );
END;
```

Usage:

```sql
SELECT * FROM get_language_programmers('Python');
```

Creating UDFs: Syntax and Options

Basic syntax:

```sql
CREATE FUNCTION function_name([parameter_list])
RETURNS return_datatype
[characteristic ...]
BEGIN
    -- Function body
    RETURN value;
END;
```

Function Characteristics:

- **DETERMINISTIC**: Always returns the same result for the same input
- **NOT DETERMINISTIC**: Result may vary for the same input (default)
- **CONTAINS SQL**: Contains SQL statements but doesn't read/write data
- **NO SQL**: Contains no SQL statements
- **READS SQL DATA**: Reads data but doesn't modify
- **MODIFIES SQL DATA**: May modify data (insert/update/delete)

Example: Salary Tier Function

```sql
CREATE FUNCTION get_salary_tier(salary DECIMAL(10,2))
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE tier VARCHAR(20);

    IF salary ≥ 16000 THEN
        SET tier = 'Senior';
    ELSEIF salary ≥ 14000 THEN
        SET tier = 'Mid-Level';
    ELSE
        SET tier = 'Junior';
    END IF;

    RETURN tier;
END;
```

Usage:

```sql
SELECT
    Programmer_Name,
    Salary,
    get_salary_tier(Salary) AS Salary_Tier
FROM programmers
ORDER BY Salary DESC;
```

Variables, Control Flow, and Logic in UDFs

Local Variables

```sql
CREATE FUNCTION calculate_software_profit(cost DECIMAL(10,2), dev_cost
DECIMAL(10,2), sold INT)
RETURNS DECIMAL(10,2) DETERMINISTIC
BEGIN
    DECLARE revenue DECIMAL(10,2);
    DECLARE profit DECIMAL(10,2);

    SET revenue = cost * sold;
    SET profit = revenue - dev_cost;

    RETURN profit;
END;
```

Conditional Logic

```sql
CREATE FUNCTION get_experience_level(join_date DATE)
RETURNS VARCHAR(50) DETERMINISTIC
BEGIN
    DECLARE years INT;
    DECLARE level VARCHAR(50);

    SET years = TIMESTAMPDIFF(YEAR, join_date, CURDATE());

    CASE
        WHEN years ≥ 20 THEN SET level = 'Veteran';
        WHEN years ≥ 10 THEN SET level = 'Experienced';
        WHEN years ≥ 5 THEN SET level = 'Mid-Level';
        ELSE SET level = 'Beginner';
    END CASE;

    RETURN level;
END;
```

Loop Structures

```SQL
CREATE FUNCTION factorial(n INT)
RETURNS BIGINT DETERMINISTIC
BEGIN
    DECLARE result BIGINT DEFAULT 1;
    DECLARE i INT DEFAULT 1;

    WHILE i ≤ n DO
        SET result = result * i;
        SET i = i + 1;
    END WHILE;

    RETURN result;
END;
```

Real-World Examples Using Our Dev Database

1. ROI Calculator Function

```SQL
CREATE FUNCTION calculate_roi(revenue DECIMAL(10,2), cost DECIMAL(10,2))
RETURNS DECIMAL(10,2) DETERMINISTIC
BEGIN
    IF cost = 0 THEN
        RETURN 0;
    END IF;

    RETURN ((revenue - cost) / cost) * 100;
END;
```

Usage:

```SQL
SELECT
    Programmer_Name,
    Software_Name,
    Software_Cost * Sold AS Total_Revenue,
    Development_Cost,
    calculate_roi(Software_Cost * Sold, Development_Cost) AS ROI_Percentage
FROM software
ORDER BY ROI_Percentage DESC;
```

2. Age Calculator Function

```SQL
CREATE FUNCTION calculate_age(birth_date DATE)
RETURNS INT DETERMINISTIC
BEGIN
    RETURN TIMESTAMPDIFF(YEAR, birth_date, CURDATE());
END;
```

Usage:

```SQL
SELECT
    Programmer_Name,
    DOB,
    calculate_age(DOB) AS Age
FROM programmers
ORDER BY Age;
```

3. Software Profitability Status Function

```SQL
CREATE FUNCTION get_profitability_status(revenue DECIMAL(10,2), cost DECIMAL(10,2))
RETURNS VARCHAR(20) DETERMINISTIC
BEGIN
    DECLARE profit_margin DECIMAL(10,2);

    IF revenue = 0 THEN
        RETURN 'No Sales';
    END IF;

    SET profit_margin = ((revenue - cost) / revenue) * 100;

    CASE
        WHEN profit_margin ≥ 50 THEN RETURN 'Highly Profitable';
        WHEN profit_margin ≥ 20 THEN RETURN 'Profitable';
        WHEN profit_margin ≥ 0 THEN RETURN 'Break Even';
        ELSE RETURN 'Loss Making';
    END CASE;
END;
```

Usage:

```sql
SELECT
    Programmer_Name,
    Software_Name,
    Software_Cost * Sold AS Revenue,
    Development_Cost AS Cost,
    get_profitability_status(Software_Cost * Sold, Development_Cost) AS Status
FROM software
ORDER BY Status;
```

4. Language Expertise Level Function

```sql
CREATE FUNCTION get_language_expertise(primary_lang VARCHAR(100), secondary_lang
VARCHAR(100), experience_years INT)
RETURNS VARCHAR(50) DETERMINISTIC
BEGIN
    DECLARE expertise VARCHAR(50);

    IF experience_years ≥ 15 THEN
        RETURN CONCAT('Expert in ', primary_lang, ' and ', secondary_lang);
    ELSEIF experience_years ≥ 8 THEN
        RETURN CONCAT('Advanced ', primary_lang, ' Developer');
    ELSEIF experience_years ≥ 3 THEN
        RETURN CONCAT('Intermediate ', primary_lang, ' Developer');
    ELSE
        RETURN CONCAT('Beginning ', primary_lang, ' Developer');
    END IF;
END;
```

Usage:

```sql
SELECT
    Programmer_Name,
    Primary_Language,
    Secondary_Language,
    TIMESTAMPDIFF(YEAR, DOJ, CURDATE()) AS Experience,
    get_language_expertise(Primary_Language, Secondary_Language,
TIMESTAMPDIFF(YEAR, DOJ, CURDATE())) AS Expertise_Level
FROM programmers
ORDER BY Experience DESC;
```

Managing UDFs

Viewing Function Definitions

```sql
SHOW FUNCTION STATUS WHERE Db = 'devs';
SHOW CREATE FUNCTION calculate_experience;
```

Modifying Functions

```sql
DROP FUNCTION IF EXISTS calculate_experience;
-- Then recreate with new definition
```

Getting Functions from Information Schema

```sql
SELECT
    routine_name,
    routine_definition,
    created,
    last_altered
FROM information_schema.routines
WHERE routine_schema = 'devs' AND routine_type = 'FUNCTION';
```

Advanced UDF Techniques

1. UDFs with String Manipulation

```SQL
CREATE FUNCTION extract_language_skill(programmer_name VARCHAR(100))
RETURNS VARCHAR(200) READS SQL DATA
BEGIN
    DECLARE primary_lang VARCHAR(100);
    DECLARE secondary_lang VARCHAR(100);

    SELECT
        Primary_Language,
        Secondary_Language
    INTO
        primary_lang,
        secondary_lang
    FROM programmers
    WHERE Programmer_Name = programmer_name;

    IF primary_lang IS NULL THEN
        RETURN 'Programmer not found';
    END IF;

    RETURN CONCAT('Proficient in ', primary_lang, ' with ', secondary_lang, '
experience');
END;
```

2. UDFs with Date Calculations

```SQL
CREATE FUNCTION years_until_retirement(dob DATE, retirement_age INT)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE current_age INT;
    DECLARE years_left INT;

    SET current_age = TIMESTAMPDIFF(YEAR, dob, CURDATE());
    SET years_left = retirement_age - current_age;

    IF years_left < 0 THEN
        RETURN 0;
    ELSE
        RETURN years_left;
    END IF;
END;
```

Usage:

```sql
SELECT
    Programmer_Name,
    DOB,
    calculate_age(DOB) AS Current_Age,
    years_until_retirement(DOB, 65) AS Years_To_Retirement
FROM programmers
ORDER BY Years_To_Retirement;
```

3. Using Recursive Functions

```sql
CREATE FUNCTION fibonacci(n INT)
RETURNS BIGINT DETERMINISTIC
BEGIN
    IF n ≤ 1 THEN
        RETURN n;
    ELSE
        RETURN fibonacci(n-1) + fibonacci(n-2);
    END IF;
END;
```

Note: This implementation is inefficient for large values of n. In practice, you'd use iterative algorithms for performance.

Differences Between UDFs and Stored Procedures

Feature	UDFs	Stored Procedures
Return Value	Must return a value	Return value optional
Usage in SQL	Can be used in SELECT, WHERE, etc.	Cannot be part of SQL expressions
DML Operations	Cannot perform DML operations	Can perform INSERT, UPDATE, DELETE
Transaction Control	Cannot use transaction statements	Can use COMMIT, ROLLBACK
Error Handling	Limited error handling	Can include extensive error handling
OUT Parameters	No OUT or INOUT parameters	Supports IN, OUT, and INOUT parameters
Calling	Called as part of SQL expression	Called with CALL statement

Security Considerations

- UDFs require the CREATE ROUTINE privilege to create
- EXECUTE privilege is needed to run functions
- Functions run with the privileges of the creator (definer) by default
- Security concerns around SQL injection must be addressed

```sql
-- Create a function with definer security
CREATE DEFINER = 'admin'@'localhost' FUNCTION safe_get_salary(name VARCHAR(100))
RETURNS DECIMAL(10,2) READS SQL DATA SQL SECURITY DEFINER
BEGIN
    DECLARE salary_value DECIMAL(10,2);

    SELECT Salary INTO salary_value
    FROM programmers
    WHERE Programmer_Name = name;

    RETURN COALESCE(salary_value, 0);
END;
```

Best Practices for UDFs

1. **Name functions clearly**: Use descriptive names that indicate action
2. **Comment your functions**: Explain complex logic
3. **Use appropriate characteristics**: Mark DETERMINISTIC functions correctly for optimization
4. **Keep functions focused**: Each function should do one thing well
5. **Input validation**: Check parameters before processing
6. **Error handling**: Return meaningful values for error conditions
7. **Performance considerations**: Avoid expensive operations in frequently called functions
8. **Avoid overuse**: Don't create functions for simple operations already covered by built-in functions

Performance Considerations

- UDFs can be slower than equivalent inline SQL
- Each function call has overhead
- UDFs with DETERMINISTIC characteristic can be better optimized
- UDFs called for each row in a large result set can impact performance
- Functions reading data from tables can prevent index usage in some cases

Summary

User-Defined Functions in MySQL provide:

- A way to encapsulate and reuse complex logic
- Standardization of calculations across applications
- Abstraction of business rules
- Enhanced query capabilities
- Modular, maintainable code

By mastering UDFs, developers can write more efficient, readable, and maintainable SQL code while extending MySQL's capabilities to meet specific business requirements.