MySQL HTTP POST Plugin Documentation

Overview

This MySQL plugin provides a User Defined Function (UDF) called `http_post` that enables HTTP POST requests directly from MySQL queries, stored procedures, and triggers. This is particularly useful for implementing webhooks and real-time notifications from database events.

Prerequisites

- MySQL development headers
- libcurl development library
- g++ compiler
- MySQL server with plugin support enabled

Install Dependencies

On Ubuntu/Debian:

```bash
sudo apt-get install libmysqlclient-dev libcurl4-openssl-dev g++ mysql-server
```

On CentOS/RHEL:

```bash
sudo yum install mysql-devel libcurl-devel gcc-c++ mysql-server
```

MySQL Version Compatibility

This plugin is compatible with:

- MySQL 5.7 and earlier
- MySQL 8.0 and later
- MariaDB 10.x and later

The plugin automatically handles the `my_bool` to `bool` type change in MySQL 8.0.1+.

Compilation

1. Save the C++ code as `http_post.cpp`

2. Compile the plugin:

```bash
g++ -shared -fPIC -o http_post.so http_post.cpp -I/usr/include/mysql -lcurl
```

3. Copy the compiled plugin to MySQL plugin directory:

```bash
# Find MySQL plugin directory
mysql -e "SHOW VARIABLES LIKE 'plugin_dir';"

# Copy the plugin (adjust path as needed)
sudo cp http_post.so /usr/lib/mysql/plugin/
```

Installation in MySQL

1. Connect to MySQL as root:

```bash
mysql -u root -p
```

2. Create the function:

```sql
CREATE FUNCTION http_post RETURNS STRING SONAME 'http_post.so';
```

3. Verify installation:

```sql
SELECT * FROM mysql.func WHERE name = 'http_post';
```

Usage

Basic Syntax

```sql
SELECT http_post(url, data, [headers]);
```

Parameters:

- `url` (required): The webhook URL
- `data` (required): The POST data (typically JSON)
- `headers` (optional): HTTP headers separated by semicolons

Examples

1. Simple JSON POST

```sql
SELECT http_post(
    'https://webhook.site/your-webhook-id',
    '{"event": "user_created", "user_id": 123}'
);
```

2. POST with Headers

```sql
SELECT http_post(
    'https://api.example.com/webhook',
    '{"event": "order_completed", "order_id": 456}',
    'Content-Type: application/json;Authorization: Bearer your-token'
);
```

3. In a Stored Procedure

sql

```sql
DELIMITER $$

CREATE PROCEDURE notify_user_creation(IN user_id INT, IN user_email VARCHAR(255))
BEGIN
    DECLARE webhook_response TEXT;

    SET webhook_response = http_post(
        'https://api.example.com/webhooks/user-created',
        CONCAT('{"user_id": ', user_id, ', "email": "', user_email, '"}'),
        'Content-Type: application/json'
    );

    -- Log the response
    INSERT INTO webhook_logs (endpoint, response, created_at)
    VALUES ('user-created', webhook_response, NOW());
END$$

DELIMITER ;
```

4. In a Trigger

sql

```sql
DELIMITER $$

CREATE TRIGGER after_order_insert
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    DECLARE webhook_data TEXT;
    DECLARE webhook_response TEXT;

    SET webhook_data = JSON_OBJECT(
        'event', 'new_order',
        'order_id', NEW.id,
        'customer_id', NEW.customer_id,
        'total_amount', NEW.total_amount,
        'created_at', NEW.created_at
    );

    SET webhook_response = http_post(
        'https://api.example.com/webhooks/orders',
        webhook_data,
        'Content-Type: application/json;X-Webhook-Secret: your-secret'
    );
END$$

DELIMITER ;
```

5. Error Handling

sql

```sql
DELIMITER $$

CREATE FUNCTION safe_webhook_call(url TEXT, data TEXT)
RETURNS VARCHAR(255)
DETERMINISTIC
BEGIN
    DECLARE response TEXT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        RETURN 'Error: Webhook call failed';
    END;

    SET response = http_post(url, data, 'Content-Type: application/json');

    IF response LIKE 'CURL error:%' THEN
        RETURN CONCAT('Failed: ', response);
    ELSE
        RETURN 'Success';
    END IF;
END$$

DELIMITER ;
```

Security Considerations

1. **Permissions**: Restrict access to the `http_post` function:

sql

```sql
-- Grant permission to specific users
GRANT EXECUTE ON FUNCTION your_database.http_post TO 'webhook_user'@'localhost';
```

2. **URL Validation**: Consider creating a wrapper function that validates URLs:

```sql
CREATE FUNCTION safe_http_post(url TEXT, data TEXT)
RETURNS TEXT
DETERMINISTIC
BEGIN
    -- Only allow specific domains
    IF url NOT LIKE 'https://api.yourdomain.com/%' THEN
        RETURN 'Error: Unauthorized URL';
    END IF;

    RETURN http_post(url, data, 'Content-Type: application/json');
END;
```

3. **Rate Limiting**: Implement rate limiting in your procedures to prevent abuse.
4. **SSL/TLS**: Always use HTTPS URLs for secure communication.

Troubleshooting

Common Issues

1. **Function not found error**:
   - Check if the plugin is in the correct directory
   - Verify MySQL has permission to read the .so file
   - Check MySQL error log for loading issues
2. **Compilation errors**:
   - Ensure all dependencies are installed
   - Check MySQL include path matches your installation
   - For MySQL 8.0+, the plugin handles `my_bool` deprecation automatically
3. **CURL errors**:
   - Verify the URL is accessible from the MySQL server
   - Check firewall rules
   - Test timeout values for slow endpoints
4. **MySQL 8.0+ specific issues**:
   - If you get `my_bool` errors, ensure you're using the latest version of the plugin
   - The plugin includes compatibility code for MySQL 8.0.1+ where `my_bool` was replaced with `bool`

Debugging

Enable MySQL general query log to see function calls:

```sql
SET GLOBAL general_log = 'ON';
SET GLOBAL general_log_file = '/var/log/mysql/general.log';
```

Uninstallation

To remove the plugin:

```sql
DROP FUNCTION IF EXISTS http_post;
```

Then remove the .so file from the plugin directory.

Performance Considerations

- HTTP calls are blocking and can slow down queries
- Consider using asynchronous patterns with a queue table
- Set appropriate timeouts (default is 30 seconds)
- Monitor webhook response times

Alternative Async Pattern

For better performance, consider this pattern:

1. Create a webhook queue table:

```sql
CREATE TABLE webhook_queue (
    id INT AUTO_INCREMENT PRIMARY KEY,
    url VARCHAR(255),
    data TEXT,
    headers TEXT,
    status ENUM('pending', 'sent', 'failed') DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed_at TIMESTAMP NULL
);
```

2. Insert into queue instead of direct calls:

```sql
INSERT INTO webhook_queue (url, data, headers)
VALUES ('https://api.example.com/webhook', '{"event": "test"}', 'Content-Type: application/jsor
```

3. Process queue with a scheduled event or external worker.

This approach prevents blocking and provides better reliability with retry capabilities.