19 DCL

MySQL Data Control Language (DCL)

Introduction to Data Control Language (DCL)

Data Control Language (DCL) is a subset of SQL that deals with controlling access to data stored in a database. Unlike DDL (Data Definition Language) which defines database structure, or DML (Data Manipulation Language) which manipulates data, DCL focuses specifically on who can access what data and what operations they can perform.

The two primary DCL commands in MySQL are:

- `GRANT` - gives users access privileges to the database
- `REVOKE` - withdraws access privileges given with the GRANT command

Why DCL Matters

Database security is crucial in real-world applications. Consider these scenarios:

1. A financial application where only certain employees should access customer account information
2. A content management system where editors can update content but not delete it
3. A hospital database where different departments need different levels of access to patient data

DCL commands enable precise control over user permissions based on application requirements.

MySQL User Management

Creating Users

Before granting permissions, we need to create users who will access the database.

```
-- Create a new user with a password
CREATE USER 'web_app'@'localhost' IDENTIFIED BY 'secure_password123';

-- Create a user that can connect from any host
CREATE USER 'reports_user'@'%' IDENTIFIED BY 'reporting_pw456';
```

In a real e-commerce application, you might create different users for different components:

- `inventory_manager` - for inventory management module
- `order_processor` - for order processing system
- `analytics_user` - for reporting and analytics

Viewing Existing Users

To see what users exist in your MySQL instance:

```
-- Show all users in the MySQL user table
SELECT user, host FROM mysql.user;
```

The GRANT Command

The GRANT command assigns specific privileges to users.

Basic Syntax

```
GRANT privilege_type [(column_list)]
ON database_name.table_name
```

```
TO 'username'@'host';
```

Common Privileges

MySQL offers many privilege types, including:

- `ALL PRIVILEGES` - grants all privileges
- `SELECT` - allows reading data
- `INSERT` - allows adding new rows
- `UPDATE` - allows modifying existing rows
- `DELETE` - allows removing rows
- `CREATE` - allows creating new tables or databases
- `DROP` - allows deleting tables or databases
- `ALTER` - allows modifying table structure
- `INDEX` - allows creating or removing indexes
- `EXECUTE` - allows executing stored procedures

Real-World Examples

Example 1: Basic Read-Only Access

For a reporting dashboard that only needs to view product data:

```
-- Grant read-only access to the products table
GRANT SELECT ON ecommerce.products TO 'reports_user'@'localhost';

-- Allow the user to see all orders but not customer emails
GRANT SELECT (order_id, order_date, total_amount) ON ecommerce.orders TO
'reports_user'@'localhost';
```

Example 2: Application User Access

For a web application that manages a blog:

```
-- Blog administrator with full access
GRANT ALL PRIVILEGES ON blog_db.* TO 'blog_admin'@'localhost';

-- Blog author can add and edit posts but not delete them
GRANT SELECT, INSERT, UPDATE ON blog_db.posts TO 'blog_author'@'localhost';
GRANT SELECT, INSERT ON blog_db.comments TO 'blog_author'@'localhost';
```

```
Show grants;
Show grants on user@host;
```

Example 3: Granting Multiple Privileges

For an inventory management system:

```
-- Inventory manager can view, add, and update products
GRANT SELECT, INSERT, UPDATE ON inventory_db.products TO 'inventory_user'@'localhost';

-- Also allow access to the stock_levels table
GRANT SELECT, UPDATE ON inventory_db.stock_levels TO 'inventory_user'@'localhost';
```

Example 4: Database-Level Access

For a database administrator who needs full access to a specific database:

```
-- Grant all privileges on all tables in customer_db
GRANT ALL PRIVILEGES ON customer_db.* TO 'db_admin'@'localhost';
```

Example 5: Column-Level Access

For enhanced security in an HR application:

```
-- Allow HR assistant to see employee info but not salary data
GRANT SELECT (employee_id, first_name, last_name, department, position)
ON hr_db.employees TO 'hr_assistant'@'localhost';

-- Allow HR manager to see and update all employee data, including salary
GRANT SELECT, UPDATE ON hr_db.employees TO 'hr_manager'@'localhost';
```

Making Privileges Take Effect

After granting privileges, you need to refresh the privilege tables:

```
FLUSH PRIVILEGES;
```

The REVOKE Command

REVOKE removes previously granted privileges.

Basic Syntax

```
REVOKE privilege_type [(column_list)]
ON database_name.table_name
FROM 'username'@'host';
```

Real-World Examples

Example 1: Revoking Specific Privileges

When an employee changes roles:

```
-- Remove the ability to delete orders
REVOKE DELETE ON store_db.orders FROM 'sales_user'@'localhost';
```

Example 2: Revoking All Privileges

When a contractor's project is complete:

```
-- Remove all privileges from project_db
REVOKE ALL PRIVILEGES ON project_db.* FROM 'contractor'@'localhost';
```

Example 3: Revoking Database-Level Access

```
-- Remove all privileges on a specific database
REVOKE ALL PRIVILEGES ON archived_data.* FROM 'analyst'@'localhost';
```

Role-Based Access Control (MySQL 8.0+)

MySQL 8.0 introduced role-based access control, allowing administrators to group privileges into roles that can be assigned to users.

Creating and Managing Roles

```
-- Create roles
CREATE ROLE 'app_read', 'app_write', 'app_admin';

-- Grant privileges to roles
GRANT SELECT ON app_db.* TO 'app_read';
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write';
GRANT ALL PRIVILEGES ON app_db.* TO 'app_admin';

-- Assign roles to users
GRANT 'app_read' TO 'reporting_user'@'localhost';
GRANT 'app_read', 'app_write' TO 'content_editor'@'localhost';
GRANT 'app_admin' TO 'site_administrator'@'localhost';
```

Activating Roles

Users need to activate assigned roles:

```
-- Set default role
SET DEFAULT ROLE 'app_read' TO 'reporting_user'@'localhost';

-- User can switch roles during session if they have multiple
SET ROLE 'app_write';
```

Real-World Role Example

For an e-commerce platform with different user types:

```
-- Create roles for different functions
CREATE ROLE 'store_manager', 'inventory_control', 'customer_service', 'analytics';

-- Grant appropriate privileges to each role
GRANT SELECT, INSERT, UPDATE, DELETE ON store_db.products TO 'store_manager';
GRANT SELECT, INSERT, UPDATE, DELETE ON store_db.orders TO 'store_manager';

GRANT SELECT, UPDATE ON store_db.products TO 'inventory_control';
GRANT SELECT, UPDATE ON store_db.inventory TO 'inventory_control';

GRANT SELECT ON store_db.customers TO 'customer_service';
GRANT SELECT, UPDATE ON store_db.orders TO 'customer_service';

GRANT SELECT ON store_db.* TO 'analytics';

-- Assign roles to users
GRANT 'store_manager' TO 'maria'@'localhost';
GRANT 'inventory_control' TO 'john'@'localhost';
GRANT 'customer_service' TO 'sarah'@'localhost';
GRANT 'analytics' TO 'data_team'@'%';
```

Practical DCL Security Patterns

Least Privilege Principle

Always grant the minimum necessary privileges:

```
-- Instead of granting ALL PRIVILEGES
-- Specifically grant only what's needed
GRANT SELECT, INSERT, UPDATE ON customer_db.accounts TO 'app_user'@'localhost';
```

Using Database Users vs. Application Users

Create different database users for different components of your application:

```
-- Create separate users for different app components
CREATE USER 'auth_service'@'localhost' IDENTIFIED BY 'auth_pw123';
CREATE USER 'profile_service'@'localhost' IDENTIFIED BY 'profile_pw456';
CREATE USER 'order_service'@'localhost' IDENTIFIED BY 'order_pw789';

-- Grant specific privileges to each service
GRANT SELECT, UPDATE ON users_db.credentials TO 'auth_service'@'localhost';
GRANT SELECT, UPDATE ON users_db.profiles TO 'profile_service'@'localhost';
GRANT SELECT, INSERT, UPDATE ON orders_db.orders TO 'order_service'@'localhost';
```

Regular Security Audits

Regularly review who has access to what:

```
-- Show privileges for a specific user
SHOW GRANTS FOR 'app_user'@'localhost';

-- Find all users with global privileges
SELECT * FROM mysql.user WHERE Grant_priv = 'Y';
```

Common DCL Issues and Solutions

Issue: Too Broad Privileges

```
-- Problem: This grants too much access
GRANT ALL PRIVILEGES ON *.* TO 'web_app'@'localhost';

-- Better approach: Grant specific privileges on specific tables
GRANT SELECT, INSERT, UPDATE ON app_db.users TO 'web_app'@'localhost';
GRANT SELECT, INSERT ON app_db.logs TO 'web_app'@'localhost';
```

Issue: Hardcoded Database Credentials

```
// Problem in PHP code: Hardcoded credentials
$conn = new mysqli("localhost", "root", "password", "app_db");

// Better: Use environment variables
$conn = new mysqli(
    getenv("DB_HOST"),
    getenv("DB_USER"),
    getenv("DB_PASSWORD"),
    getenv("DB_NAME")
);
```

Issue: Not Revoking Access

Always revoke access when no longer needed:

```
-- When an employee leaves
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'former_employee'@'localhost';
DROP USER 'former_employee'@'localhost';
```

Advanced DCL Topics

Resource Limits

Prevent users from overloading the server:

```
-- Limit number of queries per hour
CREATE USER 'api_user'@'%' IDENTIFIED BY 'password'
WITH MAX_QUERIES_PER_HOUR 1000;

-- Set multiple limits
ALTER USER 'reports_user'@'localhost'
WITH MAX_CONNECTIONS_PER_HOUR 20
MAX_USER_CONNECTIONS 5;
```

Password Policies

Enforce password security:

```
-- Set password expiration
ALTER USER 'financial_user'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;

-- Require strong passwords (MySQL 8.0+)
CREATE USER 'new_user'@'localhost' IDENTIFIED BY 'Str0ng_P@ssw0rd'
PASSWORD HISTORY 5 -- Can't reuse last 5 passwords
PASSWORD REUSE INTERVAL 365 DAY; -- Can't reuse password within a year
```

Putting It All Together: DCL for a Complete Web Application

Let's create a comprehensive security setup for a fictional e-commerce application:

```
-- Create database
CREATE DATABASE ecommerce;
USE ecommerce;

-- Create application users with specific purposes
CREATE USER 'ecomm_admin'@'localhost' IDENTIFIED BY 'admin_pw123';


CREATE USER 'customer_app'@'%' IDENTIFIED BY 'app_pw456'
    WITH MAX_CONNECTIONS_PER_HOUR 1000;


CREATE USER 'reporting'@'localhost' IDENTIFIED BY 'report_pw789'
    WITH MAX_USER_CONNECTIONS 5;
CREATE USER 'backup_service'@'backup-server.internal' IDENTIFIED BY 'backup_pw101';

-- Create roles (MySQL 8.0+)
CREATE ROLE 'admin_role', 'app_role', 'reporting_role', 'backup_role';

-- Admin role - full access
GRANT ALL PRIVILEGES ON ecommerce.* TO 'admin_role';
```

```
-- App role - specific permissions needed for the application
GRANT SELECT, INSERT, UPDATE ON ecommerce.products TO 'app_role';
GRANT SELECT, INSERT ON ecommerce.customers TO 'app_role';
GRANT SELECT, INSERT, UPDATE ON ecommerce.orders TO 'app_role';
GRANT SELECT, INSERT ON ecommerce.order_items TO 'app_role';
GRANT SELECT ON ecommerce.inventory TO 'app_role';

-- Reporting role - read-only access
GRANT SELECT ON ecommerce.* TO 'reporting_role';

-- Backup role - needed for database backups
GRANT SELECT, LOCK TABLES, SHOW VIEW, TRIGGER, EVENT
    ON ecommerce.* TO 'backup_role';

-- Assign roles to users
GRANT 'admin_role' TO 'ecomm_admin'@'localhost';
GRANT 'app_role' TO 'customer_app'@'%';
GRANT 'reporting_role' TO 'reporting'@'localhost';
GRANT 'backup_role' TO 'backup_service'@'backup-server.internal';

-- Set default roles for users
SET DEFAULT ROLE 'admin_role' TO 'ecomm_admin'@'localhost';
SET DEFAULT ROLE 'app_role' TO 'customer_app'@'%';
SET DEFAULT ROLE 'reporting_role' TO 'reporting'@'localhost';
SET DEFAULT ROLE 'backup_role' TO 'backup_service'@'backup-server.internal';

-- Apply changes
FLUSH PRIVILEGES;
```

Best Practices Summary

1. **Follow the Principle of Least Privilege**: Grant only the permissions necessary for each user to perform their function.
2. **Use Role-Based Access Control**: Define roles based on job functions and assign privileges to roles rather than directly to users.
3. **Regularly Audit Permissions**: Review who has access to what data on a regular schedule.
4. **Revoke Unused Access**: Promptly remove access for users who no longer need it.
5. **Implement Password Policies**: Set password expiration and complexity requirements.
6. **Avoid Using Root**: Never use the root account for application access.
7. **Set Resource Limits**: Prevent any single user from monopolizing database resources.
8. **Use SSL/TLS**: Require encrypted connections for sensitive data.
9. **Store Credentials Securely**: Never hardcode database credentials in application code.
10. **Document Access Control**: Maintain documentation of who has what access and why.

By implementing these DCL practices, you'll create a secure foundation for your MySQL databases in real-world applications.