Computer Science and Interdisciplinary Research Journal
Volume 2, Issue 1, ISSN: 3033-1218, DOI: 10.70862/CSIR.2025.0201-10, 2025

A Flexible Approach to MySQL Backup Creation – flexmysqlbackup

Dimitar Todorov

University of Plovdiv Paisii Hilendarski, 236, Bulgaria Blvd, 4027 Plovdiv, Bulgaria

dimitar.todorov@uni-plovdiv.bg

Abstract – Creating backups in MySQL is an integral part of database management, particularly crucial in environments with large data volumes and critical information. Data recovery in the event of a system crash, data loss, or migration requirements is made possible by timely and accurate backups. MySQL is one of the most popular databases, utilized by numerous companies and projects. Mysqldump is a standard tool for backup and recovery. While effective for standard backups, mysqldump encounters several challenges in more complex scenarios, particularly when compatibility across different MySQL versions is required. This paper proposes a new approach, flexmysqlbackup, which provides more flexible backup options and rapid recovery without the limitations imposed by MySQL versions. Implementation scripts are presented, along with experiments demonstrating and proving its effectiveness.

Keywords – database backup, database recovery, storage system, MySQL, flexmysqlbackup

1. Introduction

Database backup and restore are essential aspects of database management. It is crucial for database professionals to possess the necessary knowledge and practical experience in data backup and recovery procedures.

DOI: https://doi.org/10.70862/CSIR.2025.0201-10

Corresponding author: Dimitar Todorov,
University of Plovdiv Paisii Hilendarski
Email: dimitar.todorov@uni-plovdiv.bg

Received: 26 April 2025.
Revised: 29 May 2025.
Accepted: 29 May 2025.
Published: 2 June 2025.

© 2024 Dimitar Todorov. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

The article is published with Open Access at http://csirjournal.com/

System failures can result from various circumstances, including hardware malfunctions, malware, data corruption, or physical server inaccessibility due to extreme events such as fires or natural disasters. Such situations can range from accidental errors, like connecting to the wrong server or executing a command such as ALTER TABLE [1], to more severe incidents, including malicious attacks or software bugs [2]. Although the probability of any single disaster occurring is relatively low, the accumulation of various risks significantly increases the possibility of data loss or service interruption. It is also surprisingly common to encounter situations where users delete data and subsequently request its restoration.

In cases of application errors leading to incorrect data entries, or during legal proceedings, it is crucial to be able to determine the state of the data or schema at a specific point in the past. Many hosting service providers do not back up MySQL servers at all, while others simply copy the files while the server is running. This approach can potentially create a corrupted backup, rendering it useless [3]. The mysqldump tool provides capabilities for backing up MySQL databases. Despite its popularity and usefulness, mysqldump has limitations that hinder its use in more complex backup and migration scenarios [4,5].

This paper proposes an alternative method, flexmysqlbackup, which offers more flexible and compatible solutions for database backup and recovery across different MySQL versions. Implementation scripts for flexmysqlbackup have been developed and are available at https://github.com/tuthon/flexmysqlbackup.

2. Limitations of Mysqldump

Key limitations of mysqldump include:

- Lack of selectivity in backing up objects;
- Limited compatibility between different MySQL versions;

- Difficulty in migrating specific objects;
- Client/server communication settings issues.

One of the most significant limitations of mysqldump is its lack of options for selectively backing up specific objects such as triggers, events, procedures, or views. For example, if it is necessary to back up only a specific trigger [6] or view [7] from a particular schema, mysqldump does not offer this capability. The available options force users to back up either all objects of a certain type (like all triggers associated with the dumped tables) or the entire database structure, which generates unnecessarily large backups for larger databases and increasing recovery time.

When creating a backup, mysqldump includes system parameters specific to the current MySQL version in the generated script. This frequently leads to errors, difficulties [8], and the need for manual modifications [9] when restoring the data to a different MySQL version. For example, MySQL 8.0 uses a new default collation, utf8mb4_0900_ai_ci, which does not exist in older versions like MySQL 5.7. Attempting to restore a backup created with mysqldump from version 8.0 onto a server running MySQL 5.7 results in an error that halts the restoration process [10].

```
mysql> CREATE TABLE test (id int NOT NULL)
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
- Unknown collation: 'utf8mb4_0900_ai_ci'
```

When the server and the client exchange data, they might use different character sets [11]. The server performs the necessary conversions (Figure 1).

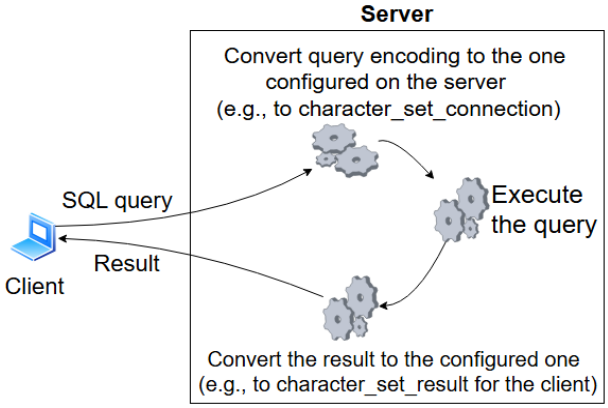


Figure 1. Client and server character sets.

- The server assumes that the client sends queries (or statements) in the character set specified by character_set_client.
- After the server receives a query from the client, it converts it to the character set specified by character_set_connection, which also determines how numbers are converted to strings.
- When the server returns results or error messages to the client, they are converted to the character set specified by character_set_result.

The solution involves using the server commands SET NAMES and SET CHARACTER SET. Client applications and APIs must also be configured with the correct character set, and mysql_real_escape_string() should be used to escape values [12,13].

During database migrations, it is often necessary to back up only specific objects or schemas to minimize errors and optimize the process. However, mysqldump does not offer sufficiently flexible options to perform this selectively. As a result, the database is frequently transferred with a larger volume of data than required, complicating both the restoration process and change management during migration.

A key disadvantage of mysqldump is the automatic inclusion of the mysql system schema when using options such as --all-databases. The mysql schema stores information regarding users, privileges, and other server-specific settings [14,15]. This can cause serious problems when restoring the backup onto a different server, such as:

- Overwriting existing users and privileges, which can lead to loss of access or unauthorized access.
- Incompatibilities if the servers use different MySQL versions or have specific configurations in the mysql schema.

These risks require careful handling and often necessitate manual editing of the backup file to exclude system schemas during migration.

To minimize these risks, it is recommended to:

- Exclude the mysql system schema by using more specific mysqldump options (instead of options like --all-databases).

```
mysqldump -u root -p --databases db1 db2 > backup.sql
```

- Manually edit the backup file to remove unwanted statements related to the mysql schema.
- Create separate backups for system data (like user grants and roles) if they are needed for restoring privileges or roles.

Due to these and other limitations, mysqldump is not always sufficiently flexible or reliable for modern application requirements.

A comparison between the main features of mysqldump and flexmysqlbackup is presented in Table 1.

Table 1. Comparison of main features of mysqldump and flexmysqlbackup.

Function	mysqldump	flexmysqlbackup
Version support	Good for the current version; some incompatibility between versions, manual intervention often required.	Better compatibility with versions 5.6 to 8.0
Archiving by schema	Partial (inconvenient)	Yes, through filters by schema and objects
Archiving of procedures, functions, triggers	Limited	Yes
Selective restore	Difficult or unsupported	Supported
Complexity of use	Low	Moderate
Performance	Average	Depends on configuration and version. In most cases, faster.

3. The Flexmysqlbackup Method for Flexible Backup

The proposed flexmysqlbackup method offers a flexible and effective solution for creating MySQL backups, overcoming the limitations of mysqldump. Its core concept relies on customized SQL scripts that extract definitions from the information schema database, the Stream Editor (sed) for text processing, and automation via Bash scripting. This approach provides flexibility, efficiency, and compatibility in backup management.

3.1. Core Elements of Flexmysqlbackup

INFORMATION_SCHEMA is a standard schema, supported by MySQL and other relational database management systems (RDBMS), which provides metadata about database objects such as tables, columns, views, triggers, and procedures. It serves as a read-only "data dictionary" or "catalog" providing access to information about the structure of the databases. MySQL introduced this schema in version 5.0, adhering to the SQL:2003 standard, but also including some MySQL-specific views. INFORMATION_SCHEMA is a key tool for administrators and developers as it allows standardized access to information about database structure. This facilitates database management and query optimization [16]. The most important tables within INFORMATION_SCHEMA are presented in Table 2.

A significant part of the flexmysqlbackup logic is implemented using stored procedures.

Stored procedures allow developers to group SQL statements that perform specific operations, such as transaction processing or data management [17]. They offer advantages like enhanced code reliability, increased security by limiting direct database access, and easier migration of business logic from applications to the database.

Table 2. Key tables in INFORMATION_SCHEMA.

Table	Description
TABLES	Contains information about all tables, such as name, type (BASE TABLE or VIEW), storage engine used (ENGINE), and other metadata
COLUMNS	Provides details about columns in tables, such as column name, data type (DATA_TYPE), character maximum length, and whether NULL values are allowed.
SCHEMATA	Lists available schemas, showing the default character set and collation.
VIEWS	Contains information about views, including their SQL definition and check option (update constraints).
TRIGGERS	Contains data about triggers, such as trigger name, event (INSERT, UPDATE, DELETE), and the table it is associated with.
EVENTS	Maintains information about scheduled events, such as start and end times, and the event type.

Bash is a compatible successor to the Bourne shell, incorporating key features from the C shell (csh) and the Korn shell (ksh), such as directory manipulation, job control, and brace expansion. One of the main reasons for Bash's popularity among database administrators is its compatibility with existing Unix shell environments and its extensive feature set that facilitates automation [18].

Bash scripts automate and coordinate the entire process by:

- Managing the execution of SQL commands.
- Generating backups for specific databases, tables, or procedures.
- Automating the processing of large data volumes by splitting backups into separate files for easier management.

Sed (stream editor) is a UNIX utility for filtering and transforming text. It operates directly on text streams or files, making it highly efficient for processing large files without requiring manual opening and editing [19, 20, 21]. Sed can be used for tasks such as:

- Removing DEFINER clauses to avoid permission issues during restoration.
- Adding supplementary SQL commands (e.g., USE database).

This ensures script compatibility across different environments.

3.2. Backup Process with Flexmysqlbackup

The main steps involved in the backup process using the flexmysqlbackup method are:

- Selecting objects for backup (e.g., schemas, tables, data, procedures, functions, triggers, events, views).

- Choose the scope for the selected object type (e.g., a single specific object, all objects of that type, or a defined group of objects).
- Generating SQL scripts.
- Use INFORMATION_SCHEMA to query metadata and structure definitions.
- Process the generated scripts using sed.
- Execute the scripts and store the results.
- Save the generated SQL scripts into appropriately named files.
- Arrange the backup files within a specific directory structure for easy access and management.

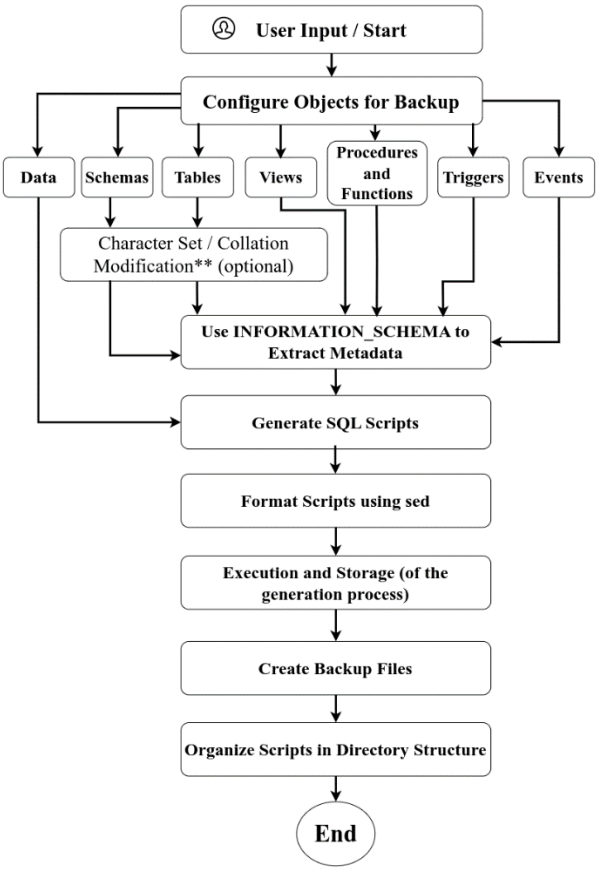


Figure 2. Backup process with flexmysqlbackup.

The primary advantages of flexmysqlbackup include:

- Selective Backup;
- Cross-Version Compatibility;
- Automation;
- Flexibility.

Selective Backup gives users the ability to choose precisely which objects to back up, such as:

- Specific tables from a given schema;
- Views and their dependencies;
- Stored procedures and functions.

This minimizes backup volume and restoration time.

Cross-Version Compatibility: Backups are automatically adapted for different MySQL versions, ensuring compatibility by:

- Removing unsupported or version-specific attributes.
- Ensuring consistency of character sets and collations during the restoration process.

Automation includes creating integrated backup processes using Bash:

- Simultaneous processing of multiple tasks, which reduces backup time.
- Easy integration into existing database management systems.

Flexibility offers the possibility to extend and customize the method according to the specific needs of the user:

- Archiving to specified engines, such as InnoDB or MyISAM.
- Changing character sets and collations during the process.

4. Implementation of Flexmysqlbackup

To implement the core object backup functionalities, stored procedures have been created for the following tasks:

1. Backing up schemas (database definitions);
2. Backing up table structures (CREATE TABLE statements);
3. Backing up table data (INSERT statements);
4. Backing up views;
5. Backing up procedures and functions;
6. Backing up triggers;
7. Backing up events.

For backing up procedures and functions, SQL queries implemented within a Bash script are used.

4.1. Backing up Schemas

The stored procedure gen_schemes_script is designed to generate an SQL script for creating databases (schemas) in MySQL, excluding the system databases (mysql, information_schema, performance_schema, sys). The procedure accepts an optional input parameter specifying the name of a single schema to process. If no schema name is provided, the procedure processes all non-system schemas.

The generated SQL code contains the necessary CREATE DATABASE statements.

An example Bash script command to generate a backup of all schemas and save it to a file named all_schemes.sql is:

```
$ mysql -hlocalhost -P3306 -udba -p --default-character-set=utf8mb4 --comments -e "CALL tools.gen_schemes_script(' ');" | sed '1d' > all_schemes.sql
Enter password: *******
```

The elements of the script are presented in table 3.

Table 3. Elements in the bash script for creating an archive of all schemas.

Element	Description
mysql	The MySQL client command used to interact with a MySQL server from the terminal.
-hlocalhost	The -h option specifies the host (server) the MySQL client will connect to; localhost indicates the server is running locally (on the same machine).
-P3306	The -P option specifies the port on which the MySQL server accepts connections. 3306 is the default port for MySQL.
-udba	The -u option specifies the username for the MySQL connection, which is dba in this example. *(Assuming 'dba' was a typo for 'dba')*
-p	The -p option indicates that a password is required. When the command is executed, it will prompt the user to enter the password for the dba user.
--default-character-set=utf8mb4	Specifies that the client should use utf8mb4 as the default character set. This is important to ensure correct encoding of text data when sending to and receiving from the MySQL server.
--comments	Instructs the MySQL client to preserve comments in SQL statements when they are sent to the server.
-e "CALL ..."	The -e option allows executing an SQL statement directly from the command line, in this case, "CALL tools.gen_schemes_script(' ');".
| sed '1d'	The symbol | (pipe) redirects the output from the mysql command to the sed command. In this case: '1d' instructs sed to delete the first line of the output, which usually contains the column headers returned by MySQL.
> all_dbs.sql	The > symbol redirects the result (after processing by sed) to a file named all_dbs.sql.

The generated SQL code for creating the databases is of the following form:

```
CREATE DATABASE IF NOT EXISTS `test_1`
CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
CREATE DATABASE IF NOT EXISTS `test_2`
CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

If an error occurs, the procedure returns a message containing an error code and a description.

4.2. Backing up Table Structures

The stored procedure gen_table_script generates SQL commands for creating tables, including column definitions, indexes, and keys. The following parameters can be provided to the gen_table_script procedure:

- database_name: The specific database to process, or all non-system databases if no name is specified.
- table_name: A specific table within the specified database. (If database_name is provided, this parameter allows targeting a single table within it)
- engine_name: If specified, only tables using this storage engine will be backed up.
- character set: If specified, the CREATE TABLE statement and string-based column definitions will be modified to use this character set.
- collation: If specified, the table default collation and column collations will be modified to use this collation.

By default, if the procedure is executed with empty parameters (e.g., CALL tools.gen_table_script(‘’, ‘’, ‘’, ‘’, ‘’);), it will back up the structure of all tables across all non-system databases.

The procedure can be executed with only the database_name parameter specified (e.g., CALL tools.gen_table_script('my_db', ‘’, ‘’, ‘’, ‘’);). In this case, it will generate a backup script for all tables within that specific database.

Specifying the table_name parameter requires that the database_name parameter also be provided (e.g., CALL tools.gen_table_script('my_db', 'my_table', ‘’, ‘’, ‘’);). It is not possible to specify only a table name without its database.

The engine_name parameter can be used independently. That is, if all other parameters are empty (‘’) and engine_name is set to, for example, INNODB or MYISAM (e.g., CALL tools.gen_table_script(‘’, ‘’, 'INNODB', ‘’, ‘’);), the procedure will generate a backup script for all tables (across non-system databases) that use the specified storage engine. This parameter can also be used in conjunction with database_name (e.g., CALL tools.gen_table_script('my_db', ‘’, 'INNODB', ‘’, ‘’);). In this scenario, it will create a backup script containing only those tables within the specified database that use the specified storage engine.

The character_set and collation parameters can be used in combination with the other parameters, or they can be left empty (‘’) if there is no need to modify the default character set and collation settings for the tables and columns during the backup generation.

An example command execution for the gen_table_script procedure to generate a backup of all table structures (across non-system databases) and save it to a file named all_tables.sql is:

```
mysql --default-character-set=utf8mb4 --comments -e "CALL tools.gen_table_script(' ', ' ', ' ', ' ', ' ');" | sed 's/\\n/\n/g' | sed '1d' > all_tables.sql
```

4.3. Backing up Table Data

The procedure gen_data_sql_script automates the creation of mysqldump commands specifically tailored for backing up data from a specific table, schema, or schemas. It provides high flexibility and is easily integrated into scripts. This procedure is executed on the source (primary) machine to generate a list of mysqldump commands required for creating the data backup. The parameters for the procedure are:

- ShemaName: The specific database (schema) to process, or all non-system databases if no name is specified.
- TableName: This parameter requires SchemaName to also be specified. It targets a specific table within the provided schema. If left empty (NULL) while SchemaName is specified, it defaults to including all tables within that schema.
- EngineName: This parameter functions identically to the corresponding parameter in the gen_table_script procedure, allowing filtering of tables based on their storage engine.

If an error occurs, the procedure returns a message containing an error code and a description.

The execution process can be understood in two main stages. First, a Bash script file (e.g., backup_data.sh) is created. Then, the gen_data_sql_script stored procedure is executed with the desired parameters. The output of this procedure, which consists of properly configured mysqldump commands for each targeted table or database, is redirected and saved into this .sh file. Finally, the generated Bash script (.sh file) is executed. Running this script executes the embedded mysqldump commands, which in turn produce the actual SQL file(s) containing the backed-up data (INSERT statements). An example execution sequence to create a data backup for all non-system databases is:

```
echo '#!/bin/bash' > all_data_script.sh
mysql  -h -P -u -p -e "call tools.gen_data_script(' ', ' ', ' ');" --silent
--raw --skip-column-names >> all_data_script.sh
```

The all_data_script.sh file, generated by executing the command that calls gen_data_sql_script, will contain content similar to this:

```
#!/bin/bash
mysqldump -P [port] -h [server_address] -u [username] -
p[password] --no-create-db --no-create-info --column-statistics=0
--skip-triggers --extended-insert [database_name_1]
[list_of_tables_1] > data_backup_db1.sql
mysqldump -P [port] -h [server_address] -u [username] -
p[password] --no-create-db --no-create-info --column-statistics=0
--skip-triggers --extended-insert [database_name_2]
[list_of_tables_2] > data_backup_db2.sql
# ... other mysqldump commands for different databases/tables as
generated by the procedure
```

Once the .sh file is created, it needs to be made executable. In Unix-based operating systems, this is done using the *chmod +x* command, as described in [22, 23].

If the system contains a large number of databases, and thus the generated .sh script produces many individual .sql backup files, it is recommended to distribute these generated SQL files across two or more directories. This organization allows the restoration process for these files to be performed in parallel (e.g., running multiple restore commands simultaneously, each targeting a different directory), which can significantly reduce the overall recovery time.

4.4. Backing up Views

The stored procedure gen_view_script.sql generates SQL commands (CREATE VIEW statements) for backing up views. The following parameters can be provided to the gen_view_script procedure:

- TableScheme: The specific database (schema) containing the views to back up. If left empty (‘’), it defaults to processing views across all non-system databases.
- ViewName: If this parameter is specified, the TableScheme parameter must also be provided. If both parameters are provided, only the specified view within the specified schema will be backed up. If ViewName is left empty (‘’), the procedure will back up all views either within the specified TableScheme (if provided) or across all non-system databases (if TableScheme is also empty).

An example command execution for the gen_view_script procedure to generate a backup of all existing views (across non-system databases) is:

```
$ mysql -hlocalhost -P3306 -udba -p --default-character-
set=utf8mb4 --comments -e "CALL tools.gen_view_script(' ', '
');" | sed 's/\\n/\n/g' | sed '1d' > sakila_views.sql
Enter password: *******
```

4.5. Backing up Procedures and Functions

The script file process_sql_opt.sh is used to generate SQL commands for backing up stored procedures and functions. It requires configuration settings to be filled in before it is executed.

Table 4. Parameters for the process_sql_opt.sh.

Parameter	Description
MYSQL_USER	The username for connecting to the database.
MYSQL_HOST	The address (hostname or IP) of the MySQL server.
MYSQL_PORT	The port number of the MySQL server.

Executing the process_sql_opt.sh script will display a menu on the screen. From this menu, the user must select the desired type of backup to create.

Table 5. Options for creating procedure and function backups.

Option	Description
1.All	Create a backup of all procedures and functions from all (non-system) databases.
2.Specific	Create a backup of only one specific procedure or function. Sub-options require specifying the Procedure/Function Name and the Database Name.
3.AllFromDb	Create a backup of all procedures and functions from a specific database. A sub-option requires specifying the Database Name.

An example diagram illustrating the execution process of the process_sql_opt.sh script for creating a backup of procedures and functions is provided in Figure 3.

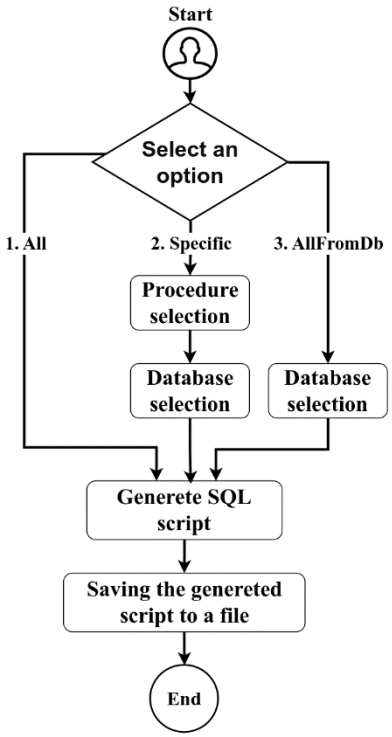


Figure 3. Procedure and Function Backup Process.

4.6. Backing up Triggers

The stored procedure gen_triggers_script generates SQL commands (CREATE TRIGGER statements) for backing up triggers. The following parameters can be provided to the procedure:

- ShemaName: The specific database (schema) containing the triggers to back up. If left empty (‘’), it defaults to processing triggers across all non-system databases.
- TriggerName: if this parameter is specified, the SchemaName parameter must also be provided. If both parameters are provided, only the specified trigger will be backed up. If TriggerName is left empty (‘’), the procedure will back up all triggers from the specified schema, or from all non-system schemas if none is provided.

An example command execution for the gen_triggers_script stored procedure to generate a backup of all triggers and save it to a file named all_triggers.sql is:

```
$ mysql -hlocalhost -P3306 -udba -p --comments -e "CALL
tools.gen_triggers_script(' ', ' ');" | sed 's/\\n/\n/g' | sed '1d' >
all_triggers.sql
Enter password: *******
```

4.7. Backing up Events

The stored procedure gen_event_script generates SQL commands (CREATE EVENT statements) for backing up scheduled events. The following parameters can be provided to the gen_event_script procedure:

- ShemaName: The specific database (schema) where the events are defined. If left empty (‘’), it defaults to processing events across all non-system databases.
- EventName: If this parameter is specified, the SchemaName parameter must also be provided. If both parameters are provided, only the specified event within the specified schema will be backed up. If EventName is left empty (‘’), the procedure will back up all events either within the specified SchemaName (if provided) or across all non-system databases (if SchemaName is also empty).

An example command execution for the gen_event_script stored procedure to generate a backup of all events (across non-system databases) and save it to a file named all_events.sql is:

```
$ mysql -hlocalhost -P3306 -udba -p --comments -e "CALL
tools.gen_event_script(' ', ' ');" | sed 's/\\n/\n/g' | sed '1d' >
all_events.sql
Enter password: *******
```

5. Experiments and Results

MySQL 5.1.6 and newer versions feature an event scheduler, which allows the execution of database operations activated at scheduled times. An event is essentially a stored program associated with a schedule. The schedule determines the time or time intervals during which the event executes, and optionally, when it expires (ceases to exist). Events are particularly useful for performing automated administrative tasks, such as periodic updates of summary reports, purging old data, or rotating log tables [12].

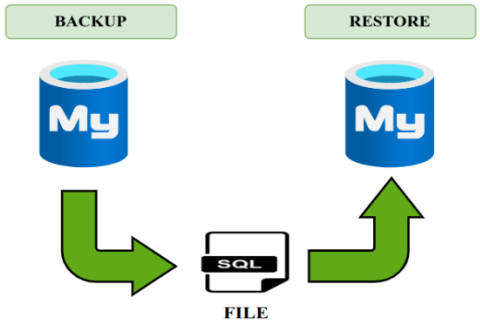


Figure 4. Simplified Backup and Restore/Deployment Model.

Figure 4 illustrates a simplified model for MySQL database backup/export and restore/import. The process begins with the backup (export) phase, where data and structure from the database are extracted and saved into an SQL file. This file contains a text-based representation of the database's structure and/or content. During the restore (import/deployment) phase, the SQL file is used to recreate the database in a new or existing MySQL instance.

This model ensures data portability and facilitates recovery in case of failures or migration to a new system. If the system utilizes automatic (scheduled) events that generate data, it is mandatory to execute the following command on both the source (backup) machine and the target (restore) ma-chine before creating the data backup and before restoring it:

```
SET GLOBAL event_schaduler = OFF
```

After the entire backup has been restored, execute the following command:

```
SET GLOBAL event_schaduler = ON
```

5.1. Example: Creating and Restoring a Full Backup

When creating a backup with flexmysqlbackup, the specific order in which the individual backup components (schemas, table structures, data, views, routines, etc.) are generated is generally not critical. However, to create a complete backup of the entire system, each relevant script or procedure call must be executed. The following sequence shows an example of how this can be done:

```
mysql -h -P -u -p -e "CALL tools.gen_schemes_script(' ');" | sed
'1d' > all_schemes.sql
mysql -h -P -u -p -e "CALL tools.gen_table_script(1, ' ', ' ', ' ', ' ', '
');" | sed 's/\\n/\n/g' | sed '1d' > all_tables.sql
mysql -h -P -u -p -e "CALL tools.gen_view_script(' ', ' ');" | sed
's/\\n/\n/g' | sed '1d' > all_views.sql
./process_sql_opt.sh
mysql -h -P -u -p -e "CALL tools.gen_triggers_script(' ', ' ');" |
sed 's/\\n/\n/g' | sed 's/\\t//g' | sed '1d' > all_triggers.sql
mysql -h -P -u -p -e "CALL tools.gen_event_script(' ', ' ');" | sed
's/\\n/\n/g' | sed '1d' > all_events.sql
echo '#!/bin/bash' > all_data_script.sh
mysql  -h -P -u -p -e "call tools.gen_data_script(' ', ' ', ' ');" –silent
--raw --skip-column-names >> all_data.sql
```

During the restoration (deployment) of the backup, the steps must follow a specific order because MySQL has dependencies between different database components. For example, a stored procedure cannot be created if its code references a table that does not yet exist. Similarly, a trigger cannot be created if the table it is attached to does not exist. The steps for restoring the backup, followed in the correct order, are:

```
mysql -h -P -u -p < all_schemes.sql
mysql -h -P -u -p < all_tables.sql
mysql -h -P -u -p < all_views.sql
mysql -h -P -u -p --comments < all_procedures.sql
mysql -h -P -u -p < all_data.sql
mysql -h -P -u -p < all_triggers.sql
mysql -h -P -u -p < all_events.sql
```

Figure 5 illustrates the process of database backup and restoration using SQL files. The diagram depicts the main stages of interaction between the user, the primary database server, the storage server, and the secondary (restore) server.

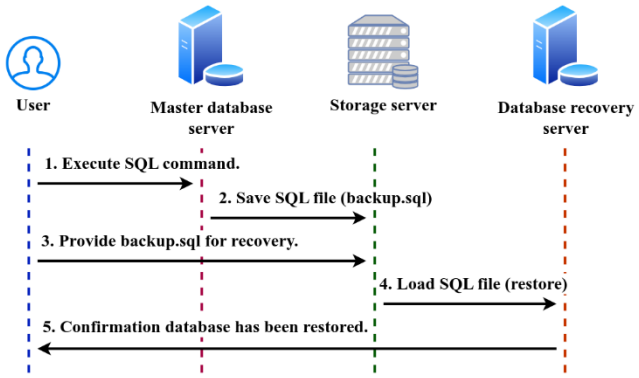


Figure 5. Unified Diagram of the Backup and Restoration Process.

The key steps in the backup and restoration process are:

- The user executes an SQL command to initiate the backup process.
- The primary database server generates and saves an SQL file – backup.sql – which contains the structure and data of the database.
- The backup file is transferred to and stored on a dedicated storage server.
- The backup.sql file is loaded onto a secondary database management server to perform the restoration.
- Finally, the successful restoration of the data is confirmed/verified.

This sequence provides clarity on the steps involved in backup management and recovery, ensuring data protection and minimizing potential downtime during database transfers between servers.

5.2. Experiments

The testing methodology consisted of repeatedly exporting and importing the Sakila database under controlled conditions. Each test measured the execution time and the success of full restoration under different MySQL versions and configurations. The tests were conducted in both clustered and standalone environments, as described in Section 5.2.1.

The sample Sakila database was developed by the MySQL team. It was created as a more complex alternative to the previous "world" sample database and serves for training purposes, as well as for testing SQL queries. The database can be freely downloaded and used from the official MySQL website [24].

Additionally, five events were added to the Sakila database. Two bash scripts were created for automated backup and restoration of the Sakila database. One script uses mysqldump, and the other uses flexmysqlbackup.

The scripts also incorporate *bc* (Basic Calculator), a powerful command-line utility for performing high-precision arithmetic calculations. It supports operations with integers and floating-point numbers, as well as complex mathematical functions, and can be used both interactively and within scripts [25]. In this context, *bc* is used solely for the purpose of the experiments: to measure the time taken for backup creation and restoration by each method, calculate the arithmetic mean of these times over multiple runs, and subsequently compare the performance values between flexmysqlbackup and mysqldump.

The testing process for each method (mysqldump and flexmysqlbackup) involved the following steps:

- Creating a backup of the sample sakila database.
- Restoring the database from the created backup.
- Comparing the results using the following metrics:
  - Size of the generated SQL backup file(s);
  - Time taken for the backup process;
  - Time taken for the restoration process.

It is important to note that the mysqldump utility varies between different MySQL versions and is optimized for the specific version it accompanies. Backups created with mysqldump from one version might not be restorable on another MySQL version without manual modifications to the script. In contrast, flexmysqlbackup is designed as a universal tool that can be used with various MySQL versions, and the backups created with it can be restored across different versions, potentially including future ones.

5.2.1. Test Environment and Hardware Configuration

Tests with MySQL 5.6 were conducted in a clustered environment with the NDB Storage Engine, based on MySQL 5.6.35 with NDB Cluster 7.4.14.

Cluster architecture:

- Data Node (ndbd): 2 nodes;
- Management Node (ndb_mgmd): 1 node;
- MySQL API Node (mysqld): 1 active node;
- Total: 2 data nodes, 1 MGM, 1 MySQL API node;
- The machines were virtualized using KVM/QEMU.

Memory usage in the NDB cluster (retrieved with ndb_mgm):

- Node 2: Data usage is 63%;
- Node 2: Index usage is 48%;
- Node 3: Data usage is 63%;
- Node 3: Index usage is 48%.

Table 6. Hardware configuration of MySQL nodes.

Compo-nent	CPU (model and cores)	RAM	Disk (model and size)	Disk usage
MySQL API Node	Common KVM processor, 2 cores	23 GiB	QEMU HARDDISK, 150 GB	~30 GB out of 142 GB (approx. 21%)
Manage-ment Node	Common KVM processor, 2 cores	2 GiB	QEMU HARDDISK, 30 GB	18 GB out of 27 GB (67%)
Data Nodes (1 and 2)	Common KVM processor, 4 cores	16 GiB	QEMU HARDDISK, 150 GB	106 GB out of 147 GB (72%)

Tests with MySQL 8.0.40 were conducted on a separate physical machine without NDB Cluster. Hardware configuration:

- System: Micro-Star International Prestige 15 A11SCX;
- RAM: 64 GiB (2x32GiB DDR4 3200 MHz);
- CPU: 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz;
- Cache: 192 KiB L1, 5 MiB L2, 12 MiB L3;
- Disk: 1 TB NVMe (Phison SM2801T24GKBB4S-E162);
- Kernel: Linux 6.11.0-26-generic.

Table 7. Operating Systems.

Cluster version	Component	Operating system
NDB 7.4.14	MySQL API Node	Debian GNU/Linux 8 (jessie)
	MGM Node	Debian GNU/Linux 9 (stretch)
	Data Node 1	Debian GNU/Linux 9 (stretch)
	Data Node 2	Debian GNU/Linux 9 (stretch)
MySQL 8.0.40	Standalone	Ubuntu 22.04.1 LTS

5.2.2. Test 1

Test 1 was conducted on MySQL version 8.0.40. The execution times for individual parts of the process and the sizes of the resulting backup scripts are presented in Tables 8 and 9, respectively.

Table 8. Test 1 Results – Time Metrics.

Metric	Mysqldump (sec)	Flexmysqlbackup (sec)	Difference (sec)	Percentage Difference
Average creation time	0.247	0.191	-0.056	-22.67% (faster)
Average restore time	1.578	1.584	+0.006	+0.38% (slower)
Average total time (create + restore)	1.825	1.776	-0.049	-2.68% (faster)

Flexmysqlbackup is significantly faster during the creation phase (≈22.7% faster). Restoration time is nearly identical for both methods, with flexmysqlbackup being marginally slower by only 0.38%. The overall time (creation + restore) is, on average, 2.68% better (faster) when using flexmysqlbackup.

The data files, which consist of INSERT statements, are identical in size for both methods. For the other database objects – procedures, triggers, events, table structures, and views – flexmysqlbackup demonstrates an advantage, consistently producing smaller backup files (table 9).

Table 9. Test 1 Results – Backup File Sizes.

Category	Mysqldump (KB)	Flexmysqlbackup (KB)	Reduction (%)
Data	3276.8	3276.8	0.00%
Stored Procedures & Functions	12	6.1	49.17%
Events	7.3	1.6	78.08%
Tables & Views (structure)	30	13	56.67%
Triggers	7.1	5.8	18.31%

5.2.3. Test 2

Test 2 was conducted on MySQL 5.6. In this test, flexmysqlbackup was significantly slower during the creation phase – more than 3 times slower. However, the restoration process was approximately 19% faster with flexmysqlbackup. The overall time (creation + restore) was more than twice as long when using flexmysqlbackup compared to mysqldump.

Further analysis revealed that the reason for the slowdown lies within the script responsible for generating table structures, specifically the part that constructs the foreign key constraints. When executing queries that retrieve foreign key information from INFORMATION_SCHEMA (using the REFERENTIAL_CONSTRAINTS and KEY_COLUMN_USAGE tables), significantly lower performance is observed in MySQL 5.6 compared to MySQL 8.0. The primary reason is that in MySQL 5.6, INFORMATION_SCHEMA is implemented using dynamically generated, non-indexed views. In contrast, MySQL 8.0 stores metadata in a centralized and indexed InnoDB-based structure known as the Data Dictionary, which considerably speeds up access [26].

Table 10. Test 2 Results – Time Metrics.

Metric	Mysqldump (sec)	Flexmysqlbackup (sec)	Difference (sec)	Percentage Difference
Average creation time	1.441	6.043	+4.602	+319.3% (slower)
Average restore time	1.431	1.160	-0.271	-18.94% (faster)
Average total time (create + restore)	2.584	7.204	+4.620	+178.8% (slower)

In a specific test case, a query retrieving foreign keys executed in approximately ~1 second on MySQL 5.6, compared to ~0.1 seconds on MySQL 8.0 - demonstrating roughly a 90% faster execution time on the newer version.

5.2.4. Test 3

In Test 3, a backup created on MySQL 8.0 was restored onto MySQL 5.6 (Table 11). It is important to note that for this test, the backup created using mysqldump required manual modifications. Specifically, the collations for tables, columns, procedures, triggers, and events had to be changed from utf8mb4_0900_ai_ci to utf8mb4_general_ci. This is typically done using a command like:

```
sed -i 's/utf8mb4_0900_ai_ci/utf8mb4_general_ci/g' *.sql
```

Furthermore, it's important to highlight that the address table contains a column named location defined with the geometry data type, and this column also has a SPATIAL index defined on it. Restoring this on MySQL 5.6 requires additional manual script transformations, including ensuring the table uses the MyISAM engine (as SPATIAL indexes were primarily supported on MyISAM in versions up to 5.6) and potentially adjusting the column and index definitions appropriately.

In contrast to mysqldump, flexmysqlbackup did not require these manual script transformations for the restoration. In the conducted experiment, disregarding the backward compatibility issues and the manual effort required for the mysqldump backup when migrating from version 8 to 5.6, the restoration time for both methods was found to be comparable (Table 11).

Table 11. Test 3 Results – Restore Time (MySQL 8.0 backup to MySQL 5.6).

Metric	Mysqldump (sec)	Flexmysqlbackup (sec)	Difference (sec)	Percentage Difference
Average restore time	1.111	1.150	+0.039	+3.51% (slower)

In Test 3, where a backup from MySQL 8.0 was restored onto MySQL 5.6, flexmysqlbackup was approximately 3.51% slower during the actual restoration process compared to mysqldump. However, an important clarification is that mysqldump required manual corrections (for collations, data types, and indexes), which makes it less convenient in this scenario, despite the slight time advantage.

5.2.5. Test 4

In test 4 backup created on MySQL 5.6 was restored onto MySQL 8.0 (Table 12).

Table 12. Test 4 results.

Metric	Mysqldump (sec)	Flexmysqlbackup (sec)	Difference (sec)	Percentage Difference
Average restore time	1.631	1.510	-0.121	-7.42% (faster)

In Test 4, flexmysqlbackup was 7.42% faster during the restoration process.

6. Future Work and Community Involvement

Further development of the flexmysqlbackup tool is planned, with a focus on higher automation, performance improvement, and fixing existing bugs. Expanding its functionality is planned through the creation of a user-friendly graphical user interface (GUI) to facilitate its use by a wider range of users, including those without advanced technical skills.

Tests have also been performed with MySQL version 8.4.4, where flexmysqlbackup demonstrated excellent stability and compatibility. Additional tests are planned with newer versions, including MySQL 9.3, aimed at ensuring maximum compatibility and reliability.

The goal is for the tool to become a stable, widely used solution that system administrators and developers can rely on in all kinds of environments – from small installations to corporate infrastructures. The project is already publicly available on GitHub, where community participation is encouraged. Suggestions, bug reports, and pull requests are welcomed from anyone wishing to contribute to the future development and improvement of the tool.

7. Conclusion

Flexmysqlbackup represents a significant advancement in database backup management, overcoming key limitations inherent in the standard mysqldump utility. It provides the necessary flexibility to adapt to the specific needs of various projects and organizations.

One of its primary advantages is the capability for selective backup. This significantly reduces restoration times and optimizes resource utilization. In complex systems where only critical information needs to be backed up, the ability to select specific objects – such as tables, triggers, or views – eliminates unnecessary processing and storage overhead.

Cross-version compatibility between different MySQL releases is another central aspect of flexmysqlbackup. By eliminating the need for manual intervention to handle differences in collations or version-specific parameters, administrators can perform migrations with minimal disruption and high confidence in successful recovery. This is particularly crucial for large companies that rely on complex and critical databases for their daily operations.

References

[1] Reese, G. (2007). *MySQL pocket reference: SQL functions and utilities* (pp. 37–41). O'Reilly Media, Inc.

[2] Norman, M. (2004). Changing data. In *Database design manual: Using MySQL for Windows* (pp. 87-101). Springer. https://doi.org/10.1007/1-85233-859-8_8

[3] Schwartz, B., Zaitsev, P., & Tkachenko, V. (2012). *High performance MySQL: Optimization, backups, and replication* (pp. 622–623). O'Reilly Media, Inc.

[4] Kofler, M. (2005). *mysql, mysqladmin, and mysqldump.* In *The Definitive Guide to MySQL 5* (pp. 61–70). https://doi.org/10.1007/978-1-4302-0071-0_4

[5] MySQL Documentation. (n.d.). *Backup and recovery*. MySQL 8.0 Reference Manual. Retrieved March 20, 2025, from https://dev.mysql.com/doc/refman/8.0/en/backup-and-recovery.html

[6] Reese, G. (2003). *MySQL pocket reference* (pp. 3, 61). O'Reilly Media, Inc.

[7] Kromann, F. M. (2018). MySQL views. In *Beginning PHP and MySQL* (pp. 733-750). Apress. https://doi.org/10.1007/978-1-4302-6044-8_31

[8] MySQL Documentation. (n.d.). *mysqldump — A database backup program*. MySQL 8.0 Reference Manual. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html

[9] MySQL. (n.d.). *Upgrading to MySQL 8.0? Here is what you need to know...* Retrieved from https://dev.mysql.com/blog-archive/upgrading-to-mysql-8-0-here-is-what-you-need-to-know/

[10] Christudas, B. (2019). *MySQL* (pp. 877–884). Apress. https://doi.org/10.1007/978-1-4842-4501-9_27

[11] Dyer, R. (2008). *MySQL in a nutshell* (pp. 68, 105–109, 235, 315–317, 323, 362–363). O'Reilly Media, Inc.

[12] DuBois, P. (2013). *MySQL* (pp. 212–213, 684–685, 1005, 1078, 1110, 1148). Addison-Wesley. ISBN 9780321833877.

[13] Schwartz, B., Zaitsev, P., & Tkachenko, V. (2012). *High performance MySQL: Optimization, backups, and replication*. O'Reilly Media, pp. 298–305.

[14] Bell, C., Kindahl, M., & Thalmann, L. (2014). *MySQL high availability: Tools for building robust data centers*. O'Reilly Media.

[15] Tahaghoghi, S. M. M., & Williams, H. E. (2006). *Learning MySQL*. O'Reilly Media, Inc. ISBN 9780596008642.

[16] Krogh, J. W. (2020). *MySQL 8 query performance tuning: A systematic method for improving execution speeds* (pp. 111–131). Apress. https://doi.org/10.1007/978-1-4842-5584-1

[17] Harrison, G., & Feuerstein, S. (2006). *MySQL Stored Procedure Programming: Building High-Performance Web Applications in MySQL*. O'Reilly Media.

[18] Newham, C. (2005). *Learning the bash shell: Unix shell programming* (p. 16). O'Reilly Media.

[19] Richardson, L. (2005). Text manipulation with sed. *Linux Journal*, Slashdot Media.

[20] Natarajan R. (2007). *Sed and awk 101 hacks*. Publ. The Geek Stuff.

[21] Dougherty, D., & Robbins, A. (1997). *sed & awk: UNIX power tools*. O'Reilly Media, Inc.

[22] Shotts, W. (2012). *The Linux command line: A complete introduction* (pp. 81–87). No Starch Press.

[23] Nemeth, E., Snyder, G., Hein, T. R., Whaley, B., & Mackin, D. (2018). *UNIX and Linux system administration handbook* (pp. 156–158). USENIX Open Access Policy, 59.

[24] MySQL Documentation. (n.d.). *Sakila sample database*. Retrieved April 1, 2025, from https://dev.mysql.com/doc/sakila/en/

[25] Blum, R. (2008). *Linux command line and shell scripting bible* (p. 228). Wiley Publishing, Inc. ISBN 978-0-470-25128-7.

[26] MySQL. (n.d.). *MySQL 8.0 Reference Manual: MySQL data dictionary*. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/data-dictionary.html