MYSQL – USING SEQUENCES

**Using AUTO\_INCREMENT Column & Renumbering an Existing Sequence in MySQL Workbench**

The AUTO\_INCREMENT feature in MySQL is used to automatically generate unique numeric values for a column, typically used for primary keys. However, sometimes you may need to **renumber an existing sequence** to remove gaps caused by deletions or other modifications.

**1. Using AUTO\_INCREMENT Column**

When you define a column with AUTO\_INCREMENT, MySQL automatically assigns an increasing integer value to new rows.

**Example: Creating a Table with AUTO\_INCREMENT**

CREATE TABLE employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(100)

);

* The id column is automatically assigned a unique number starting from 1 by default.
* Each new row inserted gets the next available number.

**Inserting Data**

INSERT INTO employees (name, department) VALUES ('Alice', 'HR');

INSERT INTO employees (name, department) VALUES ('Bob', 'IT');

INSERT INTO employees (name, department) VALUES ('Charlie', 'Finance');

This will result in:

+----+---------+------------+

| id | name | department |

+----+---------+------------+

| 1 | Alice | HR |

| 2 | Bob | IT |

| 3 | Charlie | Finance |

+----+---------+------------+

**2. Renumbering an Existing Sequence**

**Why Renumber?**

If rows are deleted, the AUTO\_INCREMENT sequence does not automatically fill the gaps. Example:

DELETE FROM employees WHERE id = 2;

After deletion:

+----+---------+------------+

| id | name | department |

+----+---------+------------+

| 1 | Alice | HR |

| 3 | Charlie | Finance |

+----+---------+------------+

Now, id = 2 is missing.

**Steps to Renumber the IDs**

1. **Create a Backup (Optional but Recommended)**

CREATE TABLE employees\_backup AS SELECT \* FROM employees;

1. **Reset AUTO\_INCREMENT to Start from 1**

ALTER TABLE employees AUTO\_INCREMENT = 1;

1. **Reassign Sequential Numbers**

Create a new temporary table without AUTO\_INCREMENT:

CREATE TABLE temp\_employees AS

SELECT @row := @row + 1 AS id, name, department

FROM employees, (SELECT @row := 0) AS init

ORDER BY id;

This assigns new sequential id values.

1. **Replace the Original Table**

DROP TABLE employees;

ALTER TABLE temp\_employees RENAME TO employees;

1. **Re-enable AUTO\_INCREMENT**

ALTER TABLE employees MODIFY id INT AUTO\_INCREMENT PRIMARY KEY;

**Final Table (After Renumbering)**

+----+---------+------------+

| id | name | department |

+----+---------+------------+

| 1 | Alice | HR |

| 2 | Charlie | Finance |

+----+---------+------------+

Now, the id values are sequential again.

**Conclusion**

* AUTO\_INCREMENT ensures unique numbers for a column.
* Deleted rows create gaps that **do not auto-fill**.
* You can renumber the column using **temporary tables** and **reassignment techniques**.

**Handling Duplicates in MySQL Workbench**

Duplicate data in a table can cause inconsistencies and errors in data processing. This guide will cover:

1. **Preventing Duplicates from Occurring in a Table**
2. **Counting and Identifying Duplicates**
3. **Eliminating Duplicates from a Query Result**
4. **Removing Duplicates Using Table Replacement**

## ****1. Preventing Duplicates from Occurring in a Table****

To prevent duplicates, you can use **PRIMARY KEY**, **UNIQUE CONSTRAINT**, or **INDEX**.

### ****Example: Using PRIMARY KEY****

A primary key enforces uniqueness in a column.

CREATE TABLE employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

email VARCHAR(255) UNIQUE,

name VARCHAR(100),

department VARCHAR(100)

);

* The email column is marked as UNIQUE, preventing duplicate email entries.

### ****Example: Using a Composite Unique Constraint****

If duplicates are determined by multiple columns (e.g., name and department together must be unique), use:

CREATE TABLE employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(100),

UNIQUE (name, department)

);

This prevents inserting the same name in the same department multiple times.

## ****2. Counting and Identifying Duplicates****

To find duplicate records, use GROUP BY with HAVING COUNT(\*) > 1.

### ****Example: Identifying Duplicates****

SELECT name, department, COUNT(\*) AS count

FROM employees

GROUP BY name, department

HAVING COUNT(\*) > 1;

This query will return rows where name and department appear more than once.

### ****Example: Finding Duplicate Email Entries****

SELECT email, COUNT(\*)

FROM employees

GROUP BY email

HAVING COUNT(\*) > 1;

If duplicates exist, the output might look like:

+---------------------+-------+

| email | count |

+---------------------+-------+

| john@example.com | 2 |

| alice@example.com | 3 |

+---------------------+-------+

## ****3. Eliminating Duplicates from a Query Result****

To **show only distinct records** without modifying the data, use DISTINCT.

### ****Example: Using DISTINCT****

SELECT DISTINCT name, department FROM employees;

This removes duplicates from the result but **does not delete them from the table**.

## ****4. Removing Duplicates Using Table Replacement****

If duplicates already exist and need to be removed permanently, follow these steps:

### ****Step 1: Create a Temporary Table Without Duplicates****

CREATE TABLE temp\_employees AS

SELECT \* FROM employees

GROUP BY name, department; -- Adjust columns based on what defines a duplicate

This creates a copy of the table, keeping only one instance of each duplicate.

### ****Step 2: Delete the Original Table****

DROP TABLE employees;

### ****Step 3: Rename the Temporary Table****

ALTER TABLE temp\_employees RENAME TO employees;

### ****Alternative: Delete Duplicates Using**** ROW\_NUMBER()

If your MySQL version supports **window functions (MySQL 8+),** you can use:

WITH ranked AS (

SELECT \*, ROW\_NUMBER() OVER (PARTITION BY name, department ORDER BY id) AS row\_num

FROM employees

)

DELETE FROM employees WHERE id IN (

SELECT id FROM ranked WHERE row\_num > 1

);

This keeps the first occurrence and removes the rest.

## ****Conclusion****

* **Prevent duplicates** with UNIQUE constraints.
* **Identify duplicates** using GROUP BY HAVING COUNT(\*) > 1.
* **Eliminate duplicates from queries** using DISTINCT.
* **Remove duplicates permanently** by creating a new table or using ROW\_NUMBER().

**Preventing SQL Injection & The LIKE Quandary in MySQL Workbench**

## ****1. Preventing SQL Injection****

### ****Step 1: Create a Sample**** users ****Table****

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) UNIQUE,

password VARCHAR(255)

);

### ****Step 2: Insert Sample Data****

INSERT INTO users (username, password) VALUES

('john\_doe', 'secure123'),

('alice\_smith', 'pass456'),

('hacker', 'badpassword');

### ****🚨 Vulnerable Query (SQL Injection)****

If a user enters:

username: ' OR '1'='1

password: anything

It generates this SQL query:

SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = 'anything';

Since '1'='1' is always **TRUE**, this query **returns all users** instead of one.

✅ **Output of SQL Injection Attack**

+----+-------------+-------------+

| id | username | password |

+----+-------------+-------------+

| 1 | john\_doe | secure123 |

| 2 | alice\_smith | pass456 |

| 3 | hacker | badpassword |

+----+-------------+-------------+

🚨 **Security Risk: Attacker gains unauthorized access to all users!**

### ****✅ Preventing SQL Injection Using Prepared Statements****

PREPARE stmt FROM 'SELECT \* FROM users WHERE username = ? AND password = ?';

SET @username = 'john\_doe';

SET @password = 'secure123';

EXECUTE stmt USING @username, @password;

✅ **Output (Only the correct user is returned)**

+----+-----------+-------------+

| id | username | password |

+----+-----------+-------------+

| 1 | john\_doe | secure123 |

+----+-----------+-------------+

🚀 **Fix: Prevents SQL injection by binding user inputs safely!**

## ****2. Handling The LIKE Quandary****

### ****Step 1: Insert More Data****

INSERT INTO users (username, password) VALUES

('john%', 'tricky'),

('john\_doe1', 'testpass'),

('johnny', 'randompass');

### ****🚨 Problem: Searching for "john" Using LIKE****

SELECT \* FROM users WHERE username LIKE '%john%';

✅ **Output (Incorrectly returns unintended matches due to %)**

+----+-----------+-------------+

| id | username | password |

+----+-----------+-------------+

| 1 | john\_doe | secure123 |

| 4 | john% | tricky |

| 5 | john\_doe1 | testpass |

| 6 | johnny | randompass |

+----+-----------+-------------+

🚨 **Problem: john% is treated as a wildcard match, which may return unintended results!**

### ****✅ Solution 1: Escape Wildcards****

SELECT \* FROM users WHERE username LIKE 'john\%' ESCAPE '\';

✅ **Output (Now only matches john%)**

+----+--------+---------+

| id | username | password |

+----+--------+---------+

| 4 | john% | tricky |

+----+--------+---------+

🚀 **Fix: Treats % as a normal character instead of a wildcard.**

### ****✅ Solution 2: Replace Wildcards in User Input****

SET @search\_input = REPLACE(REPLACE('john%', '%', '\%'), '\_', '\\_');

SELECT \* FROM users WHERE username LIKE CONCAT('%', @search\_input, '%') ESCAPE '\';

✅ **Output (Safely handles wildcards in input)**

+----+-----------+-------------+

| id | username | password |

+----+-----------+-------------+

| 1 | john\_doe | secure123 |

| 5 | john\_doe1 | testpass |

| 6 | johnny | randompass |

+----+-----------+-------------+

🚀 **Fix: Ensures that user input is safely used with LIKE.**

### ****✅ Solution 3: Case-Sensitive Search****

SELECT \* FROM users WHERE BINARY username LIKE 'John%';

✅ **Output (Returns only exact case matches)**

Empty set

🚀 **Fix: Enforces case-sensitive matching, preventing unintended case-insensitive matches.**

## ****Conclusion****

| **Issue** | **Problem** | **Fix** |
| --- | --- | --- |
| **SQL Injection** | Attackers manipulate queries using input | Use **Prepared Statements** |
| **LIKE Quandary** | % wildcard causes unexpected matches | Escape % and \_ with \ |
| **Case-Insensitive Matching** | LIKE treats John and john the same | Use BINARY for strict case matching |

**Exporting Data in MySQL Workbench**

In MySQL Workbench, you can export data in multiple ways, including:

1. **Exporting Data with the SELECT ... INTO OUTFILE Statement**
2. **Exporting Tables as Raw Data**
3. **Copying Tables or Databases to Another Host**

## ****1. Exporting Data with the**** SELECT ... INTO OUTFILE ****Statement****

### ****What is**** SELECT ... INTO OUTFILE****?****

The SELECT ... INTO OUTFILE statement allows you to export data from a table into a text file (CSV, TXT, etc.).

### ****🔹 Syntax****

SELECT \* INTO OUTFILE '/path/to/output.csv'

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

FROM users;

* **/path/to/output.csv** → The full path where the file will be saved.
* **FIELDS TERMINATED BY ','** → Columns are separated by commas (CSV format).
* **ENCLOSED BY '"'** → Each column value is enclosed in double quotes.
* **LINES TERMINATED BY '\n'** → Each row is written on a new line.

### ****🔹 Example: Exporting a**** users ****Table to CSV****

#### **Step 1: Create a Sample** users **Table**

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(255),

age INT

);

#### **Step 2: Insert Sample Data**

INSERT INTO users (name, email, age) VALUES

('John Doe', 'john@example.com', 28),

('Alice Smith', 'alice@example.com', 32),

('Bob Johnson', 'bob@example.com', 25);

#### **Step 3: Export Data**

SELECT \* INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv'

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

FROM users;

✅ **Output (users.csv)**

1,"John Doe","john@example.com",28

2,"Alice Smith","alice@example.com",32

3,"Bob Johnson","bob@example.com",25

### ****🔹 Important Notes****

* The file **must be saved in a directory** MySQL has access to (e.g., C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/).
* Use **SECURE\_FILE\_PRIV** to check export permissions:

SHOW VARIABLES LIKE 'secure\_file\_priv';

## ****2. Exporting Tables as Raw Data****

### ****🔹 Using MySQL Workbench GUI****

1. **Open MySQL Workbench**.
2. **Go to Server > Data Export**.
3. Select **the database and table** you want to export.
4. Choose **Export to Self-Contained File** or **Export to Dump Project Folder**.
5. Click **Start Export**.

### ****🔹 Using**** mysqldump ****Command****

mysqldump allows you to export entire tables or databases.

#### **Export a Single Table**

mysqldump -u root -p my\_database users > users.sql

✅ **Output (users.sql)**

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(255),

age INT

);

INSERT INTO users (name, email, age) VALUES

('John Doe', 'john@example.com', 28),

('Alice Smith', 'alice@example.com', 32),

('Bob Johnson', 'bob@example.com', 25);

#### **Export an Entire Database**

mysqldump -u root -p my\_database > my\_database.sql

## ****3. Copying Tables or Databases to Another Host****

### ****🔹 Copying a Table to Another Host****

#### **Step 1: Export the Table**

mysqldump -u root -p my\_database users > users.sql

#### **Step 2: Transfer the File to Another Server**

Use scp (Linux/macOS) or any file transfer tool:

scp users.sql user@remote\_host:/home/user/

#### **Step 3: Import the Table into the Remote Server**

mysql -u root -p my\_database < /home/user/users.sql

### ****🔹 Copying an Entire Database to Another Host****

#### **Step 1: Export the Database**

mysqldump -u root -p --databases my\_database > my\_database.sql

#### **Step 2: Transfer the Dump File**

Use FTP, SCP, or another method to move the .sql file to the new host.

#### **Step 3: Import the Database into the Remote MySQL Server**

mysql -u root -p < my\_database.sql

## ****📌 Summary****

| **Task** | **Method** |
| --- | --- |
| **Exporting Data to a File** | SELECT ... INTO OUTFILE |
| **Exporting a Table** | mysqldump -u root -p my\_database users > users.sql |
| **Exporting a Database** | mysqldump -u root -p --databases my\_database > my\_database.sql |
| **Copying Data to Another Host** | Transfer .sql file using scp, FTP, or a cloud service |

**Importing Data in MySQL Workbench**

This guide covers three key ways to import data into MySQL:

1. **Using LOAD DATA INFILE**
2. **Using mysqlimport Command**
3. **Handling Quotes & Special Characters**

## ****1. Importing Data with**** LOAD DATA INFILE

LOAD DATA INFILE is a fast way to import data from a CSV or text file.

### ****🔹 Example: Import**** users.csv ****into a Table****

#### **Step 1: Create a Table**

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(255),

age INT

);

#### **Step 2: Sample** users.csv **File**

1, "John Doe", "john@example.com", 28

2, "Alice Smith", "alice@example.com", 32

3, "Bob Johnson", "bob@example.com", 25

#### **Step 3: Import the CSV File**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv'

INTO TABLE users

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

✅ **Imports the data into users table.**  
⚠ **Make sure secure\_file\_priv allows file import.**

SHOW VARIABLES LIKE 'secure\_file\_priv';

## ****2. Importing Data with**** mysqlimport

mysqlimport is a command-line tool for bulk data import.

### ****🔹 Syntax****

mysqlimport -u root -p --local my\_database users.csv

* **--local** → Allows local file import
* **my\_database** → Target database
* **users.csv** → CSV file to import

### ****🔹 Example: Import**** users.csv ****Using**** mysqlimport

mysqlimport -u root -p --local --fields-terminated-by=',' --fields-enclosed-by='"' my\_database users.csv

✅ **Same effect as LOAD DATA INFILE but from the command line.**

## ****3. Handling Quotes & Special Characters****

Sometimes, data contains **quotes (")**, **commas (,)**, or **newlines (\n)**, which need special handling.

### ****🔹 Example: Data with Special Characters (****users\_quotes.csv****)****

1, "John, Doe", "john@example.com", 28

2, "Alice \"The Genius\" Smith", "alice@example.com", 32

3, "Bob\nJohnson", "bob@example.com", 25

### ****🔹 Import with Proper Handling****

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users\_quotes.csv'

INTO TABLE users

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

✅ **Handles commas and quotes correctly.**

### ****Fixing Issues with Special Characters****

| **Issue** | **Solution** |
| --- | --- |
| **Double Quotes in Text** ("Alice "The Genius" Smith") | Use **ESCAPED BY '\'** in the query. |
| **Commas Inside a Name** ("John, Doe") | Ensure **ENCLOSED BY '"'** is used. |
| **Newlines in Fields** | Save as a **properly formatted CSV**, or replace \n with spaces. |

## ****📌 Summary****

| **Method** | **Best For** | **Command** |
| --- | --- | --- |
| LOAD DATA INFILE | Fast CSV Import | LOAD DATA INFILE 'file.csv' INTO TABLE users |
| mysqlimport | Command-line Import | mysqlimport -u root -p --local my\_database users.csv |
| Handling Quotes | Handling commas & newlines | Use ENCLOSED BY '"' ESCAPED BY '\\' |