MySql

By

Tejaswini Apte

MySql- DataTypes- Major Data Types

- Numeric Data Types
- Date and Time Data Types
- String Data Types

- Ref: https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Numeric Data Types

- Integer Types - Required Storage and Range for Integer Types Supported by MySQL

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^{63}	0	$2^{63}-1$	$2^{64}-1$

Ref: https://dev.mysql.com/doc/refman/8.0/en/integer-types.html

Fixed Point Types

- Decimal
- Numeric
- The DECIMAL and NUMERIC types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, NUMERIC is implemented as DECIMAL, so the following remarks about DECIMAL apply equally to NUMERIC. In a DECIMAL column declaration, the precision and scale can be (and usually is) specified. For example:

  salary DECIMAL(5,2)
- In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.
- Standard SQL requires that DECIMAL(5,2) be able to store any value with five digits and two decimals, so values that can be stored in the salary column range from -999.99 to 999.99.

Ref: https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html

Floating Point Types

- The FLOAT and DOUBLE types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

- For FLOAT, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword FLOAT in parentheses, that is, FLOAT(p). MySQL also supports this optional precision specification, but the precision value in FLOAT(p) is used only to determine storage size. A precision from 0 to 23 results in a 4-byte single-precision FLOAT column. A precision from 24 to 53 results in an 8-byte double-precision DOUBLE column.

Ref: https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html

Date and Time Data Types

- The date and time data types for representing temporal values are DATE, TIME, DATETIME, TIMESTAMP, and YEAR.

- DATE
- A date. The supported range is '1000-01-01' to '9999-12-31'. MySQL displays DATE values in 'YYYY-MM-DD' format, but permits assignment of values to DATE columns using either strings or numbers.

- DATETIME[(fsp)]
- A date and time combination. The supported range is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.499999'. MySQL displays DATETIME values in 'YYYY-MM-DD hh:mm:ss[.fraction]' format, but permits assignment of values to DATETIME columns using either strings or numbers.

- TIMESTAMP[(fsp)]
- A timestamp. The range is '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.499999' UTC. TIMESTAMP values are stored as the number of seconds since the epoch ('1970-01-01 00:00:00' UTC). A TIMESTAMP cannot represent the value '1970-01-01 00:00:00' because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing '0000-00-00 00:00:00', the “zero” TIMESTAMP value.

Date and Time Data Types

- TIME[(fsp)]
- A time. The range is '-838:59:59.000000' to '838:59:59.000000'. MySQL displays TIME values in 'hh:mm:ss[.fraction]' format, but permits assignment of values to TIME columns using either strings or numbers.

- YEAR[(4)]
- A year in 4-digit format. MySQL displays YEAR values in YYYY format, but permits assignment of values to YEAR columns using either strings or numbers. Values display as 1901 to 2155, or 0000.

- Ref:https://dev.mysql.com/doc/refman/8.0/en/date-and-time-type-syntax.html

String Data Types

- The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained. The CHAR and VARCHAR types are declared with a length that indicates the maximum number of characters you want to store. For example, CHAR(30) can hold up to 30 characters.

- The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed unless the PAD_CHAR_TO_FULL_LENGTH SQL mode is enabled. Values in VARCHAR columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a VARCHAR is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used.

Group Of SQL Statements

- DDL- Data Definition Language- Create, Alter, Drop, Truncate
- DML- Data Manipulation Language- Insert, Update, Delete,Merge
- DRL- Data Retrieval Language- Select
- DCL- Data Control Language- Grant,Revoke
- TCL- Transaction Control Language- Commit, Rollback

Create Database

- CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database.
  - CREATE DATABASE menagerie;

- Under Unix, database names are case-sensitive (unlike SQL keywords), so you must always refer to your database as menagerie, not as Menagerie, MENAGERIE, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query.
- Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this statement:
  - mysql> USE menagerie
  - Database changed
- Your database needs to be created only once, but you must select it for use each time you begin a mysql session. You can do this by issuing a USE statement as shown in the example. Alternatively, you can select the database on the command line when you invoke mysql. Just specify its name after any connection parameters that you might need to provide.

- Ref:https://dev.mysql.com/doc/refman/8.0/en/creating-database.html

Create Table

- Creating the database is the easy part, but at this point it is empty, as SHOW TABLES tells you:

- mysql> SHOW TABLES;
- Empty set (0.00 sec)

- CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
- DESCRIBE pet;
- INSERT INTO pet VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL); // DML statement inserting the values

- Ref:https://dev.mysql.com/doc/refman/8.0/en/creating-tables.html

Create Table- Auto Increment

- The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows:
- CREATE TABLE animals (id MEDIUMINT NOT NULL AUTO_INCREMENT, name CHAR(30) NOT NULL, PRIMARY KEY (id));
- INSERT INTO animals (name) VALUES ('dog'),('cat'),('penguin'), ('lax'),('whale'),('ostrich');
- SELECT * FROM animals;
- No value was specified for the AUTO_INCREMENT column, so MySQL assigned sequence numbers automatically. You can also explicitly assign 0 to the column to generate sequence numbers, unless the NO_AUTO_VALUE_ON_ZERO SQL mode is enabled. For example:
- INSERT INTO animals (id,name) VALUES(0,'groundhog');
- If the column is declared NOT NULL, it is also possible to assign NULL to the column to generate sequence numbers. For example:
- INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
- When you insert any other value into an AUTO_INCREMENT column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the largest column value. For example:

Create Table- Auto Increment

- To start with an AUTO_INCREMENT value other than 1, set that value with CREATE TABLE or ALTER TABLE, like this:
- mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;

- Ref: https://dev.mysql.com/doc/refman/8.0/en/example-auto-increment.html

MySQL- Copying table from Another Table

- In MySQL, the easiest way to copy a table with its data between two databases is to use the CREATE TABLE AS statement, but note, that you need to provide the target database name as a table prefix.

- CREATE TABLE new-database-name.new-table-name
- AS
- SELECT * FROM old-database.old-table-name;

- https://www.devart.com/dbforge/mysql/studio/mysql-copy-table.html#:~:text=In%20MySQL%2C%20the%20easiest%20way,SELECT%20*%20FROM%20old%2Ddatabase.

MySQL- Copying Data from another Table

- After creating a destination table like the source table. Now copy all the data from the source table to the destination table.
- insert into destination_table select * from source_table // copy all data
- insert into destination_table select * from source_table where city='New York‘ // copy specific rows
- insert into destination_table_new (address,city,pincode) select address,city,pincode from source_table; // copy based on selected columns

Insert Statement

- INSERT INTO tbl_name () VALUES();
- An expression expr can refer to any column that was set earlier in a value list. For example, you can do this because the value for col2 refers to col1, which has previously been assigned:
    - INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
- But the following is not legal, because the value for col1 refers to col2, which is assigned after col1:
    - INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
- INSERT statements that use VALUES syntax can insert multiple rows. To do this, include multiple lists of comma-separated column values, with lists enclosed within parentheses and separated by commas. Example:
    - INSERT INTO tbl_name (a,b,c) VALUES(1,2,3), (4,5,6), (7,8,9);
- Each values list must contain exactly as many values as are to be inserted per row. The following statement is invalid because it contains one list of nine values, rather than three lists of three values each:
    - INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
- VALUE is a synonym for VALUES in this context. Neither implies anything about the number of values lists, nor about the number of values per list. Either may be used whether there is a single values list or multiple lists, and regardless of the number of values per list.
- Ref: https://dev.mysql.com/doc/refman/8.0/en/insert.html

Delete Statement

- DELETE is a DML statement that removes rows from a table.

- The conditions in the optional WHERE clause identify which rows to delete. With no WHERE clause, all rows are deleted.

- where_condition is an expression that evaluates to true for each row to be deleted

- DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name [[AS] tbl_alias]

[PARTITION (partition_name [, partition_name] ...)]

[WHERE where_condition]

[ORDER BY ...]

[LIMIT row_count]

Ref:https://dev.mysql.com/doc/refman/8.0/en/delete.html

Update Statement

- UPDATE is a DML statement that modifies rows in a table.
- For the single-table syntax, the UPDATE statement updates columns of existing rows in the named table with new values. The SET clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword DEFAULT to set a column explicitly to its default value. The WHERE clause, if given, specifies the conditions that identify which rows to update. With no WHERE clause, all rows are updated.
  - UPDATE [LOW_PRIORITY] [IGNORE]
  - *table_reference* SET *assignment_list*
  - [WHERE *where_condition*] [ORDER BY ...] [LIMIT *row_count*]

- Ref: https://dev.mysql.com/doc/refman/8.0/en/update.html

Alter Table

- ALTER TABLE changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself.
- ALTER TABLE *tbl_name* [*alter_option* [, *alter_option*] ...] [*partition_options*]
- Options are:
- ADD [COLUMN] (*col_name column_definition*,...)
- ADD [CONSTRAINT [*symbol*]] PRIMARY KEY
- DROP {CHECK | CONSTRAINT} *symbol*
- ALTER [COLUMN] *col_name* { SET DEFAULT {*literal* | (*expr*)} | SET {VISIBLE | INVISIBLE} | DROP DEFAULT }
- ALTER INDEX index_name {VISIBLE | INVISIBLE}

Alter Table

- | CHANGE [COLUMN] old_col_name new_col_name column_definition
- [FIRST | AFTER col_name]
- | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
- | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
- | {DISABLE | ENABLE} KEYS
- | {DISCARD | IMPORT} TABLESPACE
- | DROP [COLUMN] col_name
- | DROP {INDEX | KEY} index_name
- | DROP PRIMARY KEY
- | DROP FOREIGN KEY fk_symbol

Alter Table

- MODIFY [COLUMN] col_name column_definition
- [FIRST | AFTER col_name]
- | ORDER BY col_name [, col_name] ...
- | RENAME COLUMN old_col_name TO new_col_name
- | RENAME {INDEX | KEY} old_index_name TO new_index_name
- | RENAME [TO | AS] new_tbl_name

- Ref: https://dev.mysql.com/doc/refman/8.0/en/alter-table.html

Drop Table Statement

- DROP TABLE removes one or more tables. You must have the DROP privilege for each table.

- Be careful with this statement! For each table, it removes the table definition and all table data. If the table is partitioned, the statement removes the table definition, all its partitions, all data stored in those partitions, and all partition definitions associated with the dropped table.
- Dropping a table also drops any triggers for the table.

- DROP TABLE causes an implicit commit, except when used with the TEMPORARY keyword.

- Ref:https://dev.mysql.com/doc/refman/8.0/en/drop-table.html

Data Retrieval Language(DRL) – Select Statements

- SELECT is used to retrieve rows selected from one or more tables, and can include UNION operations and subqueries.
    - mysql> SELECT 1 + 1;
    - -> 2
- You are permitted to specify DUAL as a dummy table name in situations where no tables are referenced:
    - mysql> SELECT 1 + 1 FROM DUAL;
    - -> 2
- DUAL is purely for the convenience of people who require that all SELECT statements should have FROM and possibly other clauses. MySQL may ignore the clauses. MySQL does not require FROM DUAL if no tables are referenced.
- Select '*' from <table name> // Select All data
- Select col1,col2 from <table name> // Select Particular Column
- Select col1,col2 from <table name> where <condition> // Selecting particular row
- Ref: https://dev.mysql.com/doc/refman/8.0/en/select.html

Order By Clause

- The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the DESC keyword to the name of the column you are sorting by:

- SELECT name, birth FROM pet ORDER BY birth;
- SELECT name, birth FROM pet ORDER BY birth DESC;

Date Calculation

- To determine how many years old each of your pets is, use the TIMESTAMPDIFF() function. Its arguments are the unit in which you want the result expressed, and the two dates for which to take the difference. The following query shows, for each pet, the birth date, the current date, and the age in years. An alias (age) is used to make the final output column label more meaningful.

- SELECT name, birth, CURDATE(), TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age FROM pet;
- MySQL provides several functions for extracting parts of dates, such as YEAR(), MONTH(), and DAYOFMONTH(). MONTH() is the appropriate function here. To see how it works, run a simple query that displays the value of both birth and MONTH(birth):
- SELECT name, birth, MONTH(birth) FROM pet;
- SELECT name, birth FROM pet WHERE MONTH(birth) = 5;

- Ref: https://dev.mysql.com/doc/refman/8.0/en/date-calculations.html

Date Calculation

- DATE_ADD() enables you to add a time interval to a given date. If you add a month to the value of CURDATE(), then extract the month part with MONTH(), the result produces the month in which to look for birthdays:
  - mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
- A different way to accomplish the same task is to add 1 to get the next month after the current one after using the modulo function (MOD) to wrap the month value to 0 if it is currently 12:
  - mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;

- Ref: https://dev.mysql.com/doc/refman/8.0/en/date-calculations.html

Working with NULLs

- The NULL value can be surprising until you get used to it. Conceptually, NULL means “a missing unknown value” and it is treated somewhat differently from other values.

- To test for NULL, use the IS NULL and IS NOT NULL operators, as shown here:

- mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
- Because the result of any arithmetic comparison with NULL is also NULL, you cannot obtain any meaningful results from such comparisons.

- Ref: https://dev.mysql.com/doc/refman/8.0/en/working-with-null.html

Pattern Matching

- SQL pattern matching enables you to use _ to match any single character and % to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Do not use = or <> when you use SQL patterns. Use the LIKE or NOT LIKE comparison operators instead.

- To find names beginning with b:
  - mysql> SELECT * FROM pet WHERE name LIKE 'b%';
- To find names ending with fy:
  - mysql> SELECT * FROM pet WHERE name LIKE '%fy';
- To find names containing a w:
  - mysql> SELECT * FROM pet WHERE name LIKE '%w%';
- To find names containing exactly five characters, use five instances of the _ pattern character:
  - mysql> SELECT * FROM pet WHERE name LIKE '_____';

Ref: https://dev.mysql.com/doc/refman/8.0/en/pattern-matching.html

Aggregate Functions

- Aggregate functions that operate on sets of values. They are often used with a GROUP BY clause to group values into subsets; aggregate functions ignore NULL values. If you use an aggregate function in a statement containing no GROUP BY clause, it is equivalent to grouping on all rows.
- AVG([DISTINCT] expr) [over_clause]
- Returns the average value of expr. The DISTINCT option can be used to return the average of the distinct values of expr.
- If there are no matching rows, AVG() returns NULL. The function also returns NULL if expr is NULL.
  - mysql> SELECT student_name, AVG(test_score) FROM student GROUP BY student_name;
- COUNT(expr) [over_clause]
- Returns a count of the number of non-NULL values of expr in the rows retrieved by a SELECT statement. If there are no matching rows, COUNT() returns 0. COUNT(NULL) returns 0.
  - SELECT student.student_name,COUNT(*) FROM student,course WHERE student.student_id=course.student_id GROUP BY student_name;
- COUNT(*) is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain NULL values.
- Ref: https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_count

Aggregate Functions

- MAX([DISTINCT] expr) [over_clause]; Returns the maximum value of expr.
- MIN([DISTINCT] expr) [over_clause];Returns the minimum value of expr
  - SELECT student_name, MIN(test_score), MAX(test_score) FROM student GROUP BY student_name;.

- SUM([DISTINCT] expr) [over_clause]

  - Returns the sum of expr. If the return set has no rows, SUM() returns NULL. The DISTINCT keyword can be used to sum only the distinct values of expr.
  - If there are no matching rows, or if expr is NULL, SUM() returns NULL.

- Ref: https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html#function_sum

Group By Clause

- To summarize table contents per year, use a simple GROUP BY like this:
- mysql> SELECT year, SUM(profit) AS profit FROM sales GROUP BY year;

- Ref: https://dev.mysql.com/doc/refman/8.0/en/group-by-modifiers.html

MySQL-Joins

- Basically, MySQL join is a method of linking data from two or more tables based on the values of the common column between them, with the result being a new temporary table. In other words, we can say that a MySQL join enables us to retrieve records from two or more logically related tables in a new temporary table. The new temporary table is created based on the column(s) that the two tables share, which represents meaningful column(s) of comparison.

- The common values are usually the same column name and data-type that appears in both the participating table being joined. These column(s) are called the join key or common key.

- Ref: https://www.tutorialspoint.com/What-are-MySQL-joins#:~:text=In%20other%20words%2C%20we%20can,column(s)%20of%20comparison.

MySQL-Joins

- Cross Join-Actually, a cross join is the basic form of a join. If we have two tables, it takes each row of table1 and appends it to each row of table2. Hence if table1 has 3 rows and table2 has 2 rows then we will get total 6 rows after cross joining these tables.
- mysql> Select tbl_1.id, tbl_2.id FROM tbl_1 CROSS JOIN tbl_2;

- Ref:https://www.tutorialspoint.com/What-are-MySQL-joins#:~:text=In%20other%20words%2C%20we%20can,column(s)%20of%20comparison.

MySQL-Joins

- **Inner Join or Equi join -**To form an inner join we need to specify a particular condition which is known as join-predicate. Actually inner or equi join need rows in the two joined tables to have matching columns values. To understand it, following query will inner join the tables named ‘tbl_1’ and ‘tbl_2’.
  - SELECT tbl_1.id,tbl_2.id FROM tbl_1 INNER JOIN tbl_2 ON tbl_1.name = tbl_2.name;

- **Left Join**
  - SELECT tbl_1.id,tbl_2.id FROM tbl_1 LEFT JOIN tbl_2 ON tbl_1.name = tbl_2.name;

- **Right Join**
  - SELECT tbl_1.id,tbl_2.id FROM tbl_1 RIGHT JOIN tbl_2 ON tbl_1.name = tbl_2.name;