Wiki - 304 - Relational Databases and SQL

## Data Modeling

Data modeling is a technique used to document a software system with diagrams and symbols. It is used to represent the communication of data.

The highest level of abstraction for the data model is called the Entity Relationship Diagram (ERD). It is a graphical representation of data requirements for a database.

## Entity Relationship Diagram

The main value of carefully constructing an ERD is that it can readily be converted into a database structure.

There are three components in the ERD.

* Entities: Number of tables you need for your database.
* Attributes: Information such as properties and facts that you need to describe each table.
* Relationships: How tables are linked together.

## Entity

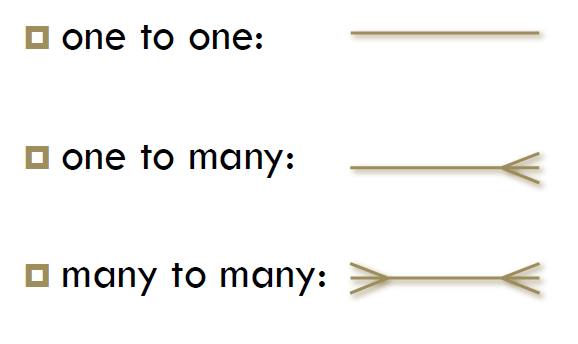
Entities are the basic objects of ERDs. These are the tables of your database. Entities are nouns, and the types usually fall into five classes: *concepts, locations, roles, events,* or *things*. For example, the things could include students, courses, books, campuses, employees, payments, and projects.

A specific example of an entity is called an instance. Each instance becomes a record or a row in a table. For example, the student John Smith is a record in a table called *students*.

## Relationships

Relationships are the associations between entities. Verbs often describe relationships between entities, which are represented by a diamond shape.

We use Crow's Foot Symbols to represent the relationships. Three types of relationships are discussed in this lab. If you read or hear cardinality ratios, it also refers to types of relationships.



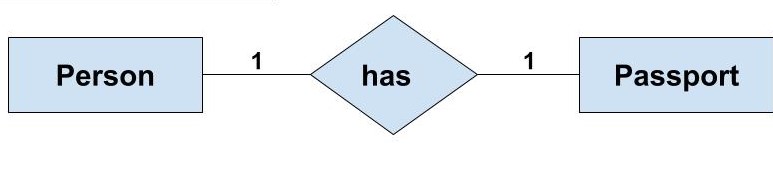
The detailed Crow's Foot Relationship symbols can be found here. [Crow's Foot Relationship Symbols](http://www.vivekmchawla.com/erd-crows-foot-relationship-symbols-cheat-sheet/)

#### **One-to-One Relationship (1:1)**

A single entity instance in one entity class is related to a single entity instance in another entity class.

**For example:**

* Each student fills one seat, and one seat is assigned to only one student.
* Each professor has one office space.
* If there are two entities, such as ‘Person’ (Id, Name, Age, Address) and ‘Passport’(Passport\_id, Passport\_no), each person can have only one passport, and each passport belongs to only one person.

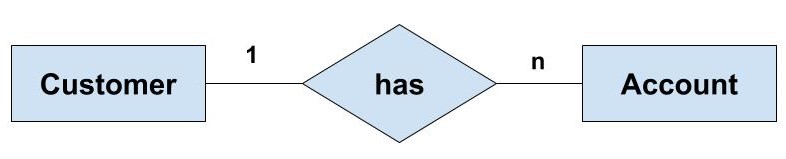


#### **One to Many Relationship (1:M)**

A single entity instance in one entity class (parent) is related to multiple entity instances in another entity class (child).

**For example:**

* One instructor can teach many courses, but one course can only be taught by one instructor.
* One instructor may teach many students in one class, but all the students have one instructor for that class.
* If there are two entity types, such as ‘Customer’ and ‘Account,’ each ‘Customer’ can have more than one ‘Account,’ but each ‘Account’ is held by only one ‘Customer.’ In this example, we can say that each Customer is associated with many Accounts. So, it is a one-to-many relationship. But, if we see it the other way (i.e., many Account(s) are associated with one Customer), we can say that it is a many-to-one relationship.

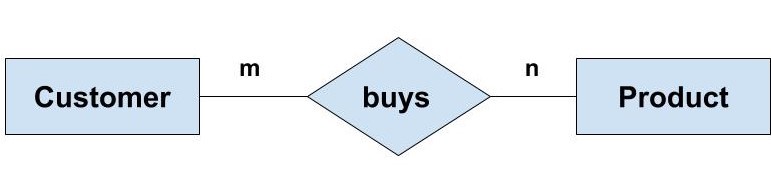


#### **Many to Many Relationship (M:M)**

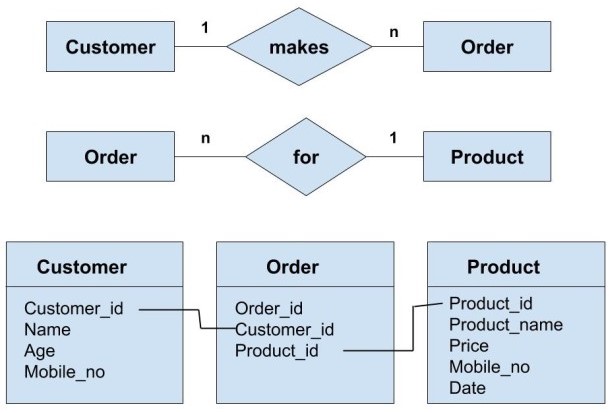
Each entity instance in one entity class is related to multiple entity instances in another entity class, and vice versa.

**For example:**

* Each student can take many classes, and each class can be taken by many students.
* Each consumer can buy many products, and each product can be bought by many consumers.
* If there are two entity types, such as ‘Customer’ and ‘Product. each customer can buy more than one product, and a product can be bought by many different customers.



* Now, to understand the concept of the linking table here, we can have the ‘Order’ entity as a linking table that links the ‘Customer’ and ‘Product’ entities. We can break this many-to-many relationship into two one-to-many relationships.
* First, each ‘Customer’ can have many ‘Order(s) whereas each ‘Order’ is related to only one ‘Customer’. Second, each ‘Order’ is related to only one Product where there can be many orders for the same Product.



* The above concept of linking can be understood with the help of taking into consideration all of the attributes of the entities 'Customer,' 'Order,' and 'Product'. We can see that the primary key of both the 'Customer' and 'Product' entities are included in the linking table (i.e., 'Order' table). These keys act as foreign keys while referring to the respective table from the 'Order' table.

Many to many relationships are difficult to represent. We must decompose a many-to-many (M:M) relationship into two one-to-many (1:M) relationships.

## Attributes.

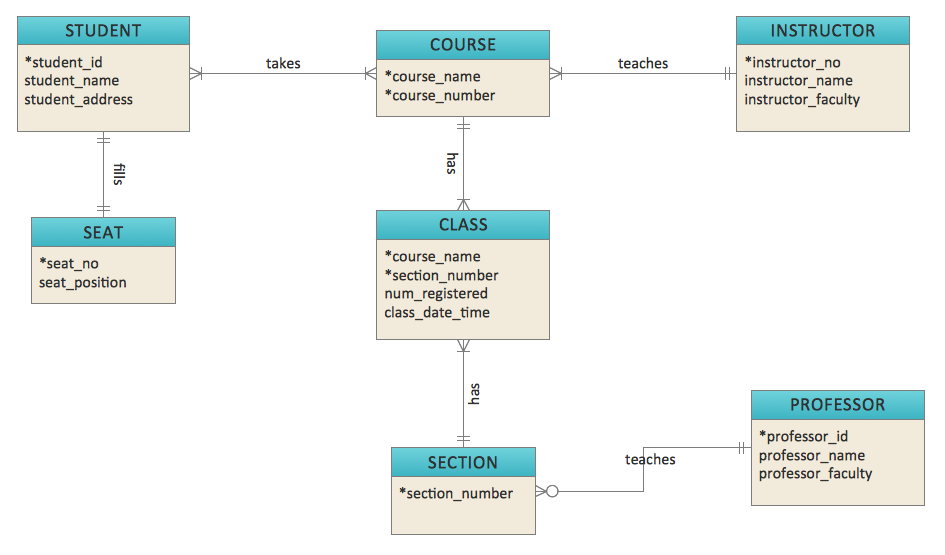
Attributes are facts or descriptions of entities. They are also often nouns and become the columns of the table. For example, for an entity student, the attributes can be first name, last name, email, address, and phone number.

#### **Primary Key**

The Primary Key\* or identifier is an attribute or a set of attributes that uniquely identifies an instance of the entity. For example, for a student entity, the student number is the primary key since no two students have the same student number. We can only have one primary key in a table. It uniquely identifies every row, and it cannot be null.

#### **Foreign key**

A foreign key+ (sometimes called a referencing key) is a key used to link two tables together. Typically, you take the primary key field from one table and insert it into the other table, where it becomes a foreign key (it remains a primary key in the original table). We can have more than one foreign key in a table.



## 

## A Brief History of SQL

* 1970 − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
* 1974 − Structured Query Language appeared.
* 1978 − IBM worked to develop Codd's ideas and released a product named System/R.
* 1986 − IBM developed the first prototype of a relational database and standardized by ANSI. The first relational database was released by Relational Software, which later came to be known as Oracle.

# MySQL Transaction Statements

MySQL provides us with the following important statements to control transactions:

* To start a transaction, you use the START TRANSACTION statement. The BEGIN or BEGIN WORK are the aliases of the START TRANSACTION.
* To commit the current transaction and make its changes permanent, you use the COMMIT statement.
* To roll back the current transaction and cancel its changes, you use the ROLLBACK statement.
* To disable or enable the auto-commit mode for the current transaction, you use the SET autocommit statement.

By default, MySQL automatically commits the changes permanently to the database. To force MySQL not to commit changes automatically, you use the following statement:

SET autocommit = 0;

Or

SET autocommit = OFF

You use the following statement to enable the autocommit mode explicitly:

SET autocommit = 1;

Or

SET autocommit = ON;

# Understanding MySQL Storage Engines

MySQL provides various storage engines for its tables:

* MyISAM
* InnoDB
* MERGE
* MEMORY (HEAP)
* ARCHIVE
* CSV
* FEDERATED

Each storage engine has its own advantages and disadvantages. It is crucial to understand each storage engine's feature, and choose the most appropriate one for your tables to maximize the performance of the database. In the following sections, we will discuss each storage engine and its features so that you can decide which one to use.

## MyISAM

MyISAM extends the former ISAM storage engine. The MyISAM tables are optimized for compression and speed. MyISAM tables are also portable between platforms and operating systems.

The size of the MyISAM table can be up to 256TB, which is huge. In addition, MyISAM tables can be compressed into read-only tables to save space. At startup, MySQL checks MyISAM tables for corruption and even repairs them in the case of errors. The MyISAM tables are not transaction-safe.

Before the MySQL version 5.5, MyISAM was the default storage engine for creating a table without specifying the storage engine explicitly. From version 5.5, MySQL uses InnoDB as the default storage engine.

## InnoDB

The InnoDB tables fully support ACID-compliant [transactions](https://www.mysqltutorial.org/mysql-transaction.aspx). They are also optimal for performance. The InnoDB table supports [foreign keys](https://www.mysqltutorial.org/mysql-foreign-key/), commit, rollback, and roll-forward operations. The size of an InnoDB table can be up to 64TB.

Like MyISAM, the InnoDB tables are portable between different platforms and operating systems. MySQL also checks and repairs InnoDB tables, if necessary, at startup.

## MERGE

A MERGE table is a virtual table that combines multiple MyISAM tables that have a similar structure to one table.

The MERGE storage engine is also known as the MRG\_MyISAM engine. The MERGE table does not have its own indexes; it uses indexes of the component tables instead.

Using a MERGE table, you can speed up performance when [joining multiple tables](https://www.mysqltutorial.org/mysql-inner-join.aspx). MySQL only allows you to perform [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx), [DELETE](https://www.mysqltutorial.org/mysql-delete-statement.aspx), [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx) and [INSERT](https://www.mysqltutorial.org/mysql-insert-statement.aspx) operations on the MERGE tables. If you use a DROP [TABLE](https://www.mysqltutorial.org/mysql-drop-table) statement on a MERGE table, only MERGE specifications are removed. The underlying tables will not be affected.

## Memory

The memory tables are stored in memory and use hash indexes so that they are faster than MyISAM tables. The lifetime of the data of the memory tables depends on the uptime of the database server. The memory storage engine is formerly known as HEAP.

## Archive

The archive storage engine allows you to store a large number of records, which for archiving purposes, change into a compressed format to save disk space. The archive storage engine compresses a record when it is inserted and decompresses it using the *zlib* library as it is read.

The archive tables only allow [INSERT](https://www.mysqltutorial.org/mysql-insert-statement.aspx) and [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statements. The ARCHIVE tables do not support indexes, so it requires a full table scanning for reading rows.

## CSV

The CSV storage engine stores data in a comma-separated values (CSV) file format. A CSV table brings a convenient way to migrate data into non-SQL applications such as spreadsheet software.

The CSV table does not support NULL data types. In addition, the read operation requires a full table scan.

## FEDERATED

The FEDERATED storage engine allows you to manage data from a remote MySQL server without using the cluster or replication technology. The local federated table stores no data. When you query data from a local federated table, the data is pulled automatically from the remote federated tables.

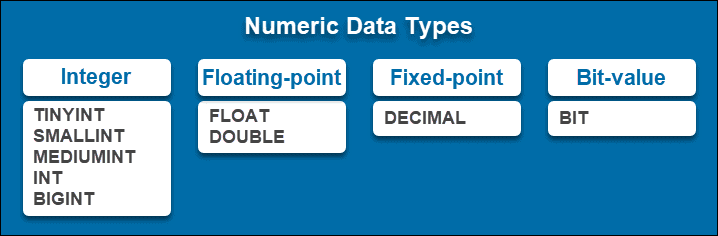
# 

# Numeric Data Types

When storing numbers in a database column, use one of the numeric data types. MySQL supports both exact and approximate numeric data types.

The numeric category is further subdivided into the following groups:

* Integer data types.
* Floating-point data types.
* Fixed-point data types.
* Bit-value data types.



## Integer Types

Integer data types are used for whole numbers (integers). They include both positive and negative values. However, they do not handle fractional numbers. Therefore, numbers such as 30 and -5435 can be stored as integer data types, while numbers such as 5,3 or 1/5 cannot.

Integer types are signed or unsigned. They are further subdivided based on their size, differing by their length and range.

|  | **Bytes** | **Range (unsigned)** | **Range (signed)** |
| --- | --- | --- | --- |
| TINYINT | 1 | from 0 to 255 | from -128 to 127 |
| SMALLINT | 2 | from 0 to 65535 | from -32768 to 32767 |
| MEDIUMINT | 3 | from 0 to 16777215 | from -8388608 to 8388607 |
| INT | 4 | from 0 to 4294967295 | from -2147483648 to 2147483647 |
| BIGINT | 8 | from 0 to 18446744073709551615 | from -9223372036854775808 to 9223372036854775807 |

TINYINT is a very small integer that uses 1 byte of storage. It consists of up to 4 digits. Its unsigned range is from 0 to 255. If it is signed, it has a range from -128 to 127.

SMALLINT is a small integer that uses 2 bytes of storage. It consists of up to 5 digits. Its unsigned range is from 0 to 65535. When signed, it has a range from -32768 to 32767.

MEDIUMINT is a medium-sized integer that uses 3 bytes of storage. It has up to 9 digits. If unsigned, it has a range from 0 to 16777215. Signed, its minimum value is -8388608, while its maximum value is 8388607.

INT is an integer that uses 4 bytes of storage. It uses up to 11 digits. When unsigned, it ranges from 0 to 4294967295. If it is signed, it has the range from -2147483648 to 2147483647.

BIGINT is a big integer that uses 8 bytes of storage. It has up to 20 digits. Its minimum signed value is 0, while its maximum signed value is 18446744073709551615. If signed, it has a range from -9223372036854775808 to 9223372036854775807.

## 

## Floating-Point Types

Floating-point numeric data types are rational numbers used for representing approximate values. Use floating-point data types for high-precision calculations.

Floating-point types include:

* FLOAT represents single-precision values that use 4 bytes and include up to 6 or 7 significant digits.
* DOUBLE represent double-precision values that use 8 bytes and include up to 15 or 16 significant digits.

The basic syntax for defining FLOAT/DOUBLE data types is FLOAT(M,D)/DOUBLE(M,D).

M represents the total number of digits, while D is the number of decimals. For example, the value 5143,234 would be defined as 7,3, as it has 7 digits in total and 3 digits after the decimal point.

You can also use the FLOAT(P) syntax to specify a floating-point data type, where P specifies the precision. If P has a value from 0 to 23, it is a single-precision column. If the precision is between 24 and 53, it is a double-precision column.

|  | **Bytes** | **Range (unsigned)** | **Range (signed)** |
| --- | --- | --- | --- |
| FLOAT | 4 | from 1.175494351E-38 to 3.402823466E+38 | from -3.402823466E+38 to -1.175494351E-38 |
| DOUBLE | 8 | from 0 and 2.22507385850720 14E- 308 to 1.797693134862315 7E+ 308 | from -1.7976931348623 157E+ 308 to -2.22507385850720 14E- 308 |

**Note:** Do not use floating-point numbers when comparing values, especially when dealing with monetary data. Instead, use the DECIMAL data type for such use cases.

## Fixed-Point Types

To store exact numeric values, use the fixed-point data type – DECIMAL. As it represents an exact number, this data type is mostly used for data that relies on precise values (such as monetary data).

The basic syntax is DECIMAL(P,D), where P stands for precision (the number of significant digits) and D stands for scale (the number of digits after the decimal point).

The maximum number of digits for precision is 65, while the maximum value for scale is 30.

If you do not define the precision and scale, the column uses default values. By default, the values for P,D are 10,0.

## Bit-Value Types

The BIT data type stores binary values. When creating a column that will store such values, you define the number of bit values ranging from 1 to 64.

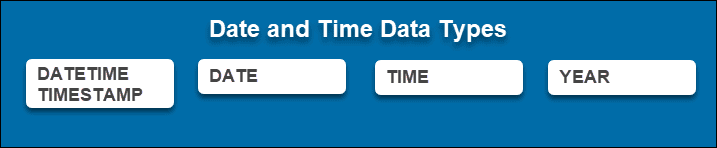
The syntax for this MySQL data type is BIT(N). If you do not specify N, the default value is 1.

## Date and Time Data Types

Date and time are commonly used data types. Whether you are storing the time of a data entry, a date of birth, or the current timestamp, you can use one of the following columns.

Date and time data types include:

* DATETIME, TIMESTAMP
* DATE
* TIME
* YEAR



#### **DATETIME, TIMESTAMP**

To store date and time values, use either DATETIME or TIMESTAMP. Both data types store information in the YYYY-MM-DD HH:MM:SS format. It includes the year, month, day, hour, minutes, and seconds.

The main difference between the two is their range:

* DATETIME values range from 1000-01-01 00:00:00 to 9999-12-31 23:59:59.
* TIMESTAMP values range from 1970-01-01 00:00:01 to 2038-01-19 03:14:07.

You can include fractional seconds for both options. To do so, specify the precision following the syntax DATETIME(p)/TIMESTAMP(p).

For example, to store the timestamp 10:53 PM on March 1st, 2021, with three fractional seconds, specify the data type TIMESTAMP(3). The entry is stored as: 2021-03-01 22:53:35.346.

#### **DATE**

DATE is used for storing date values in the format YYYY-MM-DD (year, month, and date).

The data type supports the range from 1000-01-01 to 9999-12-31.

#### **TIME**

TIME is used to store time values as HH-MM-SS (hours, minutes, seconds) or HHH-MM-SS. Entries showing elapsed time or time differences are stored and retrieved in a longer format (if they need more digits for hours).

The data type supports the range from 1000-01-01 to 9999-12-31.

#### 

#### **YEAR**

YEAR stores year values in the format YYYY. It supports values within the range 1901-2155.

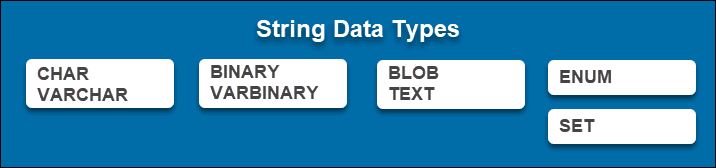
While versions older than MySQL 5.7.5 supported both 2 and 4 digit entries for YEAR, there has been no 2-digit support since the 5.7.5 release.

## String Data Types

When storing strings of data, use one of the string data types. They can contain letters, numbers, images, or files.

Accordingly, there are several different string data types:

* CHAR and VARCHAR
* BINARY and VARBINARY
* BLOB and TEXT
* ENUM
* SET



## 

## CHAR and VARCHAR

CHAR and VARCHAR are data types used to store non-binary strings. The main difference between the two is how they store data.

CHAR stores fixed-length strings (up to 255 characters). When creating a CHAR column, you specify the length using the CHAR(N) syntax. N is the number of characters you want to take up. If you do not define the length, it uses the default value 1.

These values are stored right-padded with the specified length. Therefore, if you set up a CHAR(5) column and store a three-character entry into it, it still takes up five characters.

VARCHAR stores variable-length strings. While the length has to be defined when creating a column, the values are not right-padded. They have a maximum limit, but the length is not fixed and varies depending on the data.

Before, the range of entries was from 0 to 255. After the release of MySQL 5.0.3 , VARCHAR range is up to 65,535 characters.

## BINARY and VARBINARY

BINARY and VARBINARY data types are similar to the previously mentioned CHAR and VARCHAR. The main difference between these two groups is that BINARY and VARBINARY are used for binary strings.

BINARY is used for fixed-length binary strings, up to 255 bytes. The main syntax for defining such a column is BINARY(N), where N is the number of bytes.

VARBINARY stores variable-length binary strings. MySQL version 5.0.3 and newer stores up to 65 535 bytes.

## BLOB and TEXT

Both BLOB and TEXT are used to store large amounts of data.

BLOB handles Binary Large Objects (that is, large sets of binary data such as images, audio, or PDF files).

There are 4 kinds of BLOB data types to use, depending on the size your data requires:

* TINYBLOB (0 – 255; 255 bytes)
* BLOB (0 – 65,535; 16 KB)
* MEDIUMBLOB (0 – 16,777,215; 16 MB)
* LONGBLOB (0 – 4,294,967,295; 4 GB)

TEXT data types are for storing longer strings of text. According to the amount of data required, there is:

* TINYTEXT (0 – 255; 255 bytes)
* TEXT (0 – 65,535; 16 KB)
* MEDIUMTEXT (0 – 16,777,215; 16 MB)
* LONGTEXT (0 – 4,294,967,295; 4 GB)

## 

## ENUM

The ENUM data type is used to store one of the predefined possible values in a column. The column can have up to 65535 distinct values.

When creating an ENUM table column in MySQL, you specify a list of all the permitted values.

## SET

Like ENUM, the SET data type has a predefined list of possible values stored in the column.

The main difference between the two is that SET allows an entry to have more than one value.

For example, if the column is defined as SET('Red','Orange','Yellow','Green') and has four possible values in the list, one entry could have the value ‘Red’, while another could have the value ‘Red’,’Yellow’.

The maximum number of permitted values is 64.

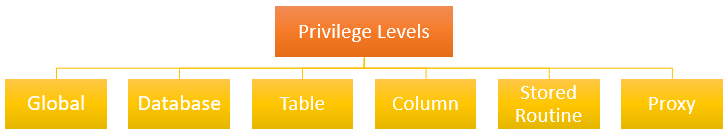
## JSON Data Types

Since version 5.7.8, MySQL included support for the native JSON data type, allowing users to store and manage JSON documents through a database.

MySQL makes sure that the JSON documents are valid and stores them in the JSON column.

# MySQL privilege levels

MySQL supports the following main privilege levels:



Global privileges apply to all databases in a MySQL Server. To assign **global privileges**, you use the **\*.\*** syntax. For example:

| GRANT SELECT  ON \*.\*  TO testUser; |
| --- |

The account user ***testUser*** can query data from all tables in all databases of the current MySQL Server.

Database privileges apply to all objects in a database. To assign **database-level** privileges, you use the ON database\_name.\* syntax. For example:

| GRANT INSERT  ON classicmodels.\*  TO testUser; |
| --- |

In the above example, testUser can insert data into all tables in the classicmodels database.

Table privileges apply to all columns in a table. To assign **table-level** privileges, you use the ON database\_name.table\_name syntax. For example:

| GRANT DELETE  ON classicmodels.employees  TO testUser; |
| --- |

In the above example, testUser can delete rows from the table employees in the database classicmodels.

If you skip the database name, MySQL uses the default database or issues an error if there is no default database.

**Column privileges** apply to single columns in a table. You must specify the column or columns for each privilege. For example:

| GRANT  SELECT (employeeNumner,lastName, firstName,email),  UPDATE(lastName)  ON employees  TO bob@localhost; |
| --- |

In this example, testUser can select data from four columns: employeeNumber, lastName, firstName, and email, and update only the lastName column in the employees table.

**Stored routine privileges** apply to stored procedures and stored functions. For example:

| GRANT EXECUTE  ON PROCEDURE CheckCredit  TO bob@localhost; |
| --- |

In this example, testUser can execute the stored procedure CheckCredit in the current database.

Proxy user privileges allow one user to be a proxy for another. The proxy user gets all privileges of the proxied user. For example:

| GRANT PROXY  ON root  TO testUser; |
| --- |

In this example, alice@localhost assumes all privileges of root.

Finally, specify the account name of the user that you want to grant privileges after the TO keyword.

Notice that in order to use the GRANT statement, you must have the GRANT OPTION privilege and the privileges that you are granting. If the read\_only system variable is enabled, you need to have the SUPER privilege to execute the GRANT statement.

## Permissible privileges for GRANT statement

The following table illustrates all the permissible privileges that you can use for the **GRANT** and **REVOKE** statements:

| **Privilege** | **Meaning** | **Level** |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Global | Database | Table | Column | Stored Routine | Proxy |  |  |
| ALL [PRIVILEGES] | Grant all privileges at specified access level except GRANT OPTION |  |  |  |  |  |  |
| ALTER | Allow user to use of [ALTER TABLE](https://www.mysqltutorial.org/mysql-alter-table.aspx)statement | X | X | X |  |  |  |
| ALTER ROUTINE | Allow users to alter and drop stored procedures or stored functions. | X | X |  |  | X |  |
| CREATE | Allow user to create databases and tables | X | X | X |  |  |  |
| CREATE ROUTINE | Allow user to create stored procedures and stored functions | X | X |  |  |  |  |
| CREATE TABLESPACE | Allow user to create, alter or drop tablespaces and log file groups | X |  |  |  |  |  |
| CREATE TEMPORARY TABLES | Allow a user to create a temporary table by using the CREATE TEMPORARY TABLE statement. | X | X |  |  |  |  |
| CREATE USER | Allow user to use the CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES statements. | X |  |  |  |  |  |
| CREATE VIEW | Allow users to create or modify the view. | X | X | X |  |  |  |
| DELETE | Allow users to use DELETE statements. | X | X | X |  |  |  |
| DROP | Allow users to drop databases, tables, and views. | X | X | X |  |  |  |
| EVENT | Enable use of events for the Event Scheduler. | X | X |  |  |  |  |
| EXECUTE | Allow users to execute stored routines. | X | X | X |  |  |  |
| FILE | Allow users to read any file in the database directory. | X |  |  |  |  |  |
| GRANT OPTION | Allow users to have privileges to grant or revoke privileges from other accounts. | X | X | X |  | X | X |
| INDEX | Allow users to create or drop indexes. | X | X | X |  |  |  |
| INSERT | Allow users to use the INSERT statement. | X | X | X | X |  |  |
| LOCK TABLES | Allow users to use LOCK TABLES on tables for which you have the SELECT privilege. | X | X |  |  |  |  |
| PROCESS | Allow users to see all processes with SHOW PROCESSLIST statements. | X |  |  |  |  |  |
| PROXY | Enable user proxying. |  |  |  |  |  |  |
| REFERENCES | Allow users to create a foreign key. | X | X | X | X |  |  |
| RELOAD | Allow users to use FLUSH statements. | X |  |  |  |  |  |
| REPLICATION CLIENT | Allow users to query to see where master or slave servers are located. | X |  |  |  |  |  |
| REPLICATION SLAVE | Allow the user to use replicate slaves to read binary log events from the master. | X |  |  |  |  |  |
| SELECT | Allow users to use SELECT statements. | X | X | X | X |  |  |
| SHOW DATABASES | Allow users to show all databases. | X |  |  |  |  |  |
| SHOW VIEW | Allow users to use SHOW CREATE VIEW statements. | X | X | X |  |  |  |
| SHUTDOWN | Allow users to use mysqladmin shutdown command. | X |  |  |  |  |  |
| SUPER | Allow user to use other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin command. | X |  |  |  |  |  |
| TRIGGER | Allow users to use TRIGGER operations. | X | X | X |  |  |  |
| UPDATE | Allow users to use the UPDATE statement. | X | X | X | X |  |  |
| USAGE | Equivalent to “no privileges.” |  |  |  |  |  |  |

# Introduction to SQL UUID

UUID stands for Universally Unique IDentifier (UUID). UUID is defined based on [RFC 4122](https://tools.ietf.org/html/rfc4122), “a Universally Unique Identifier (UUID) URN Namespace).

UUID is designed as a number that is unique globally in space and time. Two UUID values are expected to be distinct, even if they are generated on two independent servers.

In SQL, a UUID value is a 128-bit number represented as a utf8 string of five hexadecimal numbers in the following format:

aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee

To generate UUID values, you use the UUID() function as follows:

UUID()

The UUID() function returns a UUID value in compliance with UUID version 1 described in the RFC 4122.

# SQL UUID vs. Auto-Increment INT as Primary Key

### **Pros**

Using UUIDfor a primary key brings the following advantages:

* UUID values are unique across tables, databases, and even servers that allow you to merge rows from different databases or distribute databases across servers.
* UUID values do not expose the information about your data, so they are safer to use in a URL. For example, if a customer with id 10 accesses his accountvia the http://www.example.com/customers/10/ URL, it is easy to guess that there is a customer 11, 12, etc., and this could be a target for an attack.
* UUID values can be generated anywhere that avoids a round trip to the database server. It also simplifies the logic in the application. For example, to insert data into a parent table and child tables, you have to insert it into the parent table first, get the generated id, and then insert the data into the child tables. By using UUID, you can generate the primary key value of the parent table up front, and insert rows into both the parent and child tables at the same time within a transaction.

### **Cons**

Besides the advantages, UUID values also comes with some disadvantages:

* Storing UUID values (16-bytes) takes more storage than integers (4-bytes) or even big integers(8-bytes).
* Debugging seems to be more difficult. Imagine the expression WHERE id = 'df3b7cb7-6a95-11e7-8846-b05adad3f0ae' instead of WHERE id = 10
* Using UUID values may cause performance issues due to their size and not being ordered.

In MySQL, you can store **UUID** values in a compact format (BINARY) and display them in a human-readable format (VARCHAR) with the help of the following functions:

* UUID\_TO\_BIN()
* BIN\_TO\_UUID()
* IS\_UUID()

Notice that UUID\_TO\_BIN(), BIN\_TO\_UUID(), and IS\_UUID() functions are only available in MySQL 8.0 or later.

The UUID\_TO\_BIN() function converts a UUID from a human-readable format (VARCHAR) into a compact format (BINARY) format for storing, and the BIN\_TO\_UUID() function converts a UUID from the compact format (BINARY)to human-readable format (VARCHAR) for displaying.

The IS\_UUID() function returns 1 if the argument is a valid string-format UUID. If the argument is not valid string format UUID, the IS\_UUID function returns 0. In case the argument is NULL, the IS\_UUID() function returns NULL.

# Introduction to index

An index is a data structure such as a B-Tree that improves the speed of data retrieval on a table at the cost of additional writing and storage to maintain it.

The query optimizer may use indexes to quickly locate data without having to scan every row in a table for a given query.

When you create a table with a primary key or unique key, MySQL automatically creates a special index named PRIMARY. This index is called the clustered index.

The PRIMARY index is special because the index itself is stored together with the data in the same table. The clustered index enforces the order of rows in the table.

Other indexes other than the PRIMARY index are called secondary indexes or non-clustered indexes.