SQL Overview. SQL DDL. Data Types.

Summary

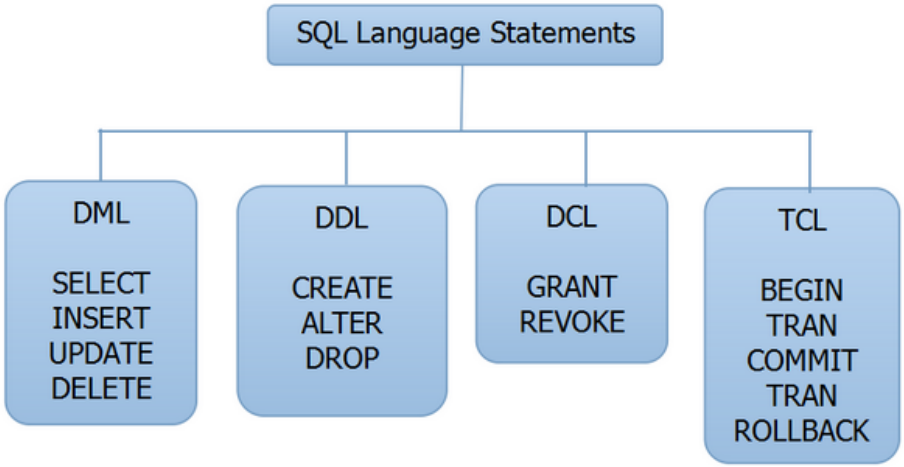
- SQL overview.
  - SQL Syntax.
  - SQL DDL
  - SQL DML
  - SQL DCL
  - SQL TCL
- MySQL DDL Realization
  - MySQL Create/Show/Drop Databases
  - MySQL Create/Alter/Drop Tables
  - ER Diagram Forward Engineering
- MySQL Data Types
  - Numeric Data Types
  - Date/Time Data Types
  - Text Data Types
  - Null Data.
  - Data Types Definition Example
- SQL Cheat Sheet.

SQL Overview

SQL stands for Structured Query Language. SQL is a programming language for Relational Databases. It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS.

Starting MySQL: mysql client starts using a "Command line":

```
mysql -u username -p password
mysql> use database_name;
```



SQL include:

- DML – Data Manipulation Language;
- DDL – Data Definition Language;
- DCL – Data Control Language;
- TCL – Transactional Control Language.

SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language. However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

Note: Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

SQL syntax features and Best practices short:

- Upper and lower case letters are not distinguished in SQL commands (except for the contents of character strings).
- Use upper case letters for SQL keywords i.e. "DROP SCHEMA IF EXISTS 'MyDatabase';"
- Good, portable rules: First character should be alphabetical; Remaining characters should be alphanumeric or underscore '_'.
- Use same case in DML that you use in DDL.
- End all your SQL commands using semi colons ';'.
- Avoid using spaces in schema, table and field names. Use underscores instead to separate schema, table or field names.
- The character and character string are enclosed in single quotes: 'A', '2', 'line', 'other line'
- A one-line comment starts with the characters '--'.
- Multi-line comment is in the characters / * ... * /.

SQL Syntax

SQL is a declarative language, therefore, its syntax reads like a natural language. An SQL statement begins with a verb that describes the action, for example, SELECT, INSERT, UPDATE or DELETE. Following the verb are the subject and predicate.

A predicate specifies conditions that can be evaluated as true, false, or unknown.

See the following SQL statement →

As you see, it reads like a normal sentence.

```
Get the first names of employees who were hired in 2000.
```

The SELECT first_name, FROM employees, and WHERE are clauses in the SQL statement. Some clauses are mandatory e.g., the SELECT and FROM clauses whereas others are optional such as the WHERE clause.

SQL commands

SQL is made up of many commands terminated with a semicolon (;). For example, the following are two different SQL commands.

```
SELECT
	first_name, last_name
FROM
	employees;

DELETE FROM employees
WHERE
	hire_date < '1990-01-01';
```

Each command is composed of **tokens** that can be **literals, keywords, identifiers, or expressions**. Tokens are **separated** by space, tabs, or newlines.

Literals

Literals are explicit values which are also known as constants. SQL provides three kinds of literals: string, numeric, and binary.

String literal consists of one or more alphanumeric characters surrounded by single quotes, for example:

```
'John'
'1990-01-01'
'50'
''
```

50 is a number. However, if you surround it with single quotes e.g., '50', SQL treats it as a string literal. '' - clear string. Typically, SQL is *case sensitive* with respect to string literals, so the value 'John' is not the same as 'JOHN'.

Numeric literals are the integer, decimal, or scientific notation, for example:

```
200
-5
6.0221415E23
```

Binary literals. SQL represents binary value using the notation x'0000', where each digit is hexadecimal value, for example:

```
x'01'
x'0f0ff'
```

A simple rule for us to remember what to use in which case (that is part of SQL-92 standard):
[S]ingle quotes are for [S]trings ; [D]ouble quotes are for [D]atabase identifiers;

Keywords

SQL has many keywords that have special meanings such as SELECT, INSERT, UPDATE, DELETE, DROP, WHERE, FROM, SET, VIEW, TABLE, INT, VARCHAR, BETWEEN, NULL, etc.

These keywords are the **reserved words**, therefore, you **cannot use** them as the name of tables, columns, indexes, views, stored procedures, triggers, or other database objects.

Identifiers

Identifiers refer to specific objects in the database such as tables, columns, indexes, etc.

SQL is case-insensitive with respect to keywords and identifiers. The following statements are equivalent.

```
Select * From employees;
SELECT * FROM EMPLOYEES;
select * from employees;
SELECT * FROM employees;
```

To make the SQL commands more readable and clear, we will use the SQL keywords in uppercase and identifiers in lower case.

Comments

To document SQL statements, you use the SQL comments. When parsing SQL statements with comments, the database engine ignores the characters in the comments.

A comment is denoted by two consecutive hyphens (--) that allow you to comment the remaining line. See the following example.

```
SELECT
	employee_id, salary
FROM
	employees
WHERE
	salary < 3000;-- employees with low salary
```

To document the code that can span multiple lines, you use the multiline C-style notation (/**/) as the shown in the following statement:

```
/* increase 5% for employees
whose salary is less than 3,000 */
UPDATE employees
SET
	salary = salary * 1.05
WHERE
	salary < 3000;
```

SQL DDL - Data Definition Language

SQL uses the following set of commands to define database schema:

- Create,
- Drop,
- Alter.

CREATE

Creates new databases, tables, index and views from RDBMS.

For example

```
Create database tutorialspoint;
Create table article;
Create view for_students;
```

DROP

Drops commands, views, tables, index and databases from RDBMS.

For example−

```
Drop object_type object_name;
Drop database tutorialspoint;
Drop table article;
Drop view for_students;
```

ALTER

Modifies database schema.

Syntax.

```
Alter object_type object_name parameters;
```

For example.

```
Alter table article add subject varchar;
```

This command adds an attribute in the relation **article** with the name **subject** of string type.

SQL DML - Data Manipulation Language

SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section:

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

SELECT/FROM/WHERE

To query data from a table, you use the SQL `SELECT` statement, where contains the syntax for selecting columns, selecting rows, grouping data, joining tables, and performing simple calculations.

The `SELECT` statement is one of the most complex commands in SQL.

- **SELECT** − This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
- **FROM** − This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
- **WHERE** – This clause defines predicate or conditions for filtering data based on a specified condition.
- **ORDER BY** – for sorting the result set
- **LIMIT** – for limiting rows returned
- **JOIN** – for querying data from multiple related tables
- **GROUP BY** – for grouping data based on one or more columns
- **HAVING** – for filtering groups

You will learn about these clauses in the subsequent tutorials on Practice Works PW-01 and PW-02.

Syntax

The following illustrates the *basic* syntax of the `SELECT` statement that retrieves data from a single table.

```
SELECT
	column1, column2, column3, ...
FROM
	table_name;
```

In this syntax, you specify a list of comma-separated columns from which you want to query the data in the `SELECT` clause and specify the table name in the `FROM` clause. When evaluating the `SELECT` statement, the database system evaluates the `FROM` clause first and then the `SELECT` clause.

The semicolon (;) is not the part of a query. Typically, the database system uses the semicolon to separate two SQL queries.

In case you want to query data from all columns of a table, you can use the asterisk (*) operator instead of the column list as shown below.

```
SELECT
	*
FROM
	table_name;
```

For example

```
SELECT author_name as COVID_19_risk_author
FROM book_author
WHERE age > 60;
```

This command will yield the names of authors from the relation **book_author** whose age is greater than 60 (Author with COVID-19 Risk).

INSERT INTO/VALUES

This command is used for inserting values into one or many rows of a table (relation).

Syntax

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [, value2, value3 ... ])
```

For example one row

```
INSERT INTO tutorialspoint (Author, Subject) VALUES ("anonymous", "computers");
```

For example many rows

```
INSERT INTO Course (course_id, course_duration, course_cost)
VALUES
      ('3', '64', '25'),
      ('2', NULL, NULL);
```

UPDATE/SET/WHERE

This command is used for updating or modifying the values of columns in a table (relation). Without WHERE clause updated all table rows.

Syntax

```
UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition]
```

For example

```
UPDATE tutorialspoint SET Author="webmaster" WHERE Author="anonymous";
```

DELETE/FROM/WHERE

This command is used for removing one or more rows from a table (relation). Without WHERE clause deleted all table rows.

Syntax

```
DELETE FROM table_name [WHERE condition];
```

For example

```
DELETE FROM tutorialspoints
   WHERE Author="unknown";
```

SQL DCL - Data Control Language

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

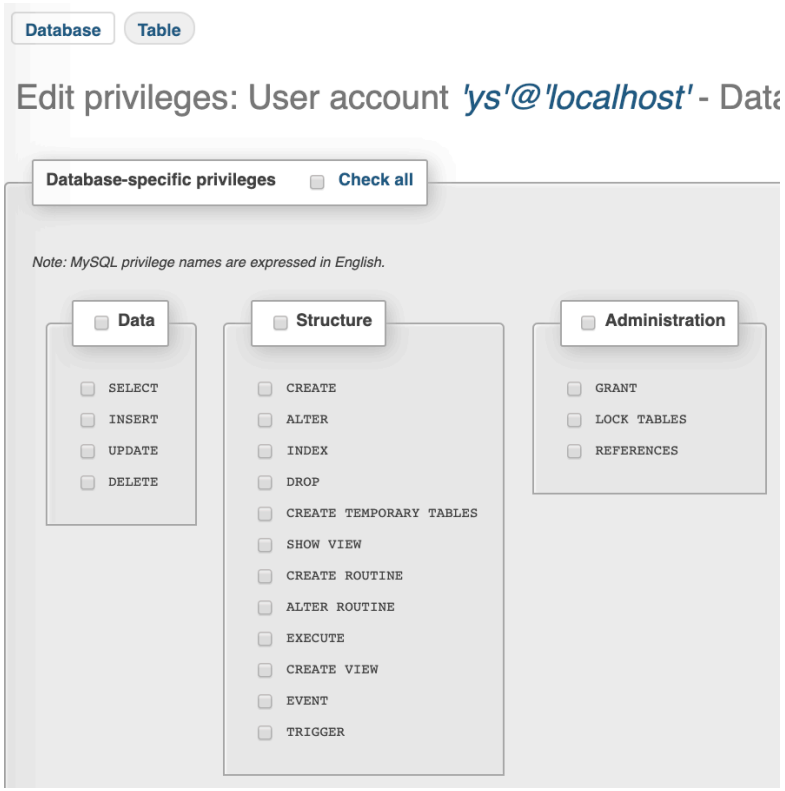
GRANT - command gives user's access privileges to database.

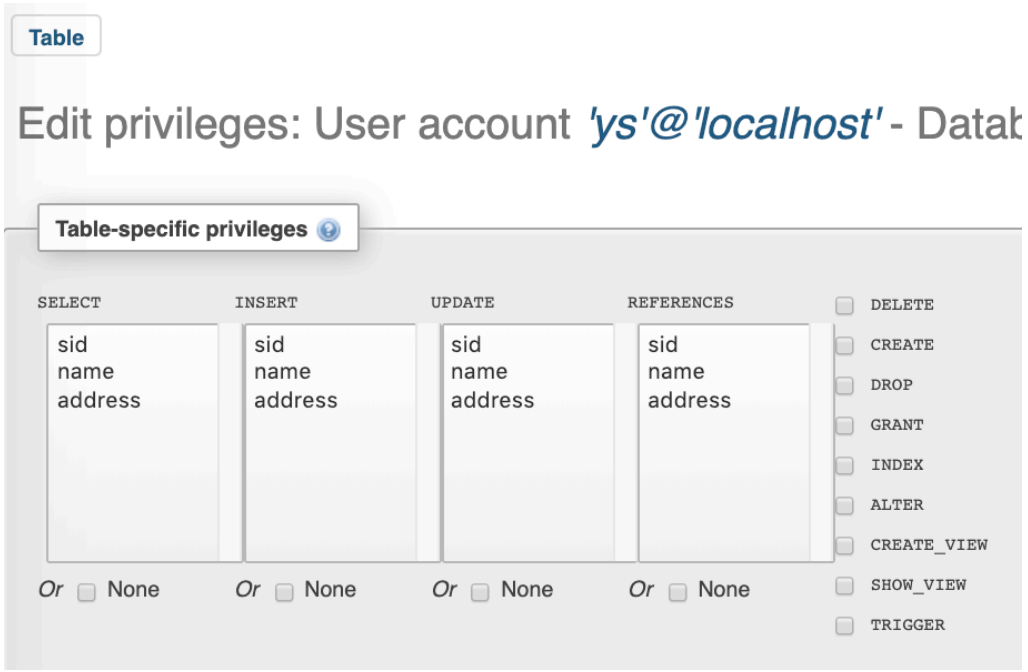
```
grant select, insert, delete, references, update to userName
```

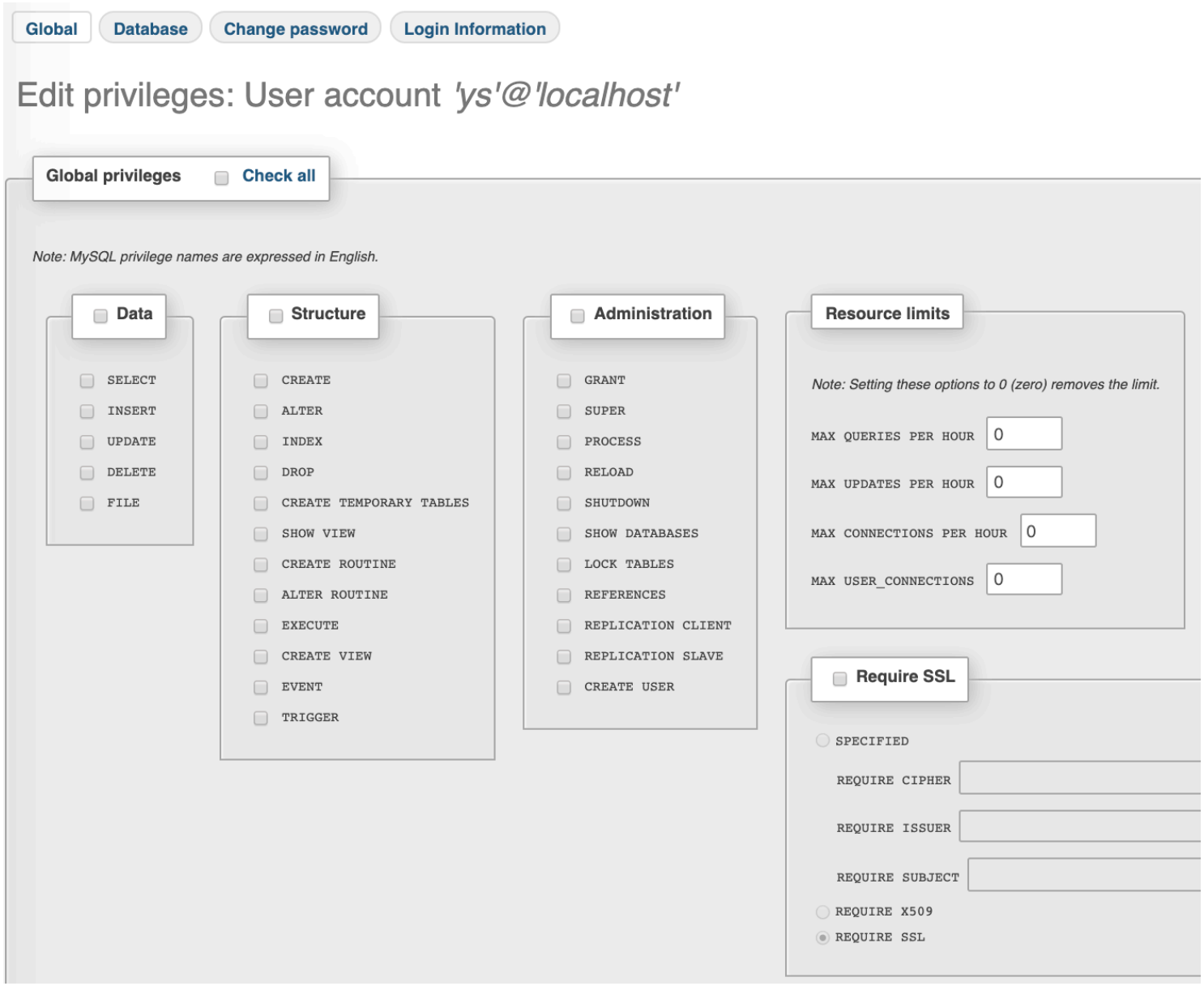
REVOKE -withdraw user's access privileges given by using the GRANT command.

```
revoke insert, delete, references, update to userName
```

PhpMyAdmin usage for grant/revoke privileges to Database, Table and Global:






Global
Database
Change password
Login Information
Edit privileges: User account 'ys'@'localhost'
Global privileges
Check all
Note: MySQL privilege names are expressed in English.
Data
SELECT
INSERT
UPDATE
DELETE
FILE
Structure
CREATE
ALTER
INDEX
DROP
CREATE TEMPORARY TABLES
SHOW VIEW
CREATE ROUTINE
ALTER ROUTINE
EXECUTE
CREATE VIEW
EVENT
TRIGGER
Administration
GRANT
SUPER
PROCESS
RELOAD
SHUTDOWN
SHOW DATABASES
LOCK TABLES
REFERENCES
REPLICATION CLIENT
REPLICATION SLAVE
CREATE USER
Resource limits
Note: Setting these options to 0 (zero) removes the limit.
MAX QUERIES PER HOUR 0
MAX UPDATES PER HOUR 0
MAX CONNECTIONS PER HOUR 0
MAX USER_CONNECTIONS 0
Require SSL
SPECIFIED
REQUIRE CIPHER
REQUIRE ISSUER
REQUIRE SUBJECT
REQUIRE X509
REQUIRE SSL

SQL TCL - Transaction Control Language

TCL commands deals with the transaction within the database.

BEGIN – begin a Transaction.

COMMIT– commits a Transaction.

ROLLBACK– rollbacks a transaction in case of any error occurs.

SAVEPOINT–sets a savepoint within a transaction.

SET TRANSACTION–specify characteristics for the transaction.

MySQL DDL Realization

MySQL Create/Show/Drop Databases

CREATE DATABASE is the SQL command for creating a database. Imagine you need to create a database with name "movies". You can do it by executing following SQL command.

```
CREATE DATABASE movies;
```

Note: you can also use the command CREATE SCHEMA instead of CREATE DATABASE

You can see list of existing databases by running following SQL command.

```
SHOW DATABASES
```

You can delete of existing databases by running following SQL command.

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

Now let's improve our SQL query adding more parameters and specifications.

IF NOT EXISTS clause

A single MySQL server could have multiple databases. If you are not the only one accessing the same MySQL server or if you have to deal with multiple databases there is a probability of attempting to create a new database with name of an existing database . **IF NOT EXISTS** let you to instruct MySQL server to check the existence of a database with a similar name prior to creating database.

```
CREATE DATABASE IF NOT EXISTS movies;
```

Collation and Character Set

Collation is set of **rules used in comparison.** Many people use MySQL to store data other than English. Data is stored in MySQL using a specific character set. The character set can be defined at different levels viz, server , database , table and columns.

You need to select the rules of collation which in turn depend on the character set chosen. For instance, the **latin1** character set uses the **latin1_swedish_ci** collation which is the Swedish case insensitive order.

The best practice while using local languages like Arabic, Chinese, Russian etc is to select Unicode (utf-8) character set which has several collations or just stick to default collation utf8-general-ci.

```
CREATE DATABASE IF NOT EXISTS movies CHARACTER SET utf8 COLLATE utf8_general_ci
```

You can find the list of all collations and character sets here (http://dev.mysql.com/doc/refman/5.5/en/charset-charsets.html).

MySQL Create Tables

Syntax. Tables can be created using CREATE TABLE statement and it actually has the following syntax.

```
CREATE TABLE [IF NOT EXISTS] [`DatabaseName`.]`TableName` (
  `fieldname1` dataType1 [optional parameters1]
  [, `fieldname2` dataType2 [optional parameters2]…
  [, table optional parameters]
  [, table optional parameters2]…)
[ENGINE = storage Engine];
```

- "CREATE TABLE" is the one responsible for the creation of the table in the database.
- "[IF NOT EXISTS]" is optional and only create the table if no matching table name is found.
- "`fieldName`" is the name of the field and "data Type" defines the nature of the data to be stored in the field.
- "[optional parameters]" - additional information about a field such as "AUTO_INCREMENT" , NOT NULL etc.

Example

```
CREATE  TABLE IF NOT EXISTS `MyFlixDB`.`Members` (
  `membership_number` INT  AUTO_INCREMENT ,
  `full_names` VARCHAR(150) NOT NULL ,
  `gender` VARCHAR(6) ,
  `date_of_birth` DATE ,
  `postal_address` VARCHAR(255) ,
  PRIMARY KEY (`membership_number`) )
ENGINE = InnoDB;
```

Primary Key Constraints

- The CREATE TABLE syntax also allows “[table optional parameters”] - additional information about a table such as PRIMARY KEY.
- Table optional parameters generally specified after attributes are listed.

Example with attribute constraint

```
CREATE TABLE account (
  acct_id CHAR(10) PRIMARY KEY,
  person_name CHAR(20),
  email VARCHAR(255),
  balance NUMERIC(12, 2)
  );
```

Example with table constraint

```
CREATE TABLE account (
  acct_id CHAR(10),
  person_name CHAR(20),
  email VARCHAR(255),
  balance NUMERIC(12, 2),
  PRIMARY KEY (acct_id)
);
```

- Database won’t allow two rows with same account ID
- A primary key can have multiple attributes

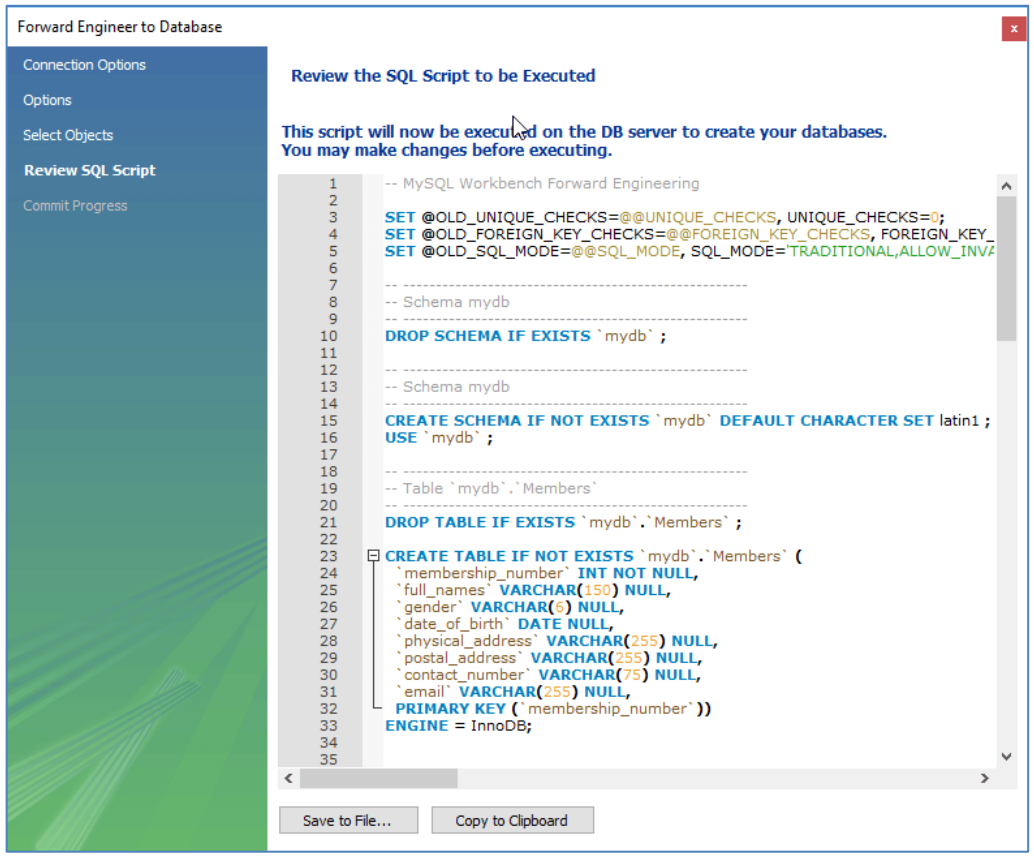
Example

```
CREATE TABLE depositor (
  customer_name VARCHAR(30),
  acct_id CHAR(10),
  PRIMARY KEY (customer_name, acct_id)
);
```

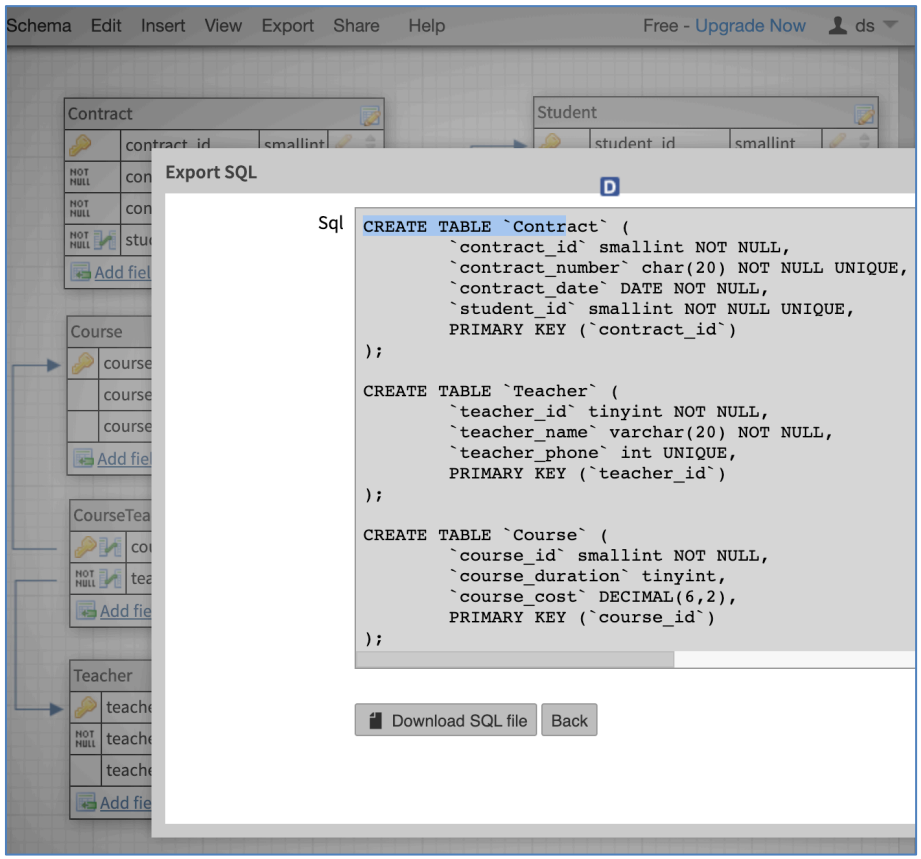
- A table can’t have multiple primary keys – (obvious)
- Many other kinds of constraints too – Will cover in future lectures!

ER Diagram Forward Engineering

Many Offline and Online CASE Tools (for example, MySQL Workbench, phpMyAdmin, DB Designer, Adminer) has utilities that support forward engineering process of **translating a logical model into a physical implement automatically** (SQL scripts to create the physical database).



MySQL Workbench Create SQL Example



DB Designer Create SQL Example

MySQL Data Types

Data types define the nature of the data that can be stored in a particular column of a table

MySQL has **many** categories of data types:

1. Numeric,
2. Text
3. Date/Time.
4. NULL
5. and other

Read more on

- https://dev.mysql.com/doc/refman/8.0/en/data-types.html,
- https://www.w3schools.com/sql/sql_datatypes.asp,
- https://www.sqltutorial.org/sql-data-types/

Numeric Data Types

Data Type	Storage	Range	Description
BIT or BIT[(M)]	1 Byte	0 or 1	A bit-value type. M indicates the number of bits per value, from 1 to 64. The default is 1 if M is omitted.
TINYINT or TINYINT[(M)] [UNSIGNED]	1 Byte	-128 to 127 normal 0 to 255 unsigned	Integer. Can be declared positive using the UNSIGNED keyword, then the column elements cannot be assigned a negative value. Optional parameter M - the number of characters allocated for the number of characters. Examples: TINYINT - Stores any number in the range from -128 to 127. TINYINT UNSIGNED - Stores any number in the range from 0 to 255. TINYINT (2) - it is assumed that the values will be two-digit, but in fact will store three-digit ones.

BOOL or BOOLEAN	1 Byte	0 or 1	These types are synonyms for TINYINT(1). A value of zero is considered false. Nonzero values are considered true.
SMALLINT or SMALLINT[(M)] [UNSIGNED]	2 Byte	-32768 to 32767 normal 0 to 65535 unsigned	Similar to TINYINT, but with a large range.
MEDIUMINT or MEDIUMINT[(M)] [UNSIGNED]	3 Byte	-8388608 to 8388607 normal 0 to 16777215 unsigned	Similar to TINYINT, but with a large range.
INT or INT[(M)] [UNSIGNED]	4 Byte	-2147483648 to 2147483647 normal 0 to 4294967295 unsigned	Similar to TINYINT, but with a large range.
BIGINT or BIGINT[(M)] [UNSIGNED]	8 Byte	-2^63 to 2^63-1 normal 0 to 2^64-1 unsigned	Similar to TINYINT, but with a large range.
FLOAT (M,D)	4 Byte	min value +(-) 1.175494351 * 10^-39 max value +(-) 3. 402823466 * 10^38	Real number (floating point). May have a parameter UNSIGNED, prohibiting negative numbers, but the range of values from this will not change. M - the number allocated to the number of characters. D is the number of characters of the fractional part. Example: FLOAT (5,2) - will store numbers of 5 characters, 2 of which will come after the decimal point (for example: 46.58).
DOUBLE (M,D)	8 Byte	min value +(-) 2.2250738585072015*10^-308 max value +(-) 1.797693134862315 * 10^308	Similar to FLOAT, but with a large range.
DECIMAL(M,D) or DEC(M,D) or NUMERIC(M,D)	M+2 Bytes	depend on parameters M and D	A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values. They are used for increased accuracy values, for example, for monetary data. M is the number of characters allocated for the number of characters (the maximum value is 64). D is the number of decimal places (maximum value is 30). Example: DECIMAL (5,2) - will store numbers from -999.99 to 999.99.

Text Data Types

Data Type	Storage	Range	Description
CHAR(M) or BINARY(M)	M characters	0 to 255	A fixed long string. The length can be specified as a value from 0 to 65535. BINARY similar to CHAR, difference is texts are stored in binary format. Example: CHAR (6) – stores strings of 6 characters and takes 6 bytes. For example, any of the following values: an empty string ‘ ‘, ‘Kim’, ‘Ivan’, ‘Sergey’ will occupy 6 bytes of memory. And when you try to enter the value ‘Alexander’, it will be truncated to ‘Alexan’.
VARCHAR(M) or VARBINARY(M)	M+1 characters	0 to 65535	A variable long string. The length can be specified as a value from 0 to 65535. The **effective maximum length** of a VARCHAR is subject to the maximum row size (65535 bytes, which is shared among all columns) and the character set used (1B, 2B, etc). VARBINARY similar to VARCHAR, difference is texts are stored in binary format. Example: VARCHAR (3) - stores strings with a maximum of 3 characters, but an empty string ' ' occupies 1 byte of memory, a string 'a' - 2 bytes, a string 'aaa' - 4 bytes (if 1 byte character set use). Values greater than 3 characters will be truncated to 3.
TINYTEXT	M+1 characters	0 to 255	A string with a maximum length of 255 characters.
TEXT or BLOB	M+2 characters	0 to 65535	Allow you to store large amounts of text. The TEXT type is used to store text, and BLOB – to store images, sound, files, etc.
MEDIUMTEXT or MEDIUMBLOB	M+3 characters	0 to 2^24-1	Similar to TEXT or BLOB, but with a large range.
LONGTEXT or LONGBLOB	M+4 characters	0 to 2^32-1	Similar to TEXT or BLOB, but with a large range.
ENUM('value1',...,'valueN')	1 or 2 bytes	0 to 65535 elements	Strings of this type can take **only one** of the values of the specified set. Example: ENUM ('yes', 'no', ‘I don’t know’) - only one of the available values can be stored in a column with this type. It is convenient to use if it is provided that the answer to the question should be stored in the column.
SET('value1',...,'valueN')	1,2,3,4,8 bytes	1-8, 9-16, 17-24, 25-32, 33-64 elements	This is also used for storing text values chosen from a list of predefined text values. It can have **multiple values**. Example: SET ('first', 'second', ‘third’) - Strings may accept any or several or all elements from the values of the specified set, or the value may be absent altogether.

Date/Time Data Types

Data Type	Storage	Range	Description
DATE	3 bytes	'1000-01-01' to '9999-12-31'	Designed for storing dates. The first value is the year in the format "YYYY", after a minus the month in the format "MM", and then the day in the format "DD". The separator can be not only a minus, but any character other than a digit. Example: CHAR (6) – stores strings of 6 characters and takes 6 bytes. For example, any of the following values: an empty string ' ', 'Kim', 'Ivan', 'Sergey' will occupy 6 bytes of memory. And when you try to enter the value 'Alexander', it will be truncated to 'Alexan'.
TIME	3 bytes	'-838:59:59' to '838:59:59'	Designed to store the time of day. The value is entered and stored in the usual format: hh: mm: ss, where hh is hours, mm is minutes, ss is seconds. Any character other than a digit can be used as a separator.
DATETIME	8 bytes	'1000-01-01 00:00:00' to '9999-12-31 23:59:59'	Designed for storage of both date and time of day. The value is entered and stored in the format: YYYY-MM-DD hh: mm: ss. Separators can be any characters other than numbers.
TIMESTAMP	4 bytes	'1970-01-01 00:00:00' to '2037-12-31 23:59:59'	Designed to store the date and time of day as the number of seconds that have passed since midnight on January 1, 1970 (the beginning of the UNIX era). The value is entered in the format: YYYYMMDDHHMMSS.
YEAR(M)	1 byte	1970 to 2069 for M=2 1901 to 2155 for M=4	Designed for storage only a year. M - sets the format of the year. For example, YEAR (2) is 70, and YEAR (4) is 1970. If parameter M is not specified, then by default it is considered to be 4.

Null Data

In fact, this is a pointer to the possibility of a lack of value, i.e. required and optional fields. In order to store such information in the database, two values are used:

- NOT NULL (value cannot be absent) for fields login and password,
- NULL (value may be absent) for the fields date of birth and gender.
- By default, all columns are set to NOT NULL, so you can omit it explicitly.

Example:

```
create table users (login varchar(20), passw varchar(15), gender enum('man', 'woman') NULL, dob NULL);
```

Data Types Definition Example.

Now let's see a sample SQL query for creating a table which has data of many data types.

Task. Study it and identify how each data type is defined.

```
CREATE TABLE`all_data_types` (
    `varchar` VARCHAR( 20 )  ,
    `tinyint` TINYINT  ,
    `text` TEXT  ,
    `date` DATE  ,
    `smallint` SMALLINT  ,
    `mediumint` MEDIUMINT  ,
    `int` INT  ,
    `bigint` BIGINT  ,
    `float` FLOAT( 10, 2 )  ,
    `double` DOUBLE  ,
    `decimal` DECIMAL( 10, 2 )  ,
    `datetime` DATETIME  ,
    `timestamp` TIMESTAMP  ,
    `time` TIME  ,
    `year` YEAR  ,
    `char` CHAR( 10 )  ,
    `tinyblob` TINYBLOB  ,
    `tinytext` TINYTEXT  ,
    `blob` BLOB  ,
    `mediumblob` MEDIUMBLOB  ,
    `mediumtext` MEDIUMTEXT  ,
    `longblob` LONGBLOB  ,
    `longtext` LONGTEXT  ,
    `enum` ENUM( '1', '2', '3' )  ,
    `set` SET( '1', '2', '3' )  ,
    `bool` BOOL  ,
    `binary` BINARY( 20 )  ,
    `varbinary` VARBINARY( 20 )
) ENGINE= MYISAM ;
```

SQL Cheat Sheet

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE  c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

MANAGING TABLES

```
CREATE TABLE t (
    id INT PRIMARY KEY,
    name VARCHAR NOT NULL,
    price INT DEFAULT 0
);
```
Create a new table with three columns

```
DROP TABLE t ;
```
Delete the table from the database

```
ALTER TABLE t ADD column;
```
Add a new column to the table

```
ALTER TABLE t DROP COLUMN c ;
```
Drop column c from the table

```
ALTER TABLE t ADD constraint;
```
Add a constraint

```
ALTER TABLE t DROP constraint;
```
Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```
Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2 ;
```
Rename column c1 to c2

```
TRUNCATE TABLE t;
```
Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
   c1 INT, c2 INT, c3 VARCHAR,
   PRIMARY KEY (c1,c2)
);
```
Set c1 and c2 as a primary key

```
CREATE TABLE t1(
   c1 INT PRIMARY KEY,
   c2 INT,
   FOREIGN KEY (c2) REFERENCES t2(c2)
);
```
Set c2 column as a foreign key

```
CREATE TABLE t(
   c1 INT, c1 INT,
   UNIQUE(c2,c3)
);
```
Make the values in c1 and c2 unique

```
CREATE TABLE t(
  c1 INT, c2 INT,
  CHECK(c1> 0 AND c1 >= c2)
);
```
Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 VARCHAR NOT NULL
);
```
Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
           (value_list), ….;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
      c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name;
```
Delete a view

MANAGING INDEXES

```
CREATE INDEX idx_name
ON t(c1,c2);
```
Create an index on c1 and c2 of the table t

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
```
Create a unique index on c3, c4 of the table t

```
DROP INDEX idx_name;
```
Drop an index

SQL AGGREGATE FUNCTIONS

AVG returns the average of a list

COUNT returns the number of elements of a list

SUM returns the total of a list

MAX returns the maximum value in a list

MIN returns the minimum value in a list

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```
Create or modify a trigger

WHEN
- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

EVENT
- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

TRIGGER_TYPE
- **FOR EACH ROW**
- **FOR EACH STATEMENT**

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```
Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```
Delete a specific trigger