FUTURE CONNECT TRAINING
Part of Future Connect Accountants

MySQL Basic To Advance

Course Introduction

MySQL™

Table of Contents

MY SQL Introduction

MySQL is a very popular open-source relational database management system (RDBMS).

What is MySQL?

- MySQL is a relational database management system
- MySQL is open-source
- MySQL is free
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, scalable, and easy to use
- MySQL is cross-platform
- MySQL is compliant with the ANSI SQL standard
- MySQL was first released in 1995
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

Who Uses MySQL?

- Huge websites like Facebook, Twitter, Airbnb, Booking.com, Uber, GitHub, YouTube, etc.
- Content Management Systems like WordPress, Drupal, Joomla!, Contao, etc.
- A very large number of web developers around the world

Show Data On Your Web Site:

- To build a web site that shows data from a database, you will need:
- An RDBMS database program (like MySQL)
- A server-side scripting language, like PHP
- To use SQL to get the data you want.
- To use HTML / CSS to style the page.

Why MYSQL

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.

History

- 1970 -- Dr. E. F. "Ted" 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 relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

MYSQL Commands

Some of The Most Important SQL Commands

- SELECT - extracts data from a database.
- UPDATE - updates data in a database.
- DELETE - deletes data from a database.
- INSERT INTO - inserts new data into a database.
- CREATE DATABASE - creates a new database.
- ALTER DATABASE - modifies a database.
- CREATE TABLE - creates a new table.
- ALTER TABLE - modifies a table.
- DROP TABLE - deletes a table.
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index.

CLC- Command Line Client

What is a MySQL client? MySQL client is a common name for tools that are designed to connect to MySQL Server. Client programs are used to send commands or queries to the server and allow managing data in the databases stored on the server.

MySQL sends each SQL statement that you issue to the server to be executed. There is also a set of commands that MySQL itself interprets. For a list of these commands, type help or \h at the MySQL> prompt:

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given
               outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing
               binlog with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
resetconnection(\x) Clean session context.
query_attributes Sets string parameters (name1 value1 name2 value2 ...)
for the next query to pick up.
ssl_session_data_print Serializes the current SSL session data to stdout
or file.

For server side help, type 'help contents'
```

DDL- Data Definition Language

DDL is an abbreviation for Data Definition Language. It is concerned with database schemas and descriptions of how data should be stored in the database. DDL statements are auto committed, meaning the changes are immediately made to the database and cannot be rolled back. These commands enable database administrators and developers to manage and optimize MySQL databases effectively.

DDL Command	Description
`CREATE DATABASE`	Creates a new database.
`DROP DATABASE`	Deletes a database.
`CREATE TABLE`	Creates a new table in a database.
`ALTER TABLE`	Alters the structure of an existing table.
`DROP TABLE`	Removes a table from a database.
`CREATE INDEX`	Creates an index on a table to improve a specific query performance.
`CREATE VIEW`	Creates a view, a virtual table based on one or more existing tables.
`CREATE PROCEDURE`	Creates a stored procedure, a precompiled SQL statement that can be run multiple times with different parameters.
`CREATE FUNCTION`	Creates a custom user-defined function that can be utilized in SQL statements.
`CREATE TRIGGER`	Creates a trigger, a type of stored procedure that is automatically executed when certain events occur, such as inserting, updating, or deleting data in a table.

DML- Data Manipulation Language

DML stands for Data Manipulation Language. It deals with data manipulation and includes the most common SQL statements such as SELECT, INSERT, UPDATE, DELETE, etc. DML statements are not auto committed, meaning the changes can be rolled back if necessary. By mastering these DML commands, you can efficiently manipulate data in MySQL databases.

DML Command	Description
SELECT	Retrieves data from a table.
INSERT	Inserts new data into a table.
UPDATE	Updates existing data in a table.
DELETE	Deletes data from a table.
REPLACE	Updates or inserts a record into a table.
MERGE	Performs a UPSERT operation (insert or update) on a table.
CALL	Calls a stored procedure or Java subprogram.
EXPLAIN	Displays the execution plan for a given query.
LOCK TABLE	Locks a table to prevent other users from modifying it while a transaction progresses.

DCL

DCL stands for Data Control Language. It includes commands such as GRANT and is primarily concerned with rights, permissions, and other controls of the database system. DCL statements are also auto committed.

DCL Command	Description
GRANT	Grants permissions to a user or group of users.
REVOKE	Revokes permissions from a user or group of users.

TCL

TCL stands for Transaction Control Language. It deals with transactions within a database, which are groups of related DML statements treated as a single unit.

TCL Command	Description
COMMIT	Commits a transaction, making the changes permanent.
ROLLBACK	Rolls back a transaction, undoing all the changes made.
SAVEPOINT	Creates a savepoint within a transaction so that the transaction can be rolled back to that point if necessary.
SET TRANSACTION	Specifies the characteristics of a transaction, such as its isolation level.

Select Statement

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

SELECT Syntax:

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

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

Where Clause

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.

WHERE Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

AND, OR, NOT

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

ORDER BY

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

INSERT INTO

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax:

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

NULL VALUES

What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

UPDATE

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

DELETE

The DELETE statement is used to delete existing records in a table.

DELETE Syntax:

```
DELETE FROM table_name WHERE condition;
```

LIMIT

The LIMIT clause is used to specify the number of records to return.

The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

MIN, MAX

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax:

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax:

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

COUNT, AVG, SUM

- The COUNT() function returns the number of rows that matches a specified criterion.

COUNT() Syntax:

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

- The AVG() function returns the average value of a numeric column.

AVG() Syntax:

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

- The SUM() function returns the total sum of a numeric column.

SUM() Syntax:

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

LIKE

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters.
- The underscore sign (_) represents one, single character.

The percent sign and the underscore can also be used in combinations!

LIKE Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

IN

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

OR:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

BETWEEN

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

ALIASES

- Aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.

Alias Column Syntax:

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

JOINS

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

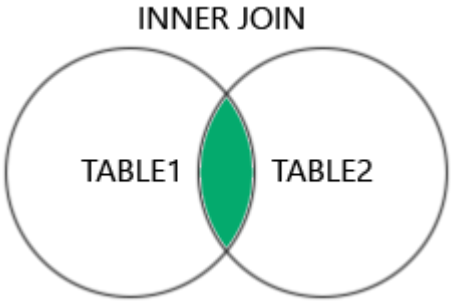
```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

INNER JOIN

The INNER JOIN keyword selects records that have matching values in both tables.

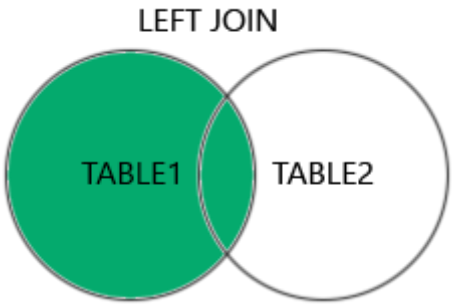


INNER JOIN Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).

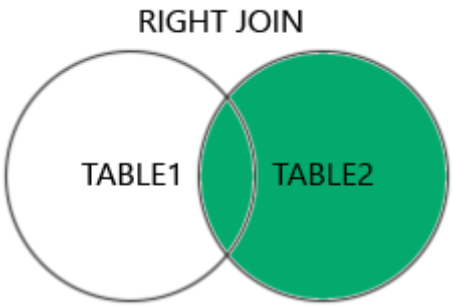


LEFT JOIN Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).

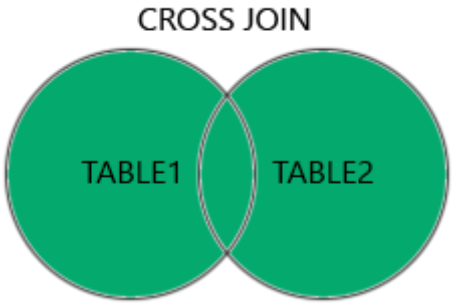


RIGHT JOIN Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

CROSS JOIN

The CROSS JOIN keyword returns all records from both tables (table1 and table2).



CROSS JOIN Syntax:

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

Note: CROSS JOIN can potentially return very large result-sets!

SELF JOIN

A self join is a regular join, but the table is joined with itself.

Self-Join Syntax:

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and *T2* are different table aliases for the same table.

UNION

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax:

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Note: The column names in the result-set are usually equal to the column names in the first SELECT statement.

GROUP BY

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

HAVING

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

EXIST

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

ANY, ALL

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT:

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

ALL Syntax With WHERE or HAVING:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

COMMENTS

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single Line Comments:

Single line comments start with - -

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

```
-- Select all:
SELECT * FROM Customers;
```

Multi-line Comments:

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored.

The following example uses a multi-line comment as an explanation:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

OPERATORS

MySQL Arithmetic Operators:

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

MySQL Bitwise Operators:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

MySQL Comparison Operators:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

MySQL Compound Operators:

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

MySQL Logical Operators:

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

MySQL Database

MySQL Create DB

The CREATE DATABASE statement is used to create a new SQL database.

Syntax:

```
CREATE DATABASE databasename;
```

MySQL Drop DB

The DROP DATABASE statement is used to drop an existing SQL database.

Syntax:

```
DROP DATABASE databasename;
```

MySQL Create Table

The CREATE TABLE statement is used to create a new table in a database.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

MySQL Drop Table

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

```
DROP TABLE table_name;
```

MySQL Alter Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column:

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

ALTER TABLE - DROP COLUMN:

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

ALTER TABLE - MODIFY COLUMN:

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

MySQL Constraints

MYSQL constraints are used to specify rules for data in a table.

Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

MySQL Constraints

MYSQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in MYSQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

MySQL Not Null

MySQL NOT NULL Constraint:

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

NOT NULL on CREATE TABLE:

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

NOT NULL on ALTER TABLE;

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

MySQL Unique

MySQL UNIQUE Constraint:

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

UNIQUE Constraint on CREATE TABLE:

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

UNIQUE Constraint on ALTER TABLE:

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

MySQL Primary Key

MySQL PRIMARY KEY Constraint:

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

PRIMARY KEY on CREATE TABLE:

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Note: In the example above there is only ONE PRIMARY KEY (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

PRIMARY KEY on ALTER TABLE:

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

Note: If you use ALTER TABLE to add a primary key, the primary key column(s) must have been declared to not contain NULL values (when the table was first created).

DROP a PRIMARY KEY Constraint:

To drop a PRIMARY KEY constraint, use the following SQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

MySQL Foreign Key

MySQL FOREIGN KEY Constraint:

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Persons Table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

FOREIGN KEY on CREATE TABLE:

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

FOREIGN KEY on ALTER TABLE:

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

DROP a FOREIGN KEY Constraint:

To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

MySQL Check

MySQL CHECK Constraint:

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

CHECK on CREATE TABLE:

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

CHECK on ALTER TABLE:

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

DROP a CHECK Constraint:

To drop a CHECK constraint, use the following SQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

MySQL Default

MySQL DEFAULT Constraint:

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

DEFAULT on CREATE TABLE:

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

The DEFAULT constraint can also be used to insert system values, by using functions like CURRENT_DATE():

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT CURRENT_DATE()
);
```

DEFAULT on ALTER TABLE:

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

DROP a DEFAULT Constraint:

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

MySQL Create Index

MySQL CREATE INDEX Statement:

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax:

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX Syntax:

Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

DROP INDEX Statement:

The DROP INDEX statement is used to delete an index in a table.

```
ALTER TABLE table_name
DROP INDEX index_name;
```

MySQL Auto Increment

What is an AUTO INCREMENT Field?

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

MySQL AUTO_INCREMENT Keyword:

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

MySQL Dates

MySQL Dates:

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

MySQL Date Data Types:

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Note: The date data type are set for a column when you create a new table in your database!

Working with Dates:

Look at the following table:

Orders Table:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Now we want to select the records with an OrderDate of "2008-11-11" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

The result-set will look like this:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

MySQL Views

MySQL CREATE VIEW Statement:

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

MySQL Data Types

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

MySQL Data Types (Version 8.0):

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

In MySQL there are three main data types: **string, numeric**, and **date** and **time.**

String Data Types:

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric Data Types:

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Note: All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

Date and Time Data Types:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

MySQL Functions

MySQL has many built-in functions.

This reference contains **string, numeric, date**, and some **advanced functions** in MySQL.

MySQL String Functions:

Function	Description
ASCII	Returns the ASCII value of a character
CHAR_LENGTH	Returns the length of a string.
CHARACTER_LENGTH	Returns the length of a string
CONCAT	Concatenates two or more expressions.
CONCAT_WS	Concatenates with a separator.
FIELD	Returns the index of value in a list.
FIND_IN_SET	Returns the index of a string within a list.
FORMAT	Changes the format/representation.
INSERT	Inserts a string within a string at a given index.
INSTR	Returns the index of the first occurrence of a string in another one.
LCASE	Converts an entire string to lowercase.
LEFT	Extracts a length of characters from the left of a string.
LENGTH	Returns the string length in bytes.

LOCATE	Returns the location of the first occurrence of a substring in a given string
LOWER	Converts an entire string to lowercase.
LPAD	Left-pads a string with a given string.
LTRIM	Removes spaces from the left of a string.
MID	Extracts a substring from a string at a given position.
POSITION	Returns the location of the first occurrence of a substring in a given string
REPEAT	Repeats the string the number of times the user specifies.
REPLACE	Replaces occurrences of a substring in a string with another substring.
REVERSE	Reverses the string.
RIGHT	Extracts a length of characters from the right of a string.
RPAD	Right-pads a string with a given string.
RTRIM	Removes spaces from the right of a string.
STRCMP	Checks whether two strings are equal.
SUBSTR	Extracts a substring from a string at a position mentioned by the user.
SUBSTRING	Same as substr.
TRIM	Trims leading and trailing spaces from a string as specified by the user.
UCASE	Converts an entire string to uppercase.
UPPER	Converts an entire string to uppercase.

NUMERIC FUNCTIONS:

Function	Description
ABS	Returns the absolute value.
ACOS	Returns the cosine inverse.
ASIN	Returns the sine inverse.
ATAN	Returns the tan inverse of one or two numbers.
ATAN2	Returns the tan inverse of two numbers.
AVG	Returns the mean value.
CEIL	Returns the smallest integer that is greater than or equal to the number
CEILING	Returns the smallest integer that is greater than or equal to the number
COS	Returns the cosine.
COT	Returns the cotangent.
COUNT	Returns the number of records returned by a query.
DEGREES	Converts angle in Radians to Degrees.

DIV	Integer division
EXP	Returns e raised to the power of value mentioned.
FLOOR	Returns the largest integer that is less than or equal to a number
GREATEST	Returns the largest value in the list.
LEAST	Returns the smallest value in the list.
LN	Calculates logarithm to the base e.
LOG	Calculates logarithm to the base e.
LOG10	Calculates logarithm to the base 10.
LOG2	Calculates logarithm to the base 2.
MAX	Returns the largest value in a set.
MIN	Returns the least value in a set.
MOD	Returns the remainder after division of two numbers.
PI	Returns value of π
POW	Used for exponents.
POWER	Used for exponents.
RADIANS	Converts angle in Degree to Radians.
RAND	Generates a random number.
ROUND	Rounds the number to the nearst decimal place.
SIGN	Returns the sign of a number
SIN	Returns the sine.
SQRT	Returns the root of a number.
SUM	Calculates the sum of a set.
TAN	Returns the tangent.

MYSQL DATE FUNCTION:

Function	Description
ADDDATE	Adds a date interval and return the value.
ADDTIME	Adds a time interval and then returns the value.
CURDATE	Returns today's date
CURRENT_DATE	Same as CURDATE
CURRENT_TIME	Returns the time at the moment
CURRENT_TIMESTAMP	Returns date and time at the moment.
CURTIME	Returns time at the moment.
DATE	Picks up the date from an expression of Date/Time.
DATEDIFF	Returns number of days between two given dates.
DATE_ADD	Similar to ADDDATE
DATE_FORMAT	Changes the format in which Date is displayed.
DATE_SUB	Subtracts a time interval and returns the value.
DAY	Returns the weekday for today.
DAYNAME	Returns the weekday name for any date.
DAYOFMONTH	Used to retrieve the index of the day of the month of any date.
DAYOFWEEK	Used to retrieve the index of the weekday of any date.
DAYOFYEAR	Used to retrieve the index of the day of a year of any date.
EXTRACT	Extracts a part of any date.
HOUR	Returns the "hours" in a given time.
LAST_DAY	Return the last day of the given month.
LOCALTIME	Returns the date and time at the moment.
LOCALTIMESTAMP	Similar to LOCALTIME.
MAKEDATE	Returns a date based on the year and the no. of days you specify.
MAKETIME	Returns a time based on the hours , minutes and seconds you specify.
MICROSECOND	Returns the microseconds in a given time.
MINUTE	Returns the minutes in a given time.
MONTH	Returns the month on a given date.
MONTHNAME	Same as MONTH but returns the name of the month.
NOW	Returns date and time at the moment.
PERIOD_ADD	Adds a specific number of months.
PERIOD_DIFF	Return the difference between two time periods.
SECOND	Return the seconds in a given time.

SEC_TO_TIME	Returns time in seconds.
STR_TO_DATE	Formats the date based on a particular string.
SUBDATE	Same as DATE_SUB.
SUBTIME	Subtracts a time interval.
SYSDATE	Returns the date/time reflected by the system.
TIME	Returns the time from a date/time value.
TIME_FORMAT	Time is displayed based on a certain format.
TIME_TO_SEC	Returns time in seconds.
TIMEDIFF	Returns the difference between two date-time values.
TO_DAYS	Returns the number of days between amy date and "0000-00-00"

ADVANCED MYSQL FUNCTION:

Function	Description
BIN	Returns binary value of a given number.
BINARY	Converts a given string to a binary string.
CAST	Converts data from one data type to another.
COALESCE	Returns the first non-null value in a set or list.
CONV	Converts a number from one number-base system to another
CONVERT	Similar to CAST in working
CURRENT_USER	Returns the user name and host name for the MySQL account that is currently used.
DATABASE	Returns the name of the database currently in use.
IF	IF condition statement.
SESSION_USER	Returns the current MySQL user name and host name.
SYSTEM_USER	Similar to SESSION_USER.
USER	Similar to SESSION_USER.
VERSION	Returns the current version of the MySQL server installed.

CONCLUSION:

By going through this MYSQL BOOKLET, you would have got a decent understanding/revision of MySQL. More than memorizing syntax do pay attention to practising them and solving problems.