```
There are three types of comment:
-- This comment continues to the end of line
/* This is an in-line comment */
/*
This is a
multiple-line comment
*/

CREATE DATABASE mydb; --Creating a database in MySQL

USE mydb; --Using the created database mydb

CREATE TABLE mytable --Creating a table in MySQL
(
    id int unsigned NOT NULL auto_increment,
    username varchar(100) NOT NULL,
    email varchar(100) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE Person (
PersonID INT UNSIGNED NOT NULL PRIMARY KEY,
LastName VARCHAR(66) NOT NULL,
FirstName VARCHAR(66),
Address VARCHAR(255),
City VARCHAR(66)
);

CREATE TABLE invoice_line_items (
LineNum SMALLINT UNSIGNED NOT NULL,
InvoiceNum INT UNSIGNED NOT NULL,
-- Other columns go here
PRIMARY KEY (InvoiceNum, LineNum),
FOREIGN KEY (InvoiceNum) REFERENCES -- references to an attribute of a table
);

CREATE TABLE Account (
AccountID INT UNSIGNED NOT NULL,
AccountNo INT UNSIGNED NOT NULL,GoalKicker.com – MySQL® Notes for Professionals 80
PersonID INT UNSIGNED,
PRIMARY KEY (AccountID),
FOREIGN KEY (PersonID) REFERENCES Person (PersonID)
);

--id int unsigned NOT NULL auto_increment => creates the id column, this type of field
will assign a unique numeric
--ID to each record in the table (meaning that no two rows can have the same id in this
case), MySQL will
--automatically assign a new, unique value to the record's id field (starting with 1).

INSERT INTO mytable ( username, email ) VALUES ( "myuser", "myuser@example.com" );
--Inserting a row into a MySQL table

INSERT INTO mytable ( username, email ) VALUES ( 'username', 'username@example.com' );
--The varchar a.k.a strings can be also be inserted using single quotes

UPDATE mytable SET username="myuser" WHERE id=8 --Updating a row into a MySQL table

DELETE FROM mytable WHERE id=8 --Deleting a row into a MySQL table

SELECT * FROM mytable WHERE username = "myuser"; --Selecting rows based on conditions
in MySQL

SHOW databases; --Show list of existing databases

SHOW tables; --Show tables in an existing database

DESCRIBE databaseName.tableName; --Show all the fields of a table
```

```sql
DESCRIBE tableName; --or, if already using a database

CREATE USER 'user'@'localhost' IDENTIFIED BY 'some_password'; --Will create a user that can only connect on the local machine where the database is hosted.

CREATE USER 'user'@'%' IDENTIFIED BY 'some_password'; --Will create a user that can connect from anywhere (except the local machine).

GRANT SELECT, INSERT, UPDATE ON databaseName.* TO 'userName'@'localhost';

GRANT ALL ON *.* TO 'userName'@'localhost' WITH GRANT OPTION;

--As demonstrated above, *.* targets all databases and tables, databaseName.* targets all tables of the specific
--database. It is also possible to specify database and table like so databaseName.tableName.
--WITH GRANT OPTION should be left out if the user need not be able to grant other users privileges.
--ALL => SELECT INSERT UPDATE DELETE CREATE DROP , ...

SELECT DISTINCT `name`, `price` FROM CAR; --The DISTINCT clause after SELECT eliminates duplicate rows from the result set.

SELECT * FROM stack; --SELECT All COLUMNS FROM TABLE

SELECT * FROM stack WHERE username LIKE "%adm%"; --"adm" anywhere

SELECT * FROM stack WHERE username LIKE "adm%"; --Begins with "adm"

SELECT * FROM stack WHERE username LIKE "%adm"; --Ends with "adm"

SELECT * FROM stack WHERE username LIKE "adm_n"; --Just as the % character in a LIKE clause matches any number of characters, the _ character matches just one character

SELECT st.name,
st.percentage,
CASE WHEN st.percentage >= 35 THEN 'Pass' ELSE 'Fail' END AS `Remark`
FROM student AS st ;

SELECT st.name,
st.percentage,
IF(st.percentage >= 35, 'Pass', 'Fail') AS `Remark`
FROM student AS st ;

--This means : IF st.percentage >= 35 is TRUE then return 'Pass' ELSE return 'Fail'


SELECT username AS val FROM stack; --SQL aliases are used to temporarily rename a table or a column

SELECT username val FROM stack; --AS is optional

SELECT * FROM Customers ORDER BY CustomerID LIMIT 3; --Always use ORDER BY when using LIMIT;

SELECT * FROM Customers ORDER BY CustomerID LIMIT 2,1; --skips two records and returns one (LIMIT offset,count)

SELECT * FROM stack WHERE id BETWEEN 2 and 5; --greater than equal AND less than equal

SELECT * FROM stack WHERE id NOT BETWEEN 2 and 5;

SELECT * FROM stack WHERE username = "admin" AND password = "admin";

SELECT * FROM stack WHERE username IN (SELECT username FROM signups WHERE email IS NULL);

SELECT title FROM books WHERE author_id = (SELECT id FROM authors WHERE last_name = 'Bar' AND first_name = 'Foo');

```

```
--To make sure you don't get an error in your query you have to use backticks so your query becomes:

SELECT `users`.`username`, `groups`.`group` FROM `users`
SELECT student_name, AVG(test_score) FROM student GROUP BY `group`

IS NULL/IS NOT NULL

SELECT * FROM users ORDER BY id ASC LIMIT 2 --ASC (ascending) DESC (descending)
SELECT * FROM users ORDER BY id ASC LIMIT 2 OFFSET 3 = SELECT * FROM users ORDER BY id ASC LIMIT 3,2

CREATE DATABASE IF NOT EXISTS Baseball;
DROP DATABASE IF EXISTS Baseball; -- Drops a database if it exists, avoids Error 1008
DROP DATABASE xyz; -- If xyz does not exist, ERROR 1008 will occur
CREATE DATABASE Baseball CHARACTER SET utf8 COLLATE utf8_general_ci;
SELECT @@character_set_database as cset,@@collation_database as col; --The above shows the default CHARACTER SET and Collation for the database.

SHOW GRANTS FOR 'John123'@'%'; --show users privileges

--Setting Variables:
SET @var_string = 'my_var';
SET @var_num = '2'
SET @var_date = '2015-07-20';

SET @start_yearmonth = (SELECT EXTRACT(YEAR_MONTH FROM @start_date));
SET @end_yearmonth = (SELECT EXTRACT(YEAR_MONTH FROM @end_date));

SELECT GROUP_CONCAT(partition_name)
FROM information_schema.partitions p
WHERE table_name = 'partitioned_table'
AND SUBSTRING_INDEX(partition_name,'P',-1) BETWEEN @start_yearmonth AND @end_yearmonth
INTO @partitions;

SET @query =
CONCAT('CREATE TABLE part_of_partitioned_table (PRIMARY KEY(id))
SELECT partitioned_table.*
FROM partitioned_table PARTITION(', @partitions,')
JOIN users u USING(user_id)
WHERE date(partitioned_table.date) BETWEEN ', @start_date,' AND ', @end_date);
#prepare the statement from @query
PREPARE stmt FROM @query;
EXECUTE stmt;
--put the query in a variable. You need to do this, because mysql did not recognize my variable as a
--variable in that position. You need to concat the value of the variable together with the rest of the
--query and then execute it as a stmt

--Inserting multiple rows:
INSERT INTO `my_table` (`field_1`, `field_2`) VALUES
('data_1', 'data_2'),
('data_1', 'data_3'),
('data_4', 'data_5');

--DELETE/UPDATE Parameter:
--LOW_PRIORITY => If LOW_PRIORITY is provided, the delete will be delayed until there are no processes reading from the table
--IGNORE => If IGNORE is provided, all errors encountered during the delete are ignored
--LIMIT => It controls the maximum number of records to delete from the table.

TRUNCATE tableName;
--This will delete all the data and reset AUTO_INCREMENT index. It's much faster than DELETE FROM tableName on a
--huge dataset. It can be very useful during development/testing.

DELETE FROM table_name; --This will delete everything, all rows from the table. It is the most basic example of the syntax.
```

```
184
185 DELETE FROM `table_name` WHERE `field_one` = 'value_one' LIMIT 1; --LIMITing deletes
186
187 UPDATE people
188 SET name =
189 (CASE id WHEN 1 THEN 'Karl'
190 WHEN 2 THEN 'Tom'
191 WHEN 3 THEN 'Mary'
192 END)
193 WHERE id IN (1,2,3);
194 --When updating multiple rows with different values it is much quicker to use a bulk
    update.
195
196 UPDATE [ LOW_PRIORITY ] [ IGNORE ]
197 tableName
198 SET column1 = expression1,
199 column2 = expression2,
200 ...
201 [WHERE conditions]
202 [ORDER BY expression [ ASC | DESC ]]
203 [LIMIT row_count];
204 ---> Example
205 UPDATE employees SET isConfirmed=1 ORDER BY joiningDate LIMIT 10;
206
207 ORDER BY x ASC -- same as default
208 ORDER BY x DESC -- highest to lowest
209 ORDER BY lastname, firstname -- typical name sorting; using two columns
210 ORDER BY submit_date DESC -- latest first
211 ORDER BY submit_date DESC, id ASC -- latest first, but fully specifying order.
212
213 SELECT department, COUNT(*) AS "Man_Power"
214 FROM employees
215 GROUP BY department
216 HAVING COUNT(*) >= 10;
217
218 SELECT department, MIN(salary) AS "Lowest salary"
219 FROM employees
220 GROUP BY department;
221
222 SELECT customer, COUNT(*) as orders
223 FROM orders
224 GROUP BY customer
225 ORDER BY customer
226
227 --JOIN with subquery:
228 SELECT x, ...
229 FROM ( SELECT y, ... FROM ... ) AS a
230 JOIN tbl ON tbl.x = a.y
231 WHERE ...
232
233 SELECT ...
234 FROM ( SELECT y, ... FROM ... ) AS a
235 JOIN ( SELECT x, ... FROM ... ) AS b ON b.x = a.y
236 WHERE ...
237
238 --This will get all the orders for all customers:
239 SELECT c.CustomerName, o.OrderID
240 FROM Customers AS c
241 INNER JOIN Orders AS o
242 ON c.CustomerID = o.CustomerID
243 ORDER BY c.CustomerName, o.OrderID;
244
245 --This will count the number of orders for each customer:
246 SELECT c.CustomerName, COUNT(*) AS 'Order Count'
247 FROM Customers AS c
248 INNER JOIN Orders AS o
249 ON c.CustomerID = o.CustomerID
250 GROUP BY c.CustomerID;
251 ORDER BY c.CustomerName;
```

```
252
253  --Also, counts, but probably faster:
254  SELECT c.CustomerName,
255  ( SELECT COUNT(*) FROM Orders WHERE CustomerID = c.CustomerID ) AS 'Order Count'
256  FROM Customers AS c
257  ORDER BY c.CustomerName;
258
259  --List only the customer with orders.
260  SELECT c.CustomerName,
261  FROM Customers AS c
262  WHERE EXISTS ( SELECT * FROM Orders WHERE CustomerID = c.CustomerID )
263  ORDER BY c.CustomerName;
264
265  select name, email, phone_number
266  from authors
267  UNION / UNION ALL
268  select name, email, phone_number
269  from editors;
270  --Using union by itself will strip out duplicates, but unoin all show duplicates.
271
272  --If you need to sort the results of a UNION, use this pattern:
273  ( SELECT ... )
274  UNION
275  ( SELECT ... )
276  ORDER BY ...;
277
278  ( SELECT ... ORDER BY x LIMIT 40 )
279  UNION
280  ( SELECT ... ORDER BY x LIMIT 40 )
281  ORDER BY x LIMIT 30, 10;
282
283  --Arithmetic Operators:
284  SELECT 3+5; -> 8
285  SELECT 3-5; -> -2
286  SELECT 3 * 5; -> 15
287  SELECT 20 / 4; -> 5
288  SELECT 5 DIV 2; -> 2
289  SELECT 7 % 3; -> 1
290  SELECT 15 MOD 4 -> 3
291  SELECT PI(); -> 3.141593
292
293  SELECT SIN();
294  SELECT COS();
295  SELECT TAN();
296  SELECT COT();
297  SELECT RADIANS(90) -> 1.5707963267948966
298  SELECT SIN(RADIANS(90)) -> 1
299  SELECT DEGREES(1), DEGREES(PI()) -> 57.29577951308232, 180
300
301  SELECT ROUND(4.51) -> 5
302  SELECT ROUND(4.49) -> 4
303  SELECT ROUND(-4.51) -> -5
304
305  --To round up a number use either the CEIL() or CEILING() function:
306  SELECT CEIL(1.23) -> 2
307  SELECT CEILING(4.83) -> 5
308
309  --To round down a number, use the FLOOR() function:
310  SELECT FLOOR(1.99) -> 1
311  SELECT FLOOR(-1.01), CEIL(-1.01) -> -2 and -1
312  SELECT FLOOR(-1.99), CEIL(-1.99) -> -2 and -1
313
314  --To raise a number x to a power y, use either the POW() or POWER() functions:
315  SELECT POW(2,2); => 4
316  SELECT POW(4,2); => 16
317
318  --Use the SQRT() function. If the number is negative, NULL will be returned:
319  SELECT SQRT(16); -> 4
320  SELECT SQRT(-3); -> NULL
```

```
--To generate a random number in the range a <= n <= b, you can use the following formula:
FLOOR(a + RAND() * (b - a + 1))

--A simple way to randomly return the rows in a table:
SELECT * FROM tbl ORDER BY RAND();

--Absolute Value:
SELECT ABS(2); -> 2
SELECT ABS(-46); -> 46

--Sign:
-1 when n < 0 SELECT SIGN(-3); -> -1
0 when n = 0 SELECT SIGN(0); -> 0
1 when n > 0 SELECT SIGN(42); -> 1

--String operations:

ASCII() --Return numeric value of left-most character
BIN() --Return a string containing binary representation of a number
BIT_LENGTH() --Return length of argument in bits
CHAR() --Return the character for each integer passed
CHAR_LENGTH() --Return number of characters in argument
CHARACTER_LENGTH() --Synonym for CHAR_LENGTH()
CONCAT() --Return concatenated string
CONCAT_WS() --Return concatenate with separator
ELT() --Return string at index number
EXPORT_SET() --Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD() --Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET() --Return the index position of the first argument within the second argument
FORMAT() --Return a number formatted to specified number of decimal places
FROM_BASE64() --Decode to a base-64 string and return result
HEX() --Return a hexadecimal representation of a decimal or string value
INSERT() --Insert a substring at the specified position up to the specified number of characters
INSTR() --Return the index of the first occurrence of substring
LCASE() --Synonym for LOWER()
LEFT() --Return the leftmost number of characters as specified
LENGTH() --Return the length of a string in bytes
LIKE --Simple pattern matching
LOAD_FILE() --Load the named file
LOCATE() --Return the position of the first occurrence of substring
LOWER() --Return the argument in lowercase
LPAD() --Return the string argument, left-padded with the specified string
LTRIM() --Remove leading spaces
MAKE_SET() --Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH --Perform full-text search
MID() --Return a substring starting from the specified position
NOT LIKE --Negation of simple pattern matching
NOT --REGEXP Negation of REGEXP
OCT() --Return a string containing octal representation of a number
OCTET_LENGTH() --Synonym for LENGTH()
ORD() --Return character code for leftmost character of the argument
POSITION() --Synonym for LOCATE()
QUOTE() --Escape the argument for use in an SQL statement
REGEXP --Pattern matching using regular expressions
REPEAT() --Repeat a string the specified number of times
REPLACE() --Replace occurrences of a specified string
REVERSE() --Reverse the characters in a string
RIGHT() --Return the specified rightmost number of characters
RLIKE --Synonym for REGEXP
RPAD() --Append string the specified number of times
RTRIM() --Remove trailing spaces
SOUNDEX() --Return a soundex string
SOUNDS LIKE --Compare sounds
```

```
SPACE() --Return a string of the specified number of spaces
STRCMP() --Compare two strings
SUBSTR() --Return the substring as specified
SUBSTRING() --Return the substring as specified
SUBSTRING_INDEX() --Return a substring from a string before the specified number of occurrences of the delimiter
TO_BASE64() --Return the argument converted to a base-64 string
TRIM() --Remove leading and trailing spaces
UCASE() --Synonym for UPPER()
UNHEX() --Return a string containing hex representation of a number
UPPER() --Convert to uppercase
WEIGHT_STRING() --Return the weight string for a string

SUBSTRING(str, start_position, length)
SELECT SUBSTRING('foobarbaz', 4, 3); -- 'bar'
SELECT SUBSTRING('foobarbaz', FROM 4 FOR 3); -- 'bar'

REPLACE(str, from_str, to_str)
REPLACE('foobarbaz', 'bar', 'BAR') -- 'fooBARbaz'
REPLACE('foobarbaz', 'zzz', 'ZZZ') -- 'foobarbaz'

SELECT SYSDATE();
--This function returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format,
--depending on whether the function is used in a string or numeric context. It returns the date and time in the current time zone.

SELECT NOW();
--This function is a synonym for SYSDATE().

SELECT CURDATE();
--This function returns the current date, without any time, as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending
--on whether the function is used in a string or numeric context. It returns the date in the current time zone.

--Regular Expressions:
-- use => REGEXP / RLIKE
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^N'; --Select all employees whose FIRST_NAME starts with N.
SELECT * FROM employees WHERE PHONE_NUMBER REGEXP '4569$'; --Select all employees whose PHONE_NUMBER ends with 4569.
SELECT * FROM employees WHERE FIRST_NAME NOT REGEXP '^N'; --Select all employees whose FIRST_NAME does not start with N.
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^[ABC]'; --Select all employees whose FIRST_NAME starts with A or B or C.
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^[ABC]|[rei]$'; --Select all employees whose FIRST_NAME starts with A or B or C and ends with r, e, or i

-- Create View :
CREATE VIEW test.v AS SELECT * FROM t;

CREATE VIEW myview AS
SELECT a.*, b.extra_data FROM main_table a
LEFT OUTER JOIN other_table b
ON a.id = b.id

DROP VIEW test.v;

--Cloning an existing table:

CREATE TABLE ClonedPersons LIKE Persons; --The new table will have exactly the same structure as the original table, including indexes and column attributes.

CREATE TABLE ClonedPersons SELECT * FROM Persons; --As well as manually creating a table, it is also possible to create table by selecting data from another table

--You can use any of the normal features of a SELECT statement to modify the data as you go:
```

```
CREATE TABLE ModifiedPersons
SELECT PersonID, FirstName + LastName AS FullName FROM Persons
WHERE LastName IS NOT NULL;

CREATE TABLE ModifiedPersons (PRIMARY KEY (PersonID))
SELECT PersonID, FirstName + LastName AS FullName FROM Persons
WHERE LastName IS NOT NULL;

-- create a table from another table from another database with all attributes:
CREATE TABLE stack2 AS SELECT * FROM second_db.stack;

--ALTER :
Create table stack(
id_user int NOT NULL,
username varchar(30) NOT NULL,
password varchar(30) NOT NULL
);

ALTER TABLE stack ADD COLUMN submit date NOT NULL; -- add new column
ALTER TABLE stack DROP COLUMN submit; -- drop column
ALTER TABLE stack MODIFY submit DATETIME NOT NULL; -- modify type column
ALTER TABLE stack CHANGE submit submit_date DATETIME NOT NULL; -- change type and name of column
ALTER TABLE stack ADD COLUMN mod_id INT NOT NULL AFTER id_user; -- add new column after existing column

ALTER TABLE your_table_name AUTO_INCREMENT = 101; --Change auto-increment value

RENAME TABLE `<old name>` TO `<new name>`; --Renaming a MySQL table
ALTER TABLE `<old name>` RENAME TO `<new name>`; --Renaming a MySQL table

ALTER TABLE TABLE_NAME ADD INDEX `index_name` (`column_name`) --To improve performance one might want to add indexes to columns

ALTER TABLE TABLE_NAME ADD INDEX `index_name` (`col1`,`col2`) --altering to add composite (multiple column) indexes

--Changing the type of a primary key column:
ALTER TABLE fish_data.fish DROP PRIMARY KEY;
ALTER TABLE fish_data.fish MODIFY COLUMN fish_id DECIMAL(20,0) NOT NULL PRIMARY KEY;

--Change column definition:
users (
firstname varchar(20),
lastname varchar(20),
age char(2)
)
ALTER TABLE users CHANGE age age tinyint UNSIGNED NOT NULL;

--Renaming a column in a MySQL table:
ALTER TABLE `<table name>` CHANGE `<old name>` `<new name>` <column definition>;

DROP TABLE tbl;
DROP TABLE Database.table_name

--Stored procedure with IN, OUT, INOUT parameters:
--An "IN" parameter passes a value into a procedure. The procedure might modify the value, but the modification is
--not visible to the caller when the procedure returns.
--An "OUT" parameter passes a value from the procedure back to the caller. Its initial value is NULL within the
--procedure, and its value is visible to the caller when the procedure returns.
--An "INOUT" parameter is initialized by the caller, can be modified by the procedure, and any change made by the
--procedure is visible to the caller when the procedure returns.

DELIMITER $$
DROP PROCEDURE IF EXISTS sp_nested_loop$$
CREATE PROCEDURE sp_nested_loop(IN i INT, IN j INT, OUT x INT, OUT y INT, INOUT z INT)
```

```
BEGIN
DECLARE a INTEGER DEFAULT 0;
DECLARE b INTEGER DEFAULT 0;
DECLARE c INTEGER DEFAULT 0;
WHILE a < i DO
WHILE b < j DO
SET c = c + 1;
SET b = b + 1;
END WHILE;
SET a = a + 1;
SET b = 0;
END WHILE;
SET x = a, y = c;
SET z = x + y + z;
END $$
DELIMITER ;

--Invokes (CALL) the stored procedure:
SET @z = 30;
call sp_nested_loop(10, 20, @x, @y, @z);
SELECT @x, @y, @z;

--Create a Function:
DELIMITER ||
CREATE FUNCTION functionname()
RETURNS INT
BEGIN
RETURN 12;
END;
||
DELIMITER ;

--Execution this function is as follows:
SELECT functionname();

DELIMITER $$
CREATE FUNCTION add_2 ( my_arg INT )
RETURNS INT
BEGIN
RETURN (my_arg + 2);
END;
$$
DELIMITER ;

--Note the use of a different argument to the DELIMITER directive. You can actually use any character sequence that
--does not appear in the CREATE statement body, but the usual practice is to use a doubled non-alphanumeric character such as \\, || or $$.

--"Indexing makes queries faster" This is the simplest definition of indexes.

-- Create an index for column 'name' in table 'my_table':
CREATE INDEX idx_name ON my_table(name);

-- Creates a unique index for column 'name' in table 'my_table':
CREATE UNIQUE INDEX idx_name ON my_table(name);

--Create composite index:
CREATE INDEX idx_mycol_myothercol ON my_table(mycol, myothercol);

-- Drop an index for column 'name' in table 'my_table'
DROP INDEX idx_name ON my_table;

SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
PREPARE stmt2 FROM @s;
SET @a = 6;
SET @b = 8;
EXECUTE stmt2 USING @a, @b;

```

```
--Create simple table with a primary key and JSON field
CREATE TABLE table_name (
id INT NOT NULL AUTO_INCREMENT,
json_col JSON,
PRIMARY KEY(id)
);

INSERT INTO
table_name (json_col)
VALUES
('{"City": "Galle", "Description": "Best damn city in the world"}');

--Updating a JSON field:
UPDATE
myjson
SET
dict=JSON_ARRAY_APPEND(dict,'$.variations','scheveningen')
WHERE
id = 2;

mysqladmin -u root -p'old-password' password 'new-password' --Change root password

DROP DATABASE database_name;
DROP SCHEMA database_name;

--TRIGGERS:
--There are two trigger action time modifiers :
--BEFORE trigger activates before executing the request,
--AFTER trigger fire after change.

--There are three events that triggers can be attached to:
--INSERT
--UPDATE
--DELETE

--Before Insert trigger example
DELIMITER $$
CREATE TRIGGER insert_date
BEFORE INSERT ON stack
FOR EACH ROW
BEGIN
-- set the insert_date field in the request before the insert
SET NEW.insert_date = NOW();
END;
$$
DELIMITER ;


--Before Update trigger example
DELIMITER $$
CREATE TRIGGER update_date
BEFORE UPDATE ON stack
FOR EACH ROW
BEGIN
-- set the update_date field in the request before the update
SET NEW.update_date = NOW();
END;
$$
DELIMITER ;


--After Delete trigger example
DELIMITER $$
CREATE TRIGGER deletion_date
AFTER DELETE ON stack
FOR EACH ROW
BEGIN
-- add a log entry after a successful delete
INSERT INTO log_action(stack_id, deleted_date) VALUES(OLD.id, NOW());
```

```
END;
$$
DELIMITER ;

SET [GLOBAL | SESSION] group_concat_max_len = val;
--Setting the GLOBAL variable will ensure a permanent change, whereas setting the SESSION variable will set the value for the current session.

--EVENTS :
--Think of Events as Stored Procedures that are scheduled to run on recurring intervals.
DROP EVENT IF EXISTS `delete7DayOldMessages`;
DELIMITER $$
CREATE EVENT `delete7DayOldMessages`
ON SCHEDULE EVERY 1 DAY STARTS '2015-09-01 00:00:00'
ON COMPLETION PRESERVE
DO BEGIN
DELETE FROM theMessages
WHERE datediff(now(),updateDt)>6; -- not terribly exact, yesterday but <24hrs is still 1 day
-- Other code here
END$$

DROP EVENT IF EXISTS `Every_10_Minutes_Cleanup`;
DELIMITER $$
CREATE EVENT `Every_10_Minutes_Cleanup`
ON SCHEDULE EVERY 10 MINUTE STARTS '2015-09-01 00:00:00'
ON COMPLETION PRESERVE
DO BEGIN
DELETE FROM theMessages
WHERE TIMESTAMPDIFF(HOUR, updateDt, now())>168; -- messages over 1 week old (168 hours)
-- Other code here
END$$
DELIMITER ;

DROP EVENT someEventName; -- Deletes the event and its code

--ENUM:
reply ENUM('yes', 'no')
gender ENUM('male', 'female', 'other', 'decline-to-state')

ALTER TABLE tbl MODIFY COLUMN type ENUM('fish','mammal','bird','insect');

--A transaction is a sequential group of SQL statements such as select,insert,update or delete, which is performed as one single work unit.
--In other words, a transaction will never be complete unless each individual operation within the group is successful.
--If any operation within the transaction fails, the entire transaction will fail.

--Properties of Transactions:
Transactions have the following four standard properties
--Atomicity: ensures that all operations within the work unit are completed successfully
--otherwise, the ransaction is aborted at the point of failure, and previous operations are rolled back to their former state.
--Consistency: ensures that the database properly changes states upon a successfully committed transaction.
--Isolation: enables transactions to operate independently of and transparent to each other.
--Durability: ensures that the result or effect of a committed transaction persists in case of a system failure

START TRANSACTION;
SET @transAmt = '500';
SELECT @availableAmt:=ledgerAmt FROM accTable WHERE customerId=1 FOR UPDATE;
UPDATE accTable SET ledgerAmt=ledgerAmt-@transAmt WHERE customerId=1;
UPDATE accTable SET ledgerAmt=ledgerAmt+@transAmt WHERE customerId=2;
COMMIT;

--PARTITION:
--BY RANGE
```

```sql
ALTER TABLE employees PARTITION BY RANGE (store_id) (
PARTITION p0 VALUES LESS THAN (6),
PARTITION p1 VALUES LESS THAN (11),
PARTITION p2 VALUES LESS THAN (16),
PARTITION p3 VALUES LESS THAN MAXVALUE
);
--BY LIST
ALTER TABLE employees PARTITION BY LIST (store_id) (
PARTITION pNorth VALUES IN (3,5,6,9,17),
PARTITION pEast VALUES IN (1,2,10,11,19,20),
PARTITION pWest VALUES IN (4,12,13,14,18),
PARTITION pCentral VALUES IN (7,8,15,16)
);

--LOAD DATA INFILE:
--Data is like these:
--1;max;male;manager;12-7-1985
--2;jack;male;executive;21-8-1990
...
--1000000;marta;female;accountant;15-6-1992

LOAD DATA INFILE 'path of the file/file_name.txt'
INTO TABLE employee
FIELDS TERMINATED BY ';' //specify the delimiter separating the values
LINES TERMINATED BY '\r\n'
(id,name,sex,designation,dob)

--Load data with duplicates:
LOAD DATA INFILE 'path of the file/file_name.txt'
REPLACE INTO TABLE employee;

LOAD DATA INFILE 'path of the file/file_name.txt'
IGNORE INTO TABLE employee;

--Import a CSV file into a MySQL table:
load data infile '/tmp/file.csv'
into table my_table
fields terminated by ','
optionally enclosed by '"'
escaped by '"'
lines terminated by '\n'
ignore 1 lines; -- skip the header row

--Temporary tables could be very useful to keep temporary data.

--->Basic temporary table creation
CREATE TEMPORARY TABLE tempTable1(
id INT NOT NULL AUTO_INCREMENT,
title VARCHAR(100) NOT NULL,
PRIMARY KEY ( id )
);

--->Temporary table creation from select query
CREATE TEMPORARY TABLE tempTable1
SELECT ColumnName1,ColumnName2,... FROM table1;

CREATE TEMPORARY TABLE IF NOT EXISTS tempTable1
SELECT ColumnName1,ColumnName2,... FROM table1;

DROP TEMPORARY TABLE tempTable1;
DROP TEMPORARY TABLE IF EXISTS tempTable1;
```