MySQL Notes

Syntax

Comments

Either `--` for single-line comments or `/* */` for multiline comments.

Case

mySQL commands are not case sensitive. However, it is recommended to use UPPERCASE for commands.

Semicolons

Semicolons should be used to end a command.

Select/View

```
SELECT columnA[, columnB...] FROM tableName [WHERE condition [AND anotherCondition...]] [ORDER BY colA [DESC/ASC] [, colB [DESC/ASC]...]
```

You can also use the `SELECT` statement to perform calculations, such as `1+2`.

Distinct

You can use the `DISTINCT` keyword to only select distinct or unique values: `SELECT DISTINCT colName FROM tableName`

As

`SELECT columnA AS 'string'` allows you to change the name of the column, referencing it using an alias.

If the string does not have spaces or other syntax, it does not need to be in single quotes, and can be referenced in that command:

```
SELECT colA AS someText FROM tableName ORDER BY tableName
```

Regular Expressions

```
SELECT ... WHERE col REGEXP 'aRegExp'
```

Update Row:

```
UPDATE tableName SET Column = Value [WHERE Conditional]
```

Insert Row:

```
INSERT INTO tableName [(colA, colB, colD...)] VALUES (valA, valB, valD...)```
The `[(colA, colB, colD...)]` is only necessary if not all values will be entered.
```

However, by using the statement below, you can insert rows from another table:

```
INSERT INTO tableName(colA, colB, colC) SELECT FROM table2Name colJ, colD, colZ [WHERE Conditional]
```

Delete Row:

```
DELETE FROM tableName [WHERE Conditional]
```

Ordering SELECT statements:

```
... ORDER BY col1 [ASC/DESC], col2...
```

Limit

```
SELECT colA ... FROM tableName LIMIT n [OFFSET j]
```

This limits the query to `n` rows, and the optional `OFFSET` parameter means it returns `n` rows starting at the `j` th row.

Create Table:

```
CREATE TABLE tableName (
    colA colType,
    colB colType,
    colC colType,
...);
/*No comma after the last one*/
```

This defines all the options for the table. Inside the brackets has the table definition for all the columns.

To set a column to an id, you can use the `SERIAL` keyword:

```
...
    id SERIAL,
    /* SERIAL = BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE */
...
```

Describe

`DESCRIBE tableName` is a mySQL specific command which outputs information about a table.

Delete Table:

`DROP TABLE tableName` You may want to use `DROP TABLE IF EXISTS tableName`

Concatonating Strings:

Just have a space between two strings (USE SINGLE QUOTES). For example, `SELECT 'Hello' ' World'` outputs `"Hello World"`

Boolean Operators:

`<>` means is not equal to
`=` means equal to
`>=`, `>`, `<=`, `<` are the standard comparison operators
`IS [NOT] NULL` tells you if a value is (not) null

Null

Null is the lack of a value, and so you cannot compare or test. Thus, something like `SELECT * FROM tableName where colA = NULL` will not work. You must use `...WHERE colA IS [NOT] NULL`.

Databases

This is very similar to with tables.

To create a database, use `CREATE DATABASE databaseName;`

To delete: `DROP DATABASE databaseName;`

To use a table in the database, you can use: `USE tableName;`

SHOW TABLE STATUS

The `SHOW TABLE STATUS;` command gives you information about all the tables in a database. You can use selectors such as `LIKE` to show information about some tables only. This command is MySQL specific.

`SHOW CREATE TABLE tablename` is another command, which outputs the command mySQL used to create the table.

`SHOW VARIABLES` is a command which shows internal mySQL variables.

Index

To use a column as a index, use `INDEX(columnName)` or `INDEX indexName(columnName)` in the declaration of a table:

```
DROP TABLE IF EXISTS tableName;
CREATE TABLE tableName (
    id INTEGER,
    a VARCHAR(255),
    b VARCHAR(255),
    INDEX [nameForIndex](a)
);
```

A index can have unique or non-unique values, meaning that the value of the index is always unique.

Column Constraints/Behaviour

In the table declaration, you can define the behaviour of columns.
To disallow null values, use: `colName colType NOT NULL`. A error will occur if you try and insert a row with a null value.
To give it a default value, use `colName colType DEFAULT defVal`
Unique is the same as making a column a index: `colName colType UNIQUE`. However, a column could have multiple `NULL` values as null is not a value. Use `UNIQUE` along with `NOT NULL` in order to stop this.

ID

A ID column must be set up to be `UNIQUE NOT NULL`. A `PRIMARY KEY` constraint has both of these, and being the *primary* key, there can only be one per table. To generate sequential values, use `AUTO_INCREMENT`.
However, you can shortcut the `UNIQUE NOT NULL...` using the `SERIAL` data type, which expands to `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`:

```
CREATE TABLE tableName(
id SERIAL,
...
PRIMARY KEY(id));
```

Foreign Key Constraints

Often, you reference columns, such as the ID of other tables. However, errors can occur if a row you select does not exist. Thus, foreign key constraints exist:

```
CREATE TABLE tableName(
    colA type,
    colB type,
    FOREIGN KEY (colInThisTable) REFERENCES tableName(colInOtherTable)
);
```

This means that in addition to disallowing `INSERT INTO` row creation into `tableName`, it can disallow modifications such as the removal of rows into another table if that row is referenced. In addition, `DROP TABLE` statements must be made in a specific order to prevent reference errors.

Alter table after definition

Add Column

You can add a column after the definition using:

```
ALTER TABLE tableName ADD colName colType;
```

By default, it adds the column to the end. To specify the position, use:

```
ALTER TABLE tableName ADD colName ColType [AFTER anotherCol]/[FIRST]
```

Also by default, all the values in the column are initialized to `NULL`. To change this, use:

```
ALTER TABLE tableName ADD colName colType DEFAULT defVal
```

Anything after the `ALTER TABLE tableName` is the same as in a normal table definition.

Delete Column

To delete a column, use the similar syntax of:

```
ALTER TABLE tableName DROP colName;
```

Be warned that it will delete data.

Data types

This differs across systems. For mySQL, there are three fundamental data types: Numeric, String, and Date/Time. There is a hard limit of 65535 bytes of storage per row.

Numeric

Integer

There are 5 types of integers, each with signed and unsigned.

Type	Storage (bytes)
TINYINT	1
SMALLINT	2
MEDIUMINT	3
INTEGER/INT	4
BIGINT	8

Decimal

Used for fixed precision values: `DECIMAL(p, s)`. The `p` specifies the precision- the number of digits stored, both before and after the decimal point. the `s` specifies the scale: the position of the decimal point. `0` means there is no decimal point, `2` means 2 decimal places. For example, `123456.789` would be of type `DECIMAL(9,3)`. `DECIMAL` and `NUMERIC` are both accepted. By default, `DECIMAL` equals `DECIMAL(10,0)`, so can store a number such as `1234567890`.

Floating point

Type	Range	Number of bits
FLOAT/REAL	+/- 3.4e38	24
DOUBLE	+/- 1.79e308	53

Strings

Character Strings

There are two types, `CHAR(n)` and `VARCHAR(n)`. The first is a fixed length string, and if the string is less than `n` characters long, it is padded with spaces, which are removed when retrieved. The latter's `n` value specifies the maximum size of the string, which can store `n+1` characters.

Binary Strings

Basically the same as character strings, which types `BINARY(n)` and `VARBINARY(n)`. However, they have no character set and use binary collation. The first of the two are padded with 0's instead of spaces, which are stripped when being removed.

Large Object Storage

Type	Storage (bytes)
TINYBLOB, TINYTEXT	up to 256 + 1
BLOB, TEXT	up to 65536 + 2
MEDIUMBLOB, MEDIUMTEXT	up to 16777216 + 3
LONGBLOB, LONGTEXT	up to 4294967296 + 4

The size of these columns do not count against the hard limit as they are stored separately. `BLOB`'s have no character set, which `TEXT`'s do.

Date and time

Standard SQL date-time format is `YYYY-MM-DD HH:MM:SS[.DECIMAL]`.

The `TIMESTAMP` type is one that is updated on the creation and update of a row. It is of date-time format, and sets itself to the value of `CURRENT_TIMESTAMP`, which has the current date and time.

Bit

This is a datatype designed to use as little storage as possible. This specifies the number of bits: `BIT(n)`.

Thus, the largest value it can store is $2^n - 1$. If a value is greater than the maximum, it gets the value of the maximum.

Booleans

The `BOOL` or `BOOLEAN` type maps to a `TINYINT`. It can have the value `TRUE` or `FALSE`, which maps to `1` or `0`.

Enumeration

`ENUM` is a list of strings, and can type one of the values specified:

```
DROP TABLE IF EXISTS tableName;
CREATE TABLE tableName (
  colName ENUM('strA', 'strB', 'strC'...)
);
INSERT INTO tableName (colName) VALUES ('strA OR strB OR strC... BUT NOT ANYTHING ELSE');
INSERT INTO tableName (colName) VALUES(1 OR 2 OR 3...);
```

Typing the string is the same as typing the index of of the string, starting at 1. It saves space as only an integer is stored.

Set

A `SET` type is almost the same as `ENUM`, except that a cell can have more than one value in the list. Separate the values with a comma INSIDE the string:

```
DROP TABLE IF EXISTS tableName;
CREATE TABLE tableName (
  colName SET('strA', 'strB', 'strC'...)
);
INSERT INTO tableName (colName) VALUES ('strA, strB...');
```

This can also be represented as a number. With `n` values, the maximum integer value for the `SET` will be $2^n - 1$.

A Subset of String Functions

Length

`LENGTH(str)` gives you the length of a string. Note that accented characters and other characters such as emoji may be longer than one character, as the LENGTH `function counts the number of bytes, and some characters take up more than one byte in UTF-8.`

If you use `CHAR_LENGTH(str)`, it counts the number of characters. However, it will take a bit longer. `CHARACTER_LENGTH` is the same as `CHAR_LENGTH`.

Substring

The `LEFT(str, n)` function gets the first `n` characters from a string. For example, `LEFT('Hello World', 4)` will return `Hell`.

The `RIGHT(str, n)` is similar, getting the last `n` characters. `RIGHT('Hello World', 4)` returns `orld`.

`MID(str, n, o)` returns a string `o` characters long starting from the `n`th character. `MID('Hello World', 2, 4)` will return `ello`.

Concatenate

The `CONCAT(strA, strB...)` function concatenates, or joins two or more strings together. The `CONCAT_WS(sep, strA, strB...)` concatenates two or more strings together with the `sep` string in between two strings.

Locate

The `LOCATE(strA, strB)` function finds the location of `strA` in `strB`. For example, `LOCATE('Hello', 'ABCD Hello World')` will return `6` as the 6th character in `strB` is the start of the match. A value of `0` means there was no match, as it is not zero-indexed: the first character is `1`.

Miscellaneous functions

`UPPER(str)` and `LOWER(str)` return the upper and lower case values of a string. This also works for accented characters in UTF-8.

The `REVERSE(str)` reverses a string.

A Subset of Numeric Functions

`+, -, *` are some basic numeric operators. The `/` divides, returning a float, while the `DIV` keyword returns an integer.

`MOD`, or `%` returns the remainder of a number.

To get the power, use `POWER(base, exp)` or `POW(base, exp)`, not `^`.

The absolute value function is `ABS(int)`. `SIGN` gives you the sign of a number, returning `-1` or `1` for negative or positive numbers.

The `CONV(valA, baseA, baseB)` converts a number, `valA` from `baseA` to `baseB`. The limit for the bases is from between base 2 and 36 (the number of alphanumeric characters).

`PI()` returns pi

Rounding

`ROUND(n [, dp])` rounds to nearest whole number, or if the second argument is filled in, to the nearest `dp` decimal points.

`TRUNCATE(n [,dp])` truncates, or effectively rounds down `n` to either a whole number or to `dp` decimal places.

`CEIL(n)` and `FLOOR(n)` always rounds up or down respectively.

Random

The `RAND()` returns a random number between 0 and 1. There is a optional parameter of a seed.

A Subset of Date and time functions

Current timestamp

`NOW()` returns the current timestamp. This is equivalent to `CURRENT_TIMESTAMP[()]`. `UNIX_TIMESTAMP` gives the number of seconds since epoch.

Day of

The function, `DAYNAME(datetime)` returns a string representing the name of the day of the week.

`DAYOFMONTH(datetime)` returns the day of the month

`DAYOFWEEK(datetime)` returns an integer representing the day of the week. `2` is Monday.

`DAYOFYEAR(datetime)` returns the number of days since January 1.

`MONTHNAME(datetime)` returns a string representing the name of the month.

Modifying

`TIME_TO_SEC(datetime)` gives the number of seconds since midnight that day. The timestamp can be either date and time or just time.

`SEC_TO_TIME(sec)` does the opposite, returning a time-only timestamp, which may be larger than 24 hours.

`ADDTIME(datetime, time)` adds a datetime/time and time together, returning either a datetime or time value.

`SUBTIME(datetime, time)` does the same, subtracting `time` from `datetime.` The result can be negative.

`ADDDATE(date, INTERVAL n units)` adds `n` units to `date`. `n` can be negative.

The `INTERVAL` keyword can take the values of: `MICROSECOND, SECOND, MINUTE, HOUR, DAY, MONTH, QUARTER, YEAR`. `MICROSECOND` may not be supported, and note that everything is singular.

`SUBDATE(date, INTERVAL n units)` does the same, except subtracting. It is essentially the same as `ADDDATE(date, INTERVAL -n units)`.

Time zones

`SET time_zone = tz_timezone e.g. 'Pacific/Auckland'`

This sets the session time zone setting.

Formatting dates

`DATE_FORMAT(datetime, 'str')` outputs a formatted string. The following options apply to the `str`:

Format	Description
`%a`	Abbreviated weekday name (Sun-Sat)
`%b`	Abbreviated month name (Jan-Dec)
`%c`	Month, numeric (0-12)
`%D`	Day of month with English suffix (0th, 1st, 2nd, 3rd, 4th...)
`%d`	Day of month, numeric (00-31)
`%e`	Day of month, numeric (0-31)
`%f`	Microseconds (000000-999999)
`%H`	Hour (00-23)
`%h`	Hour (01-12)
`%I`	Hour (01-12)
`%i`	Minutes, numeric (00-59)
`%j`	Day of year (001-366)
`%k`	Hour (0-23)
`%l`	Hour (1-12)
`%M`	Month name (January-December)
`%m`	Month, numeric (00-12)
`%p`	AM or PM
`%r`	Time, 12-hour (hh:mm:ss followed by AM or PM)
`%S`	Seconds (00-59)
`%s`	Seconds (00-59)
`%T`	Time, 24-hour (hh:mm:ss)
`%U`	Week (00-53) where Sunday is the first day of week
`%u`	Week (00-53) where Monday is the first day of week
`%V`	Week (01-53) where Sunday is the first day of week, used with `%X`
`%v`	Week (01-53) where Monday is the first day of week, used with `%x`
`%W`	Weekday name (Sunday-Saturday)
`%w`	Day of the week (0=Sunday, 6=Saturday)
`%X`	Year for the week where Sunday is the first day of week, four digits, used with `%V`

Format	Description
`%x`	Year for the week where Monday is the first day of week, four digits, used with `%v`
`%Y`	Year, numeric, four digits
`%y`	Year, numeric, two digits

Example: `DATE_FORMAT(NOW(), 'Currently %h:%i:%s%p, %W, %D of %M, %Y')`

Aggregate Functions

Functions which run across rows. Examples of these include the `COUNT(column)` functions, which counts the number of non-NULL values in a column, or `*` for the number of rows in a table. The `DISTINCT` tells you how many distinct values are in a column. Use it as `SELECT COUNT(DISTINCT column) FROM tableName.`

The `GROUP BY` selector groups together elements with the same value in some column. This is often used in conjuction with aggregate functions. For example, one ould be: `SELECT SUM(columnA), columnB FROM tableName GROUP BY columnB;`

`GROUP_CONCAT([DISTINCT] colName [SEPARATOR 'sep'])` is another function which concatonates the values of a column into a single string.

`AVG(col)`, `MIN(col)`, `MAX(col)`, `STD(col)` (standard deviation) and `SUM(col)` are some other aggregate functions.

Conditional Expressions

There are two similar syntaxes for a kind of if-else statement:

```
SELECT
  CASE WHEN boolExp THEN valA [ELSE valB] END [OTHER STUFF],
  CASE col WHEN someVal THEN valA [ELSE valB] END [AS alias...]
  FROM tableName
;
```

The column you change in the `THEN valA`/`ELSE valB` must be the referenced column in the boolean statement.

The `END` ends the conditional statement so that other stuff, such as alias etc. can be added. The `ELSE` is optional.

Transactions

A transaction allows you to check that everything inside it is correct and can occur (i.e. adding non-unique value to a unique column) before either committing or rolling-back the table.

```
START TRANSACTION;
...
COMMIT / ROLLBACK;
```

You can use a script to start some changes to a database, and if a error is returned or you are missing data, you can roll back changes to maintain the integrity of the database. Using transactions also increases the time taken for large changes.

Triggers

A event that occurs when some behavior occurs in a table, such as when a row in another table is updated or inserted. For example:

```
...
CREATE TRIGGER triggerName AFTER INSERT ON someTable
 FOR EACH ROW /*MySQL requires the FOR EACH ROW*/
    UPDATE otherTable SET colInOtherTable = NEW.colName WHERE id = NEW.id /*You probably have a
column with the same ID that links the tables */
;
```

The `NEW` is a pseudo-table, or rather, a single row which points to the row that was updated or inserted. You can also use triggers to log changes. For example:

```
CREATE TABLE log(id SERIAL, stamp TIMESTAMP, eventType VARCHAR(255), tableName VARCHAR(255),
rowID INT);

...
CREATE TRIGGER triggerName AFTER INSERT ON someTable
...
      INSERT INTO log (eventType, tableName, rowID) VALUES ('INSERT', 'someTable', NEW.id);
...
```

The `TIMESTAMP` will be automatically created on the insert, so you have the time, table, row affected and type of change to the table.

You can also `DROP` triggers. However, if you drop the table associated with the trigger, that trigger is also automatically dropped. The command to `DROP` triggers is similar: `DROP TRIGGER [IF EXISTS] triggerName;`

Preventing changes

You can also use triggers to stop updates using `BEFORE UPDATE`.

```
CREATE TABLE tableA(id SERIAL, ...);
CREATE TABLE tableB(id SERIAL, idForTableA INT, allowChanges INT...);
INSERT INTO tableB (..., allowChanges) VALUES (..., 0); /*allowChanges is false*/


DELIMITER // /*Allows the use of semicolons without terminating the statement. The delimiter is
now '//' */
CREATE TRIGGER triggerName BEFORE UPDATE ON tableA
  FOR EACH ROW
  BEGIN
    IF (SELECT allowChanges FROM tableB WHERE idForTableA = NEW.id) = 0 THEN
      /*Get value of allowChanges where id in A equals id in B. If false, throw error*/
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error messaeg'; /*Error message which makes
transaction fail*/
    END IF;
  END
// /*The statement is now terminated*/
DELIMITER; /*Changes the terminator/delimiter back to a semicolon*/

TRANSACTION;
UPDATE tableB set colA = otherVal WHERE id = someInt; //Will fail
COMMIT; /*Will not commit if there is an error*/
```

Subselects

The result of a `SELECT` statement effectively returns a table. Thus, you can use a `SELECT` statement as a data source for another `SELECT` statement, called a subselect.

```
SELECT sub.colA, sub.ColB, sub.colC FROM (
  SELECT subX As colA, subY AS colB, subZ AS colC FROM tableName
) AS sub;
```

Inside the `()`, you have a select statement selecting three columns from the table, `tableName` and are calling the table generated from the `SELECT` as `sub`. Then the outer `SELECT` statement selects the three columns from the `sub` table.

You can also use subselects to select all columns where the id matches some criteria:

```
SELECT * FROM tableName WHERE id IN
(SELECT DISTINCT idInTableName FROM anotherTable WHERE someCriteria);
```