SQL = Structured Query Language

==========

Connecting and Disconnecting from THE SERVER

==========

shell> **mysql -h *host* -u *user* -p //host can be omitted when on localhost !!!**
Enter password: ********

mysql> = mysql is ready to receive statements.
-> = prompt for multi-line statements
-> **\c** = ESCAPE CHARACTER
'> "> `> /*> = waiting for completion of a string with ' " or and indetifier with ` or a comment that began with /*

ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)
== the SERVER daemon/windows service is not running.

Shell> mysql //anonymous login
Then: mysql> QUIT

A QUERY consists of an SQL statement followed by a SEMICOLON **;**
In some cases, like QUIT, the semicolon can be omitted.

KEYWORDS may be entered in any lettercase.

mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date; //all the same!
mysql> SeLeCt vErSiOn(), current_DATE

==========

2. Creating and Using a Database

==========

Create: mysql> CREATE DATABASE menagerie;
Using: mysql> USE menagerie **OR**
shell> mysql -h host -u user -p menagerie

Mysql>**SHOW TABLES;** //shows the tables in the DB
Empty set (0.00 sec)

Create Table:

Mysql> **CREATE TABLE**
(
pet (name **VARCHAR**(20) *constraint_name*,
owner VARCHAR(20),
species VARCHAR(20),
sec **CHAR**(1),
birth **DATE**,
death **DATE**
);

VARCHAR(1 - 65536)

Constraints:
rules for the data in the table, specified when the table is **created** or with **ALTER TABLE statement**.
NOT NULL - cannot store NULL value
UNIQUE - each column must have a unique value (ID has to be unique, or SN)
PRIMARY KEY - NOT NULL + UNIQUE - a column (or more) have a unique identity, making it quick and easy to find **(ID)**
FOREIGN KEY - reference to values in another table (**users**.ID and **sales**.ID)
CHECK - ensures the value meets a specific condition
DEFAULT - default value for a column

Writing a Basic SQL Statement

==========

First_name OR [First Name] - with a space, or in use of a keyword like [user]!

USE MySampleDB;
SELECT product_description FROM product; // select column

SELECT …
FROM …
WHERE condition;

==========

Restricting and Sorting Data

==========

LIMIT 3; // first 3 results

LIMIT 10, 15; // results 10 to 15

SELECT **DISTINCT** …. // no duplicates

ORDER BY column_name // order by another column (0-9, a-z,…)

ORDER BY column1, column2 // order first by column1, and then column2

ORDER BY … ASC (default)/DESC //ascending or descending order

WHERE name = 'iPhone 6S+';

Comparison Operators: =,!=,<>,>,<,<=,>=, BETWEEN x AND y (0-9, a-z)

NULL Values:

WHERE name **IS** NULL; // not with **=**

Advanced:

AND, OR, NOT, IN

AND > OR // precedence, no matter the order of writing

WHERE price **IN (49, 100, 999)**; // range of criteria

Wildcards:

_ - any ONE character
% - any number of characters

… WHERE name LIKE 'b%';
'%fy'; // ending with -fy
'%w%'; //w in the middle
'_____' // any five characters!!!
'se_en' // seven, se7en…

… WHERE year BETWEEN 1990 AND 2000;
name BETWEEN 'B' AND 'M'; // names starting with B through M

Regular Expressions(Advanced Searching):
- More flexible!
- **Character Matching:**

SELECT … WHERE prod_name REGEXP 'Gr**.**y Computer Case';
. - single character wildcard
WHERE prod_name REGEXP 'Gr[**ae**]y Computer Case';
[xy] - **group** of characters
WHERE prod_name REGEXP 'Model [**1-6**]543';
[a-z] - **range** of characters
WHERE prod_name REGEXP 'Model \\[7543\\]';
\\ - **escape** characters
\\n - new line; \\f - form feed; \\t - tab
\\r - carriage return; \\v - vertical tab
WHERE prod_name REGEXP 'One[[**:digit:**]]One';
[[:digit:]] - class digit; **alpha** - any letter (upper, lower); **blank** - space/tab
graph - any char without space; **lower/upper; punct; space; xdigit** - hex

FOREIGN KEY - reference to values in another table (**users**.ID and **sales**.ID)
CHECK - ensures the value meets a specific condition
DEFAULT - default value for a column

ALTER TABLE Persons // adds a UNIQUE constraint to the **id** column
ADD UNIQUE (id);
OR
DROP CONSTRAINT (id); // ???
OR
ADD CONSTRAINT constraint_name UNIQUE (column_1, column_2) // on multiple columns

Mysql> **DESCRIBE** pet; //in case we forgot the names of our columns.(visualises table)

Populating a table:
You can save a .txt file with ONE record per line, with values separated by TABS and in the given order. For unknown values we can use NULL as \N
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;

Mysql> **INSERT INTO** pets
    -> **VALUES** ('Puffball', 'Diane', 'hamster', 'f', '1999-03-30', NULL);

```
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

Selecting Particular Rows:
To verify the change to bowser's record:
Mysql> SELECT * FROM pet WHERE name = 'Bowser';

String comparisons are case **IN**sensitive !

Mysql> SELECT * FROM pet WHERE birth **>=** '1998-1-1';
//born after 1998
Mysql> SELECT * FROM pet WHERE species = 'dog' **AND** sex = 'f'; // female dogs

species = 'snake' **OR** species = 'bird';
//snake or bird

(species = 'cat' AND sex ='m') OR
(species='dog' AND sex = 'f'); // AND > OR

Selecting Particular Columns:
If you don't want to see entire rows from your table, just name the columns.

Mysql> **SELECT name, birth FROM pet**; mysql> **SELECT owner FROM pet**;

```
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
```

\\r - carriage return; \\v - vertical tab
WHERE prod_name REGEXP 'One[[**:digit:**]]One';
[[:digit:]] - class digit; **alpha** - any letter (upper, lower); **blank** - space/tab **graph** - any char without space; **lower/upper; punct; space; xdigit** - hex

WHERE prod_name REGEXP '[[:digit:]]**{3}**';
WHERE prod_name REGEXP 'Drives**?**'; // **s** - optional (**useful for plurals**)
***** - any number of matches; **+** - one or more; **{n}** - n matches; **{n,}** - NOT less than n matches; **{n1, n2}** - between n1 and n2 matches; **?** - optional single char. Match.

WHERE prod_name REGEXP '**^**[[:digit:]]';
WHERE prod_name REGEXP 'Phone**$**';
^ - start of text; **$** - end of text; **[[:<:]]** - start of word; **[[:>:]]** - end of word

==========

Single Row Functions

==========

Single row functions work on a single row and return one output per row.

e.g. length and case conversion.

They can be character specific, numeric, date, and conversion functions.

General: (NULL Handling) - NVL, NVL2, NULLIF, COALESCE, CASE, DECODE

Case Conversion: UPPER, LOWER, INITCAP(First_big)

Character: CHAR in CHAR out: CONCAT, LENGTH, REPLACE, SUBSTR,

TRIM,

INSTR - return numeric position of char in string

LPAD/RPAD - pad (FILL UP) the given string up to a specific length with given character (auto same MAX width ?)

REPLACE - replace character from string with a given character

Numeric: NUM in NUM out -

MOD - remainder of the division

ROUND, TRUNC - round and truncate the number

Date functions: MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC

==========

Aggregating Data using Group Functions

==========

Group(aggregate) functions operate on sets of values and are normally used with a **GROUP BY** clause.

What is the average salary of employees in **each department**?
How many employees work in **each department**?
How many employees are working on a **particular project**?

Can be used in both SELECT and HAVING clauses.

AVG(), COUNT(*), MAX(), MIN(), SUM()

AV/COUNT/SUM([ALL | DISTINCT] expression) // ALL - default

SELECT … FROM….

GROUP BY…

HAVING price > 200;

WHERE - before grouping

```
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

mysql>**SELECT DISTINCT owner FROM pet;** //only UNIQUE entries = Benny, Diane, Gwen, Harold

Combine Row and Column Selection:
Mysql> SELECT name, species, birth FROM pet
WHERE species = 'dog' OR species = 'cat';

Sorting Rows:
Mysql> **SELECT name, birth FROM pet ORDER BY birth** (DESC); // default ASCENDING, DESC is optional
Mysql> **SELECT name, species, birth FROM pet**
ORDER BY species, birth DESC; // first order by species ASC, then date within species
GROUP BY price; // groups by one of the columns.

DESC applies only to the keyword immediately preceding it!

Date Calculations:
Mysql> SELECT name, birth, CURDATE(),
TIMESTAMPDIFF(YEAR, birth, CURDATE()) AS age //difference in YEARS btw. Birth and now
FROM pet WHERE death IS NOT NULL; // only for the LIVING

NULL is a special value! So no comparison operators!

MONTH(birth) = month of birth e.g. 2 = February
DAYOFMONTH(birth) = day e.g. 3th of February

SELECT name, birth FROM pet WHERE MONTH(birth) = 5; // born in month

MONTH(DATE_ADD(CURDATE(), INTERVAL 1 MONTH)); //next month !

Working with NULL Values:

NULL = missing unknown value
Test/Comparison:
1 **IS NULL** = 0
1 IS **NOT** NULL = 1

ORDER BY … ASC > NULL First
DESC > NULL Last
LIMIT 3; - show only the first 3 entries
LIMIT 10, 15; - show from 10 to 15

Pattern Matching:

... WHERE name LIKE 'b%';
'%fy'; // ending with -fy
'%w%'; //w in the middle
'_____' // any five characters!!!
'se_en' // seven, se7en…

… WHERE year BETWEEN 1990 AND 2000;
name BETWEEN 'B' AND 'M'; // names starting with B through M

WHERE - before grouping

HAVING - after grouping

==========

Writing Subqueries

==========

```
SELECT * FROM items
WHERE cost >
(
  SELECT AVG(cost) FROM items
)
ORDER BY cost DESC;

SELECT name, MIN(cost) FROM items
WHERE name LIKE '%frogs%'
AND seller_id IN                    // IN(list)
(
  SELECT seller_id FROM items
    WHERE name LIKE '%frogs%'
);
```

==========

Manipulating Data - Data Manipulation Language (DML) Commands

==========

https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_1001.htm#i2099257

==========

Creating Tables

==========

Mysql> **CREATE TABLE**
(
pet (name **VARCHAR**(20) *constraint_name*,
owner VARCHAR(20),
species VARCHAR(20),
sec **CHAR**(1),
birth **DATE**,
death **DATE**
);

VARCHAR(1 - 65536)

==========

Including Constraints

==========

Rules for the data in the table, specified when the table is **created** or with ALTER TABLE statement.
NOT NULL - cannot store NULL value
UNIQUE - each column must have a unique value (ID has to be unique, or SN)
PRIMARY KEY - NOT NULL + UNIQUE - a column (or more) have a unique identity, making it quick and easy to find **(ID)**
FOREIGN KEY - reference to values in another table (**users**.ID and **sales**.ID)
CHECK - ensures the value meets a specific condition
DEFAULT - default value for a column

ALTER TABLE Persons // adds a UNIQUE constraint to the **id** column
ADD UNIQUE (id);
OR
DROP CONSTRAINT (id); // ???
OR

... WHERE year BETWEEN 1990 AND 2000;
name BETWEEN 'B' AND 'M'; // names starting with B through M

_ - any ONE character
% - any number of characters

Numerical Functions:

SELECT SUM(price) ... // select the total sum of a numeric column e.g. sum of prices
MAX(price) ... // select the Max or Min price of the column
MIN(price) ...
AVG(price) ... // calculate the Average price of the column
ROUND

Extended Regular Expressions:

REGEXP and NOT REGEXP
'.' - any single character
[abc] - matches **a or b or c**.
[a - z] - range of characters, **[0 - 9]** - range of numbers -- matches any character/any number
***** - matches zero or more instances of the thing **PRECEDING**. **'x*'** - matches any 'x' characters, **[0-9]*** - matches any number of digits, **'.*'** - matches any number of anything.

REGEXP succeeds if the pattern matches **ANYWHERE** in the tested value, unlike **LIKE**, which succeeds if it matches the **ENTIRE** value.

^b - at the beginning
Fy$ - at the end

SELECT * FROM pet WHERE name REGEXP **'^b'**;
REGEXP **BINARY** '^b'; // **CASE SENSITIVE!!**
REGEXP **'w'**; // containing 'w'
REGEXP **'^.....$'** // names containing exactly **FIVE** chars.
REGEXP **'^.{5}$'** // **{n}** repeat-n-times

Counting Rows:
COUNT(column_name) - how many rows are there in the column
SELECT owner, COUNT(*) FROM pet GROUP BY owner; // how many pets each of them has.
COUNT(*) AS count FROM ... WHERE name LIKE 'G%'; counts the TOTAL number of name that start with a G

Using More Than One Table:

Table_name.column_name = FULLY QUALIFIED NAMES
Now we have also a 'event' table with the events occurred with our animals (name, date, type, remark)

Ages of the pet when it gave birth.
SELECT pet.name,
(YEAR(date)-YEAR(birth)) - (RIGHT(date, 5) <RIGHT(birth,5)) AS age, // **RIGHT** = substring on the right, len=5
remark
FROM pet INNER JOIN event // INNER JOIN = **ONLY** when conditions meets on the **ON** clause
ON pet.name = event.name
WHERE event.type = 'litter';

name | age | remark |
+--------+------+----------------------------+
| Fluffy | 2 | 4 kittens, 3 female, 1 male |
| Buffy | 4 | 5 puppies, 2 female, 3 male |

OR
DROP CONSTRAINT (id); // ???
OR
ADD CONSTRAINT constraint_name UNIQUE (column_1, column_2) // on multiple columns

==========

Creating Views

==========

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

It has rows and columns, like a table. The fields in a view are from one or more real tables in the DB.

A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

(Like a shortcut)

Most commonly used with JOINS.

CREATE VIEW mostbids AS

SELECT id, name, bids FROM items ORDER BY bids DESC LIMIT 10;

SELECT * FROM mostbids;

Updating/Dropping a View:

CREATE OR REPLACE VIEW name AS

SELECT old_query, NEW_QUERY

FROM table_name

WHERE condition;

DROP VIEW view_name

Joining Tables:

==========

Other Database Objects

==========

==========

Controlling User Access

==========

MySQL limits BOTH USERS and WHAT they can do.

SELECT user FROM user; // users and their privileges

Creating Users:

CREATE 'username'@'localhost' IDENTIFIED BY 'password';

- can only connect FROM localhost!! > **'username' @ '%' = anywhere!**

- password will be encrypted!

Verify it:

SELECT host, user, password FROM user WHERE user = 'username';

MISC:

DROP USER 'username' @ 'localhost';

RENAME USER ...@... TO ... @ ...;

SET **PASSWORD** FOR ...@... = Password('new_password');

User Privileges:

A newly created user can log into the MySQL server, but has no privileges to

name | age | remark |
+--------+------+----------------------------+
Fluffy	2	4 kittens, 3 female, 1 male
Buffy	4	5 puppies, 2 female, 3 male
Buffy	5	3 puppies, 3 female

==========

3. Getting Information About Databases and Tables

==========

Forgot the name of your database or table ?

>SHOW DATABASES; // DBs managed by the server
>SELECT DATABASE(); // DB currently in use
>SHOW TABLES; // default DB's tables
>DESCRIBE table_name; // prints the structure of the table

Field	Type	Null	Key	Default	Extra
name	varchar(20)	YES		NULL	
owner	varchar(20)	YES		NULL	
species	varchar(20)	YES		NULL	
sex	char(1)	YES		NULL	
birth	date	YES		NULL	
death	date	YES		NULL	

KEY = indexed ?
Extra = auto_increment ?

SHOW CREATE TABLE = show needed
statement for the CREATE TABLE

SET **PASSWORD** FOR …@... = Password('new_password');

User Privileges:

A newly created user can log into the MySQL server, but has no privileges to do anything.

After creating a user is to GRANT privileges.

SHOW GRANTS FOR …@...;

GRANT USAGE ON *.* TO …@... IDENTIFIED BY …

USAGE ON *.* = no privileges!

GRANT SELECT, INSERT on MySampleDB.* TO …@...;

User can perform **SELECT** and **INSERT** statements on **ANY** tables in the MySampleDB.

==========