DESCRIBE and SHOW COLUMNS can print out columns of a table

DESC <tablename>;   Describe <tablename>;

SHOW COLUMNS FROM <tablename>;

DROP TABLE <name>;

help;

show databases;

SELECT database()

USE <database name>; - enters a database

    cannot drop a database in use. must switch to a new db first, like Master db

select @@hostname;

mysql-ctl start;

Creating Databases Code

Start the CLI:

mysql-ctl cli;

List available databases:

show databases;

The general command for creating a database:

CREATE DATABASE database_name;

DROP DATABASE <database>

ALTER and EXEC (execute)

Syntax to delete/drop column

    ALTER TABLE <<tablename>> DROP COLUMN <columnname>;

    ATLER TABLE Customers DROP COLUMN Agel;

ALTER DATABASE <database> Modify Name MyLearningDB

ALTER DATABASE <database> SET Single_User WITH ROLLBACK IMMEDIATE

Syntax to rename column name:

```
   EXEC sp_rename '<<tablename>>.<<columnname>>' ,                        <<newcolumnname>>.'COLUMN';
   EXEC sp_rename 'Customers.Country' , CountryName, 'COLUMN';
```

Syntax to rename table name:

```
   EXEC sp_rename <<tablename>> , <<New_table_name>>;
   EXEC sp_rename Customers, CustomersInfo;
```

Synatx to alter table. Change a column's type. i.e. age int

```
   ALTER TABLE Customers ALTER COLUMN Age int
```

TABLES

```
CREATE TABLE tablename
  (
   column_name data_type,
   column_name data_type
  );
```

DATATYPES

VARCHAR - only english characters

NVARCHAR - any type of character can be excepted

TIME

TIMESTAMP

BIG INT

INT

SMALLINT

FLOAT

DECIMAL

CHAR

INSERT, UPDATE, DELETE

INSET into UserInfo(UserName) values('User 1')

```
   INSERT into Customers(CustomerName.Country.City.CustomerID) Values ('...');
UPDATE dbo.Products        // updates already inserted data
   SET ProductName = 'Flat Head Screwdriver'
   WHERE ProductID = 50;
UPDATE cats SET breed='Shorthair' WHERE breed='Tabby';
UPDATE cats SET age=14 WHERE name='Misty';
```

DELETE

```
DELETE FROM cats WHERE name='Egg';
DELETE FROM cats WHERE age=4;
DELETE FROM cats WHERE cat_id=age
```

DROP, INSERT, SHOW TABLES

Dropping Tables

```
DROP TABLE <tablename>;
SELECT * FROM <tablename>;
INSERT INTO table_name(column_name) VALUES (data);
SHOW TABLES;
INSERT INTO table_name
            (column_name, column_name)
VALUES      (value, value),
            (value, value),
            (value, value);
```

If you're wondering how to insert a string (VARCHAR) value that contains quotations, then here's how.

You can do it a couple of ways:

Escape the quotes with a backslash: "This text has \"quotes\" in it" or 'This text has \'quotes\' in it'

Alternate single and double quotes: "This text has 'quotes' in it" or 'This text has "quotes" in it'

```
SHOW WARNINGS;
```

CHAR_LENGTH

CHAR_LENGTH returns sizeof varchar array

NULL & NOT NULL

NULL field - does not matter if it has a value

- 'NULL' may be passed as a value for a field

NOT NULL - name VARCHAR(100) NOT NULL

- name cannot be empty when passing a new entry to a table

IFNULL(title, 'missing')

```
SELECT
   first_name,
   IFNULL(title, 'missing')
   IFNULL(grade, 0)
```

Primary Key

Primay Key -- a unique identifier column because there could be people with the same name in a database

PRIMARY KEY(table variable) -- prevents duplicate entry

```
CREATE TABLE unique_cats
  (
   cat_id INT NOT NULL,
   name VARCHAR(100),
   age INT,
   PRIMARY KEY (cat_id)
  );
```

AUTO_INCREMENT --

```
   CREATE TABLE unique_cats2(cat_id NOT NULL AUTO_INCREMENT, name, age)
```

- now cat_id increments and does not need to be declared when making entries

DEFAULT - setting for a field if nothing provided

```
   CREATE TABLE cats3(name VARCHAR(20) DEFAULT 'no name provided')
```

- this will put 'no name provided' in when a name isn't passed to the table

CRUD

CREATE

- INSERT INTO

READ

- SELECT
- * - means select all columns from the table
- SELECT age FROM <tablename> - returns only a column of ages
- SELECT age, name FROM <tablename> - returns column for age and name
- WHERE
- SELECT * FROM <tablename> WHERE age = 4;
- returns columns for entries with an age of 4
- AS
- SELECT cat_id AS id FROM <tablename> - returns the cat_id column now named as id

UPDATE

- UPDATE cats SET breed='SHORTHAIR' WHERE breed='Tabby';

DELETE

Running SQL files

SOURCE <filename.sql> - runs an sql script

SOURCE <foldername/filename.sql> - runs an sql script from the folder <foldername>

ls - shows files in SQL instance

CONCAT

CONCAT(x,y,z) - concatenates columns of data together

- SELECT CONCAT(author_firstname, ' ', author_lastname) as fullname FROM books;
- CONCAT_WS('-', author_firstname, author_lastname) - returns dashes in-between column names being concatenated

SUBSTRING- SUBSTR()

SELECT SUBSTRING('Hello World', 1, 4); - returns indexed values from 1 to 4 i.e. returns 'hell'

- SUBSTRING('hello world', 7); - returns indexed value from index 7 to the end i.e. returns 'world'
- SUBSTRING('hello world', -3); - returns indexed value starting at the end moving left i.e. returns 'rld'

REPLACE

SELECT REPLACE('Hello World', 'Hell', '%$#@'); - returns '%$#@o World'

SELECT REPLACE(<string>, <designate_part_of_string>, <replace_with_this_string>);

SELECT REPLACE(<column_name>, <string_to_replace>, <replace_with_this_string>) FROM <tablename>;

REVERSE

REVERSE - reverses a string index to return a flipped string

CHAR_LENGTH

CHAR_LENGTH - returns strlen

UPPER & LOWER

SELECT UPPER(value); - returns value in all uppercase

SELECT LOWER(value); - returns value in all lowercase

DISTINCT

SELECT DISTINCT - returns unique names in a column. if there's a duplicate, only one item is returned. i.e. a list of 10 books, but 3 are duplicate, then only 8 items are returned

SELECT DISTINCT author_lname FROM books; - returns list of unique author's last names

ORDER BY

SELECT author_lname FROM books

    ORDER BY author_lname; - returns alphabetical list of author's last names

SELECT author_lname FROM books ORDER BY author_lname DESC;

- Here DESC means descending. so Z at the top, A at the bottom. highest number at the top, lowest at the bottom

SELECT title, author_fname, author_lname

- FROM books ORDER BY 2; - returns list ordered by author_fname... the second item SELECTED

LIMIT

SELECT title, released_year FROM books

    ORDER BY released_year DESC LIMIT 5;

- This code limits the results returned to 5; ergo the top 5.
- ORDER BY released_year DESC LIMIT 0,5;
- LIMIT returned results from 0 to 5... descending

OFFSET

- offset moves down x rows and starts selection there
  - i.e. OFFSET 3 - starts on the 4th row

LIKE

WHERE author_fname LIKE '%da%'

```
SELECT title, author_fname FROM books WHERE author_fname LIKE '%da'%;
                               ^      ^
                                  WILDCARDS
```

- this code looks through author_fname column for names with a 'da' string in them
- WHERE title LIKE '%\%%'
- WHERE title LIKE '%_%'
- % = anything, _ escape command
- this is to search for a percent sign

COUNT

SELECT COUNT(*) FROM books; - returns number of entries in the table

GROUP BY & HAVING

GROUP BY aggregates identical data into single rows

HAVING limits a query based on constraints. see example

```
SELECT author_lname, COUNT(*)
   FROM books GROUP BY author_lname;
```

- returns a column with the number of entries per author_lname ... i.e. the # of books written by each one

GROUP BY titles.pub_id

HAVING publishers.state = 'CA';   // use having in conjunction with GROUP BY

MIN & MAX

SELECT MIN(released_year) FROM books; - returns book from earliest release year

- SELECT title, pages FROM books

WHERE pages = (SELECT Max(pages)

FROM books);

- chooses the book with the most pages
- SELECT author_fname, author_lname, Min(released_year)
- FROM books
- GROUP BY author_lname, author_fname;
- This code returns a table with columns: author_lname, author_fname, Minimum released year.
- entries returned are the minimum release year for each book entry

AVG

SELECT AVG(released_year) FROM books; - returns the average release year of books

DECIMAL, FLOAT, & DOUBLE

CREATE TABLE thingies (price FLOAT);

- price is type FLOAT

CREATE TABLE thingies (price DECMIAL(5,2));

- price is 5 digits long and 2 are after the decimal... i.e. price = 999.99.
- in addition, if there are 5 digits and 2 are after the decimal, then price = 999.99 is the max number that can be held in the table

DATE, TIME, and DATETIME

CREATE TABLE people (name VARCHAR(100), birthdate DATE, birthtime TIME, birthdt DATETIME);

INSERT INTO people (name, birthdate, birthtime, birthdt)

VALUES('Padma', '1983-11-11', '10:07:35', '1983-11-11 10:07:35');

INSERT INTO people (name, birthdate, birthtime, birthdt)

VALUES('Larry', '1943-12-25', '04:10:42', '1943-12-25 04:10:42');

SELECT * FROM people;

FORMATING DATES - p. 427

SELECT name, DAY(birthdate) FROM people;

DAY   DAYNAME   DAYOFWEEK   DAYOFYEAR

MONTH   MONTHNAME

YEAR

HOUR MINUTE

DATE MATH - p.

DATEDIFF(expr1, expr2) - date difference in days

DATE_ADD - adds an interval to the current date

MOD - wrapper for date for values that exceed index (i.e. december - 12 - 12 + 1 wraps to next   year)

LOGICAL OPERATORS

Not Equal --> !=

LIKE - LIKE 'W'   LIKE 'W%'   LIKE '%W%'   LIKE 'W%'

NOT LIKE

Greater than --> >=, >

Less than --> <=, <

AND --> && or AND

OR --> || or OR

BETWEEN - as it says... value between value

CAST - or CONVERT - converts expression of one data type to another

```
   CAST(_value_ AS _datatype_)
```

IN - instead of saying: WHERE author_lname = 'Carver' OR author_lname = 'Lahiri' OR 'author_lname = 'Smith';

   do this: WHERE author_lname IN('Carver','Lahiri','Smith')

NOT IN - ... opposite of IN

CASE STATEMENTS

CASE:

```
   SELECT title, released_year,
           CASE
               WHEN released_year >= 2000 THEN 'Modern Lit'
               ELSE '20th Century Lit'
           END AS GENRE    // genre column for 'modern lit' and '20th century lit'
       FROM books;
```

JOIN, LEFT JOIN, RIGHT JOIN

JOIN takes the overlap between A and B

LEFT JOIN takes the A and overlap with B

RIGHT JOIN takes B and the overlap with A

CURDATE, CURTIME, NOW

MONEY

money datatype

TRIGGERS

```
DELIMITER $$    // instead of a semicolon, a $$ is now our delimiter
CREATE TRIGGER must_be_adult... <trigger_name>... so we can delete it by name if needed
   BEFORE<trigger_time> INSERT ON<trigger event>  users<tablename> FOR EACH ROW
   BEGIN
      IF NEW.<column_name>  < 18
      THEN
         SIGNAL SQLSTATE '45000'    // wildcard state. this throws a trigger
            SET MESSAGE_TEXT = 'Must be an adult!';    // note semicolon
      END IF;        // note semicolon
   END;    note semicolon
$$    // delimiter allows us to place semicolons needed for compilation
```

```
DELIMITER $$
   CREATE TRIGGER capture_unfollow // throw a trigger when unfollowing a profile
   AFTER DELETE ON follows FOR EACH ROW
   BEGIN
      INSERT INTO unfollows
      SET
         follower_id = OLD.follower_id,
         followee_id = OLD.followee_id;
      END;
```

```
$$


DELIMITER $$
CREATE  TRIGGER trigger_name
   trigger_time trigger_event ON table_name FOR EACH ROW
   BEGIN
   END;
$$
```

DCL - Data Control Language

GRANT - it gives users permission to access database

REVOKE - withdraw the permission given by GRANT to access the database.

DML - data manipulation language

UPDATE	MERGE	SELECT	UPDATETEXT
INSERT	READTEXT	BULK INSERTQ	
DELETE	WRITETEXT		

DDL - Data Definition Language

CREATE - create an object: database, table, trigger, index, view, functions, stored procedures

DROP - delete table or database

ALTER - alter an existing database or its objects' structures

TRUNCATE - removes records from a table

RENAME - rename the database objects

TCL - Transaction Control Lanuage

COMMIT - commit the running transaction

ROLLBACK - rollback current transaction

SAVEPOINT - You can set a save point so that, next time it will start from here

SET TRANSACTION -

SUBQUEIRIES

comparison - An expression and a comparison operator that compares the expression with the results of the subquery.

expression - An expression for which the result set of the subquery is searched

sqlstatement -   a SELECT statement

uniqueidentifier

NEWID

NEWSEQUENTIALID

ANY & ALL

VIEWS

NEWID

NEWSEQUENTIALID

EXAMPLES

1. select all books written by eggers or chabon

```
  SELECT title, author_lname FROM books
     WHERE author_lname = 'Eggers' OR author_lname = 'Chabon';
```

... or

```
     WHERE author_lname IN ('Eggers','Chabon');
```

2. select all books written by lahiri, published after 2000

```
  SELECT title, author_lname, released_year FROM books
     WHERE author_lname = 'Lahiri' &&
     WHERE released_year > 2000;
```

3. select all books with page counts between 100 and 200

```
  SELECT title, pagesFROM books
     WHERE COUNT(pages) BETWEEN 100 AND 200;
```

4. select all books where author_lname starts with a 'C' or an 'S'

```
  SELECT title, author_lname FROM books
     WHERE author_lname LIKE 'C%' OR
     WHERE author_lname LIKE 'S%';
```

... or   WHERE SUBSTR(author_lname, 1, 1) = 'C' OR

        WHERE SUBSTR(author_lname, 1, 1) = 'S';

...or   WHERE SUBSTR(author_lname, 1, 1) = 'C';

5. case statements:

using %:

putting % at the beginning means anything before " " then stories after wards

putting % at the end means " " stories then anything afterwards

%" "% means stories anywhere in the string

```
6. SELECT author_lname,
    CASE
        WHEN COUNT(*) = 1 THEN '1 book'
        ELSE CONCAT(COUNT(*),' books'
    END AS COUNT
    FROM books
    GROUP BY author_lname, author_fname;
7. CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY
    first_name VARCHAR(100)
    last_name VARCHAR(100)
    email VARCHAR(100)
CREATE TABLE orders(
    id INT AUTO_INCREMENT PRIMARY KEY
    order_date DATE
    amount DECIMAL(8,2)    // max of 8 significant figures. 2 figures after decimal
    customer_id INT,
    FOREIGN KEY(customer_id) REFERENCES customers(id)
    )
```

8. implicit join

```
SELECT * FROM customers, orders
   WHERE customers.id = orders.customer_id;
```

9. explicit inner join

```
   SELECT * FROM customers
   JOIN orders
      ON customers.id = orders.customer_id; // where primary key = foreign key
```

10.
```
SELECT
   first_name,
   IFNULL(AVG(grade),0) AS grade
   CASE
      WHEN grade >= 70 THEN 'PASSING'
      WHEN grade IS NULL THEN 'FAILING'
      ELSE 'FAILING'
   END AS 'STATUS'
FROM students
LEFT JOIN papers
   ON students.id = papers.student_id
GROUP BY students.id
ORDER BY grade DESC;
```

11.
```
CREATE TABLE users(
   username VARCHAR(100),
   age INT
);
```

12.
```
INSERT INTO users(username, age) VALUES("bobby", 23)
```

USER DEFINED FUNCTIONS

-multi-statement scalar function (scalar UDF)

The function takes one input value, a ProductID, and returns a single data value, the aggregated quantity of the specified product in inventory.

```
IF OBJECT_ID (N'dbo.ufnGetInventoryStock', N'FN') IS NOT NULL

   DROP FUNCTION ufnGetInventoryStock;

GO

CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)

RETURNS int

AS

-- Returns the stock level for the product.

BEGIN

   DECLARE @ret int;

   SELECT @ret = SUM(p.Quantity)

   FROM Production.ProductInventory p

   WHERE p.ProductID = @ProductID

      AND p.LocationID = '6';

    IF (@ret IS NULL)

      SET @ret = 0;

   RETURN @ret;

END;
```

IF STATEMENT

```
IF NOT EXISTS(SLECT CustomerID FROM Customers WHERE (CustomerName = 'Rock')

BEGIN

INSERT INTO Customers(CustomerName, Country, City, Age) VALUES('Rock', 'India', 'Bangalore', 25))

END

ELSE

BEGIN

SELECT 'EmailExists' AS ReturnMessage

END
```

SCHEMABINDING