NOTES

MySQL Tutorial for Beginners

By: Programming with Josh

Database is a collection of data stored in a format that can easily be accessed.

In order to manage databases, we use a software application called **Database Management System (DBMS).**

2 Categories of DBMS:

- Relational
  Stores data that are link to each other using relationship.
  Structured Query Language (SQL) is the language that is used

  Relational Database Management Systems (RDBMS)

  MySQL

  SQL Server

  Oracle

- NoSQL
  No tables or relationship.
  Don't understand SQL.

SEQUEL Structured English Query Language was originally developed by IBM in 70's and back. But they change to SQL Structured Query Language.

Installing MySQL on computer.

https://dev.mysql.com/downloads/

Overview of workbench interface

On top left: Tool Bar – creating new tab for writing SQL code.

Opening a file, creating database, table and so on.

Left side: Navigator panel with two tabs;

Administration- starting or stopping server, importing and exporting and so on. And

Schemas- shows the databases in the current server.

In the middle: query editor window

Right side: context help and snippets

Top right side: showing or hiding these panel button.

Creating Databases

(download the zip file attached below the video**)**

USE sql_store;

SELECT * FROM sql_store.customers;

--we can see all the data in this table this is comment

SELECT * FROM sql_store.orders;

--order table

SELECT Statement

USE sql_store;

SELECT *

FROM customers

--select all the customers given in the table

--using two clause

WHERE customer_id = 1

--this is the where clause

--only get the customer id 1

ORDER BY first_name

–specify the columns that were going to sort

SELECT CLAUSE

SELECT last_name , first_name

FROM customers

--select only the lastname and firstname

(another example)

SELECT state

FROM customer

— Removing duplicates

SELECT DISTINCT state

FROM customers

WHERE Clause

We use the WHERE clause to filter data.

Comparison operators:

- Greater than: >
- Greater than or equal to: >=
- Less than: <
- Less than or equal to: <=
- Equal: =
- Not equal: <>
- Not equal: !=

SELECT *

FROM customers

WHERE state = ‘VA’

Logical Operators

—- AND (both conditions must be True)

SELECT *

FROM customers

WHERE birthdate > ‘1990-01-01’ AND points > 1000

—- OR (at least one condition must be True)

SELECT *

FROM customers

WHERE birthdate > ‘1990-01-01’ OR points > 1000

—- NOT (to negate a condition)

SELECT *

FROM customers

WHERE NOT (birthdate > ‘1990-01-01’)

Exercise

--From the order_items table, get items

-- for order #6

-- where the total price is greater than 30

Solution

SELECT *

FROM order_items

WHERE order_id = 6 AND unit_price * quantity > 30

IN Operator

—- Returns customers in any of these states: VA, NY, CA

--in a shorter way

SELECT *

FROM customers

WHERE state IN (‘VA’, ‘NY’, ‘CA’)

SELECT *

FROM customers

WHERE state NOT IN (‘VA’, ‘NY’, ‘CA’)

Exercise

--Return products with

-- quantity in stock equal to 49, 38, 72

--

Solution

SELECT *

FROM products

WHERE quantity in stock IN (49, 38, 72)

BETWEEN Operator

--shorter and cleaner

```
SELECT *
 FROM customers
WHERE points BETWEEN 1000 AND 3000
```

Exercise

--Return customers born

**--	between 1/1/1990 and 1/1/2000**

Solution

```
SELECT *
 FROM customers
WHERE birth_date BETWEEN ‘1990-0-01’ AND  ‘2000-01-01’
```

LIKE Operator

—- Returns customers whose first name starts with b

```
SELECT *
FROM customers
WHERE first_name LIKE ‘b%’
```

```
SELECT *
FROM customers
WHERE first_name LIKE ‘%b%’
```

```
SELECT *
FROM customers
WHERE first_name LIKE ‘%b’
```

SELECT *

FROM customers

WHERE first_name LIKE ‘______y’

- **% any number of characters**
- **_ exactly one character**

Exercises

--Get the customers whose

-- addresses contain TRAIL or AVENUE

-- phone numbers end with 9

Solution

```
SELECT *
FROM customers
WHERE address LIKE ‘%trail%’ OR
        address LIKE ‘%avenue%’
```

```
SELECT *
FROM customers
WHERE phone LIKE ‘%9’
```

REGEXP Operator

--regular expression

—- Returns customers whose first name starts with a

```
SELECT *
FROM customers
WHERE first_name REGEXP ‘^a’
```

```
SELECT *

FROM customers

WHERE first_name REGEXP ‘a$’
```

```
SELECT *

FROM customers

WHERE first_name REGEXP ‘field|mac’
```

```
SELECT *

FROM customers

WHERE first_name REGEXP ‘[abc]a’
```

```
SELECT *

FROM customers

WHERE first_name REGEXP ‘[a-g]a’
```

• ^: beginning of a string

• $: end of a string

• |: logical OR

• [abc]: match any single characters

• [a-d]: any characters from a to d

Exercises

--Get the customers whose

-- first names are ELKA or AMBUR

Solution

SELECT *

FROM customers

WHERE first_names REGEXP ‘elka|ambur’

-- last names with EY or ON

SELECT *

FROM customers

WHERE last_names REGEXP ‘ey$|on$’

-- last names starts with MY or contains with SE

SELECT *

FROM customers

WHERE last_names REGEXP ‘^my|se’

-- last names contains B followed by R or U

SELECT *

FROM customers

WHERE last_names REGEXP ‘b[ru]’

IS NULL Operator

—- Returns customers who don’t have a phone number

SELECT *

FROM customers

WHERE phone IS NULL

SELECT *

FROM customers

WHERE phone IS NOT NULL

Exercise

--Get the orders that are not shipped

Solution

SELECT *

FROM orders

WHERE shipped_date IS NULL

--or shipper_id

ORDER BY Clause

—- Sort customers by state (in ascending order), and then

—- by their first name (in descending order)

SELECT *

FROM customers

ORDER BY state, first_name DESC

SELECT fisrt_name, last_name

FROM customers

ORDER BY state DESC, first_name DESC

LIMIT Clause

—- Return only 3 customers

SELECT *

FROM customers

LIMIT 3

—- Skip 6 customers and return 3

SELECT *

FROM customers

LIMIT 6, 3

Exercise

--Get the top three loyal customers

Solution

SELECT *

FROM customers

ORDER BY points DESC

LIMIT 3

--limit clause should always come at the end

Inner Joins

SELECT *

FROM orders

JOIN customers

ON orders.customer_id = customers.customer_id

```
SELECT order_id, first_name, last_name
FROM orders
JOIN customers
        ON orders.customer_id = customers.customer_id
SELECT order_id, first_name, last_name
FROM orders o
JOIN customers c
        ON o.customer_id = c.customer_id
```

JOINING ACROSS DATABASE

```
SELECT *
FROM orders_items oi
JOIN sql_inventory.products p
         ON oi.product_id = p.product_id


USE sql_inventory;
SELECT *
FROM sql_store.orders_items oi
JOIN sql_inventory.products p
         ON oi.product_id = p.product_id
```

SELF JOINS

```
USE sql_hr;

SELECT *

FROM employees e

JOIN employees m

        ON e.reports_to = m.employee_id


USE sql_hr;

SELECT

        e.employee_id,

        e.first_name,

        m.first_name AS manager

FROM employees e

JOIN employees m

        ON e.reports_to = m.employee_id
```

JOINING MULTIPLE TABLES

```
USE sql_store;

SELECT *

FROM orders o

JOIN customers c

        ON o.customer_id = c.customer_id

JOIN order_statuses os

        ON o.status = os.order_status_id

--the result of this is complicated
```

```
USE sql_store;

SELECT

        o.order_id,

        o.order_date,

        c.first_name

        c.last_name

        os.name AS status

FROM orders o

JOIN customers c

         ON o.customer_id = c.customer_id

JOIN order_statuses os

        ON o.status = os.order_status_id
```

COMPOUND JOIN CONDITIONS

```
SELECT *

FROM order_item oi

JOIN order_item_notes oin

        ON oi.order_id = oin.order_id

        AND oi.product_id = oin.product_id
```

IMPLICIT JOIN SYNTAX

```
SELECT *

FROM orders o, customers c

WHERE o.customer_id = c.customer_id
```

Outer Joins

—- Return all customers whether they have any orders or not

```
SELECT
	c.customer_id,
	c.first_name,
	o.oerder_id
FROM customers c
LEFT JOIN orders o
	ON c.customer_id = o.customer_id
ORDER BY c.customer_id
SELECT
	c.customer_id,
	c.first_name,
	o.oerder_id
FROM customers c
RIGHT JOIN orders o
	ON c.customer_id = o.customer_id
ORDER BY c.customer_id
SELECT
	c.customer_id,
	c.first_name,
	o.oerder_id
FROM customers c
RIGHT JOIN orders o
	ON c.customer_id = o.customer_id
ORDER BY c.customer_id
```

OUTER JOIN BETWEEN MULTIPLE TABLES

```
SELECT

        c.customer_id,

        c.first_name,

        o.oerder_id

        sh.name AS shipper

FROM customers c

LEFT JOIN orders o

        ON c.customer_id = o.customer_id

LEFT JOIN shippers sh

        ON o.shipper_id = sh.shipper_id

ORDER BY c.customer_id
```

SELF OUTER JOINS

```
USE sql_hr;

SELECT

        e.employee_id,

        e.first_name,

        m.first_name AS manager

FROM employees e

JOIN employees m

        ON e.reports_to = m.employee_id

USE sql_hr;

SELECT

        e.employee_id,

        e.first_name,
        m.first_name AS manager
FROM employees e
LEFT JOIN employees m
        ON e.reports_to = m.employee_id
```

USING Clause

If column names are exactly the same, you can simplify the join with the USING clause.

```
SELECT
        o.order_id,
        c.first_name
        sh.name AS shipper
 FROM orders o
JOIN customers c
        USING (customer_id)
LEFT JOIN shippers sh
        USING (shipper_id)
```

```
SELECT *
FROM order_items oi
JOIN order_item_notes oin
        USING (order_id, product_id)
```

NATURAL JOINS

```
SELECT
        o.order_id,
        c.first_name
FROM orders o
NATURAL JOIN customers c
```

Cross Joins

```
—- Combine every color with every size
SELECT *
FROM colors
 CROSS JOIN sizes


SELECT
        c.first_name AS customer,
        p.name AS product
 FROM customers c
CROSS JOIN products p
ORDER BY c.first_name


SELECT
        c.first_name AS customer,
        p.name AS product
 FROM customers c, orders o
ORDER BY c.first_name
```

Exercises

```
    --Do a cross join between shippers and products
    --      using the implicit syntax
    --      and the using explicit systax
```

Solution

```
SELECT
    sh.name AS shipper,
    p.name AS product
FROM shippers sh, products p
ORDER BY sh.name


SELECT
    sh.name AS shipper,
    p.name AS product
FROM shippers sh
CROSS JOIN products p
ORDER BY sh.name
```

Unions

—- Combine records from multiple result sets

```
SELECT name, address
FROM customers
UNION SELECT name, address
FROM clients
```

```
SELECT
        order_id,
        order_date,
        ‘Active’ AS status
FROM orders
WHERE order_date >= ’2019-01-01’
UNION
SELECT
        order_id,
        order_date,
        ‘Archived’ AS status
FROM orders
WHERE order_date < ’2019-01-01’
```

Exercise

Solution

```
SELECT
        customer_id,
        first_name,
        points,
         ‘Bronze’ AS type
FROM customers
WHERE points < 2000
UNION
SELECT
```

```
        customer_id,
        first_name,
        points,
         ‘Silver’ AS type
FROM customers
WHERE points BETWEEN 2000 AND 3000
UNION
```

Solution

```
SELECT
        customer_id,
        first_name,
        points,
         ‘Gold’ AS type
FROM customers
WHERE points > 3000
ORDER BY first_name
```

INSERTING A SINGLE ROW

```
—- Insert a single record

INSERT INTO customers (first_name, phone, points)

VALUES (‘Mosh’, NULL, DEFAULT)

INSERT INTO customers (

        first-name,

        last_name,

        birth_date,

        address,

        city,

        state)

VALUES (

        ‘John’,

        ‘Smith’,

        ‘1990-01-01’,

        ‘adress’,

        ‘city’,

        ‘CA’,)
```

INSETING MULTIPLE ROWS

```
—- Insert multiple single records

INSERT INTO customers (first_name, phone, points)

VALUES (‘Mosh’, NULL, DEFAULT), (‘Bob’, ‘1234’, 10)
```

INSERT INTO shippers (name)

VALUES (‘Shippers1’)

    (‘Shippers2’)

    (‘Shippers3’)

Exercise

--Insert three rows in the products table

Solution

INSERT INTO products (name, quantity_in_stock, unit_price)

VALUES (‘Product1’, ‘10’, ‘1.95)

(‘Product2’, ‘11’, ‘1.95)

(‘Product3’, ‘12’, ‘1.95)

INSETING HIERARCHICAL ROWS

INSERT INTO orders (customer_id, order_date, status)

VALUES (1, ‘2019-01-02’, 1);

INSERT INTO order_items

VALUES

(LAST_INSERT_ID(), 1, 1, 2.95)

(LAST_INSERT_ID(), 2, 1, 3.95)

CREATING A COPY OF A TABLE

```
CREATE TABLE orders_archived AS

SELECT *

FROM orders


INSERT INTO orders_archived

SELECT *

FROM orders

WHERE order_date < ‘2019-01-01’
```

Exercise-Solution

```
USE sql_invoicing;

SELECT *

FROM invoices i

JOIN clients c

        USING (client_id)


USE sql_invoicing;

CREATE TABLES invoices_archived AS

SELECT

        i.invoice_id,

        i,number,

        c.name AS client,

        i.ivoice_total,

        i.payment_total,

        i.invoice_date,
        i.payment_date,
        i.due_date
FROM invoices i
JOIN clients c
        USING (client_id)
WHERE payment_date IS NOT NULL
```

UPDATING A SINGLE ROW

```
UPDATE invoices
SET payment_total = 10, payment_date ‘2019-03-01’
WHERE invoice_id = 1


UPDATE invoices
SET payment_total = 0, payment_date NULL
WHERE invoice_id = 1


UPDATE invoices
SET payment_total = DEFAULT, payment_date NULL
WHERE invoice_id = 1


UPDATE invoices
SET
        payment_total = invoice_total * 0.5,
        payment_date = due_date
WHERE invoice_id = 3
```

UPDATING MULTIPLE ROWS

```
UPDATE invoices

SET

	payment_total = invoice_total * 0.5,

	payment_date = due_date

WHERE client_id = 3


UPDATE invoices

SET

	payment_total = invoice_total * 0.5,

	payment_date = due_date

WHERE client_id IN (3, 4)
```

Exercise

```
	--Write a SQL statement to

	--	give any customers born before 1990

	--	50 extra points
```

Solution

```
USE sql_store;

UPDATE customers

SET points = points + 50

WHERE birth_date < ‘1990-01-01’
```

USING SUBQUERIES IN UPDATES

Subqueries is a select statement that is within another SQL

```
UPDATE invoices
SET
        payment_total = invoice_total * 0.5,
        payment_date = due_date
WHERE client_id =
                (SELECT client_id
                FROM clients
                WHERE name = ‘Myworks’)


UPDATE invoices
SET
        payment_total = invoice_total * 0.5,
        payment_date = due_date
WHERE client_id = IN
                (SELECT client_id
                FROM clients
                WHERE state IN (‘CA’, ‘NY’))
```

DELETING ROWS

```
DELETE FROM invoices
WHERE invoice_id = (
                SELECT *
                FROM clients
                WHERE name = ‘Myworks’)
```

RESTORING THE DATABASE

In MySQL Workbench, on the top to the file menu and open SQL script. Then navigate to the directory where you stored the SQL scripts, in case you lost that directory, go back to the first section where you have downloaded the supplementary materials. So in this directory open create-databases.sql . Now execute this script to recreate all of our databases. Now open up the navigator pane, you can see the databases disappear from here, simply click on the refresh icon.

Name: Alice M. Villarubia

Year & Section: 3-BSCS-A