MySQL Tutorials for Beginners

What is a database?

* Is a collection of data stored in a format that can easily be accessed

Two categories of DBMS:

* Relational
  + Stored data in tables that are linked to each other using relationships.
* SQL (Structured Query Language) (SQUEL)
  + Is the language that we used to work with these relational database management system
  + Example:
    - SELECT \*

FROM products

WHERE category = ‘food’

ORDER BY price

* + SQUEL (Structured English Query Language)
* No SQL (Non-Relational Databases)
  + Don’t have tables or relationships
  + Don’t understand SQL

**Creating Databases**

SELECT Statement

USE sql\_store;

SELECT \*

FROM customers

--select all the customers given in the table

customer\_id = 1

--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’

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’,)

INSERTING 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)

INSERTING 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’

RESTORING THE DATABASE

In MySQL Workbench, in this directory open create-databases.sql . 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.