AI VIET NAM
@aivietnam.edu.vn

SQL in Data Analysis

(Subqueries, Procedure and Trigger, Python + SQL, and PySpark)

Vinh Dinh Nguyen
PhD in Computer Science

Outline

- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice
- Python and SQL, PySpark for Big Data

SQL Queries

Common Table Expression

AI VIET NAM
@aivietnam.edu.vn

Common Table Expression

Using the sql_invoicing database, how would you find the client with the highest total invoice amount?

client_id	name	address	city	state	phone
1	Vinte	3 Nevada Parkway	Syracuse	NY	315-252-7305
2	Myworks	34267 Glendale Parkway	Huntington	WV	304-659-1170
3	Yadel	096 Pawling Parkway	San Francisco	CA	415-144-6037
4	Kwideo	81674 Westerfield Circle	Waco	TX	254-750-0784
5	Topiclounge	0863 Farmco Road	Portland	OR	971-888-9129
NULL	NULL	NULL	NULL	NULL	NULL

invoice_id	number	client_id	invoice_total	payment_total	invoice_date	due_date	payment_date
1	91-953-3396	2	101.79	0.00	2019-03-09	2019-03-29	NULL
2	03-898-6735	5	175.32	8.18	2019-06-11	2019-07-01	2019-02-12
3	20-228-0335	5	147.99	0.00	2019-07-31	2019-08-20	NULL
4	56-934-0748	3	152.21	0.00	2019-03-08	2019-03-28	NULL
5	87-052-3121	5	169.36	0.00	2019-07-18	2019-08-07	NULL
6	75-587-6626	1	157.78	74.55	2019-01-29	2019-02-18	2019-01-03
7	68-093-9863	3	133.87	0.00	2019-09-04	2019-09-24	NULL
8	78-145-1093	1	189.12	0.00	2019-05-20	2019-06-09	NULL
9	77-593-0081	5	172.17	0.00	2019-07-09	2019-07-29	NULL
10	48-266-1517	1	159.50	0.00	2019-06-30	2019-07-20	NULL
11	20-848-0181	3	126.15	0.03	2019-01-07	2019-01-27	2019-01-11
13	41-666-1035	5	135.01	87.44	2019-06-25	2019-07-15	2019-01-26
15	55-105-9605	3	167.29	80.31	2019-11-25	2019-12-15	2019-01-15
16	10-451-8824	1	162.02	0.00	2019-03-30	2019-04-19	NULL
17	33-615-4694	3	126.38	68.10	2019-07-30	2019-08-19	2019-01-15
18	52-269-9803	5	180.17	42.77	2019-05-23	2019-06-12	2019-01-08
19	83-559-4105	1	134.47	0.00	2019-11-23	2019-12-13	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

We need to sum all the invoice values, group by the client id, then we have to order them by the sum. After that we take the first client id, which is the person have the highest total invoice and query their infomation.

How can we write this query?

Common Table Expression

This is what we want, but how can we get that values?

```
SELECT name
FROM clients
WHERE client_id = <highest_invoice_client>;
```

One solution is we can query from a join of two table.

```
SELECT c.name
FROM clients c JOIN invoices i
USING (client_id)
GROUP BY i.client_id
ORDER BY SUM(i.invoice_total) DESC
LIMIT 1;
```

name	client_id	invoice_amount
Topiclounge	5	980.02
Vinte	1	802.89
Yadel	3	705.90
Myworks	2	101.79

We will get only the first client when we set LIMIT by 1.

Common Table Expression

Another solution is to use CTE, where we first query what we want and using it for our main query

```
WITH invoice_amount AS (
        SELECT client_id, SUM(invoice_total) AS invoice_amount
        FROM invoices
        GROUP BY client_id
        ORDER BY SUM(invoice_total) DESC
        LIMIT 1
)
SELECT c.client_id, c.name, i.invoice_amount
FROM clients c JOIN invoice_amount i
USING (client_id);
```

name	client_id	invoice_amount
Topiclounge	5	980.02
Vinte	1	802.89
Yadel	3	705.90
Myworks	2	101.79

We will get only the first client when we set LIMIT by 1.

Outline

- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice
- Python and SQL, PySpark for Big Data

SQL Queries

Subqueries

Subqueries

When we get comfortable with CTE, and we don't want to join two table. Another solution is putting the query inside the WITH statement directly in ours main query using Subqueries technique.

```
SELECT client_id,
       name,
       (SELECT SUM(invoice_total)
        FROM invoices
        WHERE c.client_id = client_id) AS invoice_amount
FROM clients c
ORDER BY invoice_amount DESC
LIMIT 1;
```

The Subquery will be calculated first where it will calculate the sum of invoice based on the condition given by ours main query.

With this method we can avoid using JOIN and GROUP BY

Subqueries

```
SELECT client_id,
            (SELECT AVG(invoice_total)
             FROM invoices
             WHERE client_id = c.client_id) AS avg_invoice
FROM clients c;
```

client_id	avg_invoice
1	160.578000
2	101.790000
3	141.180000
4	NULL
5	163.336667

But both CTE and Subquery all have the same problem.
It have to re-evaluate all of ours query every time we run it.

What if we need that Subquery multiple times?

Outline

- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice
- Python and SQL, PySpark for Big Data

SQL Queries

Temp Table

Temp Table

Let take this query for examle

```
SELECT client_id,
            SUM(invoice_total) AS invoice_sum,
            AVG(invoice_total) AS invoice_avg
FROM invoices
GROUP BY client_id;
```

client_id	invoice_sum	invoice_avg
1	802.89	160.578000
2	101.79	101.790000
3	705.90	141.180000
5	980.02	163.336667

How can we save this query result to reuse multiple times?

Temp Table

We can create a temporary table to store that data

```
CREATE TEMPORARY TABLE temp_invoice (
        client_id INT,
        invoice_sum DECIMAL(10,2),
        invoice_avg DECIMAL(10,2)
);
```

Let check the table we just created

```
SELECT * FROM temp_invoice;
```

client_id	invoice_sum	invoice_avg

Ours table don’t have any data yet. How can we put the result of ours query into it?

Temp Table

By using Subquery and INSERT

```
INSERT INTO temp_invoice
SELECT client_id,
            SUM(invoice_total) AS invoice_sum,
            AVG(invoice_total) AS invoice_avg
FROM invoices
GROUP BY client_id;
```

Let check ours temporary table again

```
SELECT * FROM temp_invoice;
```

client_id	invoice_sum	invoice_avg
1	802.89	160.58
2	101.79	101.79
3	705.90	141.18
5	980.02	163.34

Now ours table have the infomation we want to stored. We can call this table any time we need to

AI VIET NAM
@aivietnam.edu.vn

Temp Table

Let use this table to solve ours previous two question.

```
SELECT client_id, name,
            (SELECT invoice_sum
             FROM temp_invoice
             WHERE c.client_id = client_id) AS invoice_sum
FROM clients c
ORDER BY invoice_sum DESC
LIMIT 1;
```

client_id	name	invoice_sum
5	Topiclounge	980.02

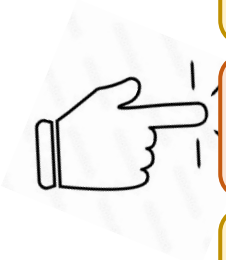
```
SELECT client_id, name,
            (SELECT invoice_avg
             FROM temp_invoice
             WHERE c.client_id = client_id) AS invoice_avg
FROM clients c;
```

client_id	name	invoice_avg
1	Vinte	160.58
2	Myworks	101.79
3	Yadel	141.18
4	Kwideo	NULL
5	Topiclounge	163.34

This is especially useful when our subquery involve heavy calculation. We only need to calculate it once and store it for later used. HOWEVER Temporary table only exists on ours session, when we close ours session, the table will be erase.

What if we don't want to rewrite everything every times?

Outline



- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice
- Python and SQL, PySpark for Big Data

SQL Queries

Stored Procedures

AI VIET NAM
@aivietnam.edu.vn

Stored Procedures

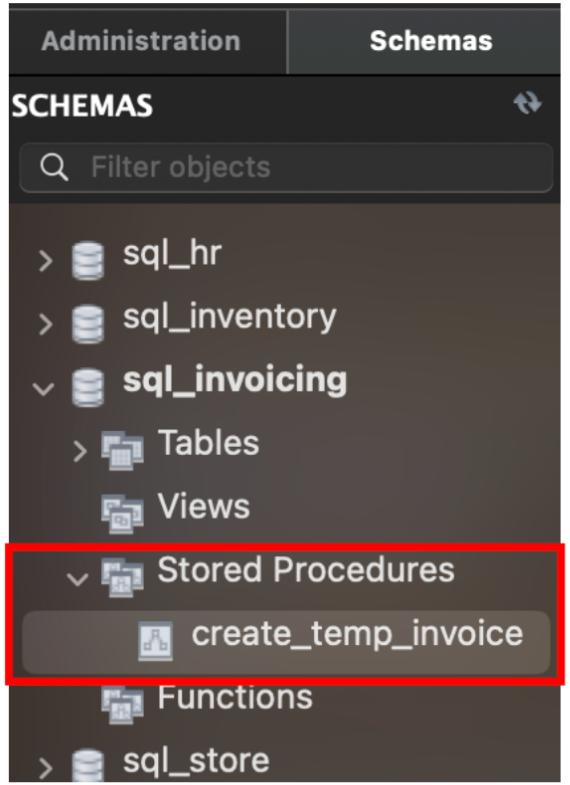
A Stored Procedures is essentially a pre-written function. For example we want a function to create our temporary table and insert the data into it.

```
DELIMITER //
CREATE PROCEDURE create_temp_invoice()
BEGIN
    DROP TABLE IF EXISTS temp_invoice;
    CREATE TEMPORARY TABLE temp_invoice (
        client_id INT,
        invoice_sum DECIMAL(10,2),
        invoice_avg DECIMAL(10,2)
    );

    INSERT INTO temp_invoice
    SELECT client_id, SUM(invoice_total) AS invoice_sum, AVG(invoice_total) AS invoice_avg
    FROM invoices
    GROUP BY client_id;
END;
//
DELIMITER ;
```

AI VIET NAM
@aivietnam.edu.vn

Stored Procedures

If we check in the sql_invoicing schema – Stored Procedures section we will we our created procedures



Now to use it we simply call that procedure

```
CALL create_temp_invoice();
```

Let check the result of ours procedure

```
SELECT * FROM temp_invoice;
```

client_id	invoice_sum	invoice_avg
1	802.89	160.58
2	101.79	101.79
3	705.90	141.18
5	980.02	163.34

Unlike temp table, the stored procedures will persist even after we close our session

Outline

- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice
- Python and SQL, PySpark for Big Data

SQL Queries

Trigger

AI VIET NAM
@aivietnam.edu.vn

Trigger

A trigger is a set of SQL statements that run every time a row is inserted, updated, or deleted in a table.

```
mysql> create table employees (
    id bigint primary key auto_increment,
    first_name varchar(100),
    last_name varchar(100));
mysql> create table hiring (emp_id bigint, hire_date timestamp);
mysql> create trigger hire_log after insert on employees
    for each row insert into hiring values (new.id, current_time());
mysql> insert into employees (first_name, last_name) values ("Tim", "Sehn");
mysql> select * from hiring;
+--------+---------------------+
| emp_id | hire_date           |
+--------+---------------------+
|      1 | 2023-06-08 12:21:27 |
+--------+---------------------+
1 row in set (0.00 sec)
```

This trigger inserts a new row into the hiring table every time a row is inserted into the employees table.

```
create trigger
    hire_log -- the name of the trigger
    after -- before or after the change
    insert -- which kind of change, (insert, update, or delete)
    on employees -- the name of the table to watch for changes
    for each row -- boilerplate to begin the trigger body
    insert into hiring values (new.id, current_time()) -- trigger body
;
```

Trigger

Let's create a different trigger to track name changes in our employees.

```
create table name_changes (
    emp_id bigint,
    old_name varchar(200),
    new_name varchar(200),
    change_time timestamp default (current_timestamp())
);

create trigger name_change_trigger
    after update on employees
    for each row
    insert into name_changes (emp_id, old_name, new_name) values
    (new.id,
    concat(old.first_name, concat(" ", old.last_name)),
    concat(new.first_name, concat(" ", new.last_name)));
update employees set last_name = "Holmes" where first_name = "Tim";
select * from name_changes;
+--------+----------+------------+---------------------+
| emp_id | old_name | new_name   | change_time         |
+--------+----------+------------+---------------------+
|      1 | Tim Sehn | Tim Holmes | 2023-06-09 08:36:54 |
+--------+----------+------------+---------------------+
1 row in set (0.00 sec)
```

Trigger: Validation

For example, let's say you want to make sure that a table uses only upper-case letters.

```
create table employees2 (
    id bigint primary key auto_increment,
    first_name varchar(100),
    last_name varchar(100),
    check (upper(first_name) = first_name),
    check (upper(last_name) = last_name)
);

create trigger upper_name_i
    before insert on employees
    for each row
    set new.first_name = upper(new.first_name),
    new.last_name = upper(new.last_name);

create trigger upper_name_u
    before update on employees
    for each row
    set new.first_name = upper(new.first_name),
    new.last_name = upper(new.last_name);
```

```
insert into employees (first_name, last_name) values ('aaron', 'son');
Query OK, 1 row affected (0.02 sec)
select * from employees where first_name = 'AARON';
+----+------------+-----------+
| id | first_name | last_name |
+----+------------+-----------+
|  2 | AARON      | SON       |
+----+------------+-----------+
```

Outline

- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice
- Python and SQL, PySpark for Big Data

AI VIET NAM
@aivietnam.edu.vn

SQL Queries

Practice

Practice #1

Using the sql_hr database, how would you retrieve a list of all employees who earn a higher salary than their manager using a subquery?

```
USE sql_hr;
SELECT e1.*
FROM employees e1
WHERE e1.salary > (
    SELECT e2.salary
    FROM employees e2
    WHERE e1.reports_to = e2.employee_id
);
```

employee_id	first_name	last_name	job_title	salary	reports_to	office_id
37851	Sayer	Matterson	Statistician III	98926	37270	1
40448	Mindy	Crissil	Staff Scientist	94860	37270	1
56274	Keriann	Alloisi	VP Marketing	110150	37270	1
67009	North	de Clerc	VP Product Management	114257	37270	2
67370	Elladine	Rising	Social Worker	96767	37270	2
72540	Guthrey	Iacopetti	Office Assistant I	117690	37270	3
72913	Kass	Hefferan	Computer Systems Analyst IV	96401	37270	3
76196	Mirilla	Janowski	Cost Accountant	119241	37270	3
80529	Lynde	Aronson	Junior Executive	77182	37270	4
80679	Mildrid	Sokale	Geologist II	67987	37270	4
84791	Hazel	Tarbert	General Manager	93760	37270	4
95213	Cole	Kesterton	Pharmacist	86119	37270	4
98374	Estrellita	Daleman	Staff Accountant IV	70187	37270	5
115357	Ivy	Fearey	Structural Engineer	92710	37270	5

Practice #2

Using the sql_hr database, create a temporary table that includes the first name, last name, and office address for each employee

```
DROP TABLE IF EXISTS temp_employee_office;
CREATE TEMPORARY TABLE temp_employee_office AS
SELECT e.first_name, e.last_name, o.address AS office_address
FROM employees e
JOIN offices o ON e.office_id = o.office_id;
```

first_name	last_name	office_address
D'arcy	Nortunen	03 Reinke Trail
Yovonnda	Magrannell	4 Bluestem Parkway
Sayer	Matterson	03 Reinke Trail
Mindy	Crissil	03 Reinke Trail
Keriann	Alloisi	03 Reinke Trail
Alaster	Scutchin	5507 Becker Terrace
North	de Clerc	5507 Becker Terrace
Elladine	Rising	5507 Becker Terrace
Nisse	Voysey	5507 Becker Terrace
Guthrey	Iacopetti	54 Northland Court

Practice #3

In the sql_invoicing database, can you write a stored procedure
-- that inserts a new payment record, then updates the corresponding invoice
-- with the payment total?

```
DROP PROCEDURE IF EXISTS update_payment_invoice;
DELIMITER //
CREATE PROCEDURE update_payment_invoice(
    IN new_payment_id INT,
    IN new_client_id INT,
    IN new_invoice_id INT,
    IN new_date DATE,
    IN new_amount DECIMAL,
    IN new_payment_method INT
)
BEGIN
    INSERT INTO payments (payment_id, client_id, invoice_id, date, amount, payment_method)
    VALUES (new_payment_id, new_client_id, new_invoice_id, new_date, new_amount, new_payment_method);

    UPDATE invoices
    SET payment_total = payment_total + new_amount
    WHERE invoice_id = new_invoice_id;
END //
DELIMITER ;
```

Practice #4

Using the sql_invoicing database, can you create a stored procedure that finds the client who has made the most payments in the last year?

```
DROP PROCEDURE IF EXISTS top_paying_client_last_year;
DELIMITER //
CREATE PROCEDURE top_paying_client_last_year()
BEGIN
    SELECT client_id, COUNT(payment_id) as payment_count
    FROM payments
    WHERE date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 YEAR)
    GROUP BY client_id
    ORDER BY payment_count DESC
    LIMIT 1;
END //
DELIMITER ;
```

Practice #5

In the sql_store database, how would you list the orders that have a total amount greater than the average order amount using a subquery?

```sql
USE sql_store;
SELECT o.order_id
FROM orders o
WHERE (
    SELECT SUM(oi.quantity * oi.unit_price)
    FROM order_items oi
    WHERE oi.order_id = o.order_id
) > (
    SELECT AVG(order_amount)
    FROM (
        SELECT oi.order_id, SUM(oi.quantity * oi.unit_price) as order_amount
        FROM order_items oi
        GROUP BY oi.order_id
    ) as average_order
);
```

Practice #6

In the sql_store database, how would you identify the product that appears most frequently in orders using a Common Table Expression?

```sql
USE sql_store;
WITH product_frequency AS (
    SELECT product_id, COUNT(*) as count
    FROM order_items
    GROUP BY product_id
)
SELECT product_id, count
FROM product_frequency
ORDER BY count DESC
LIMIT 1;
```

Practice #7

Using the sql_store database, create a temporary table that includes the customer_id, total quantity of products ordered, and total amount spent by each customer.

```sql
USE sql_store;
CREATE TEMPORARY TABLE temp_customer_totals AS
SELECT
    o.customer_id,
    SUM(oi.quantity) AS total_quantity,
    SUM(oi.quantity * oi.unit_price) AS total_spent
FROM
    orders o
    JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY
    o.customer_id;

SELECT *
FROM temp_customer_totals;
```

Practice #8

Using the sql_hr database, how would you find the employee who earns the least in each office using a Common Table Expression?

```sql
USE sql_hr;
WITH min_salary_office AS (
    SELECT office_id, MIN(salary) as min_salary
    FROM employees
    GROUP BY office_id
)
SELECT e.first_name, e.last_name, e.office_id, e.salary
FROM employees e
JOIN min_salary_office mso ON e.office_id = mso.office_id AND e.salary = mso.min_salary;
```

Practice #9

In the sql_invoicing database, can you write a stored procedure to calculate the total unpaid amount for each client?

```sql
USE sql_invoicing;
DROP PROCEDURE IF EXISTS calculate_unpaid_amount;
DELIMITER //
CREATE PROCEDURE calculate_unpaid_amount()
BEGIN
    SELECT
        c.client_id,
        c.name,
        SUM(i.invoice_total) - IFNULL(SUM(p.amount), 0) AS unpaid_amount
    FROM
        clients c
        LEFT JOIN invoices i ON c.client_id = i.client_id
        LEFT JOIN payments p ON i.invoice_id = p.invoice_id
    GROUP BY
        c.client_id, c.name;
END //
DELIMITER ;
```

Practice #10

Using the sql_invoicing database, how would you list the top 5 clients who have the most unpaid invoices using a subquery?

```sql
USE sql_invoicing;
SELECT
    c.client_id,
    c.name,
    unpaid_amounts.unpaid_amount
FROM
    clients c
    JOIN (
        SELECT
            client_id,
            SUM(invoice_total) - IFNULL(SUM(payment_total), 0) AS unpaid_amount
        FROM
            invoices
        GROUP BY
            client_id
    ) AS unpaid_amounts ON c.client_id = unpaid_amounts.client_id
ORDER BY
    unpaid_amounts.unpaid_amount DESC
LIMIT 5;
```

Practice #11

Using the sql_store database, how would you determine the total revenue made from each product using a subquery?

```sql
USE sql_store;
SELECT
    p.product_id,
    p.name,
    SUM(order_item_totals.total_amount) AS total_revenue
FROM
    products p
    JOIN (
        SELECT
            product_id,
            quantity * unit_price AS total_amount
        FROM
            order_items
    ) AS order_item_totals ON p.product_id = order_item_totals.product_id
GROUP BY
    p.product_id, p.name;
```

Practice #12

In the sql_hr database, create a stored procedure that promotes an employee to a manager position, updating their salary and reports_to fields accordingly.

```
USE sql_hr;
DROP PROCEDURE IF EXISTS promote_to_manager;
DELIMITER //
CREATE PROCEDURE promote_to_manager(
    IN emp_id INT,
    IN new_salary DECIMAL(10, 2),
    IN new_reports_to INT
)
BEGIN
    UPDATE employees
    SET salary = new_salary,
        reports_to = new_reports_to
    WHERE employee_id = emp_id;
END //
DELIMITER ;
```

Practice #13

In the sql_invoicing database, can you write a stored procedure that applies a discount to all invoices over a certain amount?

```
USE sql_invoicing;
DROP PROCEDURE IF EXISTS apply_discount;
DELIMITER //
CREATE PROCEDURE apply_discount(
    IN min_invoice_total DECIMAL(10, 2),
    IN discount_rate DECIMAL(5, 2)
)
BEGIN
    UPDATE invoices
    SET invoice_total = invoice_total * (1 - discount_rate)
    WHERE invoice_total > min_invoice_total;
END //
DELIMITER ;
```

Practice #14

In the sql_store database, how would you find the order that has the highest total amount using a Common Table Expression?

```sql
USE sql_store;
WITH order_totals AS (
    SELECT
        order_id,
        SUM(quantity * unit_price) AS total_amount
    FROM
        order_items
    GROUP BY
        order_id
)
SELECT
    order_id,
    total_amount
FROM
    order_totals
ORDER BY
    total_amount DESC
LIMIT 1;
```

Practice #15

Using the sql_store database, create a temporary table that includes the product_id, total quantity sold, and total revenue from each product.

```sql
USE sql_store;
CREATE TEMPORARY TABLE product_sales AS
SELECT
    product_id,
    SUM(quantity) AS total_quantity_sold,
    SUM(quantity * unit_price) AS total_revenue
FROM
    order_items
GROUP BY
    product_id;

SELECT * FROM product_sales;
```

Practice #16

Using the sql_hr database, can you write a stored procedure that calculates the total salary expense for each office?

```sql
USE sql_hr;
DROP PROCEDURE IF EXISTS calculate_office_salaries;
DELIMITER //
CREATE PROCEDURE calculate_office_salaries()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE o_id INT;
  DECLARE total_salary DECIMAL(10, 2);
  DECLARE office_cursor CURSOR FOR SELECT office_id FROM offices;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  CREATE TEMPORARY TABLE office_salaries (
    office_id INT,
    total_salary DECIMAL(10, 2)
  );

  OPEN office_cursor;

  read_loop: LOOP
    FETCH office_cursor INTO o_id;

    IF done THEN
      LEAVE read_loop;
    END IF;

    SELECT SUM(salary) INTO total_salary
    FROM employees
    WHERE office_id = o_id;

    INSERT INTO office_salaries VALUES (o_id, total_salary);
  END LOOP;

  CLOSE office_cursor;
END //
```

Outline

- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice
- Python and SQL, PySpark for Big Data

AI VIET NAM
@aivietnam.edu.vn

Python and MySQL

```
!pip install mysql-connector-python
```

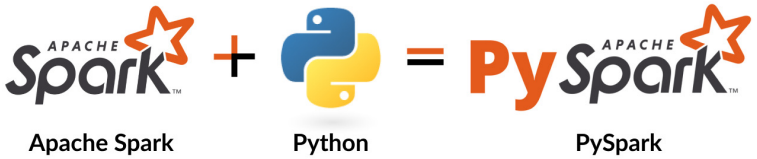
```
import mysql.connector as mysql
from mysql.connector import errorcode
db_name = "sql_invoicing"
db_host = "localhost"
db_username = "root"
db_password = "root@123"


conn = mysql.connect(host = db_host,
                     port = int(3306),
                     user = "root",
                     password = db_password,
                     db = db_name)
```

```
import pandas as pd
df = pd.read_sql_query('Select * FROM clients', conn)
```

```
df.head()
```

	client_id	name	address	city	state	phone
0	1	Vinte	3 Nevada Parkway	Syracuse	NY	315-252-7305
1	2	Myworks	34267 Glendale Parkway	Huntington	WV	304-659-1170
2	3	Yadel	096 Pawling Parkway	San Francisco	CA	415-144-6037
3	4	Kwideo	81674 Westerfield Circle	Waco	TX	254-750-0784
4	5	Topiclounge	0863 Farmco Road	Portland	OR	971-888-9129

PySpark for Big Data



PySpark for Big Data

```python
# Imports
from pyspark.sql import SparkSession

spark = SparkSession \
        .builder \
        .appName("test") \
        .config("spark.jars", "mysql-connector-j-9.0.0.jar") \
        .getOrCreate()
```

```python
df = spark.read \
    .format("jdbc") \
    .option("driver","com.mysql.cj.jdbc.Driver") \
    .option("url", "jdbc:mysql://localhost:3306/sql_invoicing") \
    .option("dbtable", "clients") \
    .option("user", "root") \
    .option("password", "root@1
    .load()
```

```python
df.show()
```

```
+---------+----------+--------------------+-------------+-----+------------+
|client_id|      name|             address|         city|state|       phone|
+---------+----------+--------------------+-------------+-----+------------+
|        1|     Vinte|    3 Nevada Parkway|     Syracuse|   NY|315-252-7305|
|        2|   Myworks|34267 Glendale Pa...|   Huntington|   WV|304-659-1170|
|        3|     Yadel| 096 Pawling Parkway|San Francisco|   CA|415-144-6037|
|        4|   Kwideo|81674 Westerfield...|         Waco|   TX|254-750-0784|
|        5|Topiclounge|    0863 Farmco Road|     Portland|   OR|971-888-9129|
+---------+----------+--------------------+-------------+-----+------------+
```

PySpark for Big Data

```
query = "select * from clients where state = 'NY'"
df = spark.read \
    .format("jdbc") \
    .option("driver","com.mysql.cj.jdbc.Driver") \
    .option("url", "jdbc:mysql://localhost:3306/sql_invoicing") \
    .option("query", query) \
    .option("user", "root") \
    .option("password", "root@123") \
    .load()
```

```
df.show()
```

```
+---------+-----+----------------+--------+-----+------------+
|client_id| name|         address|    city|state|       phone|
+---------+-----+----------------+--------+-----+------------+
|        1|Vinte|3 Nevada Parkway|Syracuse|   NY|315-252-7305|
+---------+-----+----------------+--------+-----+------------+
```