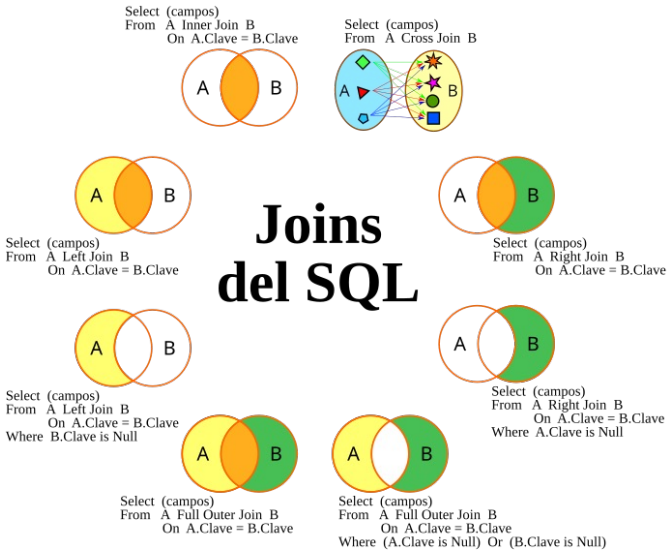
AI VIET NAM
@aivietnam.edu.vn

SQL in Data Analysis

(Subqueries, Procedure and Trigger and Advanced Topics)



Vinh Dinh Nguyen - PhD in Computer Science
Hao The Nguyen - STA

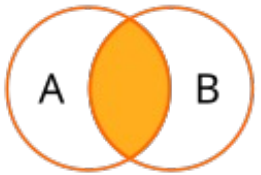
Data & Code

Outline

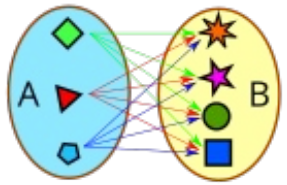
- SQL Join
- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice

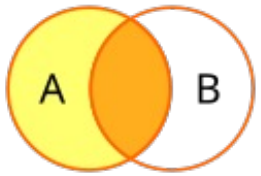
Joins del SQL

Select (campos)
From A Inner Join B
On A.Clave = B.Clave

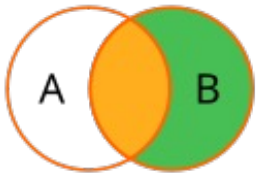


Select (campos)
From A Cross Join B

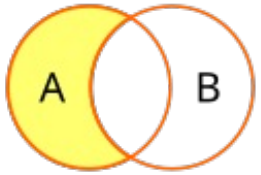




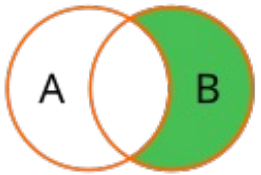
Select (campos)
From A Left Join B
On A.Clave = B.Clave



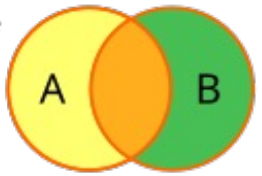
Select (campos)
From A Right Join B
On A.Clave = B.Clave



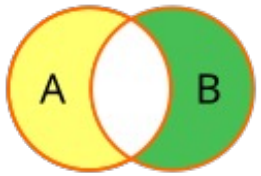
Select (campos)
From A Left Join B
On A.Clave = B.Clave
Where B.Clave is Null



Select (campos)
From A Right Join B
On A.Clave = B.Clave
Where A.Clave is Null



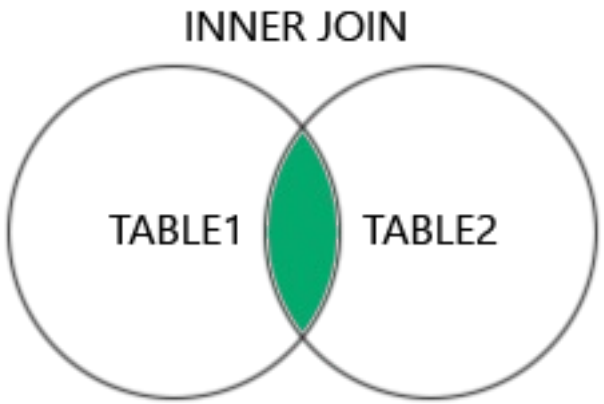
Select (campos)
From A Full Outer Join B
On A.Clave = B.Clave



Select (campos)
From A Full Outer Join B
On A.Clave = B.Clave
Where (A.Clave is Null) Or (B.Clave is Null)

SQL Queries

INNER JOIN



INNER JOIN

order_id	customer_id	order_date	status	comments	shipped_date	shipper_id
1	6	2019-01-30	1	NULL	NULL	NULL
2	7	2023-05-30	2	NULL	2018-08-03	4
3	8	2017-12-01	1	NULL	NULL	NULL
4	2	2017-01-22	1	NULL	NULL	NULL
5	5	2017-08-25	2		2017-08-26	3
6	10	2018-11-18	1	Aliquam erat volutpat. In congue.	NULL	NULL
7	2	2018-09-22	2	NULL	2018-09-23	4
8	5	2018-06-08	1	Mauris enim leo, rhoncus sed, vestibulum sit am…	NULL	NULL
9	10	2017-07-05	2	Nulla mollis molestie lorem. Quisque ut erat.	2017-07-06	1
10	6	2018-04-22	2	NULL	2018-04-23	2

How can we query across multible tables? For example: How can we retrieve which customers placing the order #5?

customer_id	first_name	last_name	birth_date	phone	address	city	state	points
1	Babara	MacCaffrey	1986-03-28	781-932-9754	0 Sage Terrace	Waltham	MA	2273
2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	MA	947
3	Freddi	Boagey	1985-02-07	719-724-7869	251 Springs Junction	Colorado Springs	CO	2967
4	Ambur	Roseburgh	1974-04-14	407-231-8017	30 Arapahoe Terrace	Orlando	CO	457
5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	TX	3073
7	Ilene	Dowson	1964-08-30	615-641-4759	50 Lillian Crossing	Nashville	TN	1672
8	Thacher	Naseby	1993-07-17	941-527-3977	538 Mosinee Center	Sarasota	TN	205
9	Romola	Rumgay	1992-05-23	559-181-3744	3520 Ohio Trail	Visalia	GA	1486
10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796

INNER JOIN

1/ Get the first and last name of every customer placing orders

```
SELECT *
FROM orders
INNER JOIN customers ON  orders.customer_id = customers.customer_id;
```

order_id	customer_id	order_date	status	comments	shipped_date	shipper_id
1	6	2019-01-30	1	NULL	NULL	NULL
2	7	2023-05-30	2	NULL	2018-08-03	4
3	8	2017-12-01	1	NULL	NULL	NULL
4	2	2017-01-22	1	NULL	NULL	NULL
5	5	2017-08-25	2		2017-08-26	3
6	10	2018-11-18	1	Aliquam erat volutpat. In congue.	NULL	NULL
7	2	2018-09-22	2	NULL	2018-09-23	4
8	5	2018-06-08	1	Mauris enim leo, rhoncus sed, vestibulum sit am...	NULL	NULL
9	10	2017-07-05	2	Nulla mollis molestie lorem. Quisque ut erat.	2017-07-06	1
10	6	2018-04-22	2	NULL	2018-04-23	2

customer_id	first_name	last_name	birth_date	phone	address	city	state	points
6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	TX	3073
7	Ilene	Dowson	1964-08-30	615-641-4759	50 Lillian Crossing	Nashville	TN	1672
8	Thacher	Naseby	1993-07-17	941-527-3977	538 Mosinee Center	Sarasota	TN	205
2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	MA	947
5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	MA	947
5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	TX	3073

You can see that the customer_id from the customers table now match with the customer_id from the orders table.

Let refine our query.

INNER JOIN

1/ Get the order id, first and last name of every customer placing orders

```
SELECT order_id, first_name, last_name
FROM orders
INNER JOIN customers ON  orders.customer_id = customers.customer_id;
```

order_id	first_name	last_name
4	Ines	Brushfield
7	Ines	Brushfield
5	Clemmie	Betchley
8	Clemmie	Betchley
1	Elka	Twiddell
10	Elka	Twiddell
2	Ilene	Dowson
3	Thacher	Naseby
6	Levy	Mynett
9	Levy	Mynett

We don’t want to keep repeating our table name every time. We can use Alias to keep our query short.

Let refine our query.

INNER JOIN

1/ Get the order id, first and last name of every customer placing orders

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

order_id	first_name	last_name
4	Ines	Brushfield
7	Ines	Brushfield
5	Clemmie	Betchley
8	Clemmie	Betchley
1	Elka	Twiddell
10	Elka	Twiddell
2	Ilene	Dowson
3	Thacher	Naseby
6	Levy	Mynett
9	Levy	Mynett

INNER JOIN

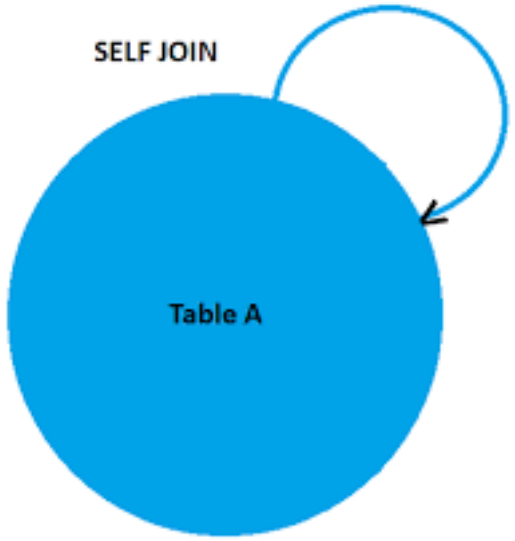
2/ Join the order_items table with the products table return the product_id, name, quantity, unit_price, and price = quantity * unit_price.

```
SELECT oi.product_id, p.name, oi.quantity, oi.unit_price, quantity * oi.unit_price AS price
FROM order_items oi
JOIN products p
ON  oi.product_id = p.product_id;
```

produc...	name	quantity	unit_price	price
1	Foam Dinner Plate	2	9.10	18.20
1	Foam Dinner Plate	4	8.65	34.60
1	Foam Dinner Plate	10	6.01	60.10
2	Pork - Bacon,back Peameal	3	9.89	29.67
2	Pork - Bacon,back Peameal	4	3.28	13.12
3	Lettuce - Romaine, Heart	10	9.12	91.20
3	Lettuce - Romaine, Heart	7	6.99	48.93
3	Lettuce - Romaine, Heart	4	7.46	29.84
3	Lettuce - Romaine, Heart	7	9.17	64.19
4	Brocolinni - Gaylan, Chinese	4	3.74	14.96
4	Brocolinni - Gaylan, Chinese	4	1.66	6.64
5	Sauce - Ranch Dressing	1	3.45	3.45
5	Sauce - Ranch Dressing	2	6.94	13.88
6	Petit Baguette	2	2.94	5.88
6	Petit Baguette	5	7.28	36.40
8	Island Oasis - Raspberry	2	8.59	17.18
9	Longan	9	4.28	38.52
10	Broom - Push	7	6.40	44.80

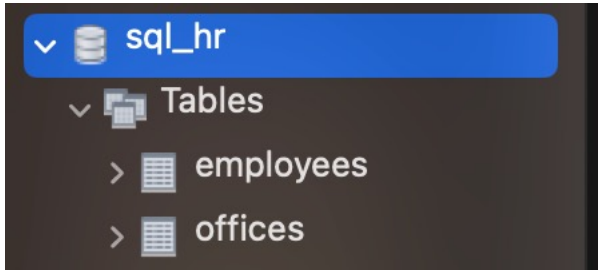
SQL Queries

SELF JOIN



SELF JOIN

Let take a look at the sql_hr schema, here is the data from the employees tables



employee_id	first_name	last_name	job_title	salary	reports_to	office_id
33391	D'arcy	Nortunen	Account Executive	62871	37270	1
37270	Yovonnda	Magrannell	Executive Secretary	63996	NULL	10
37851	Sayer	Matterson	Statistician III	98926	37270	1
40448	Mindy	Crissil	Staff Scientist	94860	37270	1
56274	Keriann	Alloisi	VP Marketing	110150	37270	1
63196	Alaster	Scutchin	Assistant Professor	32179	37270	2
67009	North	de Clerc	VP Product Management	114257	37270	2
67370	Elladine	Rising	Social Worker	96767	37270	2
68249	Nisse	Voysey	Financial Advisor	52832	37270	2
72540	Guthrey	Iacopetti	Office Assistant I	117690	37270	3
72913	Kass	Hefferan	Computer Systems Ana…	96401	37270	3
75900	Virge	Goodrum	Information Systems M…	54578	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
96513	Theresa	Binney	Food Chemist	47354	37270	5
98374	Estrellita	Daleman	Staff Accountant IV	70187	37270	5
115357	Ivy	Fearey	Structural Engineer	92710	37270	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL

This is the ID of the manager for that employee.

Now we can see that the manager ID is related back to the same table.

So how can we get the name of each employee and their manager?

SELF JOIN

1/ Get the first and last name of every employees and their respective manager

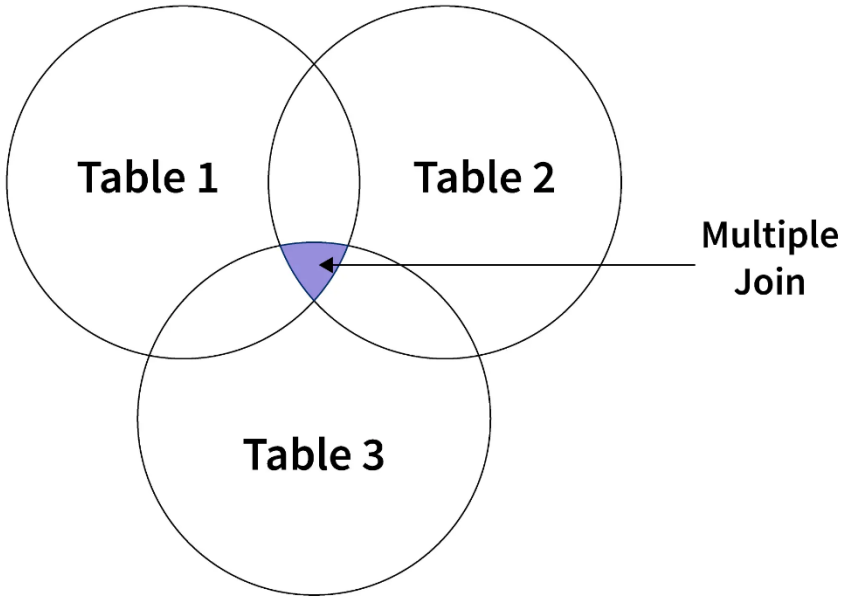
```
USE sql_hr;

SELECT
        e.employee_id,
        e.first_name,
        e.last_name,
        m.employee_id as manager_id,
        m.first_name,
        m.last_name
FROM employees e
JOIN employees m
        ON  e.reports_to = m.employee_id;
```

employee_id	first_name	last_name	manager_id	first_name	last_name
33391	D'arcy	Nortunen	37270	Yovonnda	Magrannell
37851	Sayer	Matterson	37270	Yovonnda	Magrannell
40448	Mindy	Crissil	37270	Yovonnda	Magrannell
56274	Keriann	Alloisi	37270	Yovonnda	Magrannell
63196	Alaster	Scutchin	37270	Yovonnda	Magrannell
67009	North	de Clerc	37270	Yovonnda	Magrannell
67370	Elladine	Rising	37270	Yovonnda	Magrannell
68249	Nisse	Voysey	37270	Yovonnda	Magrannell
72540	Guthrey	Iacopetti	37270	Yovonnda	Magrannell
72913	Kass	Hefferan	37270	Yovonnda	Magrannell
75900	Virge	Goodrum	37270	Yovonnda	Magrannell
76196	Mirilla	Janowski	37270	Yovonnda	Magrannell
80529	Lynde	Aronson	37270	Yovonnda	Magrannell
80679	Mildrid	Sokale	37270	Yovonnda	Magrannell
84791	Hazel	Tarbert	37270	Yovonnda	Magrannell
95213	Cole	Kesterton	37270	Yovonnda	Magrannell
96513	Theresa	Binney	37270	Yovonnda	Magrannell
98374	Estrellita	Daleman	37270	Yovonnda	Magrannell
115357	Ivy	Fearey	37270	Yovonnda	Magrannell

NOTE: When doing a self join we must make sure that we use different alias's for our tables

SQL Queries

JOINNING MUTIPLE TABLES



JOINNING MUTIPLE TABLES

Let go back to our sql_store schema and take a look at orders tables

order_id	customer_id	order_date	status	comments	shipped_date	shipper_id
1	6	2019-01-30	1	NULL	NULL	NULL
2	7	2023-05-30	2	NULL	2018-08-03	4
3	8	2017-12-01	1	NULL	NULL	NULL
4	2	2017-01-22	1	NULL	NULL	NULL
5	5	2017-08-25	2		2017-08-26	3
6	10	2018-11-18	1	Aliquam erat volutpat. In congue.	NULL	NULL
7	2	2018-09-22	2	NULL	2018-09-23	4
8	5	2018-06-08	1	Mauris enim leo, rhoncus sed, vestibulum sit amet, cursus id, turpis.	ULL	
9	10	2017-07-05	2	Nulla mollis molestie lorem. Quisque ut erat.	2017-07-06	1
10	6	2018-04-22	2	NULL	2018-04-23	2

This point to our customers table

This point to our orders_statuses table

How can we retrieve the infomation from three tables? For example: We want to get the order_id, order_date from the orders table, the first and last name of the customer placing the order, and the status of that order?

JOINNING MUTIPLE TABLES

1/ Retrieve the order_id, order_date, first_name, last_name, and status of each orders

```
USE sql_store;

SELECT
        o.order_id,
        o.order_date,
        c.last_name,
        c.first_name,
        s.name AS status
FROM orders o
JOIN customers c
        ON  o.customer_id = c.customer_id
JOIN order_statuses s
        ON o.status = s.order_status_id;
```

order_id	order_date	last_name	first_name	status
8	2018-06-08	Betchley	Clemmie	Processed
6	2018-11-18	Mynett	Levy	Processed
4	2017-01-22	Brushfield	Ines	Processed
3	2017-12-01	Naseby	Thacher	Processed
1	2019-01-30	Twiddell	Elka	Processed
10	2018-04-22	Twiddell	Elka	Shipped
9	2017-07-05	Mynett	Levy	Shipped
7	2018-09-22	Brushfield	Ines	Shipped
5	2017-08-25	Betchley	Clemmie	Shipped
2	2023-05-30	Dowson	Ilene	Shipped

JOINNING MUTIPLE TABLES

2/ From the invoicing database, retrieve respective to payments table

- **payment_id, date, amount from payments table**
- **name, phone from clients table**
- **the invoice number, invoice_total, payment_total from invoices table**
- **and the payment method name from payment_methods table**

payment_id	date	amount	name	phone	number	invoice_total	payment_total	payment_meth...
1	2019-02-12	8.18	Topiclounge	971-888-9129	03-898-6735	175.32	8.18	Credit Card
2	2019-01-03	74.55	Vinte	315-252-7305	75-587-6626	157.78	74.55	Credit Card
3	2019-01-11	0.03	Yadel	415-144-6037	20-848-0181	126.15	0.03	Credit Card
4	2019-01-26	87.44	Topiclounge	971-888-9129	41-666-1035	135.01	87.44	Credit Card
5	2019-01-15	80.31	Yadel	415-144-6037	55-105-9605	167.29	80.31	Credit Card
6	2019-01-15	68.10	Yadel	415-144-6037	33-615-4694	126.38	68.10	Credit Card
7	2019-01-08	32.77	Topiclounge	971-888-9129	52-269-9803	180.17	42.77	Credit Card
8	2019-01-08	10.00	Topiclounge	971-888-9129	52-269-9803	180.17	42.77	Cash

```
USE sql_invoicing;
SELECT
        p.payment_id, p.date, p.amount,
        c.name, c.phone,
        i.number, i.invoice_total, i.payment_total,
        pm.name AS payment_method
FROM payments p
JOIN clients c ON p.client_id = c.client_id
JOIN invoices i ON p.invoice_id = i.invoice_id
JOIN payment_methods pm ON p.payment_method = pm.payment_method_id;
```

SQL Queries

COMPOUND JOIN CONDITION

COMPOUND JOIN CONDITION

Let go back to our sql_store schema and take a look at orders_items tables

order_id	product_id	quantity	unit_price
1	4	4	3.74
2	1	2	9.10
2	4	4	1.66
2	6	2	2.94
3	3	10	9.12
4	3	7	6.99
4	10	7	6.40
5	2	3	9.89
6	1	4	8.65
6	2	4	3.28
6	3	4	7.46
6	5	1	3.45
7	3	7	9.17
8	5	2	6.94
8	8	2	8.59
9	6	5	7.28
10	1	10	6.01
10	9	9	4.28

Both of these column are not unique

This is call composite primary key, that mean we need both column to uniquely identify a record.

note_id	order_Id	product_id	note
1	1	2	first note
2	1	2	second note

So how can we join a table with composite key?

COMPOUND JOIN CONDITION

1/ Retrieve the note for the respective order item

```
USE sql_store;

SELECT *
FROM order_items o
JOIN order_item_notes n
        ON o.order_id = n.order_id
        AND o.product_id = n.product_id;
```

order_id	product_id	quantity	unit_price	note_id	order_Id	product_id	note
2	6	2	2.94	1	2	6	first note
5	2	3	9.89	2	5	2	second note

SQL Queries

IMPLICIT JOIN SYNTAX

IMPLICIT JOIN SYNTAX

Let see an example JOIN statment.

```
SELECT *
FROM orders o
JOIN customer c
      ON o.customer_id = c.customer_id;
```

✅ Recommended

Separates **join logic** from **filtering logic**

And it Implicit Join Syntax

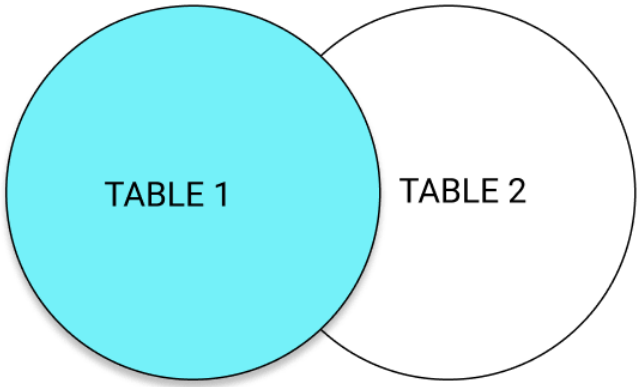
```
SELECT *
FROM orders o, customers c
WHERE o.customer_id = c.customer_id;
```

Harder to distinguish between join conditions and filtering conditions.

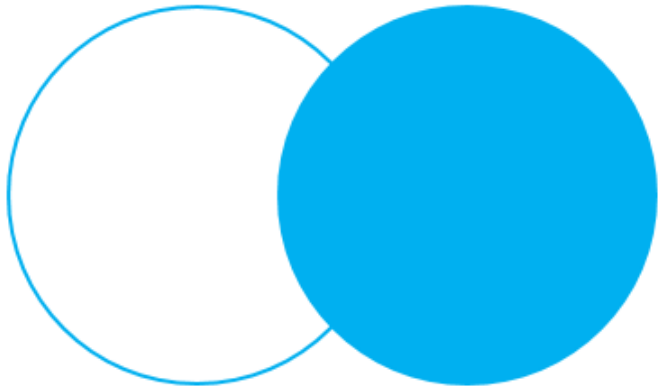
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	customer_id	order_date	status	comments	shipped_date	shipper_id	customer_id	first_name	last_name	birth_date	phone	address	city	state	points
1	6	2019-01-30	1	NULL	NULL	NULL	6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	IL	3073
2	7	2018-08-02	2	NULL	2018-08-03	4	7	Ilene	Dowson	1964-08-30	615-641-4759	50 Lillian Crossing	Nashville	TN	1672
3	8	2017-12-01	1	NULL	NULL	NULL	8	Thacher	Naseby	1993-07-17	941-527-3977	538 Mosinee Center	Sarasota	FL	205
4	2	2017-01-22	1	NULL	NULL	NULL	2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	VA	947
5	5	2017-08-25	2		2017-08-26	3	5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
6	10	2018-11-18	1	Aliquam erat volutpat. In congue.	NULL	NULL	10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
7	2	2018-09-22	2	NULL	2018-09-23	4	2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	VA	947
8	5	2018-06-08	1	Mauris enim leo, rhoncus sed, vestibulum sit am...	NULL	NULL	5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
9	10	2017-07-05	2	Nulla mollis molestie lorem. Quisque ut erat.	2017-07-06	1	10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
10	6	2018-04-22	2	NULL	2018-04-23	2	6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	IL	3073

SQL Queries
OUTER JOIN



LEFT OUTER JOIN



RIGHT OUTER JOIN

OUTER JOIN

Let see an example JOIN statment.

```
SELECT
	c.customer_id,
	c.first_name,
	o.order_id
FROM customers c
JOIN orders o
	ON c.customer_id = o.customer_id
ORDER BY  c.customer_id;
```

customer_id	first_name	order_id
2	Ines	4
2	Ines	7
5	Clemmie	5
5	Clemmie	8
6	Elka	1
6	Elka	10
7	Ilene	2
8	Thacher	3
10	Levy	6
10	Levy	9

We can see that only the customer who have place an order is shown in our result.
What if we want to see all customer ?

OUTER JOIN

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

When using LEFT JOIN all records from the left table i.e customers are returned whether the condition is true or not

All customer id records will be returned

customer_id	first_name	order_id
2	Ines	4
2	Ines	7
5	Clemmie	5
5	Clemmie	8
6	Elka	1
6	Elka	10
7	Ilene	2
8	Thacher	3
10	Levy	6
10	Levy	9

OUTER JOIN

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

When using RIGHT JOIN all records from the right table i.e customers are returned whether the condition is true or not

customer_id	first_name	order_id
2	Ines	4
2	Ines	7
5	Clemmie	5
5	Clemmie	8
6	Elka	1
6	Elka	10
7	Ilene	2
8	Thacher	3
10	Levy	6
10	Levy	9

All order_id records will be returned

OUTER JOIN

1/ Write a query to get all customer id, their first name, their order id, and shipper name for that order

```
SELECT
        c.customer_id,
        c.first_name,
        o.order_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;
```

Result Grid | Filter Rows: | Export:

customer_id	first_name	order_id	shipper
1	Babara	NULL	NULL
2	Ines	4	NULL
2	Ines	7	Mraz, Renner and Nolan
3	Freddi	NULL	NULL
4	Ambur	NULL	NULL
5	Clemmie	5	Satterfield LLC
5	Clemmie	8	NULL
6	Elka	1	NULL
6	Elka	10	Schinner-Predovic
7	Ilene	2	Mraz, Renner and Nolan
8	Thacher	3	NULL
9	Romola	NULL	NULL
10	Levy	6	NULL
10	Levy	9	Hettinger LLC

SQL Queries

USING CLAUSE

USING CLAUSE

Let return to ours example JOIN statement.

```
SELECT
        c.customer_id,
        c.first_name,
        o.order_id
FROM customers c
JOIN orders o
        ON c.customer_id = o.customer_id
ORDER BY  c.customer_id;
```

customer_id	first_name	order_id
2	Ines	4
2	Ines	7
5	Clemmie	5
5	Clemmie	8
6	Elka	1
6	Elka	10
7	Ilene	2
8	Thacher	3
10	Levy	6
10	Levy	9

If both column of the JOIN condition have the same name i.e. “customer_id” we can simplify the query

```
SELECT
        c.customer_id,
        c.first_name,
        o.order_id
FROM customers c
JOIN orders o
        USING (customer_id)
ORDER BY  c.customer_id;
```

USING CLAUSE

Let return to ours example JOIN statement.

```
SELECT
        c.customer_id,
        c.first_name,
        o.order_id
FROM customers c
JOIN orders o
        ON c.customer_id = o.customer_id
ORDER BY  c.customer_id;
```

customer_id	first_name	order_id
2	Ines	4
2	Ines	7
5	Clemmie	5
5	Clemmie	8
6	Elka	1
6	Elka	10
7	Ilene	2
8	Thacher	3
10	Levy	6
10	Levy	9

If both column of the JOIN condition have the same name i.e. “customer_id” we can simplify the query

```
SELECT
        c.customer_id,
        c.first_name,
        o.order_id
FROM customers c
JOIN orders o
        USING (customer_id)
ORDER BY  c.customer_id;
```

USING CLAUSE

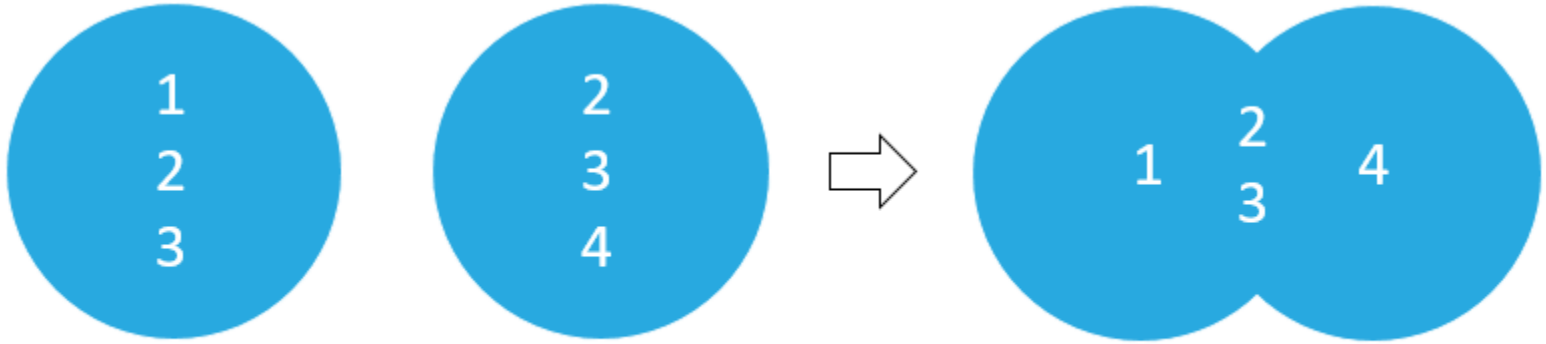
This is extremely useful when writing compind join conditions.

```
SELECT *
FROM order_items o
JOIN order_item_notes n
        ON o.order_id = n.order_id
        AND  o.product_id  = n.product_id;
```

```
SELECT *
FROM order_items o
JOIN order_item_notes n
        USING (order_id, product_id);
```

order_id	product_id	quantity	unit_price	note_id	order_id	product_id	note
2	6	2	2.94	1	2	6	first note
5	2	3	9.89	2	5	2	second note

SQL Queries
UNIONS



UNIONS

Let see the result of this query.

```
SELECT customer_id, first_name, points
FROM customers;
```

customer_id	first_name	points
1	Babara	2273
2	Ines	947
3	Freddi	2967
4	Ambur	457
5	Clemmie	3675
6	Elka	3073
7	Ilene	1672
8	Thacher	205
9	Romola	1486
10	Levy	796

What if we tank to rank them by tier:

- **<2000 point: Bronze**
- **2000-3000 point: Silver**
- **>3000 point: Gold**

customer_id	first_name	points	tier
4	Ambur	457	Bronze
1	Babara	2273	Silver
5	Clemmie	3675	Gold
6	Elka	3073	Gold
3	Freddi	2967	Silver
7	Ilene	1672	Bronze
2	Ines	947	Bronze
10	Levy	796	Bronze
9	Romola	1486	Bronze
8	Thacher	205	Bronze

UNIONS

Let query the result for Bronze tier.

```
SELECT customer_id, first_name, points, 'Bronze' AS tier
FROM customers
WHERE points < 2000
```

customer_id	first_name	points	tier
2	Ines	947	Bronze
4	Ambur	457	Bronze
7	Ilene	1672	Bronze
8	Thacher	205	Bronze
9	Romola	1486	Bronze
10	Levy	796	Bronze

UNION

For Silver tier.

```
SELECT customer_id, first_name, points, 'Silver' AS tier
FROM customers
WHERE points BETWEEN 2000 AND 3000
```

customer_id	first_name	points	tier
1	Babara	2273	Silver
3	Freddi	2967	Silver

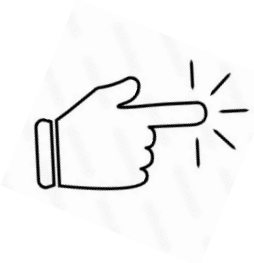
UNION

For Gold tier.

```
SELECT customer_id, first_name, points, 'Gold' AS tier
FROM customers
WHERE points > 3000;
```

customer_id	first_name	points	tier
5	Clemmie	3675	Gold
6	Elka	3073	Gold

Outline



- SQL Join
- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice

SQL Queries
Common Table Expression

```
WITH ten_cte AS (
    SELECT ...
    FROM ...
    WHERE ...
)
SELECT *
FROM ten_cte
WHERE ...;
```

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

- SQL Join
- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice

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

- SQL Join
- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice

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?

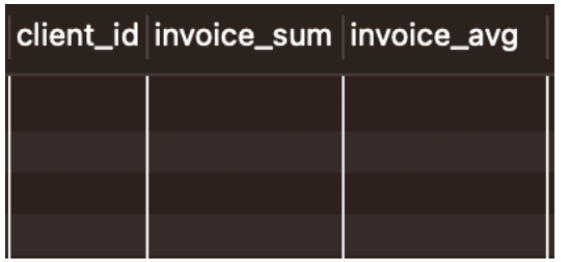
Temp Table

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;
```



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

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

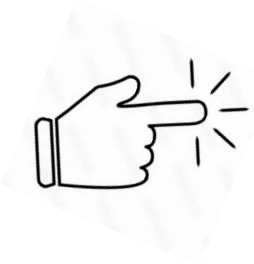
```
SELECT client_id, name,
            (SELECT invoice_avg
             FROM temp_invoice
             WHERE c.client_id = client_id) AS invoice_avg
FROM clients c;
```

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
- Advanced SQL Topics

SQL Queries

Stored Procedures

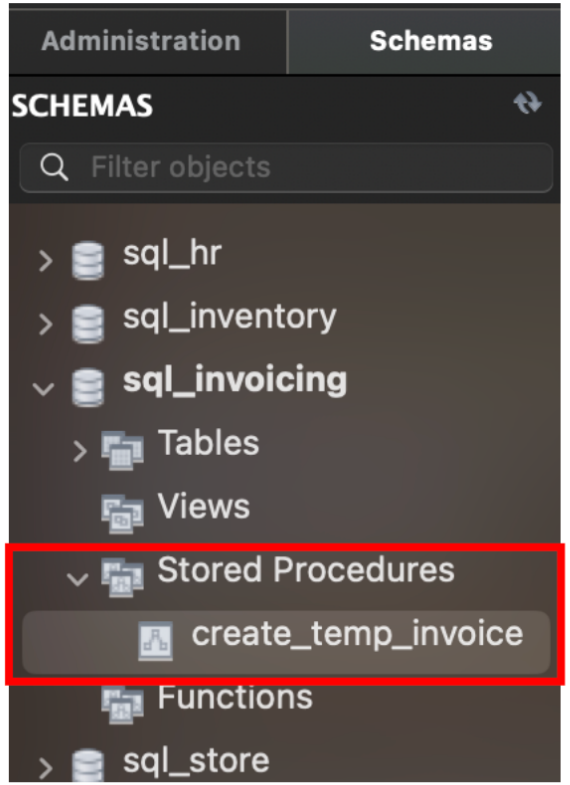
Stored Procedures

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 ;
```

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

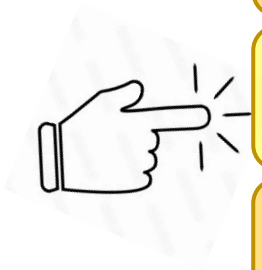
```
SELECT * FROM temp_invoice;
```

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
- Advanced SQL Topics



SQL Queries

Trigger

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

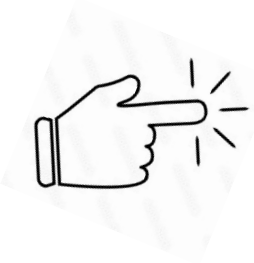
For example, let's say you want to make sure that a table uses only upper-case letters.

```
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

- SQL Join
- Common Table Expression
- SQL Subqueries
- SQL Temp Table
- SQL Store Procedures
- SQL Trigger
- Practice



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?

```
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?

```
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 ;
```

FriendName	City	State	Country
María	Acapulco	Guerrero	México
Fernando	Caracas	Distrito Capital	Venezuela
Gerson	Medellín	Antioquía	Colombia
Mónica	Bogotá	Cundinamarca	Colombia
Paul	Bogotá	Cundinamarca	Colombia
Kevin	Lexington	Kentucky	USA
Cecilia	Godoy Cruz	Mendoza	Argentina
Pablo	Atlántida	Canelones	Uruguay
Andrea	Cdad. Mendoza	Mendoza	Argentina
Marlon	Sao Paulo	Sao Paulo	Brasil
Joao	Rio de Janeiro	Rio de Janeiro	Brasil
Andrés	Bariloche	Río Negro	Argentina
Mariano	Miami	Florida	USA

```
SELECT
  Country,
  COUNT(*) AS HowMany
FROM WorldWideFriends
GROUP BY Country;
```

Country	HowMany
Argentina	3
Venezuela	1
Colombia	3
Brasil	2
USA	2
México	1
Uruguay	1

FriendName	City	State	Country
María	Acapulco	Guerrero	México
Fernando	Caracas	Distrito Capital	Venezuela
Gerson	Medellín	Antioquía	Colombia
Mónica	Bogotá	Cundinamarca	Colombia
Paul	Bogotá	Cundinamarca	Colombia
Kevin	Lexington	Kentucky	USA
Cecilia	Godoy Cruz	Mendoza	Argentina
Pablo	Atlántida	Canelones	Uruguay
Andrea	Cdad. Mendoza	Mendoza	Argentina
Marlon	Sao Paulo	Sao Paulo	Brasil
Joao	Rio de Janeiro	Rio de Janeiro	Brasil
Andrés	Bariloche	Río Negro	Argentina
Mariano	Miami	Florida	USA

```
SELECT
  Country,
  State,
  COUNT(*) AS HowMany
FROM WorldWideFriends
GROUP BY Country, State;
```

Country	State	HowMany
Argentina	Mendoza	2
Argentina	Río Negro	1
Venezuela	Distrito Capital	1
Colombia	Antioquía	1
Colombia	Cundinamarca	2
Brasil	Rio de Janeiro	1
Brasil	Sao Paulo	1
USA	Kentucky	1
USA	Florida	1
México	Guerrero	1
Uruguay	Canelones	1

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?

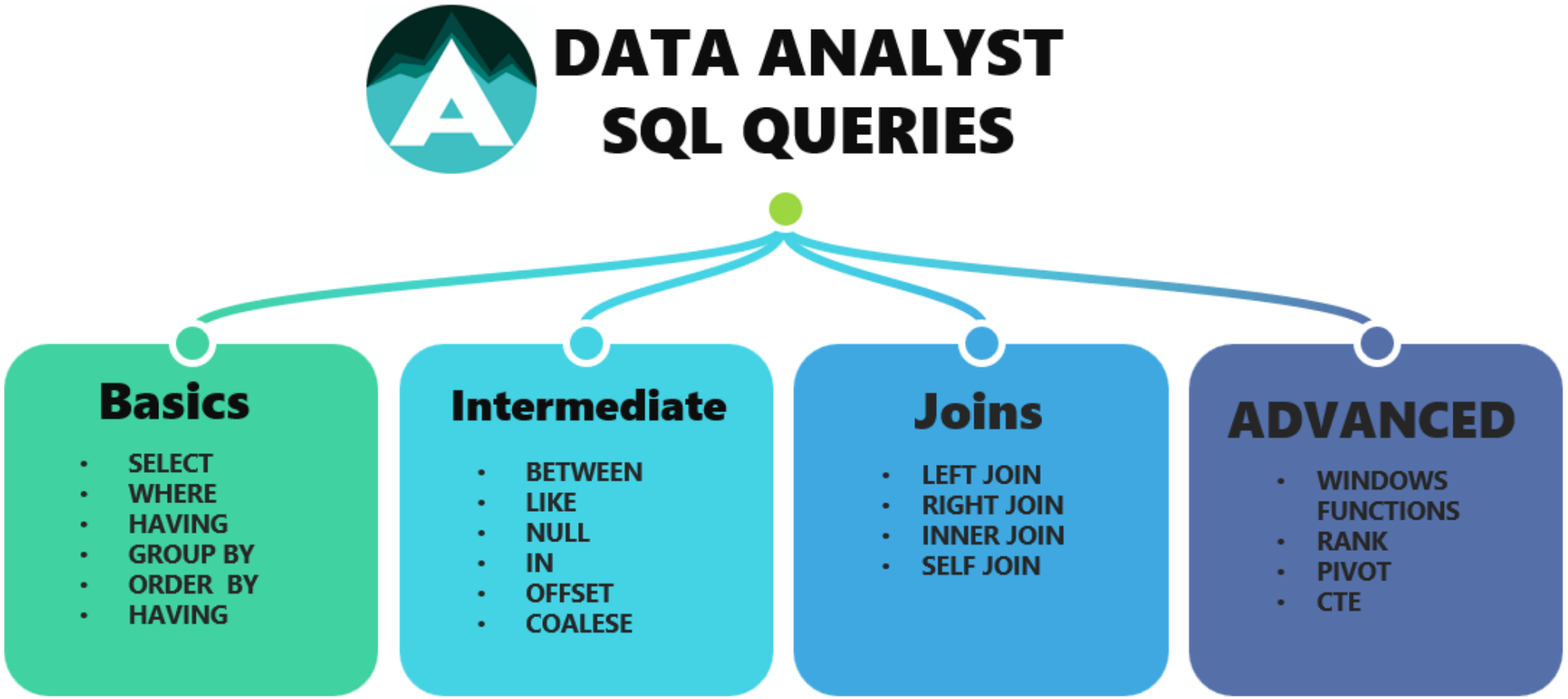
```
USE sql_store;
WITH order_totals AS (
    SELECT
        order_id,
        SUM(quantity * unit_price) AS total_amount
    FROM
        order_items
    GROUP BY
        order_id
)
SELECT
    order_id,
    total_amount
FROM
    order_totals
ORDER BY
    total_amount DESC
LIMIT 1;
```

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.

```
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;
```

Summary


DATA ANALYST
SQL QUERIES
Basics
SELECT
WHERE
HAVING
GROUP BY
ORDER BY
HAVING
Intermediate
BETWEEN
LIKE
NULL
IN
OFFSET
COALESE
Joins
LEFT JOIN
RIGHT JOIN
INNER JOIN
SELF JOIN
ADVANCED
WINDOWS FUNCTIONS
RANK
PIVOT
CTE

RECURSIVE IN SQL

```sql
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(100),
    manager_id INT -- NULL nếu là CEO
);

INSERT INTO employees (employee_id, employee_name, manager_id) VALUES
(1, 'Alice (CEO)', NULL),
(2, 'Bob', 1),
(3, 'Charlie', NULL),
(4, 'David', 2),
(5, 'Eve', 2),
(6, 'Frank', 3);
```

employee_id	employee_name	manager_id
1	Alice (CEO)	NULL
2	Bob	1
4	David	2
5	Eve	2

```sql
WITH RECURSIVE Subordinates AS (
    -- Phần Neo (Anchor Member)
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE employee_name = 'Alice (CEO)'

    UNION ALL

    -- Phần Đệ quy (Recursive Member)
    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates;
```

```sql
WITH RECURSIVE Subordinates AS (
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE manager_id IS NULL

    UNION ALL

    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.employee_id = s.manager_id
)
SELECT * FROM Subordinates;
```

```sql
WITH RECURSIVE Subordinates AS (
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE employee_name = 'Alice (CEO)'

    UNION ALL

    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates WHERE employee_id != (SELECT employee_id FROM
                    employees WHERE employee_name = 'Alice (CEO)');
```

```sql
WITH RECURSIVE Subordinates AS (
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE manager_id = 1 -- Giả sử employee_id của Alice là 1

    UNION ALL

    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates;
```

```sql
WITH RECURSIVE Subordinates AS (
    -- Phần Neo (Anchor Member)
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE employee_name = 'Alice (CEO)'

    UNION ALL

    -- Phần Đệ quy (Recursive Member)
    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates;
```

Lần lặp 1: Xử lý Phần Neo (Anchor Member)

- Truy vấn:

```
SQL
```

```
SELECT employee_id, employee_name, manager_id
FROM employees
WHERE employee_name = 'Alice (CEO)';
```

T_0 = { (1, 'Alice (CEO)', NULL) }

Lần lặp 2: Xử lý Phần Đệ quy (Recursive Member)

- Truy vấn:

```
SQL
```

```
SELECT e.employee_id, e.employee_name, e.manager_id
FROM employees e
INNER JOIN Subordinates s ON e.manager_id = s.employee_id;
```

Tại lần lặp này, Subordinates (s) chứa kết quả của lần lặp trước đó (T_0).

T_1 = { (2, 'Bob', 1) }

Lần lặp 3: Xử lý Phần Đệ quy (Recursive Member)

- **Truy vấn:** Tương tự như trên.
  `Subordinates` (`s`) bây giờ *chứa kết quả của lần lặp trước đó* (`T_1`).
- **Thực hiện JOIN:**
  - `s` hiện tại là: `(2, 'Bob', 1)`
  - Tìm trong `employees` (`e`) các hàng mà `e.manager_id` bằng `s.employee_id` (là `2`).
  - Các nhân viên có `manager_id = 2` là: David (4) và Eve (5).

T_2 = { (4, 'David', 2), (5, 'Eve', 2) }

Lần lặp 4: Xử lý Phần Đệ quy (Recursive Member)

- **Truy vấn:** Tương tự như trên.
  `Subordinates` (`s`) bây giờ *chứa kết quả của lần lặp trước đó* (`T_2`).
- **Thực hiện JOIN:**
  - `s` hiện tại là: `(4, 'David', 2)`, `(5, 'Eve', 2)`
  - Tìm trong `employees` (`e`) các hàng mà `e.manager_id` bằng `s.employee_id` (là `4` hoặc `5`).
  - Trong dataset của chúng ta, không có nhân viên nào có `manager_id` là 4 hoặc 5.

T_3 = {0}

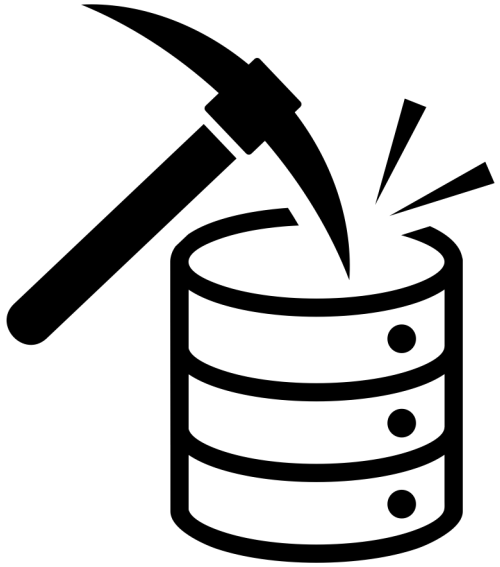
CTE `Subordinates` cuối cùng sẽ chứa tất cả các hàng được thu thập từ phần neo và các lần lặp của phần đệ quy:

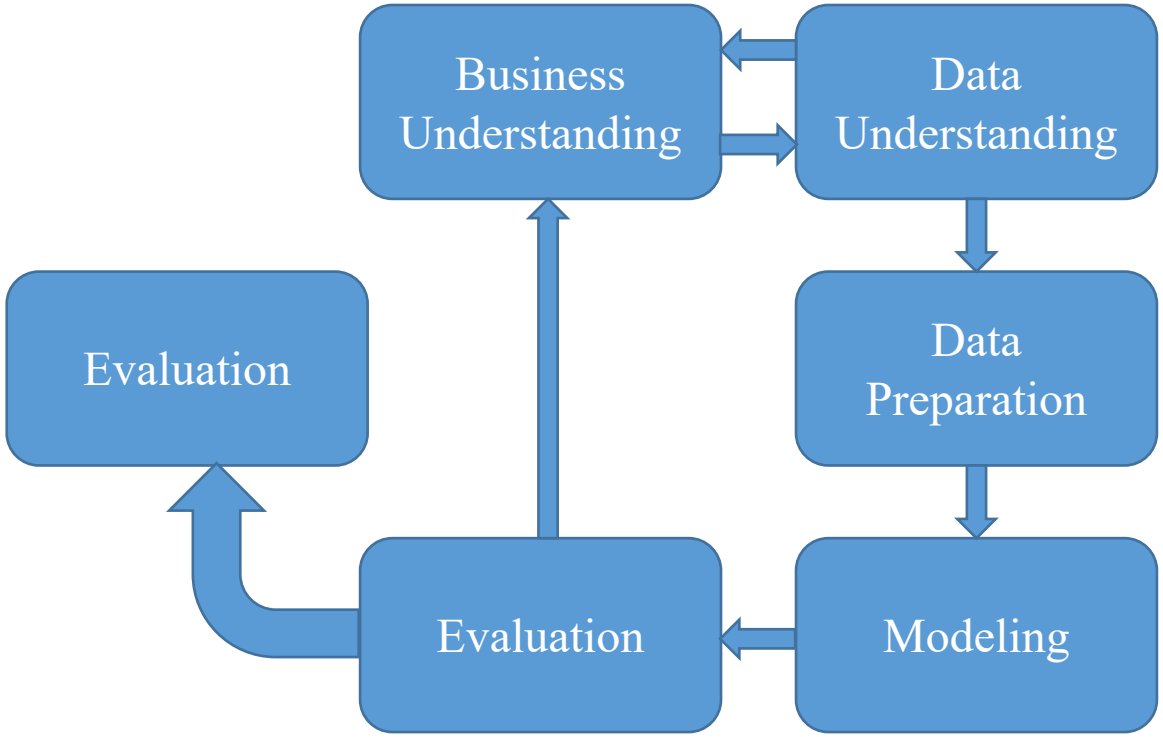
- Từ lần 1 (Neo): `(1, 'Alice (CEO)', NULL)`
- Từ lần 2: `(2, 'Bob', 1)`
- Từ lần 3: `(4, 'David', 2)`, `(5, 'Eve', 2)`

employee_id	employee_name	manager_id
1	Alice (CEO)	NULL
2	Bob	1
4	David	2
5	Eve	2

Further Reading

Data Mining





Data mining is the process of extracting meaningful patterns, trends, and knowledge from large datasets using statistical, mathematical, and machine learning techniques. It involves analyzing data to uncover hidden insights that can inform decision-making and improve business operations.

Data Mining

Case Study: Data Mining on the Sakila Movie Rental Database

Sakila Movie Rental System Overview:

- Sakila is a sample database provided by MySQL
- A simulation of a real movie rental system
- Including data about movies, customers, rental-return-payment transactions

Mining data from Sakila using Advanced SQL:

- Complex joins between multiple related tables
- Use of nested queries and aggregation functions

Apply Data Mining to:

- Analyze movie rental behavior
- Identify loyal customers
- Recommend movies based on rental trends

Tools used:

- MySQL Workbench
- RapidMiner (Data Mining and Visualization)

Data Mining

Sakila Database

Key of Sakila Database:
- 16 tables representing movie rental system
- Realistic structure: movie → inventory → customer → rental → payment
- Rich relationships between tables, including many-to-many associations

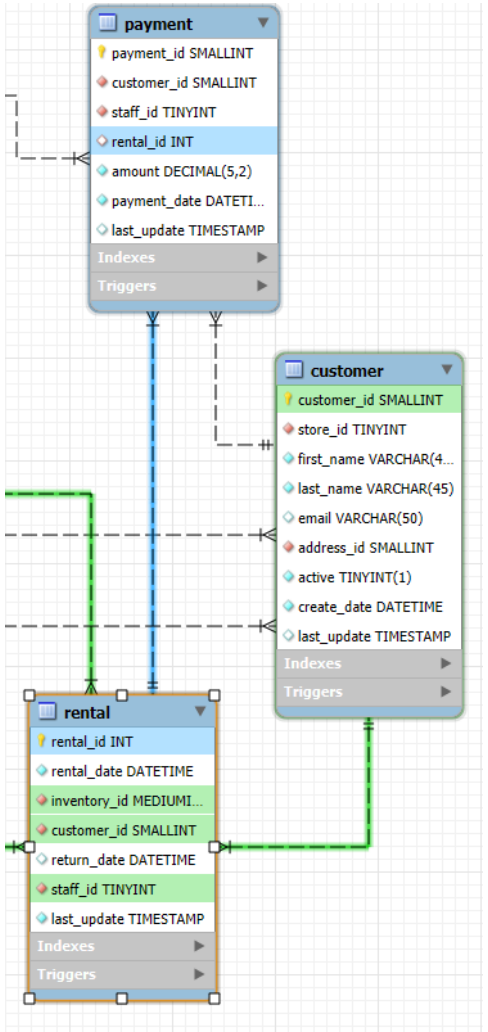
Important tables:
- Customer, Rental, Payment
- Film, Category, Inventory
- Staff, Store

Result Grid | Filter Rows:

stt	TABLE_NAME	TABLE_ROWS
1	actor	200
2	actor_info	NULL
3	address	603
4	category	16
5	city	600
6	country	109
7	customer	599
8	customer_list	NULL
9	film	1000
10	film_actor	5462
11	film_category	1000
12	film_list	NULL
13	film_text	1000
14	inventory	4581
15	language	6
16	nicer_but_sl...	NULL
17	payment	16086
18	rental	16010
19	sales_by_fil...	NULL
20	sales_by_store	NULL
21	staff	2
22	staff_list	NULL
23	store	2

Data Mining

Exploring customer behavior – Advanced SQL



Top ten customer had the most rental.

```sql
SELECT
    customer.customer_id,
    customer.first_name,
    customer.last_name,
    rental_stats.rental_count
FROM customer
JOIN (
    SELECT customer_id, COUNT(*) AS rental_count
    FROM rental
    GROUP BY customer_id
) AS rental_stats
ON customer.customer_id = rental_stats.customer_id
ORDER BY rental_stats.rental_count DESC
LIMIT 10;
```

customer_id	first_name	last_name	rental_count
148	ELEANOR	HUNT	46
526	KARL	SEAL	45
144	CLARA	SHAW	42
236	MARCIA	DEAN	42
75	TAMMY	SANDERS	41
197	SUE	PETERS	40
469	WESLEY	BULL	40
137	RHONDA	KENNEDY	39
178	MARION	SNYDER	39
468	TIM	CARY	39

Data Mining

Exploring customer behavior - Advanced SQL

```sql
DELIMITER $$

CREATE PROCEDURE top_customers(IN topN INT)
BEGIN
    SELECT customer_id, COUNT(*) AS rental_count
    FROM rental
    GROUP BY customer_id
    ORDER BY rental_count DESC
    LIMIT topN;
END $$

DELIMITER ;
```

Create a procedure named top_customers(n) to return the top n most rented customers.

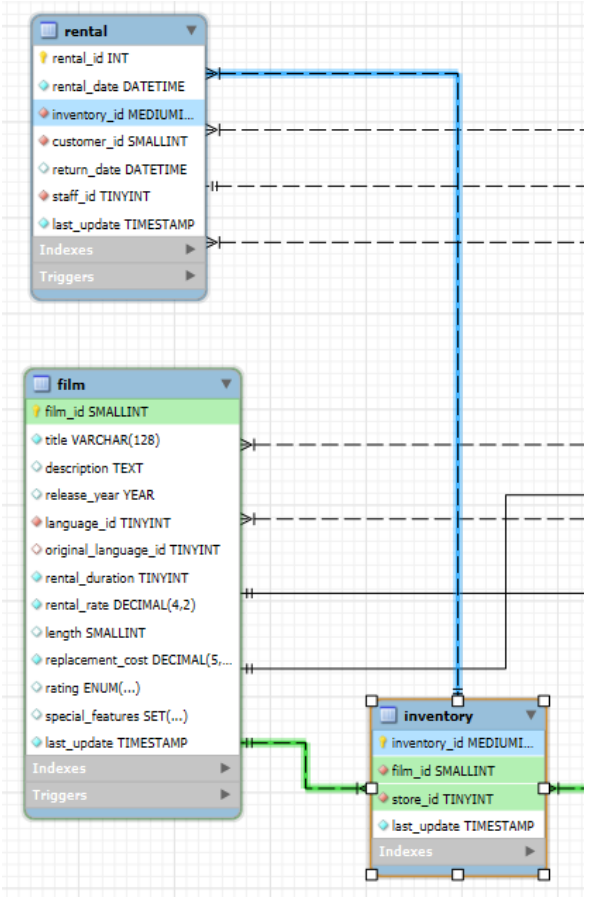
```sql
CREATE TABLE payment_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id SMALLINT,
    amount DECIMAL(5,2),
    payment_date DATETIME
);

DELIMITER $$

CREATE TRIGGER log_payment
AFTER INSERT ON payment
FOR EACH ROW
BEGIN
    INSERT INTO payment_log (customer_id, amount, payment_date)
    VALUES (NEW.customer_id, NEW.amount, NEW.payment_date);
END $$

DELIMITER ;
```

Log every time there is a payment to track active customers.

Data Mining

Exploring movie content – Advanced SQL

Top ten most rented movies.



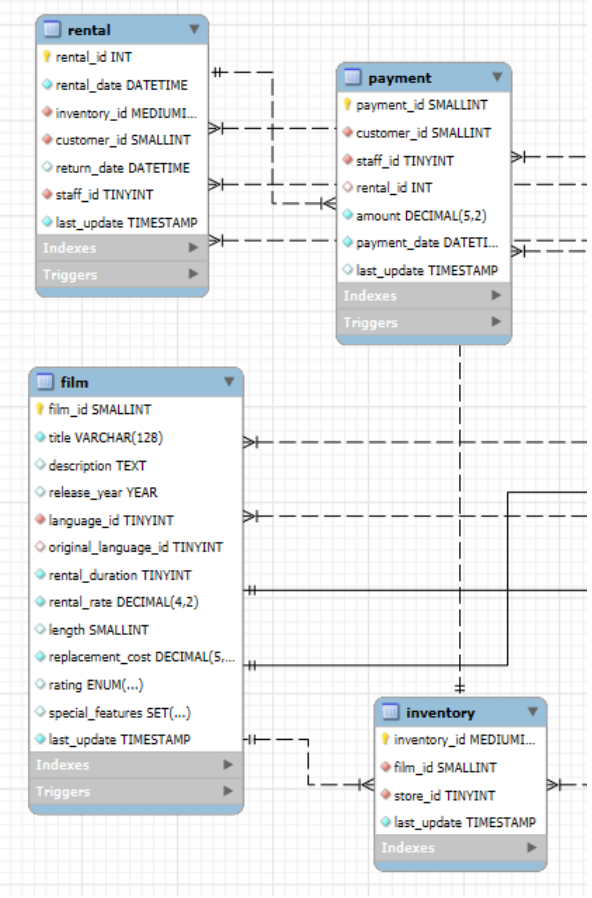

```sql
SELECT
    f.title,
    rental_counts.rental_count
FROM film f
JOIN (
    SELECT
        i.film_id,
        COUNT(*) AS rental_count
    FROM inventory i
    JOIN rental r ON i.inventory_id = r.inventory_id
    GROUP BY i.film_id
) AS rental_counts ON f.film_id = rental_counts.film_id
ORDER BY rental_counts.rental_count DESC
LIMIT 10;
```

Result Grid | Filter Rows:

title	rental_count
BUCKET BROTHERHOOD	34
ROCKETEER MOTHER	33
FORWARD TEMPLE	32
GRIT CLOCKWORK	32
JUGGLER HARDLY	32
RIDGEMONT SUBMARINE	32
SCALAWAG DUCK	32
APACHE DIVINE	31
GOODFELLAS SALUTE	31
HOBBIT ALIEN	31

Data Mining

Exploring movie content – Advanced SQL



Top ten highest grossing film.

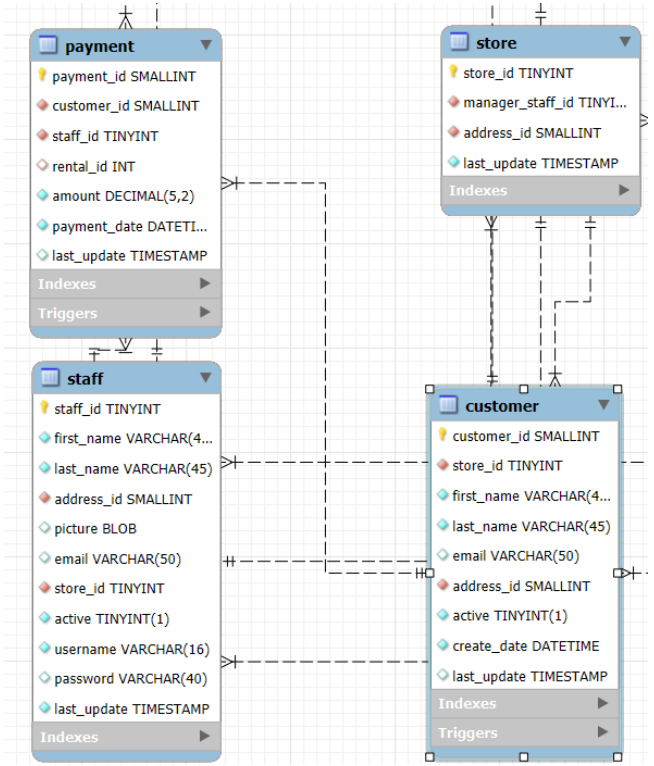
```sql
SELECT
    f.title,
    revenue_stats.total_revenue
FROM film f
JOIN (
    SELECT
        i.film_id,
        SUM(p.amount) AS total_revenue
    FROM inventory i
    JOIN rental r ON i.inventory_id = r.inventory_id
    JOIN payment p ON r.rental_id = p.rental_id
    GROUP BY i.film_id
) AS revenue_stats ON f.film_id = revenue_stats.film_id
ORDER BY revenue_stats.total_revenue DESC
LIMIT 10;
```

Result Grid | Filter Rows:

title	total_revenue
TELEGRAPH VOYAGE	231.73
WIFE TURN	223.69
ZORRO ARK	214.69
GOODFELLAS SALUTE	209.69
SATURDAY LAMBS	204.72
TITANS JERK	201.71
TORQUE BOUND	198.72
HARRY IDAHO	195.70
INNOCENT USUAL	191.74
HUSTLER PARTY	190.78

Data Mining

Exploring store and staff performance – Advanced SQL



Top ten monthly revenue by the store.

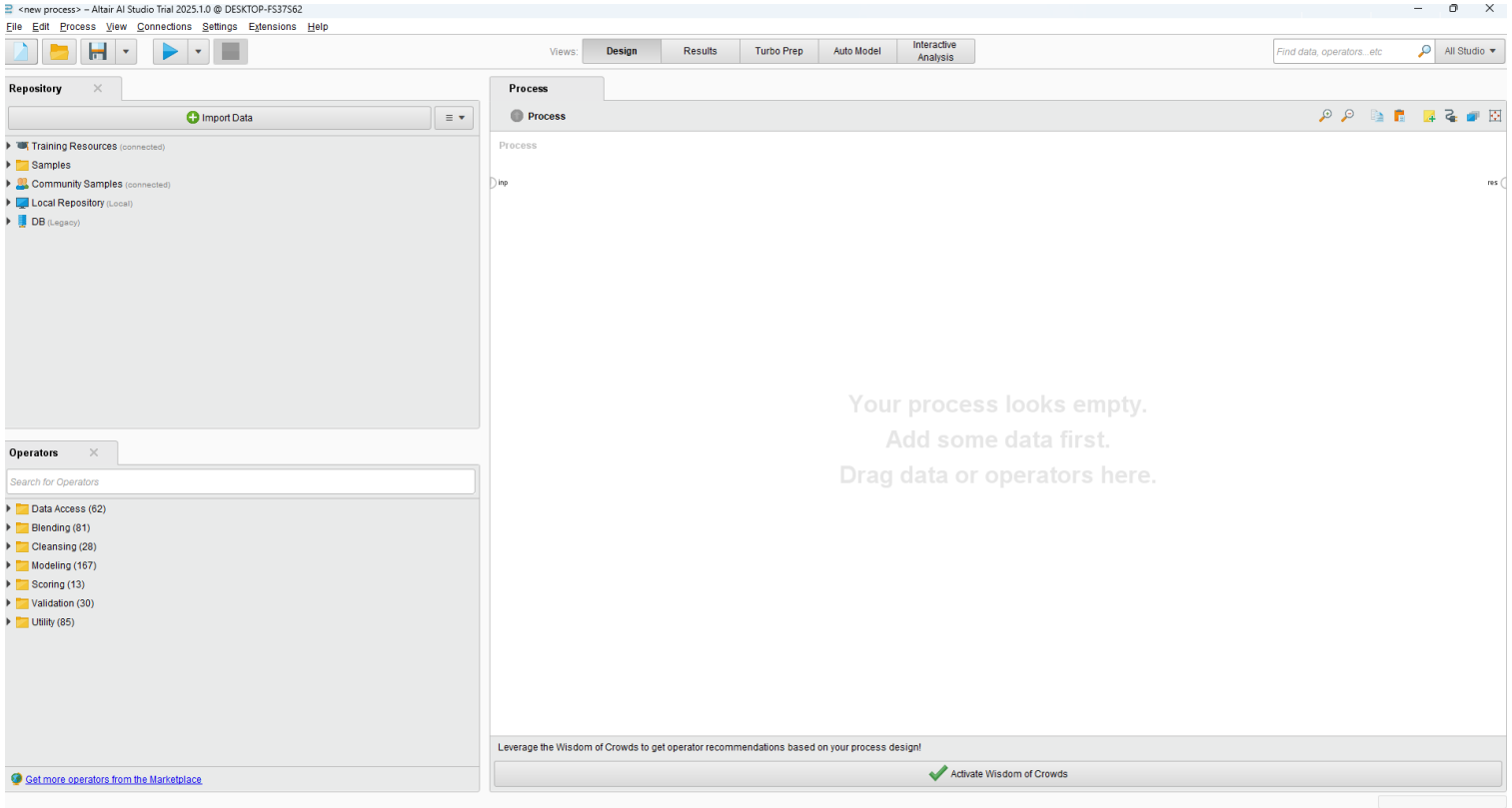
```sql
SELECT
    store_id,
    DATE_FORMAT(payment_date, '%Y-%m') AS month,
    SUM(amount) AS monthly_revenue
FROM customer
JOIN payment ON customer.customer_id = payment.customer_id
GROUP BY store_id, month
ORDER BY monthly_revenue DESC
LIMIT 10;
```

Result Grid | Filter Rows:

store_id	month	monthly_revenue
1	2005-07	15735.23
1	2005-08	13136.09
2	2005-07	12633.68
2	2005-08	10934.05
1	2005-06	5148.57
2	2005-06	4481.32
1	2005-05	2694.62
2	2005-05	2128.82
1	2006-02	283.02
2	2006-02	231.16

Data Mining

Introduction to Rapid Miner – Data Mining tool



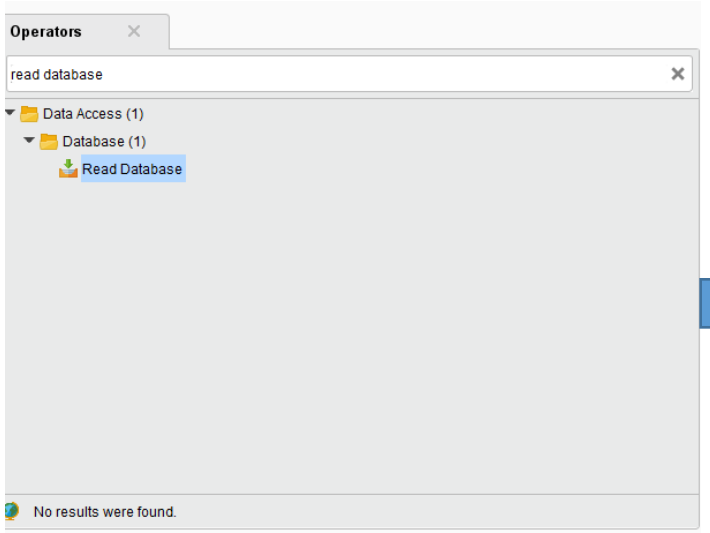
Steps:

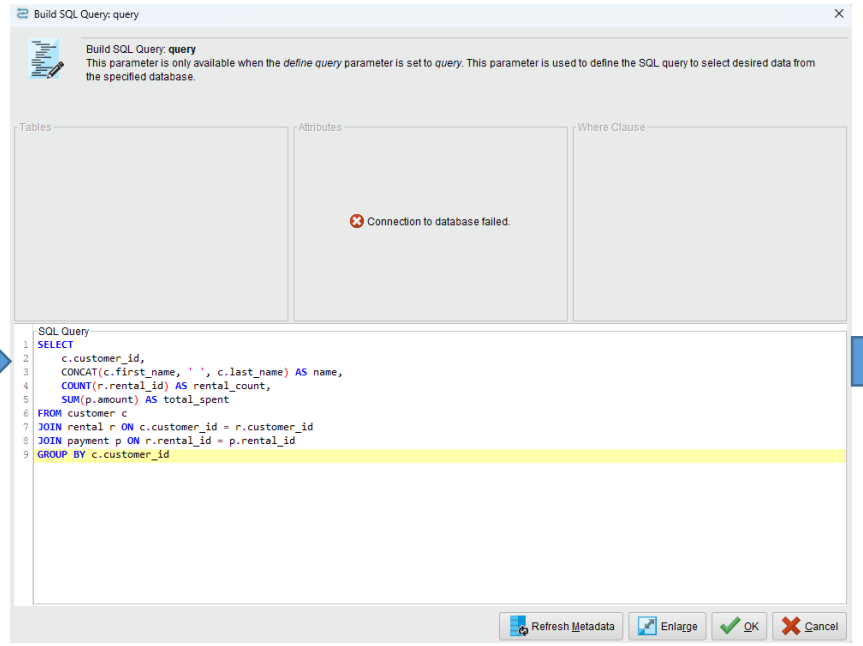
- Read and connect Database
- Write SQL and Visualize results
- Data Mining two case study use Decision Tree and K-Means clustering

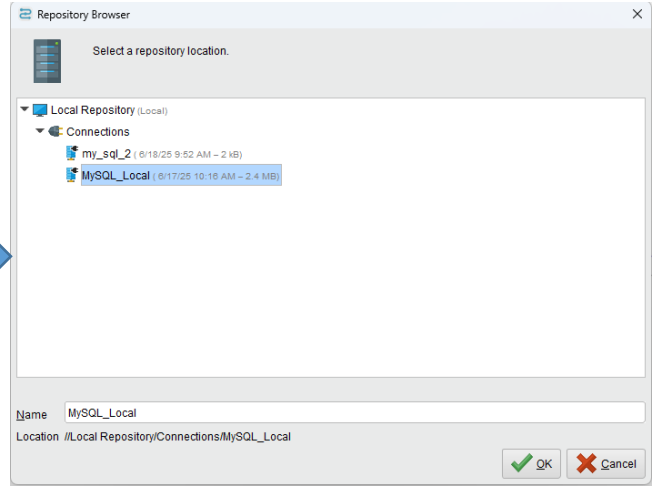
Rapid Miner is a comprehensive data science platform used for data preparation, machine learning, and predictive analytics

Data Mining

SQL with Rapid Miner







Use operator Read Database

Write SQL in operator

Connect with Mysql Local

Data Mining

View data in tables and graphs

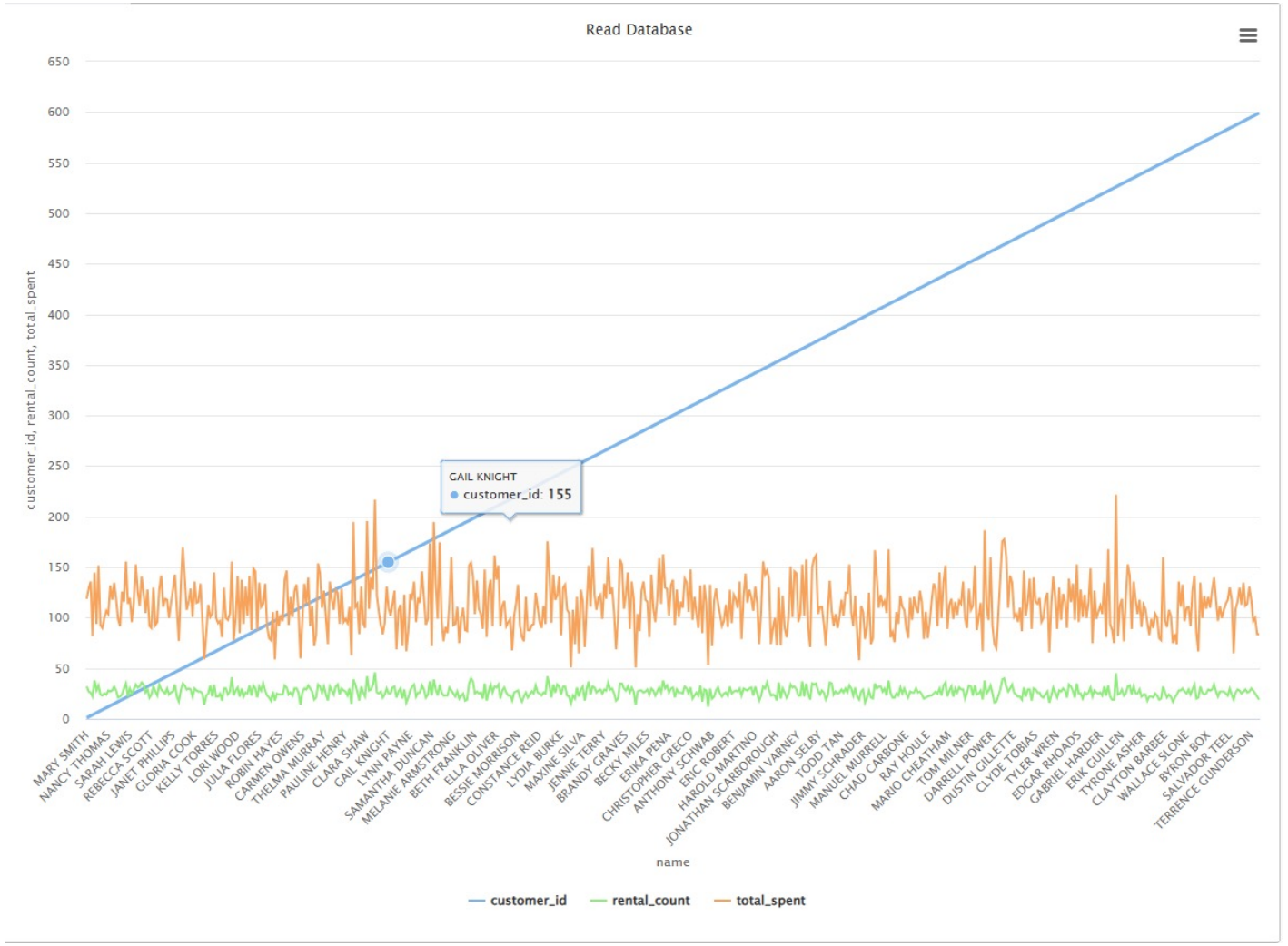
Result History | ExampleSet (Read Database)

Data | Statistics | Visualizations | Annotations

Open in: Turbo Prep | Auto Model | Interactive Analysis

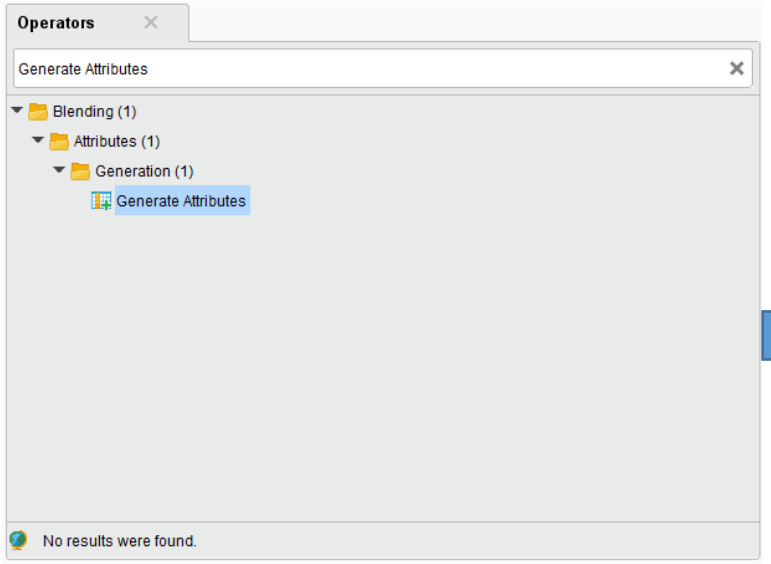
Row No.	customer_id	name	rental_count	total_spent
1	1	MARY SMITH	32	118.680
2	2	PATRICIA JO...	27	128.730
3	3	LINDA WILLI...	26	135.740
4	4	BARBARA JO...	22	81.780
5	5	ELIZABETH B...	38	144.620
6	6	JENNIFER D...	28	93.720
7	7	MARIA MILLER	33	151.670
8	8	SUSAN WILS...	24	92.760
9	9	MARGARET ...	23	89.770
10	10	DOROTHY T...	25	99.750
11	11	LISA ANDER...	24	106.760
12	12	NANCY THO...	28	103.720
13	13	KAREN JACK...	27	131.730
14	14	BETTY WHITE	28	117.720
15	15	HELEN HAR...	32	134.680
16	16	SANDRA MA...	28	118.720
17	17	DONNA THO...	21	98.790
18	18	CAROL GAR...	22	91.780
19	19	RUTH MARTI...	24	125.760
20	20	SHARON RO...	30	115.700
21	21	MICHELLE C...	35	155.650
22	22	LAURA ROD...	22	113.780
23	23	SARAH LEWIS	30	119.700
24	24	KIMBERLY L...	25	95.750
25	25	DEBORAH W...	29	115.710

ExampleSet (599 examples,0 special attributes,4 regular attributes)

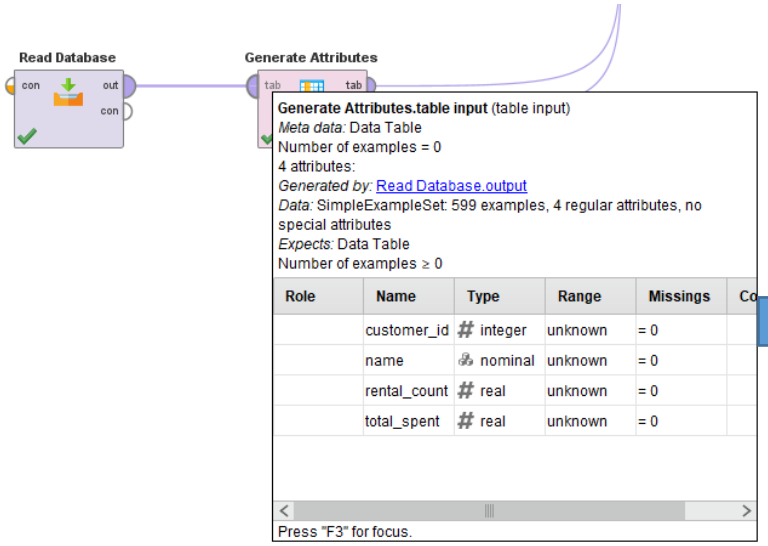


Data Mining

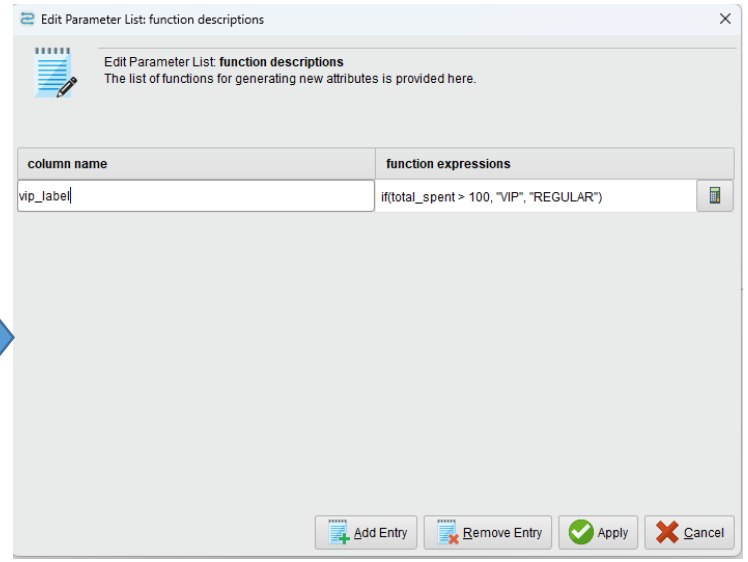
Classification with Decision Tree



Use operator Generate Attributes



Use output of
Read Database operator



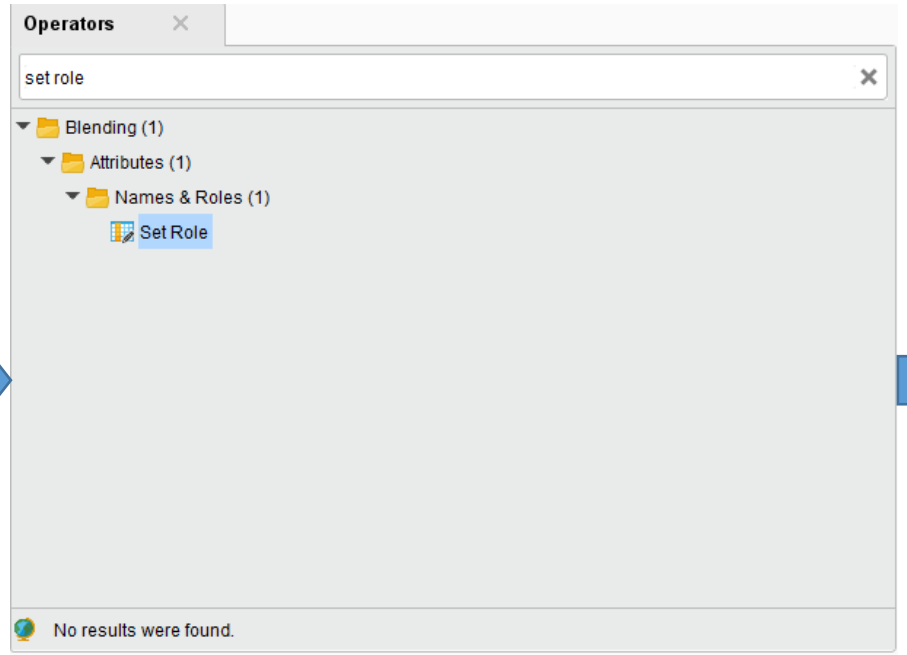
Create vip_label column
with expressions to label

Data Mining

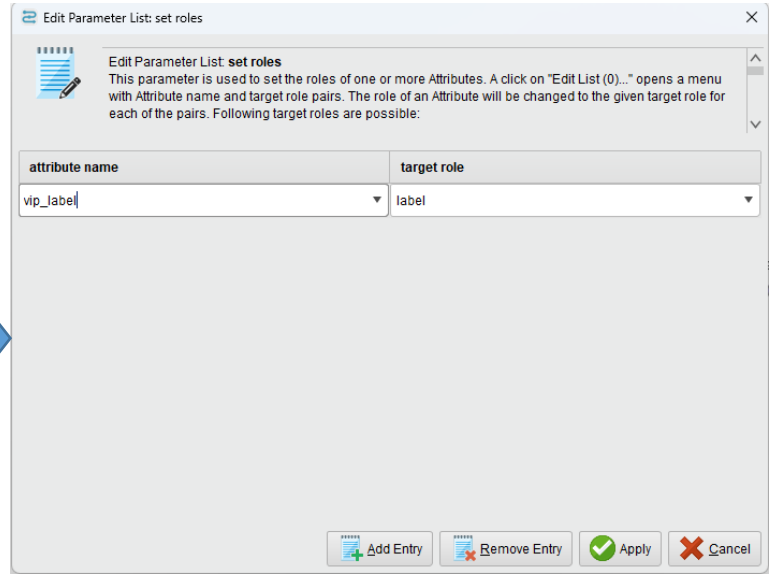
Classification with Decision Tree

Row No.	customer_id	name	rental_count	total_spent	vip_label
1	1	MARY SMITH	32	118.680	VIP
2	2	PATRICIA JO...	27	128.730	VIP
3	3	LINDA WILLI...	26	135.740	VIP
4	4	BARBARA JO...	22	81.780	REGULAR
5	5	ELIZABETH B...	38	144.620	VIP
6	6	JENNIFER D...	28	93.720	REGULAR
7	7	MARIA MILLER	33	151.670	VIP
8	8	SUSAN WILS...	24	92.760	REGULAR
9	9	MARGARET ...	23	89.770	REGULAR
10	10	DOROTHY T...	25	99.750	REGULAR
11	11	LISA ANDER...	24	106.760	VIP
12	12	NANCY THO...	28	103.720	VIP
13	13	KAREN JACK...	27	131.730	VIP
14	14	BETTY WHITE	28	117.720	VIP
15	15	HELEN HAR...	32	134.680	VIP
16	16	SANDRA MA...	28	118.720	VIP
17	17	DONNA THO...	21	98.790	REGULAR
18	18	CAROL GAR...	22	91.780	REGULAR
19	19	RUTH MARTI...	24	125.760	VIP
20	20	SHARON RO...	30	115.700	VIP
21	21	MICHELLE C...	35	155.650	VIP
22	22	LAURA ROD...	22	113.780	VIP
23	23	SARAH LEWIS	30	119.700	VIP
24	24	KIMBERLY L...	25	95.750	REGULAR
25	25	DEBORAH W...	29	115.710	VIP

See the label in result tab



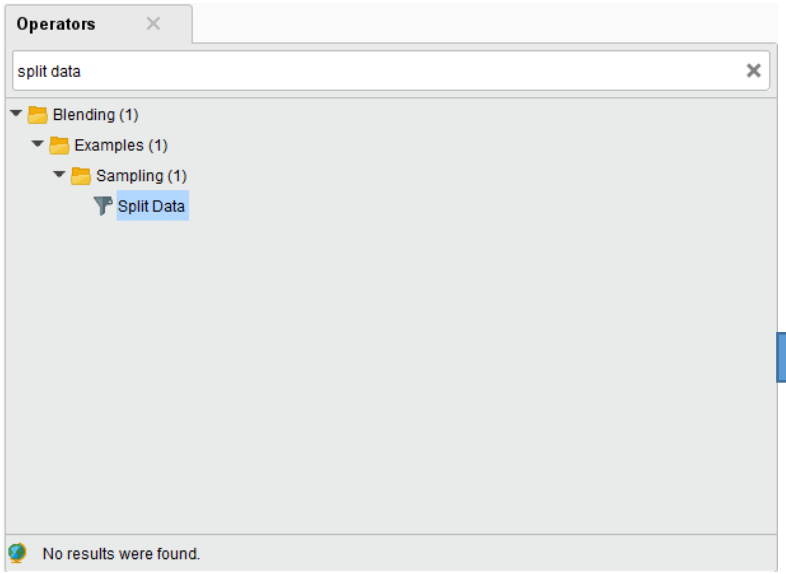
Use Set Role operator to set the vip_lable to turn vip_label column into real label

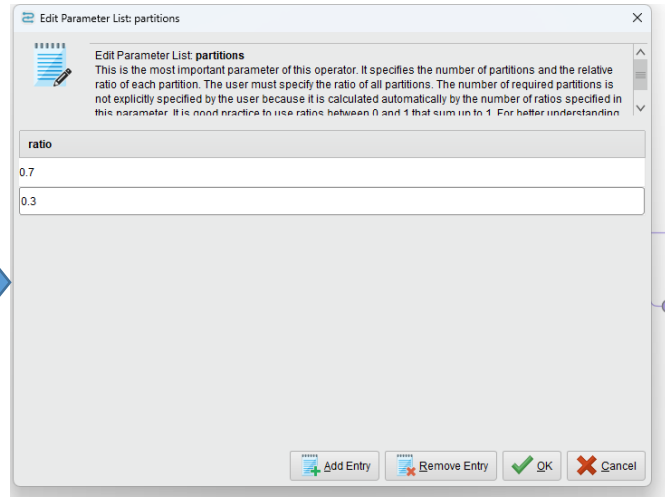


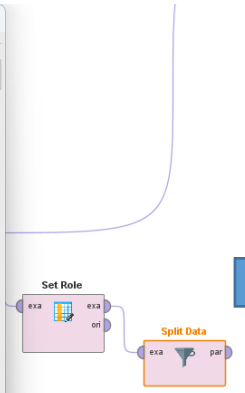
Set param in Set Role operator

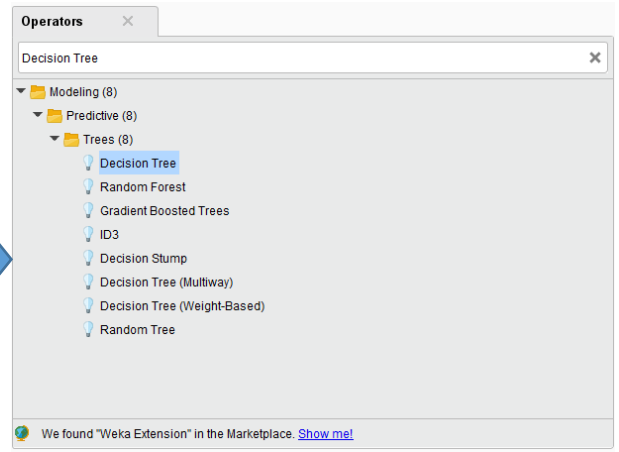
Data Mining

Classification with Decision Tree









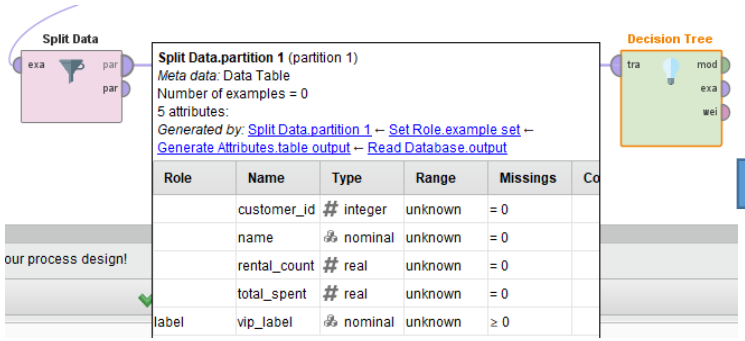
Use Split Data Operator

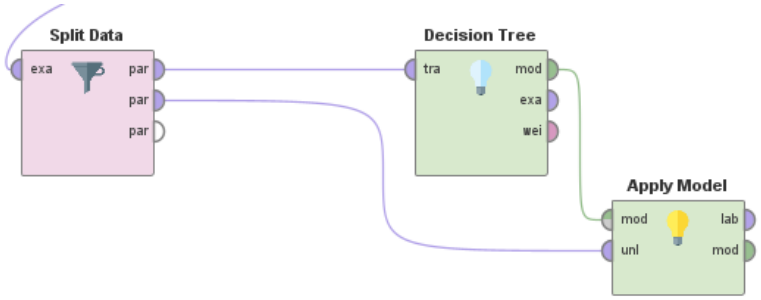
Add ratio param and connect Split Data with Set Role

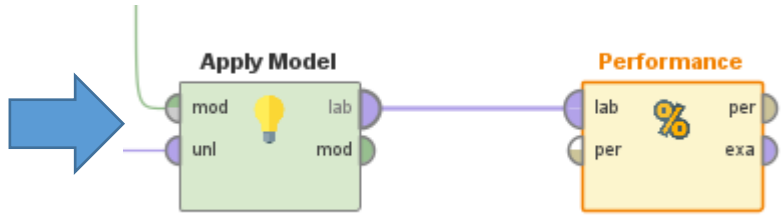
Use Decision Tree operator

Data Mining

Classification with Decision Tree







Connect Split Data partition 1 with Decision Tree

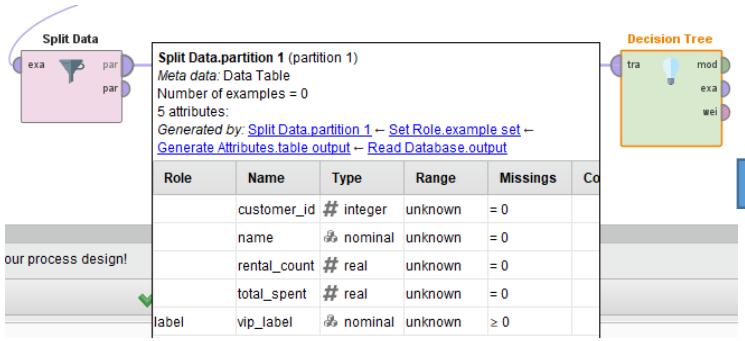
Use Apply Model operator:

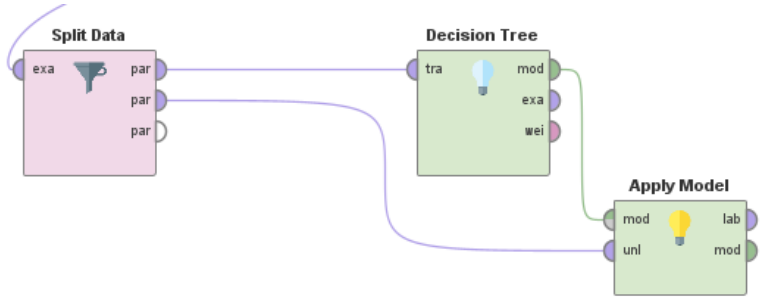
- Connect model of Decision to model of Apply Model
- Connect partition 2 with unlabelled data of Apply Model

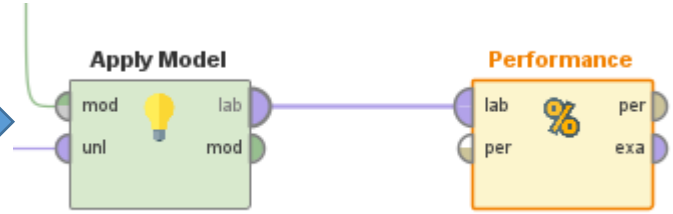
Connect Performance label to label of Apply Model

Data Mining

Classification with Decision Tree







Connect Split Data partition 1 with Decision Tree

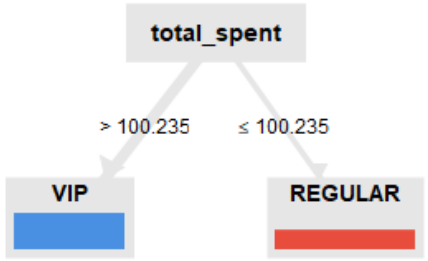
Use Apply Model operator:

- Connect model of Decision to model of Apply Model
- Connect partition 2 with unlabelled data of Apply Model

Connect Performance label to label of Apply Model

Data Mining

Observe the result.



Decision Tree Result

accuracy: 100.00%

	true VIP	true REGULAR
pred. VIP	118	0
pred. REGULAR	0	61
class recall	100.00%	100.00%

Performance result

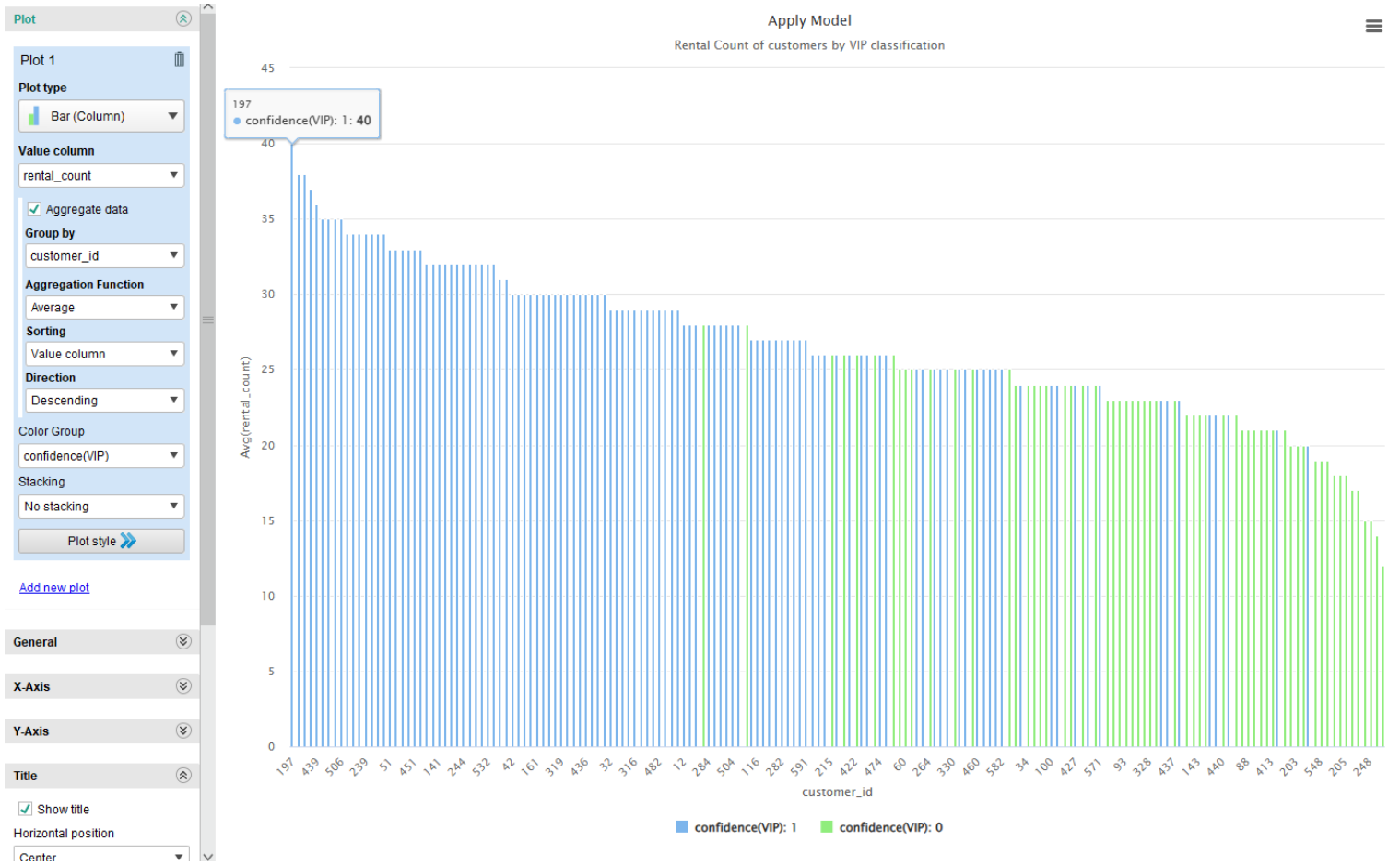
Row No.	vip_label	prediction(vi...	confidence(...	confidence(...	customer_id	name	rental_count	total_spent
1	VIP	VIP	1	0	2	PATRICIA JO...	27	128.730
2	VIP	VIP	1	0	3	LINDA WILLI...	26	135.740
3	REGULAR	REGULAR	0	1	8	SUSAN WILS...	24	92.760
4	REGULAR	REGULAR	0	1	9	MARGARET ...	23	89.770
5	VIP	VIP	1	0	12	NANCY THO...	28	103.720
6	VIP	VIP	1	0	16	SANDRA MA...	28	118.720
7	REGULAR	REGULAR	0	1	18	CAROL GAR...	22	91.780
8	VIP	VIP	1	0	19	RUTH MARTI...	24	125.760
9	VIP	VIP	1	0	21	MICHELLE C...	35	155.650
10	REGULAR	REGULAR	0	1	24	KIMBERLY L...	25	95.750
11	VIP	VIP	1	0	26	JESSICA HALL	34	152.660
12	VIP	VIP	1	0	28	CYNTHIA YO...	32	111.680
13	VIP	VIP	1	0	30	MELISSA KING	34	123.660
14	VIP	VIP	1	0	32	AMY LOPEZ	29	127.710
15	REGULAR	REGULAR	0	1	33	ANNA HILL	21	91.790
16	REGULAR	REGULAR	0	1	34	REBECCA S...	24	89.760
17	VIP	VIP	1	0	42	CAROLYN P...	30	117.700
18	REGULAR	REGULAR	0	1	43	CHRISTINE ...	24	99.760
19	VIP	VIP	1	0	50	DIANE COLLI...	35	169.650
20	VIP	VIP	1	0	51	ALICE STEW...	33	138.670
21	REGULAR	REGULAR	0	1	60	MILDRED BAI...	25	98.750
22	REGULAR	REGULAR	0	1	61	KATHERINE ...	14	58.860
23	REGULAR	REGULAR	0	1	62	JOAN COOP...	23	84.770
24	REGULAR	REGULAR	0	1	69	JUDY GRAY	25	96.750
25	REGULAR	REGULAR	0	1	73	BEVERLY BR...	24	97.760

Predict table result

Data Mining

Observe the result.

- VIP group is smaller but has high rental rate ⇒ is the target group that needs priority care/promotion.
- Non-VIP group can be new or infrequent guests ⇒ may need re-renting incentive strategy.



Visualize predict result