De SQL à Pandas

Phuong Nguyen
@fphuongnguyen

Vous connaissez SQL et souhaitez monter en compétences avec Pandas?

Dans ce carrousel, je vous partage comment passer de manière interchangeable entre les deux langages.

Pour chaque commande, je vous montre le code en SQL, et l'équivalent en Python Pandas.

C'est parti! 🚀

Phuong Nguyen
@fphuongnguyen

1. Filtrage

```
--product table
product_id  product_name      price
    1           computer        800
    2           printer         600
    3           tablet          400
    4           desk            100
    5           chair           50


-- 1/ SQL query
SELECT
  product_id,
  product_name,
  price
FROM product
WHERE price > 500

-- 2a/ Pandas: basic filtering
product[product["price"] > 500]

-- 2b/ Pandas: using loc
product.loc[product["price"] > 500]

-- 2c/ Pandas: using query
product.query("price > 500")


-- Output
product_id  product_name    price
    1         computer      800.0
    2         printer       600.0
```

Phuong Nguyen
@fphuongnguyen

2. Filtrage sur chaîne de caractères

SQL

```
--product table
>>> df
product_id  product_code    price
  1          PH.CHAR.A      800.0
  2          TK.TAB.B       600.0
  3          PH.COM.C       400.0
  4          TK.FAC.A       100.0
  5          TK.COM.C       50.0

-- 1/ Filtering products by a list
SELECT
  *
FROM df
WHERE product_code IN ('PH.CHAR.A', 'PH.COM.C')

product_id  product_code    price
  1          PH.CHAR.A      800.0
  3          PH.COM.C       400.0

-- 2/ Filtering product codes that contain 'COM'
SELECT
  *
FROM df
WHERE product_code LIKE '%COM%'

product_id  product_code    price
  3          PH.COM.C       400.0
  5          TK.COM.C       50.0

-- 3/ Filtering product codes that start with 'PH'
SELECT
  *
FROM df
WHERE product_code LIKE 'PH%'

product_id  product_code    price
  1          PH.CHAR.A      800.0
  3          PH.COM.C       400.0

-- 4/ Filtering product codes that end with 'A'
SELECT
  *
FROM df
WHERE product_code LIKE '%A'

product_id  product_code    price
  1          PH.CHAR.A      800.0
  4          TK.FAC.A       100.0
```

pandas

```
# product table
>>> df
product_id  product_code    price
  1          PH.CHAR.A      800.0
  2          TK.TAB.B       600.0
  3          PH.COM.C       400.0
  4          TK.FAC.A       100.0
  5          TK.COM.C       50.0

# 1/ Filtering products by a list

df[df["product_code"].isin(['PH.CHAR.A', 'PH.COM.C'])]

product_id  product_code    price
  1          PH.CHAR.A      800.0
  3          PH.COM.C       400.0


# 2/ Filtering product codes that contain 'COM'

df[df["product_code"].str.contains("COM")]

product_id  product_code    price
  3          PH.COM.C       400.0
  5          TK.COM.C       50.0


# 3/ Filtering product codes that start with 'PH'

df[df["product_code"].str.startswith("PH")]

product_id  product_code    price
  1          PH.CHAR.A      800.0
  3          PH.COM.C       400.0


# 4/ Filtering product codes that end with 'A'

df[df["product_code"].str.endswith("A")]

product_id  product_code    price
  1          PH.CHAR.A      800.0
  4          TK.FAC.A       100.0
```

Phuong Nguyen
@fphuongnguyen

3. COUNT & DISTINCT

SQL

```sql
-- orders table
>>> df
order_id  product      amount
  1        tea          800.0
  2        chocolate    600.0
  3        coffee       400.0
  4        tea          100.0
  5        chocolate    50.0

-- 1/ Distinct values
SELECT
  DISTINCT product
FROM df
-- Output
  product
    tea
    chocolate
    coffee

-- 2/ Count of distinct values
SELECT
  COUNT(DISTINCT product) AS product_count
FROM df
-- Output
product_count
    3

-- 3/ Count of total values
SELECT
  COUNT(order_id) AS order_count
FROM df
-- Output
order_count
  5
```

pandas

```python
# orders table
>>> df
order_id  product      amount
  1        tea          800.0
  2        chocolate    600.0
  3        coffee       400.0
  4        tea          100.0
  5        chocolate    50.0

# 1/ Distinct values
>>> df["product"].unique()

# Output
array(['tea', 'chocolate', 'coffee'], dtype=object)


# 2/ Count of distinct values
>>> df["product"].nunique()

# Output
3


# 3/ Count of total values
>>> df["order_id"].size

# Output
5
```

Phuong Nguyen
@fphuongnguyen

4. GROUP BY

```
-- orders table
>>> df
  order_id  customer_id  product_category  amount
  122154          1            tea           4.5
  122453          2            chocolate     5.0
  122476          1            coffee        4.0
  122783          3            tea           6.0
  122378          1            chocolate     5.0
  122157          2            coffee        5.5

-- 1/ SQL

SELECT
  product_category,
  COUNT(order_id) AS order_count,
  SUM(amount) AS total_amount,
  AVG(amount) AS avg_amount
FROM df
GROUP BY product_category

-- Output
  product_category  order_count  total_amount  avg_amount
      tea              2             10.5        5.25
      chocolate        2             10.0        5.00
      coffee           2             9.5         4.75


-- 2/ Pandas

df.groupby(["product_category"]).agg({
                                    "order_id": "count",
                                    "amount": ["sum", "mean"]
                                    })

-- Output
  product_category   order_id      amount
                      count       sum     mean

      chocolate         2          10.0   5.00
      coffee            2          9.5    4.75
      tea               2          10.5   5.25
```

Phuong Nguyen
@fphuongnguyen

5. ORDER BY

```
-- orders table
>>> df
  order_id  customer_id  product_category  amount
  122154          1            tea           4.5
  122453          2            chocolate     5.0
  122476          1            coffee        4.0
  122783          3            tea           6.0
  122378          1            chocolate     5.0
  122157          2            coffee        5.5

-- 1. ORDER BY A SINGLE COLUMN
-- 1a/ SQL
SELECT
  customer_id
FROM df
ORDER BY customer_id

-- 1b/ Pandas
df[["customer_id"]].sort_values(by=['customer_id'])
-- Output
customer_id
    1
    1
    1
    2
    2
    3


-- 2. ORDER BY MULTIPLE COLUMNS
-- 2a/ SQL
SELECT
  customer_id,
  product_category
FROM df
ORDER BY customer_id, product_category DESC

-- 2b/ Pandas
df[["customer_id", "product_category"]].sort_values(
                                        by=['customer_id', 'product_category'],
                                        ascending=[True, False])
-- Output
  customer_id  product_category
      1            tea
      1            coffee
      1            chocolate
      2            coffee
      2            chocolate
      3            tea
```

Phuong Nguyen
@fphuongnguyen

6. UNION & UNION ALL

```
>>> customers
customer_id  customer_name
      1          Thomas
      2          Thierry

>>> new_customers
customer_id  customer_name
      1          Thomas
      3          Marc

-- 1. UNION ALL
-- 1a/ SQL
SELECT
  customer_id,
  customer_name
FROM customers
UNION ALL
SELECT
  customer_id,
  customer_name
FROM new_customers
-- 1b/ Pandas
pd.concat([customers, new_customers], ignore_index=True)
-- Output
customer_id  customer_name
      1          Thomas
      2          Thierry
      1          Thomas
      3          Marc

-- 2. UNION
-- 2a/ SQL
SELECT
  customer_id,
  customer_name
FROM customers
UNION
SELECT
  customer_id,
  customer_name
FROM new_customers
-- 2b/ Pandas
pd.concat([customers, new_customers], ignore_index=True).drop_duplicates()
-- Output
customer_id  customer_name
      1          Thomas
      2          Thierry
      3          Marc
```

Phuong Nguyen
@fphuongnguyen

7. INNER JOIN & LEFT JOIN

```
>>> customers
customer_id  customer_name
      1           Thomas
      2           Thierry
      3           Marc

>>> orders
customer_id    product
      1         chocolate
      3         tea

-- 1. INNER JOIN
-- 1a/ SQL
SELECT
  customer_id,
  customer_name,
  product
FROM customers
INNER JOIN orders USING(customer_id)
-- 1b/ Pandas
pd.merge(customers, orders, on="customer_id", how="inner")
-- Output
customer_id  customer_name    product
      1           Thomas       chocolate
      3           Marc         tea

-- 2. LEFT JOIN
-- 2a/ SQL
SELECT
  customer_id,
  customer_name,
  product
FROM customers
LEFT JOIN orders USING(customer_id)
-- 2b/ Pandas
pd.merge(customers, orders, on="customer_id", how="left")
-- Output
customer_id  customer_name    product
      1           Thomas       chocolate
      3           Marc         tea
      2           Thierry      NULL
```

Phuong Nguyen
@fphuongnguyen

8. WINDOW FUNCTIONS: moyenne mobile et total cumulé

```
>>> sales
    date        sales
2024-01-12      500
2024-01-13      605
2024-01-14      340
2024-01-15      509

-- 1. MOVING AVERAGE
-- 1a/ SQL
SELECT
  date,
  sales,
  AVG(sales) OVER (ORDER BY date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg
FROM sales
-- 1b/ Pandas
df["moving_avg"] = df.sales.rolling(window=3, min_periods=1).mean()
-- Output
    date        sales     moving_avg
2024-01-12      500       500.000000
2024-01-13      605       552.500000
2024-01-14      340       481.666667
2024-01-15      509       484.666667

-- 2. RUNNING TOTAL
-- 2a/ SQL
SELECT
  date,
  sales,
  SUM(sales) OVER (ORDER BY date ROWS UNBOUNDED PRECEDING) AS running_total_sales
FROM sales
-- 2b/ Pandas
df["running_total_sales"] = df.sales.cumsum()
-- Output
    date        sales     running_total_sales
2024-01-12      500       500
2024-01-13      605       1105
2024-01-14      340       1445
2024-01-15      509       1954
```

Phuong Nguyen
@fphuongnguyen

9. WINDOW FUNCTIONS avec PARTITIONS: moyenne mobile et total cumulé

```
>>> sales
    date        product        sales
2024-01-12      coffee          789
2024-01-13      tea             605
2024-01-10      coffee          509
2024-01-14      tea             340
2024-01-12      tea             500
2024-01-11      coffee          423

-- 1. MOVING AVERAGE
-- 1a/ SQL
SELECT
  date,
  product,
  sales,
  AVG(sales) OVER (PARTITION BY product ORDER BY date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg
FROM sales
-- 1b/ Pandas
df["moving_avg"] = df.sort_values(['date'])\
                     .groupby('product')['sales']\
                     .transform(lambda x: x.rolling(3, min_periods=1).mean())
df.sort_values(['date', 'product']) -- make output pretty
-- Output
date          product    sales    moving_avg
2024-01-10    coffee      509     509.000000
2024-01-11    coffee      423     466.000000
2024-01-12    coffee      789     573.666667
2024-01-12    tea         500     500.000000
2024-01-13    tea         605     552.500000
2024-01-14    tea         340     481.666667

-- 2. RUNNING TOTAL
-- 2a/ SQL
SELECT
  date,
  product,
  sales,
  SUM(sales) OVER (PARTITION BY product ORDER BY date ROWS UNBOUNDED PRECEDING) AS running_total_sales
FROM sales
-- 2b/ Pandas
df["running_total_sales"] = df.sort_values(['date'])\
                              .groupby('product')['sales']\
                              .cumsum()
df.sort_values(['date', 'product']) -- make output pretty
-- Output
date          product    sales    running_total_sales
2024-01-10    coffee      509     509
2024-01-11    coffee      423     932
2024-01-12    coffee      789     1721
2024-01-12    tea         500     500
2024-01-13    tea         605     1105
2024-01-14    tea         340     1445
```

Phuong Nguyen
@fphuongnguyen

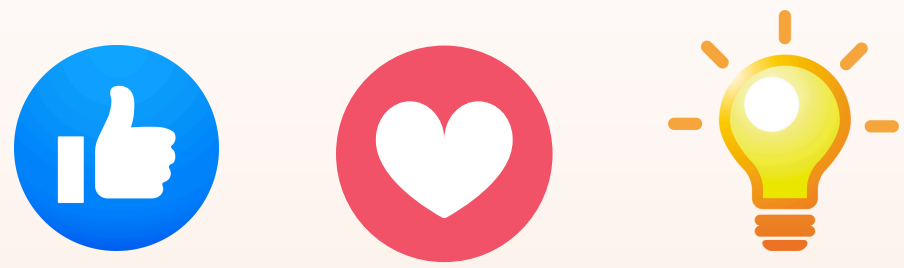
10. WINDOW FUNCTIONS avec PARTITIONS: LAG/LEAD et RANK

```
>>> sales
    date        product        sales
2024-01-12      coffee          789
2024-01-13      tea             605
2024-01-10      coffee          509
2024-01-14      tea             340
2024-01-12      tea             500
2024-01-11      coffee          423

-- 1. LAG/LEAD
-- 1a/ SQL
SELECT
  date,
  product,
  sales,
  LAG(sales) OVER (PARTITION BY product ORDER BY date) AS previous_day_sales,
  LEAD(sales) OVER (PARTITION BY product ORDER BY date) AS following_day_sales
FROM sales
-- 1b/ Pandas
df['previous_day_sales'] = df.sort_values(['date'])\
                                        .groupby('product')['sales']\
                                        .shift(1)
                # following_day_sales:   .shift(-1)
df.sort_values(['date', 'product'])
-- Output
date          product    sales    previous_day_sales    following_day_sales
2024-01-10    coffee      509           NaN                   423
2024-01-11    coffee      423           509                   789
2024-01-12    coffee      789           423                   NaN
2024-01-12    tea         500           NaN                   605
2024-01-13    tea         605           505                   340
2024-01-14    tea         340           605                   NaN

-- 2. RANK
-- 2a/ SQL
SELECT
  date,
  product,
  sales,
  RANK() OVER (PARTITION BY product ORDER BY sales DESC) AS rank_nb
FROM sales
-- 2b/ Pandas
df['rank_nb'] = df.sort_values(['date'])\
                .groupby('product')['sales']\
                .rank(ascending=False)
df.sort_values(['product', 'sales'], ascending=[True, False])
-- Output
date          product    sales    rank_nb
2024-01-12    coffee      789     1
2024-01-10    coffee      509     2
2024-01-11    coffee      423     3
2024-01-13    tea         605     1
2024-01-12    tea         500     2
2024-01-14    tea         340     3
```

Phuong Nguyen
@fphuongnguyen

Mettre en concurrence SQL et Pandas est un excellent moyen de renforcer ses connaissances et de monter en compétences, n'est-ce pas?

Mettez-moi un pouce si vous trouvez ce post utile!



Phuong Nguyen
@fphuongnguyen

Hello! Je suis Phuong Nguyen. Suis-moi pour des conseils et astuces en Data Analyse

Phuong Nguyen
@fphuongnguyen