Fastcampus Data Science Extension SCHOOL

SQL(3) - ORDER BY, JOIN, GROUP BY

Index

- review
- ORDER BY
- JOIN
- GROUP BY
- HAVING

`OrderDetails에서 Quantity가 40개 이상이며, Customers의 CustomerName이 Ernst나 Stop을 포함하는 전체 데이터를 선택하세요.`

Answer

```
SELECT * FROM [Orders]
where
OrderID in (
        SELECT OrderID FROM OrderDetails
            where Quantity > 40)
            and CustomerID in (
                SELECT CustomerID FROM [Customers]
                where CustomerName like '%Ernst%'
                or CustomerName like '%Stop%');
```

sort with pandas

```
products_df = pd.read_sql('select * from Products;', db)
products_df.sort_values('ProductName', ascending=False)\
        [["ProductName", "Price"]]
```

sort with sql - ORDER BY

```
query = """
    select ProductName, Price
    from Products
    order by ProductName desc
    ;
"""
pd.read_sql(query, db)
```

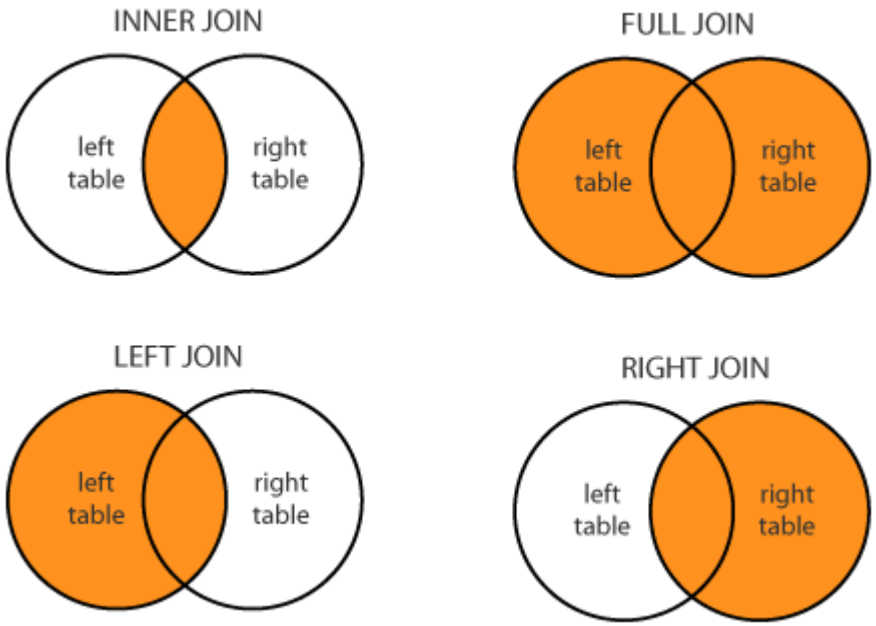
text mining - like

```
# text mining
query = """
    select ProductName, Price
    from Products
    where
        ProductName like "%Ch%"
    ;
"""
pd.read_sql(query, db)
```

JOIN

JOIN

- INNER JOIN: 양쪽의 값 비교 후 조건에 맞는 데이터 병합
- LEFT JOIN: JOIN 왼쪽을 기준으로 오른쪽의 일치하는 데이터 병합
- SELF JOIN: 자기 자신을 병합
- RIGHT JOIN: JOIN 오른쪽을 기준으로 왼쪽의 일치하는 데이터 병합
- FULL OUTER JOIN : 일치하지 않는 값까지 모두 병합
- sqlite는 RIGHT JOIN 과 FULL OUTER JOIN을 지원하지 않습니다.


INNER JOIN
left table
right table
FULL JOIN
left table
right table
LEFT JOIN
left table
right table
RIGHT JOIN
left table
right table

join in pandas - merge

```
integrated_df = orders_df.merge(df, on="CustomerID")\
        [["OrderID", "CustomerID","ContactName","Address"]]
integrated_df.head()
```

join with sql - don't

```
query = """
    select *
    from Customers, Orders
    ;
"""
pd.read_sql(query, db)
```

join with sql - better(1)

```
query = """
    select Orders.OrderID, Orders.CustomerID,
        Customers.ContactName, Customers.Address
    from Customers, Orders
    where
        Customers.CustomerID = Orders.CustomerID
    ;
"""
```

join with sql - better(2)

```
query = """
    select O.OrderID, O.CustomerID, C.ContactName, C.Address
    from Customers C, Orders O
    where
        C.CustomerID = O.CustomerID
    ;
"""
```

join with sql - best

```
query = """
    select O.OrderID, O.CustomerID, C.ContactName, C.Address
    from Customers C
        join Orders O
        on C.CustomerID = O.CustomerID
    ;
"""
```

GROUP BY in pandas

```
date_groups = orders_df.groupby("OrderDate")
date_groups.get_group("1996-07-08")

orders_df["OrderDate"].unique()
```

```
order_count_by_date = pd.DataFrame([
    {
        "OrderDate": OrderDate,
        "Count": len(date_groups.get_group(OrderDate)),
    } for OrderDate in orders_df["OrderDate"].unique()
])
order_count_by_date
```

GROUP BY

```
#sql
query = """
    select count(*), OrderDate
    from Orders
    group by OrderDate
    ;
"""
pd.read_sql(query, db)
```

Having vs where

- 공통점: condition
- where
  - 항상 from 뒤에 위치
  - 모든 필드에 대해 필터링 가능
- having
  - group by 뒤에 위치
  - group by 후 생성된 새로운 테이블에 조건을 줄때

how to use having?

```
query = """
    SELECT
        SUM(d.Quantity) "Count",
        SUM(d.Quantity * p.Price) "Sales",
        ROUND(AVG(d.Quantity * p.Price), 2) "avg",
        SUBSTR(o.OrderDate, 0, 8) "month"
    FROM
        OrderDetails d
        JOIN
            Products p
            ON p.ProductID = d.ProductID
        JOIN
            Orders o
            ON d.OrderID = o.OrderID
    GROUP BY
        substr(o.OrderDate, 0, 8)
    HAVING
        d.Quantity >= 20
    ;
"""
pd.read_sql(query, db)
```

Do It Yourself

`OrderDate`를 조작하여 yyyy-mm 의 형태로 바꾼 컬럼을 추가한 뒤, 연-월 기반의 주문횟수를 sql로 구현하세요

숙제

- 앞서 배운 groupby, join을 활용하여 월간 판매량 합과 평균 구매가격을 sql로 구현하세요.