SQL
Note 4

Financial Database
BAF 507E

Inmoo Lee

KAIST

Fall (1st half) 2025

Table of Contents

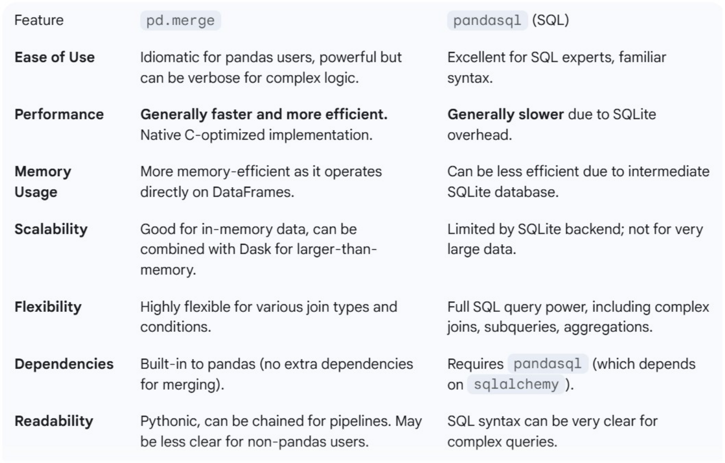
SQL: JOIN

- Combining multiple datasets is very easy in sql
- JOINS
  - Inner join
  - Left or Right join (Right join is not available in sqldf but we can use a trick to conduct Right join in sqldf)
- Cartesian join
  - All possible combinations of rows from each table is produced
  - Since it cannot be done explicitly in sqldf, you can use Left join without conditions to conduct the cartesian join.
- Check FDNote4W2025.ipynb for examples.

Pandas Merge

- In Python, you can use Pandas Merge to join Dataframes.
- **pd.merge** can do inner, left, right, outer joins as in SQL.
- It is convenient to simply merge dataframes in Python but SQL is more convenient in joining tables in a more complicated way.
- Check some examples in Appendix 1 at the last parts of FDNote4W2025.ipynb.

pandasql vs. pd.merge



Feature	`pd.merge`	`pandasql` (SQL)
Ease of Use	Idiomatic for pandas users, powerful but can be verbose for complex logic.	Excellent for SQL experts, familiar syntax.
Performance	**Generally faster and more efficient.** Native C-optimized implementation.	**Generally slower** due to SQLite overhead.
Memory Usage	More memory-efficient as it operates directly on DataFrames.	Can be less efficient due to intermediate SQLite database.
Scalability	Good for in-memory data, can be combined with Dask for larger-than-memory.	Limited by SQLite backend; not for very large data.
Flexibility	Highly flexible for various join types and conditions.	Full SQL query power, including complex joins, subqueries, aggregations.
Dependencies	Built-in to pandas (no extra dependencies for merging).	Requires `pandasql` (which depends on `sqlalchemy`).
Readability	Pythonic, can be chained for pipelines. May be less clear for non-pandas users.	SQL syntax can be very clear for complex queries.

Results from Gemni

How to add returns over a period using SQL

- You can use *group by* to apply functions to each group and use *where* or *on* to specify the data items to be included
- To calculate the sum of returns over a particular period (e.g., summing monthly returns over the next twelve months), for each monthly return in a dataframe sorted by ID and Dates, one can "join" the same dataframe using the conditions that specify the returns to be included in the summation (i.e., the returns of the firms with the same id on the dates between the day of the current return and the day after twelve months).
- Check the query used to add returns over the next twelve months.

Calculate holding period returns using SQL

- Buy-and-hold returns (BHR) over the n periods can be calculated as follows (e.g., using monthly returns, over 3 years ($n = 36$))

$$(1 + r_1) \times (1 + r_2) \times ... \times (1 + r_n) = 1 + BHR_n$$

- Since SQL does not have multiplications, we have to use a trick to calculate cumulative returns using SQL.
    - We can convert returns to log returns so that we can use 'sum' instead of 'product'.

Tricks

- Take $log = ln = log_{exp}$ in both sides of the equation in the previous page, then

$$log((1+r_1)\times(1+r_2)\times ... \times(1+r_n)) = log(1+BHR_n)$$

- Since $log(x \times y) = log(x) + log(y)$,

$$log(1+BHR_n) = log(1+r_1) + log(1+r_2)... + log(1+r_n)$$

- $log(a) = b$ implies that $exp^b = a$. If we let $log(1+BHR_n) = b$, and $1 + BHR_n = a$, then

$$exp^b = exp^{log(1+BHR_n)} = a = 1 + BHR_n$$

- Therefore, to calculate BHRs, you can sum log returns ($=log(1+r)$) over a range of dates, and then convert it to $1+$ BHR by using the exponential function.

Working with Dates

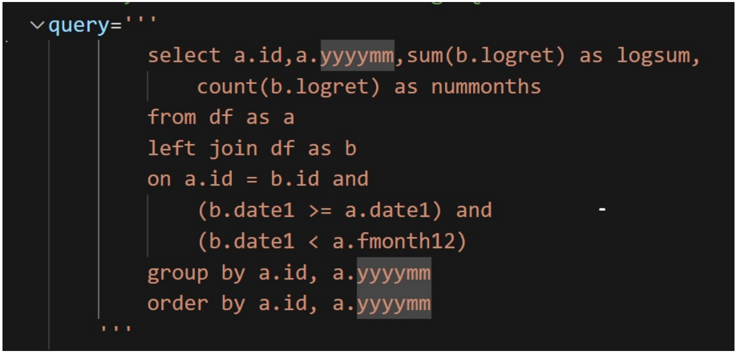
- Utilizing the following package, you can determine the end date of a month.

  ```
  from pandas.tseries.offsets import MonthEnd
  ```

- One can convert numbers to a date format that can be recognized as a date in Pandas by using pd.to_datetime()

Adding returns over the next 12-month period

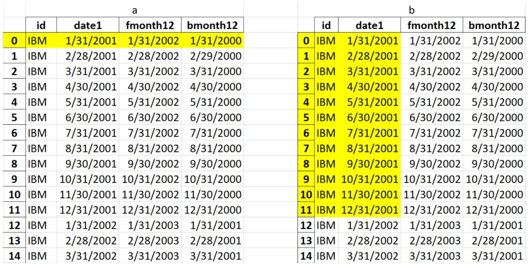
- The following SQL query statement will add 12 monthly log returns for each year-month of each id and call the result logsum.



```
query='''
        select a.id,a.yyyymm,sum(b.logret) as logsum,
            count(b.logret) as nummonths
        from df as a
        left join df as b
        on a.id = b.id and
            (b.date1 >= a.date1) and
            (b.date1 < a.fmonth12)
        group by a.id, a.yyyymm
        order by a.id, a.yyyymm
    '''
```

- The next slide explains how it works for IBM in January 2001.

Adding returns over the next 12-month period



	a			
	id	date1	fmonth12	bmonth12
0	IBM	1/31/2001	1/31/2002	1/31/2000
1	IBM	2/28/2001	2/28/2002	2/29/2000
2	IBM	3/31/2001	3/31/2002	3/31/2000
3	IBM	4/30/2001	4/30/2002	4/30/2000
4	IBM	5/31/2001	5/31/2002	5/31/2000
5	IBM	6/30/2001	6/30/2002	6/30/2000
6	IBM	7/31/2001	7/31/2002	7/31/2000
7	IBM	8/31/2001	8/31/2002	8/31/2000
8	IBM	9/30/2001	9/30/2002	9/30/2000
9	IBM	10/31/2001	10/31/2002	10/31/2000
10	IBM	11/30/2001	11/30/2002	11/30/2000
11	IBM	12/31/2001	12/31/2002	12/31/2000
12	IBM	1/31/2002	1/31/2003	1/31/2001
13	IBM	2/28/2002	2/28/2003	2/28/2001
14	IBM	3/31/2002	3/31/2003	3/31/2001

	b			
	id	date1	fmonth12	bmonth12
0	IBM	1/31/2001	1/31/2002	1/31/2000
1	IBM	2/28/2001	2/28/2002	2/29/2000
2	IBM	3/31/2001	3/31/2002	3/31/2000
3	IBM	4/30/2001	4/30/2002	4/30/2000
4	IBM	5/31/2001	5/31/2002	5/31/2000
5	IBM	6/30/2001	6/30/2002	6/30/2000
6	IBM	7/31/2001	7/31/2002	7/31/2000
7	IBM	8/31/2001	8/31/2002	8/31/2000
8	IBM	9/30/2001	9/30/2002	9/30/2000
9	IBM	10/31/2001	10/31/2002	10/31/2000
10	IBM	11/30/2001	11/30/2002	11/30/2000
11	IBM	12/31/2001	12/31/2002	12/31/2000
12	IBM	1/31/2002	1/31/2003	1/31/2001
13	IBM	2/28/2002	2/28/2003	2/28/2001
14	IBM	3/31/2002	3/31/2003	3/31/2001

- For the first row (Jan. 2001) in a, you choose the ones that satisfy the conditions (i.e., among those with the same id (IBM), choose the ones in b with the date1 being greater or euqal to the date1 in a (i.e., 1/31/2001) and less than fmonth12 in a (i.e., 1/31/2002)).
- The ones in b, which satisfy the conditions (a.id=b.id and b.date1 $>=$ a.date1 and b.date1$<$a.fmonth12) are 12 rows in yellow cells in b.
- After finding those in b, you sum the logret values of those in b and call it as logsum.

Calculation in Excel

- In the "bhrs" worksheet of the "note4data.xlsx" file, three different methods are used to calculate 12-month BHRs.
    - Array multiplication: {=PRODUCT(1+C2:C13)-1}
        - By pressing "Ctrl", "Shift" and "Enter" simultaneously while the cursor is in the formula window, the above will calculate the following $(1 + C2) \times (1 + C3) \cdots \times (1 + C13) - 1$
    - Log transformation: Calculate the sum of 12 log returns (ln(1+return)) in the 'logsum' column, and then convert it to BHR in the "exp(logsum)-1" column by using exponential function and subtracting one from its value.
    - Calculate (1+return) in the "1+return" column and then calculate the cumulative product in the "cum(1+return)" columns
        - The 12th value in the "cum(1+return)" column minus 1 is the BHR for the first month (Jan. 2001)