Contents

Tips, Tricks, and Pitfalls

Level 6

I: Inner Join

The most important, and most often misunderstood detail about joins is what join keys are, how join keys work, and which join keys to use.

We are going to use an example not related to our dataset to demonstrate this.

QTY Table		
Month	User	Quantity
Jan	1	100
Jan	2	200
Feb	1	300
Feb	2	400

Dollar Table		
Month	User	Dollar Amt
Jan	1	100
Jan	2	200
Feb	1	300
Feb	2	400
Mar	1	300
Mar	2	200

In this example we have two columns that describe our data (attributes) and one column with a value. These are both fact tables.

Let's take the following question:

"*Display the quantity and dollar amt on a single line for each month & user. Filter only on user = 1*"

Ok, so we know what we want the output to look like:

Month	User	Quantity	Dollar Amt
Jan	1	100	100
Feb	1	300	300

Now you might ask, "what about March 1?". Well, We are using an INNER JOIN. Since March is not in QTY Table, it is being excluded. If we would use a FULL JOIN (not supported in MySQL) or a LEFT JOIN, we would see March. But let's stick with INNER JOIN for now.

The most common mistake done on OUR dataset is students who only join on CUSIP. That would be the equivalent here of only joining on USER. Let's see what that would look like:

```
SELECT
        Q.MONTH,
        Q.USER,
        Q.QUANTITY,
        D.DOLLAR_AMT
        FROM QTY Q
INNER JOIN DOLLAR D
        ON Q.USER = D.USER
WHERE D.USER = 1
```

Here is what your result would look like:

Month	User	Quantity	Dollar Amt
Jan	1	100	100
Feb	1	100	300
Mar	1	100	300
Jan	1	300	100
Feb	1	300	300
Mar	1	300	300

This clearly is not the correct answer, seemingly it makes no sense.

A very good way to tell if your join is working or not is to first filter on a small dataset that you know exactly how many rows to expect. For example, in our TRADE_DATA_HIST table we have 100 rows per day, so if you filter on 20180102 only, you can safely expect 100 rows using an inner join. If your join is returning 200 , 300 or even 1000 rows, this is a sign that you are missing join keys!

Back to our example here.

We expected 2 rows and got back 6. A multiple! As mentioned in the yellow statement above this (getting back a multiple of how many rows you are expecting) is a good sign we are missing a join key. Let's dissect this output a bit.

First of all, notice Mar is in the result set. Even though this is an INNER JOIN, since we are only joining on USER, Mar does indeed have user = 1 so that join is satisfied.

QTY Table

Month	User	Quantity
Jan	1	100
Feb	1	300

Dollar Table

Month	User	Dollar Amt
Jan	1	100
Feb	1	300
Mar	1	300

We are joining "**ON Q.USER = D.USER"**

So let's go one row at a time.

The very first row in QTY has a User = 1, so we need to find all matching rows of USER=1 in the joining table. As there are three such rows, the join will be satisfied three times:

Month	User	Quantity	Dollar Amt
Jan	1	100	100
Feb	1	100	300
Mar	1	100	300

What month will display does depend on whether you write "D.MONTH" or "Q.MONTH" in your query, but the output will look generally the same.

The next row in QTY will continue the join, also finding a match here User = User a total of three time:

QTY Table

Month	User	Quantity
Jan	1	100
Feb	1	300

Dollar Table

Month	User	Dollar Amt
Jan	1	100
Feb	1	300
Mar	1	300

And the join will accomplish:

Month	User	Quantity	Dollar Amt
Jan	1	300	100
Feb	1	300	300
Mar	1	300	300

That is, this QTY = 300 row will match with all three rows of User = 1 in the other table.

But this is all clearly wrong. What we really want to do is join these two tables where User = User AND Month = Month.

That is: “Show me the QTY and Dollar Amt of each user on each month.”

Thinking about this upfront, there is only ONE row in each table where User = 1 AND Month = Jan. Therefore the join for that row will be:

Month	User	Quantity	Dollar Amt
Jan	1	100	100

Which is EXACTLY what we are looking for.

So doing this instead:

INNER JOIN DOLLAR D

ON Q.USER = D.USER

AND Q.MONTH = D.MONTH

WHERE D.USER = 1

Would result in:

Month	User	Quantity	Dollar Amt
Jan	1	100	100
Feb	1	300	300

And once again, Mar 1 just won’t show up here because Month = Month is part of the INNER JOIN now and that does not get satisfied anymore.

II: Dimension Tables

Joining on a dimension table is completely different, but has a similar concern. If we would join TDH on SECURITY_INFO just on CUSIP, let’s say we had a CUSIP “XYZ”, as we are looking at a single day of TDH, which has 100 rows.

XYZ has TWO rows in SECURITY_INFO, since it has a country change in 2015, it has a row before the change and after.

What we should do is make sure that we are looking at the correct row.

If we just blindly INNER JOIN on CUSIP = CUSIP then each of these rows will be used and **we would get 200 rows back**!

To combat this you need to also join on

TDH.COB_DATE BETWEEN SI.START_DATE AND SI.END_DATE

This will ensure that every day in TDH that is queried only joins on the rows in SI where that SI row was active. Practice with this a lot, it definitely takes some practice to get used to.

III: Left Join

I will only discuss LEFT join here since right is the same but is rarely used (since it is a bit more confusing).

A left join is when you take EVERYTHING from your "select" table and then only bring in data from the join table where the key matches are found.

Let's do a similar exercise to the one above, but a bit simpler. Let's take out the month column entirely, just for this example:

QTY Table	
User	Quantity
1	100
2	100

Dollar Table	
User	Dollar Amt
1	100
3	100

```
SELECT
        Q.USER,
        Q.QUANTITY,
        D.DOLLAR_AMT
        FROM QTY Q
LEFT JOIN DOLLAR D
        ON Q.USER = D.USER
```

Each table has only two rows in total. We are selecting from QTY table, so all rows should appear in the result, as per definition of a LEFT JOIN.

Here would be the output:

User	Quantity	Dollar Amt
1	100	100
2	100	

User = 1 is a perfect match so of course that join works.

User = 2 shows up in the result since this is a left join, but as there is no match in the Dollar table, we won't see any Dollar Amt there, not even 0 (it would be NULL).

User = 3 doesn't appear in the results at all since this is a left join and that user does not exist in the "select" table.

Use Case

When would we ever want to use a left join and not an inner join?

Keep in mind, an inner join is always the fastest.

There are many cases, and you'll get more familiar as you use SQL more and more.

A simple example for our trading data would be to see if there are any trades that have risk being calculated.

What we would do is select from the trade table, and left join on the risk table.

Any positions that come up as NULL for VaR & Duration are missing from the risk table (if any are).

IV: Full Join

As mentioned in the lecture, MySQL does not allow natural FULL JOINs. Any questions in 6.3 should be answered using syntax AS IF a full join would work in MySQL.

V: Self Join

Note that you don't just use SELF JOIN for a one day change in value. You can SELF JOIN over a month or even a year. Also, you can use it to determine items that are in one day but not in the next using LEFT JOIN (see how we figured out what was missing from the Risk table above).

Warning: *I (Mark) made a mistake in the video lectures. I demonstrated using self joins with INNER JOINs. This is dangerous because maybe there is a CUSIP in the data yesterday but not today. This should show for example as a 100% decline in Market Value. If we use an INNER JOIN, instead the CUSIP will just not appear at all. This is wrong. In general it's always best to use a LEFT JOIN for a SELF JOIN.*

There are two very cool functions not covered in this course that can simulate self joins and do some very epic stuff. I encourage you to check them out. You may use them on homework #7 (completely optional), but not before then.

They are LEAD and LAG functions known as "window functions": https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html

VI: Union

A union is much more simple than a join as it doesn't use keys at all; it just stacks data. It's pretty rare to use unions but it's good to understand how they work.

For 6.5.7 Do not answer this question "I would explain to my manager that a join is better." No. There are multiple ways to solve this using a union, but you need to get creative.

One such solution is to create a VaR column and a Duration column and set the value to NULL where there is no data. Ooops, I gave away the solution…feel free to use it ☺.

VI: Keys, a Final Tip from Mark

When you build your own dataset you have the luxury as I have to create your own PRIMARY KEYS. I knew we would be joining TRADE_DATA_HIST and RISK_DATA_HIST very frequently so I made sure the trades have the same exact position ids. These are unique to each DATE-EMPLOYEE-CUSIP-FUND. In other words, there will never be the same date/employee/fund/cusip traded on the same day in the TRADE_DATA_HIST table. Since this is unique to each date, we are safe to join just on this primary key and nothing else!

Let's say RISK_DATA_HIST had EMPLOYEE_ID and FUND_ID as well (it doesn't). We would be able to do:

SELECT … FROM DBO.TRADE_DATA_HIST TDH

INNER JOIN DBO.RISK_DATA_HIST RDH

ON RDH.CUSIP = TDH.CUSIP

AND RDH.EMPLOYEE_ID = TDH.EMPLOYEE_ID

AND RDH.COB_DATE = TDH.COB_DATE

AND RDH.FUND_ID = TDH.FUND_ID

We need ALL of these or we will have the join issue we mentioned above with multiplying rows.

However, since PRIMARY_KEY incorporates all of this information into a single column we can instead just say:

SELECT … FROM DBO.TRADE_DATA_HIST TDH

INNER JOIN DBO.RISK_DATA_HIST RDH

ON TDH.POSITION_ID = RDH.POSITION_ID

And we would get the same result.