4. Best practices

Identify whether statements below are considered as a good practice. And if not, what negative

outcome could happen and what would you recommend as a better solution? Please add your personal experiences.

1. Adding resources as a solution to slow query: Bad practice. It can be helpfull short term but it will cost more money long term. Its better to optimize the query.
2. Calculate division like "SELECT category, sum(views) / sum(active\_users) as view\_per\_active\_user FROM table GROUP BY category": The logic is fine only issue is we can get error for dividing by zero if sum of active\_users is zero for one of the categories.
3. Calculating the average on columns containing NULL values will return a number that we wanted: Avarage in sql by deafult will exclude nulls. We need to consider that in our calculations.
4. Every developer should use their own notation and tools during DWH development to keep diversity: No, this will create a chaos
5. Every job is monitored and errors are handled from one place: This is a good practice. We have all info in one place
6. External data documentation in dedicated Excel files is easier to maintain: In some cases this is fine. If the project will not grow to much. But its better to use markdown files + git
7. FTP is well standardized and secure for data interchange: No, FTP is outdated. The replacment is a SFTP but is also not widely used in modern data enginnering.
8. Full load is good for small tables: Yes. It is fine but we should consider if the project will grow. If yes then is better to implement incremental loading from begining.
9. Manual database updates or inserting rows inside the core layer of DWH: No this is a bad approach. I had a case where it resulted to chaning the whole column to NULL in production
10. Multiple nested subqueries: Hard to read and can cause performance issues. It is better rethink how we write our query if we encounter this.
11. Releasing directly to production to deliver results quicker: Bad practice. This can result to data loss, long downtimes for clients, etc.
12. Removing columns: If we figured out we dont need the info from that column anymore then its fine but it is still better aproach to back up it somewhere before deletion.
13. Repeating the same code in multiple places: Bad practice, it will make the codebase hard to maintain.
14. Switching to the hottest technologies as soon as possible: Bad practice. We dont know if the hottest technolgy will be suported long term. But also it is good to reconsider if the techstack we use is not outdated.
15. Using "SELECT \* FROM..." in ETL pipelines: Bad practice. If we pull the columns we dont need that will cost us execution time, storage and money.
16. Using the basic SQL function DATE\_TIME() is safe when deploying everywhere: I did not encounter this function. But every SQL flawor mostly has its custom approach for working with dates.
17. Using cross joins or lateral joins: Cross joins can be an issue if the tables we join are large. So we need to be carefull. I never encounterd lateral joins so i cant say anything about them.

**3. Performance issue**

Your colleague wrote a SQL transformation and released it into production. It works, but after some

time you found out that the computation time is rapidly increasing.

What are the most usual causes for this behaviour from your previous experience? What are the

possible solutions?

Some of the reasons a and solutions are talked about in next chapters questions: 1,8,10,15,17

**But some of the ideas**

- It is posible that the project simple outgrow the current cloud or local computational power tier of the Warehause(for example Snowlake).  
- The data grew out the current tech stack.

- The query is written in inefficient way.

- No indexing or wrong indexing is used. We can try to change that it depends on the logic of the query.