How do I make complex SQL queries easier to write? [closed]

35

I'm finding it very difficult to write complex SQL queries involving joins across many (at least 3-4) tables and involving several nested conditions. The queries I'm being asked to write are easily described by a few sentences, but can require a deceptive amount of code to complete. I'm finding myself often using temporary views to write these queries, which seem like a bit of a crutch. What tips can you provide that I can use to make these complex queries easier? More specifically, how do I break these queries down into the steps I need to use to actually write the SQL code?

25

Note that I'm the SQL I'm being asked to write is part of homework assignments for a database course, so I don't want software that will do the work for me. I want to actually understand the code I'm writing.

More technical details:

- The database is hosted on a PostgreSQL server running on the local machine.
- The database is very small: there are no more than seven tables and the largest table has less than about 50 rows.
- The SQL queries are being passed unchanged to the server, via LibreOffice Base.

sql tips query

share improve this question

edited Apr 16 '12 at 4:13

asked Apr 16 '12 at 2:41

bwDraco
297 1 5 13

closed as too broad by MichaelT, Scant Roger, amon Nov 25 '15 at 23:06

There are either too many possible answers, or good answers would be too long for this format. Please add details to narrow the answer set or to isolate an issue that can be answered in a few paragraphs.

If this question can be reworded to fit the rules in the help center, please edit the question.

Temporary views is actually quite useful as you can do things to such a table (like explicit complex indexes) that is very hard to hint to the SQL parser. – user1249 May 8 '12 at 11:16

Personally, I find it easier to cheat by using a GUI (such as LibreOffice Base "Create Query in Design View" or Office Access "Create" > "Query Design") and then view the SQL this produces. Sometimes it is necessary to modify the SQL given by a GUI designer, but it gives a good starting point – kurdtpage Jun 25 '15 at 20:28

add a comment

6 Answers

active oldest **votes**

I'm basing most of this on just trying to get the "right" answer, so you may discover there are some performance issues. No point in speeding up an incorrect query.

44 **Understand the table relationships** - Most will be one to many. Know the "many" table. Identify the fields required for your joins.

Think about LEFT join scenarios - Select all the employees and their paycheck from last month. What if they didn't get a paycheck last month?

Know the result set: 1) In a spreadsheet, manually enter at least one correct record for your query. 2) Write the query in a simple enough form to identify how many records should be returned. Use both of these to test your query to make sure joining a new table doesn't alter the result.

Break up your query into managable parts - You don't have to write it all at once. Complex queries can sometimes just be a collection of simple queries.

Beware of mixed levels of aggregation: If you have to put monthly, quarterly and year-to-date values in the same result set, you'll need to calculate them separately in queries grouped on different values.

Know when to UNION Sometimes it's easier to break up subgroups into their own select statements. If you have a table mixed with managers and other employees, and on each column you have to do Case statements based on membership in one of these groups, it may be easier to write a Manager query and union to an Employee query. Each one would contain their own logic. Having to include items from different tables in different rows is an obvious use.

Complex/Nested formulas - Try to consistently indent and don't be afraid to use multiple lines. "CASE WHEN CASE WHEN CASE WHEN" will drive you nuts. Take the time to think these through. Save the complex calcs for last. Get the correct records selected first. Then you attack complex formulas knowing you're working with the right values. Seeing the values used in the formulas will help you spot areas where you have to account for NULL values and where to handle the divide by zero error.

Test often as you add new tables to make sure you're still getting the desired result set and knowing which join or clause is the culprit.

share improve this answer

edited Oct 12 '15 at 6:26

josliber
103 3

answered Apr 16 '12 at 14:58

JeffO
33.2k 2 47 117

1 Excellent advice. – NoChance May 8 '12 at 11:59

1 Really excellent stuff. I want to reemphasize Jeff's points on looking for LEFT joins and splitting complex queries into smaller, more manageable ones and then combining them. I write big queries on big databases pretty much everyday and those two things in particular come up all the time. Always run your queries and sub queries as soon as you can, to make sure you are getting the data that you expect to see at each step.– CodexArcanum May 8 '12 at 13:28

@CodexArcanum - and when you run queries on big data, it doesn't hurt to use TOP ;) – JeffO May 8 '12 at 13:41

I agree on every statement of your suggestion – Alessandro Rossi Oct 16 '15 at 10:18

add a comment

27

1. **Indentation** would be the first thing to do, if you're not doing it already. Not only it's useful with even simple queries, but it is crucial when it comes to joins and queries a bit more complex than a `select top 1 [ColumnName] from [TableName]` .
2. Once indented properly, nothing forbids to **add comments** inside the query itself, when appropriate. Don't overuse them: if the code is explicit enough, adding comments will just harm the clarity of code. But they are still welcome for the less explicit parts of the query.

   Note that longer queries (including queries with comments) would mean larger bandwidth usage between your application server and your database server. Also note that unless you are working on a Google-scale product with a huge amount of requests per second, requiring an exceptional performance and resources usage, the size added by the comments may not change anything for you in terms of performance.
3. **Enforcing the same style over tables, columns, etc.** helps the readability a lot too. When a legacy database has the tables `PRODUCT` , `users` , `USERS_ObsoleteDONT_USE` , `PR_SHIPMENTS` and `HRhbYd_UU` , somebody is doing something very wrong.
4. **Enforcing the same style over queries** is important too. For example if you are writing queries for Microsoft SQL Server and you decided to use `[TableName]` instead of `TableName` , stick with it. If you go to a new line after a `select` , don't do it in only half of your queries, but all of them.
5. **Don't use** `*` , unless there are strong reasons to do it (like in `if exists(select * from [TableName] where ...)` in Microsoft SQL Server). Not only `*` has a negative performance impact in some (if not most) databases, but it's also not helpful for the developer who uses your query. In the same way, a developer must access the values by name, never by index.
6. Finally, for selects, there is nothing wrong in providing a **view**. For anything else, **stored procedures** may be also used depending on the project and the people[1] you're working[2] with.

[1] Some people hate stored procedures. Others don't like them for several (perfectly valid, at least for them) reasons.

[2] Your colleagues, the other students, your teacher, etc.

share improve this answer

edited Apr 16 '12 at 15:21

answered Apr 16 '12 at 3:37

MainMa

88.4k 20 211 355

add a comment