How to write a simple SQL query

Marcus Schaefer – IT 240

If you are working with a database, the most common requests you will get are requests for data which means you’ll have to write SQL queries. Until you become a master of SQL, here’s advice on how to go about doing that. We first list the advice, and then explain it in more detail using examples. By a simple query we mean a query of the form

SELECT <attributes>

FROM <tables>

WHERE <conditions>;

in which neither the FROM nor the WHERE clauses contain other SQL queries (nested queries). Most of the advice given below also directly applies to more complex queries, but some of it needs modification to deal with the nesting structure. The examples below are written in SQL for Microsoft Access.

**Before** you Write the Query

1. **Read** the request. Make sure you **understand** it. **Clarify** if necessary.
2. Do the query **by hand**.
3. **Reflect** on what you did.

**Writing** the Query

1. **FROM**
2. **WHERE**
   1. Join conditions
   2. Particular conditions
3. **SELECT**

**After** Writing the Query

1. **Test** the query

§1 **Before you Write the Query**

1. **Read** the request. Make sure you **understand** it. **Clarify** if necessary.

The person making the request may not know the details of what data is stored in the database; they may use terms differently from the way you/the database is using them. Or they may simply be imprecise or not have a very clear idea of what data they want. In all of these cases, it is your responsibility to clarify.

**Example**: You work for the university registrar and Dean A asks you to get a list “of all students that started four years ago”. This request may seem clear enough, but it really isn’t:

1. What is meant by “four years ago”? At least four years ago, four years ago to the day, in the academic year starting four years ago, in the calendar year starting four years ago, or something else? All of these are reasonable interpretations.
2. What does it mean that a student “started”? That they were admitted at that time? How about if they were readmitted? What if they were admitted and didn’t enroll? What if this was the first quarter they were enrolled.
3. What about “all students”? Does Dean A really mean all students, or just students who are still around? Indeed, students that have graduated (or dropped out) since they “started four years ago” may have been removed from the database (real database keep all data, nothing is ever removed, it is just made inactive, but in simpler databases, such data may simply be eliminated), so that info may not even be in the database anymore, so an answer may be incorrect.
4. Finally, what does “list of all students” mean? Do they just want their names, or their addresses/phone numbers/emails as well?

If the request is ambiguous or unclear, make sure you talk to the person that made the request and clarify it. Keep in mind that they may not be database experts, so some interpretation may be necessary. However, if you decide to interpret a request in a particular way (because of missing data or because you cannot talk to the person that requested the data), make sure you clearly state what your query actually does.

**Example**: For the above request you weren’t able to get hold of Dean A, so you may send back your list with a memo explaining that you retrieved “the names/addresses of all students who were first admitted in academic year 2006-2007 (assuming the current academic year is 2010/2011), independently of whether they enrolled or not and independently of whether they are still currently students”.

1. Do the query **by hand**.

It’s hard to overemphasize how important this step is. Intuitively, you will know how to find the information. The main problem will be to formalize this in SQL. By the way, it is quite possible that there is no answer to a particular request, maybe no students enrolled in 2006/2007. If you do the query by hand first you won’t be surprised when later on you get a surprising result like this. More importantly, if you have a rough idea of what the result looks like you have a sanity check on the later output. Of course, in general you can’t anticipate the whole output of a typical query on a large database. But maybe, by hand, you determined that John Doe started school in 2006/2007. If John Doe doesn’t show up when you run your SQL query later on, you know that something is wrong.

1. **Reflect** on what you did.

In other words, when you solve the query by hand, think about what you are doing. You can often translate what you are doing directly into an SQL query. This is particularly true for simple queries. Here are questions to keep in mind: what tables are involved? What attributes are involved?

§2 **Writing the Query**

For a simple query, that is a query of the form

SELECT <attributes>

FROM <tables>

WHERE <conditions>;

write the query in the following order: (1) FROM, (2) WHERE, (3) SELECT.

1. **FROM**: Start by including all the tables you used when you did the query by hand. Go through the process again in your head. You may have used some tables more than once. If so, you need to include them more than once in the FROM clause as well; in this case you definitely need to assign alias names to the tables. Even in general it makes sense to give names to the tables to make the query easier to read.

**Example**: You are asked to “list employees and their supervisors.” When you did the query by hand you looked up a particular employee record, went to the super\_ssn column to get the SSN of the supervisor and then looked up that SSN in the same employee table. You used the employee table twice: once as the employee (let’s call him/her E) and once as the supervisor (let’s call him/her S). So you start your query as:

SELECT <attributes>

FROM employee AS E, employee AS S

WHERE <conditions>;

1. **WHERE**: Write the conditions for the WHERE clause in two steps: first, include all conditions necessary to join the tables together; secondly, add conditions that are particular to the query. For the first step, think back to doing the query by hand. How did you link the records in the table to get meaningful information? You equated foreign keys with the primary keys they point to. Add those conditions (connected together by ANDs) to the WHERE clause. In the second step, again think back to performing the query by hand. What conditions did you use to filter out only those records you were interested in? Add those to the WHERE clause.

**Example**: Consider the following simple database schema: Employee(EID, name, sex, deptno) and Department(DeptID, Dname, ManagerID); there are foreign keys: ManagerID -> EID and deptno -> DeptID. You want to list all female employees in the Research department. The underlying query, stripped of the particular conditions is: which employees work for which department. So, to link up the tables you need deptno (since that tells you which department an employee works for):

SELECT <attributes>

FROM employee AS E, Department AS D

WHERE E.deptno = D.DeptID;

This is a good time to stop and test whether your query is going well (even more important for more complex queries); before you continue to the second step, run this query to make sure you have the right data to work with. Replace <attributes> with the primary keys (and some identifying information) from each table, e.g.

SELECT E.EID, E.name, D.DeptID, D.Dname

FROM employee AS E, Department AS D

WHERE E.deptno = D.DeptID;

If this output looks good, continue, otherwise, fix. In the second step you add the particular conditions (female and Research):

SELECT <attributes>

FROM employee AS E, Department AS D

WHERE E.deptno = D.DeptID

AND E.sex = ‘F’

AND D.Dname = ‘Research’;

If the query had been for the manager of the research department, you would have used the other connection between the tables, with

SELECT <attributes>

FROM employee AS E, Department AS D

WHERE E.EID = D.managerID;

In the second you add the particular condition (Research):

SELECT <attributes>

FROM employee AS E, Department AS D

WHERE E.EID = D.managerID

AND D.Dname = ‘Research’;

Note that in these examples there are two choices for linking up the tables: E.EID = D.managerID and E.deptno = D.DeptID ; which is the right choice? That depends on the query; the easiest way to figure it out is, as often, seeing what you did when you did the query by hand.

Connecting two tables requires one foreign key = primary key condition as illustrated above; with more tables, you need more conditions, but fortunately it’s easy to check how many conditions there need to be: if you have n tables to connect, you need n-1 equalities of the form foreign key = primary key.

1. **SELECT**: this is typically the easiest part of the query: select the attributes you want to display. If you are asked for a particular entity you typically want to include the p.k. of that entity together with other relevant information. Sometimes SELECT clauses get more interesting if you need to use functions to do calculations (string concatenation, date calculations, arithmetical calculations). The syntax for calculations depends very much on what flavor of SQL you are using.

**Example**: Your task is to list all books and their price before and after tax. Assuming a 7% sales tax, we have:

SELECT BID, title, price, price\*1.07

FROM Books;

Note that we include the book id field, (BID), since there may be many books with the same title (even from the same publisher). If there are natural fields uniquely identifying a record, they are preferable to the internal database IDs, here, for example, the following version would be better:

SELECT title, ISBN, price, price\*1.07

FROM Books;

For people databases (students, employees, etc.) never assume that a name uniquely identifies a person.

§3 **After you’ve Written the Query**

1. **Test** the query.

Run the query and see what output it produces. When you first did the query by hand, you got a couple of results. Make sure your SQL query returns those results as well. It is quite possible that your query doesn’t return any results (there may not be any records corresponding to your query in the database). In that case you should temporarily add records to the database so you do get results. Whatever you do, do a sanity check. If your query asked for all female employees and you get a male employee, or if your asked to list all employees and your list is longer than the corresponding table in the database, there is something wrong. Go back, check, and fix.

§4 **Warnings and Suggestions**

1. Don’t use magic numbers.

A magic number (or magic word) is a piece of data you’re using in your query that has no meaning in terms of the request, but only in the tables of the database.

**Example**: You are asked what department John Smith works for. You look at the tables and see that John Smith works for department number 5, so you write the query:

SELECT dname

FROM department

WHERE dnumber = 5;

5 here is the magic number. Where does it come from? If you go back to this query later on, you’ll have no idea why there is a 5 there. Worse, if you use this query as part of a larger query which is in use for years, the query may stop working for many reasons: John Smith changes departments or the internal numbering system for departments is changed, so that the research department no longer has number 5. For all these reasons, the correct answer would have been:

SELECT dname

FROM department, employee

WHERE dnumber = dno AND fname = ‘John’ AND lname = ‘Smith’;

Now there is another problem here (what if there are many John Smith’s, does your query assume that there is only one answer?) but that is a separate problem.

1. Magic doesn’t happen – Read your query.

Sitting down and just writing the query and hoping that it’ll work won’t work. Unless you are already a master of SQL in which case you don’t need the advice. Magic and hopeful thinking don’t work in SQL, SQL does exactly what you tell it to. If that’s different from what you’re expecting it too, you didn’t communicate well enough with SQL. Here is one way of dispelling the magic: read your query as if you had not written the query yourself: this way you find out what you are actually asking SQL to do for you. That should tell you what’s wrong.

**Example**: You’ve written the following query to find all employees in the Research department

SELECT E.name

FROM employee AS E, Department AS D

WHERE D.Dname = ‘Research’;

but you are getting too many records (including employees that are not in the Research department). As you read the query, here is what you are asking SQL to do: combine all records in employee with all records in department and limit to those for which Dname is ‘Research’. The problem then is that every record in employee is coupled with every record in department, leading to bad combinations (so-called spurious tuples). You are missing the join condition between employee and department.

1. Limit interpretation

Sometimes the data requests don’t match perfectly with the structure of your database; at that point you may have to put an interpretation on the request. Minimize this (and make it clear to the client).

**Example**: You are asked to list all movies in your database suitable for children; the closest you have in your database is the MPAA rating of the movie, so now you need to decide which subset of {G, PG, PG-13, R, NC-17, NR} corresponds to “suitable for children”. Make sure you tell the requester how you interpreted the request (better yet, ask them to clarify for you what they want).

**Example**: You are asked to list all students who are younger than 40, but the student table only has birthdate information. Today is 9/18/2010, so you write the query

SELECT name

FROM student

WHERE bdate > #9/18/1970#;

(you got the date and the > right, both common mistakes). However, this query will stop working tomorrow, and there is just a chance that you may get the calculation wrong (would you have blinked if you’d seen 1960 instead of 1970?). So it is better to use internal functions to do the calculation, e.g.

SELECT name

FROM student

WHERE DateDiff(‘yyyy’, bdate, Now()) < 40;

One could see this example as a violation of magic numbers (why 1970? The data request didn’t mention 1970, it mentioned 40) or the making of unwarranted assumptions (even if we run the query tomorrow, using today’s date is fine?). Now for birthday’s this may not matter so much, but imagine you are retrieving stock information.

1. Don’t make assumptions

A data request may need interpretation to fit what’s actually in the database, but that is no excuse to make unwarranted assumptions. In particular assumptions based on data currently in the database are dangerous.

**Example**: Your database list contains records for employee dependents of the following types: spouse, son, daughter. You are asked to retrieve the names of all children of employees. You write

SELECT name

FROM dependent

WHERE type <> ‘spouse’;

While this query may work in your current database state it’s not robust: if other types of relationships are added to the database (second-domiciled adults, for example) your query will stop work correctly. So keep the query accurate:

SELECT name

FROM dependent

WHERE type in (‘son’, ‘daughter’);

1. I don’t know how to do this

Happens all the time, this is when you look things up. The most important point is that you need to be able to figure out what exactly you don’t know how to do, so you know where to look.

**Example**: You are asked to list all employees with names and age. However, your database only contains the birthdate of each employee. So do you give up? No, you can write most of the query:

SELECT name, bdate

FROM employee;

Except you don’t want bdate but the employee’s age. At this point you recall that SQL has functions that calculate such information. Go to the manuals and figure out how to do it in your particular flavor of SQL. In Access, you’d end up with something like:

SELECT name, DateDiff(‘yyyy’, bdate, Now())

FROM employee;

1. Where did all those records come from?

If you have too many outputs from your query including duplicates, then chances are good these are spurious tuples resulting from not joining all the tables involved. Recall that if you are combining n tables you need n-1 conditions of the form p.k. = f.k.

1. My query isn’t working

Why not? Try to be specific in isolating what the problem is (and reading all the previous advice should help you doing that):

* Have you really understood the underlying data request?
* You have trouble writing the query? Why?
  + The logical structure is confusing.

Assuming you have understood the structure of the query, the main problem is mapping it to SQL’s conventions. Running the query by hand and reflecting on how that worked are the single-most important steps in understanding the logical structure. Then consider the structures available to you in SQL to express the request.

* + I can’t get it to run.

Are there syntactical problems? Common pitfalls: quotes around words are missing, dates are not formatted correctly. Case of words matters (in some systems, e.g. sex = ‘female’ can give different results from sex = ‘Female’, though not in all SQL systems).

* + It’s not giving me the right output.

If you have too many outputs, you may be missing joining conditions, see above. If you have no output that may actually be correct. If none of this works, reread the query as explained above to understand what you actually wrote and see how it fits in with what you were planning to write.

The main point here is that if you encounter a problem you should be able to isolate the nature of the problem: syntactical (spelling, quotes), semantical (wrong logic: wrong structure or wrong conditions) or missing knowledge. All of these are problems that can be fixed, but you need to know what problem the problem is before you can attack it.