Which is faster/best? SELECT * or SELECT column1, colum2, column3, etc

Asked 16 years, 3 months ago Modified 1 month ago Viewed 124k times

222

I've heard that `SELECT *` is generally bad practice to use when writing SQL commands because it is more efficient to `SELECT` columns you specifically need.

If I need to `SELECT` every column in a table, should I use

```
SELECT * FROM TABLE
```

or

```
SELECT column1, colum2, column3, etc. FROM TABLE
```

Does the efficiency really matter in this case? I'd think `SELECT *` would be more optimal internally if you really need all of the data, but I'm saying this with no real understanding of database.

I'm curious to know what the best practice is in this case.

UPDATE: I probably should specify that the only situation where I would really *want* to do a `SELECT *` is when I'm selecting data from one table where I know all columns will always need to be retrieved, even when new columns are added.

Given the responses I've seen however, this still seems like a bad idea and `SELECT *` should never be used for a lot more technical reasons that I ever though about.

sql database performance select sqlperformance

Share

Improve this question

Follow

edited Dec 20, 2022 at 18:32

community wiki
11 revs, 6 users 53%
Dan Herbert

4 See also: stackoverflow.com/questions/208925/… – user7675 Aug 12, 2009 at 12:47

1 see: stackoverflow.com/questions/262450/… stackoverflow.com/questions/65512/… stackoverflow.com/questions/413819/select-except stackoverflow.com/questions/208925/… stackoverflow.com/questions/487578/… stackoverflow.com/questions/1433971/… – George Stocker Jun 4, 2010 at 13:54

1 Yes, it's a duplicate of most of those. – George Stocker Jun 4, 2010 at 13:55

50 Answers

Sorted by: Highest score (default)

201

One reason that selecting specific columns is better is that it raises the probability that SQL Server can access the data from indexes rather than querying the table data.

Here's a post I wrote about it: **The real reason select queries are bad index coverage**

It's also less fragile to change, since any code that consumes the data will be getting the same data structure regardless of changes you make to the table schema in the future.

Share

Improve this answer

Follow

edited Feb 15, 2018 at 3:57

community wiki
2 revs, 2 users 71%
Jon Galloway

3 +1 for this. If all the columns referenced exist in a single index (a "covering index"), you've struck gold. – Ian Nelson Sep 15, 2008 at 18:44

31 that's not the answer to his question - "If I need to SELECT every column in a table,..." -- in that case, * vs col1, .., coln doesn't matter (but it DOES for programmer time, since * is shorter!). – Matt Rogish Sep 15, 2008 at 19:23

4 It still matters, because the select list is a form of contract, expecially if the SQL is in a stored procedure. – Eric Z Beard Sep 15, 2008 at 23:22

4 While what Jon says is completely correct, and a very valid point, I have to concur that the question AS ASKED is about if they are the already asking for all columns. Because of this part of the question, the real issues is fragility in the face of schema changes. – IDisposable Sep 17, 2008 at 20:01

1 @MattRogish sir you got it correctly, is there any performance difference between these two methods (* `vs` all_column_names) while we have thousands of rows and we perform SELECT with index (in WHERE clause) ?? – santoshe61 Oct 31, 2018 at 7:58

Given **your** specification that you **are** selecting all columns, there is little difference **at this time**. Realize, however, that database schemas do change. If you use `SELECT *` you are going to get any new columns added to the table, even though in all

likelihood, your code is not prepared to use or present that new data. This means that you are exposing your system to unexpected performance and functionality changes.

You may be willing to dismiss this as a minor cost, but realize that columns that you don't need still must be:

1. Read from database
2. Sent across the network
3. Marshalled into your process
4. (for ADO-type technologies) Saved in a data-table in-memory
5. Ignored and discarded / garbage-collected

Item #1 has many hidden costs including eliminating some potential covering index, causing data-page loads (and server cache thrashing), incurring row / page / table locks that might be otherwise avoided.

Balance this against the potential savings of specifying the columns versus an `*` and the only potential savings are:

1. Programmer doesn't need to revisit the SQL to add columns
2. The network-transport of the SQL is smaller / faster
3. SQL Server query parse / validation time
4. SQL Server query plan cache

For item 1, the reality is that you're going to add / change code to use any new column you might add anyway, so it is a wash.

For item 2, the difference is rarely enough to push you into a different packet-size or number of network packets. If you get to the point where SQL statement transmission time is the predominant issue, you probably need to reduce the rate of statements first.

For item 3, there is NO savings as the expansion of the `*` has to happen anyway, which means consulting the table(s) schema anyway. Realistically, listing the columns will incur the same cost because they have to be validated against the schema. In other words this is a complete wash.

For item 4, when you specify specific columns, your query plan cache could get larger but **only** if you are dealing with different sets of columns (which is not what you've specified). In this case, you **do want** different cache entries because you want different plans as needed.

So, this all comes down, because of the way you specified the question, to the issue resiliency in the face of eventual schema modifications. If you're burning this schema into ROM (it happens), then an `*` is perfectly acceptable.

However, my general guideline is that you should only select the columns you need, which means that **sometimes** it will look like you are asking for all of them, but DBAs and schema evolution mean that some new columns might appear that could greatly affect the query.

My advice is that you should **ALWAYS SELECT specific columns**. Remember that you get good at what you do over and over, so just get in the habit of doing it right.

If you are wondering why a schema might change without code changing, think in terms of audit logging, effective/expiration dates and other similar things that get added by DBAs for systemically for compliance issues. Another source of underhanded changes is denormalizations for performance elsewhere in the system or user-defined fields.

Share

Improve this answer

Follow

answered Sep 15, 2008 at 21:53

community wiki
IDisposable

3 "the reality is that you're going to add / change code to use any new column you might add anyway, so it is a wash." -Only if you're manually reading each column by name in your code. If you're using automatic mapping, this is not the case, and this issue becomes significant. – Josh Noe Dec 2, 2013 at 19:36

44

You should only select the columns that you need. Even if you need all columns it's still better to list column names so that the sql server does not have to query system table for columns.

Also, your application might break if someone adds columns to the table. Your program will get columns it didn't expect too and it might not know how to process them.

Apart from this if the table has a binary column then the query will be much more slower and use more network resources.

Share

Improve this answer

Follow

edited Jun 4, 2010 at 12:40

scunliffe
63.5k ● 26 ● 131 ● 165

answered Jun 4, 2010 at 6:48

Giorgi
30.8k ● 13 ● 91 ● 127

7 Aha so by using * you are adding extra work for the DB. Ok that's one reason I hadn't thought of. – Ankur Jun 4, 2010 at 6:50

1 +1 for risks of breaking/catching mistakes early. I think the discussion of efficiency is valid but YAGNI. – nailitdown Jun 4, 2010 at 7:36

6 Wouldn't the SQL server need to validate or check if "col1" is in the specified table anyway, i.e. query system table? – Patrick Jun 4, 2010 at 10:10

4 The biggest performance hit is probably related to indexing. If the column you are looking for is part of the index used to find the data the server will fetch the data right there, if you do a select * it will most likely have to do what is called a bookmark lookup, which requires an extra scan to find the rest of the underlying data, which you may not even need. – Cobusve Jun 4, 2010 at 12:12

3 @Patrick - Spot on. Lots of good reasons to avoid * but that isn't one of them. – Martin Smith Jun 4, 2010 at 12:49

39

There are four big reasons that `select *` is a bad thing:

1. The most significant practical reason is that it forces the user to magically know the order in which columns will be returned. It's better to be explicit, which also protects you against the table changing, which segues nicely into...
2. If a column name you're using changes, it's better to catch it early (at the point of the SQL call) rather than when you're trying to use the column that no longer exists (or has had its name changed, etc.)
3. Listing the column names makes your code far more self-documented, and so probably more readable.
4. If you're transferring over a network (or even if you aren't), columns you don't need are just waste.

Share Improve this answer Follow

answered Jun 4, 2010 at 7:03

pkh
3,749 ● 1 ● 24 ● 18

13 "The most significant practical reason is that it forces the user to magically know the order in which columns will be returned." I don't see how this is an issue. In any modern DB client, you read columns by name, not order. – Josh Noe Dec 2, 2013 at 19:39

I tend to run my SQL through a C interface, so I wouldn't really know what the state of the art in "DB clients" is. But I think probably the kind of client you're talking about is doing some non-standard non-SQL magic. (e.g., in SQLite, querying sqlite3_master to figure out how to change your `*` into a set of names.) – pkh Dec 2, 2013 at 21:11

And further on from this how many people write code in modern applications that either use index of column names? Most people surely are using some sort of mapper and a whole heap of caching for data that is allowed to be stale. Personally, write the code first, and then worry if you have performance issues later. – Colin Wiseman Jul 10, 2017 at 19:44

@JoshNoe "I don't see how this is an issue. In any modern DB client, you read columns by name, not order." That depends on if performance is important to you. While you can access

columns in your app by name rather than order, it will be way faster if you access the columns by order. – jmarkmurphy Aug 30, 2021 at 21:39

11

Specifying column names is definitely faster - for the server. But if

1. **performance is not a big issue** (for example, this is a website content database with hundreds, maybe thousands - but not millions - of rows in each table); AND
2. your job is to create **many small, similar applications** (e.g. public-facing content-managed websites) using a common framework, rather than creating a complex one-off application; AND
3. **flexibility is important** (lots of customization of the db schema for each site);

then you're better off sticking with SELECT *. In our framework, heavy use of SELECT * allows us to introduce a new website managed content field to a table, giving it all of the benefits of the CMS (versioning, workflow/approvals, etc.), while only touching the code at a couple of points, instead of a couple dozen points.

I know the DB gurus are going to hate me for this - go ahead, vote me down - but in my world, developer time is scarce and CPU cycles are abundant, so I adjust accordingly what I conserve and what I waste.

Share Improve this answer Follow

answered Sep 22, 2008 at 14:11

community wiki
Herb Caudill

1 It also makes ORMs much simpler to use. When queries are built by passing a query building object around, one is not necessarily aware which columns were required by what other parts of code (permission checks, what have you). So to limit the columns, one would need to investigate every time a query needs writing. This is pointless, IMO. When queries do turn out to be slow (logs!), one can improve those. – bytepusher Apr 12, 2019 at 7:26

10

Specifying the column list is *usually* the best option because your application won't be affected if someone adds/inserts a column to the table.

Share Improve this answer Follow

answered Sep 15, 2008 at 18:50

community wiki
ilitirit

SELECT * is a bad practice even if the query is not sent over a network.

6

1. Selecting more data than you need makes the query less efficient - the server has to read and transfer extra data, so it takes time and creates unnecessary load on the system (not only the network, as others mentioned, but also disk, CPU etc.). Additionally, the server is unable to optimize the query as well as it might (for example, use covering index for the query).
2. After some time your table structure might change, so SELECT * will return a different set of columns. So, your application might get a dataset of unexpected structure and break somewhere downstream. Explicitly stating the columns guarantees that you either get a dataset of known structure, or get a clear error on the database level (like 'column not found').

Of course, all this doesn't matter much for a small and simple system.

Share Improve this answer Follow

answered Jun 4, 2010 at 7:00

VladV
10.4k ● 3 ● 35 ● 48

5

Lots of good reasons answered here so far, here's another one that hasn't been mentioned.

Explicitly naming the columns will help you with maintenance down the road. At some point you're going to be making changes or troubleshooting, and find yourself asking "where the heck is that column used".

If you've got the names listed explicitly, then finding every reference to that column -- through all your stored procedures, views, etc -- is simple. Just dump a CREATE script for your DB schema, and text search through it.

Share Improve this answer Follow

answered Jun 4, 2010 at 14:00

Chris Wuestefeld
3,326 ● 3 ● 25 ● 23

4

Performance wise, SELECT with specific columns can be faster (no need to read in all the data). If your query really does use ALL the columns, SELECT with explicit parameters is still preferred. Any speed difference will be basically unnoticeable and near constant-time. One day your schema will change, and this is good insurance to prevent problems due to this.

Share
Improve this answer
Follow

answered Sep 15, 2008 at 18:42

community wiki
Yann Ramin

You are wrong about the unnoticeable as from checks I made with several DBs it was clear that selecting each column, even if all of them, is much faster. In some cases it was three times faster. – shahar eldad Jun 10, 2019 at 7:13

3

definitely defining the columns, because SQL Server will not have to do a lookup on the columns to pull them. If you define the columns, then SQL can skip that step.

Share

Improve this answer

Follow

answered Sep 15, 2008 at 18:39

community wiki
Nick Berardi

This is: 1) irrelevant, because SQL Server has to reference the table schema either way (to validate the column names or to lookup the known-valid column names) 2) Not relevant to the question asked, where all columns are being referenced. The only issue AS ASKED is fragility w/ schema changes. – IDisposable Sep 17, 2008 at 20:03

Downvoted, because it's gotta validate the columns regardless. – John Gibb Apr 8, 2010 at 17:40

3

It's always better to specify the columns you need, if you think about it one time, SQL doesn't have to think "wtf is *" every time you query. On top of that, someone later may add columns to the table that you actually do not need in your query and you'll be better off in that case by specifying all of your columns.

Share

Improve this answer

Follow

answered Sep 15, 2008 at 18:40

community wiki
BrewinBombers

1 This is not true: SQL server must still parse each column and see if it exists in the catalogs, whereas it *knows* that "*" does (and yes, * is expanded to all cols). Either way, it's trivially easy for the DBMS to do either one (unless you have 24,000 columns), so I bet it's the same either way – Matt Rogish Sep 15, 2008 at 19:25

I think the better point that many are missing and, unfortunately, this answer only addresses secondarily, is that if schema/table changes happen (i.e. new columns added) it won't break things. – Sean Hanley Sep 15, 2008 at 19:58

1 It's a complete wash as looking up the columns for the * expansion is the same as validating the column names provided. – IDisposable Sep 17, 2008 at 20:07

3

The problem with "select *" is the possibility of bringing data you don't really need. During the actual database query, the selected columns don't really add to the computation. What's really "heavy" is the data transport back to your client, and any column that you don't really need is just wasting network bandwidth and adding to the time you're waiting for you query to return.

Even if you do use all the columns brought from a "select *...", that's just for now. If in the future you change the table/view layout and add more columns, you'll start bring those in your selects even if you don't need them.

Another point in which a "select *" statement is bad is on view creation. If you create a view using "select *" and later add columns to your table, the view definition and the data returned won't match, and you'll need to recompile your views in order for them to work again.

I know that writing a "select *" is tempting, 'cause I really don't like to manually specify all the fields on my queries, but when your system start to evolve, you'll see that it's worth to spend this extra time/effort in specifying the fields rather than spending much more time and effort removing bugs on your views or optimizing your app.

Share

Improve this answer

Follow

answered Sep 15, 2008 at 18:47

community wiki
Alexandre Brasil

> The point on VIEWs is very important. Not only will you not be getting all the columns if you add columns to the table (despite what the * would make you think), but they might not even match the real layout of the table. – Euro Micelli Sep 15, 2008 at 23:23

3

While explicitly listing columns is good for performance, don't get crazy.

So if you use all the data, try SELECT * for simplicity (imagine having many columns and doing a JOIN... query may get awful). Then - measure. Compare with query with column names listed explicitly.

Don't speculate about performance, **measure it!**

Explicit listing helps most when you have some column containing big data (like body of a post or article), and don't need it in given query. Then by not returning it in your answer DB server can save time, bandwidth, and disk throughput. Your query result will also be smaller, which is good for any query cache.

Share

Improve this answer

answered Sep 15, 2008 at 18:47

community wiki
Paweł Hajdan

Follow

3

You should really be selecting only the fields you need, and only the required number, i.e.

```
SELECT Field1, Field2 FROM SomeTable WHERE --(constraints)
```

Outside of the database, dynamic queries run the risk of injection attacks and malformed data. Typically you get round this using stored procedures or parameterised queries. Also (although not really that much of a problem) the server has to generate an execution plan each time a dynamic query is executed.

Share Improve this answer Follow

answered Jun 4, 2010 at 6:50

Matthew Abbott
61.5k ● 9 ● 107 ● 130

"the server has to generate an execution plan each time a dynamic query is executed" which I assume slows down the query. Thanks. – Ankur Jun 4, 2010 at 6:52

The performance issues of using dynamic sql would probably only be realised in very high load scenarios, Sql Server is pretty good at managing query plans efficiently.
– Matthew Abbott Jun 4, 2010 at 6:57

3

This is an old post, but still valid. For reference, I have a very complicated query consisting of:

- 12 tables
- 6 Left joins
- 9 inner joins
- 108 total columns on all 12 tables
- I only need 54 columns
- A 4 column Order By clause

When I execute the query using Select *, it takes an average of 2869ms. When I execute the query using Select , it takes an average of 1513ms.

Total rows returned is 13,949.

There is no doubt selecting column names means faster performance over Select *

Share

answered May 4, 2019 at 14:29

community wiki
Premier Bowling Scorekeeper

Improve this answer

Follow

2

It is NOT faster to use explicit field names versus *, if and only if, you need to get the data for all fields.

Your client software shouldn't depend on the order of the fields returned, so that's a nonsense too.

And it's possible (though unlikely) that you need to get all fields using * because you don't yet know what fields exist (think very dynamic database structure).

Another disadvantage of using explicit field names is that if there are many of them and they're long then it makes reading the code and/or the query log more difficult.

So the rule should be: if you need all the fields, use *, if you need only a subset, name them explicitly.

Share

Improve this answer

Follow

answered Sep 15, 2008 at 19:02

community wiki
user9385

2

The result is too huge. It is slow to generate and send the result from the SQL engine to the client.

The client side, being a generic programming environment, is not and should not be designed to filter and process the results (e.g. the WHERE clause, ORDER clause), as the number of rows can be huge (e.g. tens of millions of rows).

Share Improve this answer Follow

answered Jun 4, 2010 at 6:47

kennytm
523k ● 110 ● 1.1k ● 1k

So if you needed to actually use all the different columns it would be fine ... and if your database and app are sitting on the same server again, it doesn't make much difference?
– Ankur Jun 4, 2010 at 6:49

@Ankur: Even on the same server there's cost to transmit data over the database interface.
– kennytm Jun 4, 2010 at 6:54

Naming each column you expect to get in your application also ensures your application won't break if someone alters the table, as long as your columns are still present (in any order).

2

Share Improve this answer Follow

answered Jun 4, 2010 at 6:58

Don

9,661 ● 4 ● 28 ● 25

2

Performance wise I have seen comments that both are equal. but usability aspect there are some +'s and -'s

When you use a (select *) in a query and if some one alter the table and add new fields which do not need for the previous query it is an unnecessary overhead. And what if the newly added field is a blob or an image field??? your query response time is going to be really slow then.

In other hand if you use a (select col1,col2,..) and if the table get altered and added new fields and if those fields are needed in the result set, you always need to edit your select query after table alteration.

But I suggest always to use select col1,col2,... in your queries and alter the query if the table get altered later...

Share

Improve this answer

Follow

answered Mar 12, 2015 at 9:48

community wiki

Lahiru Cooray

1

Select is equally efficient (in terms of velocity) if you use * or columns.

The difference is about memory, not velocity. When you select several columns SQL Server must allocate memory space to serve you the query, including all data for all the columns that you've requested, even if you're only using one of them.

What does matter in terms of performance is the excecution plan which in turn depends heavily on your WHERE clause and the number of JOIN, OUTER JOIN, etc ...

For your question just use SELECT *. If you need all the columns there's no performance difference.

Share

Improve this answer

Follow

answered Sep 15, 2008 at 18:43

community wiki

Jorge Córdoba

1

It depends on the version of your DB server, but modern versions of SQL can cache the plan either way. I'd say go with whatever is most maintainable with your data access code.

Share
Improve this answer
Follow

answered Sep 15, 2008 at 18:43

community wiki
Keith

1

One reason it's better practice to spell out exactly which columns you want is because of possible future changes in the table structure.

If you are reading in data manually using an index based approach to populate a data structure with the results of your query, then in the future when you add/remove a column you will have headaches trying to figure out what went wrong.

As to what is faster, I'll defer to others for their expertise.

Share
Improve this answer
Follow

answered Sep 15, 2008 at 18:44

community wiki
dpollock

1

As with most problems, it depends on what you want to achieve. If you want to create a db grid that will allow all columns in any table, then "Select *" is the answer. However, if you will only need certain columns and adding or deleting columns from the query is done infrequently, then specify them individually.

It also depends on the amount of data you want to transfer from the server. If one of the columns is a defined as memo, graphic, blob, etc. and you don't need that column, you'd better not use "Select *" or you'll get a whole bunch of data you don't want and your performance could suffer.

Share
Improve this answer
Follow

answered Sep 15, 2008 at 19:07

community wiki
Mark

1

To add on to what everyone else has said, if all of your columns that you are selecting are included in an index, your result set will be pulled from the index instead of looking up additional data from SQL.

Share

community wiki

Improve this answer

Follow

answered Sep 15, 2008 at 21:43 Dela

1

SELECT * is necessary if one wants to obtain metadata such as the number of columns.

Share

Improve this answer

Follow

answered Nov 9, 2008 at 13:33 community wiki Mark Etable

1

Gonna get slammed for this, but I do a select * because almost all my data is retrived from SQL Server Views that precombine needed values from multiple tables into a single easy to access View.

I do then want all the columns from the view which won't change when new fields are added to underlying tables. This has the added benefit of allowing me to change where data comes from. FieldA in the View may at one time be calculated and then I may change it to be static. Either way the View supplies FieldA to me.

The beauty of this is that it allows my data layer to get datasets. It then passes them to my BL which can then create objects from them. My main app only knows and interacts with the objects. I even allow my objects to self-create when passed a datarow.

Of course, I'm the only developer, so that helps too :)

Share

Improve this answer

Follow

answered Nov 5, 2009 at 16:46 community wiki klkitchens

1

What everyone above said, plus:

If you're striving for readable maintainable code, doing something like:

SELECT foo, bar FROM widgets;

is instantly readable and shows intent. If you make that call you know what you're getting back. If widgets only has foo and bar columns, then selecting * means you still

have to think about what you're getting back, confirm the order is mapped correctly, etc. However, if widgets has more columns but you're only interested in foo and bar, then your code gets messy when you query for a wildcard and then only use some of what's returned.

Share Improve this answer Follow

answered Jun 4, 2010 at 7:54

Faisal

11 ● 1

1

And remember if you have an inner join by definition you do not need all the columns as the data in the join columns is repeated.

It's not like listing columns in SQl server is hard or even time-consuming. You just drag them over from the object browser (you can get all in one go by dragging from the word columns). To put a permanent performance hit on your system (becasue this can reduce the use of indexes and becasue sending unneeded data over the network is costly) and make it more likely that you will have unexpected problems as the database changes (sometimes columns get added that you do not want the user to see for instance) just to save less than a minute of development time is short-sighted and unprofessional.

Share Improve this answer Follow

answered Jun 4, 2010 at 13:21

HLGEM

96.4k ● 15 ● 119 ● 189

1

In conclusion at least in **PostgreSQL**, the performance of **selecting all columns with and without** `*` is **almost the same**.

In **PostgreSQL**, I created `test` **table** with **10** `id_x` **columns** and **10 million rows** as shown below:

```
CREATE TABLE test AS SELECT generate_series(1, 10000000) AS id_1,
                            generate_series(1, 10000000) AS id_2,
                            generate_series(1, 10000000) AS id_3,
                            generate_series(1, 10000000) AS id_4,
                            generate_series(1, 10000000) AS id_5,
                            generate_series(1, 10000000) AS id_6,
                            generate_series(1, 10000000) AS id_7,
                            generate_series(1, 10000000) AS id_8,
                            generate_series(1, 10000000) AS id_9,
                            generate_series(1, 10000000) AS id_10;
```

Then, I ran 2 queries below alternately **20 times** in total. *Each query runs **10 times** in total:

```
SELECT * FROM test:
```

```
SELECT id_1, id_2, id_3, id_4, id_5, id_6, id_7, id_8, id_9, id_10 FROM test;
```

<Result>

	Select all columns with *	Select all columns without *
1st run	12.792 seconds	12.483 seconds
2nd run	12.803 seconds	12.608 seconds
3rd run	12.537 seconds	12.549 seconds
4th run	12.512 seconds	12.457 seconds
5th run	12.570 seconds	12.487 seconds
6th run	12.508 seconds	12.493 seconds
7th run	12.432 seconds	12.475 seconds
8th run	12.532 seconds	12.489 seconds
9th run	12.532 seconds	12.452 seconds
10th run	12.437 seconds	12.477 seconds
Average	12.565 seconds	12.497 seconds

Average of selecting all columns:

- **with `*` is 12.565 seconds**.
- **without `*` is 12.497 seconds**.

Share

Improve this answer

Follow

edited Dec 23, 2022 at 5:57

community wiki
3 revs
Kai - Kazuya Ito

Absolutely define the columns you want to SELECT every time. There is no reason not to and the performance improvement is well worth it.

They should never have given the option to "SELECT *"

0

Share

Improve this answer

Follow

answered Sep 15, 2008 at 18:43

community wiki
cazlab