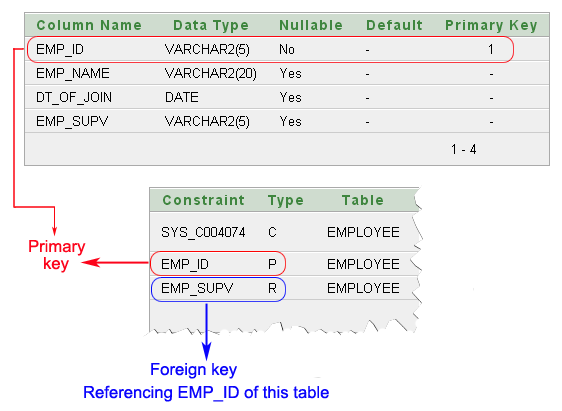
1. **Self Join**
2. **Sql do or do not**

**What is Self Join in SQL?**

A self join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

For this tutorial we have used a table EMPLOYEE, that has [one-to-many](http://www.w3resource.com/sql/question-answer.php) relationship.

**The structure of the table**



In the EMPLOYEE table displayed above, emp\_id is the primary key. emp\_supv is the foreign key (this is the supervisor’s employee id).

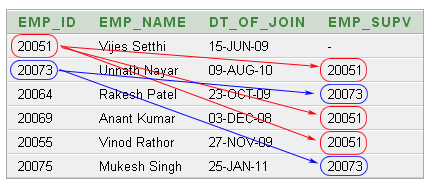
If we want a list of employees and the names of their supervisors, we’ll have to JOIN the EMPLOYEE table to itself to get this list.

**Unary relationship to employee**

**How the employees are related to themselves :**

* An employee may report to another employee (supervisor).
* An employee may supervise himself (i.e. zero) to many employee (subordinates).

We have the following data into the table EMPLOYEE.



**The above data shows :**

* UnnathNayar's supervisor is VijesSetthi
* Anant Kumar and Vinod Rathor can also report to VijesSetthi.
* Rakesh Patel and Mukesh Singh are under supervison of UnnithNayar.

**Example of SQL SELF JOIN**

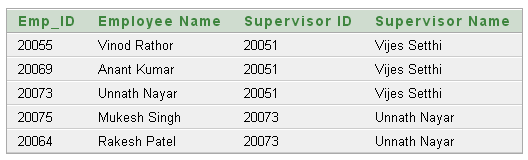
In the following example we will use the table EMPLOYEE twice and in order to do this we will use the alias of the table.

To get the list of employees and their supervisor the following sql statement has used :

[viewplainprint?](http://www.w3resource.com/sql/joins/perform-a-self-join.php)

1. SELECT a.emp\_id AS "Emp\_ID",a.emp\_name AS "Employee Name",
2. b.emp\_id AS "Supervisor ID",b.emp\_name AS "Supervisor Name"
3. FROM employee a, employee b
4. WHERE a.emp\_supv = b.emp\_id;

**Output**



http://newtonapples.com/difference-varchar-varchar2-char-data-types/  
Examine the below sql statement.

|  |
| --- |
| **SELECT**to\_date('20-MAR-1996','DD-MON-YYYY')**FROM** dual; |

Most of us get confused from the above statement that ’20-MAR-1996′ is already a date and why are we passing this value as input to TO\_DATE function.

Here is the catch whatever is the value given between single quotes ‘ ’ system consider it as string. We need to explicitly tell the system that ’20-MAR-1996′ string is a date in format ‘DD-MON-YYYY’ using TO\_DATE function.

|  |
| --- |
| Example-  For ‘20/03/1996’ – to\_date function will be TO\_DATE(‘20/03/1996’,’dd/mm/yyyy’ |

For date fields we use TO\_CHAR function to convert from one date format to another date format.

In the below SQL statement, using TO\_CHAR we are trying to convert date from ‘DD-MON-YYYY’ format to different format ‘YYYY/DD/MM’.

|  |
| --- |
| **SELECT**to\_char(to\_date('20-MAR-1996','DD-MON-YYYY'),'YYYY/DD/MM')**FROM** dual; |

From the above statement you can ask me a question saying why are we using to\_date inside to\_char why can’t we use to\_char directly for conversion. Because to\_char function expects date as a input parameter and to give date as input we need to apply to\_date first.

If we run same example without TO\_DATE then system throws an error saying ‘ORA-01722: invalid number’ as TO\_CHAR is expecting date input but we are passing a string (as earlier said anything between single quotes ‘ ’ system consider it as string)

|  |
| --- |
| **SELECT**to\_char('20-MAR-1996','YYYY/DD/MM')**FROM** dual;    Output  *------*  ORA-01722: invalid **NUMBER** |

# CHAR, VARCHAR and VARCHAR2 data types? http://newtonapples.com/difference-varchar-varchar2-char-data-types/

Most of us will have confusion between char, varchar and vahchar2 datatypes as all these datatypes store character values only.

In this article we will what is the exact difference between these datatypes.

## CHAR

Columns defined with CHAR datatype stores **fixed length character strings**. String values will be padded with space/blank to get that fixed length string before stored on disk. If this type is used to store variable length strings, it will waste a lot of disk space and will have performance issues.

|  |
| --- |
| SQL&gt; **CREATETABLE**na\_char\_test(col1 **CHAR**(10));    **TABLE** created.    SQL&gt; **INSERTINTO**na\_char\_test**VALUES**('apples');    1**ROW** created.    SQL&gt; **SELECT** col1,**LENGTH**(col1), dump(col1)"ASCII Dump"**FROM**na\_char\_test;    COL1 **LENGTH**(COL1) ASCII Dump  *---------- ------------ ------------------------------------------------------------*  apples 10Typ=96 Len=10: 97,112,112,108,101,115,32,32,32,32 |

**Result:** Actual length of ‘apples’ is 6 but as spaces are appended length shows 10. ASCII character 32 is a blank space, we can see four ‘32’ in the end for ‘ASCII Dump’ column

## VARCHAR2

**VARCHAR2** is used to store variable length character strings. That is if size of the string is 6 then it store only 6 characters and will not append any spaces.  
Oracle recommends using VARCHAR2 instead of VARCHAR as VARCHAR datatype is reserved for future usage.

|  |
| --- |
| SQL&gt; **CREATETABLE** na\_varchar2\_test (col1 VARCHAR2(10));    **TABLE** created.    SQL&gt; **INSERTINTO** na\_varchar2\_test **VALUES**('apples');    1**ROW** created.    SQL&gt; **SELECT** col1,**LENGTH**(col1), dump(col1)"ASCII Dump"**FROM** na\_varchar2\_test;    COL1 **LENGTH**(COL1) ASCII Dump  *---------- ------------ ------------------------------------------------------------*  apples 6Typ=1 Len=6: 97,112,112,108,101,115 |

**Result:** Now we can observe that no spaces are appended, actual length of ‘apples’ – 6 is returned.

## VARCHAR

Currently VARCHAR behaves exactly the same as VARCHAR2. However, this type should not be used as it is reserved for future usage.

|  |  |  |
| --- | --- | --- |
| SQL&gt; **CREATETABLE**na\_varchar\_test(col1 **VARCHAR**(10));    **TABLE** created.    SQL&gt; **INSERTINTO**na\_varchar\_test**VALUES**('apples');    1**ROW** created.    SQL&gt; **SELECT** col1,**LENGTH**(col1), dump(col1)"ASCII Dump"**FROM**na\_varchar\_test;    COL1 **LENGTH**(COL1) ASCII Dump  *---------- ------------ ------------------------------------------------------------*  apples 6Typ=1 Len=6: 97,112,112,108,101,115  xamine the below sql statement carefully we are trying to fetch full year from the date.   |  | | --- | | **SELECT**to\_char(to\_date('20-MAR-77','DD-MON-YY'),'YYYY')**FROM** dual; |   ***What will be the result of the query? Should it be ‘1977’ or ‘2077’ or ‘0077’.***  If we give a four digit year then no assumptions are made. The issue is all with 2 digit years.  For this reason Oracle has introduced ‘RRRR’ format to help people cope up with Y2K issues If the format is RR/RRRR and a two-digit year is given, then 2-digit years in the range 00 to 49 are assumed to have the same first two digits as the current year, and years given as 50 through 99 are assumed to be in the previous century. That is if the given 2-digit year is between 50 and 99 then the year will be assumed to be ‘19’.   |  | | --- | | **FOR** ‘RR’    **SELECT**to\_char(to\_date('20-MAR-77','DD-MON-RR'),'YYYY')**FROM** dual;    Output  *------*  1977    **FOR** ‘RRRR’(same **AS** RR format)    **SELECT**to\_char(to\_date('20-MAR-77','DD-MON-RRRR'),'YYYY')**FROM** dual;    Output  *------*  1977    **FOR** ‘YY’    **SELECT**to\_char(to\_date('20-MAR-77','DD-MON-YY'),'YYYY')**FROM** dual;    Output  *------*  2077    **FOR** ‘YYYY’    **SELECT**to\_char(to\_date('20-MAR-77','DD-MON-YYYY'),'YYYY')**FROM** dual;    Output  *------*  0077 |   If you have confusion on to\_date and to\_char functions then read this [article](http://newtonapples.com/difference-to_date-to_char-functions-date-fields/) . |

**10 more do's and don'ts for faster SQL queries**

More like this

* [Review: Apache Hive brings real-time queries to Hadoop](http://www.infoworld.com/article/2608271/hadoop/hadoop-review-apache-hive-brings-real-time-queries-to-hadoop.html)
* [Expert interview: How to scale Django](http://www.infoworld.com/article/2608078/application-development/expert-interview--how-to-scale-django.html)
* [7 performance tips for faster SQL queries](http://www.infoworld.com/article/2628420/database/7-performance-tips-for-faster-sql-queries.html)

Credit: iStockphoto

**Follow these tried-and-true techniques to improving both the speed and concurrency of your dawdling database**

[](http://www.infoworld.com/author/Sean-McCown/)By [Sean McCown](http://www.infoworld.com/author/Sean-McCown/)

[Follow](http://www.infoworld.com/article/2604472/database/10-more-dos-and-donts-for-faster-sql-queries.html#follow)

InfoWorld | Sep 17, 2014

Everyone wants faster database queries, and both SQL developers and DBAs can turn to many time-tested methods to achieve that goal. Unfortunately, no single method is foolproof or ironclad. But even if there is no right answer to tuning every query, there are plenty of proven do's and don'ts to help light the way. While some are RDBMS-specific, most of these tips apply to any relational database.

Featured Resource

[](http://resources.infoworld.com/ccd/assets/70734/detail)

Presented by Jive Software

[10 Commandments of Collaboration for Exceptional Customer Service](http://resources.infoworld.com/ccd/assets/70734/detail)

Read this whitepaper to discover best practices that drive brand affinity, repeat business and

[Learn More](http://resources.infoworld.com/ccd/assets/70734/detail)

My previous [collection of SQL do's and don'ts](http://www.infoworld.com/d/data-management/7-performance-tips-faster-sql-queries-262) concentrated on individual query performance. While you’ll definitely find more of that here, the main focus is on increasing concurrency in your system. By minimizing locking, I/O, and network traffic, you can not only make your queries run much faster, but even more important, you can make queries behave themselves on a system with hundreds or even thousands of concurrent users.

**[ Andrew C. Oliver answers the question on everyone's mind:** [**Which freaking database should I use?**](http://www.infoworld.com/resources/16281/database/quick-guide-which-freaking-database-should-i-use) **| Keep up with hot topics in app dev with InfoWorld's** [**Strategic Developer**](http://www.infoworld.com/blog/strategic-developer/) **blog. ]**

**1. Do use temp tables to improve cursor performance**

I hope we all know by now that it’s best to stay away from cursors if at all possible. Cursors not only suffer from speed problems, which in itself can be an issue with many operations, but they can also cause your operation to block other operations for a lot longer than is necessary. This greatly decreases concurrency in your system.

However, you can’t always avoid using cursors, and when those times arise, you may be able to get away from cursor-induced performance issues by doing the cursor operations against a temp table instead. Take, for example, a cursor that goes through a table and updates a couple of columns based on some comparison results. Instead of doing the comparison against the live table, you may be able to put that data into a temp table and do the comparison against that instead. Then you have a single UPDATE statement against the live table that’s much smaller and holds locks only for a short time.

Sniping your data modifications like this can greatly increase concurrency. I’ll finish by saying you almost never need to use a cursor. There’s almost always a set-based solution; you need to learn to see it.

**2. Don’t nest views**

Views can be convenient, but you need to be careful when using them. While views can help to obscure large queries from users and to standardize data access, you can easily find yourself in a situation where you have views that call views that call views that call views. This is called nesting views, and it can cause severe performance issues, particularly in two ways. First, you will very likely have much more data coming back than you need. Second, the query optimizer will give up and return a bad query plan.

I once had a client that loved nesting views. The client had one view it used for almost everything because it had two important joins. The problem was that the view returned a column with 2MB documents in it. Some of the documents were even larger. The client was pushing at least an extra 2MB across the network for every single row in almost every single query it ran. Naturally, query performance was abysmal.

And none of the queries actually used that column! Of course, the column was buried seven views deep, so even finding it was difficult. When I removed the document column from the view, the time for the biggest query went from 2.5 hours to 10 minutes. When I finally unraveled the nested views, which had several unnecessary joins and columns, and wrote a plain query, the time for that same query dropped to subseconds.

**3. Do use table-valued functions**

This is one of my favorite tricks of all time because it is truly one of those hidden secrets that only the experts know. When you use a scalar function in the SELECT list of a query, the function gets called for every single row in the result set. This can reduce the performance of large queries by a significant amount. However, you can greatly improve the performance by converting the scalar function to a table-valued function and using a CROSS APPLY in the query. This is a wonderful trick that can yield great improvements.

[Resources](http://www.infoworld.com/resources)

* [](http://resources.infoworld.com/ccd/assets/70900/detail)

White Paper

[IaaS: Performance Upgrade](http://resources.infoworld.com/ccd/assets/70900/detail)

* [](http://resources.infoworld.com/ccd/assets/70613/detail)

White Paper

[Enterprise Storage for Performance-driven Databases](http://resources.infoworld.com/ccd/assets/70613/detail)

[See All](http://www.infoworld.com/resources)

Top of Form



Bottom of Form

Want to know more about the APPLY operator? You'll find a full discussion in an excellent course on [Microsoft Virtual Academy](http://www.microsoftvirtualacademy.com/training-courses/boost-your-t-sql-with-the-apply-operator) by Itzik Ben-Gan.

**4. Do use partitioning to avoid large data moves**

Not everyone will be able to take advantage of this tip, which relies on partitioning in SQL Server Enterprise, but for those of you who can, it’s a great trick. Most people don’t realize that all tables in SQL Server are partitioned. You can separate a table into multiple partitions if you like, but even simple tables are partitioned from the time they’re created; however, they’re created as single partitions. If you're running SQL Server Enterprise, you already have the advantages of partitioned tables at your disposal.

This means you can use partitioning features like SWITCH to archive large amounts of data from a warehousing load. Let’s look at a real example from a client I had last year. The client had the requirement to copy the data from the current day’s table into an archive table; in case the load failed, the company could quickly recover with the current day’s table. For various reasons, it couldn’t rename the tables back and forth every time, so the company inserted the data into an archive table every day before the load, then deleted the current day’s data from the live table.

This process worked fine in the beginning, but a year later, it was taking 1.5 hours to copy each table -- and several tables had to be copied every day. The problem was only going to get worse. The solution was to scrap the INSERT and DELETE process and use the SWITCH command. The SWITCH command allowed the company to avoid all of the writes because it assigned the pages to the archive table. It’s only a metadata change. The SWITCH took on average between two and three seconds to run. If the current load ever fails, you SWITCH the data back into the original table.

This is a case where understanding that all tables are partitions slashed hours from a data load.

You Might Also Like

* [10 things you need to know about Intel's new Broadwell chips](http://www.infoworld.com/article/2864423/computer-hardware/10-things-you-need-to-know-about-intels-new-broadwell-chips.html)
* [](http://www.infoworld.com/article/2864437/it-careers/15-job-titles-getting-big-salary-boosts-in-2015.html)

[15 job titles getting big salary boosts in 2015](http://www.infoworld.com/article/2864437/it-careers/15-job-titles-getting-big-salary-boosts-in-2015.html)

* [](http://www.infoworld.com/article/2864592/mobile-technology/iphone-owners-sue-apple-over-storage-space-allege-fraud.html)

[iPhone owners sue Apple over storage space, allege fraud](http://www.infoworld.com/article/2864592/mobile-technology/iphone-owners-sue-apple-over-storage-space-allege-fraud.html)

**5. If you must use ORMs, use stored procedures**

This is one of my regular diatribes. In short, don’t use ORMs (object-relational mappers). ORMs produce some of the worst code on the planet, and they’re responsible for almost every performance issue I get involved in. ORM code generators can’t possibly write SQL as well as a person who knows what they're doing. However, if you use an ORM, write your own stored procedures and have the ORM call the stored procedure instead of writing its own queries. Look, I know all the arguments, and I know that developers and managers love ORMs because they speed you to market. But the cost is incredibly high when you see what the queries do to your database.

Stored procedures have a number of advantages. For starters, you’re pushing much less data across the network. If you have a long query, then it could take three or four round trips across the network to get the entire query to the database server. That's not including the time it takes the server to put the query back together and run it, or considering that the query may run several -- or several hundred -- times a second.

Using a stored procedure will greatly reduce that traffic because the stored procedure call will always be much shorter. Also, stored procedures are easier to trace in Profiler or any other tool. A stored procedure is an actual object in your database. That means it's much easier to get performance statistics on a stored procedure than on an ad-hoc query and, in turn, find performance issues and draw out anomalies.

In addition, stored procedures parameterize more consistently. This means you’re more likely to reuse your execution plans and even deal with caching issues, which can be difficult to pin down with ad-hoc queries. Stored procedures also make it much easier to deal with edge cases and even add auditing or change-locking behavior. A stored procedure can handle many tasks that trouble ad-hoc queries. My wife unraveled a two-page query from Entity Framework a couple of years ago. It took 25 minutes to run. When she boiled it down to its essence, she rewrote that huge query as SELECT COUNT(\*) from T1. No kidding.

OK, I kept it as short as I could. Those are the high-level points. I know many .Net coders think that business logic doesn’t belong in the database, but what can I say other than you’re outright wrong. By putting the business logic on the front end of the application, you have to bring all of the data across the wire merely to compare it. That’s not good performance. I had a client earlier this year that kept all of the logic out of the database and did everything on the front end. The company was shipping hundreds of thousands of rows of data to the front end, so it could apply the business logic and present the data it needed. It took 40 minutes to do that. I put a stored procedure on the back end and had it call from the front end; the page loaded in three seconds.

Of course, the truth is that sometimes the logic belongs on the front end and sometimes it belongs in the database. But ORMs always get me ranting.

**6. Don’t do large ops on many tables in the same batch**

This one seems obvious, but apparently it's not. I’ll use another live example because it will drive home the point much better. I had a system that suffered tons of blocking. Dozens of operations were at a standstill. As it turned out, a delete routine that ran several times a day was deleting data out of 14 tables in an explicit transaction. Handling all 14 tables in one transaction meant that the locks were held on every single table until all of the deletes were finished. The solution was to break up each table's deletes into separate transactions so that each delete transaction held locks on only one table. This freed up the other tables and reduced the blocking and allowed other operations to continue working. You always want to split up large transactions like this into separate smaller ones to prevent blocking.

**7. Don't use triggers**

This one is largely the same as the previous one, but it bears mentioning. Don’t use triggers unless it’s unavoidable -- and it’s almost always avoidable.

The problem with triggers: Whatever it is you want them to do will be done in the same transaction as the original operation. If you write a trigger to insert data into another table when you update a row in the Orders table, the lock will be held on both tables until the trigger is done. If you need to insert data into another table after the update, then put the update and the insert into a stored procedure and do them in separate transactions. If you need to roll back, you can do so easily without having to hold locks on both tables. As always, keep transactions as short as possible and don’t hold locks on more than one resource at a time if you can help it.

**8. Don’t cluster on GUID**

After all these years, I can't believe we’re still fighting this issue. But I still run into clustered GUIDs at least twice a year.

A GUID (globally unique identifier) is a 16-byte randomly generated number. Ordering your table’s data on this column will cause your table to fragment much faster than using a steadily increasing value like DATE or IDENTITY. I did a benchmark a few years ago where I inserted a bunch of data into one table with a clustered GUID and into another table with an IDENTITY column. The GUID table fragmented so severely that the performance degraded by several thousand percent in a mere 15 minutes. The IDENTITY table lost only a few percent off performance after five hours. This applies to more than GUIDs -- it goes toward any volatile column.

**9. Don’t count all rows if you only need to see if data exists**

It's a common situation. You need to see if data exists in a table or for a customer, and based on the results of that check, you’re going to perform some action. I can't tell you how often I've seen someone do a SELECT COUNT(\*) FROM dbo.T1 to check for the existence of that data:

SET @CT = (SELECT COUNT(\*) FROM dbo.T1);  
If @CT > 0  
BEGIN  
<Do something>  
END

It’s completely unnecessary. If you want to check for existence, then do this:

If EXISTS (SELECT 1 FROM dbo.T1)  
BEGIN  
<Do something>  
END

Don’t count everything in the table. Just get back the first row you find. SQL Server is smart enough to use EXISTS properly, and the second block of code returns superfast. The larger the table, the bigger difference this will make. Do the smart thing now before your data gets too big. It’s never too early to tune your database.

In fact, I just ran this example on one of my production databases against a table with 270 million rows. The first query took 15 seconds, and included 456,197 logical reads, while the second one returned in less than one second and included only five logical reads. However, if you really do need a row count on the table, and it's really big, another technique is to pull it from the system table. SELECT rows from sysindexes will get you the row counts for all of the indexes. And because the clustered index represents the data itself, you can get the table rows by adding WHERE indid = 1. Then simply include the table name and you're golden. So the final query is SELECT rows from sysindexes where object\_name(id) = 'T1' and indexid = 1. In my 270 million row table, this returned sub-second and had only six logical reads. Now that's performance.

Popular on InfoWorld

* [](http://www.infoworld.com/article/2686458/smartphones/iphone-6-review-the-little-things-make-it-a-real-star.html)

[iPhone 6 review: The little things make it a real star](http://www.infoworld.com/article/2686458/smartphones/iphone-6-review-the-little-things-make-it-a-real-star.html)

* [](http://www.infoworld.com/article/2840235/application-development/9-cutting-edge-programming-languages-worth-learning-next.html)

[9 cutting-edge programming languages worth learning now](http://www.infoworld.com/article/2840235/application-development/9-cutting-edge-programming-languages-worth-learning-next.html)

* [](http://www.infoworld.com/article/2834535/security/four-more-botched-black-tuesday-patches-kb-3000061-kb-2984972-kb-2949927-and-kb-2995388.html)

[Four more botched Microsoft patches: KB 3000061, KB 2984972, KB 2949927, KB...](http://www.infoworld.com/article/2834535/security/four-more-botched-black-tuesday-patches-kb-3000061-kb-2984972-kb-2949927-and-kb-2995388.html)

**10. Don’t do negative searches**

Take the simple query SELECT \* FROM Customers WHERE RegionID<> 3. You can’t use an index with this query because it’s a negative search that has to be compared row by row with a table scan. If you need to do something like this, you may find it performs much better if you rewrite the query to use the index. This query can easily be rewritten like this:

SELECT \* FROM Customers WHERE RegionID< 3 UNION ALL SELECT \* FROM Customers WHERE RegionID

This query will use an index, so if your data set is large it could greatly outperform the table scan version. Of course, nothing is ever that easy, right? It could also perform worse, so test this before you implement it. There are too many factors involved for me to tell you that it will work 100 percent of the time. Finally, I realize this query breaks the “[no double dipping](http://www.infoworld.com/article/2628420/database/7-performance-tips-for-faster-sql-queries.html)” tip from the last article, but that goes to show there are no hard and fast rules. Though we're double dipping here, we're doing it to avoid a costly table scan.

OK, there you go. You won’t be able to apply all of these tips all of the time, but if you keep them in mind you’ll find yourself using them as solutions to some of your biggest issues. The most important thing to remember is not to take anything I say as the gospel and implement it because I said so. Test everything in your environment, then test it again. The same solutions won’t work in every situation. But these are tactics I use all the time when addressing poor performance, and they have all served me well time and again.