

"How To" Guides for databases in general

"How To" Guides for specific database management systems (DBMS)

How To Guides for IBM DB2

How To Guides for MySQL

How To Guides for Oracle

How To Guides for Microsoft SQL Server

SQL

Data Types

Tasks

Automatic padding of character columns

When considering whether to use CHAR or VARCHAR, the decision depends more on programming considerations than on the database. The difference is that comparisons of two fixed character fields will be automatically padded to equal lengths with spaces; comparisons with VARCHAR will not.

Why not just make everything VARCHAR?

In order to save time on the database design, you might be tempted to avoid using CHAR columns and simply use VARCHAR for all character data in the database. As a result, every time one of these columns is compared to a fixed character column or host variable, the programmers will have to use an RPAD or RTRIM function, since programming languages won't do automatic padding on VARCHAR columns. Because the database cannot simply search the values in the index when a function is applied to a key column queries may unexpectedly take much longer to complete. So the time that was saved in the design phase will probably be spent many times over trying to determine why certain queries are running so poorly.

Fixed vs. variable length program variables

One reason that the VARCHAR data type is often used is that some programmers by habit use variable length character variables, which will also defeat the programming language's automatic padding for fixed-length comparisons. For example, in Java, programmers might simply use String, which is variable length, for all character variables and avoid char[], which is the appropriate type for storing data being transferred to or from a CHAR column in a database.

When to use CHAR vs. VARCHAR

In database design, we're taught how to determine whether to make a column CHAR or VARCHAR. You ask yourself, "is the length of the value significant?" For example, if the column has up to 6 characters representing a gender, is a 6-character "MALE " different from a 4-character "MALE"? In this case the answer is obviously "no" and the column should therefore be CHAR so that regardless of whether you compare it to a 4- or 6-character constant or

6-character host variable, the comparison is automatically padded and works properly regardless of whether the column contains "MALE " or "FEMALE" and whether the constant or host variable contains "MALE", "MALE " or "FEMALE". If either side of the comparison is a variable length character type, then you will need to RPAD and/or RTRIM one or both sides of the comparison to make it work properly.

Avoid functions on index columns

Applying a function to a key column will prevent the >DBMS from being able to search the index using the column value. For example, given these two queries to retrieve a particular row:

```
select * from t where keycol = :hostvar;  
select * from t where rtrim(keycol) = :hostvar;
```

The first query can use the index to find the row very quickly. For the second query, the database must retrieve every value of the key column from either the table or the index, in order to apply the function to those values.

If possible, move the function to the other side of the expression:

```
select * from t where keycol = rpad(:hostvar, n);
```

When to use a function index

Some databases allow creating an index on a function of a column. Using a function index becomes necessary when there is no inverse function that allows moving the function to a non-indexed value. For example, most databases always use case-sensitive comparisons, and there is no inverse function for a lower function. Say an application requires a query that ignores case and padding, as in:

```
select ... from schema.tablename where lower(trim(email_address)) = lower(trim(:emailaddr));
```

To avoid a table or index pagespace scan to convert all of the email_address values to lower case, a function index could be created on that column.

```
create index schema.indexname  
on schema.tablename (  
lower(trim(email_address))  
) ...;
```

Watch out for implied coercion of key columns

Implied coercion of key columns can result in the same type of problems that are caused by explicit functions. For example, when the fixed length CHAR data type is being used, one side of a comparison will be padded automatically if they are different lengths. Therefore, a key column should be compared only with constants or variables of an equal or shorter length. Here are some examples that use the same CHAR(6) "gender" column that was used in the discussion of when to use CHAR vs. VARCHAR. These examples will not cause automatic padding of the GENDER column:

```
GENDER = 'MALE'  
GENDER = 'MALE '  
GENDER = 'FEMALE'
```

These examples should be avoided because they will cause automatic padding of the GENDER column to expand its length from 6 to 7 characters:

```
GENDER = 'UNKNOWN'  
GENDER = 'FEMALE '
```

Index