Relational Data Model Best Practices

Tom Wilson

March 31, 2019

Don’t use SQL like a simple access method. (Page 16)

```
do for all tuples in S ;
   fetch S tuple into TS , TN , TT , TC ;
   do for all tuples in SP with SNO = TS ;
      fetch SP tuple into TS , TP , TQ ;
      emit TS , TN , TT , TC , TP , TQ ;
   end do ;
end do ;
```

vs

```
S JOIN SP
```

Avoid the use of any SQL construct that references physical access paths such as indexes. (Page 17)

the separation of model and implementation that allows us to achieve physical data independence

physical data independence means protecting investment in user training and applications

Don't use table to mean a base table specifically unless your intended meaning is clear from the context. (Page 24)

Don't think of views as if they were somehow different from tables. (Page 24)

Avoid coercions wherever possible. (Page 62)

Ensure that columns with the same name are of the same type. (Page 62)

Avoid type conversions where possible. When they can't be avoided, do them explicitly if you can. (Page 62)

Don’t use PAD SPACE. (Page 64)

Avoid possibly nondeterministic expressions. (Page 65)

Don’t use “typed tables,” reference values, REF types, or any SQL construct related to these features. (Page 67)

If you must talk about nulls, call them nulls and not “null values.” (Page 84)

Don't use the comparison operators "$<$", "$<=$", "$>$", and "$>=$" on rows of degree greater than one. (Page 88)

Use AS specifications whenever necessary (and possible) to give proper column names to columns that otherwise (a) wouldn't have a name at all or (b) would have a name that wasn't unique. (Pages 97, 179)

If two columns represent the same kind of information, give them the same name wherever possible. (Page 98)

Never write code that relies on left to right column positioning. (Pages 99-100)

Avoid duplicates. Make sure you know when SQL eliminates duplicates without you asking it to; when you do have to ask, make sure you know whether it matters if you don't; when it does matter, specify DISTINCT; and never specify ALL. (Page 120)

Avoid nulls: (Page 125)

# Specify NOT NULL, explicitly or implicitly, for every column in every base table.

# Don't use the keyword NULL anywhere other than in the context of such a NOT NULL specification.

# Don’t use the keyword UNKNOWN in any context whatsoever.

# Don’t omit the ELSE clause from a CASE expression unless omitting it makes no logical difference.

# Don’t use NULLIF.

# Don't use the keywords OUTER, FULL, LEFT, and RIGHT on JOIN (except, just possibly, in connection with COALESCE).

# Don't use union join.

# Don’t specify PARTIAL or FULL on MATCH.

# Don’t use MATCH on foreign key constraints.

# Don’t use IS DISTINCT FROM.

# Don’t use IS TRUE, IS NOT TRUE, IS FALSE, or IS NOT FALSE.

# Do use COALESCE on every scalar expression that might “evaluate to null” without it.

Don't use DELETE or UPDATE through a cursor unless you can be certain that integrity constraint problems will never arise in connection with such use. (Page 143)

Avoid operations that are inherently row level (e.g., row level triggers). (Pages 144, 155-156)

Specify target columns explicitly on INSERT. (Page 148)

Don't define as a key some column combination that you know to be reducible. (Page 150)

Use UNIQUE and/or PRIMARY KEY specifications to ensure that every base table has at least one key. (Page 151)

Ensure that foreign key columns have the same name as the corresponding key columns wherever possible. (Page 154)

Don’t use triggers if they violate The Assignment Principle. (Page 155)

Don't use any operation that violates the relational closure property if you want the result to be amenable to further relational processing. (Page 177)

Use NATURAL JOIN in preference to other methods of formulating a join (but make sure columns with the same name are of the same type). (Page 186)

If you use JOIN ON, make sure columns with the same name are of the same type, and make sure you rename columns appropriately. (Page 186)

If you use JOIN USING, make sure columns with the same name are of the same type. (Page 187)

If you use CROSS JOIN, make sure there aren't any common column names. (Page 187)

For UNION, INTERSECT, and EXCEPT, make sure corresponding columns have the same name and type. (Page 188)

For UNION, INTERSECT, and EXCEPT, always specify CORRESPONDING if possible. If it isn't possible, then make sure columns line up properly. Preferably avoid use of the BY option, unless it makes no difference anyway. (Page 188)

If you use GROUP BY or HAVING, make sure the table you're summarizing is the one you really want to summarize. (Page 242)

Be on the lookout for the possibility that some summarization is being done on an empty set, and use COALESCE wherever necessary. (Page 242)

Where possible, use database constraints to make up for SQL's lack of support for type constraints. (Page 287)

Specify constraints declaratively whenever you can. (Page 294)

Use immediate constraint checking whenever you can.
(Page 301)

If checking has to be deferred on some constraint, make sure the check is done before doing any operation that might rely on the constraint being satisfied. (Page 301)

In CREATE VIEW, don't use the option that allows you to specify the view column names immediately following the view name itself. (Page 325)

Specify WITH CASCADED CHECK OPTION on view definitions whenever possible. (Pages 325, 341)

Specify constraints that apply to views (e.g., key constraints) in the form of comments— typically on the view definition. (Page 332)

Never use the term view, unqualified, to mean a snapshot; never use the term materialized view; and watch out for violations of these recommendations on the part of others. (Page 351)

Be careful over the use of COUNT; in particular, don't use it where EXISTS would be more logically correct. (Page 396)

Use the techniques described in Chapter 11, at least for formulating “complex” SQL expressions. (Pages 412ff)

Don’t use ALL or ANY comparisons. (Page 433)

Don’t use “SELECT *” at the outermost level in a cursor definition or view definition, or more generally in any position where the meaning of that “*” might be subject to change. (Page 444)

Favor the use of explicit range variables, especially in "complex" expressions. (Page 448)