Selecting One Row From a Database Table

Horst Keller

This one might be trivial, but interestingly there are quiet some discussions about it. Therefore, let's wrap it up.

Selecting a Fully Specified Row

For selecting a fully specified row with Open SQL SELECT, you specify the full table key in equals conditions combined by AND.
The natural result set of an SQL statement is tabular. And so, the natural way of writing a SELECT statement also in Open SQL is:

```
SELECT *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS' AND
             msgnr = '050'
       INTO TABLE @DATA(result).
cl_demo_output=>display( result ).
```

This is independent from the number of rows expected. Of course, by specifying the full key, the result set contains only one row. Assigning the result set to internal table result of course gives an internal table with one line.
So far so good. In languages dealing with tables only, as.e.g. SAP HANA's SQLScript, this is also the only way and you always receive a result table with one line. But in ABAP, we additionally have structures. Structures are even older than internal tables. So you can write the result of a single row selection also into a structure:

```
SELECT *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS' AND
             msgnr = '050'
       INTO  @DATA(result).
ENDSELECT.
cl_demo_output=>display( result ).
```

Without specifying TABLE, result is a structure and a SELECT loop is opened that must be closed with ENDSELECT. That is an ABAP speciality of course. Now you might ask why should I open a loop, if I know that I want to read only one row into a structure. And that's exactly what the ABAP speciality SELECT SINGLE is made for. Nothing more and nothing less.

```
SELECT SINGLE *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS' AND
             msgnr = '050'
       INTO  @DATA(result).
cl_demo_output=>display( result ).
```

Same result as above. The native SQL generated from that syntax and passed to the database is also the same. There is a very small performance improvement with SELECT SINGLE, because no loop has to be opened, but normally that can be neglected. SELECT SINGLE has shorter syntax and documents the semantics: You know, that you want to read a fully specified row into a structure and you use SELECT SINGLE for that purpose. The syntax documents the semantics of your statement in the program and the extended program check warns you, if you do not specify the full key.

Selecting a Partly Specified Row

If you do not specify the full key, the result set delivered by the database normally contains multiple rows:

```
SELECT *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       INTO TABLE @DATA(result).
cl_demo_output=>display( result ).
```

The internal table result contains mulitple lines now. How can I get one line instead? You know the answer:

```
SELECT *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       INTO TABLE @DATA(result)
       UP TO 1 ROWS.
cl_demo_output=>display( result ).
```

With UP TO 1 ROWS you tell the database to pass only one row in its tabular result set and the internal table contains only one line. But be aware, that the returned row is not defined. It can be any of those specified by the partial key. In order to get a defined row in respect to the sort order, you can add an ORDER BY clause.

```
SELECT *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       ORDER BY PRIMARY KEY
       INTO TABLE @DATA(result)
       UP TO 1 ROWS.
cl_demo_output=>display( result ).
```

Of course, you can also select into a structure:

```
SELECT *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       ORDER BY PRIMARY KEY
       INTO @DATA(result)
       UP TO 1 ROWS.
ENDSELECT.
cl_demo_output=>display( result ).
```

Since UP TO always creates a tabular result set, you must use ENDSELECT.
What about SELECT SINGLE? Yeah, you can use that too:

```
SELECT SINGLE *
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       INTO @DATA(result).
cl_demo_output=>display( result ).
```

Again a single row in a structure. But this can be seen as a misuse of SELECT SINGLE. Especially, you cannot use ORDER BY in connection with SINGLE. Simply, because SELECT SINGLE is not made for this. Therefore, the returned row is always undefined. If you are interested in the contents of the single row, SELECT SINGLE should not be used with partial key specifications.

Checking the Existence of a Row

A widely discussed question is, how to check the existence of a row or rows in a database table. Especially, if you specify a partial key or even non key fields only, you want to restrict the number of lines transferred from the database to the application server as much as possible. Before 7.40, SP05, the minimal numbers of rows to be selected was 1. You can use either `SELECT SINGLE` or `UP TO 1 ROWS` **in order to restrict the number of rows to that minimum. Since you only check sy-subrc and you are not interested in the contents of the returned row, you don't have to care about** `ORDER BY`**. So both forms can be used:**

```
SELECT SINGLE sprsl
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       INTO @DATA(result).
IF sy-subrc = 0.
  …
ENDIF.

SELECT sprsl
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       INTO @DATA(result)
       UP TO 1 ROWS.
ENDSELECT.
IF sy-subrc = 0.
  …
ENDIF.
```

The syntax of SELECT SINGLE is shorter. And performance? On the database both variants take the same time because the same native code is executed there. All in all SELECT SINGLEis even a little bit faster than UP TO 1 ROWS (no loop opened), but that is negligible. Therefore, no problem in "misusing" SELECT SINGLE for merely checking the existence of rows. In that case, you know what you do and you can hide the warning of the extended program check with pragma ##WARN_OK.
With 7.40, SP05, you can use the following syntax:

```
SELECT SINGLE @abap_true
       FROM t100
       WHERE sprsl = @sy-langu AND
             arbgb = 'SABAPDEMOS'
       INTO @DATA(result).
IF sy-subrc = 0.
…
ENDIF.
```

The constant abap_true is used as a minimal SQL expression. Now, not even a single row must be transported from the database to the application server. You can also check IF result = abap_true.