Matching Data for Analysis

Rickard Olsson

95 97 2000 2002 DOWNLOAD

In this article, Rickard Olsson shows how to compare rows in SQL by loading the desired data into two tables for easy comparison. In fact, he shows two different methods and tries to figure out which method will give the best performance.

I was inspired by Mike Westcott's article in the January 2002 issue about "Ordered Calculations with SQL." He ended up with a philosophy about taxi drivers that stressed why we have to accept that handling comparisons between rows using SQL isn't problem-free. I found myself with exactly this problem with one of my customers, and I had to solve the comparison problem reliably. If I couldn't do it, the application would be worthless.

My customer was teaching air-traffic controllers using a Unix-based simulator that re-creates the work environment for air-controllers. They also use the simulator for creating situations that don't yet exist but will arise (for example, adding a third runway at Arlanda in Stockholm or increasing the traffic load in Landvetter, Sweden). The simulator logs everything that happens in a simulation exercise and loads all logged data into a large Oracle database, running on Unix.

One of the most important factors that influences the stress put on controllers is how many radio calls they have to make (and how long each takes) to carry out their mission. The data produced by the simulator (radio call start time, radio call end time) isn't much help in analyzing the level of stress. My clients needed more analysis than the software provided.

My clients wanted a fast way of getting the simulation results presented in a graphical format that could be shown to the people involved in the exercise. At the debriefing session held after each exercise, the participants discuss the outcome of the exercise. They wanted some facts to be added to their subjective experiences. I suggested an Access solution with an interface that would support simple selection and grouping of data. My client was expecting my solution to have a response time in the range of one to two seconds to produce the graph so that drawing on the numbers wouldn't slow the discussion. **Figure 1** shows the kind of

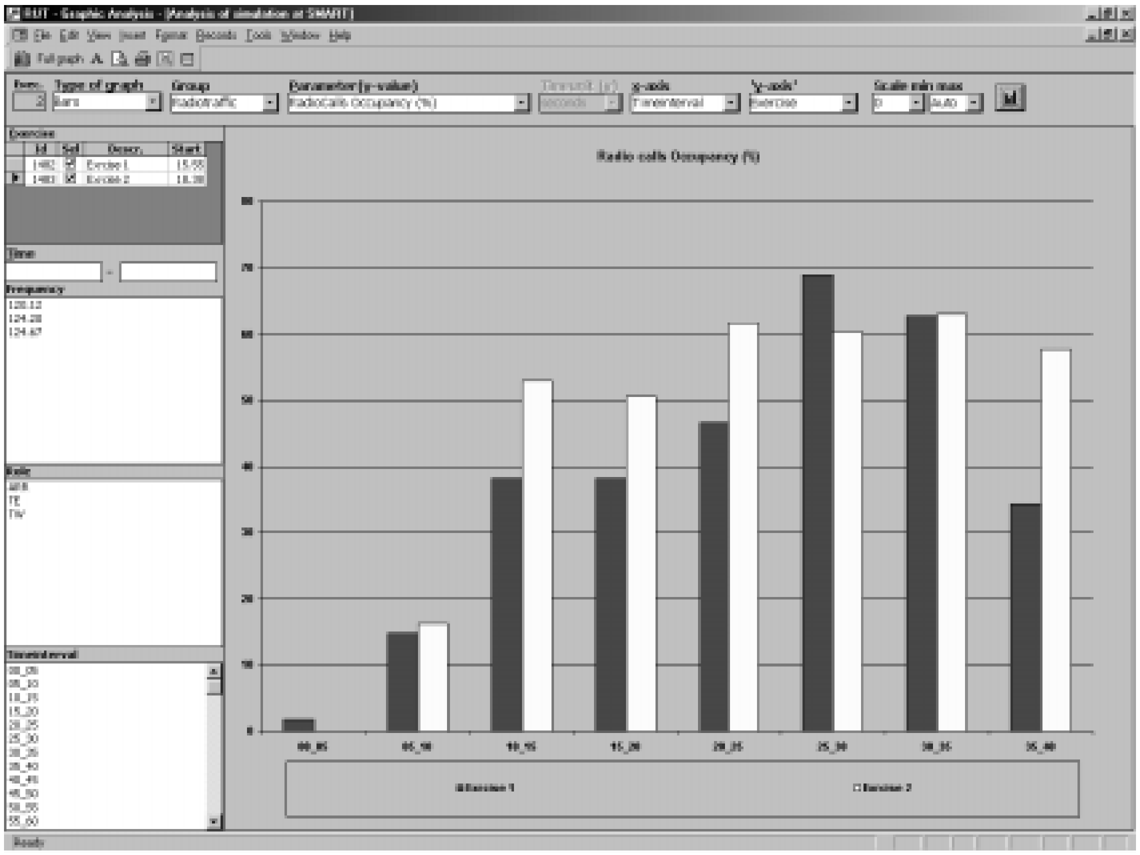


Figure 1. This is what they want.

information that my client wanted. Providing that analysis using SQL is possible, but it's not simple SQL, which is why I was called in.

After initial trials using ODBC against the Oracle database, it was clear that I wouldn't get the desired performance by querying Oracle. Not even pass-through queries gave us the desired performance. Part of the problem was that the Oracle database contained a lot of irrelevant information from older exercises that the client wouldn't use in this analysis scenario. In addition, the indexes in the database weren't what we needed for the analysis that we'd be doing. I wasn't supposed to change the design of the Oracle database (for instance, by adding new indexes) since the database belonged to a system developed by a company in England. The obvious solution in this situation was to offer the users an initial selection menu where they could select the desired exercises and then load the selected data into an mdb where it could be analyzed.

Creating tables

What you see in **Figure 2** is a view of the output of the two Oracle tables that I needed (Exercise_Event and Communication_Event). Each exercise has a unique counter field, and each logged event has a unique counter. Since it's not possible for an operator to have two radio calls at the same time on the same frequency, it's easy to arrange the events in the order they occurred by sorting on Exercice_Id, Frequency, Role, and Event_No. I put the rows with START_RADIO_TX in the Event_type fields into one work table (RadioStart) and the STOP_RADIO_TX records into another work table (RadioEnd). I then sorted the rows as I described to make it easy to pair the start and stop of each call. With this arrangement, it's possible to calculate the duration of each call. One other note: This design also assumes that it's not possible to end a radio call before it starts, which is a fairly reasonable assumption.

The simulation, for a variety of reasons, also generated calls with a duration of zero. I ignored these results as I loaded my work tables. It does mean that, on some occasions, the number of rows in the input tables exceeded the number of rows in the work tables.

So what I needed was a field in each of the two work tables that would let me match the corresponding start and end events for the calls. Effectively, this field would be a counter, identifying the start and end times for the various calls (for instance, the start and end times for the first call, the second call, and so on). There were two ways to achieve this:

- Use a Make Table query to initially load the table and then an Alter Table query to add the values for the matching field.
- Use Delete to clear out a set of existing tables and then Append to add the table. This table would have a counter field defined.

The problem with the Delete and Append solution is that the counter

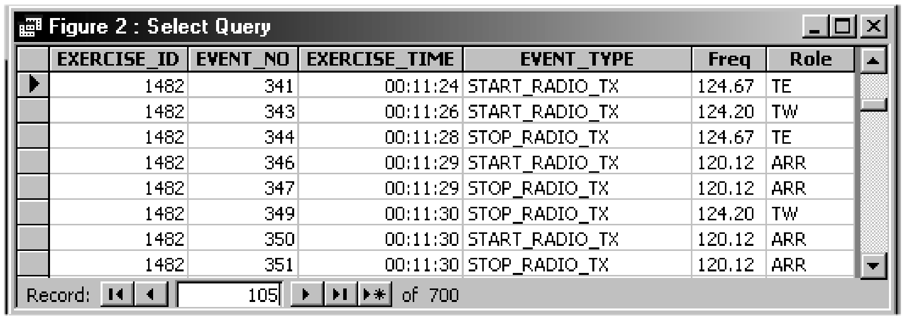
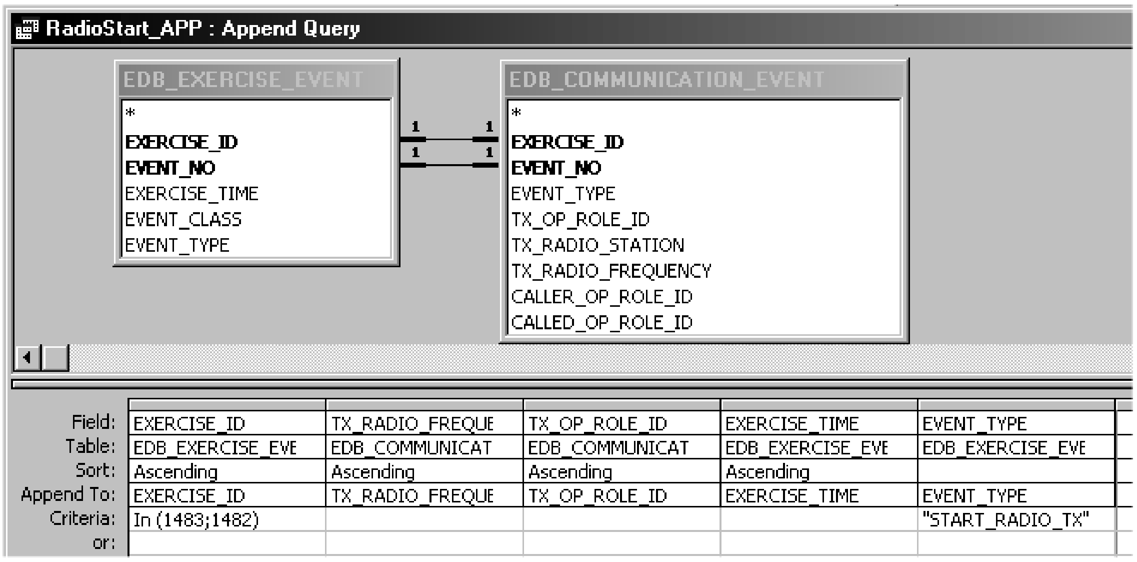


Figure 2 : Select Query

EXERCISE_ID	EVENT_NO	EXERCISE_TIME	EVENT_TYPE	Freq	Role
1482	341	00:11:24	START_RADIO_TX	124.67	TE
1482	343	00:11:26	START_RADIO_TX	124.20	TW
1482	344	00:11:28	STOP_RADIO_TX	124.67	TE
1482	346	00:11:29	START_RADIO_TX	120.12	ARR
1482	347	00:11:29	STOP_RADIO_TX	120.12	ARR
1482	349	00:11:30	STOP_RADIO_TX	124.20	TW
1482	350	00:11:30	START_RADIO_TX	120.12	ARR
1482	351	00:11:30	STOP_RADIO_TX	120.12	ARR

Record: 105 of 700

Figure 2. Sample of data collected by simulator.



RadioStart_APP : Append Query

EDB_EXERCISE_EVENT: *, EXERCISE_ID, EVENT_NO, EXERCISE_TIME, EVENT_CLASS, EVENT_TYPE

EDB_COMMUNICATION_EVENT: *, EXERCISE_ID, EVENT_NO, EVENT_TYPE, TX_OP_ROLE_ID, TX_RADIO_STATION, TX_RADIO_FREQUENCY, CALLER_OP_ROLE_ID, CALLED_OP_ROLE_ID

Field:	EXERCISE_ID	TX_RADIO_FREQUE	TX_OP_ROLE_ID	EXERCISE_TIME	EVENT_TYPE
Table:	EDB_EXERCISE_EVE	EDB_COMMUNICAT	EDB_COMMUNICAT	EDB_EXERCISE_EVE	EDB_EXERCISE_EVE
Sort:	Ascending	Ascending	Ascending	Ascending	
Append To:	EXERCISE_ID	TX_RADIO_FREQUE	TX_OP_ROLE_ID	EXERCISE_TIME	EVENT_TYPE
Criteria:	In (1483;1482)				"START_RADIO_TX"
or:					

Figure 3. Append query to collect RadioStart events.

won't be automatically reset to 1 when the rows in the table are deleted. If you're using Access 2000 or later, you can use a SQL Alter statement to set the identity value on an existing table:

```
Alter Table AutoTable & _
 Alter Column RowNo Identity (1, 1)
```

With earlier versions of Access, compacting the database would reset the autonumber fields. I solved the problem a third way by creating an empty template for my work tables and copying that template to make the two work tables.

In **Figure 3** (on page 15), you can see the Append query that fills the RadioStart table with data from the two Oracle tables. As you can see, the RadioStart table has a counter field called RowNo defined, which will be filled with what I call a "trustworthy counter" (see **Figure 4**). The counter is an ordinary field rather than an autonumber field. The Order By clause in the Append query that loads the table should allow you to depend on the order that records are inserted into the work table (at least with Jet).

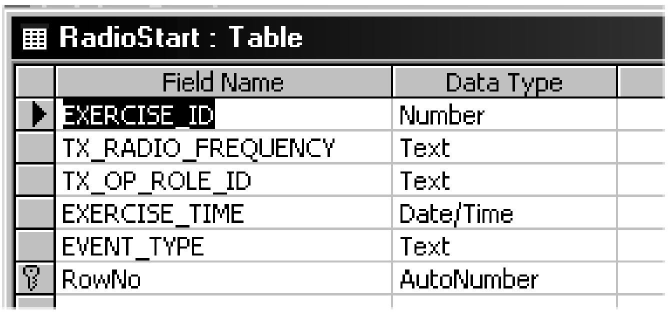
By doing a similar Append query on the ending events, I have two tables filled with the start and end events on the same row for each call. A simple join on the counters will pair the correct rows and calculate the duration of the radio calls by using the DateDiff function:

```
Select EXERCISE_ID, TX_OP_ROLE, EXERCISE_TIME,
       TX_RADIO_FREQ,
       DateDiff('s',[RadioStart].[EXERCISE_TIME],
                [RadioEnd].[EXERCISE_TIME]
    From RadioStart Inner Join RadioEnd
           On RadioStart.RowNo = RadioEnd.RowNo
```

Which method?

As I described earlier, there are two methods of loading the tables—RadioStart and RadioEnd—which raises the question of which one to use. Both techniques are equally complicated from a maintenance point of view.

To make the decision, I generated some code to see whether the two methods differ in some important characteristic—response time, for instance. The Method_1 function uses the Make Table method:



RadioStart : Table

	Field Name	Data Type
▶	EXERCISE_ID	Number
	TX_RADIO_FREQUENCY	Text
	TX_OP_ROLE_ID	Text
	EXERCISE_TIME	Date/Time
	EVENT_TYPE	Text
🔑	RowNo	AutoNumber

Figure 4. Field definition in the RadioStart table.

```
Public Sub Method_1( _
    Optional InList As Variant = "1482,1483")
Dim db As DAO.Database
    TimeBetween "Method_1 Start", False
    Set db = CurrentDb
    Drop_Table "RadioStart"
    DoCmd.CopyObject , "RadioStart", _
                acTable, "Radio_Template"

    Modify_Template "RadioStart_APP", InList
    db.Execute "RadioStart_APP"

    Drop_Table "RadioEnd"
    DoCmd.CopyObject , "RadioEnd", _
                    acTable, "Radio_Template"
    Modify_Template "RadioEnd_APP", InList
    db.Execute "RadioEnd_APP"
    TimeBetween "Method_1 End"
End Sub
```

The preceding code calls some standard functions: TimeBetween, Drop_Table, and Modify_Template. TimeBetween is used to show elapsed time in the debug window. It's called at the entrance and the exit of the procedure being timed:

```
Sub TimeBetween(p,_
          Optional Disp As Boolean = True)
    Static TPrev, PPrev
    If Disp Then
      Debug.Print PPrev & "-" & p & _
                  ":", Timer - Tprev
    End If
    TPrev = Timer
    PPrev = p
End Sub
```

Drop_Table drops a table from the current database if the table exists (and does nothing if the table doesn't exist). Modify_Template is used to modify the Where clause in the query that extracts the data from the joined tables in Oracle. This allows me to extract only the events from the last simulation and ignore irrelevant data.

The Method_2 function uses my other technique: Create the table, load it, and then add the RowNo column using an Alter Table command like this one:

```
ALTER TABLE RadioStart ADD COLUMN RowNo Counter
    CONSTRAINT PrimaryKey PRIMARY KEY
```

The same functions used for the Method_1 function are used in the Method_2 function:

```
Public Sub Method_2( _
     Optional InList As Variant = "1482,1483")
Dim db As DAO.Database
    TimeBetween "Method_2 Start", False
    Set db = CurrentDb
    Modify_Template "RadioStart_CRE", InList

    Drop_Table "RadioStart"
    db.Execute "RadioStart_CRE"
    db.Execute "RadioStart_Add_RowNo"

    Drop_Table "RadioEnd"
    Modify_Template "RadioEnd_CRE", InList
    db.Execute "RadioEnd_CRE"
    db.Execute "RadioEnd_Add_RowNo"
    TimeBetween "Method_2 End"
End Sub
```

Both methods were clocked on my 366Mhz Dell

Inspiron 7000 (running Win 2000 Professional as its operating system and Access 2000 SR-1). I tested with two different sets of Jet input tables as shown in **Table 1**. Test 1 had fewer lines in the input tables than Test 2, but both gave the same number of rows in the work table.

Table 1. Test results for a single exercise.

Table/Method	Test 1	Test 2	Oracle
Communication_Event	700	36 649	300 000
Exercise_Event	11 662	109 706	950 000
Collected rows	677	677	677
Method_1,elapsed (s)	0.52-0.53	0.91-0.93	No data
Method_2,elapsed (s)	0.41-0.43	0.71-0.74	No data

Table 1 shows that Method_2 seems slightly faster than Method_1. But what happens if the number of rows collected is something other than 677? **Table 2** shows the results that I got by running the test three times for different numbers of output records and taking an average of the runs. **Figure 5** shows the results as an Excel graphic. These results seem to indicate that Method_2 is the better choice if many rows are collected.

Table 2. Test results by record count.

Collected rows	Method_1 (s)	Method_2 (s)
0	0.13	0.09
338	0.61	0.50
677	0.93	0.77
1,430	2.05	1.73
5,600	5.50	3.84
9,939	8.70	5.45
14,773	13.30	8.25
18,216	16.10	9.70

If you let Excel make a regression line (a linear function seems a reasonable choice) for the two methods, you'll receive the following formulas for the response time in seconds as a function of collected rows:

- Method_1: RT(s) = 0.41 + 0.0009n
- Method_2: RT(s) = 0.51 + 0.0005n

Runtime isn't everything, though. If you look at the growth of the mdb file (for example, look at the "holes" produced by deleting a table and needing a Compact to remove), you'll find Method_2 is the better choice here also. So, based on this analysis, it seems obvious (at least on my computer) to use the Make Table and Alter Table approach. At my customer, though, Method_1 is the one actually running because, during the development, I didn't have the time to make this analysis. I just used my best judgment and guessed that Method_1 was the best choice.

The problem of calculating durations between different rows in SQL often occurs when analyzing computer performance. In a system that I developed for analyzing performance problems in Access, I went into a very troublesome analysis working with a log file that showed start and end times for different procedures that called each other. In that situation, I had to take into account the execution stack and its depth, which almost drove me crazy when trying to analyze the data. Since I was the one producing the log file, my solution at that time was to put some more logic into the logging functions so the problems could be avoided when analyzing the data. As always, in real life, the best problem-solving method is to avoid problems. If that doesn't work, you could always try to imagine you don't see the problem. Being lazy is often smart. ▲

DOWNLOAD **COMPARE.ZIP at www.smartaccessnewsletter.com**

Rickard Olsson is a senior database specialist in Malmoe, Sweden, with experience with databases dating to 1975. Until the beginning of the 1990s, he did most of his work with IBM mainframes using tools like APL, DL/I, DB2, and Focus. Since Access 2 arrived, most of his time has been spent with Windows and Office products, since all of his customers moved to the PC environment. Rickard holds an MCP in both Access and VB and is working on his MCP for SQL Server. Since 1985, he's run a small IT consultancy company that serves customers who want to make their work with their computers more useful. When not working, he spends most of his time with his wife at their cottage in the archipelago of Karlskrona in the south of Sweden. www.ricol.se, Ricol@sbbs.se.

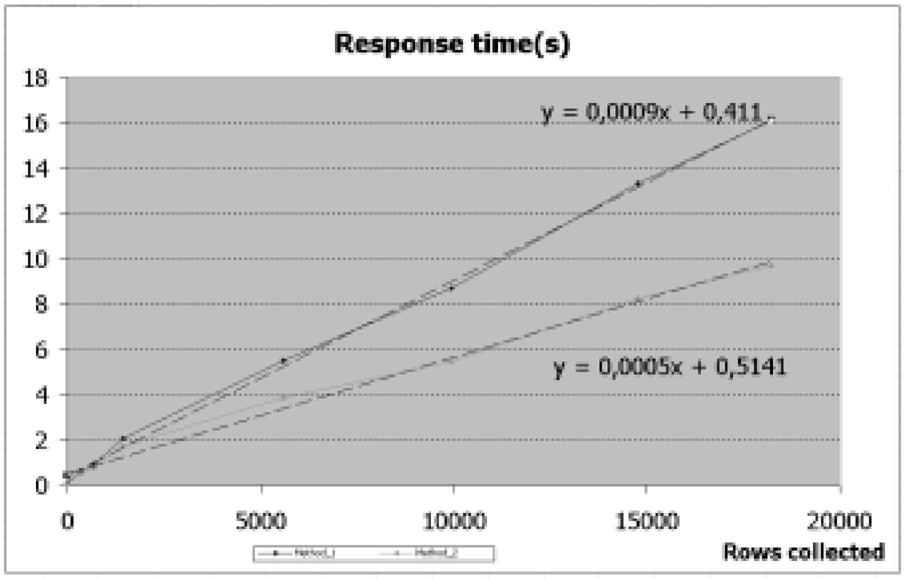


Figure 5. Elapsed time as a function of collected rows.