Using a database checkpoint

We have seen in the previous recipes how to connect and perform SQ queries programmatically. This knowledge is essential because it can give us more flexibility as to how to access our DB and how to proce the retrieved data. However, UFT provides an in-built feature to perfo DB checkpoints, which can be very useful especially when the perso implementing the automated tests is less skilled in coding.

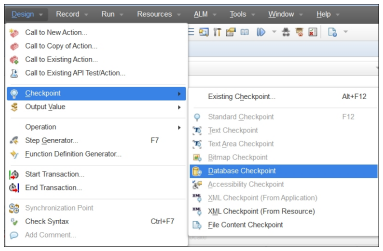
Getting ready

We will use the `flight32.mdb` file supplied with the Flight application, which we used in Chapter 1, *Data-driven Tests*. Make sure that the Microsoft Query application is installed on your machine.

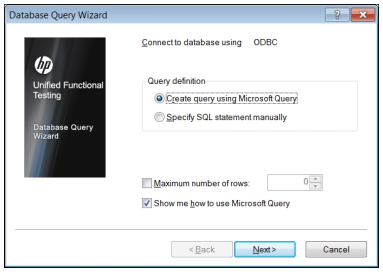
How to do it...

Proceed with the following steps:

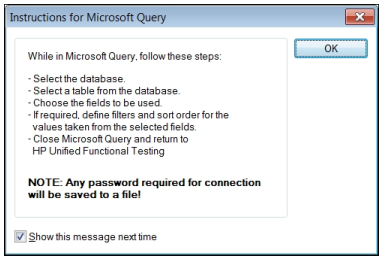
1. From the UFT menu, navigate to **Design** | **Checkpoint** | **Database Checkpoint**:



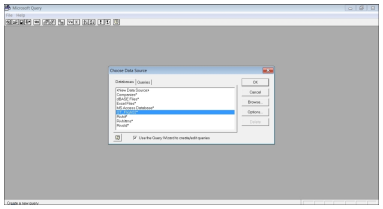
2. Now, the **Database Query Wizard** dialog will open:



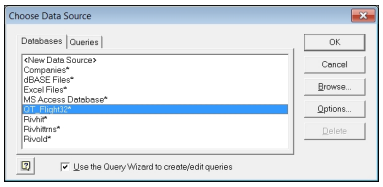
3. Now, in the **Database Query Wizard** dialog, we can choose to define our SQL statement using Microsoft Query, or we can do so manually. If we select the first option, **Create query using Microsoft Query**, and leave the **Show me how to use Microsoft Query** checkbox marked, the following dialog will appear:



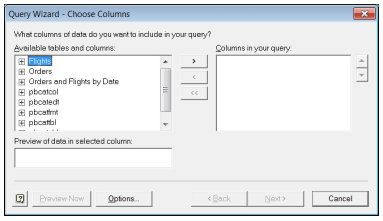
4. After closing the **Instructions for Microsoft Query** dialog, the **Microsoft Query** application will open:



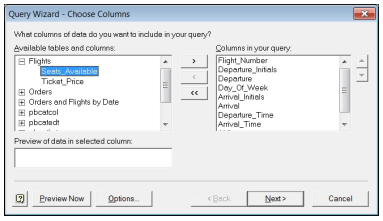
5. From the **Choose Data Source** dialog, we will select **QT_Flight32***:



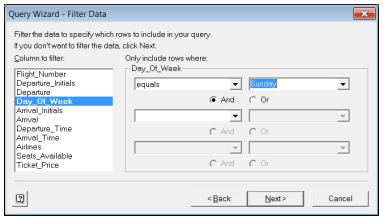
6. Next, the **Query Wizard - Choose Columns** window will open:



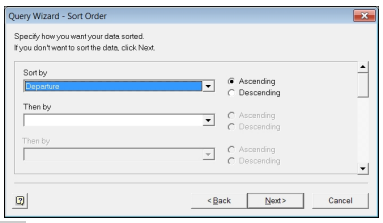
7. For our sample query, we will select all columns from the `Flights` table. You can opt to use fewer columns if you wish. Each column must be selected, and then the **>** button should be clicked on to include it in the **Columns in your query** list to the right of the window. The following screenshot shows an intermediate state of both lists (**Seats_Available** and the selected columns):



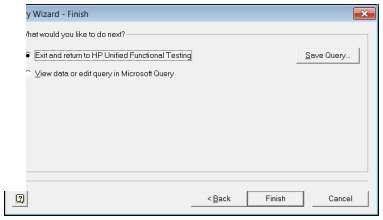
8. After finishing the selection of columns, we click on **Next**, and the next step is to build a filter for our query (equivalent to the `WHERE` statement in a SQL query). For example, we will select flights which depart on Sundays, as shown in the following screenshot:



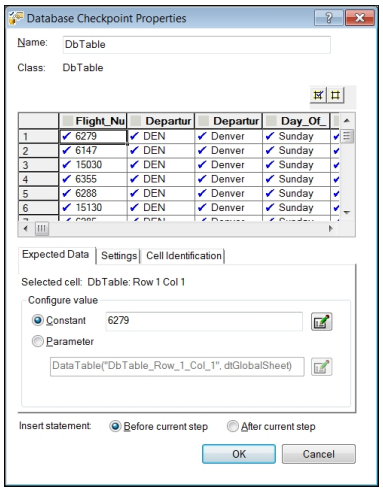
9. After clicking on **Next**, we will be able to define how we wish to sort the data. We will choose to sort the data according to the `Departure` column in ascending order, as shown:



/, clicking on **Next** will lead to the last screen **Query Wizard - ish** dialog:

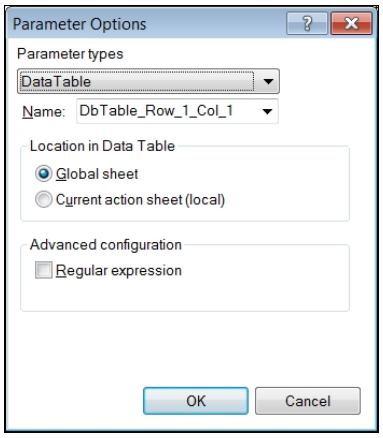


11. You can now save the query before exiting the process, view the data, or edit the query. We will stick to the default action and exit the process to return to UFT, in which the **Database Checkpoint Properties** dialog will open:

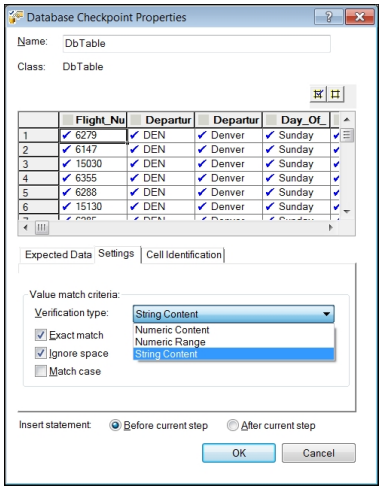


In the grid at the top of the preceding screenshot, we can see the numbered records of the `Flights` DB table, with the data in each cell for each of the columns we selected. Below the grid, we see three tabs: **Expected Data**, **Settings**, and **Cell Identification**.

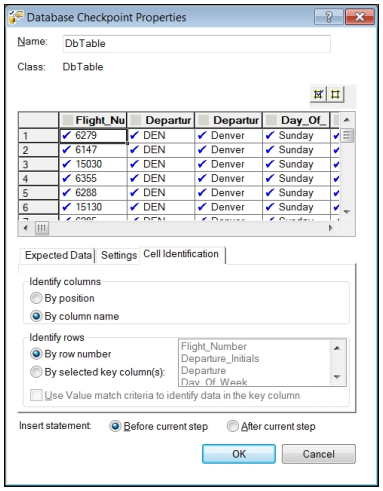
12. **Expected Data** can be **Constant** (the default option) or **Parameter**. We can configure the source of the parameter by selecting the **Parameter** radio button, and then clicking on the edit icon to the right. The **Parameter Options** dialog will open. Please note that **Parameter** will refer to the currently selected cell in the grid:



13. We can now select whether the value will be taken from the DataTable (either global or local), Environment, or a random number in the case of a DB checkpoint. It is also possible to define a parameter as a regular expression.
14. The **Settings** tab enables us to define whether the data verification will be done as a simple text or numeric comparison, or as a numeric range. We can also indicate whether we require an exact match, whether spaces should be ignored, and whether the letter case should be matched, as shown in the following screenshot:



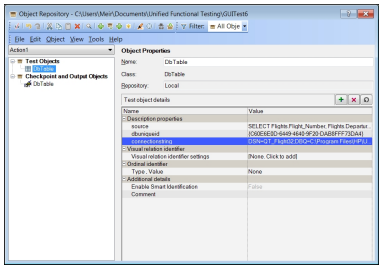
15. Finally, in the **Cell Identification** tab, we can define how we wish to identify our cells. For the rows, it is possible to use its number or key columns, and use **Value match criteria** defined in the **Settings** tab to identify a cell. For the columns, it is possible to use the column position or name, as shown:



16. After finishing the definition of our checkpoint properties, we select whether we want the resulting statement to appear before or after the current step. In our case, it does not matter, so we will leave the default value (**Before current step**) as it is, and click on **OK**. The resulting code is:

```
DbTable("DbTable").CheckCheckPoint("DbTable")
```

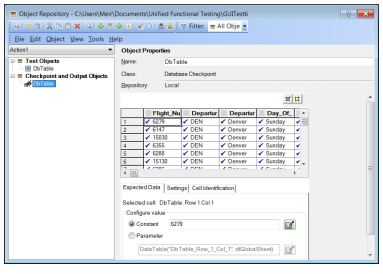
17. Our Object Repository now includes `DbTable` as **Test Object** and a `DbTable` checkpoint object, as in the following screenshot:



The `DbTable` TO will show that three description properties are used:

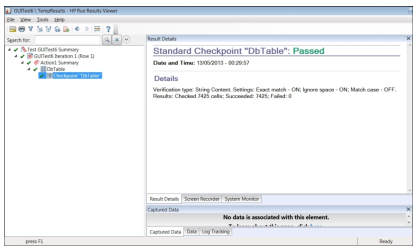
- `source`: This contains the SQL query we generated using the Microsoft Query Wizard.
- `dbuniqueid`: This contains a **Globally Unique Identifier** (**GUID**).
- `connectionstring`: This contains the connection string used to connect to the DB. We would use this to connect through raw VBScript code.

The `DbTable` checkpoint will show the settings, as we defined earlier in the **Database Checkpoint Properties** dialog:



How it works...

When invoking the Datatable, using the `DbTable.check` method with the DB checkpoint object, a connection is established using the connection string. Then, the data will be retrieved using the SQL query we defined, and each cell is compared to its identified counterpart using the value match criteria. Running the previous code will result in a results report, as follows:



Recommended / Queue
Feedback (http://community
PREV
Using SQL queri...
NEXT
Using an XML ch...