# Database Exercises – Using Visual Studio as an RDBMS

|  |  |
| --- | --- |
| **Exercise** | DBVS.1 |
| **Script** | **GameDBScript01** |
| **Purpose** | Create a database and add a single table to it. |
| **Steps** | 1. Open Visual Studio, and open the **SQL Sever Object Explorer** window (found under the **View** menu). 2. Expand the **SQL Server** entry, then the **(localdb)…** entry, and finally the **Databases** entry. 3. Right-click on **Databases**, and choose **Add New Database**. 4. In the **Create Database** dialog, enter “GameDB” in the **Data­base Name** field, and click **OK**. 5. Check that a new database named **GameDB** has been created in the **Databases** entry. If not, try to refresh the window by clicking the **Refresh** icon in the top left corner. 6. Right-click on the **GameDB** entry, and choose **New Query**. An **SQL Query** window should now open. 7. Now open the **GameDBScript01.txt** file (found in the **Scripts** folder under the **Database** chapter), and copy the entire content of the file into the **SQL Query** window. 8. Execute the script. The script creates a table **Weapon** in the **GameDB** database, popu­lated with 15 rows. 9. Go back to the **GameDB** entry in the **SQL Sever Object Explo­rer** window, and expand the **Tables** entry. A new table named **Weapon** (listed as **dbo.Weapon**) should now be found in the list of tables. 10. Right-click on the **Weapon** table, and choose **View Data**. You should now see 15 rows of data. 11. Right-click on the **Weapon** table, and choose **View Designer**. You should now see the table definition in the **Designer** view. |

|  |  |
| --- | --- |
| **Exercise** | DBVS.2 |
| **Script** | **GameDBScript01** |
| **Purpose** | Add a new table named **Character** to the **GameDB** database, by defining it in the **Designer** window. |
| **Steps** | 1. Complete exercise **DBVS.1**, such that you have created a **GameDB** data­base, and have added a table **Weapon** to the database. 2. Find the **GameDB** database in the **SQL Server Object Explorer** window (under **SQL Server** | **(localdb)** | **Databases**), expand it, right-click on the **Tables** entry, and choose **Add New Table…**. A **Designer** window should open, showing a default definition of a new table. 3. In the **T-SQL** pane of the **Designer** window, change **[dbo].[Table]** to **[dbo].[Character]**. This will change the name of the new table to **Character** 4. Now add columns corresponding to the below properties (remember to choose proper SQL types, e.g. **nvarchar** for strings, etc.):    1. **Name**: is a **string (nvarchar(50))**, cannot be ***null***.    2. **Race**: is a **string (nvarchar(50))**, cannot be ***null***.    3. **Role**: is a **string (nvarchar(50))**, cannot be ***null***.    4. **HealthPoints**: is an **int**, cannot be ***null***.    5. **WeaponLeft**: is an **int**, can be ***null***.    6. **WeaponRight**: is an **int**, can be ***null***. 5. When all column definitions have been added, click on the **Update** button in the top left corner. Check that the **Preview** dialog does not report any errors. If no errors are reported, click **Update Database**. 6. Check that a new table named **Character** has been added to the **GameDB** data­base (it should appear under the **Tables** entry under the **GameDB** database in the **SQL Server Object Explorer** window).   **Note**: For comparison, a script for a **Character** table can be found as part of the script **GameDBScript02.txt**, in the **Scripts** folder. |

|  |  |
| --- | --- |
| **Exercise** | DBVS.3 |
| **Script** | **GameDBScript01** |
| **Purpose** | Add foreign key constraints to a table, and observe the effect. |
| **Steps** | 1. Complete exercise **DBVS.1**, such that you have created a **GameDB** data­base, and have added a table **Weapon** to the database. 2. Complete exercise **DBVS.2**, such that you have added an additional table named **Character** to the **GameDB** data­base. 3. Open the **Character** table in the **Designer** window. 4. In the **Properties** pane (the top-right pane without a title…), right-click on **Foreign Keys**, and choose **Add New Foreign Key**. 5. Change the suggested name to **FK\_WeaponLeft**. 6. In the **T-SQL** pane, perform these changes to the generated line:    1. Change **[Column]** to **[WeaponLeft]**.    2. Change **[ToTable]** to **[Weapon]**.    3. Change **[ToTableColumn]** to **[Id]**. 7. If you have done the changes correctly, no errors should be reported. Now perform steps 4 to 6 again for the **WeaponRight** column, i.e. create a new foreign key named **FK\_WeaponRight**, etc.. 8. Now update the database, by clicking **Update**. Check that no errors are reported, and click **Update Database**. 9. Right-click on the **Weapon** table, and choose **View Data**. Observe that the weapons in the table have **Id** values from 1 to 15. 10. Right-click on the (just created) **Character** table, and choose **View Data**. The table should not contain any data yet. 11. Enter a row into the **Character** table (simply type values into the row which is initially filled with ***null*** values). When you enter values into the columns **WeaponLeft** and **WeaponRight**, choose values between 1 and 15 (i.e. valid **Id** values for rows in the **Weapon** table). This should pro­ceed without any errors. 12. Now add another row into the **Character** table, but this time use at least one invalid value in the columns **WeaponLeft** and **WeaponRight**. This should pro­duce an error… which is correct! This is due to the foreign key definitions added previously. It is now impossible for a character to use a non-existing weapon.   **Note**: For comparison, a script for a **Character** table (including foreign key constraints) can be found as part of the script **GameDBScript02.txt**, in the **Scripts** folder. |

|  |  |
| --- | --- |
| **Exercise** | DBVS.4 |
| **Script** | **GameDBScript01** |
| **Purpose** | Run a couple of given queries against the **GameDB** database, just to see how it works. |
| **Steps** | 1. Complete exercise **DBVS.1**, such that you have created a **GameDB** data­base, and have added a table **Weapon** to the database. 2. Find the **GameDB** database in the **SQL Server Object Explorer** window (under **SQL Server** | **(localdb)** | **Databases**), right-click on it, and choose **New Query**. An **SQL Query** window should open. 3. In the **SQL Query** window, run the query: SELECT \* FROM Weapon. This query should return 15 rows of data. 4. In the **SQL Query** window, run the query: SELECT \* FROM Weapon WHERE ItemLevel > 20. This query should return six rows of data. 5. In the **SQL Query** window, run the query: SELECT \* FROM Weapon WHERE type = 'Gun' ORDER BY Name. This query should return four rows of data, ordered in alphabetical order of the name. 6. Feel free to experiment a bit with creating and executing other queries (even though we haven’t really learned about it yet…). Also take note of how the **SQL Query** window reports syntax errors to you, and how it colors keyword, strings, etc.. |