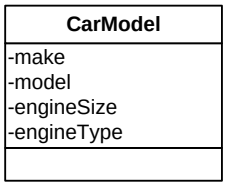
LAB 2: From data model to database

Getting started

In this lab you will **implement and test** a database to represent part of the GCUCars data model you discussed in the tutorial. You will **use SQL to create the tables** in your database.

Task 1 : Implement a table in your database to represent the *CarModel* entity

You will start by creating a table to represent the *CarModel* entity. The attributes of *CarModel* are shown in the class diagram below:



Before you actually create the table you will need to decide on the following:

- table **name** (following the usual convention this should be *CarModels*)
- **fields**
- **data types** for fields
- **primary key** for the table (a separate *id* field is often the best choice)
- any additional **constraints**, for example NOT NULL

1. Start up WebMatrix and create a new site called **gcucars**. Add a database to the site in the Databases workspace. Your database will be called **gcucars.sdf**.

2. Create a new query.

3. Enter an SQL CREATE TABLE statement which will implement the table according to the decisions you have made. You can see examples of CREATE TABLE statements in chapter 2 of your notes. You should make sure your table will have a **primary key**. You should use the **SQL data types** reference on page 2 to help you choose data types for the fields in your table.

   Your SQL statement should start like this:

```
CREATE TABLE CarModels (
```

4. Execute the query. If you have made any mistakes in the **syntax** of your SQL you will get an error message.

   *A **syntax error** tells you that you have typed something which is not valid SQL: for example, you might have missed out a bracket or a comma, or you may have misspelled one of your data types.*

   .

   If the query runs successfully, you will see a reassuring message in the results area.

5. Select **Tables** in the content pane, right-click and choose **Refresh**. Check that your new table is shown. Open your table and look at it in Data view and Definition view. Check that the correct fields are in the table. There will be no data in the table yet.

   If you are not happy with the table, you can right-click on the table in the content pane in the Databases workspace, and select Delete from the pop-up menu to remove it. You can then modify the query and run it again.

6. Save the query in a file in your Files workspace in a new SQL script file called *task1.sql*, using the method described in your lab 1 task sheet.

SQL data types quick reference

SQL Data Type	Comments
nvarchar(max)	Can specify max number of characters of text, up to 4000
int, bigint	Use for integer numbers
numeric(p,s)	Use for decimals. Can define Precision (max number of digits) and Scale (number of decimal places).
float	Use for real numbers
money	Use for amounts of money
bit	Integer data with a value of either 1 or 0
datetime	Use for date and time values. datetime field can hold both a date and a time

Notes:

- This is **not** a complete list of data types supported in SQL Server Compact
- see **http://msdn.microsoft.com/en-us/library/ms172424%28v=SQL.100%29.aspx** for a complete reference

Task 2: Entering data into the table

You should now test that you can actually store data successfully in this table.

1. Replace the contents of your query window with an SQL IINSERT statement to store a row in the *CarModels* table representing the following model:

   Vauxhall Astra with 1.8 litre petrol engine

   You can see examples of IINSERT statements in chapter 2 of your notes.

2. Execute the INSERT query.

   If you have problems with storing data, you may have to go back and modify your table. For example, you may have chosen a data type for a field which cannot store the required piece of information. You can delete the table and then modify and run your CREATE TABLE statement as described in step 5 of Task 1 above.

3. Open the table in Data view, if it is not already open, and check that the data has been stored successfully. Your may need to click **Refresh** on the **Table** ribbon tab to see the new data.

4. Save the query in a file in your Files workspace in a new SQL script file called *task2.sql*.

5. Add new rows to your table to represent the following car models. You can do this by editing your insert query, or by entering data in Data view. It's probably quicker to use Data view.

   - *Ford Focus with 1.8 litre petrol engine*
   - *Ford Mondeo with 2.0 litre diesel engine*
   - *Toyota Avensis with 2.2 litre diesel engine*
   - *Vauxhall Vectra with 2.2 litre petrol engine*
   - *Honda Jazz with 1.4 litre petrol engine*
   - *Vauxhall Corsa with 1.2 litre petrol engine*
   - *Toyota RAV-4 with 2.2 litre diesel engine*
   - *Vauxhall Antara with 1.9 litre diesel engine*
   - *Honda CR-V with 2.0 litre petrol engine*

   If you do want to use SQL for this task,, note that you can only have **one** INSERT INTO statement with a single set of VALUES in a query with SQL Server Compact. To insert multiple rows you can:

   - execute the INSERT query to insert the first row
   - edit the VALUES so that the query inserts the next row, and execute the query again
   - ...and so on

Testing your table

6. **Testing** is an important part of developing a database.

   You should try to add rows to **test any constraints** you have defined in your table, as follows:

   Create a new Word document called **gcucars_tests**. You should record all test results in this document. You will be doing tests in each task, so you can leave this document open while you work.

   If you have specified that any fields should be NOT NULL, you should check that you are not able to add a row with any of those fields left empty. For example, if the *model* field is NOT NULL, you could test and record the results in your Word document as follows:

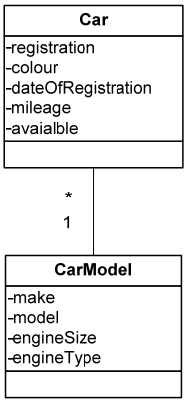
   Test for constraint: Table: *CarModels*, Field: *model*, Constraint: *NOT NULL*
   Test data: VALUES ('Honda', null, 2.0, 'petrol')
   Expected result: Error
   Result: Error - The column cannot contain null values
   Test passed

   If a test is not passed, you should find and correct the problem and re-test.

   You can then **test the primary key** by adding a row which has the same primary key value(s) as an existing row (**this test is not necessary if you used an identity field)**

Task 3: Implement a related table

In the data model, the *Car* entity represents a specific vehicle, for example a specific Vauxhall Astra. There can be many cars of any one model - GCUCars might, for example, have five 1.8 litre Astras in their fleet.



You will now implement a table to store information about cars, and define the relationship between this and the *CarModels* table.

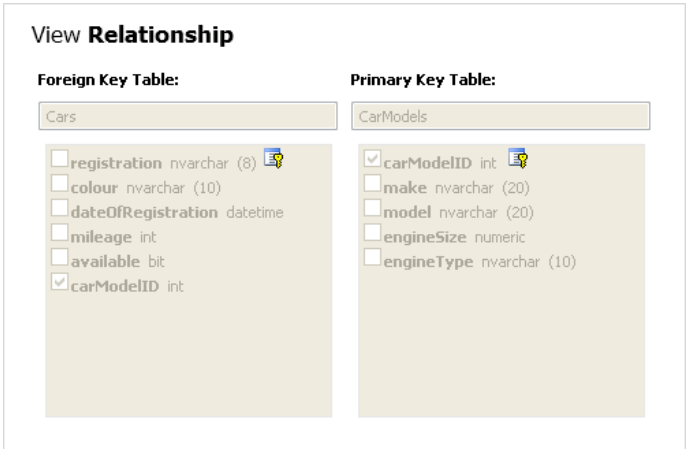
1. Design a table called *Cars* with the fields required to store the information about cars. As before, consider fields, data types, primary key and constraints.

2. Create and execute a query called which contains the SQL CREATE TABLE statement to implement this table.

   Your SQL statement should define the relationship between the *Cars* and *CarModels* tables. You will need to have a field (or fields) in *Cars* to match the primary key you defined in *CarModels*. You will also need to define the foreign key. Relevant examples can again be found in chapter 2 of your notes. Should the foreign key field(s) be NOT NULL? Why?

3. Save the query in a file in your Files workspace in a new SQL script file called *task3.sql*.

4. Open the **View Relationship** window and check that the relationship between *CarModels* and *Cars* is correctly defined. It should look like this:



Testing your new table

5. **Insert a row** in the *Cars* table representing the following car which is available for hire. You can use SQL or Data view.

   SD11XYZ, a red Astra, registered on 1/3/11, mileage 12500

   You will need to enter the *carModelID* value in your data row which corresponds to the *carCodelID* for the Vauxhall Astra model in the *CarModels* table. If you get an error message when storing this data you may have set up the foreign key incorrectly. Check this, and delete and recreate the *Cars* table if necessary.

6. **Test** that data which violate any constraints cannot be stored.

   Record all test data and the relevant constraints in your Word document.

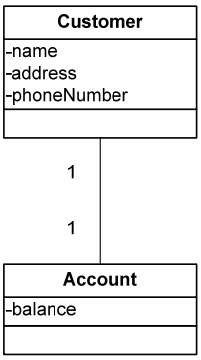
7. **Test** to check that the foreign key works as you intended:
   - Try to store a record with the foreign key left empty.
   - Try to store a record with a value in the foreign key which doesn't match a value in the primary key of *CarModels*
   - Record the results of these tests.

8. Finally, store rows representing the following cars:

   SE11XXZ, a green Focus, registered on 1/4/11, mileage 14000, available
   SF11XXY, a grey Focus, registered on 2/4/11, mileage 11800, available
   SG11XZY, a blue Astra, registered on 3/4/11, mileage 13000, available
   SH11ZXY, a silver Insignia, registered on 4/4/11, mileage 14600, available
   SJ11ZZZ, a silver Insignia, registered on 4/4/11, mileage 14900, available

Task 4: Implementing a one-to-one relationship

There is a **one-to-one** relationship between the *Customer* and *Account* entities in the data model.



In this task you should design, implement and test a table or tables to store the following information about the following customers:

Dario Franchitti, 1 First Street, 0141 987 6543
Danica Patrick, 2 Second Street, 0141 876 5432
Marino Franchitti, 3 Third Street, 0141 765 4321
Sarah Fisher, 4 Fourth Street, 0141 123 4567

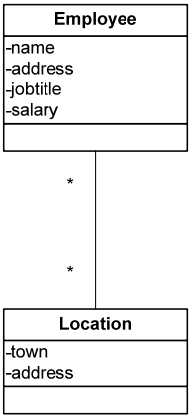
These customers all have an account with balance of zero, except Marino Franchitti, who has a balance of £156.68

Use SQL for table creation, and save your SQL CREATE TABLE statement in a file in your Files workspace in a new SQL script file called *task3.sql*.

Save any additional test data and test results in your Word document under the heading TASK 4.

Task 5: Implementing a many-to-many relationship

There is a many-to-many relationship between the *Employee* and *Location* entities in the data model.



In this task you should design, implement and test a table or tables to store the following information about the following employees and locations:

Locations:
1 First Avenue, Glasgow
2 Second Avenue, Paisley
3 Third Avenue, Irvine
4 Fourth Avenue, Kilmarnock
5 Fifth Avenue, Ayr

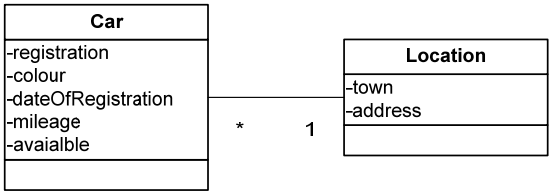
Employees:
Asif Rashid is an area manager, based at Glasgow and Paisley, earning £25000.
Jacklyn Elder is an area manager, based at Irvine, Kilmarnock and Ayr, earning £25000
Carol Koster is a CSR, based at Paisley, earning £15000
Martin Dale is a CSR, based at Kilmarnock and Ayr, earning £16000
Lennox Ward is a service engineer, based at Irvine, earning £21000
Jane Lee is a service engineer, based at Glasgow and Paisley, earning £22000

Use SQL for table creation, and save your SQL CREATE TABLE statement in a file in your Files workspace in a new SQL script file called *task5.sql*.

Save any additional test data and test results in your Word document under the heading TASK 5.

Task 6: Altering a table

You have already implemented tables to represent the *Location* and *Car* entities. However, you have not implemented the one-to-many relationship between them.



In this task you should use Alter Table statements to modify the tables to implement the relationship (actually, you should only need to modify one table). You should then test your modifications as follows:

Specify the locations for cars:
SD11XYZ at Glasgow
SE11XX, at Paisley
SF11XXY at Paisley
SG11XZY at Kilmarnock

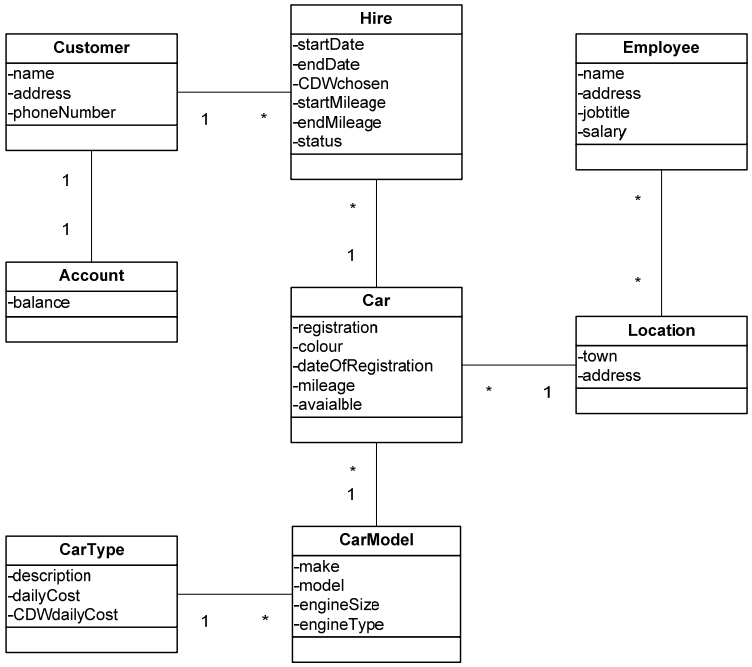
Try to specify a non-existent location for car SJ11ZXY – this should be ***unsuccessful***

Use SQL for table modification, and save your SQL ALTER TABLE statement in a file in your Files workspace in a new SQL script file called *task6.sql*.

Save any additional test data and test results in your Word document under the heading TASK 6.

TASK 7: Completing the database

If you have completed tasks 1 to 6, you should be able to complete the database so that it represents the whole data model. The entities you have not dealt with yet are *CarType* and *Hire*.



In this task you should design, implement, modify and test tables as needed to complete the database. You should be able to store the following information:

> *Vauxhall Corsa and Honda Jazz are in the **Economy** car type, daily cost £38, CDW £12*
> *Vauxhall Astra and Ford Focus are in the **Compact** car type, daily cost £45, CDW £14*
> *Ford Mondeo, Toyota Avensis and Vauxhall Insignia are in the **Intermediate** car type, daily cost £49, CDW £15*
> *Toyota RAV-4, Vauxhall Antara and Honda CR-V are in the **SUV** car type, daily cost £58, CDW £19*
>
> *Dario Franchitti hired SD11XYZ on 3/9/11 until 4/10/11 with CDW, start mileage 11800, end mileage 12500*
> *Sarah Fisher hired SG11XZY on 5/10/11 with no CDW, start mileage 13000, and this hire is not completed*

Use SQL for table creation and modification, and save your SQL statements in a file in your Files workspace in a new SQL script file called *task7.sql*.

Save any additional test data and test results in your Word document under the heading TASK 7.