

COMP 3111: Software Engineering

Managing Data Using Microsoft SQL Server Tutorial and Lab Notes

A database is an organized collection of data (i.e., somewhat like an electronic filing cabinet for data). A database management system (DBMS) is software that allows you to do the following.

- Create a database.
- Add, modify and delete data in a database.
- Retrieve data from (i.e., query) a database.
- Produce reports summarizing selected contents of a database.

These notes will explain how to create, populate and query a database using SQL Server.

GUIDELINES FOR DESIGNING A RELATIONAL DATABASE

An application supported by a DBMS contains large amounts of structured data from some application domain. The *schema* of the database describes how that data is logically structured by the DBMS. In the case of a relational database, the schema consists of one or more relations, represented as tables, each of which has one or more attributes, represented as columns of a table. Deciding which relations should be created, which attributes each of these relations should contain and what are the relationships between the relations is one of the major tasks of relational database design. The following guidelines should help you arrive at a "good" choice of relations, of attributes for each relation and of relationships between the relations.

Guideline 1: Clear Semantics for Attributes

Design a relation so that it is easy to explain its meaning (semantics). Typically, this means that you should not place attributes that describe more than one real world thing into a single relation. For example, student and course information should not be placed in one relation.

Guideline 2: Minimize Null Values in Relations

As far as possible, avoid placing attributes in relations whose values may be null (i.e., have no value) for many (most) rows of the relation. If nulls are unavoidable, make sure that they apply in *exceptional cases* only. This is because the meaning of a null value may not be clear—Is it unknown?, Does it not apply?—and null values cause problems for some arithmetic operations such as average.

Guideline 3: Minimize Redundant Values in Relations

Design relations so that no undesirable insertion, deletion or updates (called *update anomalies*¹) occur in the relations. Update anomalies usually result from redundant data in relations. Therefore, redundant data should be eliminated or minimized in a relation. If any update anomalies are present, note them clearly so that update programs will operate correctly.

CREATING A DATABASE

We will create an SQL Server database named *Enrollment* for the ASU application. To create the *Enrollment* database, do the following.

1. Open the "ASUWebApplication" project in Visual Studio.
2. Right-click the "ASUWebApplication" folder in the `Solution Explorer` and select "Add→Add New Item..." from the popup menu or pressing "ctrl+Shift+A".
3. In the Add New Item window, select "...Visual C#→Data" in the left pane, "SQL Server Database" in the centre pane and enter "Enrollment.mdf" in the "Name" field as shown in Figure 1(a).
4. Click the "Add" button.
5. Click the "Yes" button in the dialog box (see Figure 1(b)) advising that the database be saved in the App_Data folder².

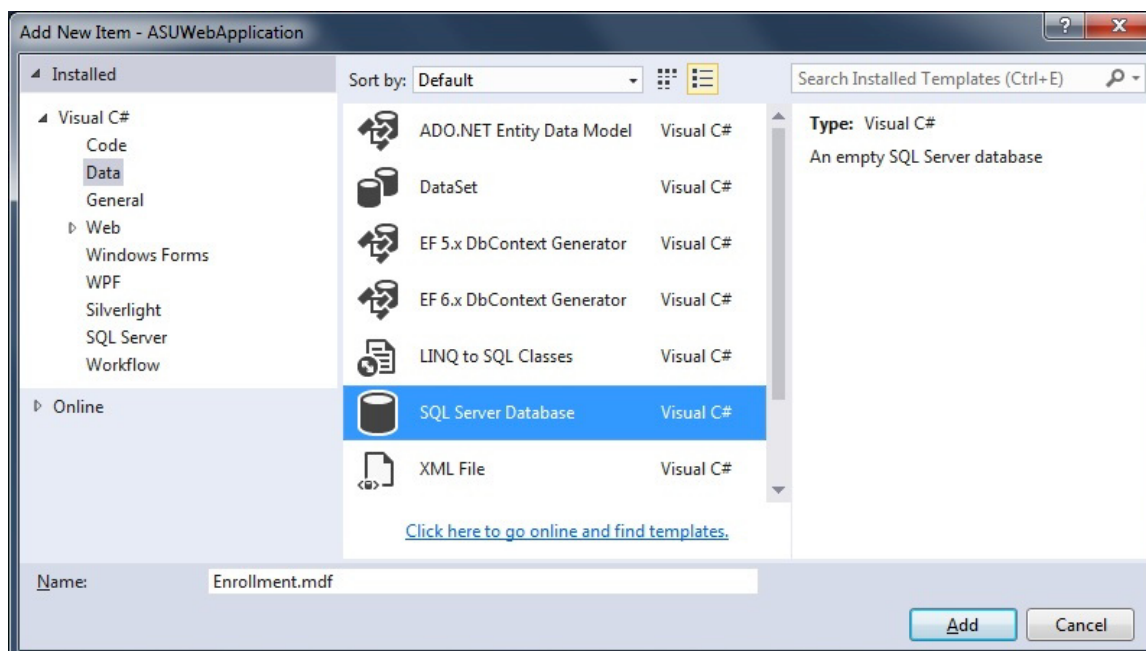
The *Enrollment* database is created inside the App_Data folder of the ASUWebApplication folder as shown in Figure 2(a) and also appears in the SQL Server Object Explorer window³ as shown in Figure 2(b).

(Note: Use the SQL Server Object Explorer window and not the Server Explorer window.)

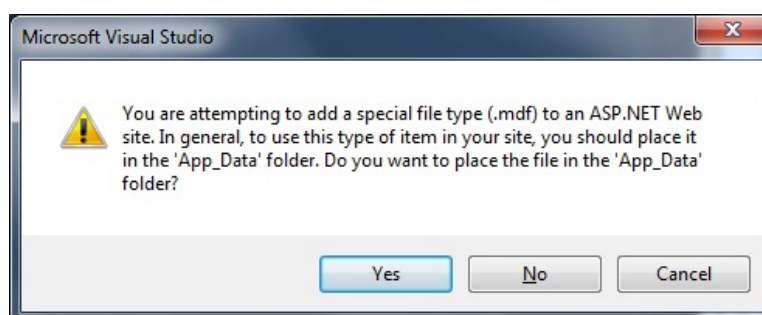
¹ An update anomaly is an unexpected or undesirable side effect of accessing a relation.

² It is strongly recommended that all permanent data for a web site be stored in the App_Data folder. This helps protect the data and provides a good design structure for your web site.

³ The SQL Sever Object Explorer window can be opened by selecting "View→SQL Server Object Explorer" in the Visual Studio menu bar or by pressing "ctrl+S".

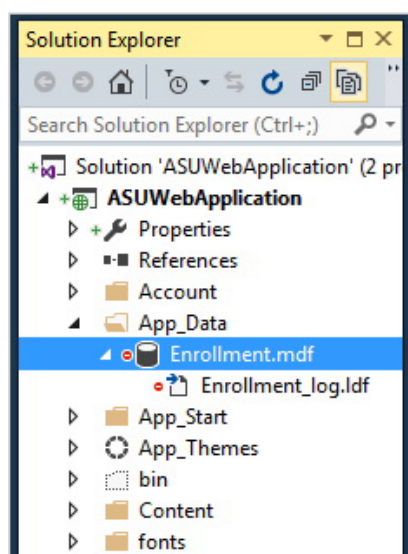


(a) Add New Item window.

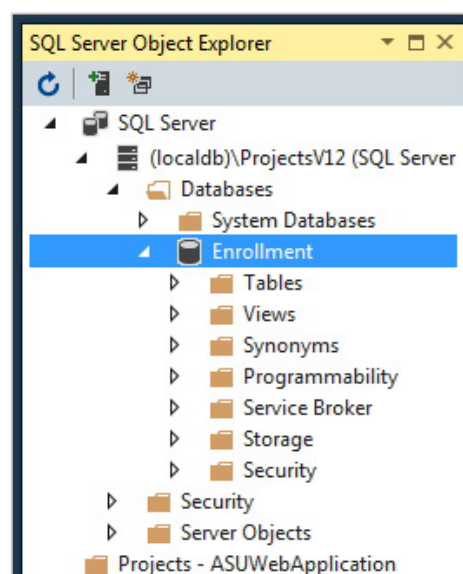


(b) Dialog box advising that the *Enrollment* database be placed in the App_Data folder.

Figure 1: Creating an SQL Server database.



(a) Solution Explorer window



(b) SQL Server Object Explorer window

Figure 2: Solution Explorer and SQL Server Object Explorer windows after creating the *Enrollment* database.

CREATING TABLES (RELATIONS)

An SQL Server database is a collection of tables (relations) that organize the data that you want to store. To create a new table in the *Enrollment* database, open the SQL Server Object Explorer window, expand the “SQL Server”, “(localdb)...”, “Databases” and “Enrollment” nodes as shown in Figure 2(b)⁴. Under the “Enrollment” node, right-click the Tables folder and select “Add New Table” from the popup menu.

A Table Designer opens in the Document window as shown in Figure 3 consisting of the Columns Grid Pane (top-left side), Context Pane (top-right side) and Script Pane (bottom). The Columns Grid Pane lists all the columns in a table and allows adding, editing and deleting columns. The Context Pane provides a logical view of the table definition (Keys, Constraints, Indexes, Foreign Keys and Triggers) and enables selecting an object to highlight its relationships to individual columns. The Script Pane shows the SQL definition of the table structure and highlights the script of the selected object in the Context Pane or Columns Grid Pane. Changes made in any of the three panes are propagated to the other two immediately.

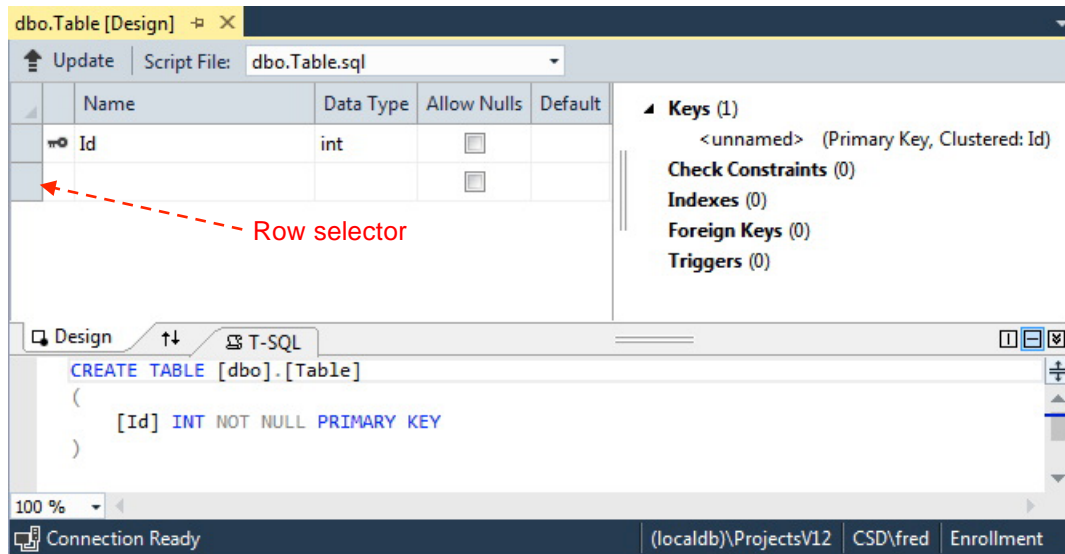


Figure 3: SQL Server Table Designer.

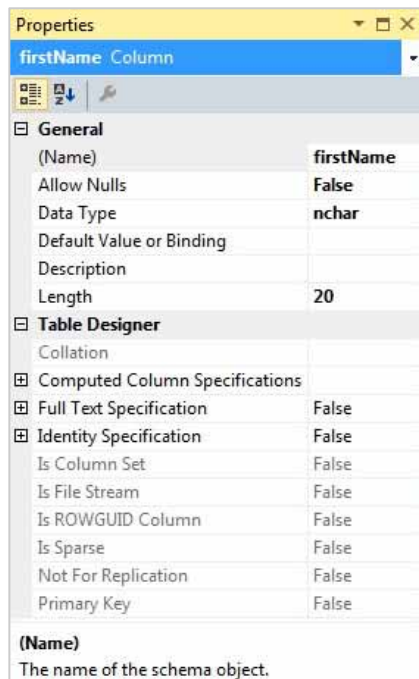


Figure 4: Column Properties window.

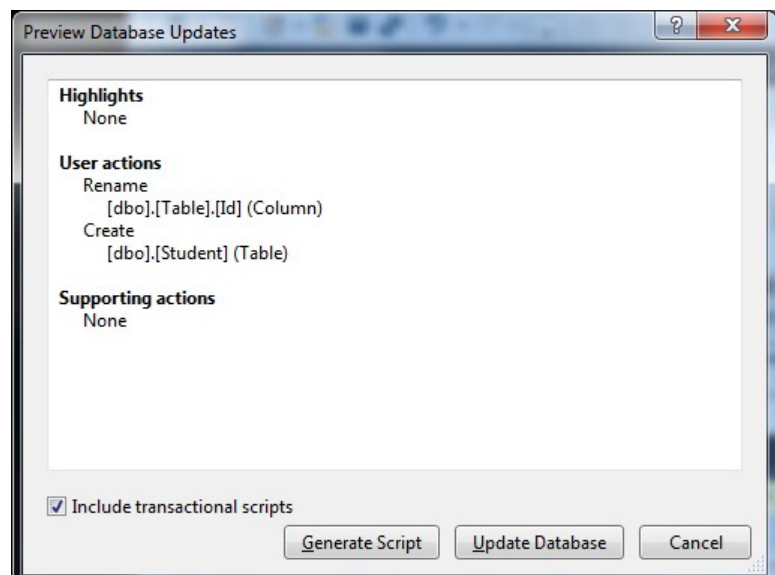


Figure 5: Preview Database Updates dialog box.

⁴ You may need to first open the Server Explorer window to connect to the database.


To complete the definition of a table, first rename the table in the Script Pane. Specifically, replace the text "Table" in the first line of the script in Figure 3 with the name of the table (see Figure 6 for an example). You can name two tables in two different databases with the same name, but not in the same database.

Next, define the columns of the table. To do so, click the row selector in the Columns Grid Pane as shown in Figure 3 and then, in each column, either enter or select values for the following⁵.

- **Name**⁶ – Enter a unique column name by typing in the Name textbox⁷. You can name two columns in two different tables with the same name, but not in the same table.
- **Data Type** – Select the type of data the column will contain from the drop down list in the column⁸.
- **Length or Precision/Scale** – The number of characters for character-based data types or the maximum number of digits and the number of digits to the right of the decimal point for numeric data types. These values can be directly edited in the Data Type column of the Columns Grid Pane or in the column's Properties window.
- **Allow Nulls** – Leave the box unchecked if null values are not allowed; otherwise check the box.
- **Default** – The default value for the column if no value is specified in an Insert statement.

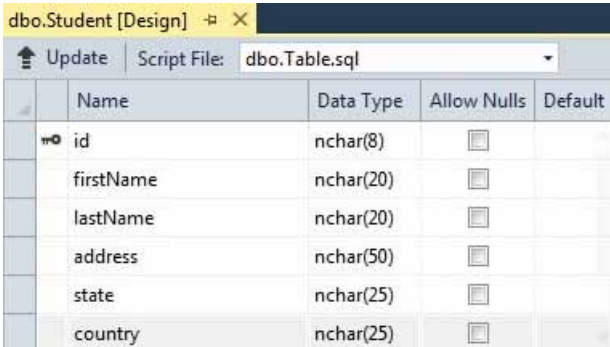
The following additional properties can be set for each column of a table⁹.

- **Description** – A meaningful textual description of the column.
- **Collation** – the collating (ordering) sequence that should be applied to the column for sorting.
- **Computed Column Specifications** – Indicates whether the data in the column is computed (derived) data rather than physically stored in the table and allows a formula for computing the value to be entered.
- **Full Text Specification** – Specifies properties specific to full-text columns.
- **Identity Specification** – Indicates whether the column value is unique in every row of the table and is automatically incremented for each new row added to the table. The column must be an integer or decimal data type. Further properties that can be set are *Identity Increment*, which indicates the increment for the numeric values (the default is 1) and *Identity Seed*, which is the numeric value assigned to the first row added to the table (the default is 1).

Finally, when you have finished defining the table, save it by clicking the  button in the upper left corner of the Table Designer as shown in Figure 3. **Do not use the save function of Visual Studio as this will not properly update the database!** The Preview Database Updates dialog box appears as shown in Figure 5 showing a summary of the actions that will be taken (e.g., creating or dropping database objects), together with potential issues that have been identified¹⁰. If no issues have been identified, then click the "Update Database" button. The newly created table will now appear in the SQL Server Object Explorer window under the Tables tab of the database.

For the ASU *Enrollment* database we will create three tables, *Student*, *Course* and *EnrolledIn*, with the following structure. Figure 6 shows the Table Designer after completing the definition of the *Student* table.

Student Table			
Column Name	Data Type	Allow Nulls	Other Properties
id	nchar(8)	No	Primary Key
firstName	nchar(20)	No	-
lastName	nchar(20)	No	-
address	nchar(50)	No	-
state	nchar(25)	No	-
country	nchar(25)	No	-



Student table defined.

⁵ These values can also be set in a column's Properties window as shown in Figure 4. However, currently only the General properties of a column can be set in a column's Properties window. Other properties must be set in the Script Pane.

⁶ It is good design style not to use column names that are reserved words of SQL Server as this may cause system errors. SQL Server will not warn you if you try to use column names that are reserved words. A list of reserved words in SQL Server can be found at <https://msdn.microsoft.com/en-us/library/ms189822.aspx>.

⁷ These notes, as well as the course notes, use the *camel style* in which the column name starts with a lowercase letter and any subsequent proper name in the column name starts with an uppercase letter (e.g., *courseCode*).

⁸ The SQL Server supported data types are described in Appendix A.

⁹ For a full description of all of the column properties see <http://technet.microsoft.com/en-us/library/ms177173.aspx>.

¹⁰ A deployment script is generated in the background and executed when the "Update Database" button is clicked.

Course Table			
Column Name	Data Type	Allow Nulls	Other Properties
code	nchar(8)	No	Primary Key
title	nchar(50)	No	-
credits	tinyint	No	Default Value: 3

Name	Data Type	Allow Nulls	Default
code	nchar(8)	<input type="checkbox"/>	
title	nchar(50)	<input type="checkbox"/>	
credits	tinyint	<input type="checkbox"/>	

Course table defined.

EnrolledIn Table			
Column Name	Data Type	Allow Nulls	Other Properties
studentId	nchar(8)	No	Primary Key
courseCode	nchar(8)	No	Primary Key
section	nchar(1)	No	-
semester	nchar(6)	No	-
year	nchar(4)	No	-
grade	numeric(4,1)	Yes	-

Name	Data Type	Allow Nulls	Default
studentId	nchar(8)	<input type="checkbox"/>	
courseCode	nchar(8)	<input type="checkbox"/>	
section	nchar(1)	<input type="checkbox"/>	
semester	nchar(6)	<input type="checkbox"/>	
year	nchar(4)	<input type="checkbox"/>	
grade	numeric(4,1)	<input checked="" type="checkbox"/>	

EnrolledIn table defined.

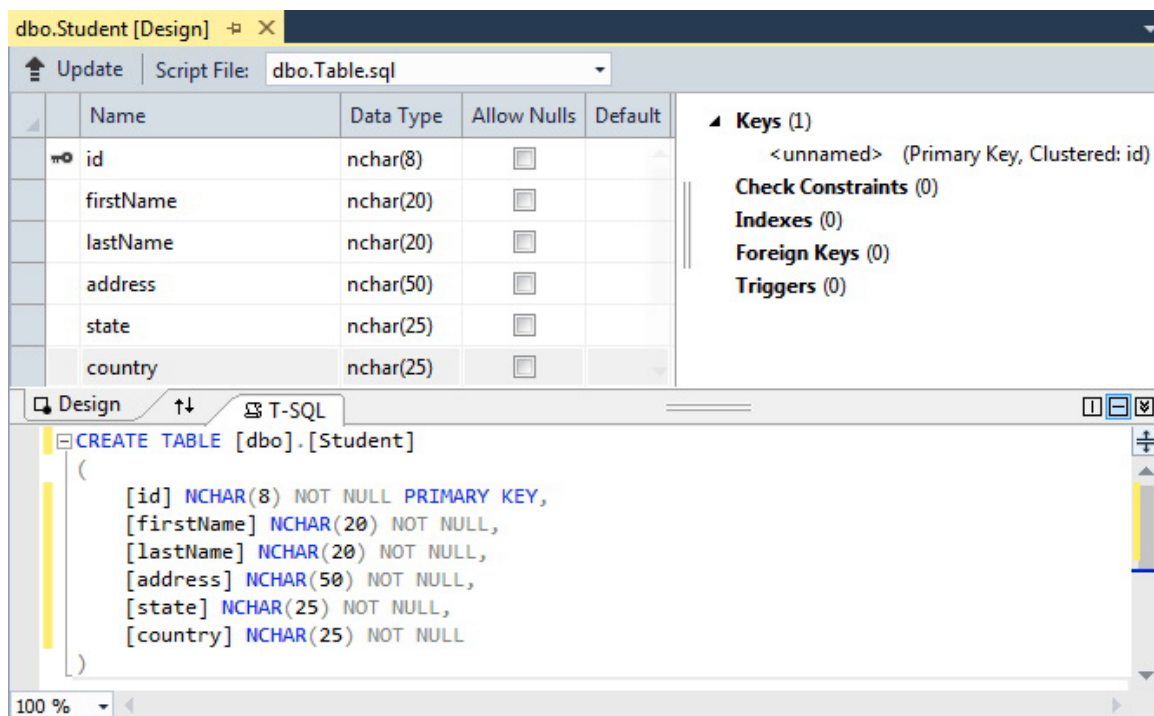


Figure 6: Table Designer showing the *Student* table after clicking the “Update Database” button¹¹.

Creating a Primary Key for a Table


A primary key¹² is the unique identifier of a row in a table. It can be either a single column or a collection of columns. For the three tables that we are creating, *id* in the *Student* table, *code* in the *Course* table and the combination of *studentId* and *courseCode* in the *EnrolledIn* table should be the primary keys.

¹¹ SQL Server uses square brackets [] in SQL statements to delimit identifiers. It is necessary to use square brackets if a table or column name is a reserved keyword or contains special characters such as a space or hyphen; otherwise the square brackets are optional.

¹² It is not necessary for every table to have a primary key, but it is highly recommended. If a primary key is specified in a table, SQL Server automatically creates an index for the primary key that speeds up queries or other operations. Moreover, when you view records, SQL Server displays them in the primary key order. Most importantly, you can make sure that every record in a table is unique since SQL Server does not allow any records with the same primary key value to exist in a table.

The first column in a table is automatically set as the primary key by default in the Table Designer. You can redefine or include additional columns as part of the primary key for a table by doing the following.

1. Select the desired column(s) by clicking on the row in the Table Designer. Clicking a row while holding down the Shift key allows you to select multiple columns.
2. Right-click and select "Set Primary Key" in the pop-up menu.

A key symbol  appears to the left of the column name after successfully creating the primary key as shown in the screen shots next to the table definitions in the previous section.

Modifying a Table's Definition

To modify the definition of a table, double-click the table name in SQL Server Object Explorer to open it in the Table Designer. Make any required changes and update the table.

Specifying Referential Integrity between Tables

Referential integrity is an essential property for relational databases that ensures that each of the values of a column (called the foreign key) in one table (called the foreign key table) matches with exactly one value of the primary key in another table (called the primary key table¹³). For example, for the *Student* and *EnrolledIn* tables, we want to ensure that each value of *studentId* in the *EnrolledIn* table matches with a value of *id* in the *Student* table. Otherwise, we could have *EnrolledIn* records for students that do not exist!

To specify referential integrity in SQL Server you define a relationship between the foreign key table and the primary key table and instruct SQL Server how you want to enforce referential integrity. SQL Server then either prohibits any updates to the database that would violate the referential integrity rules or automatically performs updates to maintain the referential integrity rules.

SQL Server can enforce two types of referential integrity rules.

- **On Update Cascade rule** – Values of a related foreign key column should be updated when the value of a primary key column changes. If this rule is specified when defining a foreign key relationship, then whenever the primary key value of a record is changed in the primary key table, SQL Server automatically updates the foreign key values to the new value in all related records in the related foreign key table. Otherwise, SQL Server prevents updates to the primary key in the primary key table. For example, if the "On Update Cascade" rule is specified for the *courseCode* column in the *EnrolledIn* table, then when a value of the *code* column in the *Course* table is changed, the values of the *courseCode* column in the *EnrolledIn* table that match the old value of the *code* column are automatically updated so that the relationship is not broken. If no such rule is specified, then SQL Server prevents values of the *code* column in the *Course* table from being changed.
- **On Delete Cascade rule** – Related records in a related foreign key table should be deleted when a record in the primary key table is deleted. If this rule is specified when defining a foreign key relationship, then whenever a record is deleted in the primary key table, SQL Server automatically deletes related records in the related foreign key table. Otherwise, SQL Server prevents records in the primary table from being deleted if there are related records in the foreign key table. For example, if the "On Delete Cascade" rule is specified for the *studentId* column in the *EnrolledIn* table, then when a student record from the *Student* table is deleted all the student's *EnrolledIn* records are automatically deleted from the *EnrolledIn* table. If no such rule is specified, then SQL Server prevents a *Student* record from being deleted if there are any *EnrolledIn* records with a *studentId* value that matches the *id* value in the *Student* record.

For the relationship between the *Student* and *EnrolledIn* tables, we want *EnrolledIn* records to be deleted whenever a student is deleted (i.e., On Delete Cascade). However, it should not be allowed to update a student's id. This referential integrity constraint is specified as follows.

1. Open the *EnrolledIn* table in the Table Designer.
2. Right-click the "Foreign Keys" node in the Context Pane of the designer, and select "Add New Foreign Key" from the popup menu. A new foreign key constraint appears in both the Context and the Script Panes as shown in Figure 7(a).
3. In the Script Pane, do the following (see Figure 7(b)):
 - a. Change the two occurrences of the text "ToTable" to "Student".
 - b. Change the text "Column" to "studentId", the name of the column in the *EnrolledIn* table.
 - c. Change the text "ToTableColumn" to "id", the name of the column in the *Student* table.
 - d. Add the text "ON DELETE CASCADE" at the end of the foreign key constraint.

¹³ A primary key table is the table on the "one" side of two related tables in a one-to-many relationship. A primary key table should have a primary key and each record should be unique.



(a) Defining referential integrity constraint between *Student* and *EnrolledIn* tables.



(b) Referential integrity constraint between *Course* and *EnrolledIn* tables.



(c) Referential integrity constraint between *Course* and *EnrolledIn* tables.

Figure 7: Specifying referential integrity constraints for the *Enrollment* database.

Specifying “On Delete Cascade” will cause any related records in the *EnrolledIn* table to be deleted when a record in the *Student* table is deleted. Omitting “On Update Cascade” prevents the *studentId* column in the *Student* table from being updated.

For the relationship between the *Course* and *EnrolledIn* tables, we want *EnrolledIn* records to be updated whenever a *Course* record’s code is updated (i.e., add an On Update Cascade rule). However, a *Course* record should not be deleted if there are related *EnrolledIn* records (i.e., omit an On Delete Cascade rule). This referential integrity constraint is defined in a similar way as the previous constraint (see Figure 7(c)).

POPULATING AND MAINTAINING A DATABASE

Adding Records to a Table

Records can be added manually one at a time to a table or they can be added to a table programmatically. To add records to a table manually one at a time, do the following.

1. Right-click the table in SQL Server Object Explorer and select “View Data” from the popup menu. An editor opens in which data can be entered, edited and deleted as shown in Figure 8(a)
2. Click in a column or use the Tab key to move around the columns and fill in the column values.

Enter the data shown in Figure 8(b) for the *Course* table.

Figure 8(a) shows the SQL Server Data Editor for the `dbo.Course` table. The table has three columns: `code`, `title`, and `credits`. The data is currently empty, with all cells showing `NULL`.

(a) After initially selecting “View Data”.

Figure 8(b) shows the SQL Server Data Editor for the `dbo.Course` table after adding data. The table has three columns: `code`, `title`, and `credits`. The data is as follows:

code	title	credits
COMP3111	Software Engineering	4
COMP3311	Database Management Systems	3
COMP4311	Principles of Database Design	3
NULL	NULL	NULL

(b) After adding course information.

Figure 8: Adding records to the *Course* table.

Records can also be added from an Excel spreadsheet by copying them to the editor. First, prepare the records in Excel making sure that there is one value in each record for each column in the table. In the editor, click on the last row selector (the one that shows “NULL” for all columns) and then copy and paste the records from Excel onto the datasheet. Make sure that the correct type of value is entered for each column and that the length of a value does not exceed the length specified for it in the table definition.

Download the files *Student.xlsx* and *EnrolledIn.xlsx* from the Tutorials section of the course web page and insert records into the *Student* and *EnrolledIn* tables by copying and pasting them from their respective Excel files. Make sure that you insert the *Student* records before you insert the *EnrolledIn* records so that the referential integrity rules between the *Student* and *EnrolledIn* tables are not violated¹⁴.

Changing Records in a Table

To change a record in a table, do the following.

1. Open the table in the editor.
2. Edit the column value(s) of the record directly in the editor.

Deleting Records from a Table

To delete a record from a table, do the following.

1. Open the table in the editor.
2. Right-click anywhere in the record and choose “Delete” from the popup menu.

QUERYING A DATABASE

A query is a question that a DBMS can understand and process to find the answer from a database. SQL Server supports SQL (Structured Query Language), pronounced *sequel*, which allows you to express queries as SELECT-FROM-WHERE statements. The general syntax of SQL statements is as follows¹⁵.

```
select [all | distinct] { * | column_specification }  
from table_specification  
[where search_condition]  
[group by column_specification]  
[having search_condition]  
[order by { column_specification [asc | desc], ...}]
```

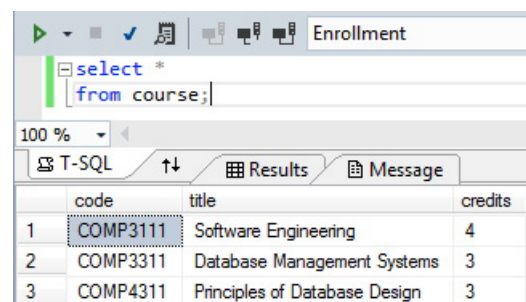



Figure 9: Query Editor showing query that selects all records in the *Course* table.

SQL Server provides the ability to construct SQL statements in a Query Editor. To create a query using the Query Editor, do the following.

1. Right-click the database node (i.e., the name of the database) in SQL Server Object Explorer and select “New Query...” from the popup menu.
2. Construct the SQL query in the Query Editor as shown in Figure 9.
3. Execute the query by clicking the green button  in the upper left corner of the Query Editor or right-clicking in the Query Editor and selecting “Execute” from the popup menu or pressing “ctrl+Shift+E”. By default the results will be displayed in a grid as shown in Figure 9. The results can also be displayed as text or saved to a file.

As shown above, an SQL query requires at least a *select* clause and a *from* clause. The other clauses are optional.

The *select* clause specifies for which columns data should be retrieved. The keyword “*” specifies that data for all columns of the tables specified in the *from* clause should be retrieved. For example, the query in Figure 10(a) retrieves the value of all columns in all records in the *Course* table. The *column_specification* option specifies a comma-separated list of the desired columns for which data should be retrieved as shown in Figure 10(b). The keyword “all” specifies that duplicate records can appear in the result, which is the default. The keyword “distinct” specifies that only unique records can appear in the result¹⁶.

¹⁴ The first row of each spreadsheet contains the column name. **Do not copy and paste this row.**

¹⁵ In the SQL statement syntax, { } means select one of the items and [] means the item is optional.

¹⁶ Although database relations, and the operations on them, are based on the mathematical theory of relations in which each relation is a mathematical set, for practical purposes relational DBMSs allow tables to have duplicate rows.


```
select *
from Course
```

(a)

```
select lastName, firstName, country
from Student
```

(b)

```
select *
from Student
where state= 'Hong Kong - SAR'
```

(c)

```
select studentId
from EnrolledIn
where courseCode='COMP3111' and grade>80
```

(d)

```
select lastName, firstName, grade
from Student, EnrolledIn
where Student.id=EnrolledIn.studentId
```

(e)

```
select lastName, firstName, avg(grade)
from Student, EnrolledIn
where Student.id=EnrolledIn.studentId
group by studentId, lastName, firstName
having avg(grade)>80
```

(f)

```
select firstName, lastName, id
from Student
where id in
((select studentId
from EnrolledIn
where courseCode='COMP3111' and [year]='2015')
intersect
(select studentId
from EnrolledIn
where courseCode='COMP3111' and [year]='2015'))
```

(g)

Figure 10: Example SQL queries.

Often only certain records that meet some condition should be retrieved from a table. For example, we may want to retrieve all students from Hong Kong. This is specified as a *search_condition* in the *where* clause using a comparison operator as shown in Figure 10(c)¹⁷. The SQL logical comparison operators are

<	less than	>=	greater than or equal to	<i>any</i>	match a scalar value with a set/list
>	greater than	<>	not equal to	<i>exists</i>	test for the existence of rows
=	equal to	<i>not</i>	reverse logic	<i>in</i>	match any value in a set/list
<=	less than or equal to	<i>like</i>	string pattern matching	<i>some</i>	same as <i>any</i>

A logical comparison operator can be used to compare a specified value with all the values in a table column. When the query is executed, only the records with values matching the criteria appear in the result

Sometimes, a query requires more than one condition to obtain the desired result. For example, if we want to find all students in COMP3111 with a grade over 80 we would need two conditions: *courseCode='COMP3111'* and *grade>80*. A record must meet *both* conditions to be included in the result. To combine two criteria in this way, the *and* logical operator is used as shown in Figure 10(d). Alternatively, we may want to specify that the result records meet at least one (any one) of several conditions. For example, if we want to find all students in COMP3111 *or* all students with a grade over 80 we would need two conditions: *courseCode='COMP3111'* as well as *grade>80*. The records only need to meet *one* of the conditions in order to be included in the result. To combine two criteria in this way, the *or* logical operator is used.

To find data that is between two values, the *between and* operator can be used in a query. This operator can be used with a column of type text, numeric or date. For example, to find all students with a grade between 60 and 70, inclusive, we would enter *grade between 60 and 70* in the *where* clause. To find all students with a grade not between 60 and 70, inclusive, (i.e., with grade less than 60 or more than 70) we would enter *grade not between 60 and 70* in the *where* clause.

¹⁷ String values in a *where* clause need to be enclosed in single quotes.

Wildcard characters are used in a query to find records when the criterion contains a pattern (such as all last names beginning with W) or is only partly known (such as the proper spelling – Baker or Bakker). Wildcards take the place of one or several letters or numbers¹⁸. The SQL wildcards are:

- (*underscore*) matches any single letter or number (e.g., W_ng finds Wong, Wang, etc. whereas 'Cha_' finds Chan, Chau, etc.).
- % (*percent*) matches one or more letters or numbers (e.g., H% finds all records that start with H; 9%/13 finds all dates in September, 2013 (assuming a date format of mm/dd/yy); and %mail% finds all records that have the word “mail” as a value of the column in any record).
- [] matches any single character within the specified range ([a-f]) or set ([abcdef]) (e.g., [C-P]arsen finds author last names ending with arsen and starting with any single character between C and P, for example Carsen, Larsen, Karsen).
- [^] matches any single character not within the specified range ([^a-f]) or set ([^abcdef]) (e.g., de[^l]% finds all author last names starting with “de” and where the following letter is not “l”).

When data needs to be retrieved from more than one table the *from* clause specifies the tables from which the data should be retrieved and the *search_condition* of the *where* clause requires a *join_condition* (one or more) to indicate how the records from the tables should be matched. For example, if we want to retrieve the last name, first name and grade of all students, then we need to retrieve data from the *Student* table, because it contains the *firstName* and *lastName* columns, as well as from the *EnrolledIn* table, because it contains the *grade* column. To match the correct grade with the correct student we need to join (i.e., match up) records in the *Student* table with the corresponding records in the *EnrolledIn* table. This is done by matching values of the primary key column *id* in the *Student* table with corresponding values of the foreign key column *studentId* in the *EnrolledIn* table. A *join_condition* in the *where* clause specifies this matching as shown in Figure 10(e).

The *group by* clause is used to aggregate data and the *having* clause is used to specify a condition on what records to include in the aggregation¹⁹. For example, the SQL query to retrieve all students that have an average grade greater than 80 is shown in Figure 10(f)²⁰.

The *order by* clause specifies whether to sort the result and the columns on which to sort. The sort order can be ascending (*asc*), the default, or descending (*desc*) for each column.

The result of a SQL query can be treated as a set. Consequently, set operations can be applied to the results of two different queries. For example, the SQL query to retrieve those students who enrolled in both COMP 3111 and COMP 4311 in 2015 is shown in Figure 10(f). This query also illustrates the use of the *in* logical comparison operator, which determines whether a specified value matches any value in a set (subquery) or a list. The set operations supported by SQL Server are described in Appendix D.

Example Queries

The following queries illustrate the use of the Query Designer to specify some queries for the *Enrollment* database. Try to construct the SQL statement for each query yourself before looking at the solutions on the following pages.

1. Find the first name and last name of the students from Canada. Sort the result in ascending order first by last name and then by first name.
2. Find the first name and last name of the students whose last names contain the string “au”.
[Hint: You need to use the *like* comparison operator and the % wild card character.]
3. Find the first name and last name of the students whose last names start with either H or Z.
4. Find the average grade in COMP 3311 in the Spring 2015 semester.
5. Find the first name and last name of all students in ascending order with no duplication who have received a grade greater than or equal to 80 in any course.
[Hint: To specify that no duplication should appear in a result, you need to include the keyword “distinct” in the *select clause*.]
6. For each course, find the course title and the number of students from the USA enrolled in the course.

¹⁸ By default, SQL Server character comparison is case insensitive.

¹⁹ Appendix B summarizes the aggregate functions available in SQL.

²⁰ When a *group by* clause is used, SQL requires that any columns in the *selection_list* must also appear in the *column_specification* of the *group by* clause.

Example Queries Solutions

- Find the first name and last name of the students from Canada. Sort the result in ascending order first by last name and then by first name.
- Find the first name and last name of the students whose last names contain the string "au".

[Hint: You need to use the *like* comparison operator and the % wild card character.]

SQLQuery1.sql

```
select firstName, lastName
from Student
where country= 'Canada'
order by lastName, firstName
```

	firstName	lastName
1	Carol	Chan
2	Stephen	Ha
3	Tom	Hui
4	Louis	Lai
5	Douglas	Lam
6	Victor	Lam
7	Keith	Lo
8	Wendy	Poon
9	John	Tse
10	Ivy	Wu

SQLQuery2.sql

```
select firstName, lastName
from Student
where lastName like '%au%'
```

	firstName	lastName
1	Thomas	Au
2	Andy	Lau
3	Joe	Chau
4	Derek	Yau

- Find the first name and last name of the students whose last names start with either H or Z.
- Find the average grade in COMP 3311 in the Spring 2015 semester²¹.

SQLQuery3.sql

```
select firstName, lastName
from Student
where lastName like 'H%'
or lastName like 'Z%'
```

	firstName	lastName
1	James	Zhang
2	Celia	Hung
3	Stephen	Ha
4	Tom	Hui

SQLQuery4.sql

```
select avg(grade) as AverageGrade
from EnrolledIn
group by courseCode, semester, year
having courseCode='COMP3311'
and semester='Spring'
and year='2015'
```

	AverageGrade
1	72.058823

²¹ The column name "year" is coloured magenta because it is a reserved keyword of SQL. To ensure that it is treated as a column name, it should be enclosed in square brackets, which will remove the colouring.

5. Find the first name and last name of all students in ascending order with no duplication who have received a grade greater than or equal to 80 in any course.
[Hint: To specify that no duplication should appear in a result, you need to include the keyword “distinct” in the *select clause*.]
6. For each course, find the course title and the number of students from the USA enrolled in the course.

SQLQuery5.sql

```

select distinct firstName, lastName
from Student, EnrolledIn
where Student.id=EnrolledIn.studentId
and grade>=80
    
```

100 %

	firstName	lastName
1	Bill	Chan
2	Carmen	Wong
3	Derek	Yau
4	John	Kung
5	John	Tse
6	Joseph	Ng
7	Otto	Wong
8	Sandy	Ng
9	Stephen	Ha

SQLQuery6.sql

```

select title, count(*) as NumberUSAEnrolled
from Student, EnrolledIn, Course
where Student.id=EnrolledIn.studentId
and EnrolledIn.courseCode=Course.code
and country='USA'
group by title
    
```

100 %

	title	NumberUSAEnrolled
1	Database Management Systems	8
2	Software Engineering	7

APPENDIX A: SQL SERVER SUPPORTED DATA TYPES²²**Exact Numerics**

Data Type	Stores	Size
bigint	-2^{63} (-9,223,372,036,854,775,808) to $2^{63}-1$ (9,223,372,036,854,775,807)	8 bytes
int	-2^{31} (-2,147,483,648) to $2^{31}-1$ (2,147,483,647)	4 bytes
smallint	-2^{15} (-32,768) to $2^{15}-1$ (32,767)	2 bytes
tinyint	0 to 255	1 byte
bit	1, 0 or NULL	1 bit
decimal[(p, s)] numeric[(p, s)]	$10^{38}+1$ through $10^{38}-1$ <i>p</i> (precision) - The maximum total number of decimal digits that will be stored, both to the left and to the right of the decimal point. The precision must be a value from 1 through the maximum precision of 38. The default precision is 18. <i>s</i> (scale) - The number of decimal digits that will be stored to the right of the decimal point. This number is subtracted from <i>p</i> to determine the maximum number of digits to the left of the decimal point. Scale must be a value from 0 through <i>p</i> . Scale can be specified only if precision is specified. The default scale is 0; therefore, $0 \leq s \leq p$.	Maximum storage sizes depend on the precision
money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
smallmoney	- 214,748.3648 to 214,748.3647	4 bytes

Approximate Numerics

Data Type	Stores	Size
float[(n)]	- 1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308 <i>n</i> is the number of bits that are used to store the mantissa of the float number in scientific notation. If <i>n</i> is specified, it must be a value between 1 - 53. The default value of <i>n</i> is 53.	Depends on the value of <i>n</i>
real	- 3.40E + 38 to -1.18E - 38, 0 and 1.18E - 38 to 3.40E + 38	4 bytes

Date and Time

Data Type	Stores	Size
date	YYYY-MM-DD	3 bytes
datetime	YYYY-MM-DD hh:mm:ss[.n*] YYYY four digits from 1753 through 9999 that represent a year. MM two digits, ranging from 01 to 12, that represent a month in the specified year. DD two digits, ranging from 01 to 31 depending on the month, that represent a day of the specified month. hh two digits, ranging from 00 to 23, that represent the hour. mm two digits, ranging from 00 to 59, that represent the minute. ss two digits, ranging from 00 to 59, that represent the second. n* zero to three digits, ranging from 0 to 999, that represent the fractional seconds.	8 bytes
smalldatetime	YYYY-MM-DD hh:mm:ss (see datetime for format meaning)	4 bytes
time	hh:mm:ss[.nnnnnnnn]	3-5 bytes

²² For a description of all of the SQL Server data types see <http://msdn.microsoft.com/en-us/library/ms187752.aspx>.

Character Strings

Data Type	Stores	Size
char[(n)]	Fixed-length non-Unicode string data. <i>n</i> defines the string length and must be a value between 1 - 8,000.	n bytes
varchar[(n max)]	Variable-length non-Unicode string data. <i>n</i> defines the string length and must be a value between 1 - 8,000. <i>max</i> specifies the maximum storage size (i.e., $2^{31}-1$ bytes (2 GB)).	Actual length of data + 2 bytes
text	Variable-length non-Unicode data with a maximum string length of $2^{30} - 1$ (1,073,741,823) bytes.	Actual length of data in bytes

Unicode Character Strings

Data Type	Stores	Size
nchar[(n)]	Fixed-length Unicode string data. <i>n</i> defines the string length and must be a value between 1 - 4,000.	n bytes
nvarchar[(n max)]	Variable-length Unicode string data. <i>n</i> defines the string length and must be a value between 1 - 4,000. <i>max</i> specifies the maximum storage size (i.e., $2^{31}-1$ bytes (2 GB)).	Two times the actual length of data + 2 bytes
ntext	Variable-length Unicode data with a maximum string length of $2^{30} - 1$ (1,073,741,823) bytes.	Two times the actual length of data in bytes

Binary Strings

Data Type	Stores	Size
binary[(n)]	Fixed-length binary data with a length of <i>n</i> bytes, where <i>n</i> is a value from 1 through 8,000.	n bytes
varbinary[(n max)]	Variable-length binary data. <i>n</i> must be a value between 1 - 8,000. <i>max</i> specifies the maximum storage size (i.e., $2^{31}-1$ bytes (2 GB)).	Actual length of the data + 2 bytes
image	Variable-length binary data from 0 through $2^{31}-1$ (2,147,483,647) bytes.	Actual length of the data

Other Data Types

Data Type	Stores	Size
uniqueidentifier	A 16-byte globally unique identifier (GUID).	16 bytes

APPENDIX B: SQL SERVER AGGREGATE FUNCTIONS²³

avg	Returns the average of the values in a group. Null values are ignored.
count	Returns the number of items in a group.
max	Returns the maximum value in the expression.
min	Returns the minimum value in the expression.
sum	Returns the sum of all the values in the expression. Sum can be used with numeric columns only. Null values are ignored.
stdev	Returns the statistical standard deviation of all values in the specified expression.
var	Returns the statistical variance of all values in the specified expression.

²³ For a complete description of the SQL aggregate functions see <https://msdn.microsoft.com/en-us/library/ms173454.aspx>.

APPENDIX C: SQL SERVER QUERY TYPES

Select	<p>A select query, which is the most common type of query, retrieves data from one or more tables and displays the results. A select query can also be used to group records and calculate sums, counts, averages and other types of totals. The syntax of a select query is:</p> <pre>select [all distinct] { * column_specification } from table_specification [where search_condition] [group by column_specification] [having search_condition] [order by {column_specification [asc desc], ...}];</pre>
Insert	<p>An insert query creates a new row and inserts values into either all or specified columns. The syntax of an insert query that inserts values into all columns is:</p> <pre>insert into table_name values (value1, value2, value3, ...);</pre> <p>The syntax of an insert query that inserts values into specified columns is:</p> <pre>insert into table_name (column1, column2, column3, ...) values (value1, value2, value3, ...);</pre> <p>Columns for which no value is specified are set to null.</p>
Update	<p>An update query makes global changes by changing the values of individual columns in a group of records in one or more tables. Which records are updated in tables other than in the table specified, depends on the referential integrity constraints defined for the tables. The syntax of an update query is:</p> <pre>update table_name set column1=value1, column2=value2, ... [where search_condition];</pre> <p>The <i>where</i> clause specifies which record or records should be updated. Omitting the <i>where</i> clause results in all records being updated.</p>
Delete	<p>A delete query deletes a group of records from one or more tables. Which records are deleted in tables other than in the table specified, depends on the referential integrity constraints defined for the tables. The syntax of a delete query is:</p> <pre>delete from table_name [where search_condition];</pre> <p>The <i>where</i> clause specifies which record or records should be deleted. Omitting the <i>where</i> clause results in all records being deleted.</p>

APPENDIX D: SQL SERVER SET OPERATIONS

union [all]	<i>union</i> specifies that multiple result sets are to be combined and returned as a single result set with duplicates removed. <i>union all</i> incorporates all rows into the results including duplicates.
intersect	Returns any distinct values that are returned by both the query on the left and right sides of the <i>intersect</i> operator.
except	Returns any distinct values from the query to the left of the <i>except</i> operator that are not also returned from the right query.