Section One – Absolute Fundamentals

Section Background

In this section, you learn the absolute fundamentals of SQL – creating and dropping a table, getting data into the table, listing the data in the table, and deleting and updating the data. You will be working with a PetStore table that has basic information about dogs as pets. When you have completed some steps in the section, the PetStore table will look as illustrated below.

PetStore Table

Name: VARCHAR(64)	Breed: VARCHAR(32)	BirthDate: DATE	Price: DECIMAL(6,2)
Angel	Golden Retriever	01-MAR-2019	89.99

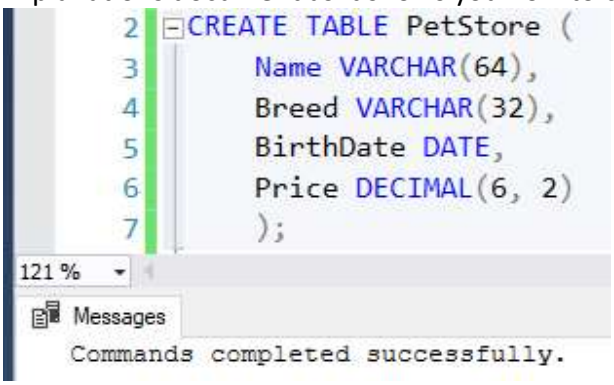
You will create this table and try out SQL commands using the table.

Do not worry if you do not recognize the structure and datatypes in the table above. The Lab 1 Explanation document and supporting lecture and textbook readings give you the information you need. Start reading the explanation document first, then iteratively complete the steps below. Each step below has an accompanying explanation in the explanation document.

For each step that requires SQL, *make sure to capture a screenshot of the command and the results of its execution.* Submissions that do not contain screenshots will be returned to you. A screenshot is more legible if you use one of the many free tools to capture only the relevant portion of the screen, rather than capturing the entire application window. A few steps ask for explanations rather than SQL; no screenshot is needed for such steps.

Section Steps

1. *Creating a Table* – Create the PetStore table. As a reminder, make sure to follow along in the Lab 1 Explanations document as it shows you how to create tables and complete the other steps.



2. *Inserting a Row* – Insert the first row where the title is “Angel”, the breed is “Golden Retriever”, the birth date is 3/1/2019, and the price is $89.99.

3. *Selecting All Rows* – Select all rows in the table to view the row you inserted.

4. *Updating All Rows* – Update the price of the row in the table to $99.99, then select all rows in the table to view the row you updated.

5. *Deleting All Rows* – Remove all rows from the table, then select all rows in the table to verify there are no rows.

6. *Dropping a Table* – Drop the PetStore table, then select all rows in the table to verify the table doesn’t exist. Explain how you would use the error message, in conjunction with the SELECT command, to diagnose the error.

The error message says that the command that caused the error starts in line 33. So the problem must be with SELECT * FROM PetStore. The error message also outputs that the object name ‘PetStore’ is invalid.
Looking back at the command, it can be seen that PetStore comes directly after FROM and must be therefore the table name. With this it can be concluded that the error is output because the object (here the object is a table) PetStore does not exist. Which makes sense since table PetStore was dropped before SELECT * FROM PetStore was executed.

Section Two – More Precise Data Handling

Section Background

In this section, you enhance your skills by more precisely working with data. In the prior section, you learned to work with all rows in the table. In this section, you add to that by learning to pinpoint specific rows to be retrieved, modified, or deleted. You also learn how to add SQL constraints to your table, and to work with nulls.

You will work with a vacation table, which will ultimately look like the below when all steps have been completed.

Vacation Table

VacationId: DECIMAL(12) Primary Key	Location: VARCHAR(64) NOT NULL	Description: VARCHAR(1024) NULL	StartedOn: DATE NOT NULL	EndedOn: DATE NOT NULL
1	Costa Rica	Relaxing Hot Springs	1/13/2019	1/21/2019
2	Bora Bora	Exciting Snorkeling	3/5/2019	3/15/2019
3	Jamaica		12/10/2018	12/28/2018

Section Steps

7. *Table Setup* – Create the Vacation table with its columns, datatypes, and constraints.

```
37  CREATE TABLE Vacation (
38      VacationId DECIMAL(12) PRIMARY KEY,
39      Location VARCHAR(64) NOT NULL,
40      Description VARCHAR(1024),
41      StartedOn DATE NOT NULL,
42      EndedOn DATE NOT NULL
43      );
```

121 %

Messages

```
Commands completed successfully.
```

8. *Table Population* – Insert the rows illustrated in the figure below. Note that the description for Jamaica is null.

Vacation Table

VacationId: DECIMAL(12) Primary Key	Location: VARCHAR(64) NOT NULL	Description: VARCHAR(1024) NULL	StartedOn: DATE NOT NULL	EndedOn: DATE NOT NULL
1	Costa Rica	Relaxing Hot Springs	1/13/2019	1/21/2019
2	Bora Bora	Exciting Snorkeling	3/5/2019	3/15/2019
3	Jamaica		12/10/2018	12/28/2018

```
46  INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
47  VALUES (1, 'Costa Rica', 'Relaxing Hot Springs', '1/13/2019', '1/21/2019');
48
49  INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
50  VALUES (2, 'Bora Bora', 'Exciting Snorkeling', '3/5/2019', '3/15/2019');
51
52  INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
53  VALUES (3, 'Jamaica', NULL, '12/10/2018', '12/28/2018');
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)
```

Select all rows from the Vacation table to show that the inserts were successful.

```
55  SELECT *
56  FROM Vacation;
```

121 %

Results | Messages

	VacationId	Location	Description	StartedOn	EndedOn
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Bora	Exciting Snorkeling	2019-03-05	2019-03-15
3	3	Jamaica	NULL	2018-12-10	2018-12-28

9. *Invalid Insertion* – Attempt to insert a row with the following values. The insert command will fail because the location column must have a value.
VacationId = 4
Location = NULL
Description = Experience the Netherlands No Other Way
StartedOn = 1/1/2020
EndedOn = 1/10/2020

```
59  INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
60  VALUES (4, NULL, 'Experience the Netherlands No Other Way', '1/1/2020', '1/10/2020');
```

121 %

Messages

```
Msg 515, Level 16, State 2, Line 59
Cannot insert the value NULL into column 'Location', table 'Assignment1.dbo.Vacation'; column does not allow nulls. INSERT fails
The statement has been terminated.
```

Explain how you would interpret the error message conclude that the location column is missing a required value.
The error message is descriptive and says exactly what the problem is: The command that starts on line 59 tries to insert NULL (NULL is used to represent 'no value') into the column named Location in the Vacation table, but this column Location does not allow NULL, meaning the Location column is missing a required (non-NULL) value. Also, the entire insert statement was unsuccessful / failed, thus no values were inserted into the table.

10. *Valid Insertion* – Now insert the row with a location intact, with the following values.
    VacationId = 4
    Location = Netherlands
    Description = Experience the Netherlands No Other Way
    StartedOn = 1/1/2020
    EndedOn = 1/10/2020

```
63  INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
64  VALUES (4, 'Netherlands', 'Experience the Netherlands No Other Way', '1/1/2020', '1/10/2020')
```

121 %

Messages

```
(1 row affected)
```

11. *Filtered Results* – Retrieve only the location and description for the Bora Bora vacation, using the primary key as the column that determines which row is retrieved. Explain why it is useful to limit the number of rows and columns returned from a SELECT statement.

```
67  SELECT Location, Description
68  FROM Vacation
69  WHERE VacationId = 2;
```

121 %

Results | Messages

	Location	Description
1	Bora Bora	Exciting Snorkeling

    It is useful to limit the number of rows and columns returned from a SELECT statement because it makes the SELECT statement more efficient: returning a large number of irrelevant columns and rows can impact the performance (all values must be searched and loaded into memory). And to only return the columns and rows that are of interest: if possible, the user should avoid having to search through the returned columns and rows again (for example, if the user only wants the maximum value in a column, the user should just return the maximum value instead of returning the whole column and then scrolling/searching through all values until the maximum value is found).

12. *Targeted Update* – The Jamaica vacation has no description. Update the row so that its description is “Aquatic Wonders”. Select all rows in the table to show that the update was successful.

```
72 UPDATE Vacation
73 SET Description = 'Aquatic Wonders'
74 WHERE Location = 'Jamaica';
75
76 SELECT *
77 FROM Vacation;
```

121 %

Results | Messages

	VacationId	Location	Description	StartedOn	EndedOn
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Bora	Exciting Snorkeling	2019-03-05	2019-03-15
3	3	Jamaica	Aquatic Wonders	2018-12-10	2018-12-28
4	4	Netherlands	Experience the Netherlands No Other Way	2020-01-01	2020-01-10

13. *Updating to Null* – Update the Jamaica vacation so that it no longer has the description (i.e. its description is null). Select all rows in the table to show that the update was successful.

```
80 UPDATE Vacation
81 SET Description = NULL
82 WHERE Location = 'Jamaica';
83
84 SELECT *
85 FROM Vacation;
```

121 %

Results | Messages

	VacationId	Location	Description	StartedOn	EndedOn
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Bora	Exciting Snorkeling	2019-03-05	2019-03-15
3	3	Jamaica	NULL	2018-12-10	2018-12-28
4	4	Netherlands	Experience the Netherlands No Other Way	2020-01-01	2020-01-10

14. *Targeted Deletion* – Delete all rows where the vacation started on a date greater than June 1st, 2019, by using the StartedOn column as the determinant of which rows are deleted. Select all rows in the table to show the delete was successful.

```
88 DELETE FROM Vacation
89 WHERE StartedOn > '6/1/2019';
90
91 SELECT *
92 FROM Vacation;
```

121 %

Results | Messages

	VacationId	Location	Description	StartedOn	EndedOn
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Bora	Exciting Snorkeling	2019-03-05	2019-03-15
3	3	Jamaica	NULL	2018-12-10	2018-12-28

Section Three – Data Anomalies and Formats

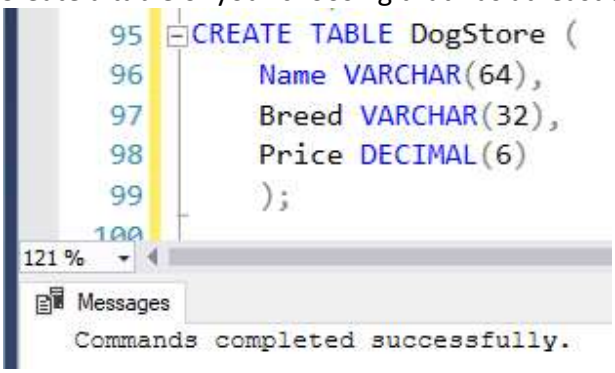
Section Background

When the same data is repeated multiple times, anomalies can result. In this section, you demonstrate and explore three such anomalies in a relational database – insert, update, and delete anomalies.

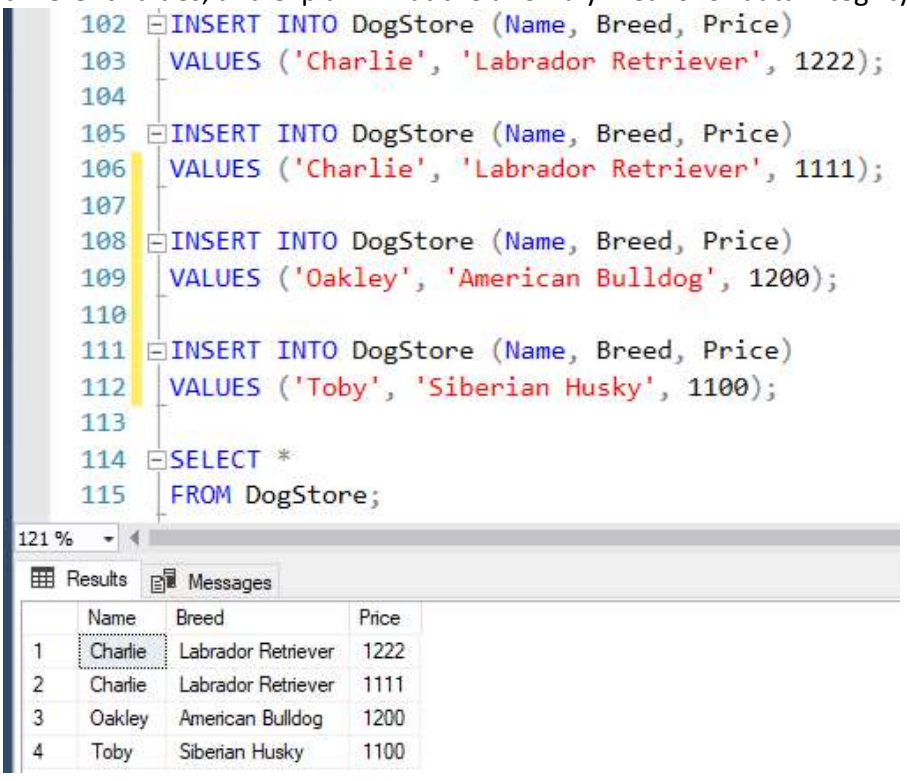
Databases provide several advantages over files. In this section, you explore putting the same data in a relational table and in a file, then compare the two.

Section Steps

15. *Data Anomalies* – In this step you demonstrate anomalies that can occur in improperly designed tables.
    a. Create a table of your choosing that has at least three columns.



    b. Using the table, demonstrate an anomaly that occurs when the same data is inserted multiple times with different values, and explain what the anomaly means for data integrity.



The third and fourth INSERT statements do not lead to an anomaly, but the first and second INSERT statements do. By inserting entries that have the same values in the Name and Breed column, but different values in the Price column, a data anomaly was created (they lead to ambiguous data in the table): it is unclear whether both dogs happen to be called Charlie, are Labrador Retrievers, and cost different amounts; or whether it is one and the same dog. Also, if they are two different dogs, it is not clear which dog is supposed to cost $1222 and which is $1111. And if it is one and the same dog, then it is also not clear which price is the correct price. This means that data integrity (data accuracy and consistency) was not ensured because to ensure data integrity "(…) use integrity constraints to enforce the business rules associated with the database and prevent the entry of invalid (or ambiguous) information into tables" – however, as shown above, the first and second INSERT statements added ambiguous information to the table.

c. Using the table, demonstrate a deletion anomaly with SQL, and explain what the anomaly means for data integrity.

```
118 DELETE FROM DogStore
119 WHERE Name = 'Charlie' AND Breed = 'Labrador Retriever';
120
121 SELECT *
122 FROM DogStore;
```

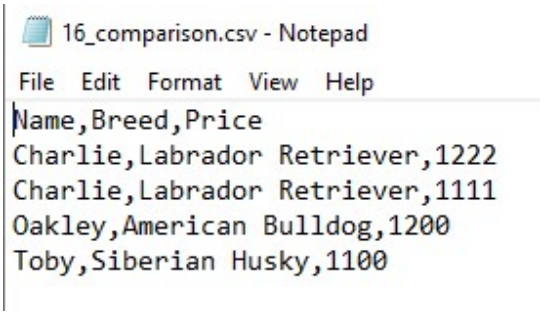
121 %

Results | Messages

	Name	Breed	Price
1	Oakley	American Bulldog	1200
2	Toby	Siberian Husky	1100

'A deletion anomaly occurs when the deletion of data inadvertently deletes other associated data'. If it is assumed that there are two different dogs named 'Charlie', then the intention of the database user was probably to delete only one dog 'Charlie' from the DogStore table via the DELETE statement above, but two rows (dogs) were deleted; either only the dog with the price $1222 should have been deleted or the dog with the price $1111. So here one of the prices was inadvertently deleted (associated data). Thus, data integrity (data accuracy and consistency) was not ensured; however, data integrity could have been maintained by an integrity constraint (e.g., use of a primary key).

16. *File and Database Table Comparison* – In this step you compare the table created in #15 with a file that contains all of the same information.
    a. Create a file in any format you'd like that contains all the same columns and at least 4 rows of information as the table you created in #15. There are many formats you can use. Some examples include XML, flatfile, binary, text, and json; this list is not exhaustive. All columns and at least 4 rows should be present in the file in its new format. Make sure to provide the file or a screenshot of the file and to explain your choices.



I decided to use a CSV (comma-separated values) file. Since values in a CSV file are separated by commas and since a CSV file is a text file (not a binary file), it is widely used in industry: The file is easy to understand, lightweight, and the values are easy to extract by parsing each row. The first row contains the column names separated by commas and the rows below contain the values for the columns. Thus, the first column has the name 'Name' and the values 'Charlie', 'Charlie', 'Oakley', 'Toby'.

b. With a few paragraphs, compare what it's like to access data in the table versus in the file. You may need to first research how applications typically access data in this type of file. Make sure to at least use these comparison points:
   i. Efficiency – If there were millions of rows of data, would it be more efficient to access a single record in the relational table, or the file, and why?
   With millions of rows of data, it would be more efficient to access a single record in the relational table than in the CSV file. Let us imagine the above relational table and the CSV file contained millions of dog names, breeds, and prices. To search the CSV file for a specific dog name, a row would have to be read, then the row would have to be split after each comma to get the individual column values of the row, and then checked if the values match the search criteria. And that needs to be done for each row in the CSV file. On the other hand, the relational table has a structure: when searching for dog name, no rows (strings) have to be split up; it is sufficient to search only the 'Name' column. Likewise, the relational tables could have indexes that further speed up access to frequently used/searched values. However, such indexes cannot be created and used for CSV files to make searching more efficient.

   ii. Security – Imagine you needed to restrict access to one specific row/record, allowing only one person to access it, while the rest of the rows could be accessed by many people. Would it be easier or more difficult to secure this row in the relational table compared to the file, and why?
   This would be much more difficult to do in the CSV file than in the relational table. In CSV files it is not possible to do this without using custom software; the prerequisite would be that the CSV file is only opened by all users via this custom software, that all instances of the software that have this CSV file open communicate with each other, and that the custom software manages the access control. However, if common programs such as a simple text editor are used to open the CSV file, then such access control is impossible. Because 'file systems only support security on the file itself, but not on the data within the file'. But relational databases on the other hand 'support row and field level security natively'. Such access control could be implemented using sessions and locks in SQL.

   iii. Structural Independence – Imagine the table structure was modified by adding or taking away columns, and equivalent changes were made to the file. Would these changes affect an app using the table differently than an app using the file, and why?
   Assuming that the app accesses the relational table and the CSV file correctly, adding or removing columns to the relational table would have little or no effect on the app. For example, a SELECT statement used in the app would continue to work after adding columns, however, when removing columns, the SELECT statement might try to access columns that no longer exist, leading to an error. The same goes for other SQL statements. However, there is a major benefit to using relational tables. It can be

ruled out that the app destroys the data integrity by adding or removing columns in the relational table because of a changing location (if column A is located besides column B or columns A is located besides columns C makes no difference). However, the situation is completely different with the CSV file. In the CSV file, if a column (values with a comma after it) is added in the middle, or if one of the columns that is not the last column is deleted, then the app, when reading the values from the CSV file by location, would read the incorrect values, which can be a problem. In summary, it can be said that a change in the CSV file and relational table can render an app inoperable or have no impact on the app. However, changing the relational table tends to be less of a problem for an app because in the CSV file the location where the data is in the file matters, while in the relational table it does not.