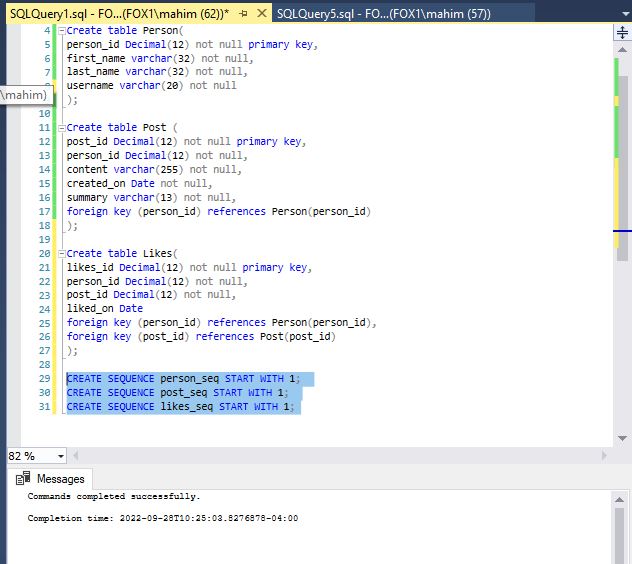
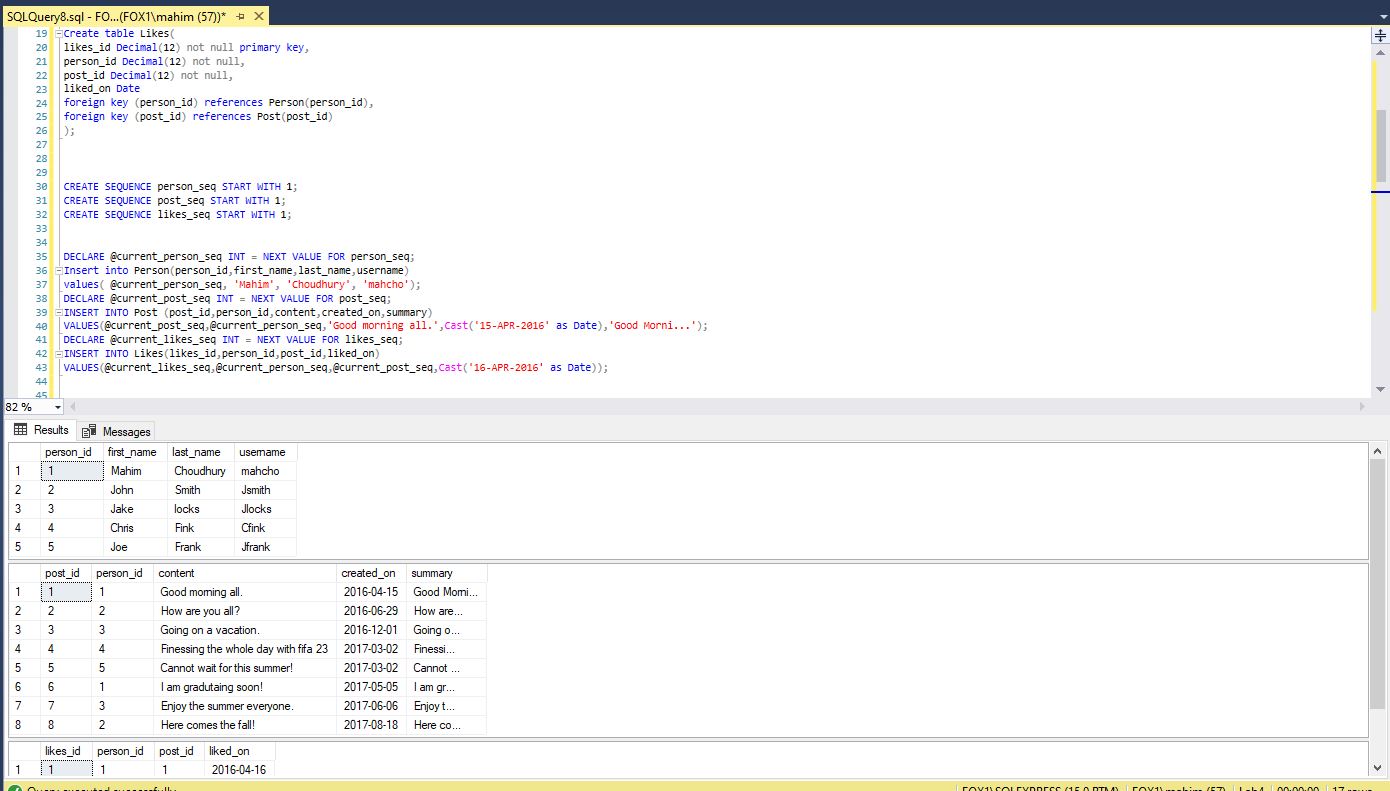
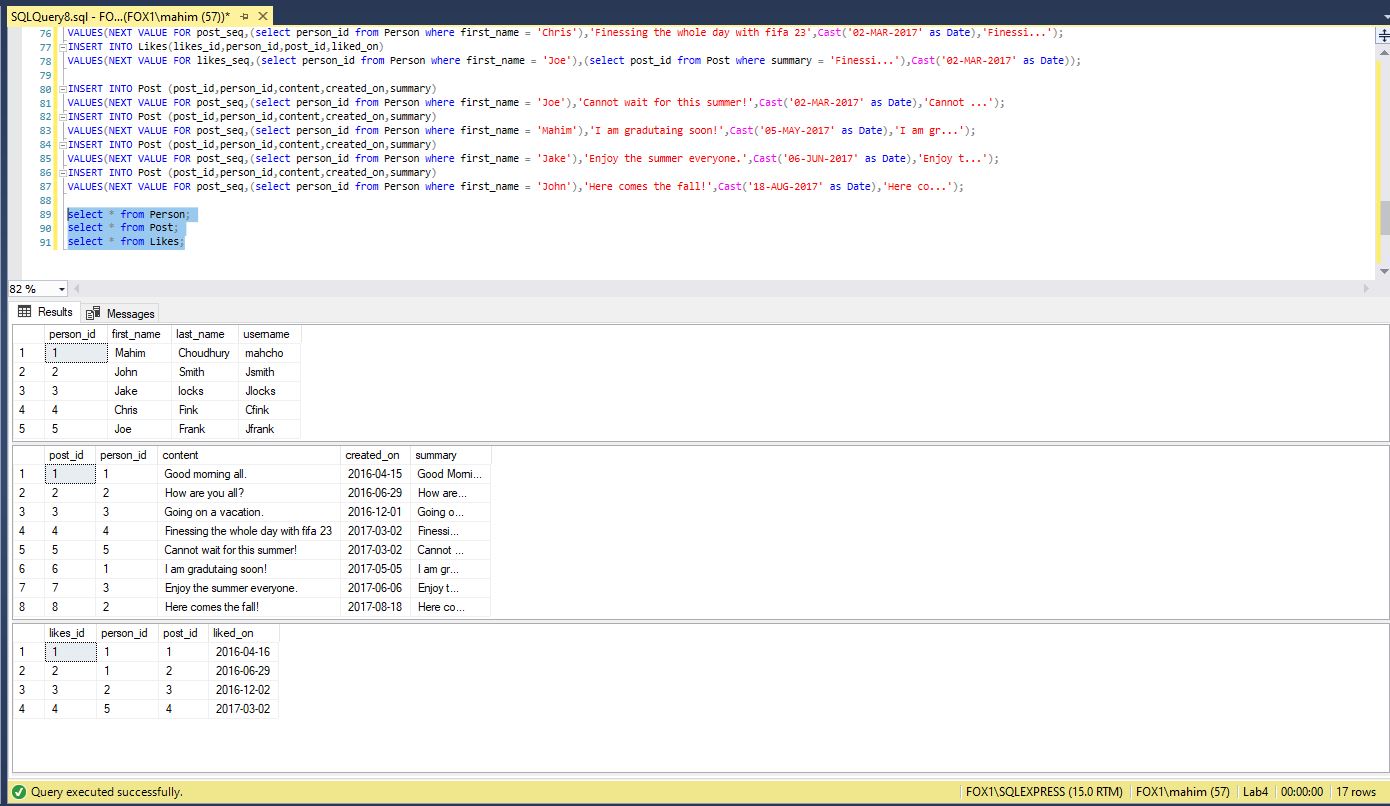
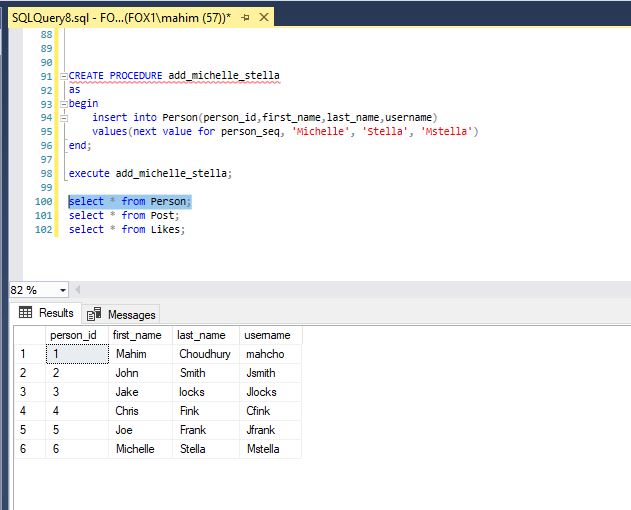
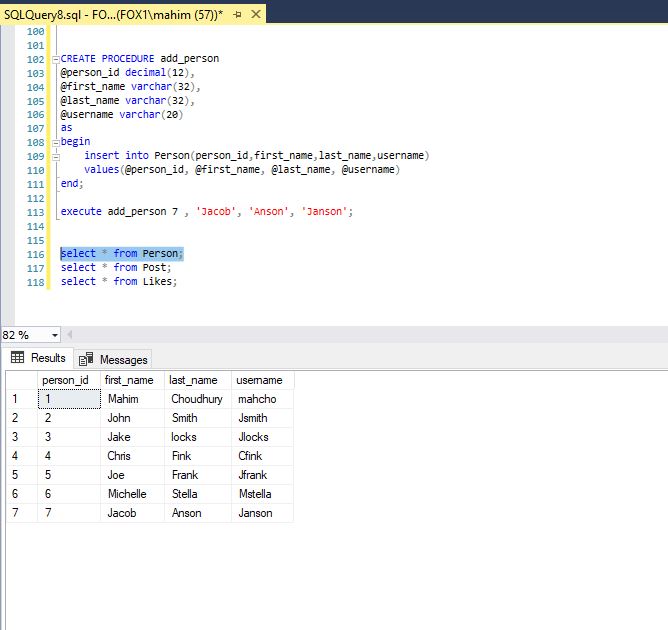
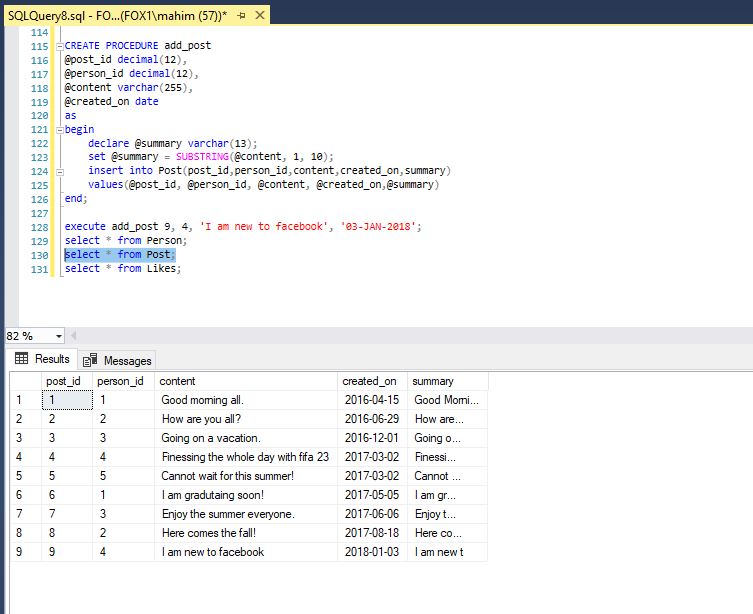
1. *Create Table Structure –* Create the tables in the social networking schema, including all of their columns, datatypes, and constraints. Create sequences for each table; these will be used to generate the primary and foreign key values in Step #2.  
   
2. *Populate Tables –* Populate the tables with data, ensuring that there are at least 5 people, at least 8 posts, and at least 4 likes. Make sure to use sequences to generate the primary and foreign key values. Most of the fields are self-explanatory. As far as the “content” field in Post, make them whatever you like, such as “Take a look at these new pics” or “Just arrived in the Bahamas”, and set the summary as the first 10 characters of the content, followed by “…”.



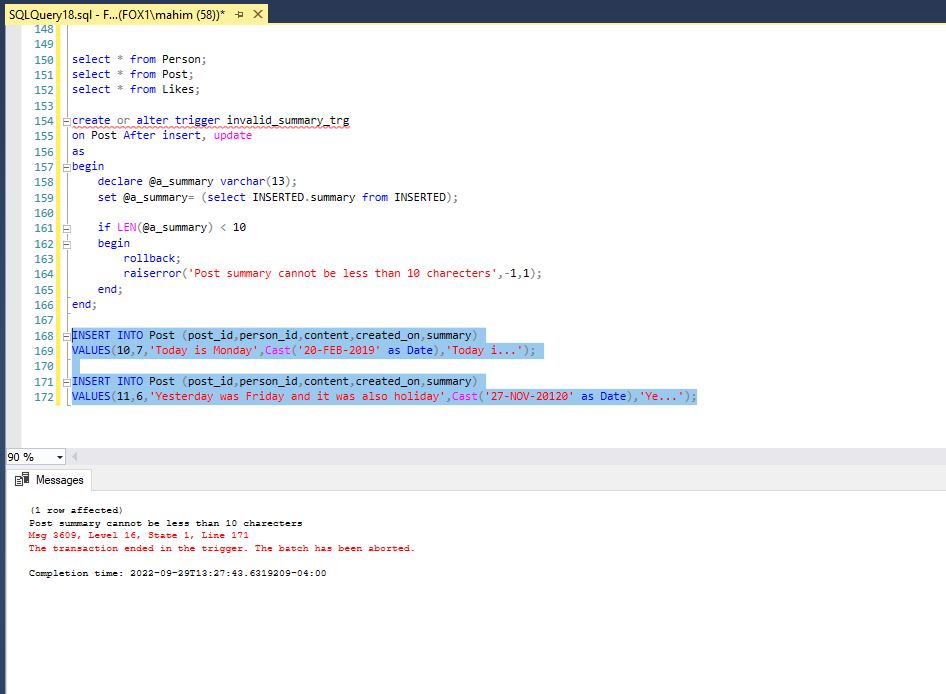


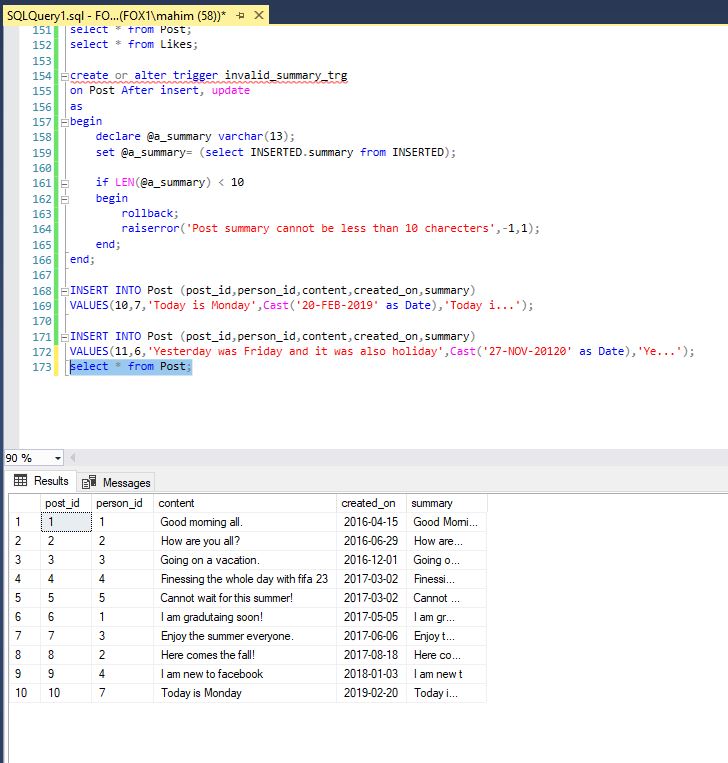
1. *Create Hardcoded Procedure –* Create a stored procedure named “add\_michelle\_stella” which has no parameters and adds a person named “Michelle Stella” to the Person table. Execute the stored procedure, and list out the rows in the Person table to show that Michelle Stella has been added.  
   
2. *Create Reusable Procedure –* Create a reusable stored procedure named “add\_person” that uses parameters and allows you to insert any new person into the Person table. Execute the stored procedure with a person of your choosing, then list out the Person table to show that the person was added to the table.   
   
3. *Create Deriving Procedure –* Create a reusable stored procedure named “add\_post” that uses parameters and allows you to insert any new post into the Post table. Instead of passing in the summary as a parameter, derive the summary from the content, storing the derivation temporarily in a variable (which is then used as part of the insert statement). Recall that the summary field stores the first 10 characters of the content followed by “…”. Execute the stored procedure to add a post of your choosing, then list out the Post table to show that the addition succeeded.  
   
4. *Create Lookup Procedure –* Create a reusable stored procedure named “add\_like” that uses parameters and allows you to insert any new “like”. Rather than passing in the person\_id value as a parameter to identify which person is liking which post, pass in the username of the person. The stored procedure should then lookup the person\_id and store it in a variable to be used in the insert statement. Execute the procedure to add a “like” of your choosing, then list out the Like table to show the addition succeeded.  
   Graphical user interface, text, application

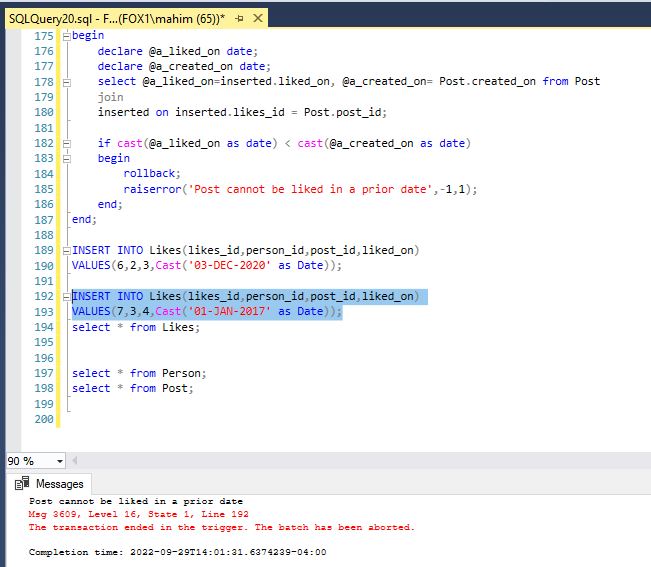
   Description automatically generated

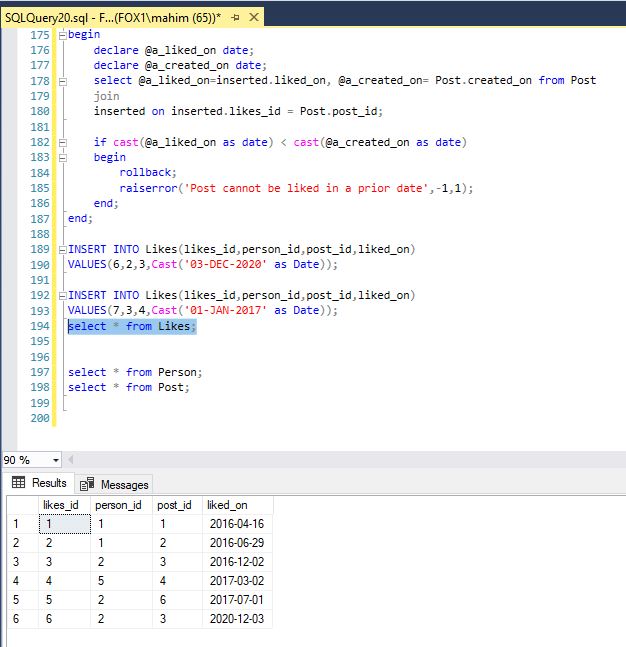
For some reason, it wasn’t allowing me to do the @a\_username to be used. Could you please provide the solution to this? In the live recording, this step was also skipped.

1. *Single Table Validation Trigger –* One practical use of a trigger is validation within a single table (that is, the validation can be performed by using columns in the table being modified). Create a trigger that validates that the summary is being inserted correctly, that is, that the summary is actually the first 10 characters of the content followed by “…”. The trigger should reject an insert that does not have a valid summary value. Verify the trigger works by issuing two insert commands – one with a correct summary, and one with an incorrect summary. List out the Post table after the inserts to show one insert was blocked and the other succeeded.

**



1. *Cross-Table Validation Trigger –* Another practical use of a trigger is cross-table validation (that is, the validation needs columns from at least one table external to the table being updated). Create a trigger that blocks a “like” from being inserted if its “liked\_on” date is before the post’s “created\_on” date. Verify the trigger works by inserting two “likes” – one that passes this validation, and one that does not. List out the Likes table after the inserts to show one insert was blocked and the other succeeded.   
   



1. *History Trigger –* Another practical use of trigger is to maintain a history of values as they change. Create a table named post\_content\_history that is used to record updates to the content of a post, then create a trigger that keeps this table up-to-date when updates happen to post contents. Verify the trigger works by updating a post’s content, then listing out the post\_content\_history table (which should have a record of the update).
2. *Creating Normalized Table Structure –* For this question, you create a set of normalized tables based upon the scenario given, and also identify some functional dependencies between the given fields.  
     
   This scenario involves a court which handles cases between a plaintiff and defendant. Here are some rules the govern how the court operates.

* The court has a list of cases it’s working with at any one time.
* Each case has one plaintiff and one defendant.
* Each case has one or more court appearances, where the plaintiff, defendant, and their attorneys attend and decisions are made about the case.
* There can be only one court appearance per day for the same case. There may be multiple appearances on the same day, but only for different cases.
* Each plaintiff and defendant may retain multiple attorneys for each court appearance.
* Multiple decisions about the case may be made at each court appearance.
* Every decision at a court appearance is assigned a number, such as decision1, decision2, and so on. This way the decision can be formally referred to by its number for an appearance.
* In a similar fashion, every attorney attending a court appearance is assigned a number, such as attorney1, attorney2, and so on.

Currently, after a court appearance is held, the court saves information a spreadsheet with each the following fields.

|  |  |
| --- | --- |
| **Field** | **Description** |
| case\_number | This is a unique number assigned to each case. Court staff refer to a case by this number. |
| case\_description | This is an explanation of what the case is about. |
| plaintiff\_first\_name | This is the first name of the plaintiff in the case. |
| plaintiff\_last\_name | This is the last name of the plaintiff in the case. |
| defendant\_first\_name | This is the first name of the defendant in the case. |
| defendant\_last\_name | This is the last name of the defendant in the case. |
| attorney1\_first\_name | This is the first name of an attorney that represents the plaintiff or defendant at the court appearance. |
| attorney1\_last\_name | This is the last name of an attorney that represents the plaintiff or defendant at the court appearance. |
| attorney2\_first\_name | This is the first name of an attorney that represents the plaintiff or defendant at the court appearance. |
| attorney2\_last\_name | This is the last name of an attorney that represents the plaintiff or defendant at the court appearance. |
| attorney3\_first\_name | This is the first name of an attorney that represents the plaintiff or defendant at the court appearance. |
| attorney3\_last\_name | This is the last name of an attorney that represents the plaintiff or defendant at the court appearance. |
| appearance\_date | This is the date a court appearance was held. |
| number\_attending | This is the number of people attending the court appearance. |
| decision1\_description | This is the first decision made at the court appearance, if any. |
| decision2\_description | This is the second decision made at the court appearance, if any. |
| extra\_appearance\_notes | If there are more than three attorneys or more than two decisions at a court appearance, this notes field identifies them. Additional appearance related information may also be stored here. |

The court would like to upgrade to using a relational database to store their information going forward.

1. Identify all functional dependencies in the set of fields listed above in the spreadsheet. These can be listed in the form of:   
     
   column1,column2,… 🡺 column3, column4…  
     
   Make sure to explain your reasoning for the functional dependency choices.

Suggest a set of normalized relational tables derived from how the court operates and the fields they store. Create a DBMS physical ERD representing this set of tables, which contains the entities, primary and foreign keys, attributes, relationships, and relationship constraints. You may add synthetic primary keys where needed. Make sure that the tables are normalized to BCNF, and to explain your choices.