Summary & Reflection

The database is for a web-based client called Automation Portal, which tracks a production laboratory's orders as they progress through the production process, as well as the permissions of the account handling them.

The structure of my database has been updated a great deal, mostly to prevent errors before the DBMS was used to perform queries. I edited an entity so that its primary key was no longer an email address, as those are bound to change and can lead to errors later down the road. I filled in missing foreign keys from the entities they reference. I introduced additional attributes to entities that previously only has one in order to give them purpose as entities within the system. I adjusted the datatype of some attributes—such as AnOrder.TargetConcentration—to accurately reflect the value it represents. These adjustments span from minor to major in terms of implementation, but all play a role in improving the efficacy of my DBMS.

I have improved upon the foundation of my SQL script, introducing stored procedures to insert rows to the tables I created in the previous iteration. I also developed a history table to track any updates that happen to the attribute, AnOrder.TargetConcentration—as adjustments to this critically affect the day-to-day operations supported by the DBMS. Entires to this history tables are recorded automatically though the use of a trigger that activates each time the attribute is updated.

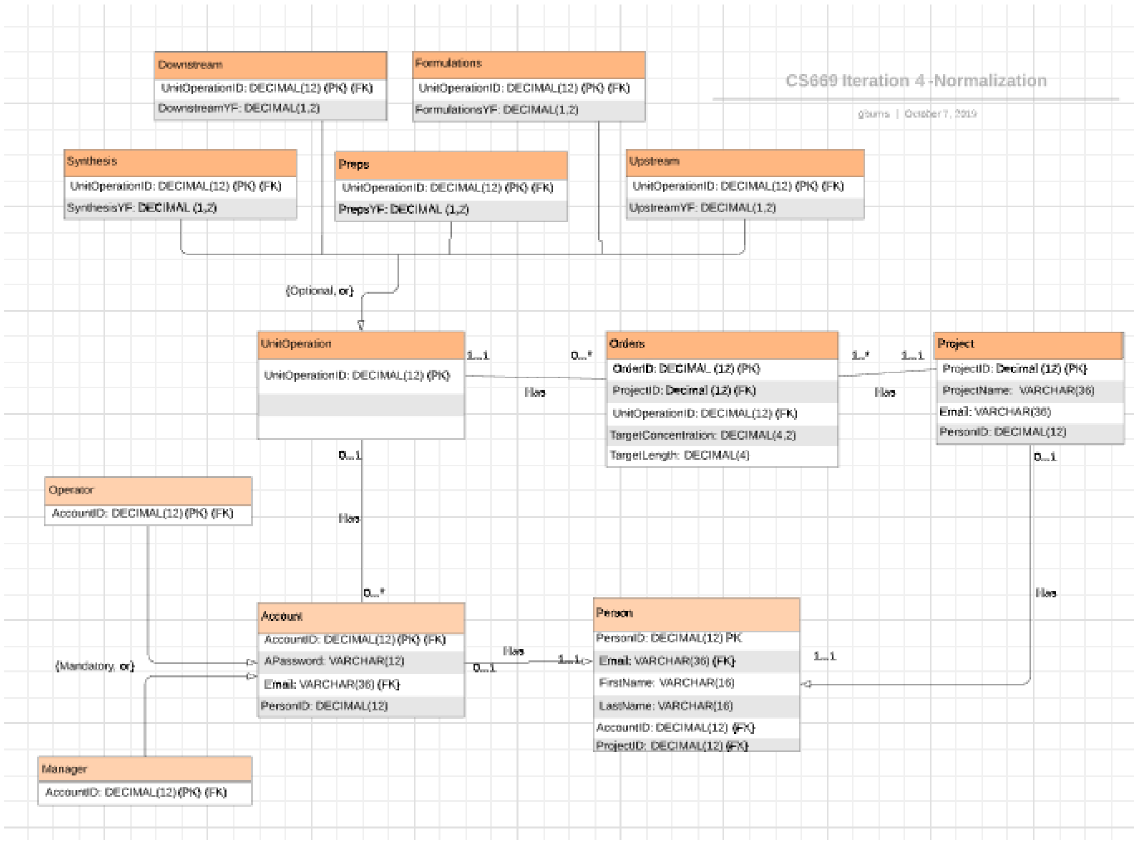
I also presented various organizational-driven queries. This was the most exciting stage of this project for me, as it actually gave me an opportunity to play with the system I had created and use it to access information in ways I had been planning since we laid out the foundation in the first iteration. While I may have not completely fulfilled the construct guidelines for my second query, I feel as though the use-case is one that would often present itself in the work environment supported by the DBMS and that it had practicality.

If there were a sixth project iteration, I would continue to introduce fields to my database and focus on organization driven queries surrounding the unit operations entities that I had introduced in the previous iteration. I would also make some adjustments to the constraints tracking my AccountID attribute, as this attribute is not currently constrained by a foreign key between Account and Operator/Manager, and led to some confusion during my insertions. While there are other suggestions I would have for this DBMS, I am satisfied with the progress I have made since the first iteration and have learned a great deal from the project!

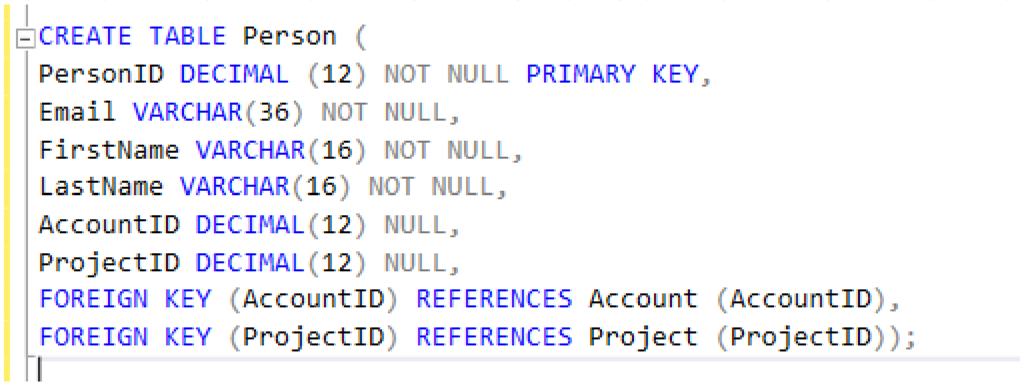
Updates Before beginning Final Iteration

Update 1: Replacing Email Address Primary Key with Synthetic Primary Key

Following feedback from iteration 4, the physical ERD and corresponding SQL code has been updated to adjust the Person entity so that the Email attribute is no longer a primary key. This edit was included because it is possible that users' emails may change, and doing so may cause future issues with primary key references. A new attribute, PersonID, has been added as the primary key for this entity; email has been made a non-key attribute. This allows the DBMS to retain the same information without the unneeded risk of reference errors.



CS669 Iteration 4 -Normalization
gburns | October 7, 2019
Downstream
UnitOperationID: DECIMAL(12) (PK) (FK)
DownstreamYF: DECIMAL(1,2)
Formulations
UnitOperationID: DECIMAL(12) (PK) (FK)
FormulationsYF: DECIMAL(1,2)
Synthesis
UnitOperationID: DECIMAL(12) (PK) (FK)
SynthesisYF: DECIMAL (1,2)
Preps
UnitOperationID: DECIMAL(12) (PK) (FK)
PrepsYF: DECIMAL (1,2)
Upstream
UnitOperationID: DECIMAL(12) (PK) (FK)
UpstreamYF: DECIMAL(1,2)
{Optional, or}
UnitOperation
UnitOperationID: DECIMAL(12) (PK)
1...1
0...*
Has
Orders
OrderID: DECIMAL (12) (PK)
ProjectID: Decimal (12) (FK)
UnitOperationID: DECIMAL(12) (FK)
TargetConcentration: DECIMAL(4,2)
TargetLength: DECIMAL(4)
1...*
1...1
Has
Project
ProjectID: Decimal (12) (PK)
ProjectName: VARCHAR(36)
Email: VARCHAR(36)
PersonID: DECIMAL(12)
0...1
Has
0...1
Operator
AccountID: DECIMAL(12) (PK) (FK)
Has
0...*
Account
AccountID: DECIMAL(12) (PK) (FK)
APassword: VARCHAR(12)
Email: VARCHAR(36) (FK)
PersonID: DECIMAL(12)
{Mandatory, or}
Has
0...1
1...1
Person
PersonID: DECIMAL(12) PK
Email: VARCHAR(36) (FK)
FirstName: VARCHAR(16)
LastName: VARCHAR(16)
AccountID: DECIMAL(12) (FK)
ProjectID: DECIMAL(12) (FK)
1...1
Manager
AccountID: DECIMAL(12) (PK) (FK)

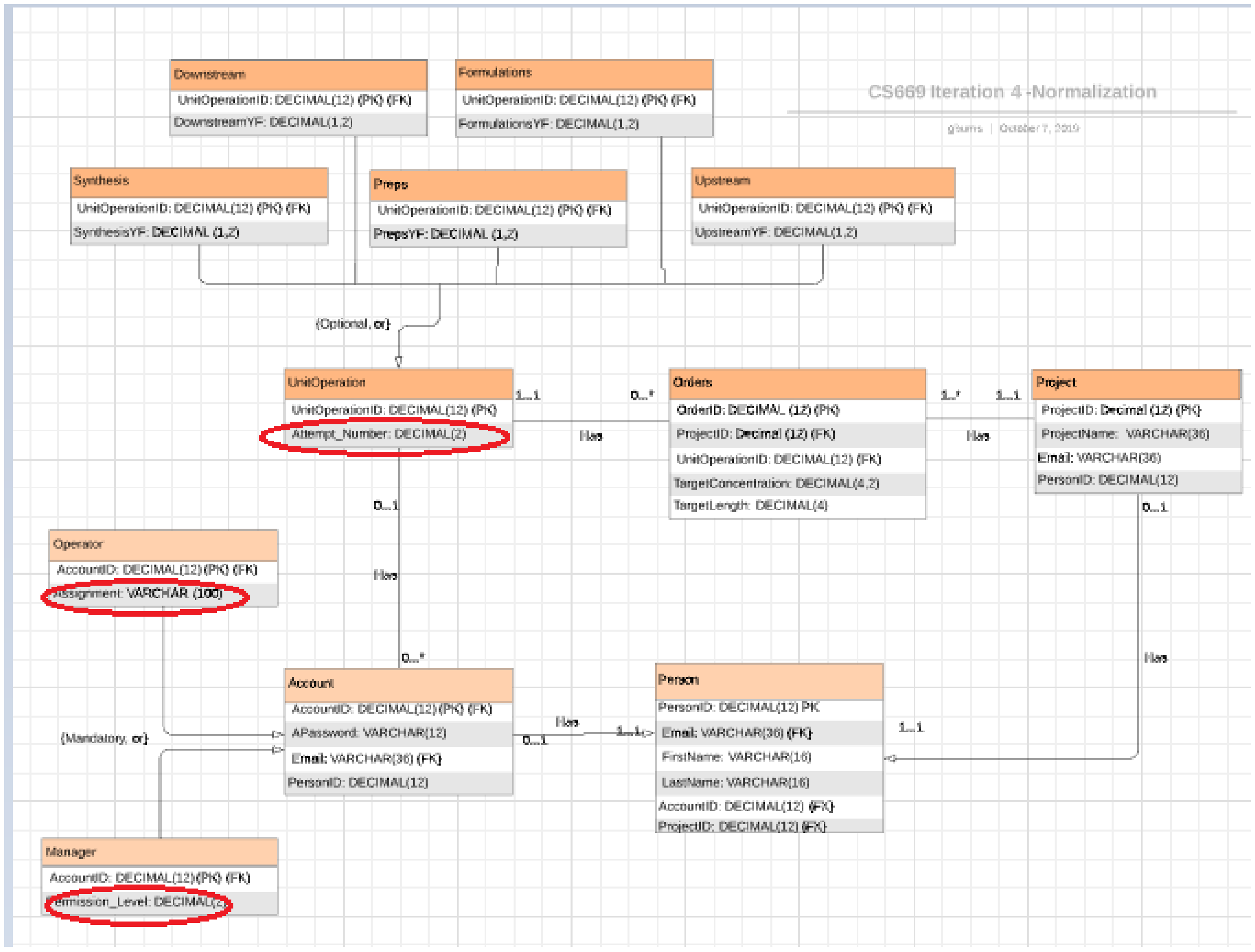


```
CREATE TABLE Person (
PersonID DECIMAL (12) NOT NULL PRIMARY KEY,
Email VARCHAR(36) NOT NULL,
FirstName VARCHAR(16) NOT NULL,
LastName VARCHAR(16) NOT NULL,
AccountID DECIMAL(12) NULL,
ProjectID DECIMAL(12) NULL,
FOREIGN KEY (AccountID) REFERENCES Account (AccountID),
FOREIGN KEY (ProjectID) REFERENCES Project (ProjectID));
```

Update 2: Adding Columns to Entities with only one Attribute

The columns circled below have been added to the ERD and corresponding SQL code.

1) UnitOperation.Attempt_Number: Sometimes a unit operation must be performed more than once. This attribute indicates what number attempt is being performed.
2) Operator.Assignment: A string representation of an operator's duty for the day.
3) Manager.Permission_Level: A decimal value indicating what level of permission a manager has within the Automation Portal.


CS669 Iteration 4 -Normalization
gburns | October 7, 2019
Downstream
UnitOperationID: DECIMAL(12) (PK) (FK)
DownstreamYF: DECIMAL(1,2)
Formulations
UnitOperationID: DECIMAL(12) (PK) (FK)
FormulationsYF: DECIMAL(1,2)
Synthesis
UnitOperationID: DECIMAL(12) (PK) (FK)
SynthesisYF: DECIMAL (1,2)
Preps
UnitOperationID: DECIMAL(12) (PK) (FK)
PrepsYF: DECIMAL (1,2)
Upstream
UnitOperationID: DECIMAL(12) (PK) (FK)
UpstreamYF: DECIMAL(1,2)
{Optional, or}
UnitOperation
UnitOperationID: DECIMAL(12) (PK)
Attempt_Number: DECIMAL(2)
1...1
0...*
Has
Orders
OrderID: DECIMAL (12) (PK)
ProjectID: Decimal (12) (FK)
UnitOperationID: DECIMAL(12) (FK)
TargetConcentration: DECIMAL(4,2)
TargetLength: DECIMAL(4)
1...*
1...1
Has
Project
ProjectID: Decimal (12) (PK)
ProjectName: VARCHAR(36)
Email: VARCHAR(36)
PersonID: DECIMAL(12)
0...1
0...1
Has
Operator
AccountID: DECIMAL(12) (PK) (FK)
Assignment: VARCHAR (100)
0...*
Has
{Mandatory, or}
Account
AccountID: DECIMAL(12) (PK) (FK)
APassword: VARCHAR(12)
Email: VARCHAR(36) (FK)
PersonID: DECIMAL(12)
Has
0...1
1...1
Person
PersonID: DECIMAL(12) PK
Email: VARCHAR(36) (FK)
FirstName: VARCHAR(16)
LastName: VARCHAR(16)
AccountID: DECIMAL(12) (FK)
ProjectID: DECIMAL(12) (FK)
1...1
Manager
AccountID: DECIMAL(12) (PK) (FK)
Permission_Level: DECIMAL(2)

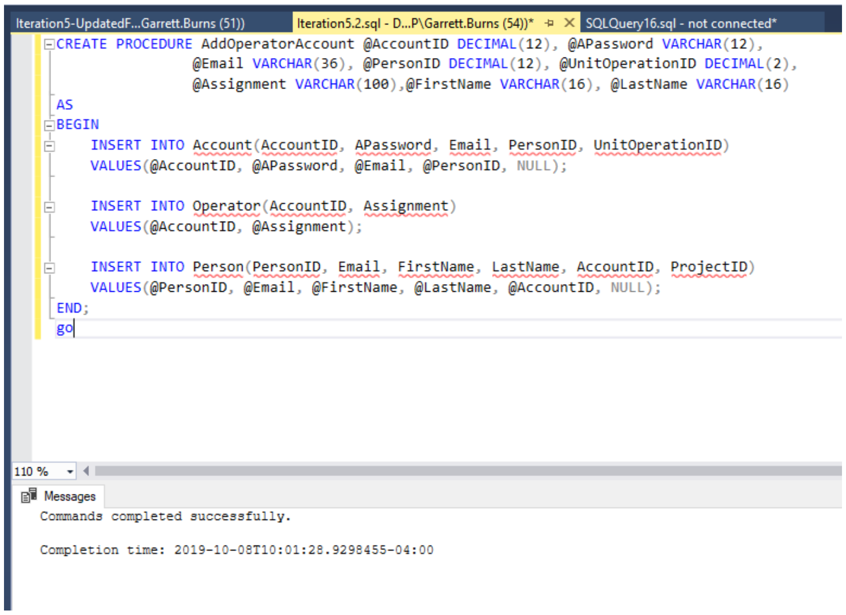
Transaction Driven, Reusable Stored Procedures

Use Case 1: User creates account in Automation Portal

1. User visits URL for web portal login page

2. User selects if they are an “operator” or “manager”
3. User inputs login credentials
4. Credentials are added to the database

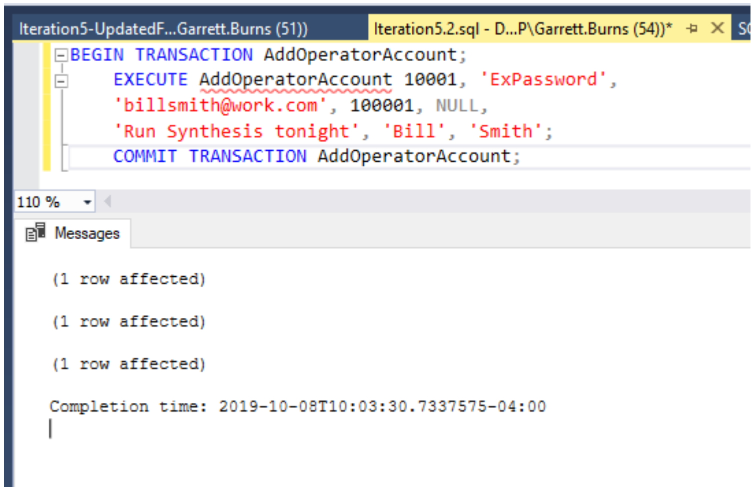
The procedure below, titled AddOperatorAccount, adds an operator to the database. Since an account is either an operator or a manager, and all accounts are people, the Account, Operator, and Person tables receive insertions.


Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
SQLQuery16.sql - not connected*
110 %
Messages

```
CREATE PROCEDURE AddOperatorAccount @AccountID DECIMAL(12), @APassword VARCHAR(12),
                 @Email VARCHAR(36), @PersonID DECIMAL(12), @UnitOperationID DECIMAL(2),
                 @Assignment VARCHAR(100),@FirstName VARCHAR(16), @LastName VARCHAR(16)
AS
BEGIN
    INSERT INTO Account(AccountID, APassword, Email, PersonID, UnitOperationID)
    VALUES(@AccountID, @APassword, @Email, @PersonID, NULL);

    INSERT INTO Operator(AccountID, Assignment)
    VALUES(@AccountID, @Assignment);

    INSERT INTO Person(PersonID, Email, FirstName, LastName, AccountID, ProjectID)
    VALUES(@PersonID, @Email, @FirstName, @LastName, @AccountID, NULL);
END;
go
```

```
Commands completed successfully.

Completion time: 2019-10-08T10:01:28.9298455-04:00
```

Here is a screenshot of the stored procedure execution in which an operator named Bill Smith is added to the database.


Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
110 %
Messages

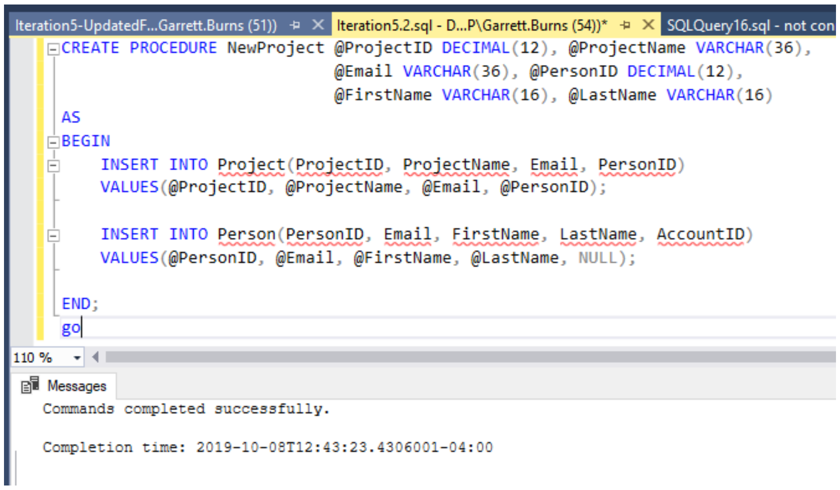
```
BEGIN TRANSACTION AddOperatorAccount;
    EXECUTE AddOperatorAccount 10001, 'ExPassword',
    'billsmith@work.com', 100001, NULL,
    'Run Synthesis tonight', 'Bill', 'Smith';
    COMMIT TRANSACTION AddOperatorAccount;
```

```
(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2019-10-08T10:03:30.7337575-04:00
```

Use Case 2: Adding a Project

1. A customer visits the Automation Portal and clicks "Create Project"
2. The customer Enters their information into the following fields: Project ID, Project Name, Email, Person ID
3. The Automation Portal stores a new Project in the database with the above attributes

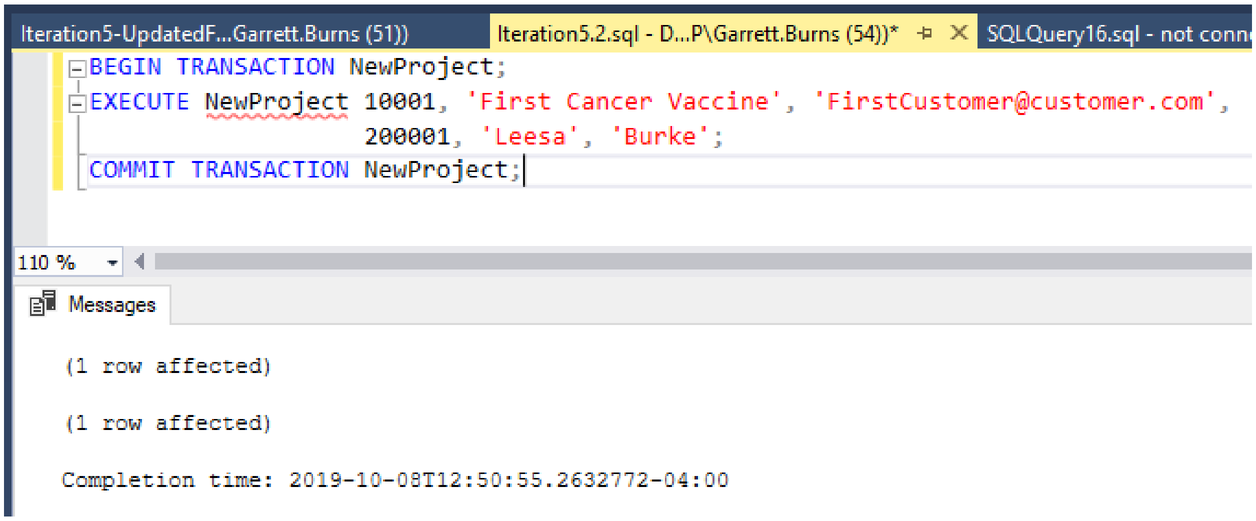
Below is the creation of the second procedure, NewProject, which adds a new row to the Project table and the customer's information as a new row to the Person table.


Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
SQLQuery16.sql - not con
110 %
Messages

```
CREATE PROCEDURE NewProject @ProjectID DECIMAL(12), @ProjectName VARCHAR(36),
                            @Email VARCHAR(36), @PersonID DECIMAL(12),
                            @FirstName VARCHAR(16), @LastName VARCHAR(16)
AS
BEGIN
    INSERT INTO Project(ProjectID, ProjectName, Email, PersonID)
    VALUES(@ProjectID, @ProjectName, @Email, @PersonID);

    INSERT INTO Person(PersonID, Email, FirstName, LastName, AccountID)
    VALUES(@PersonID, @Email, @FirstName, @LastName, NULL);

END;
go
```

```
Commands completed successfully.

Completion time: 2019-10-08T12:43:23.4306001-04:00
```

An example of the NewProject procedure, used to add the first project named 'First Cancer vaccine', submitted by customer Leesa

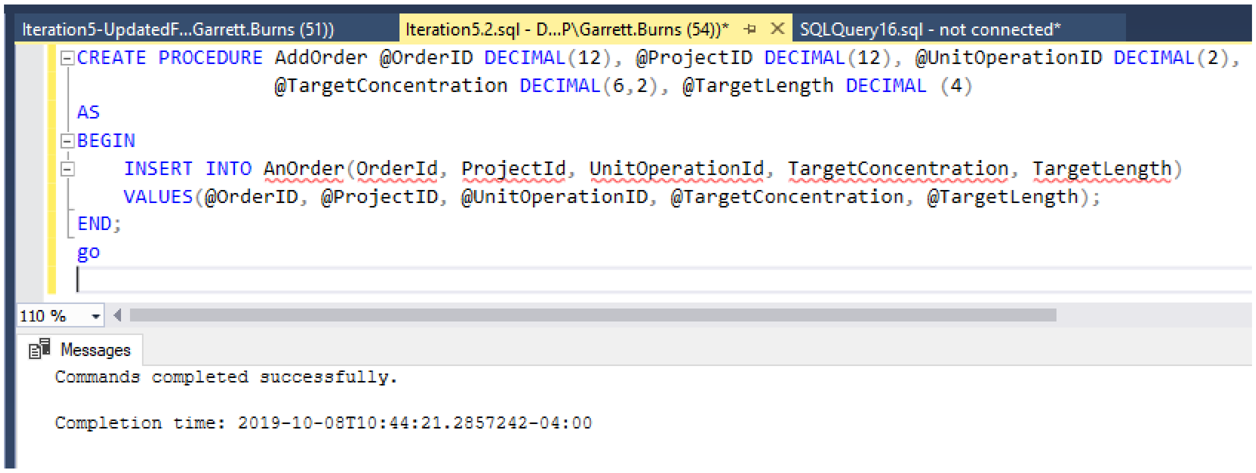

Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
SQLQuery16.sql - not conn
110 %
Messages

```
BEGIN TRANSACTION NewProject;
EXECUTE NewProject 10001, 'First Cancer Vaccine', 'FirstCustomer@customer.com',
                   200001, 'Leesa', 'Burke';
COMMIT TRANSACTION NewProject;
```

```
(1 row affected)

(1 row affected)

Completion time: 2019-10-08T12:50:55.2632772-04:00
```

Use Case 3: A new Order is added to the Automation Portal
1. User selects the “Create new order” button
2. User inputs fields for “Order ID”, “Project ID” “Unit Operation ID” “Target Concentration” and “Target Length.”
3. A new order is created and stored in the database with the above fields as its attributes.

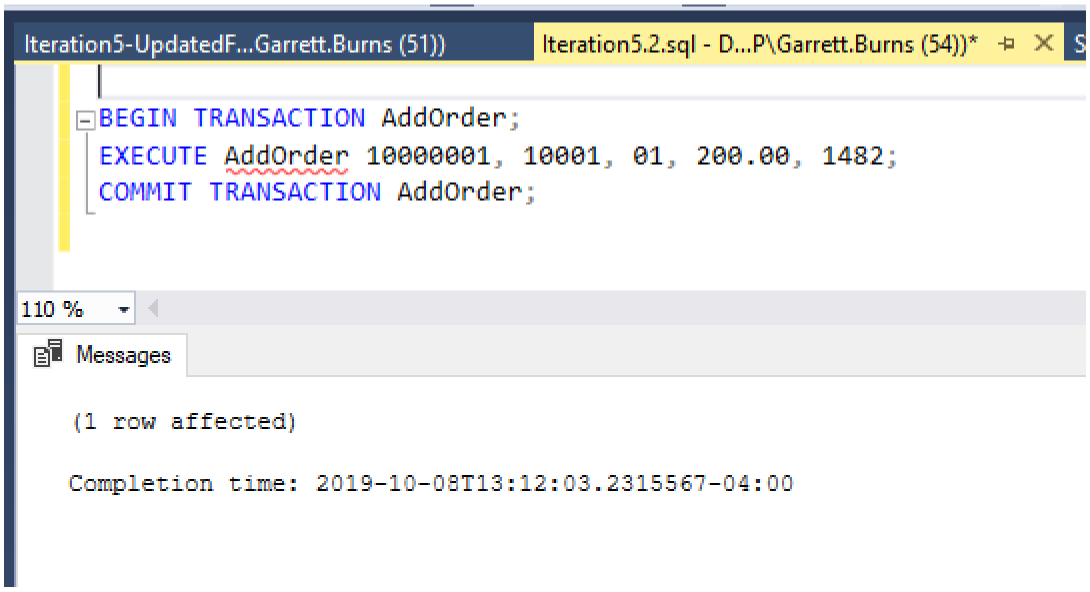
The procedure below, titled AddOrder, creates a new order in the database as a row in the AnOrder table, with the fields listed in step 2 as its attributes.


Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
SQLQuery16.sql - not connected*
110 %
Messages

```
CREATE PROCEDURE AddOrder @OrderID DECIMAL(12), @ProjectID DECIMAL(12), @UnitOperationID DECIMAL(2),
                    @TargetConcentration DECIMAL(6,2), @TargetLength DECIMAL (4)
AS
BEGIN
    INSERT INTO AnOrder(OrderId, ProjectId, UnitOperationId, TargetConcentration, TargetLength)
    VALUES(@OrderID, @ProjectID, @UnitOperationID, @TargetConcentration, @TargetLength);
END;
go
```

```
Commands completed successfully.

Completion time: 2019-10-08T10:44:21.2857242-04:00
```

The AddOrder procedure is demonstrated below, adding the first row to the AnOrder table of the database.


Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
110 %
Messages

```
BEGIN TRANSACTION AddOrder;
EXECUTE AddOrder 10000001, 10001, 01, 200.00, 1482;
COMMIT TRANSACTION AddOrder;
```

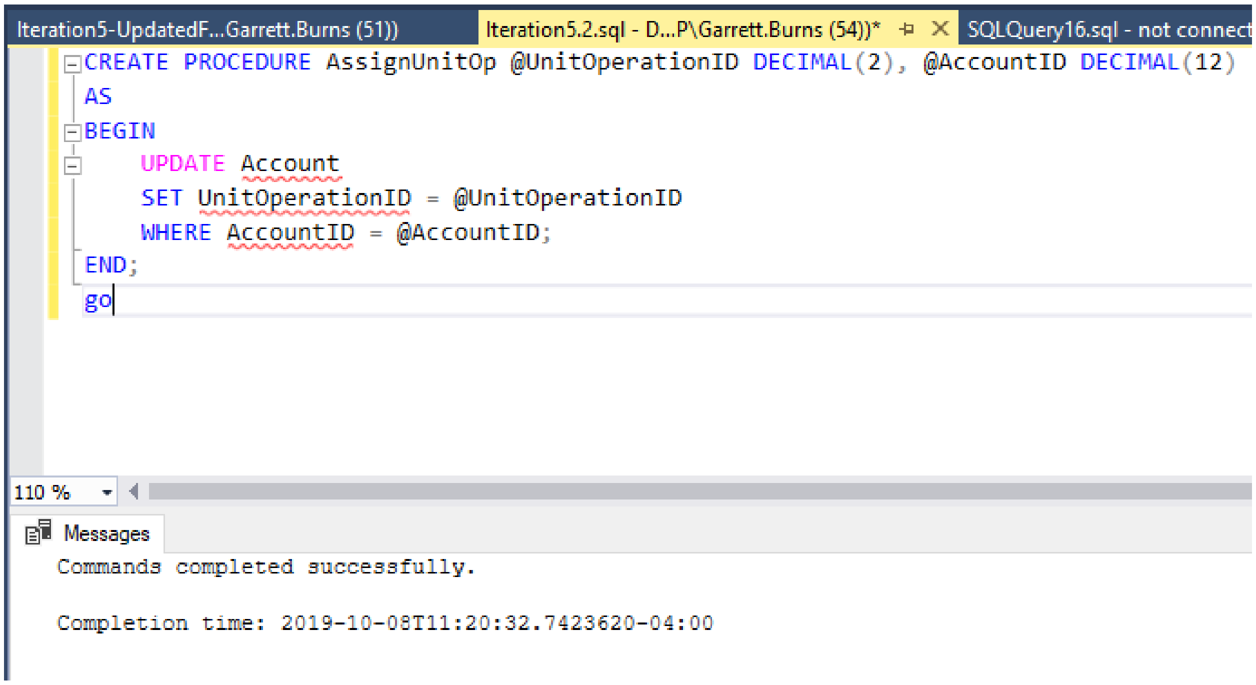
```
(1 row affected)

Completion time: 2019-10-08T13:12:03.2315567-04:00
```

Use Case 4: An operator selects a unit operation

1. Once logged into the Automation portal, the operator selects “Unit Operations”
2. The user selects from the list of unit operations: Synthesis, Preps, Upstream, Downstream, Formulations
3. The user enters their AccountID

4: The database updates the operator’s unit operation

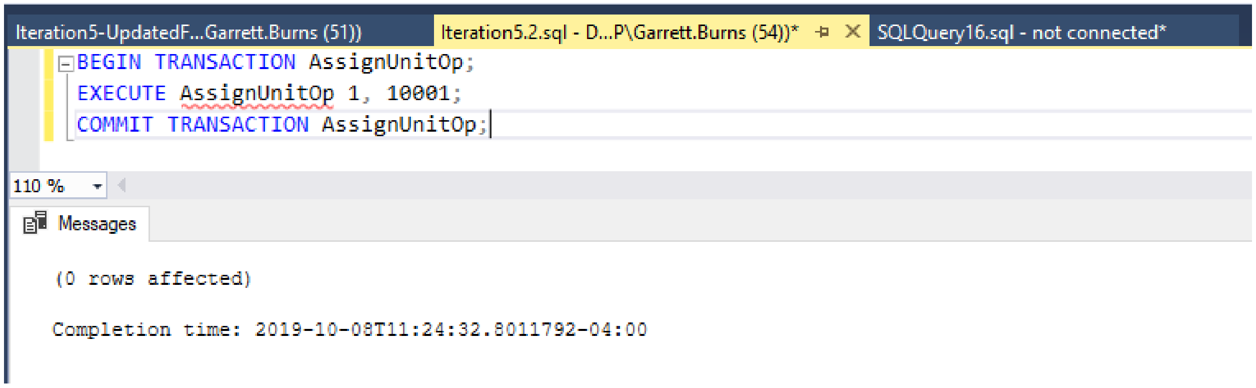
Below is the creation of the procedure AssignUnitOp, which updates an operator’s account to assign them with a selected unit operation.


Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
SQLQuery16.sql - not connect
110 %
Messages

```
CREATE PROCEDURE AssignUnitOp @UnitOperationID DECIMAL(2), @AccountID DECIMAL(12)
AS
BEGIN
    UPDATE Account
    SET UnitOperationID = @UnitOperationID
    WHERE AccountID = @AccountID;
END;
go
```

```
Commands completed successfully.

Completion time: 2019-10-08T11:20:32.7423620-04:00
```

In the example below, the AssignUnitOp procedure is used to assign the first Operator (Bill) the first unit operation (Synthesis.)


Iteration5-UpdatedF...Garrett.Burns (51))
Iteration5.2.sql - D...P\Garrett.Burns (54))*
SQLQuery16.sql - not connected*
110 %
Messages

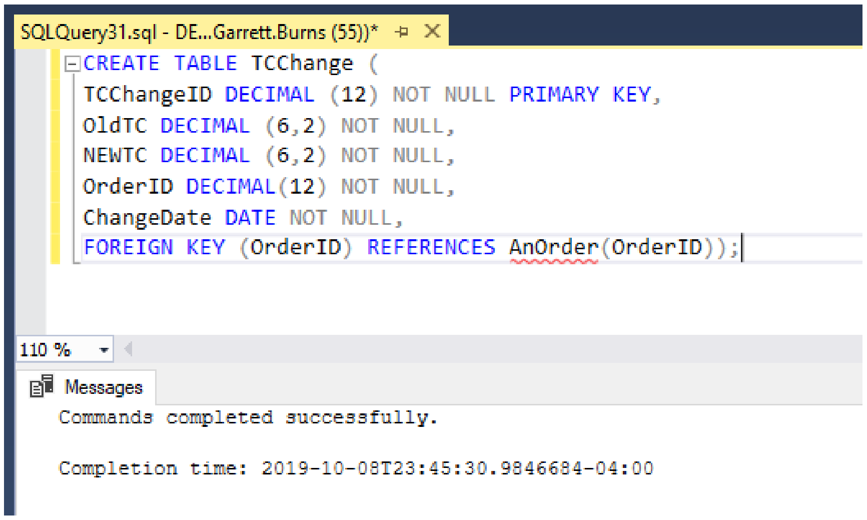
```
BEGIN TRANSACTION AssignUnitOp;
EXECUTE AssignUnitOp 1, 10001;
COMMIT TRANSACTION AssignUnitOp;
```

```
(0 rows affected)

Completion time: 2019-10-08T11:24:32.8011792-04:00
```

Maintaining History Table with Triggers

In the production lab this DBMS is modeled to support, there are often situations where a customer will update the lab and share that they need more or less product. The lab responds by scaling up or down the target concentration for the order the customer has created. To track these updates to orders in the DBMS, a history table and trigger can be created to log when the target concentration for an order is updated. This is demonstrated below using the history table "TCChange" and the trigger "TCChangeTrigger."

Here is a screenshot of the history table, TCChange:



SQLQuery31.sql - DE...Garrett.Burns (55))*

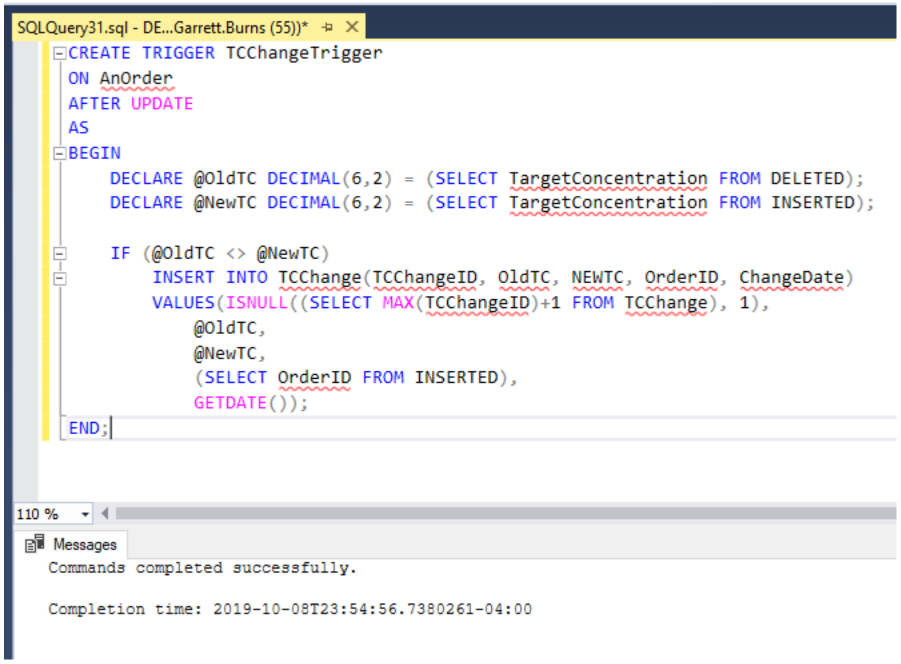
```sql
CREATE TABLE TCChange (
TCChangeID DECIMAL (12) NOT NULL PRIMARY KEY,
OldTC DECIMAL (6,2) NOT NULL,
NEWTC DECIMAL (6,2) NOT NULL,
OrderID DECIMAL(12) NOT NULL,
ChangeDate DATE NOT NULL,
FOREIGN KEY (OrderID) REFERENCES AnOrder(OrderID));
```

110 %

Messages

```
Commands completed successfully.

Completion time: 2019-10-08T23:45:30.9846684-04:00
```

And now its trigger, TCChangeTrigger:



SQLQuery31.sql - DE...Garrett.Burns (55))*

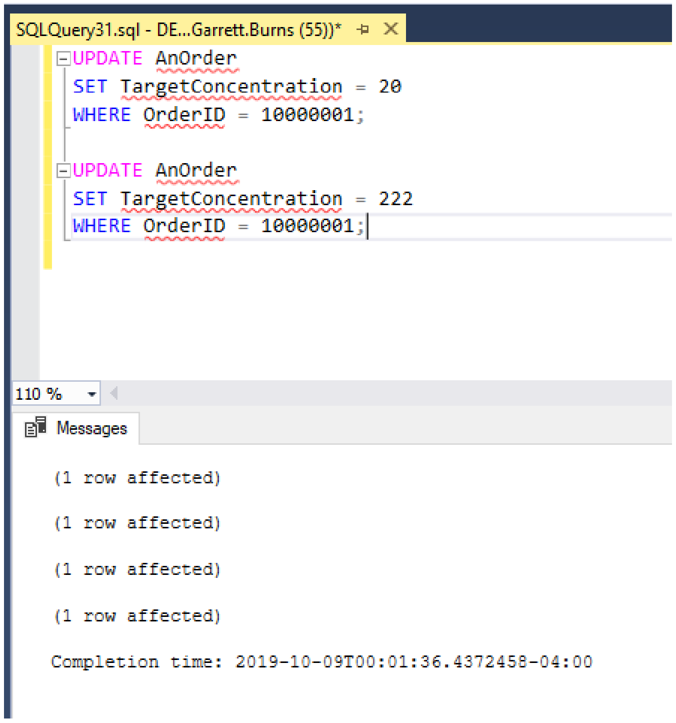
```sql
CREATE TRIGGER TCChangeTrigger
ON AnOrder
AFTER UPDATE
AS
BEGIN
    DECLARE @OldTC DECIMAL(6,2) = (SELECT TargetConcentration FROM DELETED);
    DECLARE @NewTC DECIMAL(6,2) = (SELECT TargetConcentration FROM INSERTED);

    IF (@OldTC <> @NewTC)
        INSERT INTO TCChange(TCChangeID, OldTC, NEWTC, OrderID, ChangeDate)
        VALUES(ISNULL((SELECT MAX(TCChangeID)+1 FROM TCChange), 1),
            @OldTC,
            @NewTC,
            (SELECT OrderID FROM INSERTED),
            GETDATE());
END;
```

110 %

Messages

```
Commands completed successfully.

Completion time: 2019-10-08T23:54:56.7380261-04:00
```

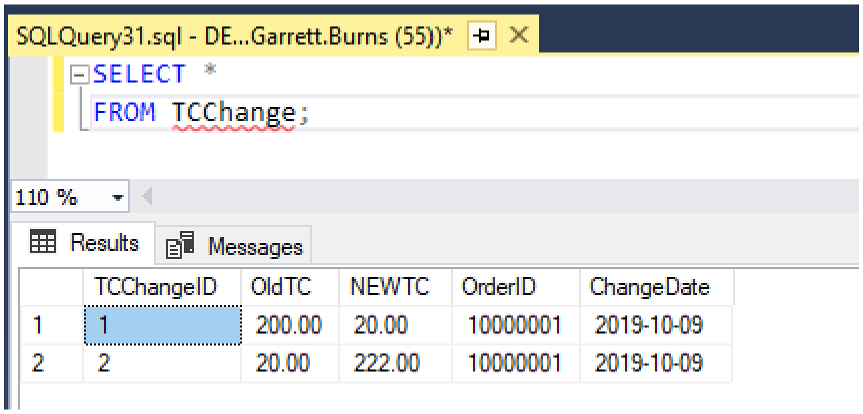
Now we'll see the trigger update the history table in action. Below is a screenshot of an order's target concentration being updated twice in the database.



```
UPDATE AnOrder
SET TargetConcentration = 20
WHERE OrderID = 10000001;

UPDATE AnOrder
SET TargetConcentration = 222
WHERE OrderID = 10000001;
```

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2019-10-09T00:01:36.4372458-04:00
```

Reporting on these updates, the history table now shows two rows: one for each adjustment to the target concentration.

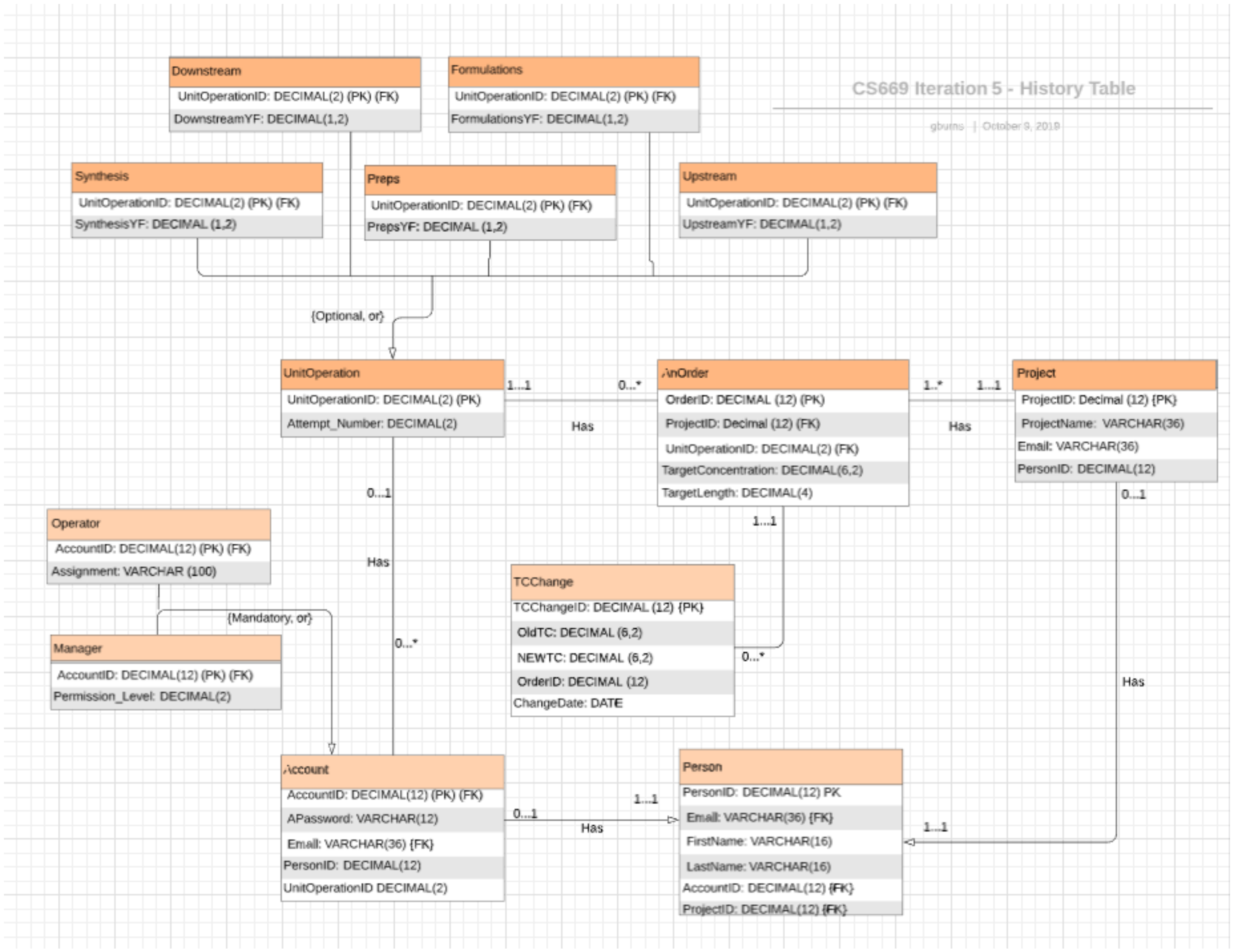


```
SELECT *
FROM TCChange;
```

	TCChangeID	OldTC	NEWTC	OrderID	ChangeDate
1	1	200.00	20.00	10000001	2019-10-09
2	2	20.00	222.00	10000001	2019-10-09

Notice that the history table has 4 columns. The TCChangeID is unique to each new update that is introduced to the history table. The OldTC column shows the original target concentration, while the NEWTC column shows what it has been updated to. The OrderID column shows which order in the database this update has been applied to, and the ChangeDate tells us when this update occurred.

Because including a history table has introduced another entity to the DBMS, the ERD has been updated to include the TCChange entity as seen below.

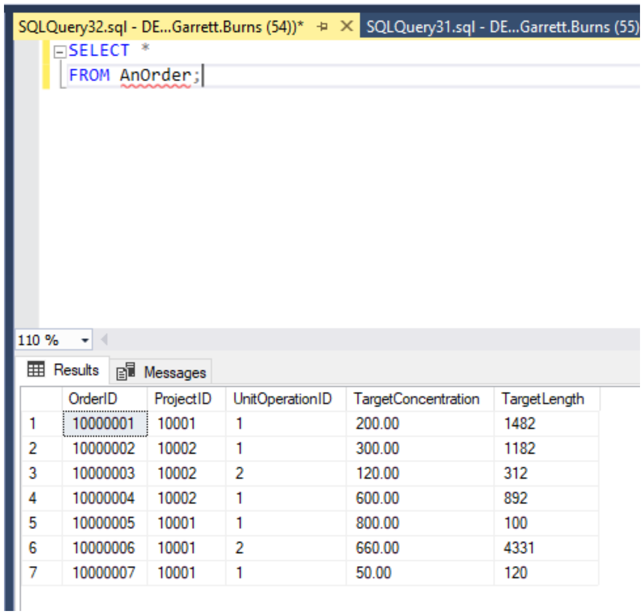


CS669 Iteration 5 - History Table
gburns | October 9, 2019
Downstream
UnitOperationID: DECIMAL(2) (PK) (FK)
DownstreamYF: DECIMAL(1,2)
Formulations
UnitOperationID: DECIMAL(2) (PK) (FK)
FormulationsYF: DECIMAL(1,2)
Synthesis
UnitOperationID: DECIMAL(2) (PK) (FK)
SynthesisYF: DECIMAL (1,2)
Preps
UnitOperationID: DECIMAL(2) (PK) (FK)
PrepsYF: DECIMAL (1,2)
Upstream
UnitOperationID: DECIMAL(2) (PK) (FK)
UpstreamYF: DECIMAL(1,2)
{Optional, or}
UnitOperation
UnitOperationID: DECIMAL(2) (PK)
Attempt_Number: DECIMAL(2)
1...1
0...*
Has
AnOrder
OrderID: DECIMAL (12) (PK)
ProjectID: Decimal (12) (FK)
UnitOperationID: DECIMAL(2) (FK)
TargetConcentration: DECIMAL(6,2)
TargetLength: DECIMAL(4)
1..*
1...1
Has
Project
ProjectID: Decimal (12) {PK}
ProjectName: VARCHAR(36)
Email: VARCHAR(36)
PersonID: DECIMAL(12)
0...1
0...1
Has
Operator
AccountID: DECIMAL(12) (PK) (FK)
Assignment: VARCHAR (100)
1...1
TCChange
TCChangeID: DECIMAL (12) {PK}
OldTC: DECIMAL (6,2)
NEWTC: DECIMAL (6,2)
OrderID: DECIMAL (12)
ChangeDate: DATE
0...*
{Mandatory, or}
Manager
AccountID: DECIMAL(12) (PK) (FK)
Permission_Level: DECIMAL(2)
0...*
Has
Account
AccountID: DECIMAL(12) (PK) (FK)
APassword: VARCHAR(12)
Email: VARCHAR(36) {FK}
PersonID: DECIMAL(12)
UnitOperationID DECIMAL(2)
0...1
Has
1...1
Person
PersonID: DECIMAL(12) PK
Email: VARCHAR(36) {FK}
FirstName: VARCHAR(16)
LastName: VARCHAR(16)
AccountID: DECIMAL(12) {FK}
ProjectID: DECIMAL(12) {FK}
1...1

Organization Driven Queries

Query 1: "What is the largest target-length order that has been submitted, and what is this length?"
Fulfills: Restriction in where clause, Aggregate function, and subquery

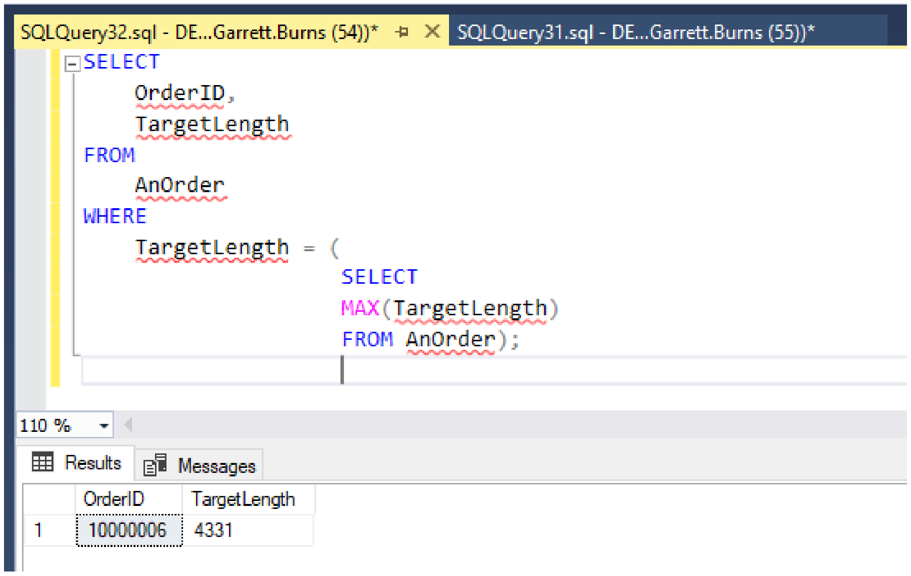
Below is a screenshot sharing a full list of all the orders currently in the system, provided by the AnOrder table. When many orders are in the database, it may be difficult to find what the order with the largest length is.


SQLQuery32.sql - DE...Garrett.Burns (54))*
SQLQuery31.sql - DE...Garrett.Burns (55)
110 %
Results
Messages

```
SELECT *
FROM AnOrder;
```

	OrderID	ProjectID	UnitOperationID	TargetConcentration	TargetLength
1	10000001	10001	1	200.00	1482
2	10000002	10002	1	300.00	1182
3	10000003	10002	2	120.00	312
4	10000004	10002	1	600.00	892
5	10000005	10001	1	800.00	100
6	10000006	10001	2	660.00	4331
7	10000007	10001	1	50.00	120

A query for this situation, and its results, are shown below. The query parsed through the rows of AnOrder and only provided us with a row representing the order number for the largest target-length, along with this value of length.

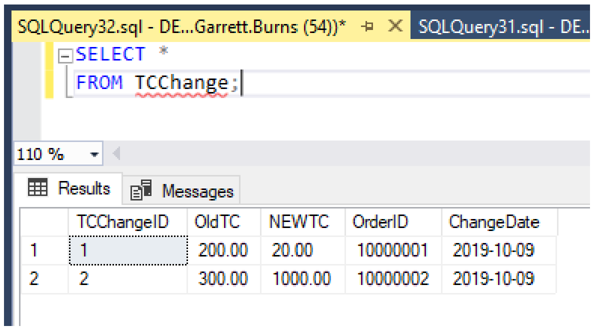

SQLQuery32.sql - DE...Garrett.Burns (54))*
SQLQuery31.sql - DE...Garrett.Burns (55))*
110 %
Results
Messages

```
SELECT
    OrderID,
    TargetLength
FROM
    AnOrder
WHERE
    TargetLength = (
                    SELECT
                    MAX(TargetLength)
                    FROM AnOrder);
```

	OrderID	TargetLength
1	10000006	4331

Query 2: A order's target concentration has just been updated from 300mg to 1000mg, but we cannot fulfill their updated request. What is the email address of the customer who placed this order, and what is their name? We need to let them know we cannot fulfill this request.
Fulfills: Where clause, 3 joins. (Features History Table)

The screenshot below shows the update to the order's concentration in row 2 of the history table.



SQLQuery32.sql - DE...Garrett.Burns (54))* SQLQuery31.sql - DE...

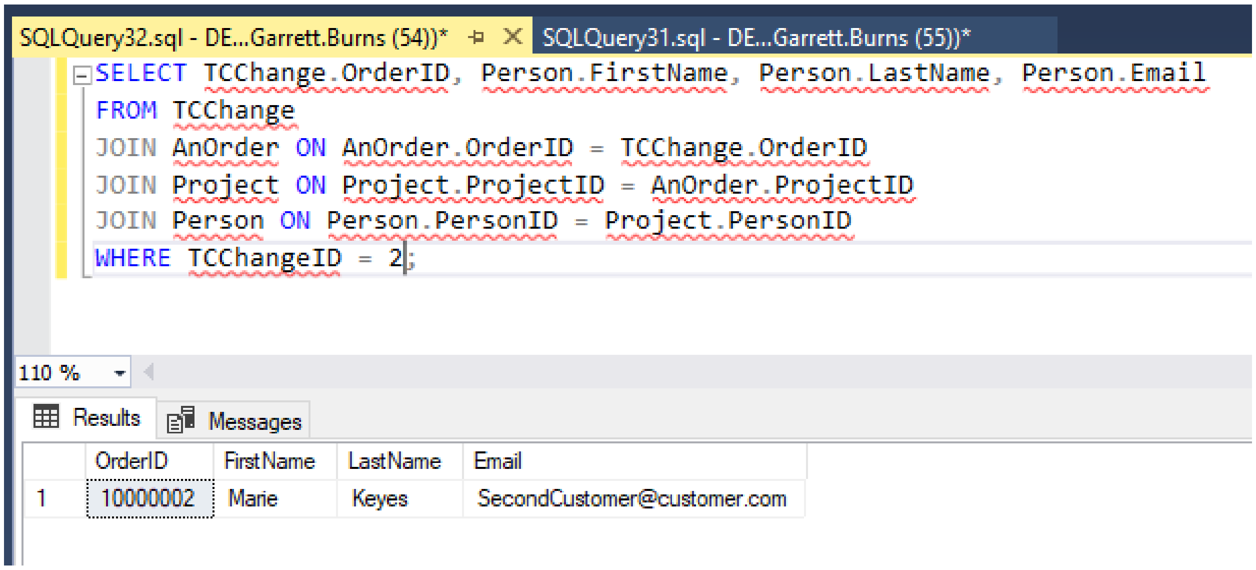
```sql
SELECT *
FROM TCChange;
```

110 %

Results Messages

	TCChangeID	OldTC	NEWTC	OrderID	ChangeDate
1	1	200.00	20.00	10000001	2019-10-09
2	2	300.00	1000.00	10000002	2019-10-09

The Query below uses joins to look up the requested information regarding the recently changed order, using a WHERE clause to identify it by the TCChange ID given in the history table.



SQLQuery32.sql - DE...Garrett.Burns (54))* SQLQuery31.sql - DE...Garrett.Burns (55))*

```sql
SELECT TCChange.OrderID, Person.FirstName, Person.LastName, Person.Email
FROM TCChange
JOIN AnOrder ON AnOrder.OrderID = TCChange.OrderID
JOIN Project ON Project.ProjectID = AnOrder.ProjectID
JOIN Person ON Person.PersonID = Project.PersonID
WHERE TCChangeID = 2;
```

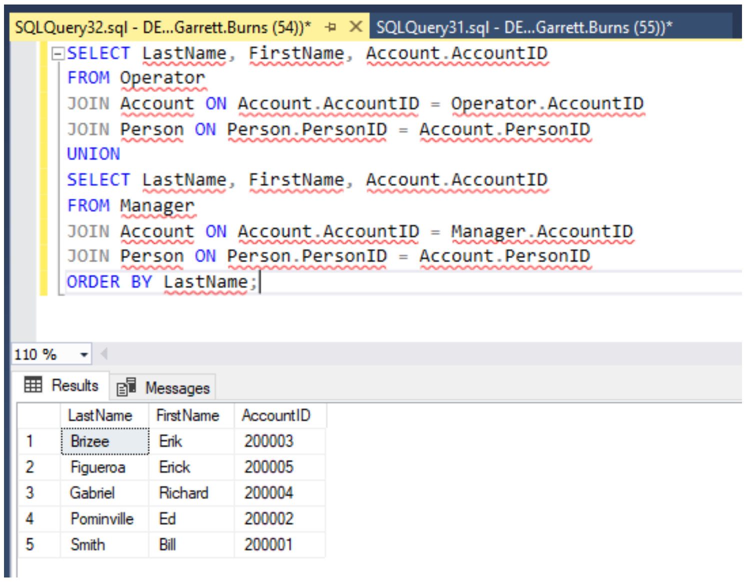
110 %

Results Messages

	OrderID	FirstName	LastName	Email
1	10000002	Marie	Keyes	SecondCustomer@customer.com

Query 3: We need a list of all operators and managers in order of their last name, along with their account number.
Fulfills: Order by, join 4 tables, Union of 2 queries

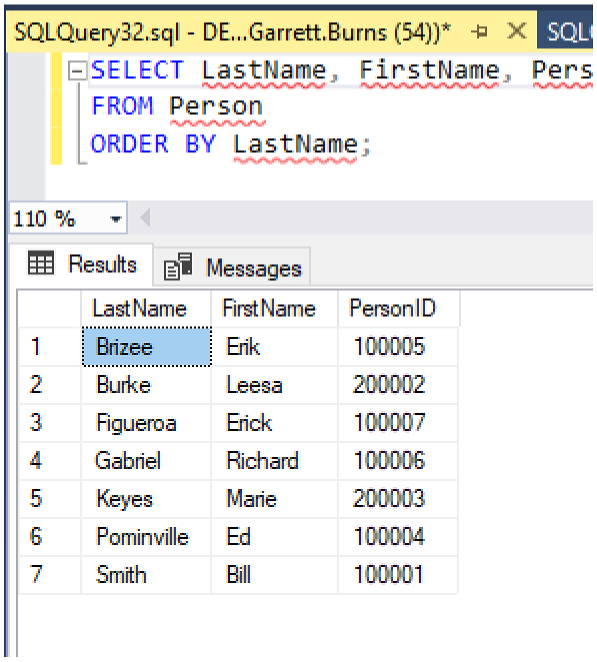
The query below performs this need by performing a union, and ordering the results by the LastName attribute.


SQLQuery32.sql - DE...Garrett.Burns (54))*
SQLQuery31.sql - DE...Garrett.Burns (55))*
110 %
Results
Messages

```
SELECT LastName, FirstName, Account.AccountID
FROM Operator
JOIN Account ON Account.AccountID = Operator.AccountID
JOIN Person ON Person.PersonID = Account.PersonID
UNION
SELECT LastName, FirstName, Account.AccountID
FROM Manager
JOIN Account ON Account.AccountID = Manager.AccountID
JOIN Person ON Person.PersonID = Account.PersonID
ORDER BY LastName;
```

	LastName	FirstName	AccountID
1	Brizee	Erik	200003
2	Figueroa	Erick	200005
3	Gabriel	Richard	200004
4	Pominville	Ed	200002
5	Smith	Bill	200001

Note that while it may seem intuitive to perform this simply by performing a query for all persons, this cannot be done—as persons are also entities that represent customers and are tracked by a PersonID rather than an AccountID. This similar but incorrect query is demonstrated below to highlight the distinction.



```
SELECT LastName, FirstName, Pers
FROM Person
ORDER BY LastName;
```

	LastName	FirstName	PersonID
1	Brizee	Erik	100005
2	Burke	Leesa	200002
3	Figueroa	Erick	100007
4	Gabriel	Richard	100006
5	Keyes	Marie	200003
6	Pominville	Ed	100004
7	Smith	Bill	100001