# Comp 2001 Set Exercises Report

## Entity Relationship Diagram

A screenshot of a computer

Description automatically generated

Assumptions made:

There is a one-to-one relationship between user and archived user, this will only happen for a short period of time when the user is getting archived and the data is stored in both tables.

## Normalisation

A screenshot of a computer

Description automatically generated

## SQL Commands for creating tables

CREATE SCHEMA CW1

CREATE TABLE CW1.[User] (

user\_id INTEGER PRIMARY KEY IDENTITY(1,1),

username VARCHAR(50),

email VARCHAR(80),

password VARCHAR(40),

account\_type VARCHAR(15)

);

A screenshot of a computer

Description automatically generated

CREATE TABLE CW1.[Archive\_User] (

user\_id INTEGER PRIMARY KEY,

username VARCHAR(50),

email VARCHAR(80),

password VARCHAR(40),

account\_type VARCHAR(15)

);

A screenshot of a computer

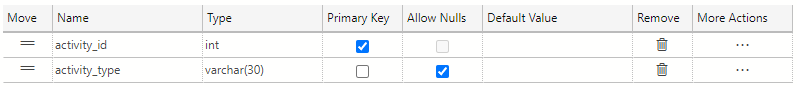
Description automatically generated

CREATE TABLE CW1.[Activity] (

activity\_id INTEGER PRIMARY KEY IDENTITY(1,1),

activity\_type VARCHAR(30)

);



CREATE TABLE CW1.[Favourite\_Activities] (

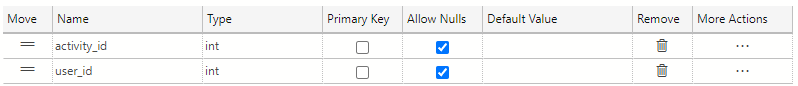
activity\_id INTEGER,

user\_id INTEGER,

FOREIGN KEY (activity\_id) REFERENCES CW1.[Activity](activity\_id),

FOREIGN KEY (user\_id) REFERENCES CW1.[User](user\_id)

);



CREATE TABLE CW1.[Archive\_Favourite\_Activities] (

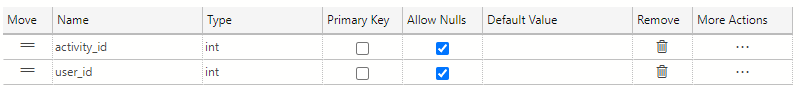
activity\_id INTEGER,

user\_id INTEGER,

FOREIGN KEY (activity\_id) REFERENCES CW1.[Activity](activity\_id),

FOREIGN KEY (user\_id) REFERENCES CW1.[Archive\_User](user\_id)

);



CREATE TABLE CW1.[Follow\_List] (

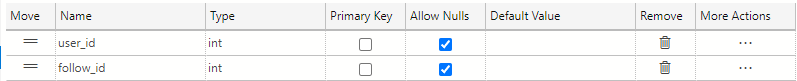
user\_id INTEGER,

FOREIGN KEY (user\_id) REFERENCES CW1.[User](user\_id),

follow\_id INTEGER,

FOREIGN KEY (follow\_id) REFERENCES CW1.[User](user\_id),

);



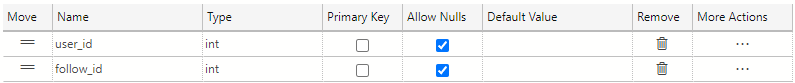
CREATE TABLE CW1.[Archive\_Follow\_List] (

user\_id INTEGER,

follow\_id INTEGER,

);

The reason that this doesn’t not have any foreign keys to the Archive\_User is that if there were, referential integrity would be broken and therefore I would not be able to implement this.



Result:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Creating Views

CREATE VIEW CW1.[Main\_View] AS

SELECT u.username AS "Username", u.email AS "Email", u.account\_type AS "Account Type",

(SELECT COUNT(\*) FROM CW1.[Follow\_List] f WHERE u.user\_id = f.user\_id) Following,

(SELECT COUNT(\*) FROM CW1.[Follow\_List] f WHERE u.user\_id = f.follow\_id) Followers,

STRING\_AGG(a.activity\_type,', ') AS "Favourite Activities"

FROM CW1.[User] u

LEFT JOIN CW1.[Favourite\_Activities] f ON u.user\_id = f.user\_id

LEFT JOIN CW1.[Activity] a ON f.activity\_id = a.activity\_id

GROUP BY u.user\_id, u.account\_type, u.username, u.email

SELECT \* FROM CW1.[Main\_View]

CREATE VIEW CW1.[Archive\_View] AS

SELECT u.username AS "Archive Username", u.email AS "Archive Email", u.account\_type AS "Archive Account Type",

(SELECT COUNT(\*) FROM CW1.[Archive\_Follow\_List] f WHERE u.user\_id = f.user\_id) Following,

(SELECT COUNT(\*) FROM CW1.[Archive\_Follow\_List] f WHERE u.user\_id = f.follow\_id) Followers,

STRING\_AGG(a.activity\_type,', ') AS "Favourite Activities"

FROM CW1.[Archive\_User] u

LEFT JOIN CW1.[Archive\_Favourite\_Activities] f ON u.user\_id = f.user\_id

LEFT JOIN CW1.[Activity] a ON f.activity\_id = a.activity\_id

GROUP BY u.user\_id, u.account\_type, u.username, u.email

SELECT \* FROM CW1.[Archive\_View]

Result: A screenshot of a computer

Description automatically generated

With Data inputted:

A screenshot of a computer

Description automatically generated

### Commands to insert data:

INSERT INTO CW1.[User] ([username],[email],[password],[account\_type])

VALUES

('Grace Hopper', 'grace@plymouth.ac.uk', 'ISAD123!', 'user'),

('Tim Berners-Lee', 'tim@plymouth.ac.uk', 'COMP2001!', 'user'),

('Veraint', 'Veraint@plymouth.ac.uk', 'COMP2001!', 'admin'),

('Patryk', 'pat@plymouth.ac.uk', 'patpassword!', 'admin'),

('Ada Lovelace', '@plymouth.ac.uk', 'insecurePassword', 'user'),

('George', 'George@students.plymouth.ac.uk', 'Georges password', 'admin');

EXEC CW1.[Follow\_User] 1,2

EXEC CW1.[Follow\_User] 4,5

EXEC CW1.[Follow\_User] 5,4

EXEC CW1.[Follow\_User] 2,1

EXEC CW1.[Follow\_User] 3,1

EXEC CW1.[Follow\_User] 1,6

EXEC CW1.[Follow\_User] 2,6

EXEC CW1.[Follow\_User] 3,6

EXEC CW1.[Follow\_User] 4,6

EXEC CW1.[Follow\_User] 6,3

EXEC CW1.[Favourite\_Activity] 1,1

EXEC CW1.[Favourite\_Activity] 1,2

EXEC CW1.[Favourite\_Activity] 2,2

EXEC CW1.[Favourite\_Activity] 2,3

EXEC CW1.[Favourite\_Activity] 3,4

EXEC CW1.[Favourite\_Activity] 3,1

EXEC CW1.[Favourite\_Activity] 4,1

EXEC CW1.[Favourite\_Activity] 4,5

EXEC CW1.[Favourite\_Activity] 1,6

EXEC CW1.[Favourite\_Activity] 2,6

EXEC CW1.[Favourite\_Activity] 5,6

## Stored procedures

### Add User

This procedure is used to add a user to the Users table

CREATE PROCEDURE CW1.[Add\_User]

@username VARCHAR(50),

@email VARCHAR(80),

@password VARCHAR(40),

@account\_type VARCHAR(15)

AS

BEGIN

INSERT INTO CW1.[User] (username,email,password,account\_type)

VALUES

(@username, @email, @password, @account\_type);

END

Code used to execute command:

EXEC CW1.[Add\_User] 'Test\_User', 'Test@students.plymouth.ac.uk', 'testpassword', 'user'

Result:

A screenshot of a computer

Description automatically generated

As you can see the user has been added to the table with the correct data.

### Edit Username

This procedure is used to edit a username of a user in the Users table

CREATE PROCEDURE CW1.[Edit\_Username]

@user\_id INTEGER,

@new\_username VARCHAR(50)

AS

BEGIN

UPDATE CW1.[User]

SET username = @new\_username

WHERE user\_id = @user\_id;

END

Code used to execute command:

EXEC CW1.[Edit\_Username] 6, "George"

This should change the user number 6’s (Test User) username to be “George”

A screenshot of a computer

Description automatically generated

As you can see this has worked correctly.

### Edit Email

This procedure is used to edit the email of a user in the Users table

CREATE PROCEDURE CW1.[Edit\_Email]

@user\_id INTEGER,

@new\_email VARCHAR(80)

AS

BEGIN

UPDATE CW1.[User]

SET email = @new\_email

WHERE user\_id = @user\_id;

END

Code used to execute command:

EXEC CW1.[Edit\_Email] 6, "George@students.plymouth.ac.uk"

This should change the user number 6’s (George) email to be “George@students.plymouth.ac.uk”

A screenshot of a computer

Description automatically generated

As you can see George’s email address is now set to what we wanted

### Edit Password

This procedure is used to edit a user’s password in the Users table

CREATE PROCEDURE CW1.[Edit\_Password]

@user\_id INTEGER,

@new\_password VARCHAR(40)

AS

BEGIN

UPDATE CW1.[User]

SET password = @new\_password

WHERE user\_id = @user\_id;

END

Code used to execute command:

EXEC CW1.[Edit\_Password] 6, "Georges password"

This should change the user number 6’s (George) password to be “Georges password”.

A screenshot of a computer

Description automatically generatedFor this I had to use a different view instead of the views I created, here I am doing SELECT \* FROM CW1.[User]. This is because the views that I created do not show the users password. However the data is still correct despite this and George’s password has been changed to what we wanted.

### Edit Account Type

This procedure will edit the account type of a user in the Users table.

CREATE PROCEDURE CW1.[Edit\_Account\_Type]

@user\_id INTEGER,

@new\_account\_type VARCHAR(15)

AS

BEGIN

UPDATE CW1.[User]

SET account\_type = @new\_account\_type

WHERE user\_id = @user\_id;

END

Code used to execute command:

EXEC CW1.[Edit\_Account\_Type] 6, "admin"

This should change the user number 6’s (George) account type to be admin.

A screenshot of a computer

Description automatically generated

As you can see George’s account type is now admin.

### Archive User

This procedure will archive a user, to do this we need to move all data that is relevant to the user into the archive tables. This requires the moving to be done in specific ordered steps as to keep referential integrity. To do this we must copy the user data, then favourite activities, then the follow lists, then finally we delete from the normal tables in the order of favourite activities, follow lists and then finally the user.

CREATE PROCEDURE CW1.[Archive\_User\_Procedure]

@user\_id INTEGER

AS

BEGIN

INSERT INTO CW1.[Archive\_User] (user\_id, username, email, password, account\_type)

SELECT \*

FROM CW1.[User]

WHERE user\_id = @user\_id;

INSERT INTO CW1.[Archive\_Favourite\_Activities] (activity\_id, user\_id)

SELECT \*

FROM CW1.[Favourite\_Activities]

WHERE user\_id = @user\_id;

INSERT INTO CW1.[Archive\_Follow\_List] (user\_id, follow\_id)

SELECT \*

FROM CW1.[Follow\_List]

WHERE user\_id = @user\_id;

INSERT INTO CW1.[Archive\_Follow\_List] (user\_id, follow\_id)

SELECT \*

FROM CW1.[Follow\_List]

WHERE follow\_id = @user\_id;

DELETE FROM CW1.[Favourite\_Activities]

WHERE user\_id = @user\_id;

DELETE FROM CW1.[Follow\_List]

WHERE user\_id = @user\_id;

DELETE FROM CW1.[Follow\_List]

WHERE follow\_id = @user\_id;

DELETE FROM CW1.[User]

WHERE user\_id = @user\_id;

END

This code creates the procedure, using insert and delete statements, also taking an input from the user to specify what user is to be archived.

A screenshot of a computer

Description automatically generated

I have added some followers and activities to show how those also get archived.

Archiving the user:

EXEC CW1.[Archive\_User\_Procedure] 6

A screenshot of a computer

Description automatically generated

As you can see the user has now been archived, preserving the followers that they have and still showing the favourite activities that they have. This means that the archive procedure is now fully working.

Here I could have created a procedure to un-archive a user, however I felt that since there was no mention of this in the specification that this was not a necessary procedure to implement.

### Follow User

CREATE PROCEDURE CW1.[Follow\_User]

@user\_id INTEGER,

@follow\_id INTEGER

AS

BEGIN

INSERT INTO CW1.[Follow\_List] (user\_id, follow\_id)

VALUES

(@user\_id, @follow\_id);

END

Before following:

A screenshot of a computer

Description automatically generated

Code to follow the user:

EXEC CW1.[Follow\_User] 5 , 3

EXEC CW1.[Follow\_User] 4 , 3

EXEC CW1.[Follow\_User] 2 , 3

EXEC CW1.[Follow\_User] 1 , 3

This should mean that the user “Veraint” now has 4 followers, being all the other users in the table.

A screenshot of a computer

Description automatically generated

As you can see Veraint now has 4 followers and all the other users in the table have increased their following count by 1.

### Un-Follow User

CREATE PROCEDURE CW1.[Unfollow\_User]

@user\_id INTEGER,

@follow\_id INTEGER

AS

BEGIN

DELETE FROM CW1.[Follow\_List]

WHERE @user\_id = user\_id AND @follow\_id = follow\_id

END

A screenshot of a computer

Description automatically generated

Currently user veraint is following user grace, we shall make veraint unfollow her.



This shows that user veriant (3) is following grace(1) in the followers table

Code to execute:

EXEC CW1.[Unfollow\_User] 3 , 1

A screenshot of a computer

Description automatically generated

Now veraints following count has gone down by 1, and graces followers count has also gone down by 1

### Add Activity

CREATE PROCEDURE CW1.[Add\_Activity]

@activity\_type VARCHAR(30)

AS

BEGIN

INSERT INTO CW1.[Activity] (activity\_type)

VALUES

(@activity\_type);

END

EXEC CW1.[Add\_Activity] "Running"

EXEC CW1.[Add\_Activity] "Walking"

EXEC CW1.[Add\_Activity] "Cycling"

EXEC CW1.[Add\_Activity] "Swimming"

EXEC CW1.[Add\_Activity] "Hiking"

A screenshot of a computer

Description automatically generated

Here all of the activities have been added to the Activity table

### Favourite Activity

CREATE PROCEDURE CW1.[Favourite\_Activity]

@user\_id INTEGER,

@activity\_id INTEGER

AS

BEGIN

INSERT INTO CW1.[Favourite\_Activities] (activity\_id, user\_id)

VALUES

(@user\_id, @activity\_id);

END

A screenshot of a computer

Description automatically generated

Currently Ada Lovelace only has swimming as a favourite activity, we shall add running to that list.

EXEC CW1.[Favourite\_Activity] 1, 5

A screenshot of a computer

Description automatically generated

### Un-Favourite Activity

CREATE PROCEDURE CW1.[Un\_Favourite\_Activity]

@user\_id INTEGER,

@activity\_id INTEGER

AS

BEGIN

DELETE FROM CW1.[Favourite\_Activities]

WHERE @user\_id = user\_id AND @activity\_id = activity\_id

END

A screenshot of a computer

Description automatically generated

Ada actually doesn’t like running, so let’s remove it:

EXEC CW1.[Un\_Favourite\_Activity] 5, 1

A screenshot of a computer

Description automatically generated

Running has now been removed from Ada’s list of favourite activities.

## Triggers

CREATE TRIGGER CW1.[Email\_Trigger]

ON CW1.[User]

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS (SELECT 1 FROM INSERTED WHERE CHARINDEX('@', email) = 0)

BEGIN

RAISERROR('Email is invalid', 16, 1);

ROLLBACK;

END

END;

To check that the trigger works correctly, we will add a user with an incorrect email address to test.

This is the code to add a user with an incorrect email address:

EXEC CW1.[Add\_User] 'Test\_User', 'Incorrect Email Address', 'testpassword', 'user'

A screenshot of a computer

Description automatically generated

As you can see the trigger does not let an email that is incorrect be added to the table, returning an error message saying email is invalid.

Here I will use a correct email format to show that it lets valid data in:

EXEC CW1.[Add\_User] 'Test\_User', 'Correct@email.com', 'testpassword', 'user'

A screenshot of a computer

Description automatically generated

Here you can see that the user has been added as they have a correct email address. There is no data for followers and activities because we need to add that later.

## Appendix

CREATE SCHEMA CW1

CREATE TABLE CW1.[User] (

user\_id INTEGER PRIMARY KEY IDENTITY(1,1),

username VARCHAR(50),

email VARCHAR(80),

password VARCHAR(40),

account\_type VARCHAR(15)

);

CREATE TABLE CW1.[Archive\_User] (

user\_id INTEGER PRIMARY KEY,

username VARCHAR(50),

email VARCHAR(80),

password VARCHAR(40),

account\_type VARCHAR(15)

);

CREATE TABLE CW1.[Activity] (

activity\_id INTEGER PRIMARY KEY IDENTITY(1,1),

activity\_type VARCHAR(30)

);

CREATE TABLE CW1.[Favourite\_Activities] (

activity\_id INTEGER,

user\_id INTEGER,

FOREIGN KEY (activity\_id) REFERENCES CW1.[Activity](activity\_id),

FOREIGN KEY (user\_id) REFERENCES CW1.[User](user\_id)

);

CREATE TABLE CW1.[Archive\_Favourite\_Activities] (

activity\_id INTEGER,

user\_id INTEGER,

FOREIGN KEY (activity\_id) REFERENCES CW1.[Activity](activity\_id),

FOREIGN KEY (user\_id) REFERENCES CW1.[Archive\_User](user\_id)

);

CREATE TABLE CW1.[Follow\_List] (

user\_id INTEGER,

FOREIGN KEY (user\_id) REFERENCES CW1.[User](user\_id),

follow\_id INTEGER,

FOREIGN KEY (follow\_id) REFERENCES CW1.[User](user\_id),

);

CREATE TABLE CW1.[Archive\_Follow\_List] (

user\_id INTEGER,

follow\_id INTEGER,

);

SQL View

CREATE VIEW CW1.[Main\_View] AS

SELECT u.username AS "Username", u.email AS "Email", u.account\_type AS "Account Type",

(SELECT COUNT(\*) FROM CW1.[Follow\_List] f WHERE u.user\_id = f.user\_id) Following,

(SELECT COUNT(\*) FROM CW1.[Follow\_List] f WHERE u.user\_id = f.follow\_id) Followers,

STRING\_AGG(a.activity\_type,', ') AS "Favourite Activities"

FROM CW1.[User] u

LEFT JOIN CW1.[Favourite\_Activities] f ON u.user\_id = f.user\_id

LEFT JOIN CW1.[Activity] a ON f.activity\_id = a.activity\_id

GROUP BY u.user\_id, u.account\_type, u.username, u.email

SELECT \* FROM CW1.[Main\_View]

CREATE VIEW CW1.[Archive\_View] AS

SELECT u.username AS "Archive Username", u.email AS "Archive Email", u.account\_type AS "Archive Account Type",

(SELECT COUNT(\*) FROM CW1.[Archive\_Follow\_List] f WHERE u.user\_id = f.user\_id) Following,

(SELECT COUNT(\*) FROM CW1.[Archive\_Follow\_List] f WHERE u.user\_id = f.follow\_id) Followers,

STRING\_AGG(a.activity\_type,', ') AS "Favourite Activities"

FROM CW1.[Archive\_User] u

LEFT JOIN CW1.[Archive\_Favourite\_Activities] f ON u.user\_id = f.user\_id

LEFT JOIN CW1.[Activity] a ON f.activity\_id = a.activity\_id

GROUP BY u.user\_id, u.account\_type, u.username, u.email

SELECT \* FROM CW1.[Archive\_View]

CREATE PROCEDURE CW1.[Add\_User]

@username VARCHAR(50),

@email VARCHAR(80),

@password VARCHAR(40),

@account\_type VARCHAR(15)

AS

BEGIN

INSERT INTO CW1.[User] (username,email,password,account\_type)

VALUES

(@username, @email, @password, @account\_type);

END

EXEC CW1.[Add\_User] 'Veraint', 'Veraint@students.plymouth.ac.uk', 'testpassword', 'admin'

CREATE PROCEDURE CW1.[Edit\_Username]

@user\_id INTEGER,

@new\_username VARCHAR(50)

AS

BEGIN

UPDATE CW1.[User]

SET username = @new\_username

WHERE user\_id = @user\_id;

END

EXEC CW1.[Edit\_Username] 4, "Pat102"

CREATE PROCEDURE CW1.[Edit\_Email]

@user\_id INTEGER,

@new\_email VARCHAR(80)

AS

BEGIN

UPDATE CW1.[User]

SET email = @new\_email

WHERE user\_id = @user\_id;

END

EXEC CW1.[Edit\_Email] 4, "Pat102@gmail.com"

CREATE PROCEDURE CW1.[Edit\_Password]

@user\_id INTEGER,

@new\_password VARCHAR(40)

AS

BEGIN

UPDATE CW1.[User]

SET password = @new\_password

WHERE user\_id = @user\_id;

END

EXEC CW1.[Edit\_Password] 4, "Pats password"

CREATE PROCEDURE CW1.[Edit\_Account\_Type]

@user\_id INTEGER,

@new\_account\_type VARCHAR(15)

AS

BEGIN

UPDATE CW1.[User]

SET account\_type = @new\_account\_type

WHERE user\_id = @user\_id;

END

EXEC CW1.[Edit\_Account\_Type] 4, "user"

CREATE PROCEDURE CW1.[Archive\_User\_Procedure]

@user\_id INTEGER

AS

BEGIN

INSERT INTO CW1.[Archive\_User] (user\_id, username, email, password, account\_type)

SELECT \*

FROM CW1.[User]

WHERE user\_id = @user\_id;

INSERT INTO CW1.[Archive\_Favourite\_Activities] (activity\_id, user\_id)

SELECT \*

FROM CW1.[Favourite\_Activities]

WHERE user\_id = @user\_id;

INSERT INTO CW1.[Archive\_Follow\_List] (user\_id, follow\_id)

SELECT \*

FROM CW1.[Follow\_List]

WHERE user\_id = @user\_id;

INSERT INTO CW1.[Archive\_Follow\_List] (user\_id, follow\_id)

SELECT \*

FROM CW1.[Follow\_List]

WHERE follow\_id = @user\_id;

DELETE FROM CW1.[Favourite\_Activities]

WHERE user\_id = @user\_id;

DELETE FROM CW1.[Follow\_List]

WHERE user\_id = @user\_id;

DELETE FROM CW1.[Follow\_List]

WHERE follow\_id = @user\_id;

DELETE FROM CW1.[User]

WHERE user\_id = @user\_id;

END

EXEC CW1.[Archive\_User\_Procedure] 2

CREATE PROCEDURE CW1.[Follow\_User]

@user\_id INTEGER,

@follow\_id INTEGER

AS

BEGIN

INSERT INTO CW1.[Follow\_List] (user\_id, follow\_id)

VALUES

(@user\_id, @follow\_id);

END

EXEC CW1.[Follow\_User] 1 , 2

CREATE PROCEDURE CW1.[Unfollow\_User]

@user\_id INTEGER,

@follow\_id INTEGER

AS

BEGIN

DELETE FROM CW1.[Follow\_List]

WHERE @user\_id = user\_id AND @follow\_id = follow\_id

END

EXEC CW1.[Unfollow\_User] 1 , 2

CREATE PROCEDURE CW1.[Add\_Activity]

@activity\_type VARCHAR(30)

AS

BEGIN

INSERT INTO CW1.[Activity] (activity\_type)

VALUES

(@activity\_type);

END

EXEC CW1.[Add\_Activity] "Running"

CREATE PROCEDURE CW1.[Favourite\_Activity]

@user\_id INTEGER,

@activity\_id INTEGER

AS

BEGIN

INSERT INTO CW1.[Favourite\_Activities] (activity\_id, user\_id)

VALUES

(@user\_id, @activity\_id);

END

EXEC CW1.[Favourite\_Activity] 3, 4

CREATE PROCEDURE CW1.[Un\_Favourite\_Activity]

@user\_id INTEGER,

@activity\_id INTEGER

AS

BEGIN

DELETE FROM CW1.[Favourite\_Activities]

WHERE @user\_id = user\_id AND @activity\_id = activity\_id

END

EXEC CW1.[Un\_Favourite\_Activity] 3, 4

CREATE TRIGGER CW1.[Email\_Trigger]

ON CW1.[User]

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS (SELECT 1 FROM INSERTED WHERE CHARINDEX('@', email) = 0)

BEGIN

RAISERROR('Email is invalid', 16, 1);

ROLLBACK;

END

END;