Exercise II - SQL

Part 4 – Toronto Bicycle Counts Data

CEE412/CET522

Transportation Data Management and Visualization

WINTER 2020

Getting Started

In this exercise we will look at some tables describing manual bicycle counts and supporting data from the city of Toronto, Canada. Counts were conducted in single day events at each location.

You will find the following tables in the CEE412_CET522_W20 database:

- E2_Cyclists (PersonID, LocationID, TimeInt, Gender, Helmet, Passenger, OnSidewalk)
- E2_Locations (LocationID, Road, Direction, Date)
- E2_Weather (Date, MaxTemp, MinTemp, MeanTemp, TotalPrecip)

The relationships in the database are as follows:

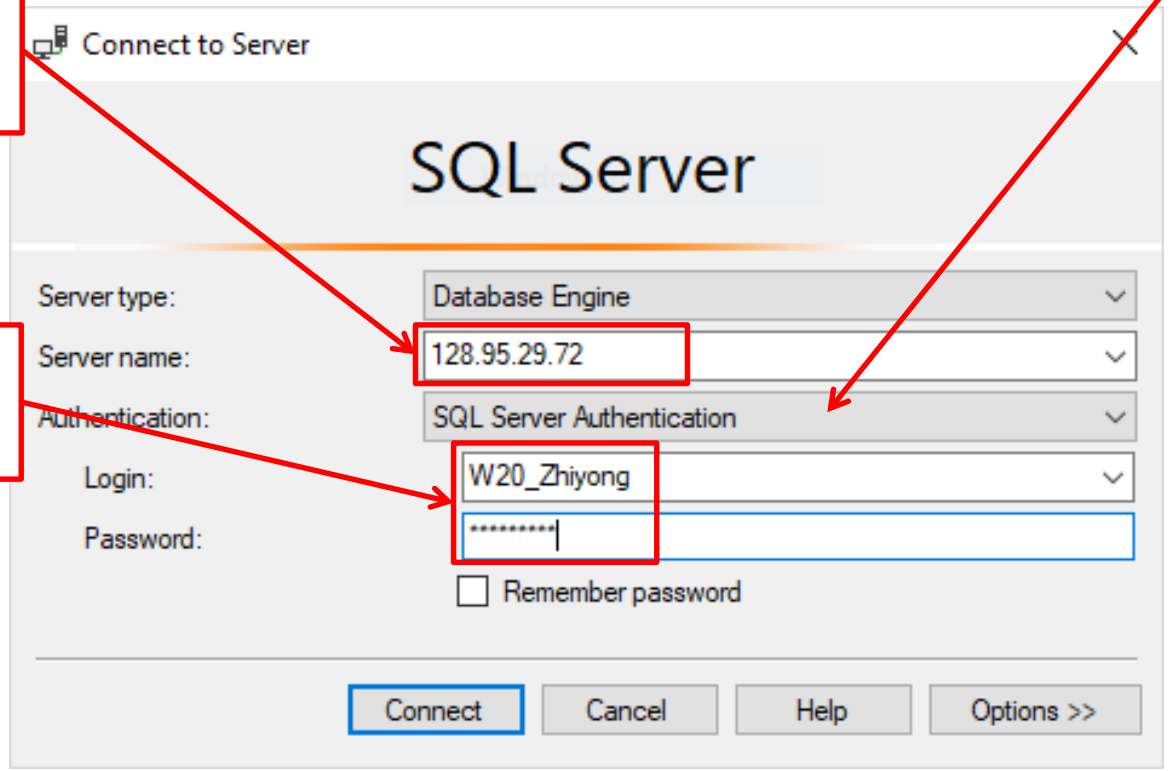
- Cyclists.LocationID = Locations.LocationID
- Locations.Date = Weather.Date

Log on Your Database Account

Use **SQL Server Authentication**

Input the class server IP address: 128.95.29.72

Input your account name and your password



Getting Started

Write some queries to look at the data in each of the three tables.

- For example, this is part of the cyclists table. You will see that time is given in 15 minute intervals, and that each row corresponds to an individual cyclist.

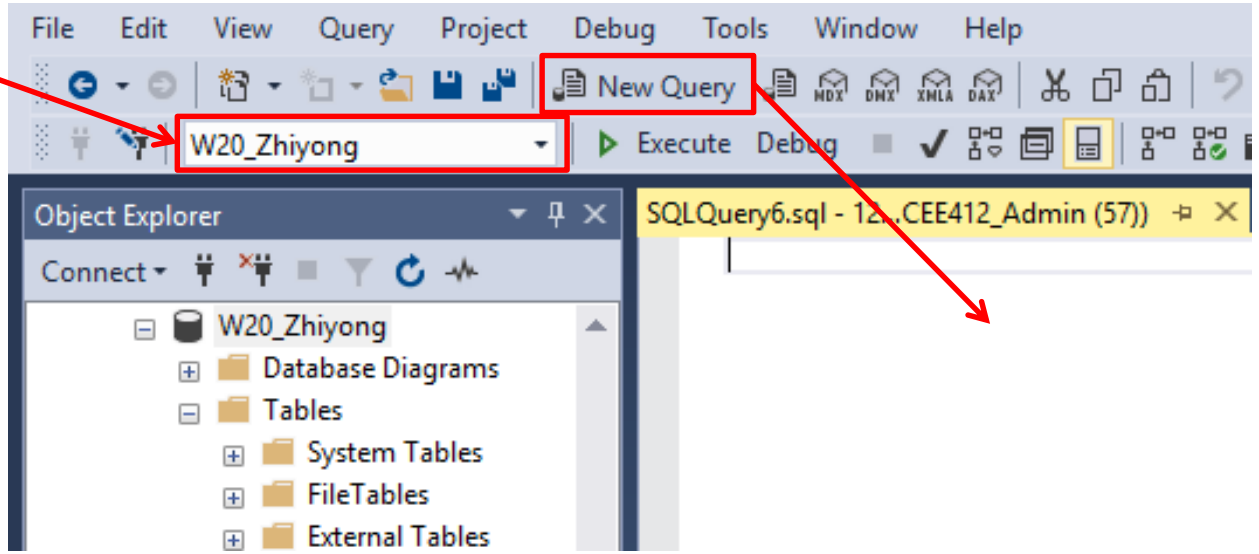
- Descriptive information is given for each cyclist, including whether or not the cyclist was wearing a helmet, whether the cyclist was on the sidewalk, etc.

PersonID	LocationID	TimeInt	Gender	Helmet	Passenger	OnSidewalk
1	1	7:30:00	Female	Yes	No	No
2	1	7:30:00	Male	No	No	No
3	1	7:45:00	Male	Yes	No	No
4	1	8:00:00	Male	Yes	No	No
5	1	8:00:00	Female	Yes	No	No
6	1	8:45:00	Female	No	No	No
7	1	8:45:00	Female	No	No	No
8	1	8:45:00	Male	Yes	No	No
9	1	8:45:00	Male	No	No	No
10	1	8:45:00	Female	No	No	No
11	1	9:00:00	Female	No	No	No
12	1	9:00:00	Male	No	No	No
13	1	9:00:00	Female	No	No	No
14	1	9:00:00	Female	Yes	No	No
15	1	9:15:00	Female	Yes	No	No
16	1	9:30:00	Male	No	No	No
…	…	…	…	…	…	…

Make Copies of the Tables

- Again at the first step, you want to copy the tables to your own database for further analysis.
- Create a new query by clicking the **New Query** button.

Make sure you are working in your own database!



Make Copies of the Tables

- Run the following SQL queries to copy tables into your own database.

```
SELECT *
INTO Cyclists
FROM CEE412_CET522_W20.dbo.E2_Cyclists

SELECT *
INTO Locations
FROM CEE412_CET522_W20.dbo.E2_Locations

SELECT *
INTO Weather
FROM CEE412_CET522_W20.dbo.E2_Weather
```

Query Practice

Question: Write a query to obtain cyclist counts for each combination of gender and day of week, considering only days for which the minimum temperature was below 12 degrees.

Tips:

- Use the following DATEPART function to get the day of week from the date column:

```
DATEPART(DW, Date)
```

- The first argument of the DATEPART function is the interval you want, and you may use YEAR, MONTH, DAY, etc. to extract the corresponding part from your date and time data.
- In this case "DW" is the code for "day of week", it will return an integer value for day of the week (1 = Sunday by default).
- The second argument is the column name or value you want to convert, in this case it is a column named "date"

Query Practice

Possible solution:

```
SELECT COUNT(*) AS Count, Gender,
       DATEPART(DW, w.Date) AS DayOfWeek
  FROM Cyclists AS c
       JOIN Locations AS l ON c.LocationID = l.LocationID
       JOIN Weather AS w ON l.Date = w.Date
 WHERE w.MinTemp < 12
 GROUP BY DATEPART(DW, w.Date), Gender
 ORDER BY DATEPART(DW, w.Date), Gender
```

Note: You can include multiple JOIN statements in the FROM clause. In this case, we are joining three tables all together using two inner joins. The condition for each join statement needs to be specified respectively.

Query Practice

Question: For each road segment, what is the fraction of cyclists using helmets for each gender?

Tips:

- To estimate the fraction of cyclists for each road segment and gender we need an overall count of cyclist as well as a helmet user count.
- This will likely require two queries, both are not very complicated.
- Let's try develop these two queries first.

Query Practice

- First, try writing a query to return the overall count for each gender in each road segment.

```
SELECT COUNT(*) AS TotCount, Road, Gender
  FROM Cyclists AS c JOIN Locations AS l
    ON c.LocationID = l.LocationID
 GROUP BY Road, Gender
```

- Then, write another query to calculate the helmet users for each gender in each road segment

```
SELECT COUNT(*) AS HelmetCount, Road, Gender
  FROM Cyclists AS c JOIN Locations AS l
    ON c.LocationID = l.LocationID
 WHERE Helmet = 'yes'
 GROUP BY Road, Gender
```

- If we can somehow combine the results from these two queries, we will be able to calculate the helmet user fractions.

Query Practice

- There are multiple ways that we can combine the results from the previous two queries. We may change one query into a subquery and use the other as the outer query, or we can make use of views or temporary tables.
- Try develop a subquery to calculate the helmet user fractions.
- Note that you may not want to use the same table name in your outer query and inner query, as it can give you error because of correlated references.
- Be careful about data types. In SQL, if you divide an integer by another integer, the result data type will also be integer (you will lose the fractional part). But if you involve a float in your calculation, the result will be a float.

Query Practice

Possible solution:

By multiplying 1.0, I can change a integer into a float. This can help me keep the fractional part in my result.

The average function here actually does nothing. But as I used group by, I must have some aggregation for TotalCount.

```
SELECT l.Road, c.Gender,
       COUNT(*)*1.0/AVG(TotCount) AS 'Helmet User Fraction'
  FROM Cyclists AS c
       JOIN Locations AS l ON c.LocationID = l.LocationID
       JOIN (SELECT COUNT(*) AS TotCount, Road, Gender
               FROM Cyclists AS cc JOIN Locations AS ll
                 ON cc.LocationID = ll.LocationID
              GROUP BY Road, Gender) AS t
    ON l.Road = t.Road AND c.Gender = t.Gender
 WHERE Helmet = 'yes'
 GROUP BY l.Road, c.Gender
 ORDER BY l.Road, c.Gender
```

Use a string to rename the column if you want a space in your column name.

This is the subquery.

Query Practice

- The previous solution may not look something readable. Can we make a better solution using views/temporary tables?
- The right side query is slightly longer than what we have before, but the logic is much more straightforward.
- I used views rather than temporary tables as I don't really need to save the intermediate results.

```
CREATE VIEW Total AS
SELECT COUNT(*) AS TotCount, Road, Gender
  FROM Cyclists AS c JOIN Locations AS l
    ON c.LocationID = l.LocationID
 GROUP BY Road, Gender

CREATE VIEW Helmet AS
SELECT COUNT(*) AS HelmetCount, Road, Gender
  FROM Cyclists AS c JOIN Locations AS l
    ON c.LocationID = l.LocationID
 WHERE Helmet = 'yes'
 GROUP BY Road, Gender

SELECT h.Road, h.Gender,
       HelmetCount*1.0/TotCount AS 'Helmet User Fraction'
  FROM Helmet AS h, Total AS t
 WHERE h.Road = t.Road AND h.Gender = t.Gender
 ORDER BY h.Road, h.Gender
```

Case Statements

With the same question, can we further improve the solution?

Here we introduce another SQL expression: CASE

- CASE statements in SQL are one way to return conditional values in a query.
- They can be slow compared to regular set-based operations, but can be very useful in some situations. The basic form of a CASE statement is as follows:

```
SELECT CASE <column name>
           WHEN <condition 1> THEN <value 1>
           WHEN <condition 1> THEN <value 1>
           ...
           ELSE <value x>
       END
```

- To be interpreted as: when the column is <condition 1>, return <value 1>, when the column is <condition 2>, then return <value 2>, …, else, return <value x>.

Case Statements

Let's look at an example:

- The following statement will return two columns from the Cyclists table. The first column is just the PersonID field, and the second one will take the value 1.0 if Helmet = 'yes', and 0.0 otherwise. Note that the column associated with the case statement will be named "Hel".

```
SELECT PersonID,
       CASE Helmet
            WHEN 'yes' THEN 1.0
            ELSE 0.0
       END AS Hel
  FROM Cyclists
```

- Let's look at how this can be used to compute the fraction of helmet users by road segment and gender.

Case Statements

Final solution:

```
SELECT Road, Gender,
       AVG(CASE Helmet
             WHEN 'yes' THEN 1.0
             ELSE 0.0
           END) AS 'Helmet User Fraction'
  FROM Cyclists AS c JOIN Locations AS l
    ON c.LocationID = l.LocationID
 GROUP BY Road, Gender
 ORDER BY Road, Gender
```

- The case statement will return 1 for helmet users and 0 for non-helmet users. Thus, taking the average of the case statement and grouping by location and gender, we get the fraction of helmet users in each group.
- Note that we have used values 1.0 and 0.0, rather than integer values 1 and 0. In this way I can keep the fractional part in my results.

Case Statements

Some final notes:

- Case statements can be useful, but often a good idea to design a database such that they will not often be needed.
- For example, if I were to design the cyclist count database for Toronto, I might indicate helmet usage and other true/false data using bit (0/1 values) data type.
- Of course, how you represent these fields depends on how you expect the database to be used.

Database Update

- Finally, let's make some simple changes to database tables.

- Add a new column to hold a cyclist count for each location in the Locations table. Consider this as a formatting step for some analysis, as it would be a bad idea for data management purposes (undesirable redundancy in the database).

- To do this, use an ALTER TABLE statement as follows to add a new empty column:

```
ALTER TABLE Locations
  ADD CyclistCount INT
```

Database Update

- Updating the new column will require data from both the Locations table and the Cyclists table. You can use a join and a subquery in the update statement as follows:

```
UPDATE Locations
   SET CyclistCount = Count
  FROM Locations AS l
       JOIN (SELECT LocationID, COUNT(*) AS Count
               FROM Cyclists
              GROUP BY LocationID) AS c
    ON l.LocationID = c.LocationID
```

Conditional Delete

- Now, delete the oldest data in the Cyclists and Locations tables (maybe it is out of date?).

- To do this, we will start by deleting the oldest data in the Locations table. Here is one possible solution (you don't have to run it, as it's not recoverable):

```
DELETE FROM Locations
 WHERE Date = (SELECT MIN(Date)
                 FROM Locations)
```

Conditional Delete

- Then, we need to delete the rows in Cyclists that no longer have a match in Locations as follows (again, you don't have to run this):

```
DELETE FROM Cyclists
 WHERE LocationID NOT IN (SELECT LocationID
                          FROM Locations)
```

- If it is true that we do not want to save data in Cyclists that cannot be matched to the Locations table, we should have a foreign key set up with ON DELETE CASCADE. This is shown in the next slide.

Define a Foreign Key

- In order to establish a foreign key, we first need a key in Locations to reference. Of course, this requires that the key be NOT NULL, so three steps are needed in total.
- Add a NOT NULL constraint to the LocationID field in Locations table:

```
ALTER TABLE Locations
ALTER COLUMN LocationID INT NOT NULL
```

- Set the field to be the primary key of Locations table:

```
ALTER TABLE Locations
  ADD CONSTRAINT pk_locations PRIMARY KEY (LocationID)
```

- Finally, add the foreign key constraint to the Cyclists table:

```
ALTER TABLE Cyclists
  ADD CONSTRAINT fk_cyclists
      FOREIGN KEY (LocationID) REFERENCES Locations(LocationID)
      ON DELETE CASCADE ON UPDATE CASCADE
```

Define a Foreign Key

- ON DELETE CASCADE means that a delete in Locations table will result in the corresponding rows in Cyclists being deleted.
- Note that you can set ON UPDATE and ON DELETE behavior differently, in this case both should probably be set to cascade.
- Alternatives to CASCADE include SET NULL, and the default NO ACTION (refuses the change)