Data Admin Concepts & Database Management

Lab 07 – Advanced Querying

Table of Contents

Data Admin Concepts & Database Management 1
Lab 07 – Advanced Querying 1
Overview 1
Learning Objectives 2
Lab Goals 2
What You Will Need to Begin 2
Part 1 – Exploratory Data Analysis 3
Setup 3
Setting Up Our Database 3
Basic Summaries – Getting the Details 4
Basic Summaries – Aggregating the Results 7
Advanced Summaries – SQL Judo to Answer Tough Questions 13
Part 2 – Putting All Together 14
What to Submit 15
Appendix A – VidCast Logical Model Diagram 16
Appendix B – Part 2 Results (Larger Size) 17

Overview

This lab is the seventh of ten labs in which we will build a database using the systematic approach covered in the asynchronous material. Each successive lab will build upon the one before and can be a useful guide for building your own database projects.

In this lab, we will use structured query language (SQL) data manipulation commands to query and modify data in the VidCast database.

Read this lab document once through before beginning.

Learning Objectives

In this lab you will

- Demonstrate data manipulation language (DML) proficiency
- Perform basic data analysis using descriptive statistics provided by SQL aggregate functions

Lab Goals

This lab consists of two sections. The first section is a walkthrough of inserting, updating, querying, and deleting data. In the second part of the lab, you will code your own DML queries to solve the problems presented.

TIP: *If you are new to SQL or programming in general, you may benefit from run through of the SQL Tutorial at https://www.w3schools.com/sql/ . While not required reading, it can be a helpful resource for new programmers to get some coding in.*

What You Will Need to Begin

- This document
- An active Internet connection (if using iSchool Remote lab)
- A blank Word (or similar) document into which you can place your answers. Please include your name, the current date, and the lab number on this document. Please also number your responses, indicating which part and question of the lab to which the answer pertains. Word docx format is preferred. If using another word processing application, please convert the document to pdf before submitting your work to ensure your instructor can open the file.
- To have completed Lab 06 – Querying, Inserting, Updating and Deleting
- Understanding of database tables and have reviewed the asynchronous material through Week 7
- One of the following means of accessing a SQL Server installation
    - A connection to the iSchool Remote Lab (https://rds.syr.edu/)
    - A local installation of SQL Server (see Developer edition here https://www.microsoft.com/en-us/sql-server/sql-server-downloads-free-trial)
    - Regardless of how you access SQL Server, you will need to use SQL Server Management Studio to do so.
- The Lab 07 Initialization Script located at https://github.com/chadondata/659Files

Part 1 – Exploratory Data Analysis

Setup

Our database is alive and it's time to start mining the data for some insights. We will use SQL SELECT statements to get some descriptive statistics from our data.

Formatting Note

Look for the "To Do" icon to point out sections of the lab you will need to do to complete the tasks.

Setting Up Our Database

To get all the data needed for this lab, we will have to run a script against your database to reset all the tables and populate them with data.

Perform the following steps to download and run the script that populates your database.

1. Visit https://github.com/chadondata/659Files
2. Download the file "Lab 07 Initialization Script.sql"
3. Open the downloaded file in SQL Server Management Studio (SSMS)
4. Ensure the correct database is selected in the Available Databases box in the toolbar.
5. Execute the script in its entirety (do not select parts of the script to run individually).
6. Close this file. You will not need it again. If something should happen and you need to set your database back to normal, you can run it again, but you're otherwise done with this script.

To ensure the database is properly set up, code and execute the following SQL SELECT statement in a New Query Window:

```
SELECT
    vc_User.UserName
    , vc_User.EmailAddress
    , vc_VidCast.vc_VidCastID
FROM vc_VidCast
JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
ORDER BY vc_User.UserName
```

Your results should look like this:

Results | Messages

	UserName	EmailAddress	vc_VidCastID
1	accurate	ln@facilisiseget.co.uk	14
2	accurate	ln@facilisiseget.co.uk	113
3	accurate	ln@facilisiseget.co.uk	149
4	accurate	ln@facilisiseget.co.uk	376
5	accurate	ln@facilisiseget.co.uk	433
6	accurate	ln@facilisiseget.co.uk	499
7	accurate	ln@facilisiseget.co.uk	540
8	accurate	ln@facilisiseget.co.uk	574
9	accurate	ln@facilisiseget.co.uk	691
10	albite	nisi@vitaemauris.org	738
11	albite	nisi@vitaemauris.org	659

Not all output rows are shown in the preceding screenshot. To confirm you have everything, look in the lower right-hand corner of SSMS. You should see a status bar entry that looks like this:

834 rows

Now, on with the show.

Basic Summaries – Getting the Details

Our stakeholders would like to know some things about how the users are using the system. To start, they would like to know how many videos each user has made. We could, if we want, take the query from above and hand count the number of videos for each user.

There are a couple problems with that. The first is it's not very sustainable. Each time we want to know these numbers, we must recreate that count by hand. Second, we don't have that kind of time. Let's make the computer earn its money.

When performing exploratory analysis using SQL, it is a common strategy to start with a query like the first one we ran. We can then ensure we have all the data points lined up before we start aggregating them.

There is a problem with our first query… See if you can spot it. Here it is again:

```
1 SELECT
2     vc_User.UserName
3     , vc_User.EmailAddress
4     , vc_VidCast.vc_VidCastID
5 FROM vc_VidCast
6 JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
7 ORDER BY vc_User.UserName
```

Not seeing it? It's hard to tell from the SQL, but we are missing some data.

Line 6 is the `JOIN` clause that brings the vc_VidCast and vc_User tables together. Syntactically, it is perfect. However, when we use a `JOIN` as shown on Line 6, it only returns rows from both tables that have perfect matches in one another.

For vc_VidCast, this is not a problem because:

1. We have made the vc_VidCast.vc_UserID column required so a value must exist
2. We have added a Foreign Key constraint to vc_VidCast.vc_UserID that ensures when we enter a value in that column, it exists in the vc_UserID of table vc_User.

This ensures that there is always a perfect match in vc_User for every row in vc_VidCast.

The reverse is not true, however. There is nothing in our database that says we can't add users to the vc_User table if they have no corresponding record in vc_VidCast. For this reason, we are likely to have users who have not made any vidcasts.

Code and execute the following SQL SELECT statement in SSMS.

```
-- Look for users who have not yet made any VidCasts
SELECT * FROM vc_User
WHERE vc_UserID NOT IN (SELECT vc_UserID FROM vc_VidCast)
```

Your results should look like this:

Results | Messages

	vc_UserID	UserName	EmailAddress	UserDescription	WebSiteURL	UserRegisteredDate
1	27	prune	enim.sit.amet@aliquet.edu	NULL	NULL	2017-06-09 10:04:48.000
2	28	embarrass	Nam.ligula@atfringilla.co.uk	NULL	NULL	2018-01-15 05:16:48.000

These two users have not made any videos and are omitted from the first SELECT statement we ran. (If you'd like, run lines 1 through 7 again and scroll down the results. You won't find these two users in the list!).

We need to tell SQL Server to include all users, even if they have no VidCasts in the database. We can do this using the RIGHT keyword before the JOIN on line 6.

Code and execute the following SQL SELECT statement in SSMS.

```
13  -- Be sure to include all vc_User records
14  SELECT
15      vc_User.UserName
16      , vc_User.EmailAddress
17      , vc_VidCast.vc_VidCastID
18  FROM vc_VidCast
19  RIGHT JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
20  ORDER BY vc_User.UserName
```

If you scroll your results window down, you will see the users who have not yet made videos are now included.

250	embarrass	Nam.ligula@atfringill...	NULL

Since there is no value to put in the vc_VidCastID column for these rows, SQL Server sends a NULL. This will be useful knowledge later.

We now have all the detail records we need to build our summary.

Basic Summaries – Aggregating the Results

Aggregate Functions

There are several built-in SQL aggregate functions we can use to show some descriptive statistics. We will use the following.

Function	Purpose
COUNT(*column_name*)	Counts the non-null instances of *column_name* grouped by the specified GROUP BY columns. Column_name can be * to count the existence of a row, but this is inadvisable in most cases because reasons.
SUM(*expression*)	Totals the non-null values in the *expression*. The expression may be a single column name or the result of a mathematical expression
MIN(*expression*)	Shows the lowest value for the the non-null values in the *expression*. The expression may be a single column name or the result of a mathematical expression
AVG(*expression*)	Shows the average of the non-null values in the *expression*. The expression may be a single column name or the result of a mathematical expression
MAX(*expression*)	Shows the highest value for the non-null values in the *expression*. The expression may be a single column name or the result of a mathematical expression

Let's try them out.

Code and execute the following SQL SELECT statement in SSMS.

```
22  -- High-level descriptive statistics for vc_VidCast
23  SELECT
24      COUNT(vc_VidCastID) as NumberOfVidCasts
25      , SUM(ScheduleDurationMinutes) as TotalScheduledMinutes
26      , MIN(ScheduleDurationMinutes) as MinScheduledMinutes
27      , AVG(ScheduleDurationMinutes) as AvgScheduledMinutes
28      , MAX(ScheduleDurationMinutes) as MaxScheduledMinutes
29  FROM vc_VidCast
```

Your results should look like this:

Results | Messages

	NumberOfVidCasts	TotalScheduledMinutes	MinScheduledMinutes	AvgScheduledMinutes	MaxScheduledMinutes
1	834	43782	15	52	90

GROUP BY clause

Whenever you add an aggregate function to a query, you must also include a GROUP BY clause in your statement. This tells SQL Server the levels at which you would like to aggregate the values used in the aggregate functions and in which order.

If you omit the GROUP BY clause or have not properly coded it, you will get an error message.

Code and execute the following SQL SELECT statement in SSMS.

```
31  SELECT
32      vc_User.UserName
33      , vc_User.EmailAddress
34      , COUNT(vc_VidCast.vc_VidCastID) CountOfVidCasts
35  FROM vc_VidCast
36  RIGHT JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
37  ORDER BY vc_User.UserName
```

TIP: *This query is a line for line copy of the previous query, lines 14 through 20 with a modification to line 34. Feel free to copy/paste if you'd like or re-type the entire query if you are a glutton for punishment.*

You will get this error message:

```
Msg 8120, Level 16, State 1, Line 32
Column 'vc_User.UserName' is invalid in the select list
because it is not contained in either an aggregate function
or the GROUP BY clause.
```

This is because we have included unaggregated columns (UserName and EmailAddress) in our SELECT list along with a column that has been aggregated. Because of this, SQL Server needs to know which grouping levels are required and in what order.

TIP: *This lab provides documentation on how to resolve this error. If you get this error in your own projects, follow these steps to resolve it. Emails about this may go unanswered.*

Amend your previous query to look like the following and execute it again.

```
SELECT
    vc_User.UserName
    , vc_User.EmailAddress
    , COUNT(vc_VidCast.vc_VidCastID) CountOfVidCasts
FROM vc_VidCast
RIGHT JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
GROUP BY
    vc_User.UserName
    , vc_User.EmailAddress
ORDER BY vc_User.UserName
```

Your results should look like this:

Results | Messages

	UserName	EmailAddress	CountOfVidCasts
1	accurate	In@facilisiseget.co.uk	9
2	albite	nisi@vitaemauris.org	13
3	architect	a.dui.Cras@mi.edu	12
4	archives	ullamcorper.velit@interdumfeugiatSed.com	16
5	bedtime	enim.Etiam@egetmollislectus.edu	11
6	bewildered	Donec.porttitor.tellus@odioAliquamvulputate.edu	10
7	bicycle	Quisque.porttitor.eros@mi.net	8
8	bid	sed.turpis@hymenaeosMaurisut.co.uk	15
9	bittem	in.consequat@loremsemper.edu	14

For brevity, not all rows have been shown here. You should see a total of **68** rows in your results.

We can now use the aggregated column as something to sort on. To do so, we simply add it to the `ORDER BY` clause.

Amend your previous query to look like the following and execute it again. Only line 40 has changed from the previous query.

```
31 SELECT
32     vc_User.UserName
33     , vc_User.EmailAddress
34     , COUNT(vc_VidCast.vc_VidCastID) CountOfVidCasts
35 FROM vc_VidCast
36 RIGHT JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
37 GROUP BY
38     vc_User.UserName
39     , vc_User.EmailAddress
40 ORDER BY CountOfVidCasts DESC, vc_User.UserName
```

Your results should now look like this:

Results | Messages

	UserName	EmailAddress	CountOfVidCasts
1	ecstatic	blandit.enim.consequat@loremutaliquam.co.uk	22
2	principle	ac.urna@miac.com	19
3	canadian	Curabitur.dictum.Phasellus@eleifendnec.com	18
4	metacarpal	et.magna.Praesent@placeratauguesSed.org	18
5	przewalski	amet@Maurismolestie.org	17
6	silly	accumsan@gravidasagittisDuis.net	17
7	archives	ullamcorper.velit@interdumfeugiatSed.com	16
8	doughnut	ipsum.primis@Cumsociis.com	16
9	groggy	ornare.In.faucibus@egestas.ca	16
10	sines	dui.nec.tempus@sitametrisus.co.uk	16
11	these	parturient.montes@ipsum.ca	16

It's still **68** records, but now the users with the highest count of VidCasts appear at the top. Because we also have UserName in the `ORDER BY` clause, in the case of a tie for VidCast count between users, SQL Server will sort within that count by UserName. See rows 3 and 4 above.

HAVING Clause

Sometimes we want to filter our result set by the result of one or more aggregate functions. By the time SQL Server begins processing the `WHERE` clause, we cannot add our conditional to the WHERE clause. Instead, conditionals based on aggregate functions must be contained within a `HAVING` clause.

The `HAVING` clause immediately follows the `GROUP BY` clause in a SQL `SELECT` statement.

In our example, out stake holders would like to know who our least prolific users are. They would like to know which users have less than 10 vidcasts in the database.

Amend your previous query to look like the following and execute it again. We have added a comment to line 42 and the HAVING clause on line 52. All else is the same as before, so feel free to save some time by copying and pasting!

```sql
-- Our least prolific users
SELECT
    vc_User.UserName
    , vc_User.EmailAddress
    , COUNT(vc_VidCast.vc_VidCastID) CountOfVidCasts
FROM vc_VidCast
RIGHT JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
GROUP BY
    vc_User.UserName
    , vc_User.EmailAddress
HAVING COUNT(vc_VidCast.vc_VidCastID) < 10
ORDER BY CountOfVidCasts DESC, vc_User.UserName
```

Your results should look like this:

Results | Messages

	UserName	EmailAddress	CountOfVidCasts
1	accurate	In@facilisiseget.co.uk	9
2	darcy	uma.justo@orci.edu	9
3	gum	ut@pharetraQuisqueac.com	9
4	spilling	ullamcorper@Mauris.net	9
5	bicycle	Quisque.porttitor.eros@mi.net	8
6	dispatcher	quam@aptenttacitisociosqu.ca	8
7	hygienist	magna.Ut@necumasuscipit.ca	7
8	stay	et.magnis@nonmagnaNam.co.uk	7
9	console	tristique@justoeuarcu.com	6
10	winter	accumsan@ascelerisque.net	6
11	embarrass	Nam.ligula@atfringilla.co.uk	0
12	prune	enim.sit.amet@aliquet.edu	0

This time, all **12** rows fit on the screen, so they are all shown here.

TIP: *The conditional for the HAVING clause requires us to repeat the execution of the function. We cannot use the alias here as we did in the ORDER BY clause.*

Why? Reasons, I guess.

Advanced Summaries – SQL Judo to Answer Tough Questions

Now we want to do some descriptive statistics on the actual duration of finished VidCasts. If we wanted to, we could store the duration, calculating it whenever a VidCast finishes. That may be a design decision we make later, but for now, we'll calculate it at run time.

In SQL Server, we can quickly calculate the amount of time elapsed between two dates. In our case, we want the number of minutes between StartDateTime and EndDateTime for all VidCasts with a Finished status.

Code and execute the following SQL SELECT statement in SSMS.

```
SELECT
    vc_User.UserName
    , vc_User.EmailAddress
    , SUM(DateDiff(n, StartDateTime, EndDateTime)) as SumActualDurationMinutes
FROM vc_VidCast
JOIN vc_User ON vc_User.vc_UserID = vc_VidCast.vc_UserID
JOIN vc_Status on vc_Status.vc_StatusID = vc_VidCast.vc_StatusID
WHERE vc_Status.StatusText = 'Finished'
GROUP BY
    vc_User.UserName
    , vc_User.EmailAddress
ORDER BY vc_User.UserName
```

Your results should look like this:

Results | Messages

	UserName	EmailAddress	SumActualDurationMinutes
1	accurate	In@facilisiseget.co.uk	1555
2	albite	nisi@vitaemauris.org	1971
3	architect	a.dui.Cras@mi.edu	1913
4	archives	ullamcorper.velit@interdumfeugiatSed.com	2374
5	bedtime	enim.Etiam@egetmollislectus.edu	1699
6	bewildered	Donec.porttitor.tellus@odioAliquamvulputate.edu	1944
7	bicycle	Quisque.porttitor.eros@mi.net	1152
8	bid	sed.turpis@hymenaeosMaurisut.co.uk	2792
9	bittern	in.consequat@loremsemper.edu	2303
10	camel	mauris@massanon.edu	1859
11	canadian	Curabitur.dictum.Phasellus@eleifendnec.com	1928
12	carpal	In.faucibus.Morbi@Mauris.ca	1641
13	chef	ultricies.sem@estMauris.edu	1902
14	console	tristique@justoeuarcu.com	1022
15	darcy	urna.justo@orci.edu	1482
16	dispatcher	quam@aptenttacitisociosqu.ca	1368

Query executed successfully.

Again, not all results are shown. You should have **66** total rows.

TIP: *Because we're only interested in VidCasts that are Finished, we do not need to worry about including vc_User records with no VidCasts, so we do not need a LEFT or RIGHT JOIN in our FROM clause.*

Part 2 – Putting All Together

In this part, you'll amend the previous query to show some more descriptive statistics for the VidCast actual duration.

Amend the query from the end of part one, adding the count of VidCasts, minimum, average, and maximum actual durations for each vc_User record. Sort the results in descending order by the count of videos, then by the UserName.

Your results should look like this:

Results | Messages

	UserName	EmailAddress	SumActualDurationMinutes	CountOfVidCasts	MinActualDurationMinutes	AvgActualDurationMinutes	MaxActualDurationMinutes
1	ecstatic	blandit.enim.consequat@loremutaliquam.co.uk	2682	22	29	121	231
2	principle	ac.uma@miac.com	3413	19	29	179	274
3	canadian	Curabitur.dictum.Phasellus@eleifendnec.com	1928	18	14	107	288
4	metacarpal	et.magna.Praesent@placerataugueSed.org	3053	18	29	169	274
5	silly	accumsan@gravidasagittisDuis.net	2851	17	15	167	288
6	groggy	ornare.In.faucibus@egestas.ca	2464	16	14	154	260
7	przewalski	amet@Maurismolestie.org	2664	16	14	166	288
8	sines	dui.nec.tempus@sitametrisus.co.uk	2316	16	14	144	288
9	archives	ullamcorper.velit@interdumfeugiatSed.com	2374	15	14	158	259
10	doughnut	ipsum.primis@Cumsociis.com	1830	15	29	122	216
11	fervent	sollicitudin.adipiscing@egestasrhoncus.net	2031	15	15	135	260
12	these	parturient.montes@ipsum.ca	2101	15	14	140	288
13	bid	sed.turpis@hymenaeosMaurisut.co.uk	2792	14	58	199	288
14	bittern	in.consequat@loremsemper.edu	2303	14	43	164	288
15	chef	ultricies.sem@estMauris.edu	1902	14	29	135	259

Query executed successfully.

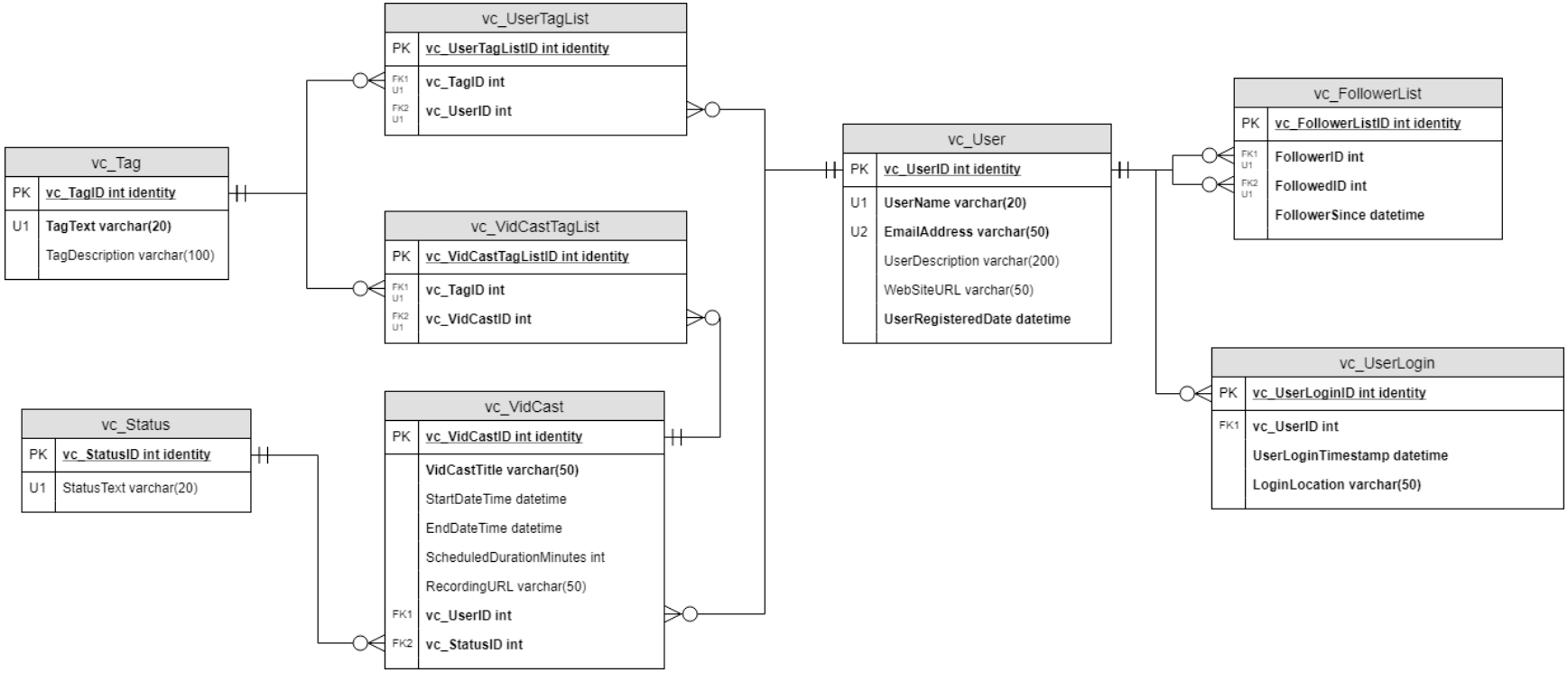
A larger-sized screenshot of the results is shown in Appendix B, below.

Again, your results should have **66** rows.

What to Submit

After completing Part 2, copy and paste the text of your SQL query file at the end of your answers document. Save this document and submit it to the appropriate section on the LMS.

Appendix A – VidCast Logical Model Diagram



For the full diagram, see https://drive.google.com/file/d/1KRqkSvQABuTMXqYAzojTCt9etTSR8Vea/view?usp=sharing

Appendix B – Part 2 Results (Larger Size)

Results | Messages

	UserName	EmailAddress	SumActualDurationMinutes	CountOfVidCasts	MinActualDurationMinutes	AvgActualDurationMinutes	MaxActualDurationMinutes
1	ecstatic	blandit.enim.consequat@loremutaliquam.co.uk	2682	22	29	121	231
2	principle	ac.urna@miac.com	3413	19	29	179	274
3	canadian	Curabitur.dictum.Phasellus@eleifendnec.com	1928	18	14	107	288
4	metacarpal	et.magna.Praesent@placerataugueSed.org	3053	18	29	169	274
5	silly	accumsan@gravidasagittisDuis.net	2851	17	15	167	288
6	groggy	ornare.In.faucibus@egestas.ca	2464	16	14	154	260
7	przewalski	amet@Maurismolestie.org	2664	16	14	166	288
8	sines	dui.nec.tempus@sitametrisus.co.uk	2316	16	14	144	288
9	archives	ullamcorper.velit@interdumfeugiatSed.com	2374	15	14	158	259
10	doughnut	ipsum.primis@Cumsociis.com	1830	15	29	122	216
11	fervent	sollicitudin.adipiscing@egestasrhoncus.net	2031	15	15	135	260
12	these	parturient.montes@ipsum.ca	2101	15	14	140	288
13	bid	sed.turpis@hymenaeosMaurisut.co.uk	2792	14	58	199	288
14	bittern	in.consequat@loremsemper.edu	2303	14	43	164	288
15	chef	ultricies.sem@estMauris.edu	1902	14	29	135	259

Query executed successfully.