The University of British Columbia
Department of Computer Science

CPSC 404: Tutorial/Assignment
Topic: SQL Server DBA Lab Exercise (including Query Plans)

Last Updated: October 18, 2023 @ 14:55

History of Changes:

- Oct. 18: Use the **remote labs** option from home, or use any of: remote labs, UBC Secure (with your own copy of SSMS), or an in-person lab if you are on Campus. This is because of tightened firewall rules within our department, at least until Nov. 1.
- Oct. 17: For Hand In #3: For clarification, I included the word "clustered" for both the Alt. 1 index and the Alt. 2 index. So, we're comparing two kinds of clustered (or clustering) indexes: Alt. 1 and Alt. 2.

Due Date: The due date is **Sunday, November 12, 2023 before 23:59** (near midnight). It is worth 10% of your course grade. Alternatively, you can do the zyBooks work for 10% of your course grade. They both have the same deadlines. Start on it early; don't risk getting putting it off, and then getting sick a few days before the deadline. Remember, we also award part marks; so, hand in what you have done by the deadline.

Late Penalty: 10% penalty per half day late

Written Answers to Questions: Hand in the written answers and screenshots, in Word or PDF format, on Canvas. We'll set up an assignment on Canvas, and you can submit it there. Put your name, student number, and CWL userid in the PDF file.

Outage on the Thursday after the 2nd Tuesday of Each Month: Be advised that our undergrad servers (including SQL Server) and PCs undergo monthly maintenance on the Thursday following the second Tuesday of each month (e.g., Thursday, November 9, 2023) from 22:00-02:00. Plan ahead, and avoid using our servers and PCs during this time.

Whose Hardware and Software? The back-end RDBMS (SQL Server 2014) has been installed and configured on a single server in the Department of Computer Science. The front-end IDE for this tutorial is *SQL Server Management Studio* (SSMS). It has been installed on all the PCs in the undergraduate PC labs, including most, if not all, of these rooms: ICCS 015, X050, X250, X251, and X350 (and maybe elsewhere). You can use your own equipment, or you can use our lab for this tutorial; the latter is recommended if you experience problems with your mix of hardware and software. **There is an additional "lab" option called "remote labs". This feature allows you to access a UBC CPSC lab PC from home, and it will be similar to a lab PC that you would be sitting at, in our CS buildings.** Instructions about how to access the remote labs are given later in this document (search for the string "remote labs").

Instead of using our lab's PC or remote labs: If you wish to install a personal copy of the front-end client software SQL Server Management Studio (about 1 GB of disk space) to allow yourself to work from home or from your laptop, and still access our back-end SQL Server (hosted by the department), you can do so for free by following this link:

https://msdn.microsoft.com/en-us/library/mt238290.aspx

If it says the version of SSMS is 2017 or 2016, then that should be fine, too.

Important note: If you are using your own laptop and you are on the UBCSecure network, then you won't need to use a VPN. However, if you are not on UBCSecure (e.g., you are off campus), then you need to launch the Cisco VPN. The Cisco VPN is available as a free download from UBC's ITServices. The VPN allows you to access our centralized single-server copy of SQL Server. This DBMS runs on a machine called `mayne`, and for security reasons, `mayne` only accepts campus IP addresses. DBA activities require higher privileges than for most undergraduate software; our tech staff have arranged these privileges for you.

What are the advantages of a single-server version of SQL Server? The tech staff can support, maintain, and configure the DBMS server on one machine instead of 100+ machines. Prior to 2015, each lab PC had a copy of SQL Server. Any time that an important software patch occurred, the tech staff had to re-image all the PCs in the lab—a time-consuming and inconvenient process. Then, they had to configure each copy of SQL Server with a set of permissions to permit student access, support database creation, allow database backups and recoveries, etc.—*and these were specific to each machine*. SQL Server requires significant privileges, and that's why our tech staff needed to configure each machine.

The single back-end server will allow you to easily *backup* and *recover* your database, and manage all of your backups on the server. Because of the strict lab/server security permissions, you will not be able to take backups from home, copy them to a USB memory device, and restore them on the Department's version of SQL Server; or vice-versa.

Questions: We have created a series of questions for you to answer as you progress through this lab. Later, you'll upload your PDF document containing your answers. We'll probably only mark a subset of the written questions. You will learn about how to:

- Create a table, index, or other object
- Delete or drop a table, index, or other object
- Navigate through SQL Server
- Run a variety of SQL queries and DBCC commands
- Extract, analyze, and explain metadata and query performance plans
- Backup and restore a database
- Perform other database and DBA activities

Learning Objectives:

- Gain a basic, working understanding of SQL Server.
- Construct and run SQL queries.
- Create, query, interpret, and delete all kinds of database objects.
- Interpret SQL Server metadata.
- Generate, analyze, and explain various query plans. This will give you practical hands-on experience to reinforce concepts from class and from Chapters 12 and 14 of our textbook (or you'll get a preview if you complete this lab exercise before we reach those topics in the lecture).
- Backup and restore a database.
- Add to your resume. After taking CPSC 304 and 404, you will have been exposed to several large, commercial RDBMSs.

Part 1: Introduction to SQL Server—Getting Started

Although you may have used SQL Server for one or two tutorials in CPSC 304 (depending on which term you took CPSC 304), some of these instructions are new, and the deliverables are considerably different for CPSC 404. Please read these instructions carefully:

Connect to SQL Server Locally or Remotely

- [If you are using your own equipment *and* your own copy of SQL Server Management Studio to access the back-end engine, that is, to access the *SQL Server* DBMS hosted by the Department of Computer Science, then perform this series of steps before moving on. "Lab PC" users, including users who are using the CPSC "Remote Labs" feature from home or elsewhere, should bypass this (one-page) section.]

  > Important note from October 18, 2023: Our CPSC tech staff state that students should use **remote labs** for now (e.g., when using your home machine or your laptop from home (or from anywhere else) … or, if you are at UBC, do one of: (a) go into a CPSC physical lab and use a PC there, or (b) log on to the UBC Secure network (and then it's OK to use these one-page "If you are using your own equipment and your own copy of SQL Server Management Studio …" instructions (i.e., SSMS and `runas`)).
  >
  > The reason that the self-install of SSMS (with `runas`, etc. on your own equipment) isn't working as an alternate access method from outside of UBC right now is because of a recent change to UBC/CPSC security protocols. The CPSC tech staff wrote: "The issue is because of the firewall tightening to treat UBC

VPN as non secure. Since undergrad students aren't part of the CS VPN pool, there's no easy way to allow them remote access." However, multifactor authentication (MFA) is coming on November 1, and the tech staff added: "MFA will be available to undergrad students. At that time, some of the firewall rules for the UBC VPN may be relaxed, and access can be restored to how it was previously." (Note that they said "may be relaxed"; so, it's not a certainty, but it is seems likely.)

- [This is still part of the "If you are using your own equipment" instructions] If you haven't already done so, install the front-end client software on your machine. (See "Whose Hardware and Software?" above for instructions.)
- If you are not on the UBCSecure network on campus, then start the VPN.
- To launch SSMS: On a Windows machine, right-click on your desktop. Select New → Shortcut. In the "location of the item" field, enter the following:

  `runas /netonly /user:`*`userid`*` ssms.exe`

  or equivalently:

  `C:\Windows\System32\runas.exe /netonly /user:`*`userid`*` ssms.exe`

  … where *`userid`* is your CWL userid. (If you have multiple versions of SSMS installed, and you find that you're executing an older version, then you can replace `ssms.exe` with the appropriate complete path in double quotes.) Then, click Next, and you'll be asked to name your shortcut (this is just a label for your convenience). Finally, click Finish. (This step only needs to be done once. *Type it exactly as is; be careful with the spaces.*)

- When you double-click this new shortcut icon, the program will ask for your SQL Server password. It will be your student number. Then, continue with the instructions in the section "All users, do this part", a few lines below this one.

- [Users of "**lab PCs**" (either **in-person** in our lab, or using the **CPSC "remote labs"** feature from your laptop or home computer), do this part.] Sign on to the lab PC with your CWL account, either at the keyboard in the lab (and then just use the PC as usual), or by signing in remotely using a web browser from elsewhere.
  - **If you're using the CPSC "remote lab" feature, here is how you get access to a lab PC:**
    - Go to https://remotelabs.ubc.ca/cs and log in with your CWL account. The **/cs** is important; otherwise, you'll be directed to the UBC Library's computers, and they do not run any SQL Server software.

- Choose ICCS EDUC on the LHS (or whatever ICCS room they have available with Windows PCs).
- Connect to any of the PCs in the room (e.g., win05-iccs-educ).
- To logon to the default STUDENTS domain: use your CWL *userid* for your user name (in these instructions, whenever you see *userid*, it means your CWL userid). Then, enter your CWL password. (Note: If your password isn't recognized, and you have one or more special characters in it, it might be that the default sign-in option is based on a French-Canadian keyboard. You can toggle this to an English or US keyboard by going to the bottom right hand corner of the blue login screen. Then, try again.)
- Once you're logged on (it might take a minute), you'll recognize the screen as being just like what you see in the CS undergrad PC labs. The resolution will be a little bit grainy; but, it's still pretty good.
- Later, you can create and edit your SQL scripts in a text file, and then simply copy-and-paste the SQL into SQL Server Management Studio (more about the scripts shortly). Note that your undergrad account's home z: drive will be available to you.

- **Instructions for all "lab PC" users (remote or in-person) are here:**
- To start *Microsoft SQL Server Management Studio 18* on Windows 10, do one of the following (the Windows defaults change from time to time):
  - Click on the Start/flag icon (lower LHS, **but make sure that it is for the lab PC** (and not your own PC if you're using the remote lab feature) → scroll through the list of applications (you can also use the magnifying glass / search function in the lower LHS to find it) → Microsoft SQL Server Tools 18 → Microsoft SQL Server Management Studio 18.
  - Alternatively, right-click on the Microsoft flag icon in the lower left hand corner of the screen. Select Search from the list, and type in `ssms`. That should get you to SQL Server Management Studio. (Note that if you don't right-click on the flag, and instead type `ssms` (without the quotation marks) into the `"Type here to search"` box in the lower LHS, then it will just search the Web for `ssms` using the Bing search engine, which isn't what you want.)
  - Occasionally, you might see only a very limited number of applications on the desktop, and maybe you can't find SQL Server Management Studio. Logging into that machine again, or into a different machine, should do the trick. (Occasionally, I encountered this situation myself.)
- Start (open) SQL Server Management Studio by selecting it.
  - If it takes more than about 60 seconds to open, verify that a warning box isn't hidden behind the big SSMS image in the middle of your screen.
  - You can dismiss/ignore the warning, if you get one. Click OK to continue.

- [All users, do this part.] Sign on to the DBMS using the "Connect to Server" pop-up box. You will be using your SQL Server userid and password which our tech staff have created for you.

  Server Type: `Database Engine`
  Server Name: `mayne.students.cs.ubc.ca,1433\CSUGRADSQL`
  1433 refers to the standard port number for SQL Server. `CSUGRADSQL` is the name of the particular database instance.
  Authentication: `SQL Server Authentication`
  Use your CWL *userid* as your userid, and your student number as the password.

  Then, click on Connect.

- In the Object Explorer panel, you will see the main components found on the server, including the databases that exist. Expand or click on Databases and open an existing database to which you have permission, such as `FoodMart 2008 (Read-Only)`. Expand the list of Tables within it. Right-click on a table name and display up to 1000 rows from the table.

- In SQL Server, tables have *fully qualified names* of the form: `Server.Database.Schema.Table`. Often, it suffices to specify just the `Schema.Table` name, or even the `Table` name (in our examples, you'll sometimes see "`dbo`" listed in Object Explorer, as the Schema name). If you plan to move code from one environment to another (e.g., test to production), it often helps to leave off the `Server.Database` qualifier; otherwise the code won't port well, that is, without modification.
  - Click on the New Query button near the top. You can type and Execute a command like one of the first two examples:

    ```
    use "FoodMart 2008"          (needs delimiter due to space)
    use [FoodMart 2008]          (square brackets are OK)
    use StatisticsTest_userid    (no delimiter needed)
    ```

    This is done to specify a default DB name (once) for the SQL statements that are subsequently issued. This avoids the need to qualify object names every time. If you ever get an "invalid object name" or "does not exist" type of SQL error, that may be a hint that you haven't issued the `use` command or provided an appropriately qualified object name.
  - Enter an SQL SELECT statement of your choice, such as:

    ```
    select  *
    from    employee;
    ```
  - Click the Execute button to run the query.

- *Transact-SQL* or *T-SQL* is Microsoft's implementation of SQL for SQL Server. It extends the ANSI SQL definition somewhat.

- *SQL Server Management Studio* is the front-end to SQL Server, from which DBAs and users can ask queries, and perform all kinds of DB-related activities. When you execute a query, you can monitor how long it takes by looking at the seconds ticking away in the yellow task bar at the bottom of the results screen.

Part 2: Tutorial/Exercise

An excellent SQL Server DBA tutorial on SQL Server Statistics is Holger Schmeling's free tutorial found at URL: http://www.red-gate.com/community/books/sql-server-statistics. Download this short DBA Handbook. We will use numerous parts of it, but we will add many other tasks and deliverables throughout this tutorial. If you Google him, you'll also find two subsets of the book, that is, "Queries, Damned Queries and Statistics", which is the first part of the DBA handbook, and "SQL Server Statistics: Problems and Solutions" which is the second part—along with a zip file of SQL Server scripts (although you can cut and paste these from his PDF documents including his short DBA Handbook (link above)). You can modify these scripts and run them on SQL Server, and this will allow you to complete the exercises and deliverables for this tutorial and exercise.

- Important Note: *Save your SQL scripts, commands, statements, notes, and answers to questions* in files. Use a Notepad file (i.e., `.txt`), a Wordpad file, or other text document for your scripts because you may need to re-create and re-run your objects, perhaps several times. For example, you may need to drop an object, change your SQL, re-create the object, load the data, and re-run your queries. So, it helps to have these files handy, for re-use. I've reused mine *many* times … and if I mess something up, no worries, I can delete the database, copy-and-paste the saved scripts, and re-execute them.
    - If you are using our PC lab (remotely), you will see that you can't copy-and-paste with your mouse from another window on your home PC (or laptop) to the lab PC's window. So, you should create the SQL scripts in your undergrad account—or simply e-mail the scripts from your own PC's file system to your undergrad account. I used Microsoft Explorer and Notepad on the remote "lab PC"—and in-person in the lab, too. All worked fine.

Tip: The following Windows keystrokes using the control (Ctrl) key on your keyboard will come in handy when you're running your SQL scripts in short pieces:

- Ctrl-a → Ctrl-c (i.e., Ctrl-a followed by Ctrl-c) will copy everything in the current window pane to temporary working storage.
    - Occasionally, you might get a copy error using remote labs when using Ctrl-c or Edit → Copy. If so, try again. Most of the time, it's just temporary. Occasionally, Ctrl-c gives an error message, but oddly enough, the Ctrl-v paste actually works for it. (I encountered a bunch of such cases when using remote labs in October 2023, and I reported it to the tech staff. If the errors happen too often, they suggest using a different browser or an incognito/private window, or clearing the cookies and browser cache related to https://*.ubc.ca.)
- Move your mouse to the destination window. Then, Ctrl-a → Ctrl-v will paste that copied text into the destination window, overwriting everything that was highlighted there. This is useful when running SQL statements.
- Alternatively, use the mouse to drag the cursor across a bunch of text in a window and then copy it via Ctrl-c, followed by Ctrl-a → Ctrl-v (which means paste the copied text into the current window, overwriting everything that was highlighted in that destination window). Sometimes, you want to run individual SQL statements, or small parts of a bigger script, one at a time—at least for the first time.

Part 2A

Follow Holger's tutorial called "SQL Server Statistics" for the following steps. Read his notes as you go along; he provides some explanations. Note that he provides a lot (but not all) of the scripts, commands, and SQL statements that we will need for this exercise. You will need to figure out the rest. For your database name, *append a suffix* consisting of an underscore plus your CWL userid to the name `StatisticsTest` so that it reads `StatisticsTest_`*`userid`* all in one string. Each student's database is unique; therefore, after creating your database, you don't have to bother putting a *userid* suffix onto any of your table names and other object names, since they're all contained within your own database.

Answers to questions below marked by the bolded characters "**Hand In #**" must be submitted for marking. There are 11 such questions.

If you try to drop an object, and SQL Server claims the object is in use (and therefore locks are being held by "someone"), note that *that person may be you*. So, you may need to log off of SQL Server Management Studio, and log back in. Don't worry about saving any of your SSMS work when exiting from SSMS; there's nothing that needs to be saved.

1. In this section, you will use and slightly modify Holger's SQL statements and SQL Server commands by copying and pasting them, or by downloading them from Holger's Web pages.
    a. Note that you can always enter a query by using the "New Query" button on the toolbar. Alternatively, you can right-click on the database name and choose "New Query".
    b. Just make sure that you specify the database name in your query (if you aren't already using a specific database—and most of the time you are, so it may not be a problem); otherwise, SQL Server won't know which database your query applies to. You can issue a global command, and SQL Server will remember your database name. Then, you can simply use the unqualified table names after that, since tables will be unique within your database. For example:

```
use StatisticsTest_userid;
go
```

        Tip: If your database is "in use" and you want to drop it, then before issuing your SQL `drop` statement, you could issue this statement:

```
use "FoodMart 2008"
go
```

        and then usually (not always) your `StatisticsTest_userid` database will no longer be locked by your session. If it is still locked, then just exit SQL Server Management Studio, and launch it again.

2. First, you will create your `StatisticsTest_userid` database along with a table called `Numbers`; however, change Holger's SQL so that we insert (load) only 1,000,000 rows, not 5,000,000. On a lightly loaded server, 1,000,000 insertions may take 1-2 minutes. You can see the time counting away in the lower RHS of the SSMS window. We will also create the empty `Product` table at this stage, using Holger's SQL. (Later on, we will also load 1,000,000 rows into the `Product` table.) The number of rows hasn't been a problem in the past, but if there are currently too many users on our shared database server, and your SQL statements start to take much more time than expected (like more than 4 minutes for this step), you may wish to reduce the number of rows and re-run your SQL scripts.
    a. Execute the query to create your database and perform your insertions, using the "Execute" button (or press F5).
    b. You may have to right-click on Databases within Object Explorer to Refresh the metadata that is displayed.

c. After creating the table, verify that the expected rows are in the table, either with an SQL SELECT statement; or find the table in the LHS Object Explorer and right-click on the table, and then choose "Select Top 1000 Rows".
d. As an exercise, drop the table that you just created; however, do *not* drop the `StatisticsTest_userid` database.
    i. Re-create the `Numbers` table (and its data) within the `StatisticsTest_userid` database. You can reuse part of your previous script.
e. Backup the `StatisticsTest_userid` database, as per the instructions in **Part 3A** near the bottom of this document. Note that:
    i. Part of those procedures include having you update a row or field in the `Numbers` table in `StatisticsTest_userid` *after* you create your backup. Then, pretend that your update was a bad series of updates, and recover the database back to the way it was before you made your change. The instructions in Part 3A will say more.
    ii. Those procedures will also have you restore (recover) the `StatisticsTest_userid` database to the way it was in Step (e), as per the instructions in **Part 3B** near the bottom of this document. Note that those instructions indicate what you're supposed to hand in as your first deliverable; and it is called: **Hand In #1**. (You should collect all your "Hand In" items in one file. You'll submit this at the end.) After the recovery, verify that the "erroneous" update that you made just after your backup … is no longer there.

3. Next, let us insert 100,000 rows with a value of 1,000 for column `c1`, and any value for column `c2`, into (new) table `T0` that your created as follows:

```
create table T0(c1 int not null, c2 nchar(200) not null default '#')
```

Afterward inserting the 100,000 rows, insert exactly one row with a value of 2000. Then, create a nonclustered (unclustered) index for `T0` on column `c1`.

a. Verify the presence of the nonclustered index by using the refreshed Properties (for the index) in Object Explorer (especially the Fragmentation option).
b. **Hand In #2:** Take a screenshot of the Fragmentation results to hand in.
c. **Hand In #3:** SQL Server uses Alt. 1 indexes, which are always clustered indexes. Answer the following 3 questions about Alt. 1 indexes:
    i. Describe the kind of SQL statement(s) that would clearly benefit from using an Alt. 1 clustered index instead of an Alt. 2 clustered index. Give an example of a table and some SQL (you don't have to create the table or run

the SQL statement). Alternatively, you can just use words to describe your example. Justify your answer for why it would benefit.

ii. Describe the kind of SQL statement(s) that would *not* benefit from using an Alt. 1 clustered index instead of an Alt. 2 clustered index. Justify your answer. Again, describe the table and provide either an SQL statement or a description of it in words.

iii. If a leaf page splits in an Alt. 1 index, will any changes be required to the nonclustering indexes (if any) that are used by this table? Explain.

4. In the next few steps, we will have you select rows from `T0` using 3 different WHERE clauses to yield the following access paths: (a) a *table scan* for `c1 = 1000`, (b) an *index seek (nonclustered)* with *RID lookup* (with fetch) for `c1 = 2000`, and (c) another *index seek (nonclustered)* for `c1 = 1500`.
    a. Table Scan case: Type—but do not execute (yet)—the following SQL statement:
        i. `SELECT c1, c2 FROM T0 where c1=1000`
    b. All queries will have a *plan*. In SQL Server Management Studio, click on the icon called *Display Estimated Execution Plan* (or ctrl-L, or Query → *Display Estimated Execution Plan*) to show the estimated query plan *before* executing your query. (Note that these are toggle switches; so, be careful that you don't turn them off when you really want them on.) Go to the Execution Plan tab near the middle of your screen, and hover (move) your mouse cursor over the icons (small images) in the Execution Plan output to see more details about the plan that the optimizer will use. You can see what kind of access path (e.g., table scan) is going to be used by the plan, the estimated I/O cost (e.g., number of reads), the estimated CPU cost, and the estimated number of rows to be returned by the query.
        i. Note: In some environments (not the lab), you may get a permissions error, or you might not see the execution plan tab. If so, you need to issue the following SQL DCL (Data Control Language) statement:
            1. `grant showplan to public;`
                a. This request is somewhat paradoxical because even though you're the database creator, you do not have permission to see the execution plan; however, as the creator, you can grant yourself permission to see the plan.
    c. Click on the icon: *Include Actual Execution Plan*. Finally, execute your query.
        i. Note that you will see tabs for Results, Messages, and the Execution Plan after the query is executed. Click on Messages, and then on Execution Plan.
        ii. Hover your cursor over the icons in the Execution Plan window. Note the actual statistics for the plan. There is nothing to hand in, yet.
    d. Repeat this estimated vs. actual procedure for the other 2 queries shown below. Examine and note the plan in each case (i.e., compare the actual statistics to the

estimated statistics for the plan operation—especially the number of rows that qualify). There is nothing to hand in, yet.

i. Index Seek case with Nested Loop Inner Join:

```
1. SELECT c1, c2 FROM T0 where c1=2000
```

ii. Index Seek case with Nested Loop Inner Join:

```
1. SELECT c1, c2 FROM T0 where c1=1500
```

e. **Hand In #4**: Explain what the 5 terms: *table scan*, *index scan*, *index seek*, *clustered index scan*, and *clustered index seek* mean in SQL Server. (One sentence each.)

5. T-SQL variables are declared with the DECLARE statement. Let us create a "template" for a query like those above (again, see Holger's tutorial), albeit with a variable for `c1 = @x`. Run it, and note the execution plan.
   a. Note that the presence of a variable changes the behaviour: a table scan is performed instead of a more efficient index seek. (At this time, your table is probably in memory; so, you might not notice the difference in performance. Both may be very fast.)

6. Create a new table `T1` with 4 columns (one of which allows null values) and populate it with 100,000 rows from `Numbers`, as per Holger's suggested script. Also, create a nonclustered index on `T1` for column `x`. Examine the execution plan.

7. Let us view some metadata in the form of statistics for use by the optimizer. This data is similar to what is collected by IBM's DB2, Oracle, and many other relational database systems. DB2's statistics gathering command/utility is called `RUNSTATS`. SQL Server uses the `CREATE STATISTICS` command.
   a. Right-click on Databases→ Refresh.
   b. Expand the objects to see the ones you've created.
   c. Expand/click on the Columns, Keys, Constraints, and Indexes objects (and their Properties, where applicable) within table `Numbers`. Examine these objects. We will deal more with them, later.
   d. Expand Statistics within `Numbers`. Right-click on Properties for the primary key `PK_Numbers`. Note the presence of a check box that lets you immediately update the statistics for the columns that apply to a given index. Otherwise, SQL Server has rules of its own for when the statistics get updated. More details are given in various places in Holger's tutorial, in case you are interested.

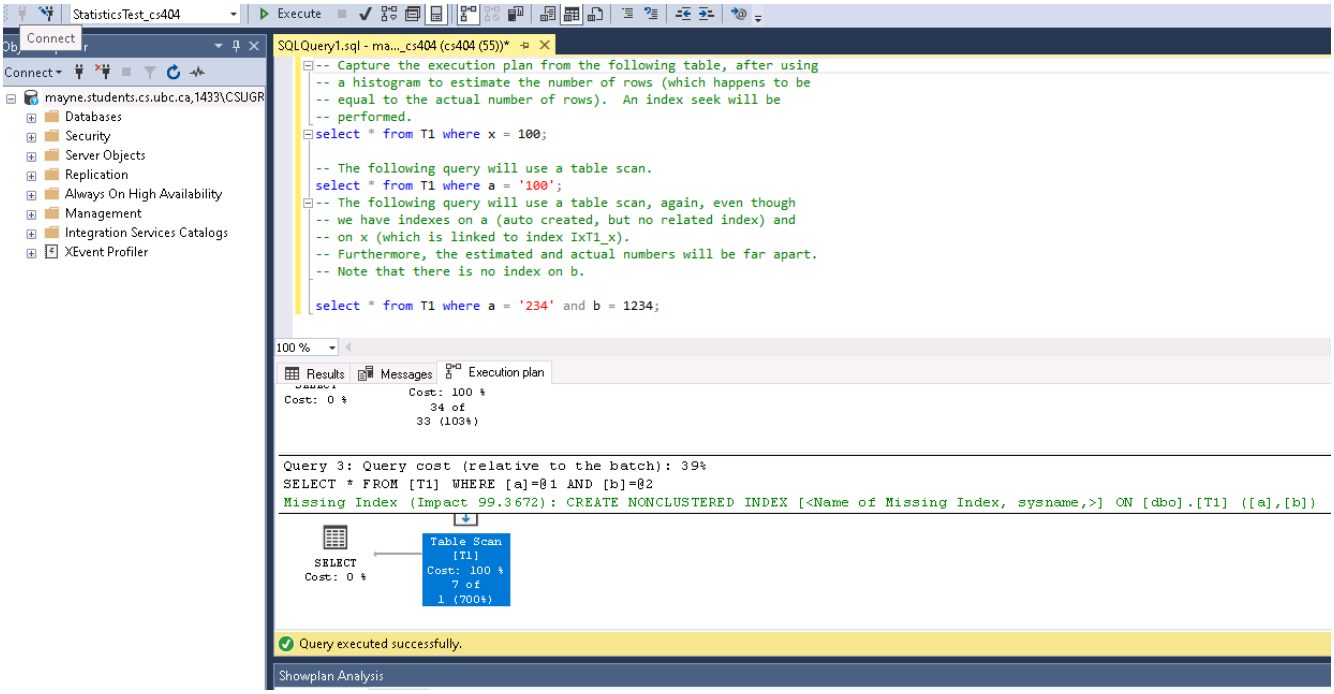
e. On the same Statistics properties panel in the upper LHS, click on Details to see a histogram of the key values.
    i. Click Cancel.

8. Let us take a look at database statistics including histograms.
    a. Run Holger's `sys.stats` query to return all existing statistics for every user table inside the selected database.
    b. The three columns show the table name, the name of the statistics *object* (not the `StatisticsTest_userid` object), and the time of the last update. The latter figure can be used to determine when SQL Server updates the statistics during its operations.
    c. Display the statistics for the index on `T1`, column `x`—if your environment allows it (if not, go to Object Explorer like in the last question, to see the histogram via Properties → Details):
        i. `DBCC SHOW_STATISTICS('dbo.T1', IxT1_x)`
            1. DBCC stands for SQL Server's *Database Console Commands*. Some DBCC commands are restricted because they affect many users; but, this one should be OK in the lab.
            2. You'll see the `RANGE_HI_KEY`, etc., similar to what you'd see in a histogram. Does this information seem reasonable? Compare it to what you see in the table, by right-clicking on the table name, and selecting the top 1000 rows.
            3. Make sure that you understand what the histogram columns mean.

9. To prepare for Hand In #5 below, let us make a note of, and examine, the *execution plan* (make sure the toggle switch is on) for each of the following queries, after using a histogram to estimate the number of rows. Read along in Holger's tutorial (page 13).
    a. An *index seek, RID lookup, and nested loops join* will be performed when issuing this SQL statement:
        i. `select * from T1 where x = 100;`
    b. A *table scan* will be performed, instead of an index scan (think about why) for:
        i. `select * from T1 where a = '100';`
        ii. Note that the Execution Plan messages include a comment about a missing index, with a potential impact of 99%+ on performance.
    c. The following query will qualify on two attributes. What do you expect the query plan to be like? (Not to be handed in.)
        i. `select * from T1 where a = '234' and b = 1234;`
        ii. Again, the Execution Plan messages include a comment about a missing index, with a potential impact of 99%+ on performance. Compare the

estimated number of rows with the actual number of rows. (Hover your mouse cursor over the plan details to see this.)

iii. **Hand In #5**: Right-click in the Execution plan window, and choose Analyze Actual Execution Plan. Click on the radio button for query (c) above—possibly called "Query 3" in the Execution Plan output (if you ran all 3 queries together)—or, it might be the only one—and then take a **screenshot** of your execution plan (for that query including its "Missing Index" message), but **be sure to include your database name** at the top LHS, so that we know this screenshot belongs to you. Below is an example. By the way, if you right-click in the query plan output, you can capture all your query plans in readable format by choosing "Save Execution Plan as …" (but that's not required for handing in).



d. Next, let us create the two-column index that the query plan recommended:
   i. `create nonclustered index IxT1_ab on T1(a,b)`
e. Refresh the Statistics for that new index in Object Explorer, and examine Properties → General.

Part 2B

We'll continue to use database `StatisticsTest_`*`userid`*.

Follow the second part of Holger's tutorial (page 27+) for the following steps. Again, he provides many—but not all—of the scripts, commands, and SQL statements that we need for this exercise. Save your scripts, commands, statements, and notes. You may need to re-create your objects. You should continue to update your "Hand In" document to answer the questions that are interleaved below.

10. Let us consider local variables in T-SQL scripts. We'll try to figure out how to make use of statistics in situations that are more difficult. The following script results in a table scan at execution time, but you might have thought that an index scan would have been faster and would have been chosen instead. Let's compare execution plans.
    a. Write down the *estimated* vs. *actual* number of rows for this query plan:

```
declare @x int
set     @x = 2000
select  c1,c2
from    T0
where   c1 = @x
```

    b. Solution #1: Create and execute a *stored procedure* (call it `getT0Values`) to try to avoid the table scan, using parameter sniffing.
        i. `create procedure getT0Values(@x int) as …`
        ii. `exec getT0Values 2000`
            - If this fails on security (it shouldn't in the lab), enter the SQL DCL statement:
                - `GRANT EXEC ON dbo.getT0Values TO PUBLIC;`
                    i. Again, this seems odd, since you don't have privileges to run it yourself; but, you can grant permission to everyone to use it—and that includes yourself.
        iii. Note how many seconds it took. Compare the output to that of Part (a). (There's nothing to hand in for this.)
        iv. This was an index scan, but unfortunately, the optimizer will assume all future invocations share the same profile as the first. Thus, a second execution, this time of the form `exec getT0Values 1000` will retrieve many rows, not just one; it is inefficient.
        v. Note how many seconds it took. Compare the output to that of Part (a). (There's nothing to hand in for this.)

    c. Solution #2: Use *dynamic SQL*. Try both the "2000" and "1000" cases. Here's how to run the "2000" case:

```
declare @x    int,
        @cmd  nvarchar(300)
set @x = 2000
set @cmd = 'select c1,c2 from T0 where c1='
```

```
            + cast(@x as nvarchar(8))
exec(@cmd)
```

i. **Hand In #6**: What is *dynamic SQL*? What is the difference, if any, between the "2000" and "1000" cases? (Hint: Look at the query plans.)

d. Solution #3: Another way of executing dynamic SQL is to use the `sp_executesql` stored procedure. Try both the "2000" and "1000" queries using the following template:

```
exec sp_executesql N'select c1, c2 from T0 where c1=@x'
                  ,N'@x int'
                  ,@x = 2000
```

11. Predicates in expressions sometimes result in a table scan. Compare the execution plans for the following 5 queries. Some are table scans and some are index seeks. (There is nothing to hand in.)

```
select      c1, c2
from        T0
where       sqrt(c1 * 10) = 100;

select      c1, c2
from        T0
where       sqrt(c1) = 100;

select      c1, c2
from        T0
where       c1 = 10000 / 10;

select      c1, c2
from        T0
where       power(c1,2) <= 1000000;

select      c1, c2
from        T0
where       power(c1,1) = 10000;
```

12. Parameterization in T-SQL scripts. Check out the estimated number of rows versus actual number of rows for the following statement. (There is nothing to hand in.)

a. In Question 10(b) above, we used a stored procedure. Let's consider a variation:

```
create procedure new_getT0Values(@x int) as
      set         @x = @x * 2
      select      c1, c2 from T0
      where       c1 = @x
```

b. In this step, we will execute the stored procedure. Note that an index scan is used in all cases, but only the *initial* invocation results in a storage of statistics; therefore, subsequent calls use the same query plan and this may be highly inefficient in some cases (such as the "500" case below where an index seek was used but a table scan would have been better).

```
   i.  exec new_getT0Values 2000;
  ii.  exec new_getT0Values 1000;
 iii.  exec new_getT0Values 500;
```

c. Drop the stored procedure called `new_getT0Values`.

13. Steps 13, 14, and 15 go together. In these steps, we're going to be dealing with a quantity *N* which is the number of rows that we're going to insert into the `Numbers` and `Product` tables. Let us start with *N* = 1,000,000 rows. This gives good, differentiable results for a user on a lightly loaded system. One million rows seem to provide conditions where we can see the advantages and disadvantages of using different plans and *access paths* such as table scans, index scans, and index seeks. (For small tables, there may not be any measurable difference in performance between various access paths. Also, when larger tables can fit into memory, we may not see a whole lot of difference in performance since there will be relatively few page faults—and, as we've seen in class, page faults are a big contributor to overall I/O costs.) The server `mayne` is our database server and it has a lot of memory: the number of buffer pages *B* is quite large for SQL Server's buffer pools.

Important Note: If you find that the value of *N* is too small (i.e., queries run too fast) or conversely that *N* is too big (i.e., queries run too slow), then you should adjust *N* upwards or downwards (e.g., try 1,500,000 or 500,000). For example, if there are many concurrent users of `mayne`, then you will likely see poorer response, especially on disk-bound jobs, and this may mean that *N* = 1,000,000 is going to take far too long. So, to avoid frustration, *and yet still get meaningful figures for comparison*, you should change the value of *N* (see the comments in Step (b) below) and re-run all of your tests. Be sure to clearly state the value of *N* that you are using, when you hand in your results.

Here is the initial set of steps. To prepare ourselves for measuring results on the `Product` table, do the following steps (most of these scripts/statements are from what you did earlier):

a. Drop the existing `StatisticsTest_`*`userid`* database.
b. Create the same (empty) database, and specify simple recovery for the log:

```
alter database StatisticsTest_userid set recovery simple
```

c. Before creating the `Numbers` table, let's run the following two SQL statements to turn off automatic statistics updates so that we can experiment with a few things during our benchmarking tests:

```
alter database StatisticsTest_userid
      set auto_update_statistics off

alter database StatisticsTest_userid
      set auto_update_statistics_async off
```

Alternatively, you can right-click on the database name in Object Explorer and choose Properties → Options. There, you can toggle the options; but, leave `auto_create_statistics` set to `True` (i.e., on).

d. Create the same (empty) `Numbers` table as you did earlier.

e. Insert N = 1,000,000 (one million) rows into the `Numbers` table (again, similar to what you did near the beginning of the tutorial). Warning: Holger's original script starts with 5,000,000 rows; therefore, you should be careful to change this to 1,000,000 rows—at least to start. You can adjust N and re-run the steps, if necessary.

- It took me about 10 seconds to insert the rows. If it takes you more than 90 seconds, then you should change N and repeat all of Step 13.

After doing the above preparatory steps, execute the following SQL statements with respect to the `Product` table. In particular, using Steps (f)-(i) below, we will create an empty `Product` table, a clustered primary key on the `ProductID` field, an unclustered index on the `LastUpdate` field, and an unclustered index on the `ListPrice` field; but, we won't populate (load) the `Product` table yet.

```
f. create table Product ...        [just like before]
g. alter table Product add constraint PK_Product
                primary key clustered (ProductId)
h. create nonclustered index ix_Product_LastUpdate on
                Product(LastUpdate)
i. create nonclustered index ix_Product_ListPrice on
                Product(ListPrice)
```

14. At this stage, we have a populated `Numbers` table and an empty `Product` table. For the following two sequences of runs (i.e., middle and right columns of the **fill-in-the-blank table below**), try to predict (without handing in the prediction) how long each step might take. **Hand In #7: You will complete and hand in the following 3-4 page table.** It will be worth a bunch of marks. For handing in: write down the amount of elapsed time that each step took. You'll find the **elapsed time** in the bottom RHS of the results pane in SQL

Server Management Studio. Elapsed time can be used as a proxy for CPU time, meaning that it will likely correlate strongly with the amount of disk operations and the amount of CPU time, although contention/locking and other overhead activity could affect your results. Please note that there are a lot more instructions, explanations, and helpful tips for this fill-in-the-blank table in Steps 13, 14, and 15 which *follow* the table—READ THEM AS YOU PROCEED THROUGH THE 3-4 PAGE FILL-IN-THE-BLANK TABLE. Don't limit yourself to just what's written in the table's notes; be sure to read along! In particular, many more useful explanations and instructions about Step 14 come after the 3-4 page table … see the notes about Step 15 further below.

a. Fill in the following table with your:
    i. Elapsed times specified as integers (seconds)
    ii. Number of logical reads
        - You get these by issuing the following statement prior to your queries—and then this option remains in effect for all of your queries:

```
set statistics io on
```

    iii. Although you won't hand in any deliverables by using the following statement, you might find it useful in the future, when you want to see summaries (rather than details) of access paths:

```
set showplan_text on
-- change "on" to "off" to discontinue
```

A summary of the instructions is found in the "Description of Task" column—i.e., the first column. Do **Method 1** in the table first (middle column); but, skip any steps that have "n/a" (not applicable) or "Skip this step" listed. Then, after finishing Method 1, move on to the start of **Method 2** found in the RHS of the table. Record all your time measurements in seconds so that you can easily compare the numbers.

(The table starts on the next page.)

Hand In #7: Hand in the following table, filled in with your actual results:

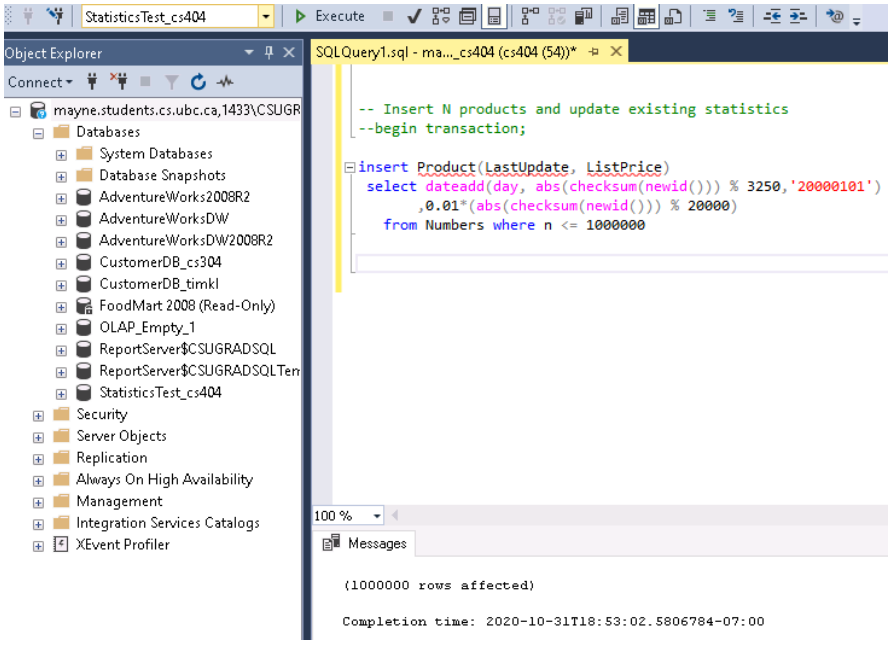
Description of Task (Note that the step numbers have more details below.)	**Method 1: Build Indexes, etc. at Start (Runtime in Seconds)**	**Method 2: Build Indexes, etc. Later (Runtime in Seconds)**
What value of *N* are you using? (The suggested default is *N* = 1,000,000 rows, but you may need to adjust this.)	*N* = _____ Use the same *N* for Method 1 and Method 2.	
Step 13(f)—Create `Product` table, but don't insert data yet	(already done) Time = 0 (sec.)	Time = 0 (sec.)
Step 13(g)—Create clustered PK `ProductID` index	(already done) Time = 0	Skip this step.
Step 13(h)—Create unclustered `LastUpdate` index	(already done) Time = 0	Skip this step.
Step 13(i)—Create unclustered `ListPrice` index	(already done) Time = 0	Skip this step.
Step 15(a)—Insert *N* rows into `Product` table.	Time = _____ **Hand In #8:** Take a screenshot of your work, including your userid. An example is shown below, after this table.	Time = _____
Step 15(b) Report the size of your database, broken down into 2 parts: (a) data portion, (b) log portion.	Data = _____ MB Log = _____ MB	Data = _____ MB Log = _____ MB
Step 15(c) Report the size of your database, after using the Shrink utility.	Data = _____ MB Log = _____ MB	Data = _____ MB Log = _____ MB

Step 15(d)—View the statistics timestamps	How many statistics objects are there for this DB? _____ What was the most recent timestamp? _____	How many statistics objects are there for this DB? _____ What was the most recent timestamp? _____
Step 15(e)—7 SQL queries Specify "Include Actual Execution Plan" before running these queries. Afterwards, right-click in the *Execution plan* window/tab, and choose Analyze Actual Execution Plan. Then, one-by-one write down the type of access path used. Choose from these only, and ignore other details like parallelism and hash match: • Table Scan • Index Scan • Index Seek • Clustered Index Scan • Clustered Index Seek The number of logical reads comes from the Messages tab. Some examples are provided; but you can overwrite them.	Time = _____ <u>General Access Plan</u>: Q1) e.g., Index Scan Q2) Q3) Q4) Q5) Q6) Q7) <u># Logical Reads for Queries</u>: Q1) e.g., 1,616 Q2) Q3) Q4) Q5) Q6) Q7)	Time = _____ <u>General Access Plan</u>: Q1) e.g., Table Scan Q2) Q3) Q4) Q5) Q6) Q7) <u># Logical Reads for Queries</u>: Q1) e.g., 142,858 Q2) Q3) Q4) Q5) Q6) Q7)
Step 15(d), again—View the statistics timestamps.	Were there any new statistics? _____ If so, what was the most recent timestamp? _____	How many statistics objects are there for this DB? _____ What was the most recent timestamp? _____
Step 13(g)—Create clustered PK `ProductID` index	n/a (= not required for the rest of this middle column)	Time = _____

Step 15(d), again—View the statistics timestamps	n/a	How many statistics objects are there for this DB? _____ What was the most recent timestamp? _____
Step 15(e), again—7 SQL queries (similar to above, but new results) • Table Scan • Index Scan • Index Seek • Clustered Index Scan • Clustered Index Seek	n/a	Time = _____ <u>General Access Plan</u>: Q1) e.g., Clustered Index Seek Q2) Q3) Q4) Q5) Q6) Q7) <u># Logical Reads for Queries</u>: Q1) e.g., 7 Q2) Q3) Q4) Q5) Q6) Q7)
Step 13(h)—Create unclustered `LastUpdate` index	n/a	Time = _____
Step 13(i)—Create unclustered `ListPrice` index	n/a	Time = _____

Step 15(e), again—7 SQL queries (similar to above, but mostly new results) • Table Scan • Index Scan • Index Seek • Clustered Index Scan • Clustered Index Seek	n/a	Time = _____ General Access Plan: Q1) e.g., Clustered Index Seek Q2) Q3) Q4) Q5) Q6) Q7) # Logical Reads for Queries: Q1) e.g., 7 Q2) Q3) Q4) Q5) Q6) Q7)
Step 15(b) Report the size of your database, broken down into 2 parts: (a) data portion, (b) log portion.	(same as before)	Data = _____ MB Log = _____ MB

Here is an example of the screenshot for **Hand In #8** (with the DB name in the upper LHS):



Hand In #9: When the whole 3-4 page table is filled in, compare your results for Method 1 and Method 2, for the 7 queries done at various points. Point out some reasons for why the results (logical reads) were good for the middle column right off the bat, and the right column started off slow, but then greatly improved. Then, comment on whether or not you think it is better to create the indexes and statistics early, or later.

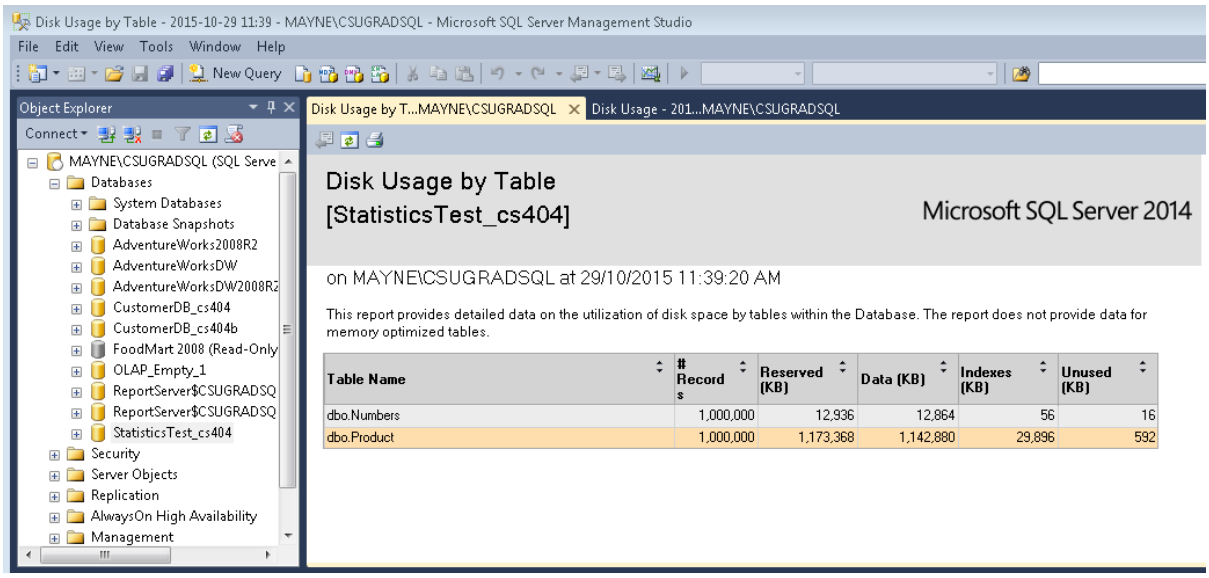
15. As per the SQL DML statement below, we're going to insert *N* products into the `StatisticsTest_userid`'s `Product` table.

    a. As before:

```
insert Product(LastUpdate, ListPrice)
        select      dateadd(day,
                    abs(checksum(newid())) % 3250,
                        '20000101'),
                    0.01*(abs(checksum(newid())) % 20000)
        from        Numbers where n <= 1000000;
```

        i. It took me about a minute to insert the *N* = 1,000,000 rows. If it takes you less than 15 seconds or more than 5 minutes, then consider changing *N* and repeating all of Step 13 and Step 15(a).
        ii. At this stage, the `Numbers` and `Product` tables and indexes have no statistics gathered on them.

    b. Determine the size of your database (in MB), as follows:
        i. In Object Explorer, right-click on Databases, and choose Refresh.
        ii. Expand the list of databases.
        iii. Right-click on `StatisticsTest_userid`.
        iv. Click on Reports → Standard Reports→ Disk Usage.
        v. Write down the size of each of these two major components/files of your database: data and log. It is OK to round up to the nearest integer (ceiling function).
        vi. **Hand In #10**: To prove to your marker that you have successfully created the database objects so far, click on Reports → Standard Reports→ Disk Usage by Table, and take a screenshot of your image, like you did for Hand In #1. The portion that you must hand in looks like this:



c. Use the database Shrink utility to reduce the size of your log file.
   i. In Object Explorer, right-click on Databases, and choose Refresh.
   ii. Expand the list of databases.
   iii. In Object Explorer, right-click on `StatisticsTest_`*`userid`*.
   iv. Click on Tasks → Shrink→ Database.
      1. OK
   v. Write down the new size of your database *log*, as found in Reports → Standard Reports→ Disk Usage. Round your answer up to the nearest integer by taking the ceiling. There is no need to do a screen capture for this step.

d. Run the following statistics metadata query to determine the time at which various statistics were gathered.

```
select    object_name(object_id) as table_name,
          name as stats_name,
          stats_date(object_id, stats_id) as last_update
from      sys.stats
where     objectproperty(object_id, 'IsUserTable') = 1
order by  last_update;
```

e. Next, we will execute the following 7 queries (one after another, in one script, but read the next few paragraphs for (e) first). You will write down how much elapsed time it took to run them (one elapsed time total for all 7 queries), and also write down each of the 7 queries' access paths and their number of logical reads for the `Product` table (only).

You probably only need to execute this command once, and you'll get it for all of your SQL queries from this point on:

```
set statistics io on
```

Specify "Include Actual Execution Plan" before running the following 7 queries. Afterwards, right-click in the *Execution plan* window/tab, and choose Analyze Actual Execution Plan. Then, one-by-one write down the type of access path used. You can get the number of logical reads from the Messages tab produced by the following SQL Server command.

The 7 queries that we will regularly use for the remainder of this tutorial are as follows. They are not found in Holger's tutorial notes.

```
-- Query 1
select      count(*)
from        Product
where       ProductID between 10000 and 10019;

-- Query 2
select      count(*)
from        Product
where       ProductID < 50000;

-- Query 3
select      count(*)
from        Product
where       ListPrice <= 0.10;

-- Query 4
select      *
from        Product
where       ListPrice <= 0.10;

-- Query 5
select      LastUpdate, count(*) as count
from        Product
group by    LastUpdate;

-- Query 6
select      ListPrice, count(*) as count
from        Product
where       ListPrice < 0.05 and LastUpdate < '2001-01-10'
group by    ListPrice;

-- Query 7
select      filler, count(*) as count
from        Product
group by    filler;
```

(These 7 queries complete the middle column of the fill-in-the-blank table.)

f. **What's mentioned next applies to the RHS column in the fill-in-the-blank table.**

Use the same value of *N* that you did for the middle column of your table. We're going to re-create the `StatisticsTest_userid` database. Repeat Steps 13(a)-(f), but do not build any indexes on the `Product` table, yet.

Fill in your answer to Step 15(a) in the RHS column.

Then, do Steps 15(b) and 15(c).

Next, you'll need to run the statistics metadata query from Step 15(d) to see if there are any changes. (You don't have to explain anything, yet.)

And, once again, to gather logical I/O and other statistics:

```
set statistics io on
```

Continue with Step 15(e), and then Step 15(d) again.

g. After creating the clustering index described in Step 13(g) and shown below in the `alter` statement, re-run the statistics metadata query in Step 15(d).

```
alter table Product add constraint PK_Product
      primary key clustered (ProductId)
```

h. Now, re-run the 7 queries to see if there are any changes in elapsed time or logical reads.

i. Create an unclustered index on LastUpdate, as described in Step 13(h):

```
create nonclustered index ix_Product_LastUpdate on
      Product(LastUpdate)
```

j. Create an unclustered index on ListPrice, as described in Step 13(i):

```
create nonclustered index ix_Product_ListPrice on
      Product(ListPrice)
```

k. Re-run the 7 SQL queries in Step 15(e).

l. Determine the size of your database, as described in Step 15(b).

m. Expand the Indexes and Statistics for your table, using Object Explorer. Note the current list of indexes and statistics (but there's nothing to hand in).

16. For our final set of activities, we will perform an **Export** of data followed by an **Import** of data. This assumes that you have N = 1,000,000 rows (or equivalent) in the Product table … so that means that you can continue to use your existing data in this series of steps.

    To Export:

- In Object Explorer, right-click on the database name
- Choose Tasks → Export Data …
- Using the SQL Server Import and Export Wizard:
    - Next
    - For Data Source, choose SQL Server Native Client 11.0
        - The server should automatically be `mayne...`
    - For Authentication: Use SQL Server Authentication
        - Provide your CWL ID
        - Provide your SQL Server password (e.g., student number)
    - Your database name should already appear
    - Next
- For Destination, choose Flat File Destination
    - Provide the output file name `z:\Product.txt`
        - Use the Browse… button to verify that this file doesn't already exist in your Z: home directory
            - If it already exists, right-click on the file name, and delete it.
        - Cancel (the popup box).
    - Next
    - Choose the radio button: "Copy data from one or more tables or views"
    - Next
- For "Configure Flat File Destination", pick the table to export (e.g., [dbo].[Product]):
    - Click Preview
        - Verify that it looks OK
        - OK
    - Click Edit Mappings…
        - Verify that the Source and Destination (at the top) are: `[dbo].[Product]` and `z:\Product.txt`
        - Check "Create destination file"
        - Verify that the Type looks OK for each field (you might have to expand (drag the right boundary of) the header column for Type to see these):

            - DT_I4
            - DT_NUMERIC
            - DT_DBDATE
            - DT_WSTR
        - OK
    - Next
    - Choose: Run immediately
    - Next
    - Finish
        - It should take about 30 seconds to export 1,000,000 rows.
        - The result will be a very large `z:\Product.txt` file (about 514MB).
- Close

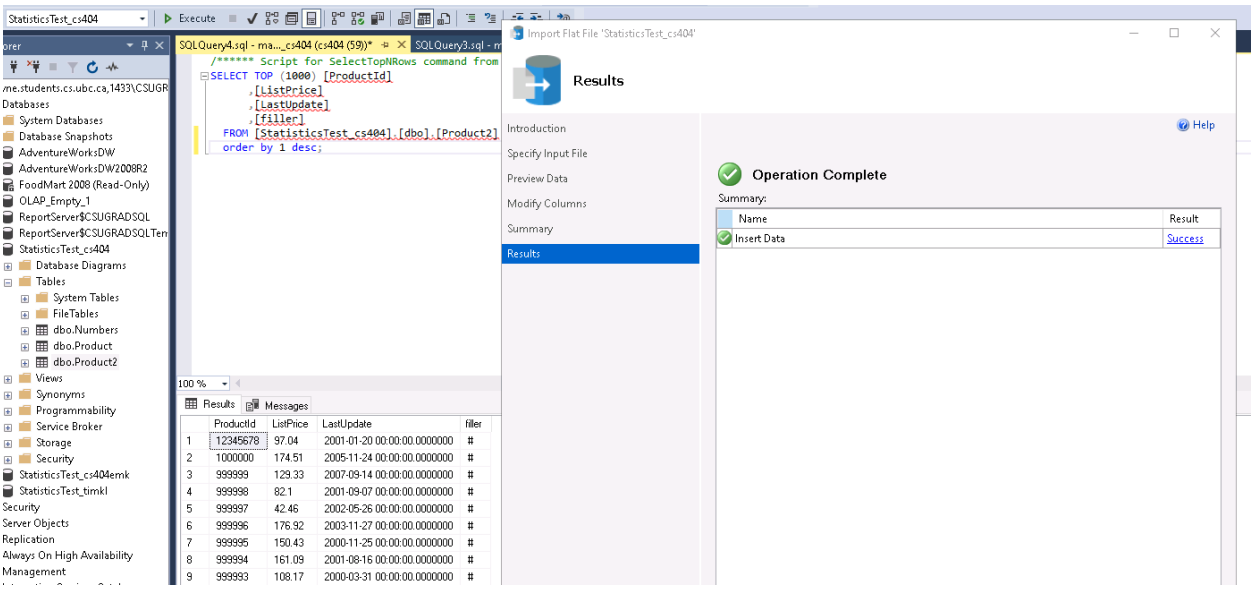
Before Importing:

- First, make a change to the large text file that you just exported.
    - Open the .txt file using a text editor like Notepad++ or Notepad.
    - Change the first `ProductId` number (i.e., ID #1) to your student number.
    - Save the file (using the same file name). Don't bother making a second copy (or even a backup of it) since the file is so large.

Finally, to Import:

- In Object Explorer, right-click on the database name
- Choose Tasks → Import Flat File…
- Next (to begin importing your data)
- Browse for the file location
- Specify a new table name: call it `Product2`
- Next
- Verify that the fields look good, and that your student number appears.
- Then, make sure that the "Use Rich Data Type Detection" is NOT checked. (This will prevent long floating point approximations in the monetary field.)
- Next
- Check the Primary Key box for `ProductId`
- Next
- Finish
- It will take about 2 minutes to create the `Product2` table, but don't close the window (that says Success) yet.
- **Hand In #11**: We will take a screen shot with the following details:

- Refresh your database within Object Explorer, so that the `Product2` table shows.
- Right-click on `Product2` to Select Top 1000 Rows.
- Modify the SQL to show 1,000 rows in *descending* order. Do this by including the "**order by 1 desc**" clause at the end of your SQL statement. This will allow us to see your student number in the `Product2` table.
- Show the success window (i.e., Operation Complete for your "Import Flat File"), so that it appears beside your results. Take a screenshot of all of these components together, so that you clearly show your database name (upper LHS), your student number in the SQL output, and now the Operation Complete success window after you imported the flat file.
- Here is an example of a single screenshot that shows all of these components. Your screenshot should be similar, albeit with your own database name and student number:



- Close the "Operation Complete" success window.

  Some closing notes (and you don't have to do what's described in this paragraph): The importing of data is not the same as a bulk load with the automatic rebuilding of indexes. The indexes have to be rebuilt separately. When you did your import, SQL Server automatically created an Alt. 1 clustering index on the PK, and if you re-run the 7 queries against the `Product2` table, you'll see that the access paths are suboptimal with many extra logical page reads compared to what you most recently experienced with the `Product` table.

- Finally, to save some space:
  - Delete the `Product.txt` file (since it's very large).
  - Delete the `Product2` table.

- Shrink the size of your database.

This is the end of the tutorial/exercise on SQL Server query evaluation and optimization.

Part 3: Database Backups and Restores (aka Recoveries)

This section is used by procedures described earlier in this SQL Server exercise.

In this part, you will learn how to backup and restore your database.

- Databases are frequently backed up by a DBA (or automated job)—sometimes several times per day.
- If a database gets corrupted due to a power failure, unrecoverable hardware error, human error or accident (e.g., using the wrong input file when inserting, updating or deleting records; or running a job twice and getting duplicate updates), then we can restore the database to a prior version.

Part 3A: Backing Up a Database: Creating an "Image Copy"

First, a note about permissions in the lab. Unlike most laptops and home machines, lab permissions are very tight. You cannot restore an offsite SQL Server database backup to a lab PC, and you cannot copy a server-controlled backup to a USB device for external use. In the lab, all the backups are stored on the server; you will not be able to export them. (The company you work for in the future may have its own restrictions on database backups and permissions. It almost certainly has procedures for off-site backups. DBMSs tend to have very tight security controls.)

- Here's an example of how to **backup** a database (including all of the objects and logs that the database contains). Unlike IBM's DB2, for example, there is no central, DBMS-wide log; instead, SQL Server maintains logs at the database level.
  - In Object Explorer in SQL Server Management Studio, expand Databases, and right-click on database `StatisticsTest_`*`userid`* → Tasks → Back Up…
    - We'll specify a full backup with the simple recovery model. This is probably the default on your machine (the lab machine may use a FULL recovery model—that's fine). In a moment, we will use the server's

`C:\Temp` local drive (note that this is not the same as your client PC's `C:\Temp` local drive). By the way, the server's E: drive is not your local PC's USB drive, either.

- For Destination, choose "Add…"
  - Choose "…" You might get a warning message; it is OK to ignore it.
  - For the file name, type a distinguishable name (and your own CWL *userid*) that other DBAs will recognize like `c:\Temp\StatisticsTest_`*`userid`*`_20231012_1423.bak` and note that it is good practice to include the date and time of the backup in the file name because many backups might be taken in a single day. Note that the above naming convention will keep your backup file names sorted within a file system. This will be handy later.
  - OK (but first, you might have to click Yes if asked about verifying the existence of the backup location—you might be asked this in the lab; so, just verify that your folder/file name is correct and click on Yes)
  - To avoid error messages, remove any other existing path/file name that's in the list by highlighting its name, and clicking on Remove. (This does not delete the old backup files that are stored on the server.)
  - The (empty) file that you want to use for the backup should be the only one that is left and is highlighted.
  - OK
    - Upon completion, you'll get the message, "`The backup of database 'StatisticsTest_`*`userid`*`' completed successfully.`"
    - OK

- Before recovering our database, let us put our backup and recovery exercise to a practical test.
  - While you're still in SQL Server Management Studio, go back to one of your tables in `StatisticsTest_`*`userid`*, and modify a value/row. You can easily do so by either issuing an `SQL UPDATE` query, or (if permitted) by right-clicking on the table that you want to update and choosing "Edit Top 200 Rows". Change some value, and remember what you changed. For example, change the value 1 to 888
  - 88888, and verify the change via an SQL statement.
  - Let's pretend that the update that you just did represented a series of errors, and that you want to restore the database (including all of its components) to its state as of the last full backup. Proceed to recover (restore) your database, as follows.

Part 3B: Restoring a Database

- Here's an example of how to **restore** a database. Again, we'll use `StatisticsTest_userid` as our example. Make sure you get out of the database first, since we can't restore an in-use database, especially if SQL Server thinks there are uncommitted transactions. One way to do this is to exit SQL Server Management Studio (don't worry about saving anything in SSMS), and re-launch SSMS. Then:
    - Without expanding the list of databases, right-click on Databases → Restore Database…
        - For Source:
            - For the database name to be restored, use the drop-down list to select your existing `StatisticsTest_userid` database.
                - The destination database will automatically be the same.
            - For the "Restore" location, we'll use the latest backup (the default).
            - Check the box beside the name of the backup to restore (the time of the backup file, if you scroll to the right, should be consistent with when you generated the backup). This is useful as a confirmation; but don't click OK, yet.
                - **Hand In #1**: Take a screenshot of the entire Restore Database pop-up window that shows the Start Date, Finish Date, Size, and User Name—you'll need to scroll to the right so that the marker will be able to see this information. This information is what you will see as you scroll to the right to look at the date and time of the database backup that you want to recover. Note that screenshots can be taken using Microsoft's "Windows Accessories" program called "Snipping Tool"—accessible via the list of programs in the start menu (lower LHS Windows icon (or just use the Print Screen key on your keyboard, etc.) Save them as a .jpg file that you can later paste into a Word file and/or convert to PDF.
            - To avoid an error message saying that the "tail of the log" of `StatisticsTest_userid` has not been backed up, and therefore the recovery failed, go to the Options panel, and check the "Overwrite the existing database (WITH REPLACE)" box, and click OK.
            - You'll see the progress wheel turning and giving you an indication of how much of the restoration process has completed. It should complete fairly quickly; but, if there are many objects or there is a large backup file, it can take a while. Upon completion, you should

get the message: "Database 'StatisticsTest_*userid*' restored successfully."

- OK

- To verify the recovery of the database: In Object Explorer, right-click on Databases → Refresh.
  - Expand StatisticsTest_*userid*.
  - Expand Tables (for your database).
  - Choose the table that you previously updated "in error" after you did your backup.
  - Confirm that the table is restored properly (and doesn't have the "erroneous" data still in it). To do so, and see that your 8888888 update is no longer in the table you updated, you may wish to run an SQL query specifying a descending order: ORDER BY 1 DESC;
  - This completes the recovery.