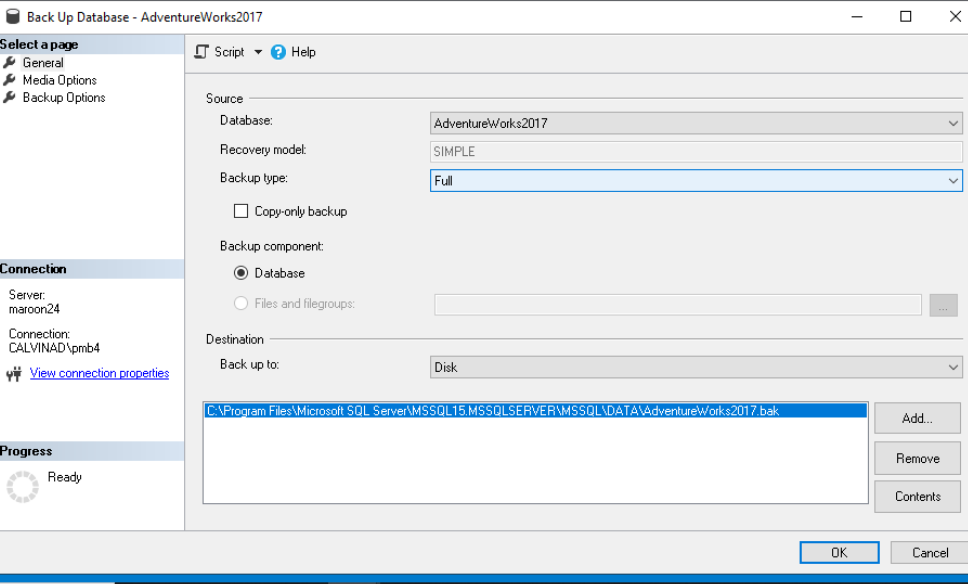
**Goal:** Analyze Performance Issues

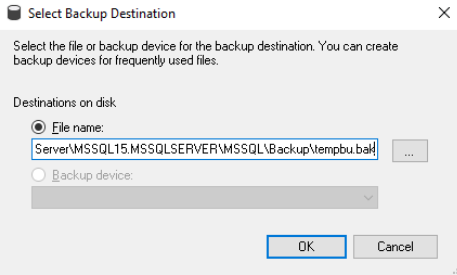
**Overview:** In this exercise using AdventureWorks2017, queries will be made in a before and after fashion to gain a sense of the impact on performance using indexes and different SQL statements. You will be asked for responses. Please place them into the Microsoft Word template document that’s provided on Moodle (do not save as an Open Office document, please!) You are encouraged to partner with someone. Each should have a workstation, one executes queries without the indexes and the other (on their separate work station) executes queries after the indexes are made. As partners, you may submit one document.

**Instructions:**

**Optional, but safe:** You may want to make a backup of the AdventureWorks2017 database in the event you need to refresh. (This applies to our computer lab. For those using their own laptop, just be ready to reuse the backup that was originally on Moodle). To do so, in Sql Server Management Studio, open the Databases folder in object explorer. Right click on AdventureWorks2017, and choose **Tasks->Back Up…** In the dialog box, make sure all the settings are similar to the illustration below. Then, select “Remove” and then select “Add” to establish a new backup file.



When prompted for the new file name, go to the end of the path in the textbox and append “tempbu.bak”



Click on OK. When back to the main dialog box click on OK again. You should get a message it was successful. Acknowledge it and continue.

**Setup for Homework:** You have been given the script increaseSH.sql on Moodle. Either open that file in SQL Server Management studio or copy and paste its contents into a query window. Take time to examine the script.

HOMEWORK RESPONSE 1: Explain what the script is doing. Consider the values for the COMMENT column. Also, consider what columns are left out, and explain why you think they were left out.

Run the script. Take time for a break. This will take a few minutes.

Open up a separate query window with AdventureWorks2017 as the database context.

HOMEWORK RESPONSE 2 Question: How many rows are now in the Sales.SalesOrderHeader table?

Enter the following:

SET STATISTICS TIME ON

At this point, there are sets of queries below along with a task to create an index and/or observe significant performance differences. When experimenting with a before/after index set, run each query at least five times before you run the index command and five times after you run the index command. For each, in the response document note the average length of time before and after indexes are added. Also, take time to observe the execution plan before and after indexes are added. After you do the before and after runs of the queries, comment on differences (or lack of) in the column provided:

**SET 1**

Query 1A: select CreditCardId, CurrencyRateId from sales.SalesOrderHeader

Query 1B:

select CreditCardId, CurrencyRateId from sales.SalesOrderHeader where CreditCardId = 8936

Query 1C:

select CreditCardId, CurrencyRateId, CustomerID, Comment from sales.SalesOrderHeader where CreditCardId = 8936

**Now add a composite index on the columns CreditCardId and CurrencyRateId in the SalesOrderHeader table. Name the index idx\_cccr. Rerun the queries above, and then observe the average time and what is noticed in the execution plan. Did the index help or not on each query? Explain why it may have or not.**

**SET 2**

Query 2A: select territoryid from sales.SalesOrderHeader where territoryid = 6

Query 2B: select comment, duedate from sales.SalesOrderHeader where TerritoryID = 6

Query 2C: select CreditCardId, CurrencyRateId from sales.SalesOrderHeader where TerritoryID = 6

Query 2D: select territoryId, count(\*) from sales.salesOrderHeader group by territoryId

**Now add an index that will use the territory ID in the sales.SalesOrderHeader table. Name the index idx\_cccti. Again, rerun the queries above, observe the average time and what is noticed in the execution plan. Did the index help or not on each query. Explain why it may have or not.**

**SET 3**

Query 3A: select comment from sales.SalesOrderHeader where comment like '%544%'

Query 3B: select comment from sales.SalesOrderHeader where comment like '544%'

**Now add an index that will use the comment column in the sales.SalesOrderHeader table. . Name the index idx\_cmt. Again, rerun the queries above, and then observe the average time and what is noticed in the execution plan. Did the index help or not on each query? Explain why it may have or not.**

**SET 4: Examine the two queries below. Run each individually three times and record their**

**times. Which ran faster? If one is faster than the other, take time to review the execution**

**plan. Then, explain what you observe and how either query could be improved.**

Query 4A: select \* from sales.SalesOrderDetail d

inner join sales.SalesOrderHeader h on d.SalesOrderID = h.SalesOrderID

Query 4B: select d.SalesOrderID ,d.productid ,p.name from sales.SalesOrderDetail d

inner join production.Product p on d.ProductID = p.ProductID

**Set 5 This query is the one used for the market basket. Take time to examine its execution plan and times. Can it be improved? What index does it rely on for performance? What join algorithm was used, and why do you think that was?**

SELECT b.baseId "Base Product",

b.pCount as "# Orders With Base",

s.AffId "Accompanied By Product",

s.Frequency "# Accompanied w/Base",

FORMAT(CAST (s.Frequency as Decimal)

/CAST(b.pCount as decimal), 'P') "Affinity %"

FROM

(SELECT p.productid as baseId, --this is essentially a grouping

t.productid as affId, --on cartesian products within

count(\*) as Frequency --each order

FROM Sales.SalesOrderDetail p join

Sales.SalesOrderDetail t

ON p.SalesOrderID = t.SalesOrderID

WHERE p.ProductID <> t.ProductID

GROUP BY p.ProductID, t.ProductID

HAVING count(\*) > 1 ) as s --The Having is to filter out the trivial

INNER JOIN

(SELECT productid baseId, COUNT(\*) as pCount

FROM sales.SalesOrderDetail

GROUP by productid

HAVING COUNT(\*) >= 10) as b

ON b.baseId = s.baseId

ORDER BY 5 DESC