

LAB SCRIPTS

Fill in your lab environment details

|  |  |
| --- | --- |
| Your VM name: |  |
| VM IP: |  |
| User: |  |
| Password: |  |
| Lab folder: |  |



**LAB #1**

**SQL Server Architecture Overview and Tools**

**Exercise #1. Functions**

**TO DO LIST**

1. Open SSMS and connect using Windows Authentication to the default SQL Server Instance.
2. Explore BuiltIn functions using the examples in script **C:\SQLMDEV\_LabFiles\M1\BuiltIn\_Functions.sql** in the lab folder
3. Write down is there something new you didn’t know about functions.

|  |
| --- |
|  |
|  |

**Exercise #2. Working with NULLs**

**TO DO LIST**

1. Explore the examples of ISNULL and COALESE in the script **C:\SQLMDEV\_LabFiles\M1\ISNULL\_COALESCE.sql**.
2. Explain the main differences and when you are going to use what function
3. Explain why NOT IN queries will work differently when there are NULLs in the subquery and how to avoid such cases

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #3. Write a good code**

**TO DO LIST**

1. Correct and optimize the code in the script **C:\SQLMDEV\_LabFiles\M1\Write\_a\_good\_code.sql**
2. Explain what rules of good coding did you follow

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



**LAB #2A**

**Data and Index structures**

**Exercise #1. Getting Index metadata info**

**TO DO LIST**

1. Review and execute the statements in the **C:\SQLMDEV\_LabFiles\M2\Data Structures\*Index metadata info.sql*** (Use database AdventureWorks2016CTP3).
2. Explain the index information you consider important to have and will use in your work
3. Why the field choice for the clustered index is so important?

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #2. Optimizing queries with indexes**

**TO DO LIST**

1. There are queries listed in QueriesForOptimizations.sql script. Open, review and try to optimize queries by either make some slught changes in the code and/or by adding indexes**.**
2. List the 3 most important principles you followed for optimizing the listed queries

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #3. SQL Server CSI**

**TO DO LIST**

1. Open and execute the scripts **C:\SQLMDEV\_LabFiles\M2\CSI\Clustered Columnstore Sample Queries.sql**. Make a summary note of important things to remember for CSI

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #**4**. Dealing with increasing the identity type (optional)**

**TO DO LIST**

In this exercise, you will see an approach that you can take if you absolutely need to move to bigint, need to minimize downtime, and have plenty of time for planning.

At a very high level, the approach is to create a set of shadow tables, where all the inserts are directed to a new copy of the table (with the larger data type), and the existence of the two sets of tables is as transparent as possible to the application and its users.

At a more granular level, the set of steps would be as follows:

1. Create shadow copies of the tables, with the right data types.
2. Alter the stored procedures (or ad hoc code) to use bigint for parameters.
3. Rename the old tables, and create views with those names that union the old and new tables.
   1. Those views will have instead of triggers to properly direct DML operations to the appropriate table(s), so that data can still be modified during the migration.
   2. This also requires SCHEMABINDING to be dropped from any indexed views, existing views to have unions between new and old tables, and procedures relying on SCOPE\_IDENTITY() to be modified.
4. Migrate the old data to the new tables in chunks.
5. Clean up, consisting of:
   1. Dropping the temporary views (which will drop the INSTEAD OF triggers).
   2. Renaming the new tables back to the original names.
   3. Fixing the stored procedures to revert to SCOPE\_IDENTITY().
   4. Dropping the old, now-empty tables.
   5. Putting SCHEMABINDING back on indexed views and re-creating clustered indexes.

Open and execute the scripts **C:\SQLMDEV\_LabFiles\M2\IDENTITY column.sql**. Make a summary note of important things to remember.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



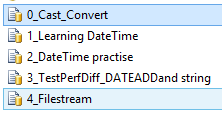
**LAB #2B**

**Diving into data types**

**Exercise #1. Exploring special data types**

**TO DO LIST**

1. Open, review and execute the scripts in their order. Some of the scripts contain tasks descriptions in comments that you have to complete.



1. Write down the important thinks that you learned for working with date and time data types

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #2. Exploring JSON (If time permits)**

**TO DO LIST**

1. You have example in **C:\SQLMDEV\_LabFiles\M2\JSON\_if\_Time\_Permits\JSON Sample Queries procedures views and indexes.sql**. Open and review it in order to learn how you can use JSON. You can start from 0\_JSON.sql and 1\_JSON.sql if you want and intro first.
2. Describe your main findings

|  |
| --- |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



**LAB #3**

**Complex T-SQL statements**

**Exercise #1. TVP**

**TO DO LIST**

1. Open, review and execute the **C:\SQLMDEV\_LabFiles\M3\1\_Table Valued Parameters.sql** script.
2. Describe the main advantages of the TVP

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #2. Windowing Functions**

**TO DO LIST**

Create a database TSQL2012 if there is no such. Use script in the lab folder **C:\SQLMDEV\_LabFiles\M3\TSQL2012\_dbscript.sql**

1. **Gaps and Islands case**

Gaps and Islands are classic problems in SQL that manifest themselves in practice in many forms.

The basic concept is that you have some sort of sequence of numbers or date and time values where

there’s supposed to be a fixed interval between the entries, but some entries could be missing. Then

the gaps problem involves identifying all ranges of missing values in the sequence, and the islands

problem involves identifying all ranges of existing values.

Review the provided solution in the **C:\SQLMDEV\_LabFiles\M3\WindowingF\_Solutions.sql** script and examples and explain the alternatives as well as which should be chosen when.

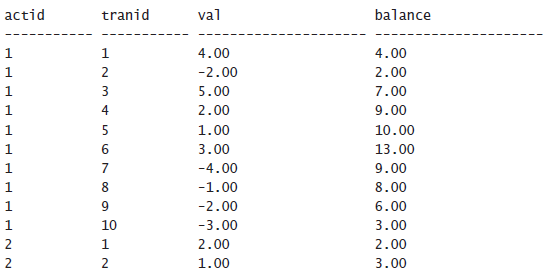
|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

1. Compare **running totals** solutions and describe the difference in approaches

Calculating running totals is a very common need. The basic idea is to keep accumulating the values in one attribute (the aggregated element) based on ordering defined by another attribute or attributes (the ordering element), possibly within partitions of rows defined by yet another attribute or attributes (the partitioning element). There are many examples in life for calculating running totals, including calculating bank account balances, tracking product stock levels in a warehouse, tracking cumulative sales values, and so on.

Each row in the Transactions table represents a transaction in some bank account. When the transaction is a deposit, the amount in the val column is positive; when it’s a withdrawal, the amount is negative.

Your task is to compute the account balance at each point by accumulating the amounts in the val column based on ordering defined by the tranid column, within each account independently. The desired results should look like this for the small set of sample data:



To test the performance of the solutions, you need a larger set of sample data. You can use the

Code in the **C:\SQLMDEV\_LabFiles\M3\WindowingF\_Solutions.sql** lab file

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

1. Implement a **TOP N per group** solution

The Top-N-per-Group problem is a common querying problem that involves filtering a requested number of rows from each group, or partition, of rows, based on some defined ordering. A request to query the Sales.Orders (AdventureWork\_XXXX database) table and return, for each customer, the three most recent orders is an example for the Top-N-per-Group problem.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

1. Sort Hierarchies **(if time permits)**

Suppose that you need to present information from some hierarchy in a sorted fashion. You’re supposed to present a parent before its child elements. Also, you need to be able to control the order among siblings.

For sample data, use the code from the **C:\SQLMDEV\_LabFiles\M3\2\_WindowingF\_Solutions.sql** script file, which creates and populates a table called **dbo.Employees**. You need to present employees in hierarchical order—always presenting the manager before subordinates—and sort siblings by empname. To achieve this task, you can use two main tools: the ROW\_NUMBER function and a recursive CTE.

Write down both ways of performing sorting hierarchies

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #3. Dynamic String Execution**

**TO DO LIST**

1. You need to implement a script which executes the following select statement but with parameterized database and table names

-- This is what we want parameterized

SELECT \* FROM @DBName.dbo.@TableName

How do you implement this task?

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

1. You can review the **C:\SQLMDEV\_LabFiles\M4\Dynamic String Execution.sql** script in the folder for an idea of a solution
2. Security problem of the DSE. Review the three procedures in the **C:\SQLMDEV\_LabFiles\M4\DynamicStringExecution SecurityProblems.sql** script. They demonstrate how dynamic string execution can be used securely and how - if not prepared - users can creatively delete/destroy data.

What is the way to securely use DSE?

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



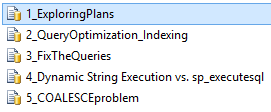
**LAB #4**

**Query Execution and Query Plans**

**Exercise #1. Exploring plan and optimizing queries**

**TO DO LIST**

1. Open, review and execute the scripts in their order. Some of them requires you to optimize queries.



Describe the main problems in query plan that you have encountered and how did you perform optimizations

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



**LAB #5**

**T-SQL Programming objects**

**Exercise #1. Optimize the procedures**

**TO DO LIST**

1. You saw stored procedures which have inconsistent execution. They are listed in **C:\SQLMDEV\_LabFiles\M5\CodeTemplates.sql**. Optimize the code in order to get it predictable. Explain what you did for fixing the problem.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #2. Write a good code**

**TO DO LIST**

1. The script file **C:\SQLMDEV\_LabFiles\M5\CodeFixing.sql** contains code that definitely needs optimization or re-writing. Implement changes and describe what rules you have followed during the process.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #3. Temporal**

**TO DO LIST**

1. Execute script **C:\ SQLMDEV\_LabFiles\M6\Temporal\AW 2016 CTP3 Temporal Setup.sql**
2. Open script **C:\ SQLMDEV\_LabFiles\M6\Temporal\Temporal System-Versioning Sample.sql** andfollow the instructions in it.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #4. STRING SPLIT – If Time permits**

**TO DO LIST**

1. What’s wrong with that **C:\SQLMDEV\_LabFiles\M6\Stored Procedure to Sort Items in a Database Table.sql**. Rewrite the code using better programming objects.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



**LAB #7**

**Transactions and lock management**

**Exercise #1. Writing transactions**

**TO DO LIST**

1. Write a transaction that inserts an order for selling a product with specific ProductID in Northwind database and updates the inventory of the product. Tables that you have to work with are Orders, OrderDetails, Product.
2. Enclose the transaction in a stored procedure ProductSale.
3. Check the parameter values in the stored procedure according to best practices. Take into account that you cannot sale a product if the inventory level is < product quantity requested
4. Extend the solution in order to cover selling >1 product with one order.
5. Include error handling for the transaction and test the transaction. Return common error info.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**Exercise #2. Error handling in stored procedure**

**TO DO LIST**

1. Create a stored procedure that performs the following:
   1. Modifies Vendor data (CreditRating and PreferredVendorStatus, table Purchasing.Vendor in AdventureWorks database) based on VendorID insput procedure parameter
   2. Perform error checking for the input parameters and the update statement.

|  |
| --- |
|  |
|  |

**Exercise #3. Write a good code**

**TO DO LIST**

1. Open file **C:\SQLMDEV\_LabFiles\M6\CodeFixing.sql** in the respective folder. Review and correct the code according to the good coding practice and requirements.
2. Wire down the good and bad coding practices that you applied in the file

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



**LAB #7**

**InMemory OLTP**

**Exercise #1. InMemory OLTP**

**TO DO LIST**

Your task is to optimize a solution for data loading and data manipulating that your company uses. Write down important results and considerations

1. Open, review and perform the instructions in **C:\SQLMDEV\_LabFiles\M8\InMemoryOLTP\1\_BasicInMemoryOLTP.sql** script**.** Run the **C:\SQLMDEV\_LabFiles\M8\InMemoryOLTP\0\_InMemory\_DBandObjects** to create the database and its objects first**.**
2. Write down test results and final decisions that you take during the tests
3. Delete the database InMemoryTest after completing the exercise to release resources for the next one

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Exercise #2. InMemory Usage Scenarios**

**TO DO LIST**

1. Open, review and perform the instructions in **C:\SQLMDEV\_LabFiles\M7\InMemoryOLTP\_UsageEsamnples** script**.**
2. Write down the most important thinks to consider for InMemory OLTP tables and indexes

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



**LAB #8**

**Data Loading for Performance**

**Exercise #1. Partitioning**

**TO DO LIST**

1. Open and review the commands in the script **C:\SQLMDEV\_LabFiles\M8\Table-Partitioning-Tutorial\Table Partitioning Tutorial.sql**
2. As part of the script perform loading of data and deleting of data using partition switching

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |