OPTIMIZING SQL QUERIES

UNITED WHOLESALE MORTGAGE

TABLE OF CONTENTS

UNITED WHOLESALE MORTGAGE

OPTIMIZING SQL QUERIES

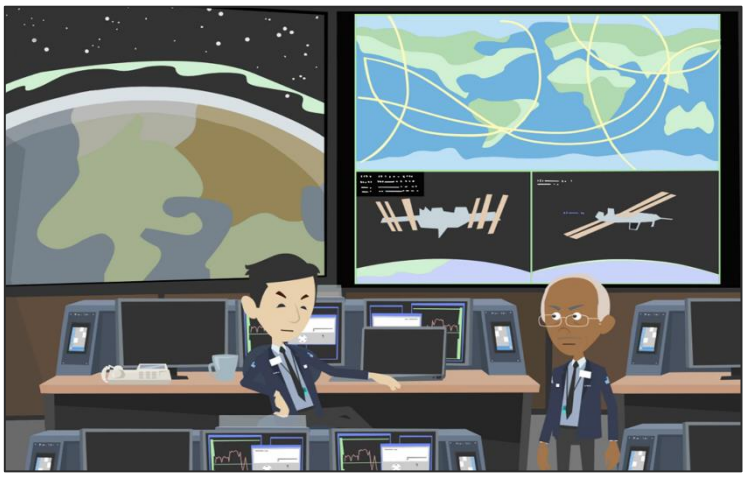
OVERVIEW

As a Developer who often writes SQL code, continuously learning to optimize your SQL queries is crucial when working in the Shore 1 Server. This hands-on training will focus on the use of a query optimization tool within SQL Server Management Studio (SSMS), the use of indexes, and the use of JOINs with 5 or more tables. Learning to optimize your SQL queries results in faster running queries, which is a quicker return on data. Additionally, it will reduce server slowness and potential risk of the server crashing.

MISSIONS TO COMPLETE

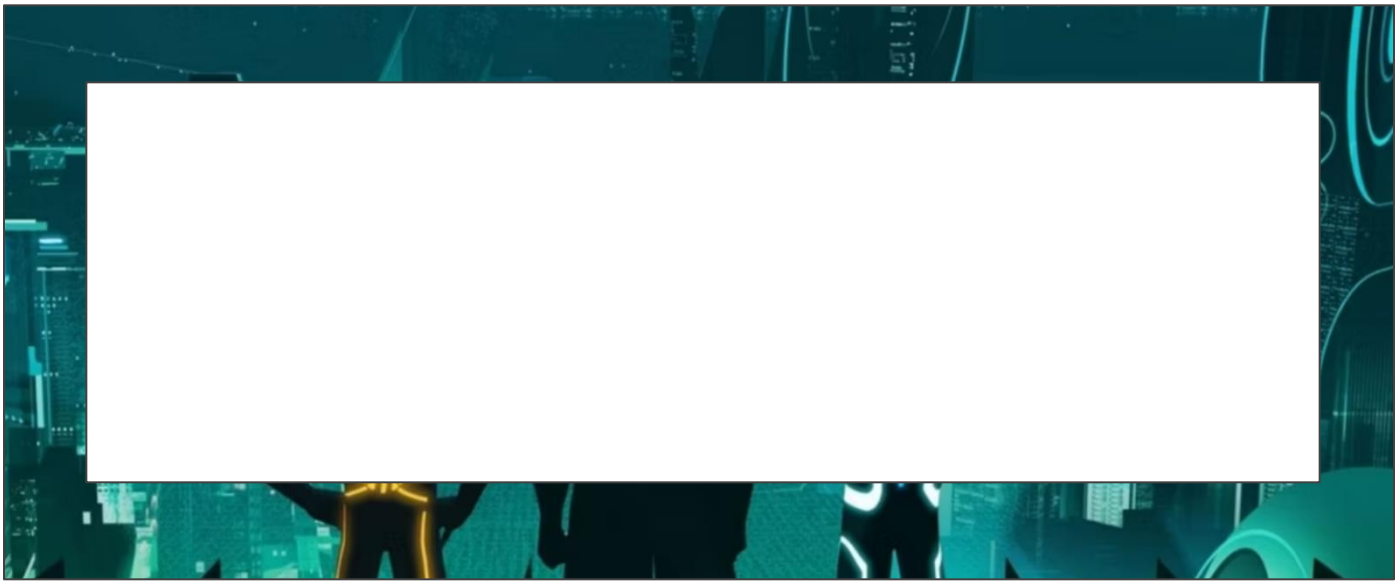
Accomplish these three missions to save the city (Shore 1 Server) from ticking time bombs (inefficient queries)!

- Practice generating and interpreting execution plans for several SQL queries.
- Practice optimizing several SQL queries using indexes.
- Practice optimizing several SQL queries through proficient use of JOINs with 5 or more tables at a time.



EXERCISE | TRAFFIC ANALOGY

After watching the second video shown by your trainer, use the space provided to notate anything you see that is causing the slowness and traffic jams, as well as ways that it can be avoided.



MISSION TOOL

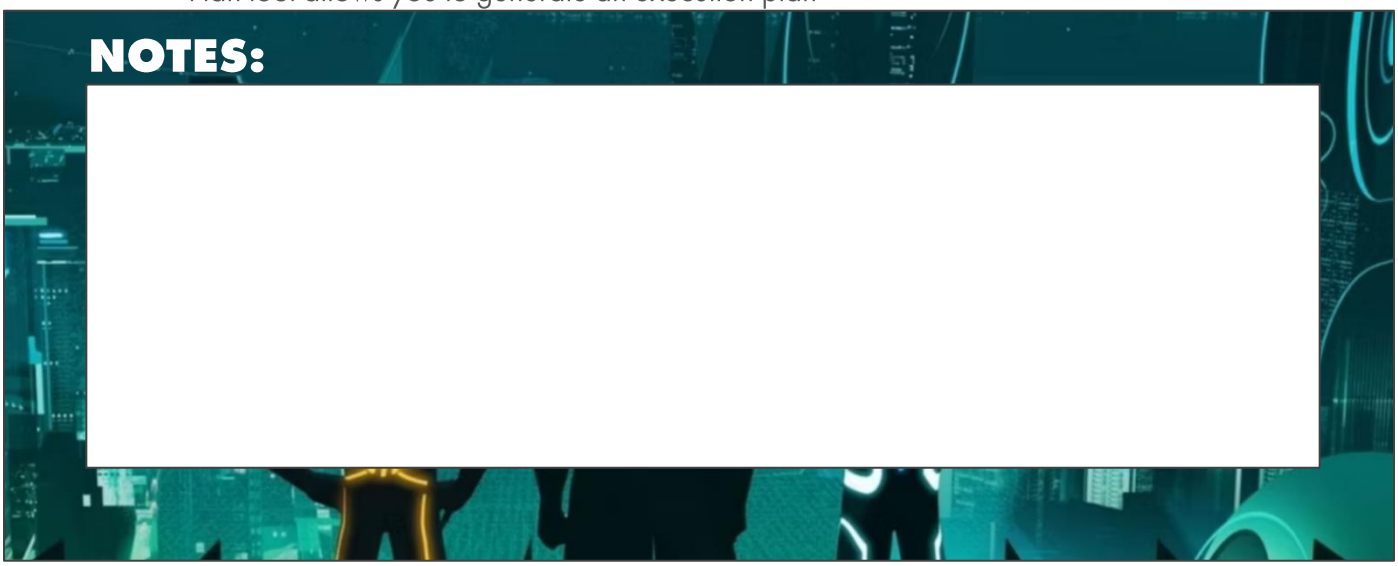
The SSMS has a tool that is crucial to understanding and optimizing SQL queries – Execution Plan tool. When enabled, this tool allows you to generate an execution plan for your query, and the plan shows the overall performance of your query. It also breaks your query down to point out areas where your query could be more efficient, so that you can modify your query based on the findings. This tool is required to be used with all your written queries.

EXECUTION PLAN TOOL

The Execution
for a written query. The execution plan gives you a visual representation or route of the steps that SQL Server takes to execute your query, as well as key information about your query, like estimated cost and amount of processed data.

- "GPS Navigation System" for query.
- Generates an execution plan.
- Outlines key information about the query.
    - Cost of operations.
    - Amount of data processed.

Plan tool allows you to generate an execution plan

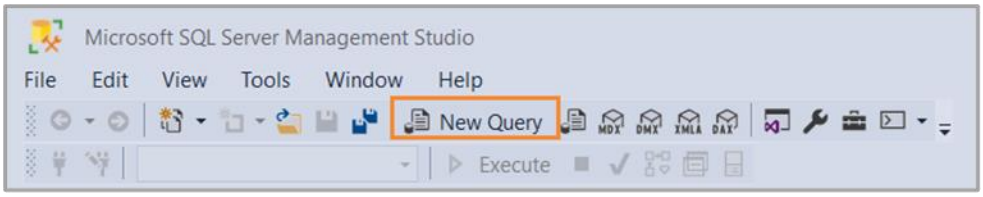


CLASS EXERCISE | GENERATING AND INTERPRETING

Once the SQL query is written, you can use the Execution Plan tool to generate an execution plan for that query by turning the tool on and then executing your query. The guidance below shows you how to enable this tool and how to read the execution plan.

1. Open the SSMS and connect to the following server: **S01VDSHR01AGL**.

2. Click **New Query**.



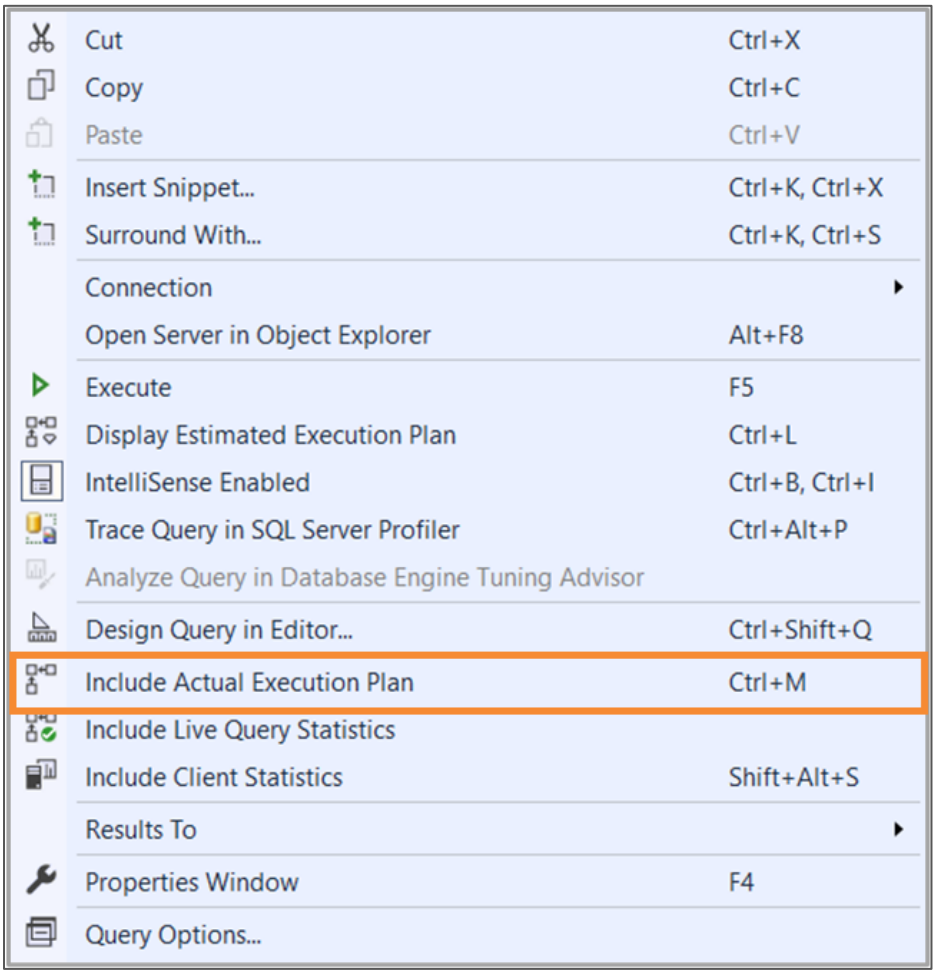
3. Copy and paste the following SQL query into the new query window.

```
SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT TOP 1000 * from  [dbo].[loan_conditions]
WHERE loanrecordid = 21534067
ORDER BY categoryid

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

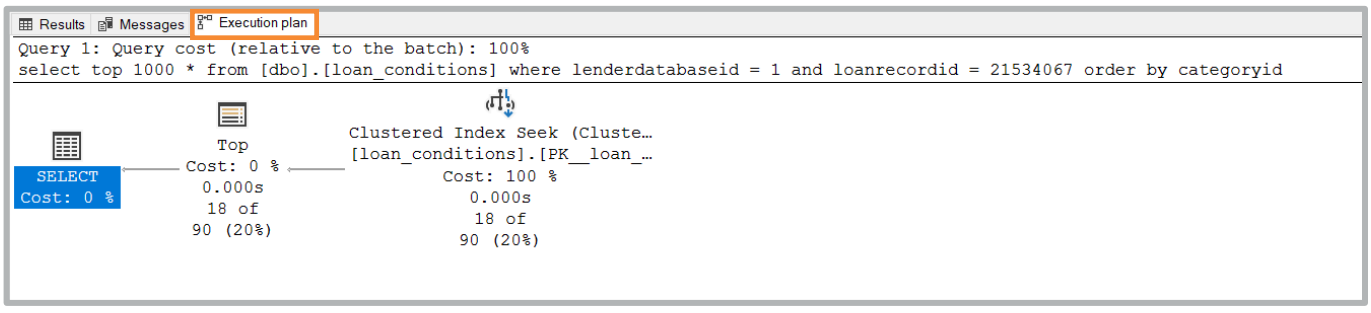
NOTE: Including `SET STATISTICS IO/TIME` in your query makes it so that you can monitor the CPU time and number of Logical Reads when you're query runs. These two metrics can be found within the "Messages" tab.

4. Enable the tool in the SSMS.

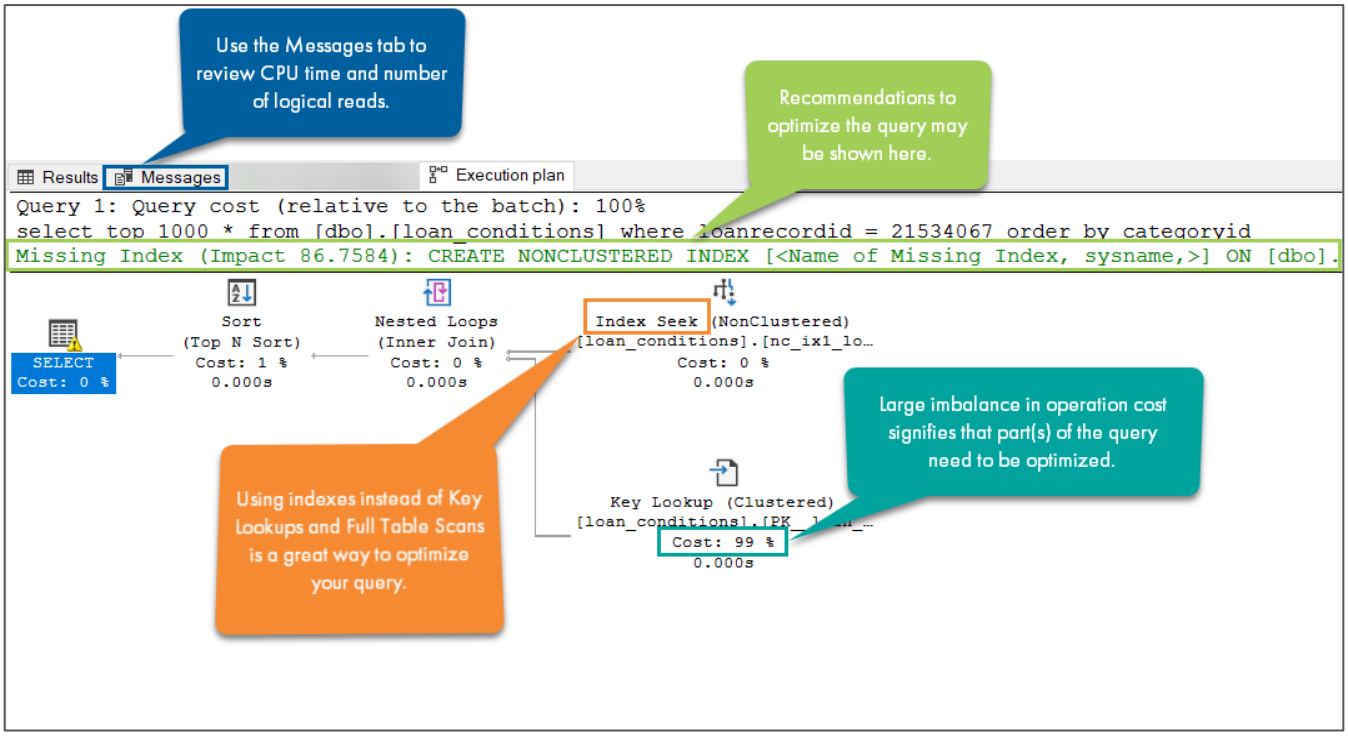
    - To enable the Execution Plan tool, right-click anywhere in the query window and select **Include Actual Execution Plan**.



NOTE: Once you have enabled the Execution Plan tool, execute your written query. A new tab for **Execution plan** will display in the bottom window.



EXAMPLE: When reviewing an execution plan, focus on these key areas, including number of rows being processed.



Key Lookup (Clustered)

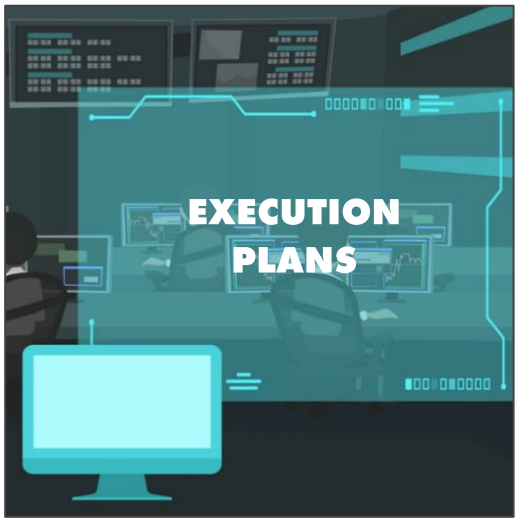
Uses a supplied clustering key to lookup on a table that has a clustered index.

Physical Operation	Key Lookup
Logical Operation	Key Lookup
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	18
Actual Number of Rows Read	18
Actual Number of Batches	0
Estimated Operator Cost	3.77351 (99%)

This number should make sense with what the query is asking for.

INDEPENDENT EXERCISE

Copy and paste the following SQL query into the SSMS. Generate and interpret the execution plan, focusing on the operator types (i.e., Index Seek, Key Lookup), cost, CPU time, and logical reads. Use the space provided to summarize the differences between this query and the previous query.




```
SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT TOP 1000 * FROM  [dbo].[loan_conditions]
WHERE lenderdatabaseid = 1 AND loanrecordid = 21534067
ORDER BY categoryid

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

DIFFERENCES:

- Second query (pp. 9) uses significantly fewer logical read (6 vs 85)

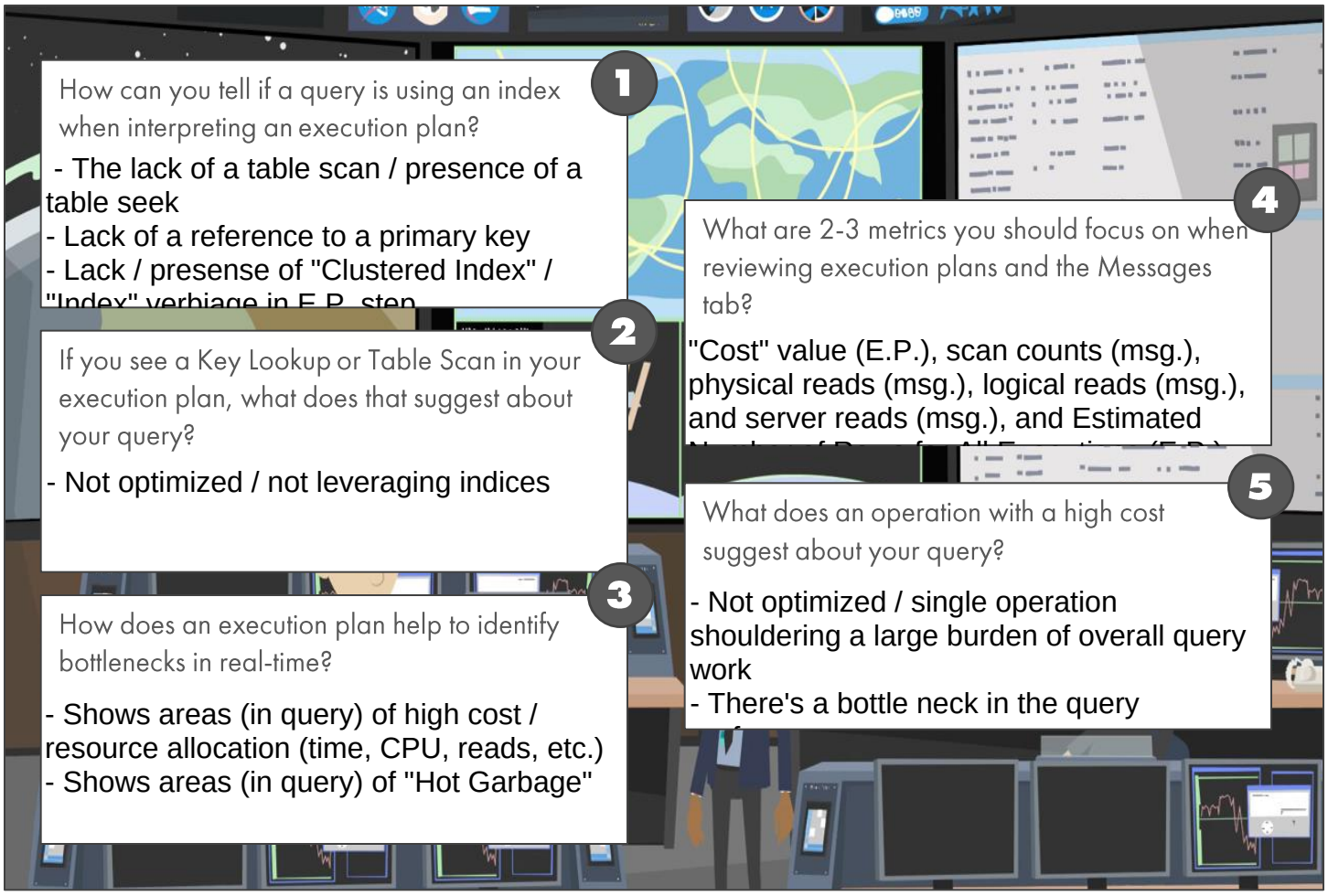
- Fewer actual steps (3 vs 5)

- Second query (pp. 9) uses CLUSTERED INDEX SEEK instead of Index Seek AND a Key lookup (as pp. 6 query is doing)

- Generally speaking, table scans are very bad and table seeks are (to some degree) better

MISSION 1 | REVISIT

Complete the following questions related to execution plans to allow yourself to critically think about what we just learned.



UNITED WHOLESALE MORTGAGE

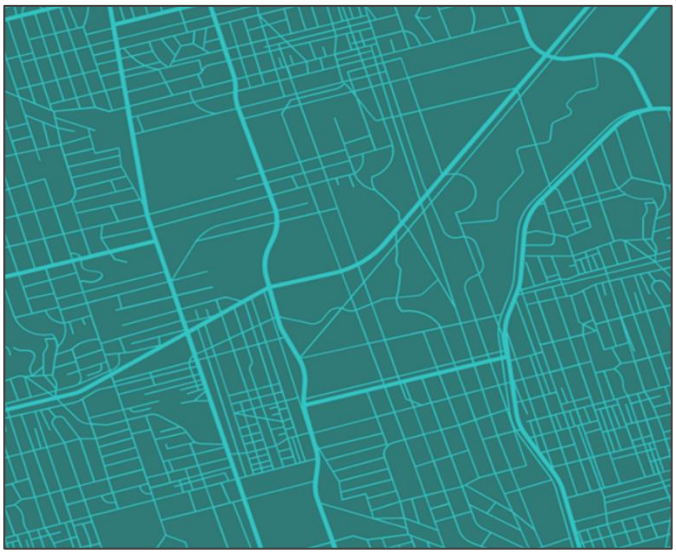
MISSION MAP

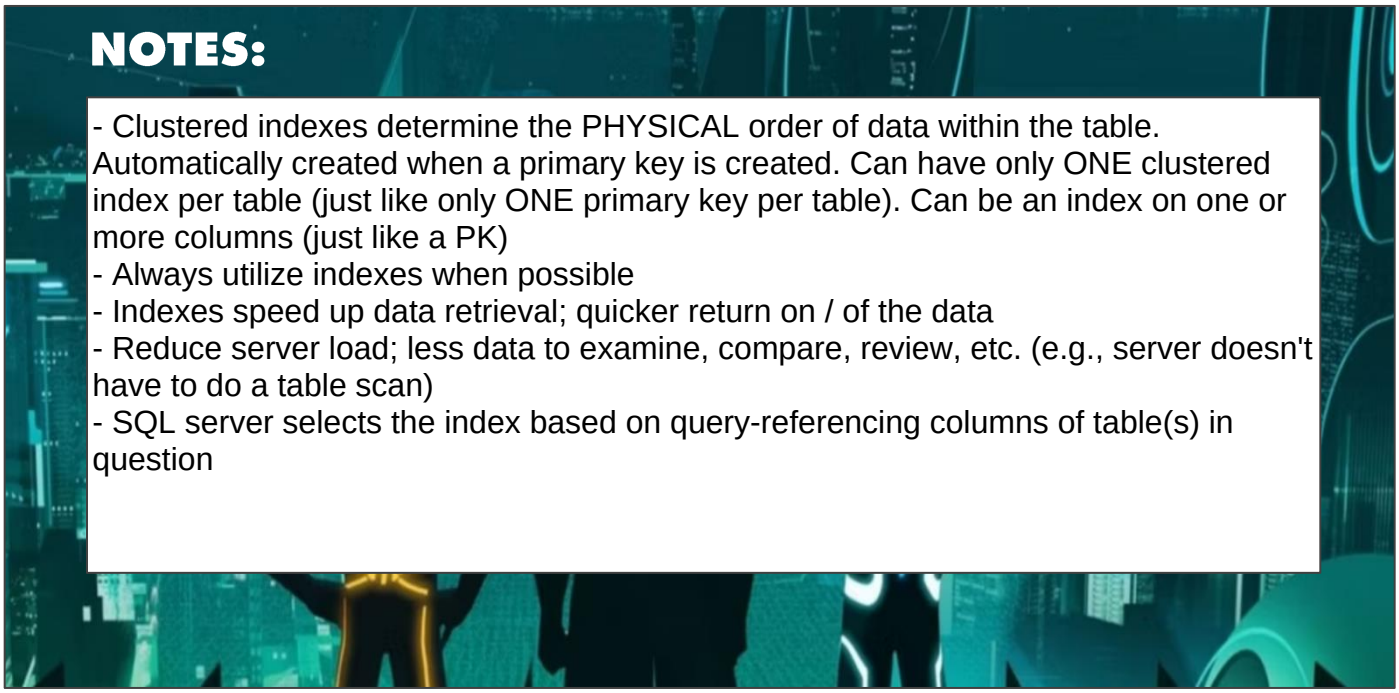
Similarly to a detailed map of the "Shore 1 Server" city, an index is a way to quickly to find and return data from a table without the server having to sort through every row in the table every time a database table is accessed. Indexes should always be reviewed prior to optimizing your SQL query to identify what indexes are available and what columns of the table they cover. It is not always possible to use an index, but you must always try.

INDEXES

Indexes speed up the return on data and it's a cheaper operator to use compared to Table Scans and Key Lookups.

- Always utilize indexes, when possible.
- Speeds up data retrieval, quicker return on data.
- Reduce server load, less data to examine.
- SQL server selects index, based on written query.

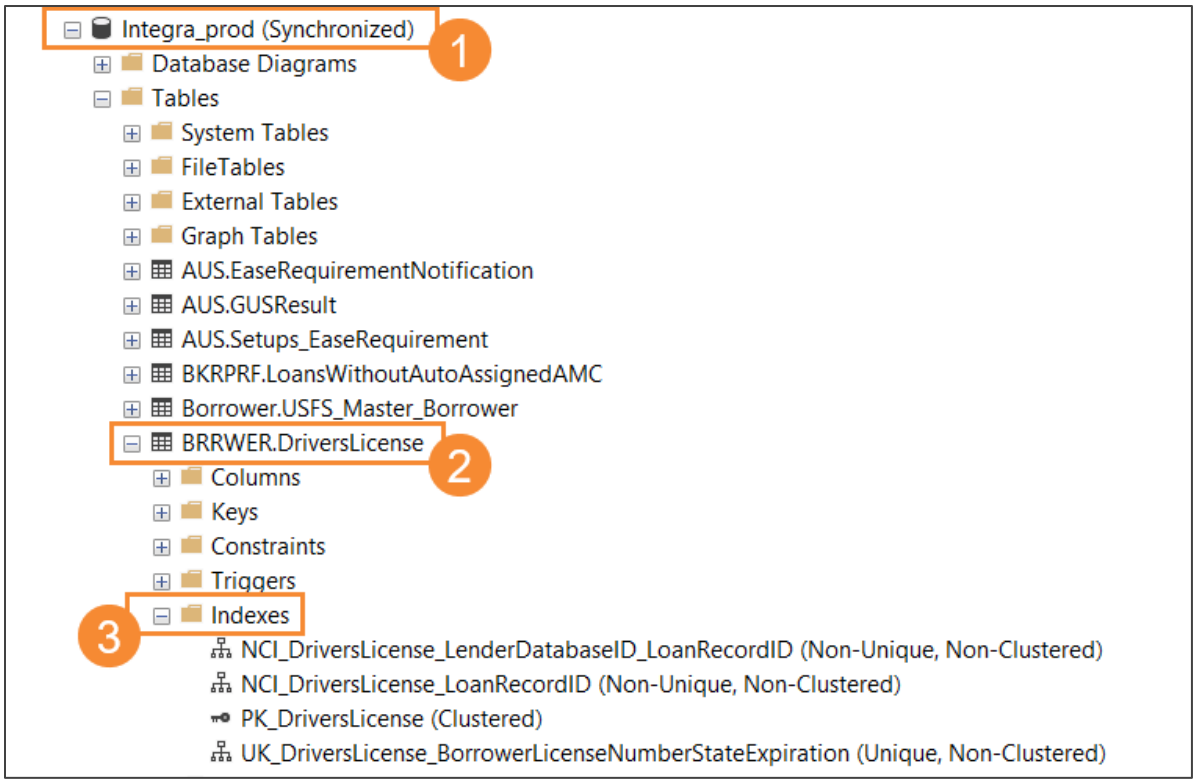




LOCATING INDEXES

To be able to optimize your queries so the SQL server likely will utilize an index, we must know where to find the index for a table and what columns of the table that the index covers. Follow the guidance below to do this.

1. Click on the necessary database **(i.e., Integra_prod (Synchronized)**.
2. Click on the necessary table you're using (i.e., **BRRWER.DriversLicense**).
3. Click on the **Indexes folder** to expand the available index list.



NOTE: Not all tables have indexes created. If one doesn't exist and you think the floor could benefit from having one, reach out to the DBA and DEA teams to create one, but always include your Team Lead in the communication.

CLUSTERED VS. NON-CLUSTERED

There are clustered and non-clustered indexes. Clustered indexes are faster than non-clustered indexes. See the table below for the differences.

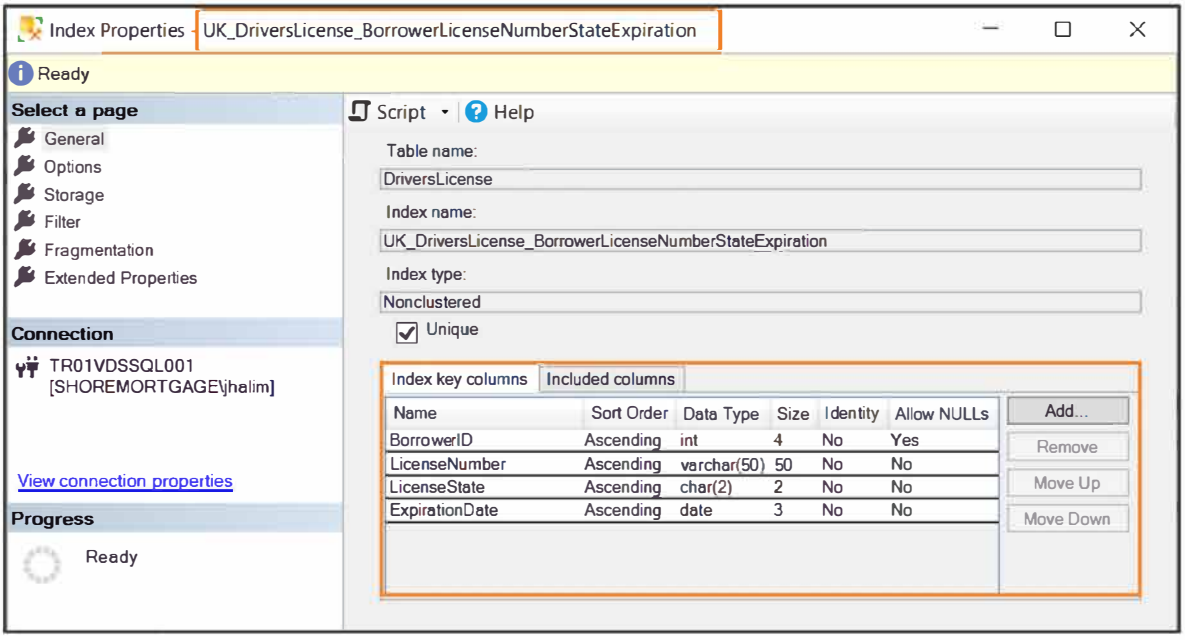
CLUSTERED VS NON-CLUSTERED INDEXES

FEATURE	CLUSTERED	NON-CLUSTERED
Physical Order	Affects physical order of data in table.	Physical data unchanged in table.
Number per Table	One	Multiple
Data Retrieval	Faster	Slower
Best For	Range queries	Specific column queries

INDEXED COLUMNS

Knowing which **key columns**, or "keys", are covered by an index improves your query planning. You cannot modify your query in a way that the SQL server is likely to use that index unless you know what columns are indexed. Double-click on a specific index to see what columns are covered.

NOTE: For the SQL Server to utilize an index, include all indexed columns or minimally, the top column(s) in your query criteria. For instance, if using the below index, incorporate all four columns or at least the BorrowerID. The BorrowerID can't be skipped as it's the top listed column.



UWM
UNITED WHOLESALE MORTGAGE

CLASS EXERCISE | UTILIZING INDEXES

We talked about the importance of modifying queries in a way that they use available indexes to increase query optimization. Let's look at the difference in efficiency between using indexes and using Table Scans and Key Lookups. Follow along with the trainer as you generate an execution plan on two SQL queries shown below. As a class, make note of the differences when using an index vs. not using an index.

```
--De-optimized:
SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT *
FROM dbo.USFS_OrgChart uoc WITH(NOLOCK)
WHERE uoc.userid = 22316

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

```
--Optimized:
SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT o.FullName
      ,o.JobTitle
      ,o.manager_FullName
FROM dbo.USFS_OrgChart o WITH(NOLOCK)
WHERE o.domainname = 'jhalim'
      AND o.userid = 22316

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

DIFFERENCES:

- Optimized query is leveraging a Cluster index (table PK of "domainname")

INDEPENDENT EXERCISE

Generate and interpret an execution plan for the following query, then optimize the query. Remember to review the available indexes prior to optimizing the query.


```
--de-optimized:

SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT re.[name]
      ,re.alias2
      ,re.entityid
FROM dbo.rolodex_entity re WITH(NOLOCK)
WHERE re.entityid = 89;

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

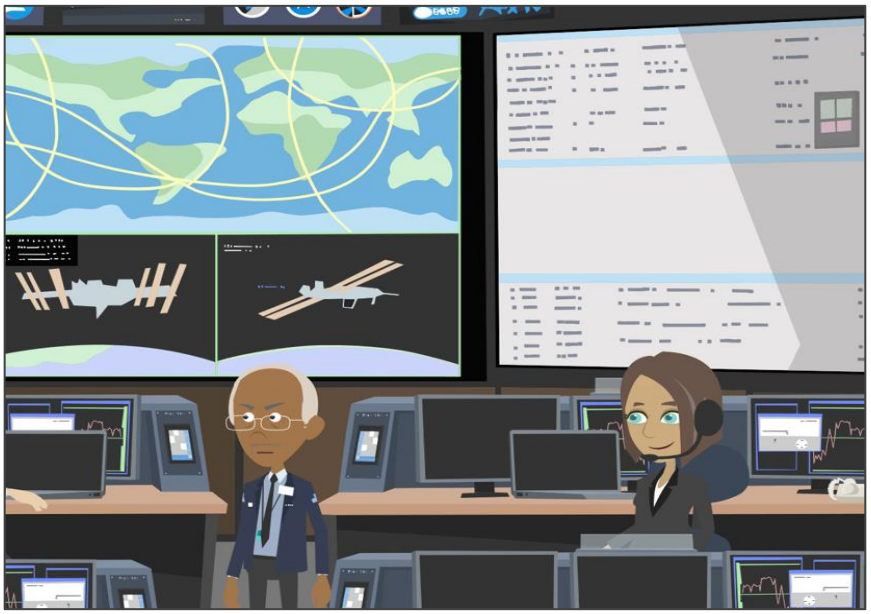
OPTIMIZED QUERY:

Change WHERE clause to:

```
WHERE re.lenderdatabaseid = 1
 AND re.entityid = 89;
```

MISSION 2 | REVISIT

Think about what you've learned and your biggest challenge(s) when using indexes. Using the space provided, write down what your biggest challenge will be with using indexes when you go back to your desk and why.



MISSION BRIDGE

JOINs act as the bridges in Shore 1 Server city. Bridges in the city connect us to different areas, allowing us to move from one side of the city to the other more quickly to diffuse the ticking time bombs! JOINs serve a similar purpose in SQL by allowing us to combine data from separate tables to be accessed together, making data retrieval faster and more efficient on the server.

JOINS

Below are key points to keep in mind when JOINing five or more tables.

- Order by smallest, most filtered result sets first.
    - This is not always most optimal.
- Concept of JOIN type remains the same.
- Use INNER JOIN when possible.
- JOIN on indexed columns. (Order matters).

NOTE: JOINs used in conjunction with a Function must be reviewed by sr. team member or team lead.

EXAMPLE: Order JOINs by smallest, most filtered result sets first. The below example shows this by starting with selecting all the borrowers with a credit score over 700 first. Then, it JOINs the smaller result set of borrowers to the Loans table (since it has the least number of rows), and then the Payments table to the smaller set of loans. This way, the server is filtering first and then JOINing smaller sets of data.

Tables | Find Credit Scores Above 700
Loans | 1,000 Rows
Borrowers | 10,000 Rows
Payments | 100,000 Rows

```
SELECT *
FROM Borrowers
JOIN Loans ON Borrowers.borrower_id = Loans.borrower_id
JOIN Payments ON Loans.loan_id = Payments.loan_id
WHERE Borrowers.credit_score > 700;
```

UNITED WHOLESALE MORTGAGE

JOINING AT LEAST FIVE TABLES

We talked about the importance of JOINing on indexed key columns, ensuring the key columns are listed in the order in which they appear in the indexes, and ensuring not to use any tables that are not necessary. Let's get practice as a class with identifying these things in a query and modifying the query to be more optimized.

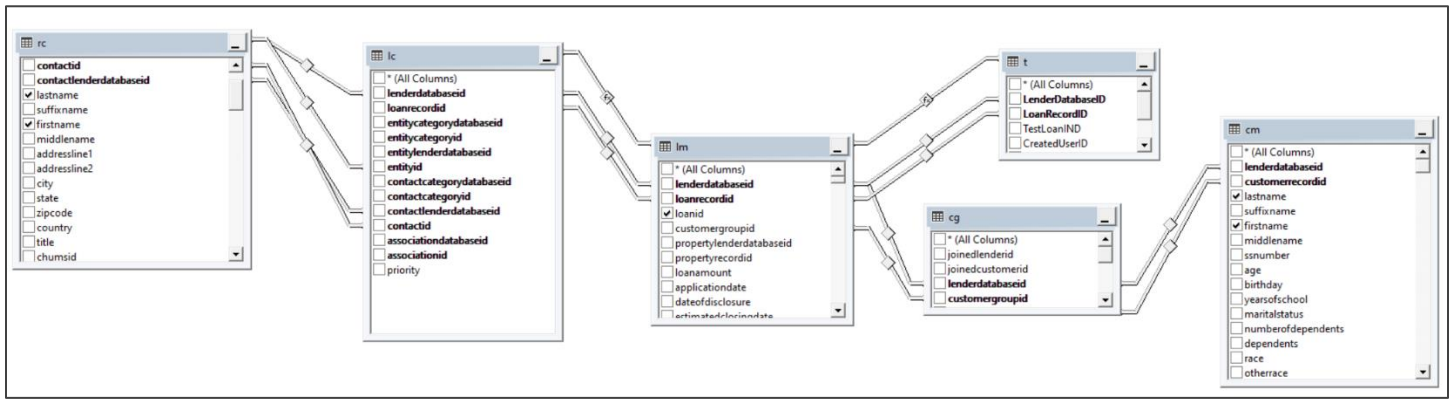
```
--De-Optimized:

--Loans Currently in Submission status. We want to grab the Loan number, Borrowers name and the
Senior UW on the loan.

SELECT lm.loanid
      ,CONCAT(cm.firstname, ' ', cm.lastname) BorrowerName
      ,CONCAT(rc.firstname, ' ', rc.lastname) srUWName
FROM dbo.customer_main cm WITH (NOLOCK)
    INNER JOIN dbo.customer_group cg WITH (NOLOCK)
        ON cg.customerrecordid = cm.customerrecordid
           AND cg.lenderdatabaseid = cm.lenderdatabaseid
    INNER JOIN dbo.loan_main lm WITH (NOLOCK)
        ON lm.customergroupid = cg.customergroupid
           AND lm.lenderdatabaseid = cg.lenderdatabaseid
    INNER JOIN dbo.loan_status ls WITH (NOLOCK)
        ON ls.lenderdatabaseid = lm.lenderdatabaseid
           AND ls.statusid = lm.statusid
           AND ls.loanrecordid = lm.loanrecordid
    INNER JOIN dbo.loan_channelcontacts LC WITH (NOLOCK)
        ON lc.loanrecordid = ls.loanrecordid
           AND lc.lenderdatabaseid = LS.LENDERDATABASEID
           AND lc.contactcategoryid = 8 --senior underwriter
    INNER JOIN dbo.rolodex_contacts rc WITH (NOLOCK)
        ON rc.contactid = lc.contactid
        AND rc.entityid = lc.entityid
    INNER JOIN dbo.usfs_testloanindicator T WITH (NOLOCK)
        ON t.lenderdatabaseid = lm.lenderdatabaseid
           AND t.loanrecordid = lm.loanrecordid
           AND t.testloanind = 0 --removing test loans
WHERE ls.statusid IN ( 2, 49 ) --UW status/submission
```

```
--Optimized:


SELECT lm.loanid
      ,CONCAT(cm.firstname, ' ', cm.lastname) BorrowerName
      ,CONCAT(rc.firstname, ' ', rc.lastname) srUWName
FROM dbo.loan_main lm WITH (NOLOCK)
```

```
    INNER JOIN dbo.customer_group cg WITH (NOLOCK)
        ON cg.lenderdatabaseid = lm.lenderdatabaseid
           AND cg.customergroupid = lm.customergroupid
    INNER JOIN dbo.customer_main cm WITH (NOLOCK)
        ON lm.lenderdatabaseid = lm.lenderdatabaseid
           AND cg.customerrecordid = cm.customerrecordid
    INNER JOIN dbo.loan_channelcontacts lc WITH (NOLOCK)
        ON lc.loanrecordid = lm.loanrecordid
           AND lc.lenderdatabaseid = lm.lenderdatabaseid
           AND lc.contactcategoryid = 8 --srUW
    INNER JOIN dbo.rolodex_contacts rc WITH (NOLOCK)
        ON rc.lenderdatabaseid = lc.lenderdatabaseid
           AND rc.entityid = lc.entityid
           AND rc.contactlenderdatabaseid = lc.contactlenderdatabaseid
           AND rc.contactid = lc.contactid
WHERE lm.statusid IN ( 2, 49 );
```

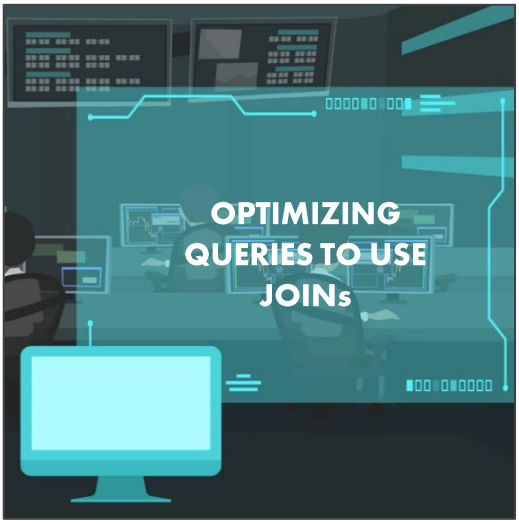
TABLE JOIN RELATIONSHIP

The below diagram represents all the tables and what each table is JOINed to from the above query example. The bolded names are the keys that each table have in common.



INDEPENDENT EXERCISE

It's your turn to practice optimizing a query with multiple JOINs. Review the query below, then optimize the query.



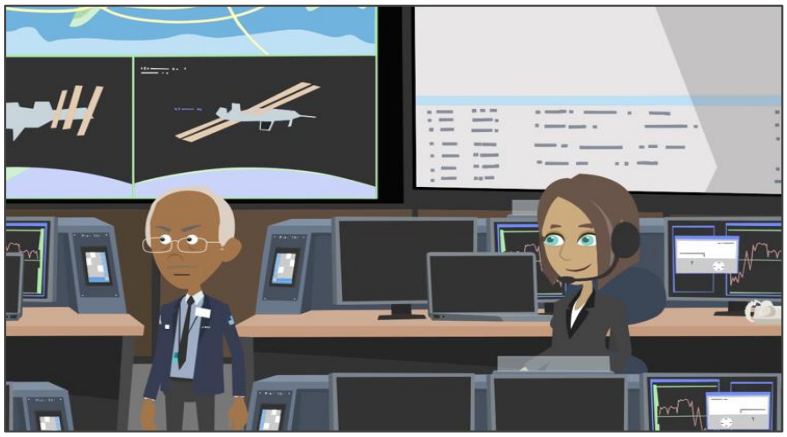

```
--De-optimized:

SELECT lm.loanid
      ,CONCAT(cm.firstname, ' ', cm.lastname)
BorrowerName
      ,CONCAT(rc.firstname, ' ', rc.lastname) srUWName
FROM dbo.loan_main lm WITH (NOLOCK)
    JOIN dbo.customer_group cg WITH (NOLOCK)
        ON  cg.customergroupid = lm.customergroupid
    JOIN dbo.customer_main cm WITH (NOLOCK)
        ON cg.customerrecordid = cm.customerrecordid
    JOIN dbo.loan_channelcontacts lcc WITH (NOLOCK)
        ON lcc.loanrecordid = lm.loanrecordid
            AND lcc.contactcategoryid = 8 --loan officer
    JOIN dbo.rolodex_contacts rc WITH (NOLOCK)
        ON rc.contactid = lcc.contactid
    JOIN dbo.USFS_TestLoanIndicator t WITH (NOLOCK)
        ON t.LoanRecordID = lm.loanrecordid
            AND t.TestLoanIND = 0 --removing test loans
WHERE lm.statusid IN ( 2, 49 );
```

MISSION 3 | REVISIT

Take a moment to critically think about the effects of properly JOINing tables, and answer the following question using the space provided: *How will the proper use of JOINs in your queries make your queries run faster and what impact does that have on the Shore 1 server?*

Proper filtering of JOINs (at the "Join" level) will cause an initial reduction in the size of th data set even before the JOIN conditions are evaluated.

Doing this will reduce the amount of evaluations which need to be done there by reducing the workload SQL Server must perform and finally, in turn, reduce the size of the workload being placed on, ultimately, Shore 1



CLOSING EXERCISE

Review the following queries and then optimize them. Remember to run them through the Execution Plan tool to review things like cost, operator type, CPU time, and number of logical reads. Remember to review available indexes and JOIN criteria. Copy and paste your optimized query in the spaces provided for future reference.

1. Optimize the following query:

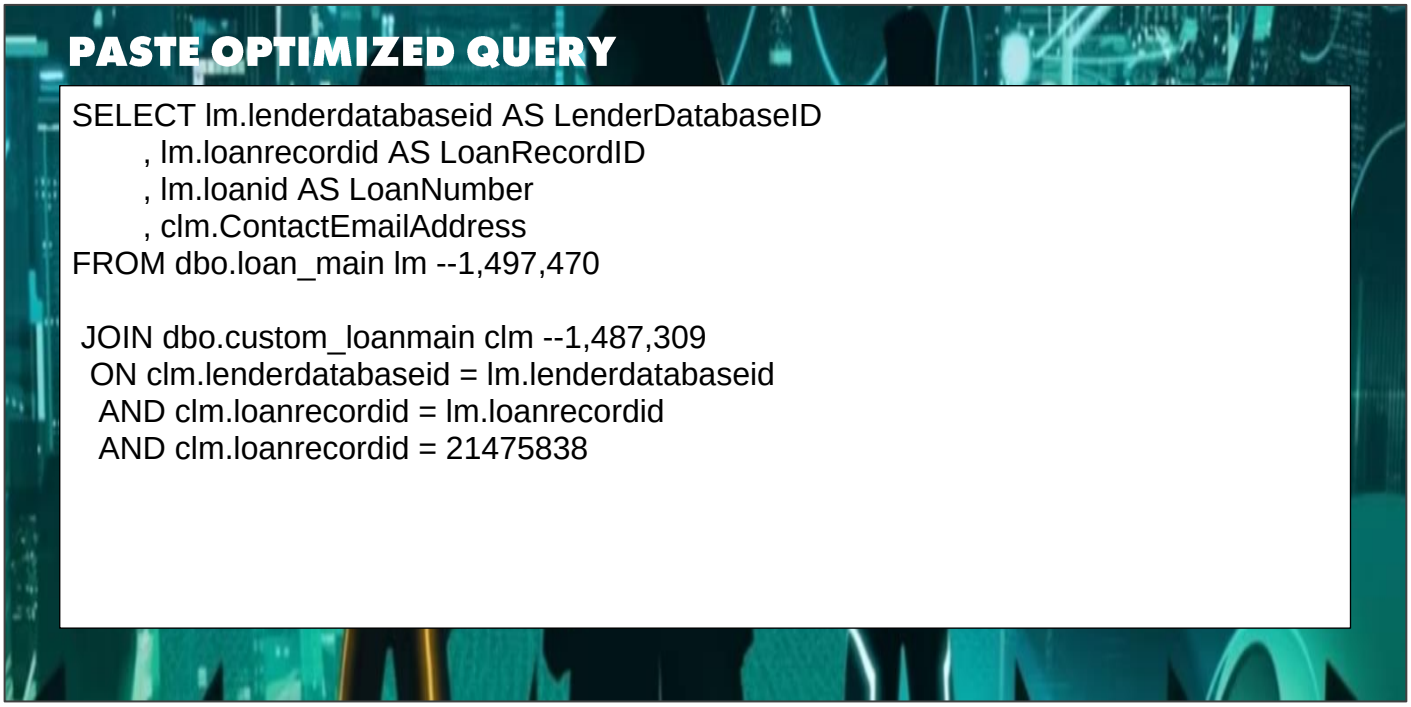
```
--De-optimized:

SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT  lcd.debtid, cd.name, cd.numberofpayments, lcd.exclusionreason, cd.debttype,
lcd.includepayment, lcd.includebalance, cd.balance, cd.payment, cd.lienposition,
lcd.payoffrequired
   FROM    loan_customerdebt lcd
   JOIN  loan_main lm  ON  lm.loanrecordid = lcd.loanrecordid
   JOIN  customer_debt cd  ON cd.debtid = lcd.debtid
           AND cd.debtldid = lcd.debtldid
   WHERE  lcd.loanrecordid = 21493577

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```

PASTE OPTIMIZED QUERY

```
SELECT lcd.debtid
       , cd.name
       , cd.numberofpayments
       , lcd.exclusionreason
       , cd.debttype
       , lcd.includepayment
       , lcd.includebalance
       , cd.balance
       , cd.payment
       , cd.lienposition
       , lcd.payoffrequired
FROM loan_customerdebt lcd

 JOIN loan_main lm --1,497,470
```

2. Optimize the following query:

```
--De-optimized:

SET STATISTICS IO ON
SET STATISTICS TIME ON

SELECT  lm.lenderdatabaseid AS LenderDatabaseID, lm.loanrecordid AS LoanRecordID,
lm.loanid AS LoanNumber, clm.ContactEmailAddress
        FROM dbo.loan_main lm
        JOIN dbo.custom_loanmain clm ON clm.loanrecordid = lm.loanrecordid AND
clm.lenderdatabaseid = lm.lenderdatabaseid

        WHERE lm.loanrecordid =  21475838

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
```



3. Optimize the following query:

```
--de-optimized

SET STATISTICS IO ON;
SET STATISTICS TIME ON;

SELECT lm.lenderdatabaseid
      ,lm.loanrecordid
      ,lm.loanid
      ,lpc.dispositiondate
      ,sls.statusdescription
      ,p.productdescription
FROM dbo.loan_main AS lm WITH (NOLOCK)
    JOIN dbo.loan_postclosing AS lpc WITH (NOLOCK)
        ON lpc.loanrecordid = lm.loanrecordid
           AND lpc.disposition = 1
    JOIN dbo.property_main AS pm WITH (NOLOCK)
        ON pm.propertyrecordid = lm.propertyrecordid
           AND pm.state = 'ND'
    JOIN dbo.loan_huditem AS lhi WITH (NOLOCK)
        ON lhi.loanrecordid = lm.loanrecordid
           AND lhi.lenderdatabaseid = lm.lenderdatabaseid
           AND lhi.currenthudnumber IN ( 804, 805 )
    JOIN dbo.setups_loanstatus sls WITH (NOLOCK)
        ON sls.statusid = lm.statusid
    JOIN dbo.product_main p WITH (NOLOCK)
        ON p.productid = lm.productid;

SET STATISTICS IO ON;
SET STATISTICS TIME ON;
```

PASTE OPTIMIZED QUERY

```
--What I have so far but I'm moderately confident it's not right...

SELECT lm.lenderdatabaseid
      ,lm.loanrecordid
      ,lm.loanid
      ,lpc.dispositiondate
      ,sls.statusdescription
      ,p.productdescription

FROM dbo.loan_main AS lm WITH (NOLOCK)

 JOIN dbo.setups_loanstatus sls WITH (NOLOCK) --60
```

COMPLETED MISSIONS

Congratulations! You have accomplished these three missions and saved the city from all ticking time bombs!

- Practice generating and interpreting execution plans for several SQL queries.
- Practice optimizing several SQL queries using indexes.
- Practice optimizing several SQL queries through proficient use of JOINs with 5 or more tables at a time.

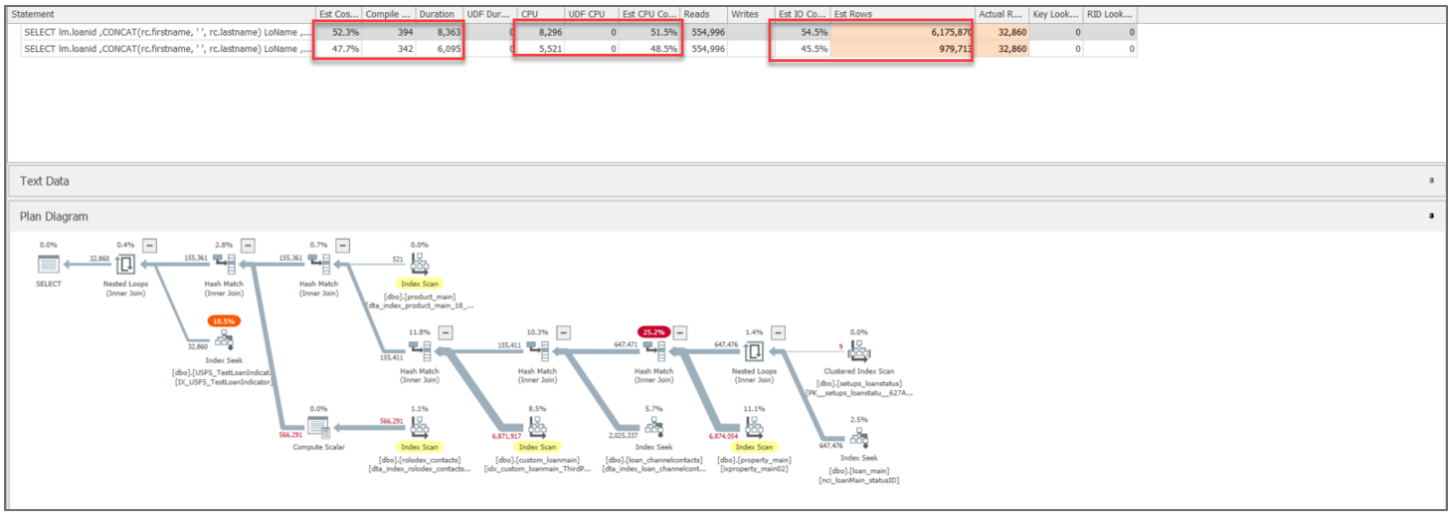
ADDITIONAL RESOURCES

See the below additional videos and readings that cover what was included in today's SQL training.

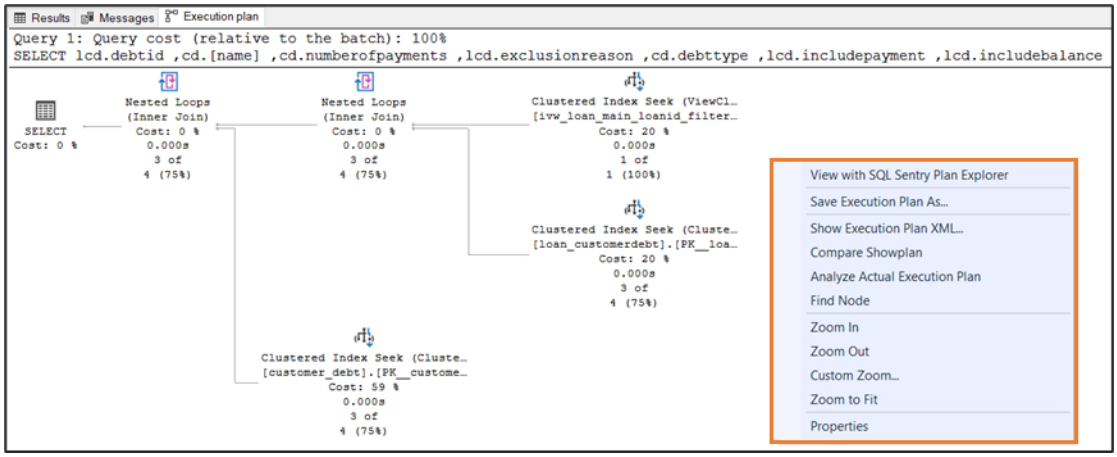
- *https://www.youtube.com/watch?v=SMw2knRullE)*
- *https://www.youtube.com/watch?v=uqkf1VVZulg&t=474s*
- *https://learning.oreilly.com/library/view/sql-tuning/0596005733/*

SQL SENTRY PLAN EXPLORER TOOL

SQL Sentry Plan Explorer tool is another version of an execution plan that can be installed through a third-party website based upon personal preference. Some team members find it slightly easier to compare execution plans between multiple queries; however, both execution plan tools run the same way and provide the same information.



Click here to install. Once installed, turn on actual execution plan and run query in SSMS. Once query runs, right click anywhere in the execution plan and select **View with SQL Sentry Plan Explorer**.



If you have any questions or concerns, please reach out to ittraining@uwm.com.