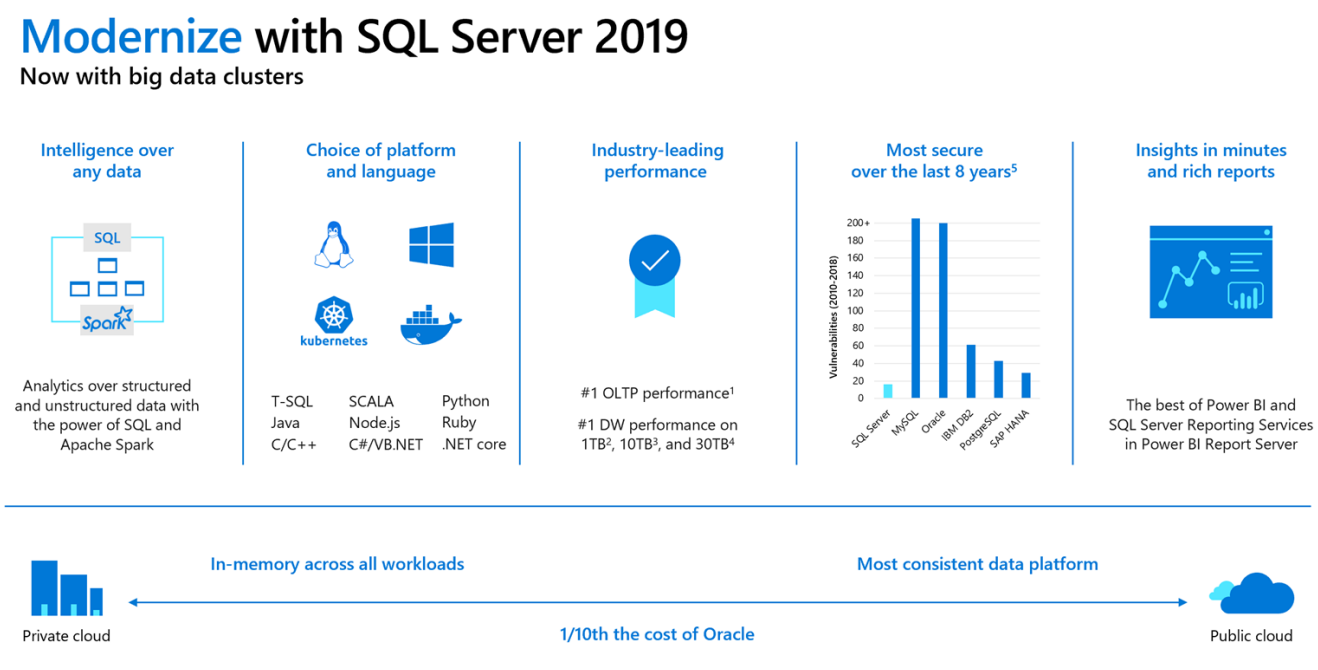
Data & AI Tech Immersion Workshop – Product Review Guide and Lab Instructions

Data, Experience 1 - Business Critical Performance and Security with SQL Server 2019

- Data & AI Tech Immersion Workshop – Product Review Guide and Lab Instructions
  - Data, Experience 1 - Business Critical Performance and Security with SQL Server 2019
  - Technology overview
  - Scenario overview
  - Experience requirements
  - Task 1: Connect to SQL Server 2019 with SSMS
  - Task 2: Query performance improvements with intelligent query processing
  - Task 3: Identify PII and GDPR-related compliance issues using Data Discovery & Classification in SSMS
  - Task 4: Fix compliance issues with dynamic data masking
  - Task 5: Restrict data access with Row-level security
  - Task 6: Always Encrypted with secure enclaves
    - Configure a secure enclave
    - Provision enclave-enabled keys
    - Encrypt customer email column
    - Run rich queries against an encrypted column
  - Wrap-up
  - Additional resources and more information

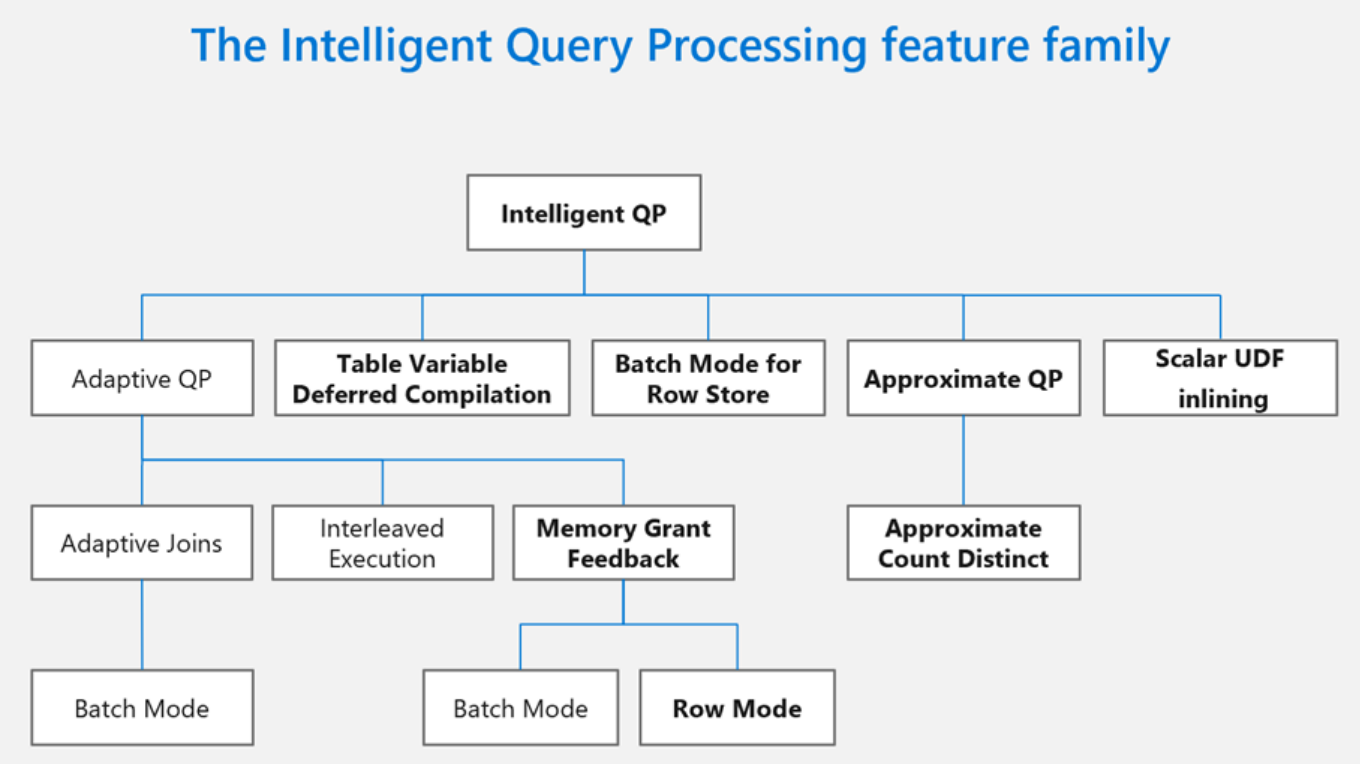
Technology overview



All TPC Claims as of 1/19/2018.

[1] http://www.tpc.org/4081; [2] http://www.tpc.org/3331; [3] http://www.tpc.org/3326; [4] http://www.tpc.org/3321; [5] National Institute of Standards and Technology Comprehensive Vulnerability Database

Modernize with SQL Server 2019 with new features that help combine new technologies, and updates to existing SQL Server features you use today. New big data clusters brings Apache Spark and enhanced PolyBase features that allow you to easily access structured and unstructured data stored in HDFS and external relational stores, enabling data virtualization and analytics at scale.



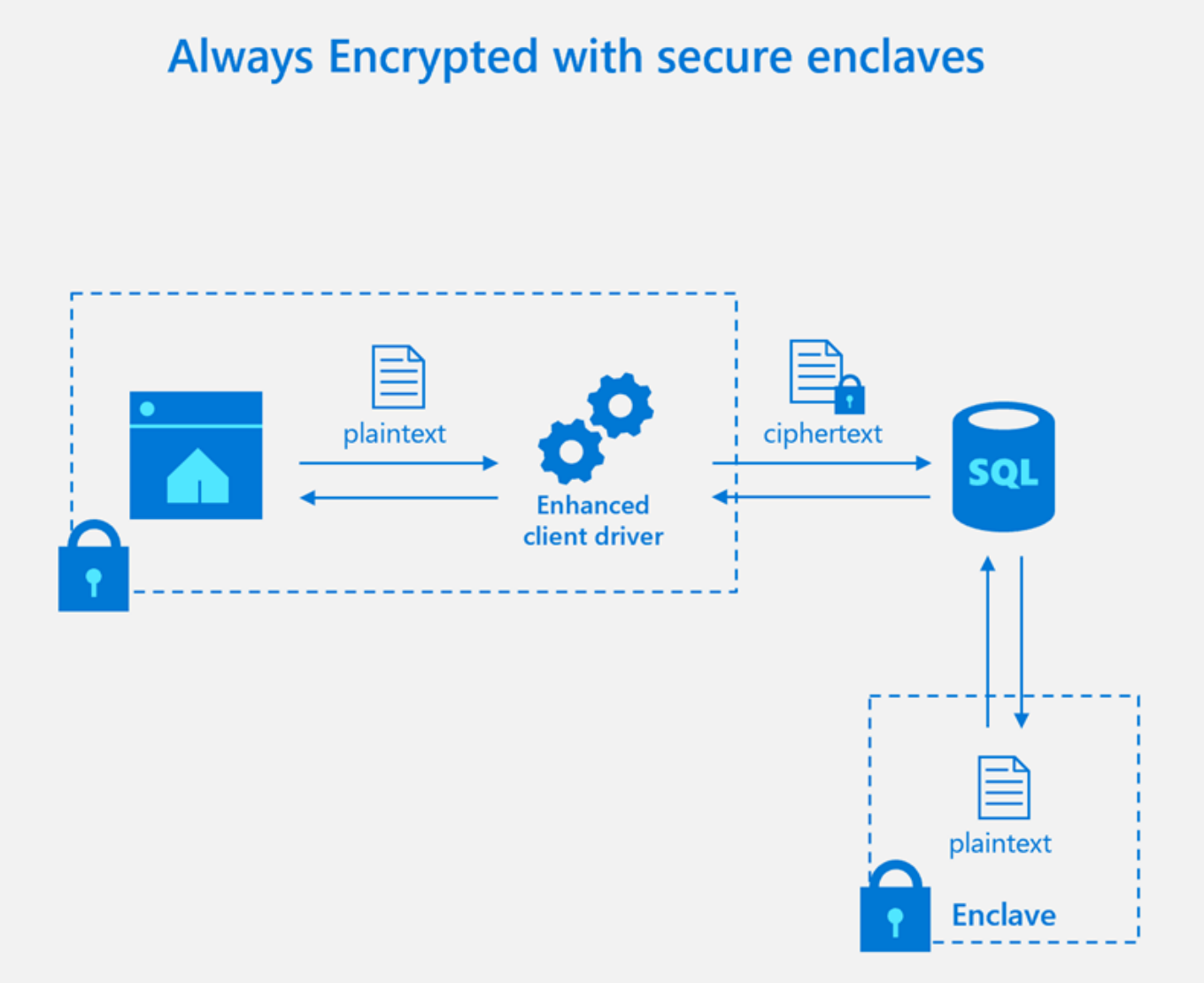
> The IQP features shown in bold are new and improved features in SQL Server 2019.

Intelligent Query Processing (IQP) features in SQL Server 2019 improve scaling of queries, and Automatic Plan Correction resolves performance problems. In addition, you can gain performance insights anytime and anywhere with Lightweight Query Profiling.

To further boost performance, SQL Server 2019 provides more in-memory database options than ever before, such as:

- Hybrid Buffer Pool
- Memory-Optimized TempDB Metadata
- In-Memory OLTP
- Persistent Memory Support

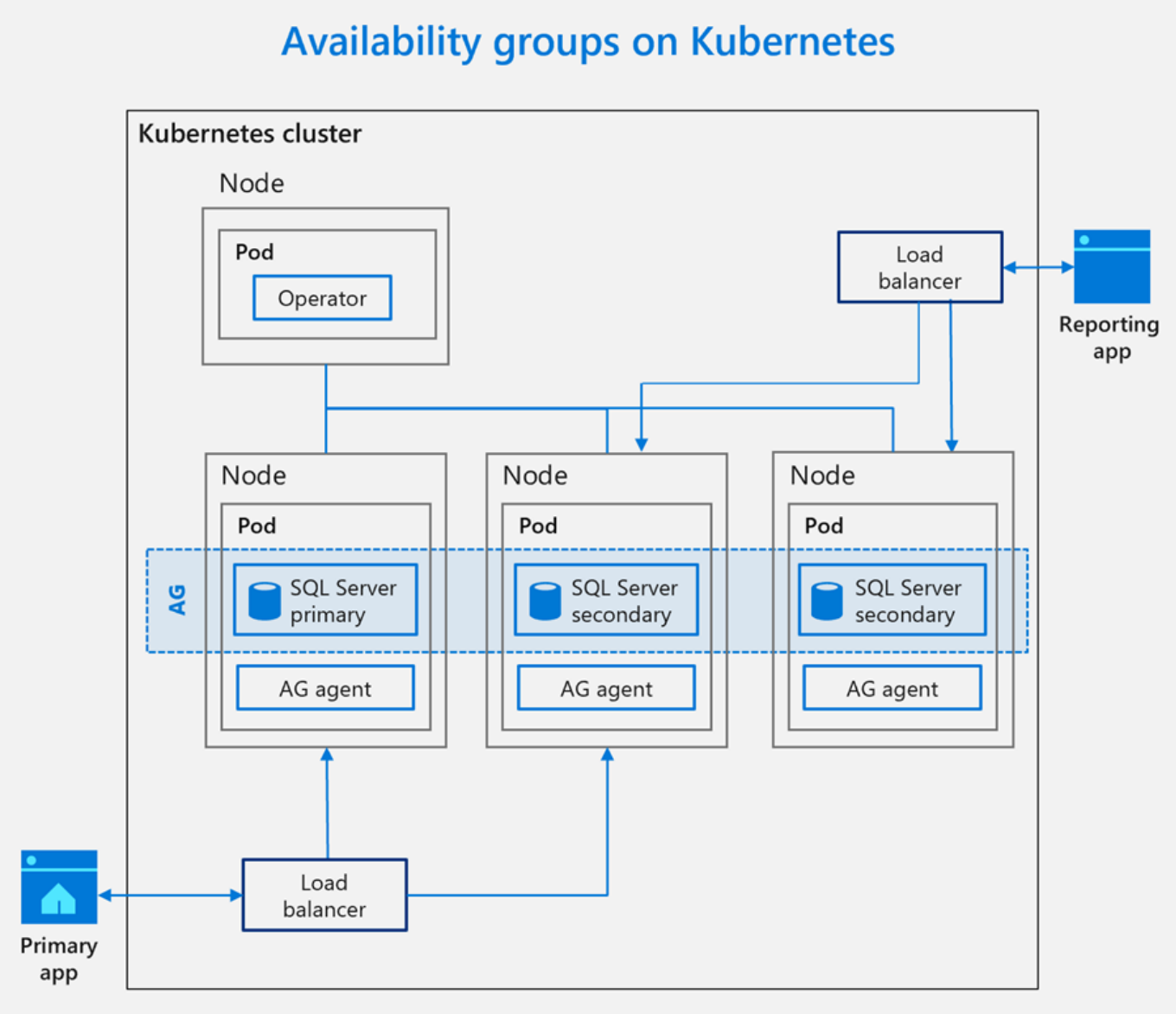
The Hybrid Buffer Pool allows the database engine to directly access data pages in database files stored on persistent memory (PMEM) devices. SQL Server will automatically detect if data files reside on an appropriately formatted PMEM device and perform memory mapping in user space. This mapping happens upon startup, when a new database is attached, restored, created, or when the hybrid buffer pool feature is enabled for a database.



SQL Server 2019 enables several layers of security including protection of computations in Always Encrypted with secure enclaves. Customers can track compliance with sophisticated tools such as Data Discovery & Classification labeling for GDPR and Vulnerability Assessment tool. Transparent Data Encryption (TDE) encryption scanning now offers more control with suspend and resume syntax so that you can pause the scan while the workload on the system is heavy, or during business-critical hours, and then resume the scan later.

SSL/TLS certificates are widely used to secure access to SQL Server. In previous editions, certificate management was a more manual and time-consuming process, through developing scripts and running manual commands. With SQL Server 2019, certificate management is integrated into the SQL Server Configuration Manager, simplifying common tasks such as:

- Viewing and validating certificates installed in a SQL Server instance.
- Identifying which certificates may be close to expiring.
- Deploying certificates across Availability Group machines from the node holding the primary replica.
- Deploying certificates across machines participating in a Failover Cluster instance from the active node.



For High Availability and Disaster Recovery, SQL Server 2019 now supports up to eight secondary replicas in an Always On Availability Group. Customers can also run Always On Availability Groups on containers using Kubernetes.

SQL Server 2019 also has powerful tools for Business Intelligence including Analysis Services and Power BI Report Server which provide visual data exploration and interactive analysis of business data.

Scenario overview

This experience will highlight the new features of SQL Server 2019 with a focus on performance and security. You will begin by performing an assessment of Contoso Auto's on-premises database to determine feasibility for migrating to SQL Server on a VM in Azure, and then complete the database migration. Next, you will gain hands-on experience by running queries using some of the new query performance enhancements and evaluating the results. You will evaluate the data security and compliance features provided by SQL Server 2019 by using the Data Discovery & Classification tool in SSMS to identify tables and columns with PII and GDPR-related compliance issues. You will then address some of the security issues by layering on dynamic data masking, row-level security, and Always Encrypted with secure enclaves.

Experience requirements

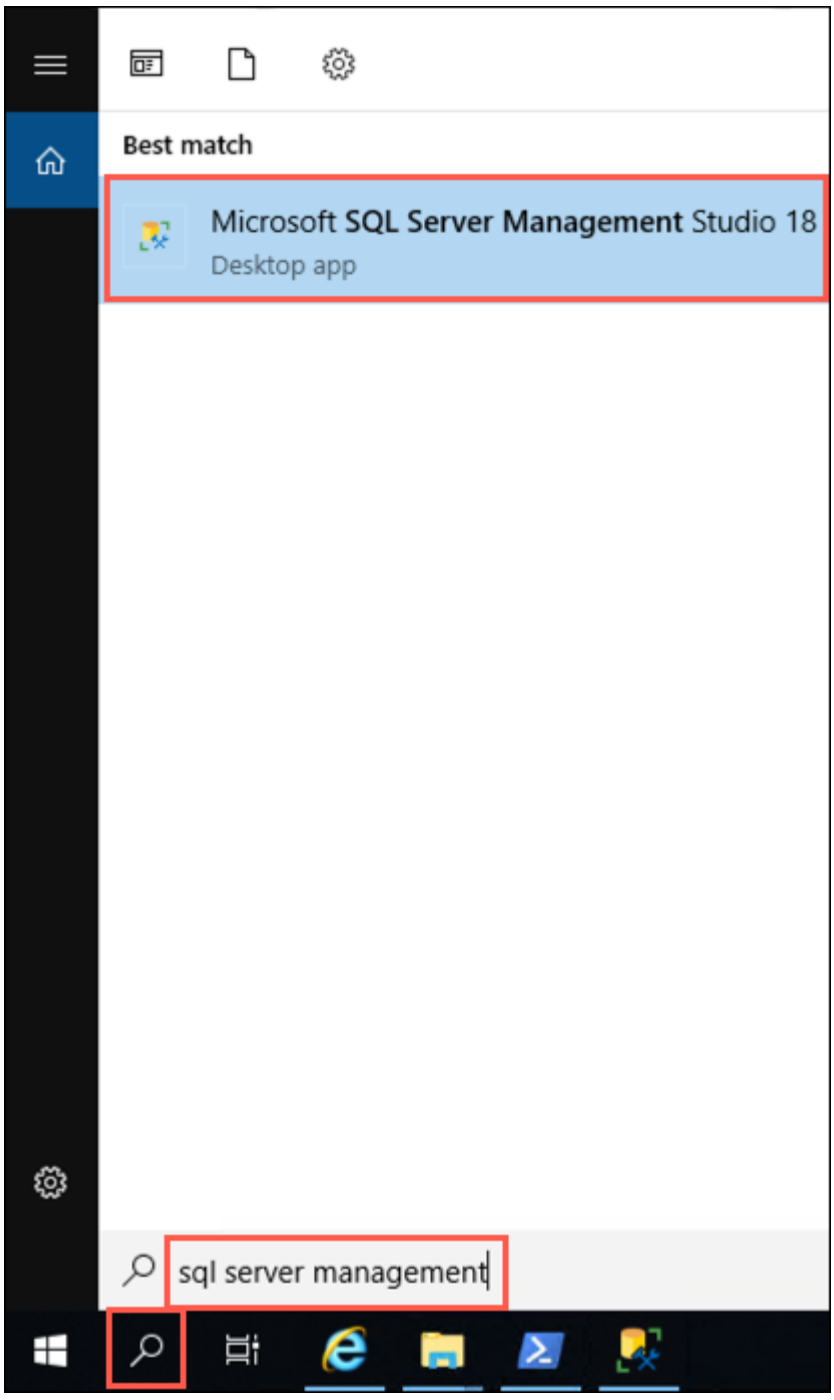
Before you begin this lab, you need to find the following information on the Tech Immersion Mega Data & AI Workshop On Demand Lab environment details page, or the document provided to you for this experience:

- SQL Server 2019 VM IP address: `SQL_SERVER_2019_VM_IP`
- Sales database name (your unique copy): `SALES_DB`

Task 1: Connect to SQL Server 2019 with SSMS

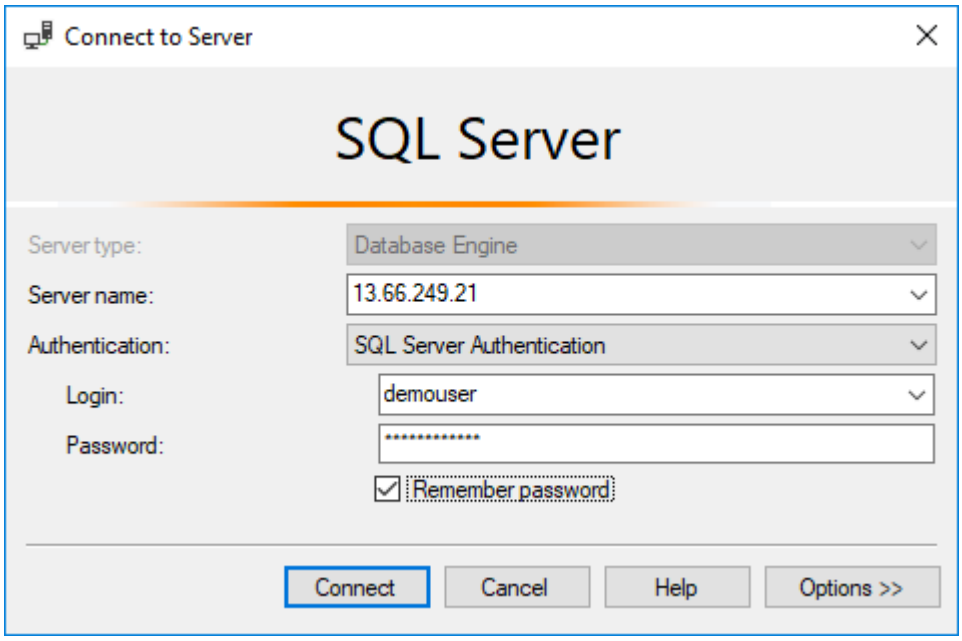
You will be accessing a shared SQL Server 2019 server for this experience. Please follow the steps below to connect to the SQL Server 2019 VM with SQL Server Management Studio (SSMS).

1. On the bottom-left corner of your Windows desktop, locate the search box next to the Start Menu. Type **SQL Server Management** into the search box, then select the SQL Server Management Studio 18 desktop app in the search results.



2. Within the Connection dialog that appears, configure the following:

    - **Server name:** Enter the SQL Server 2019 VM IP address. Use the value from the `SQL_SERVER_2019_VM_IP` for this from the environment documentation.
    - **Authentication:** Select SQL Server Authentication.
    - **Login:** Enter `demouser`
    - **Password:** Enter `Password.1!!`
    - **Remember password:** Check this box.



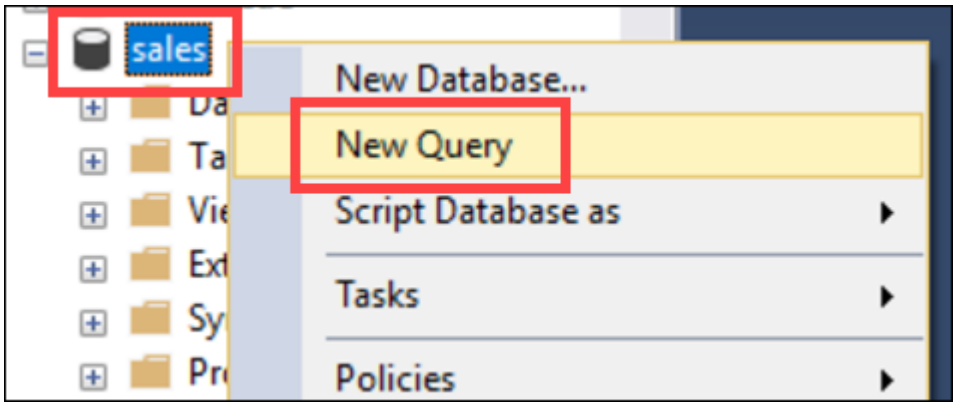
3. Select **Connect**.

Task 2: Query performance improvements with intelligent query processing

In this task, you will execute a series of SQL scripts in SQL Server Management Studio (SSMS) to explore the improvements to the family of intelligent query processing (QP) features in SQL Server 2019. These features improve the performance of existing workloads with minimal work on your part to implement. The key to enabling these features in SQL Server 2019 is to set the database compatibility level to `150`. You will be executing these queries against the `sales_XXXXX` database (where XXXXX is the unique identifier assigned to you for this workshop).

To learn more, read intelligent query processing in SQL databases.

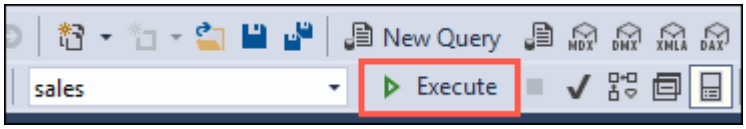
1. To get started, expand databases in the SQL Server Management Studio (SSMS) Object Explorer, right-click the `sales_XXXXX` database (where XXXXX is the unique identifier assigned to you for this workshop), and then select **New Query**.



2. The first query you will run is to set the database compatibility level to `150`, which is the new compatibility level for SQL Server 2019, enabling the most recent intelligent QP features. Copy the SQL script below and paste it into the new query window. Replace `XXXXX` with the unique identifier you have been given for this workshop in both the `USE` and `ALTER DATABASE` statements.

```
USE sales_XXXXX;
GO

ALTER DATABASE sales_XXXXX
SET COMPATIBILITY_LEVEL = 150;
GO
```

3. To run the query, select **Execute** in the SSMS toolbar.



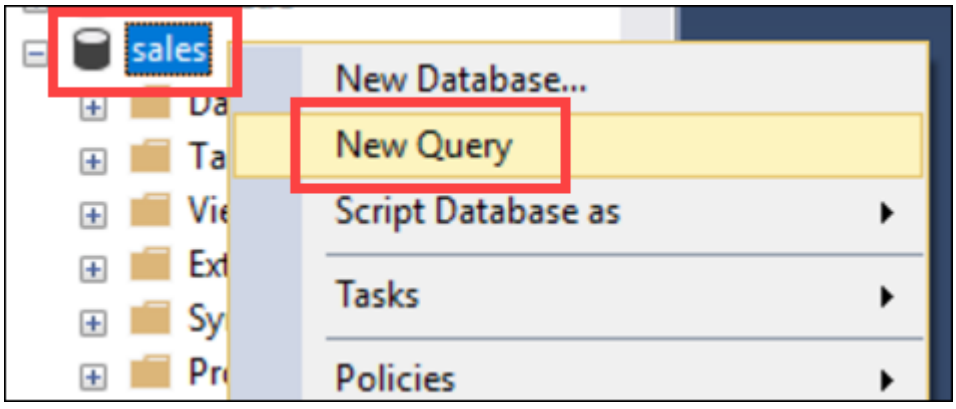
4. Next, you will run a query to create a user-defined function (UDF) named `customer_category`. This UDF contains several steps to identify the discount price category for each customer. Notice that at the top of the query we run to create this UDF sets the database compatibility level to `150`, which is the new compatibility level for SQL Server 2019, enabling the most recent intelligent QP features. This UDF will be called inline from the two queries that follow in order to show QP improvements on scalar UDF inlining. Paste the following SQL code into your query window, overwriting the current content, replace `XXXXX` in the `USE` statement with the unique identifier assigned to you for this workshop, and then select **Execute** on the SSMS toolbar.

```
USE sales_XXXXX;
GO

ALTER DATABASE SCOPED CONFIGURATION
CLEAR PROCEDURE_CACHE;
GO

CREATE OR ALTER FUNCTION
  dbo.customer_category(@CustomerKey INT)
RETURNS CHAR(10) AS
```

```
BEGIN
  DECLARE @total_amount DECIMAL(18,2);
  DECLARE @category CHAR(10);

  SELECT @total_amount =
  SUM([ws_net_paid_inc_ship])
  FROM [dbo].[web_sales]
  WHERE [ws_bill_customer_sk] = @CustomerKey;

  IF @total_amount < 50000
    SET @category = 'REGULAR';
  ELSE IF @total_amount < 100000
    SET @category = 'GOLD';
  ELSE
    SET @category = 'PLATINUM';

  RETURN @category;
END
GO
```

> Scalar UDF inlining automatically transforms scalar UDFs into relational expressions. It embeds them in the calling SQL query. This transformation improves the performance of workloads that take advantage of scalar UDFs. Scalar UDF inlining facilitates cost-based optimization of operations inside UDFs. The results are efficient, set-oriented, and parallel instead of inefficient, iterative, serial execution plans. This feature is enabled by default under database compatibility level 150. *For more information, see Scalar UDF inlining*.

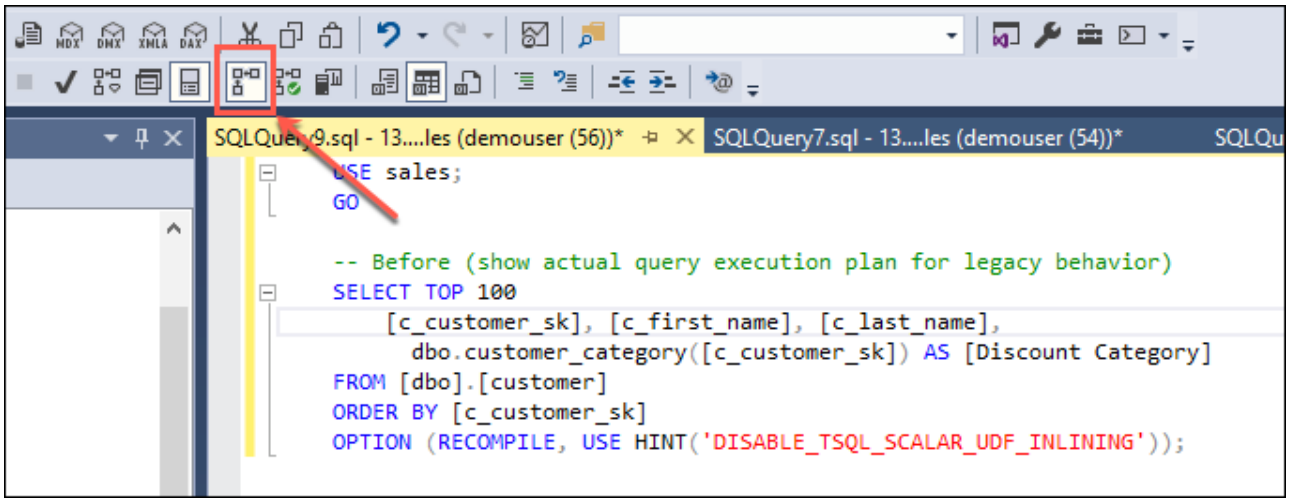
5. Right-click on the `sales_XXXXX` database (where XXXXX is the unique identifier assigned to you for this workshop), then select **New Query**. This will open a new query window into which you can paste the following queries. You may wish to reuse the same query window, replacing its contents with each SQL statement blocks below, or follow these same steps to create new query windows for each.



6. The query below selects the top 100 rows from the `customer` table, calling the `customer_category` user-defined function (UDF) inline for each row. It uses the `DISABLE_TSQL_SCALAR_UDF_INLINING` hint to disable the new scalar UDF inlining QP feature. Paste the following query into the the empty query window. Replace `XXXXX` in the `USE` statement with the unique identifier assigned to you for this workshop. **Do not execute yet**.

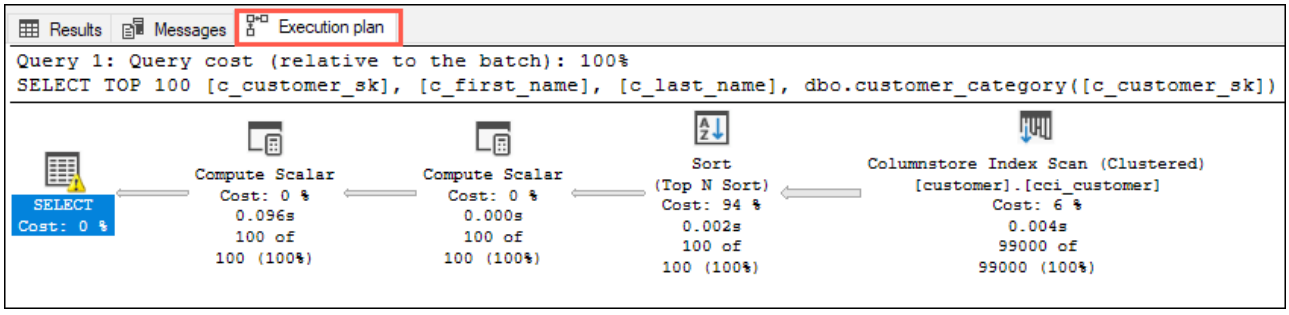
```
USE sales_XXXXX;
GO

-- Before (show actual query execution plan for legacy behavior)
SELECT TOP 100
    [c_customer_sk], [c_first_name], [c_last_name],
      dbo.customer_category([c_customer_sk]) AS [Discount Category]
FROM [dbo].[customer]
ORDER BY [c_customer_sk]
OPTION (RECOMPILE, USE HINT('DISABLE_TSQL_SCALAR_UDF_INLINING'));
```

7. Select the **Include Actual Execution Plan** (Ctrl+M) button in the toolbar above the query window. This will allow us to view the actual (not estimated) query plan after executing the query.



8. Execute the query by selecting **Execute** from the SSMS toolbar.

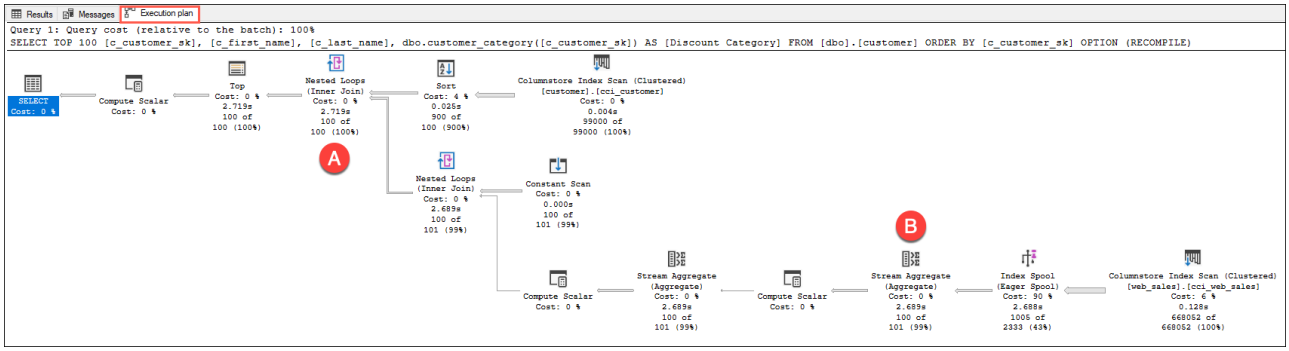
9. After the query executes, select the **Execution plan** tab. As the plan shows, SQL Server adopts a simple strategy here: for every tuple in the `customer` table, invoke the UDF and output the results (single line from the clustered index scan to compute scalar). This strategy is naïve and inefficient, especially with more complex queries.



10. Clear the query window, or open a new one, then paste the following query that makes use of the scalar UDF inlining QP feature. Replace `XXXXX` in the `USE` statement with the unique identifier assigned to you for this workshop. If you opened a new query window instead of reusing this one, make sure to select the **Include Actual Execution Plan** button to enable it. **Execute** the query.

```
USE sales_XXXXX;
GO

-- After (show actual query execution plan for legacy behavior)
SELECT TOP 100
    [c_customer_sk], [c_first_name], [c_last_name],
      dbo.customer_category([c_customer_sk]) AS [Discount Category]
FROM [dbo].[customer]
ORDER BY [c_customer_sk]
OPTION (RECOMPILE);
```

11. After the query executes, select the **Execution plan** tab once again. With scalar UDF inlining, this UDF is transformed into equivalent scalar subqueries, which are substituted in the calling query in place of the UDF.



> As you can see, the query plan no longer has a user-defined function operator, but its effects are now observable in the plan, like views or inline TVFs. Here are some key observations from the above plan:

A. SQL Server has inferred the implicit join between `dbo.customer` and `dbo.web_sales` and made that explicit via a join operator.

B. SQL Server has also inferred the implicit `GROUP BY [Customer Key] on dbo.web_sales` and has used the IndexSpool + StreamAggregate to implement it.

> Depending upon the complexity of the logic in the UDF, the resulting query plan might also get bigger and more complex. As we can see, the operations inside the UDF are now no longer a black box, and hence the query optimizer is able to cost and optimize those operations. Also, since the UDF is no longer in the plan, iterative UDF invocation is replaced by a plan that completely avoids function call overhead.

12. Either highlight and delete everything in the query window, or open a new query window. Paste the following query into the query window, replacing `XXXXX` in the `USE` statement with the unique identifier assigned to you for this workshop. This query makes use of the table variable deferred compilation feature, since the database compatibility level is set to `150`. If you opened a new query window instead of reusing this one, make sure to click the **Include Actual Execution Plan** button to enable it. **Execute** the query.

```
USE sales_XXXXX
GO

DECLARE @ItemClick TABLE (
  [itemKey] BIGINT NOT NULL,
  [clickDate] BIGINT NOT NULL
);

INSERT @ItemClick
SELECT [wcs_item_sk], [wcs_click_date_sk]
FROM [dbo].[web_clickstreams]

-- Look at estimated rows, speed, join algorithm
SELECT i.[i_item_sk], i.[i_current_price], c.[clickDate]
FROM dbo.item AS i
INNER JOIN @ItemClick AS c
  ON i.[i_item_sk] = c.[itemKey]
WHERE i.[i_current_price] > 90
ORDER BY i.[i_current_price] DESC
OPTION (USE HINT('DISABLE_DEFERRED_COMPILATION_TV'));
GO
```

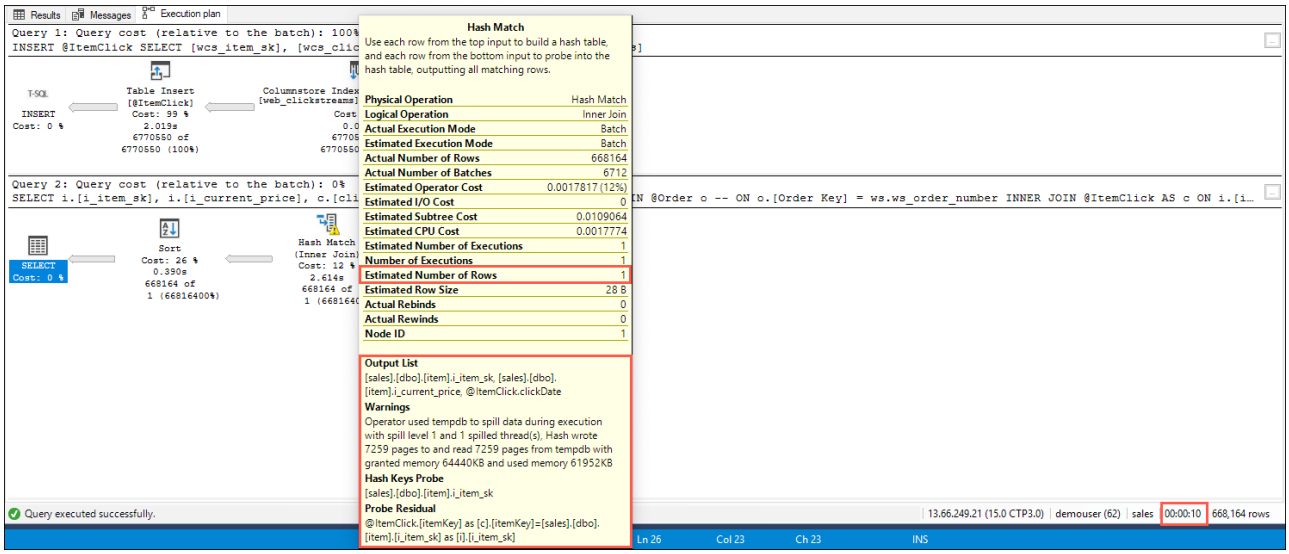
> The script above assigns a table variable, `@ItemClick`, storing the `itemKey` and `clickDate` fields from the `web_clickstreams` table to be used in an INNER JOIN below.

> The `DISABLE_DEFERRED_COMPILATION_TV` hint **disables** the table-deferred compilation feature.

Old method

In prior versions of SQL Server (compatibility level of 140 or lower), the table variable deferred compilation QP feature is not used (more on this below).

There are two plans. The one you want to observe is the second query plan. Because we disabled the table-deferred compilation feature with the `DISABLE_DEFERRED_COMPILATION_TV` hint, when we mouse over the INNER JOIN to view the estimated number of rows and the output list, which shows the join algorithm. The estimated number of rows is 1. Also, observe the execution time. In our case, it took 10 seconds to complete.

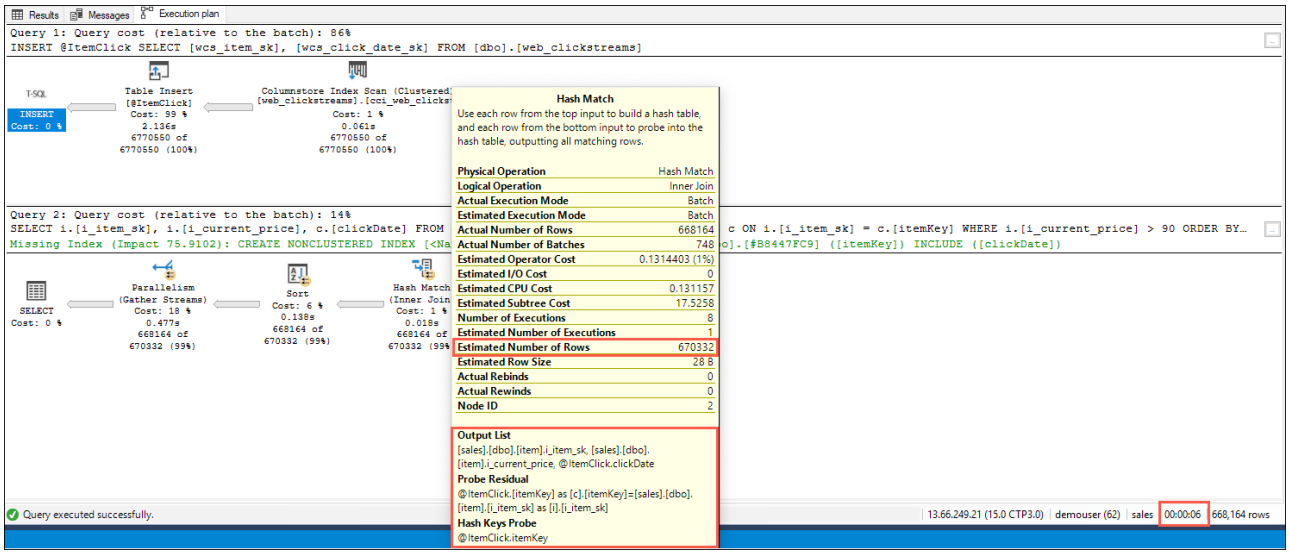


New method

Execute the following updated query, which removes the hint we used in the previous query to disable the table-deferred compilation feature:

```
USE sales_XXXXX
GO

DECLARE @ItemClick TABLE (
  [itemKey] BIGINT NOT NULL,
  [clickDate] BIGINT NOT NULL
);

INSERT @ItemClick
SELECT [wcs_item_sk], [wcs_click_date_sk]
FROM [dbo].[web_clickstreams]

-- Look at estimated rows, speed, join algorithm
SELECT i.[i_item_sk], i.[i_current_price], c.[clickDate]
FROM dbo.item AS i
INNER JOIN @ItemClick AS c
  ON i.[i_item_sk] = c.[itemKey]
WHERE i.[i_current_price] > 90
ORDER BY i.[i_current_price] DESC;
GO
```

After the query above executes, select the **Execution plan** tab once again. Since our database compatibility level is set to 150, notice that the join algorithm is a hash match, and that the overall query execution plan looks different. When you hover over the INNER JOIN, notice that there is a high value for estimated number of rows and that the output list shows the use of hash keys and an optimized join algorithm. Once again, observe the execution time. In our case, it took 6 seconds to complete, which is approximately half the time it took to execute without the table variable deferred compilation feature.



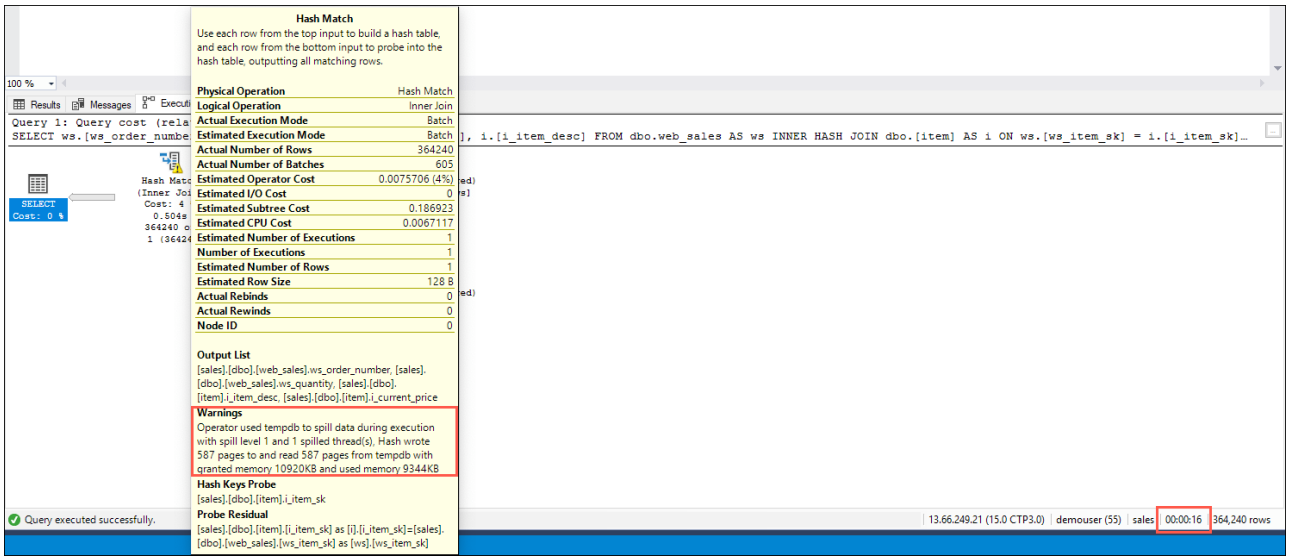
> Table variable deferred compilation improves plan quality and overall performance for queries that reference table variables. During optimization and initial compilation, this feature propagates cardinality estimates that are based on actual table variable row counts. This accurate row count information optimizes downstream plan operations. Table variable deferred compilation defers compilation of a statement that references a table variable until the first actual run of the statement. This deferred compilation behavior is the same as that of temporary tables. This change results in the use of actual cardinality instead of the original one-row guess. *For more information, see Table variable deferred compilation.*

13. Either highlight and delete everything in the query window, or open a new query window. Paste the following query to simulate out-of-date statistics on the `web_sales` table, followed by a query that executes a hash match, replacing `XXXXX` in the `USE` statement with the unique identifier assigned to you for this workshop. If you opened a new query window instead of reusing this one, make sure to click the **Include Actual Execution Plan** button to enable it. **Execute** the query.

```
USE sales_XXXXX;
GO

-- Simulate out-of-date stats
UPDATE STATISTICS dbo.web_sales
WITH ROWCOUNT = 1;
GO

SELECT
  ws.[ws_order_number], ws.ws_quantity,
  i.[i_current_price], i.[i_item_desc]
FROM    dbo.web_sales AS ws
INNER HASH JOIN dbo.[item] AS i
  ON ws.[ws_item_sk] = i.[i_item_sk]
WHERE   i.[i_current_price] > 10
  AND ws.[ws_quantity] > 40;
```

14. After the query executes, select the **Execution plan** tab. Hover over the Hash Match step of the execution plan. You should see a warning toward the bottom of the Hash Match dialog showing spilled

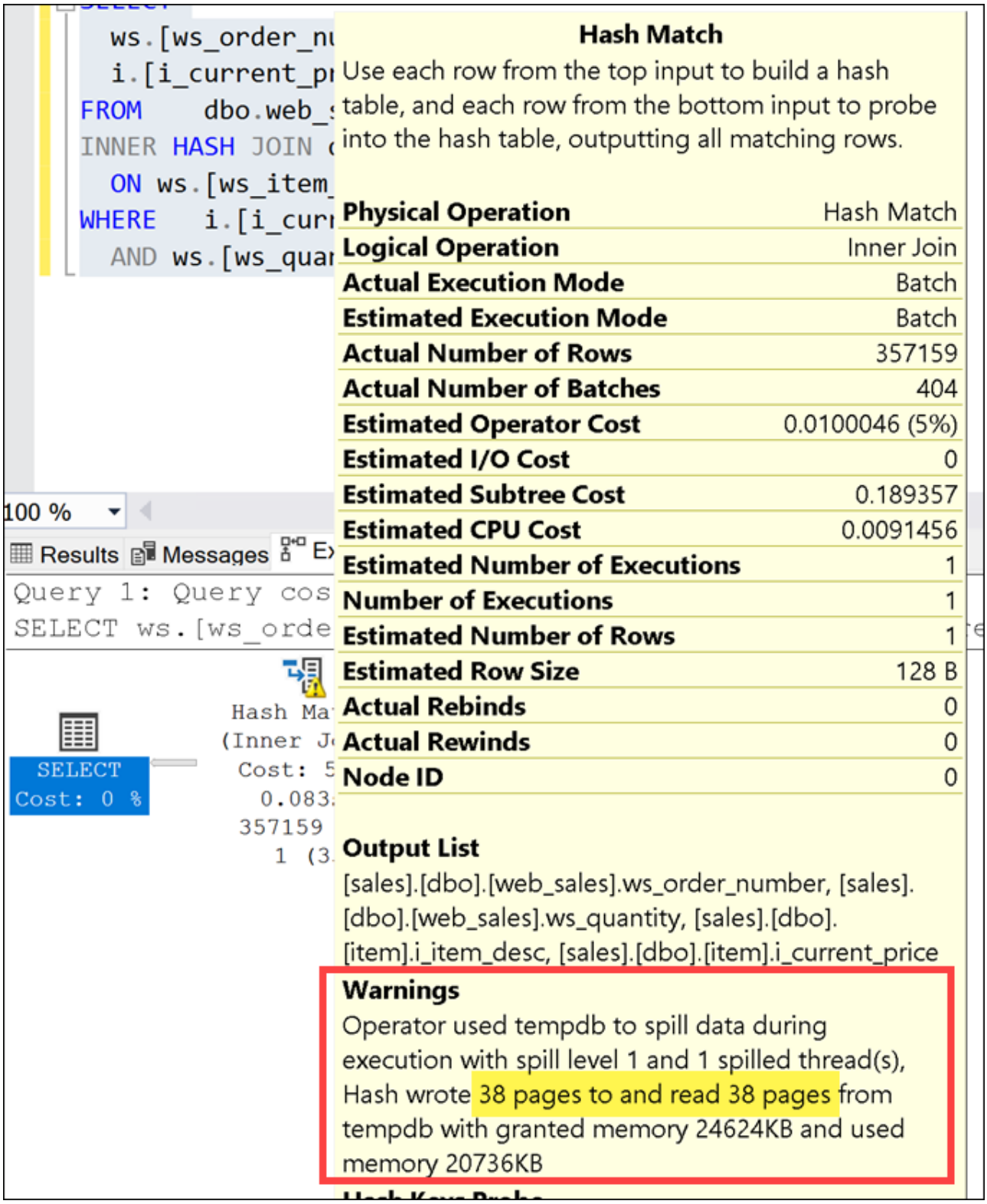
data. Also observe the execution time. In our case, this query took 16 seconds to execute.



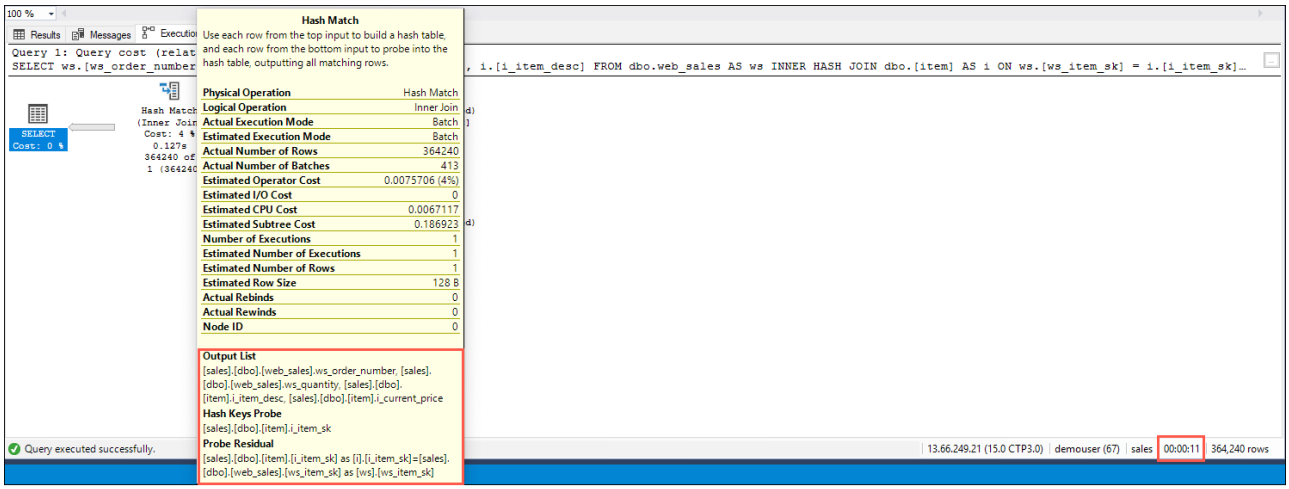
15. Either highlight and delete everything in the query window, or open a new query window. Paste the following query to execute the select query that contains the hash match once more. If you opened a new query window instead of reusing this one, make sure to click the **Include Actual Execution Plan** button to enable it. **Execute** the query.

```
USE sales_XXXXX;
GO

SELECT
  ws.[ws_order_number], ws.ws_quantity,
  i.[i_current_price], i.[i_item_desc]
FROM    dbo.web_sales AS ws
INNER HASH JOIN dbo.[item] AS i
  ON ws.[ws_item_sk] = i.[i_item_sk]
WHERE   i.[i_current_price] > 10
  AND ws.[ws_quantity] > 40;
```

16. After the query executes, select the **Execution plan** tab. Hover over the Hash Match step of the execution plan. You may no longer see a warning about spilled data. If you do, the **number of pages Hash wrote** should have decreased. This happens as the STATISTICS table is updated with each run.



17. Execute the query 2-3 more times. Each time, select the **Execution plan** tab and hover over the Hash Match step of the execution plan. After a few executions, you should **no longer** see a warning about spilled data.



> So what happened? A query's post-execution plan in SQL Server includes the minimum required memory needed for execution and the ideal memory grant size to have all rows fit in memory. Performance suffers when memory grant sizes are incorrectly sized. Excessive grants result in wasted memory and reduced concurrency. Insufficient memory grants cause expensive spills to disk. By addressing repeating workloads, batch mode memory grant feedback recalculates the actual memory required for a query and then updates the grant value for the cached plan. **When an identical query statement is executed**, the query uses the revised memory grant size, reducing excessive memory grants that impact concurrency and fixing underestimated memory grants that cause expensive spills to disk. Row mode memory grant feedback expands on the batch mode memory grant feedback feature by adjusting memory grant sizes for both batch and row mode operators. *For more information, see Row mode memory grant feedback*.

Task 3: Identify PII and GDPR-related compliance issues using Data Discovery & Classification in SSMS

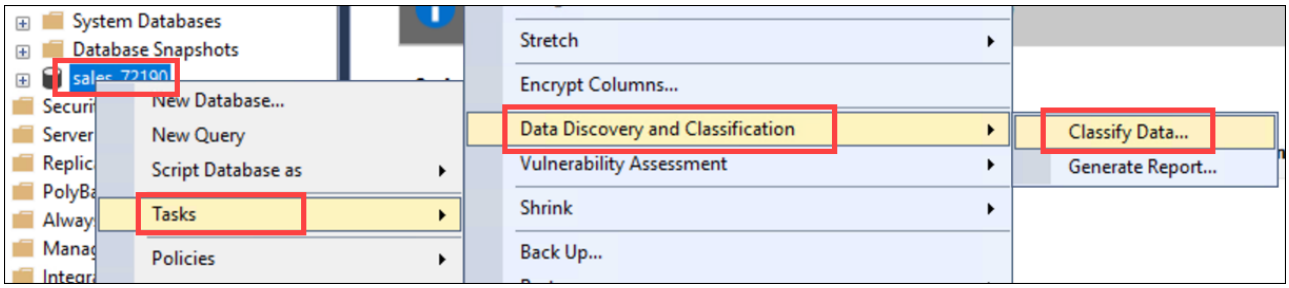
Contoso Auto has several databases that include tables containing sensitive data, such as personally identifiable information (PII) like phone numbers, social security numbers, financial data, etc. Since some of their personnel and customer data include individuals who reside within the European Union (EU), they need to adhere to the General Data Protection Regulation (GDPR) as well. Because of this, Contoso Auto is required to provide periodic data auditing reports to identify sensitive and GDPR-related data that reside within their various databases.

With SQL Server Management Studio, they are able to identify, classify, and generate reports on sensitive and GDPR-related data by using the SQL Data Discovery & Classification tool. This tool introduces a set of advanced services, forming a new SQL Information Protection paradigm aimed at protecting the data, not just the database:

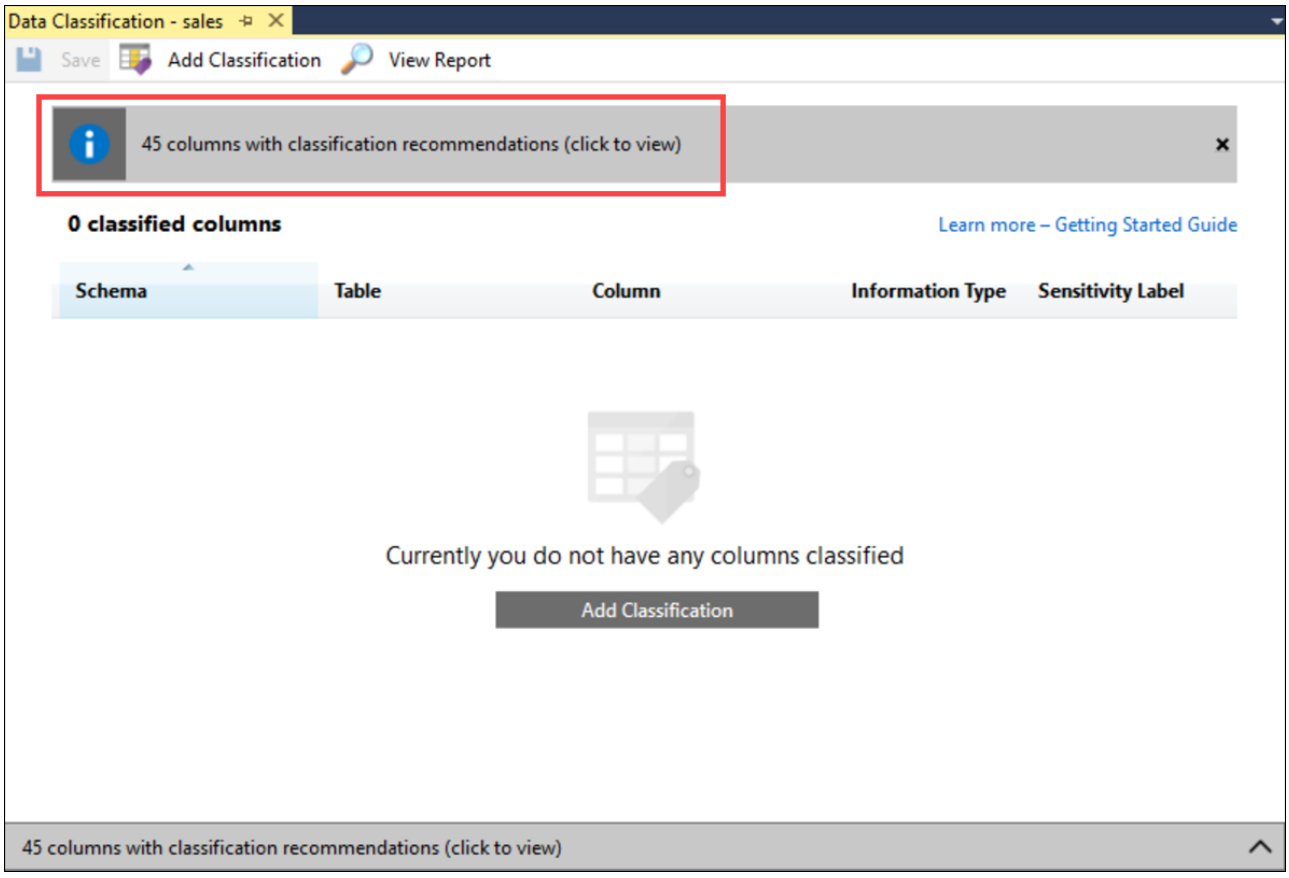
- **Discovery & recommendations** - The classification engine scans your database and identifies columns containing potentially sensitive data. It then provides you an easy way to review and apply the appropriate classification recommendations, as well as to manually classify columns.
- **Labeling** - Sensitivity classification labels can be persistently tagged on columns.
- **Visibility** - The database classification state can be viewed in a detailed report that can be printed/exported to be used for compliance & auditing purposes, as well as other needs.

In this exercise, you will run the SQL Data Discovery & Classification tool against their customer database, which includes personal, demographic, and sales data.

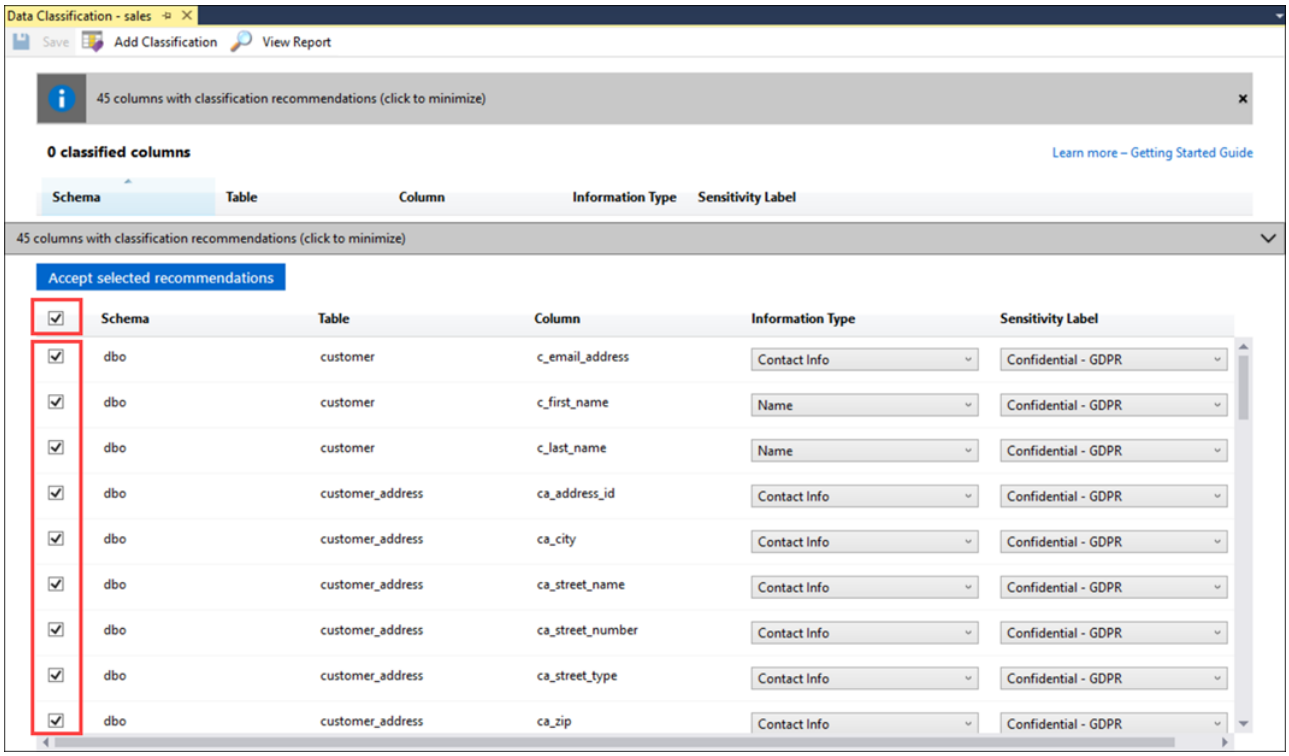
1. To get started, expand databases in the SQL Server Management Studio (SSMS) Object Explorer, right-click the `sales_XXXXX` database (where XXXXX is the unique identifier assigned to you for this workshop), and then choose **Tasks > Data Discovery and Classification > Classify Data...**.



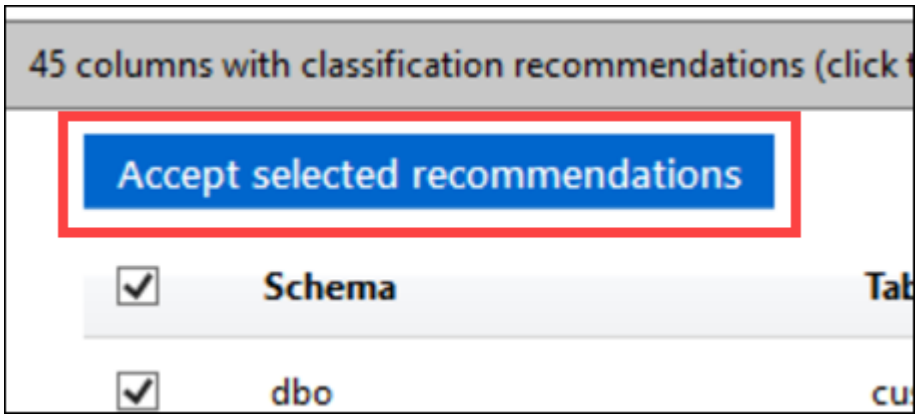
2. When the tool runs, it will analyze all of the columns within all of the tables and recommend appropriate data classifications for each. What you should see is the Data Classification dashboard showing no currently classified columns, and a classification recommendations box at the top showing that there are 45 columns that the tool identified as containing sensitive (PII) or GDPR-related data. **Click** on this classification recommendations box.



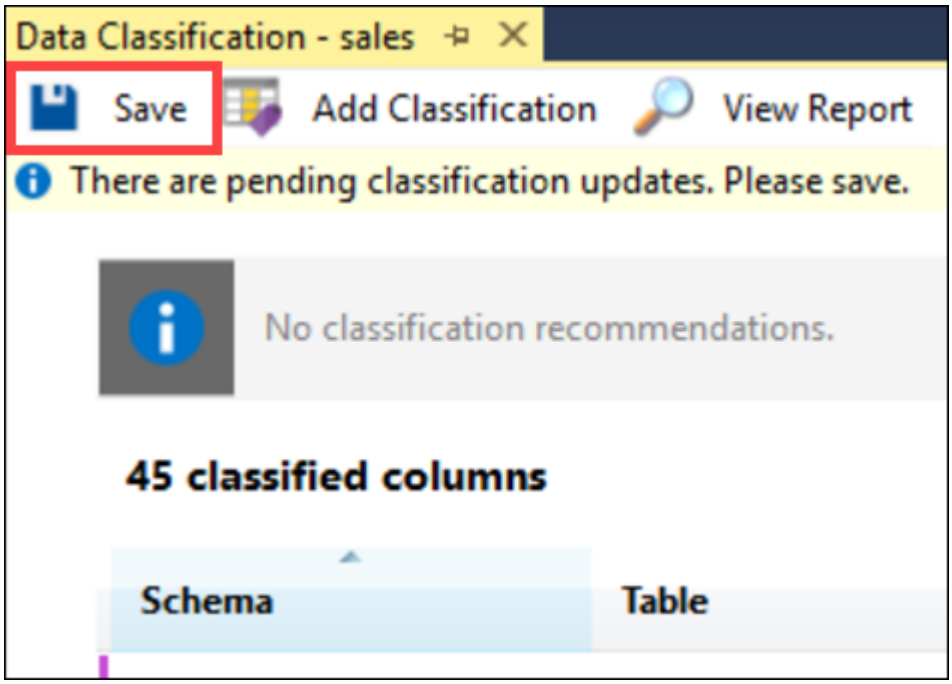
3. The list of recommendations displays the schema, table, column, type of information, and recommended sensitivity label for each identified column. You can change the information type and sensitivity labels for each if desired. In this case, accept all recommendations by **checking the checkbox** in the recommendations table header.



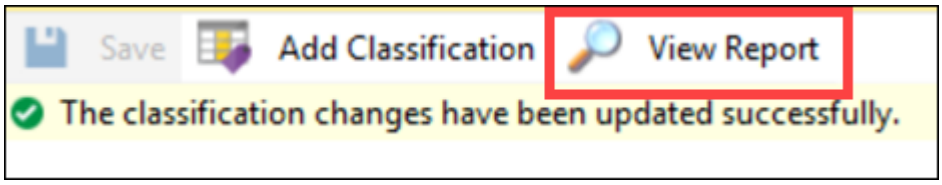
4. Click **Accept selected recommendations**.



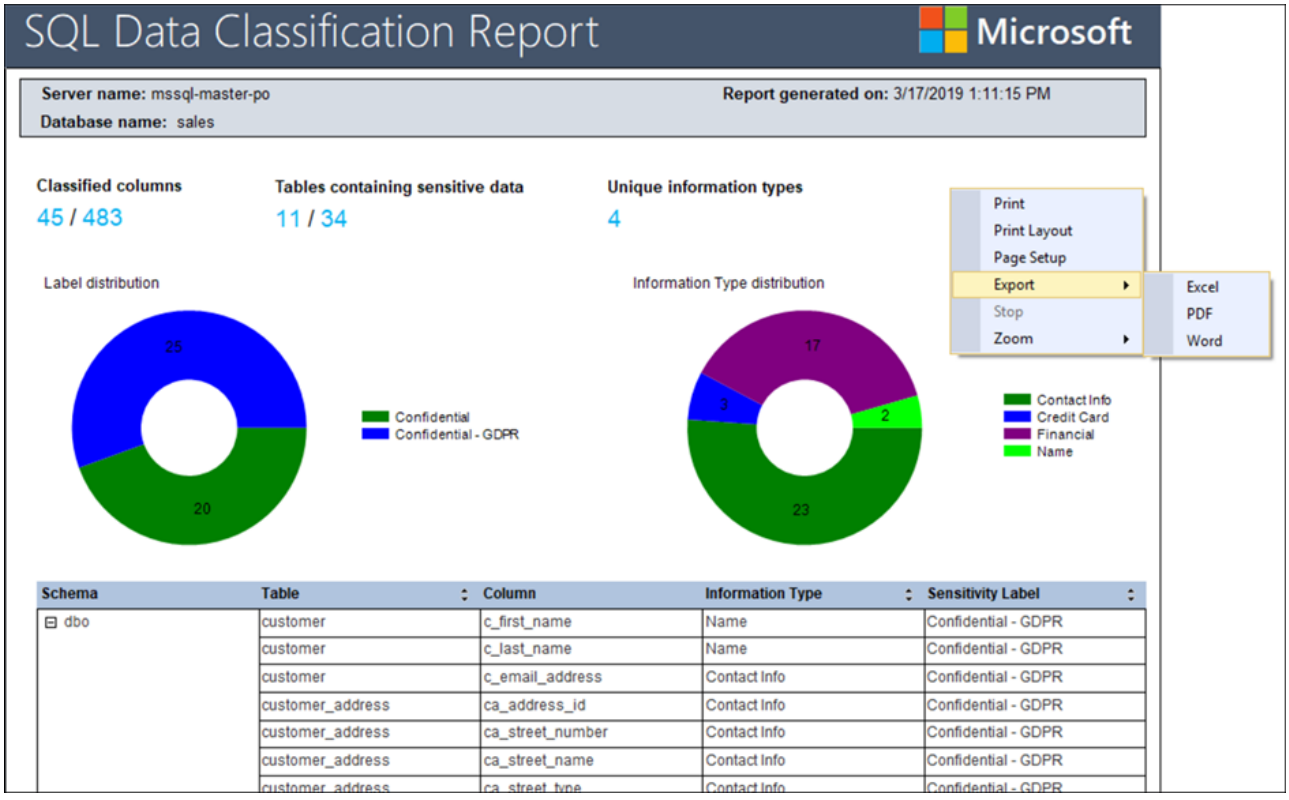
5. Click **Save** in the toolbar above to apply your changes.



6. After the changes are saved, click **View Report**.



7. What you should see is a report with a full summary of the database classification state. When you right-click on the report, you can see options to print or export the report in different formats.



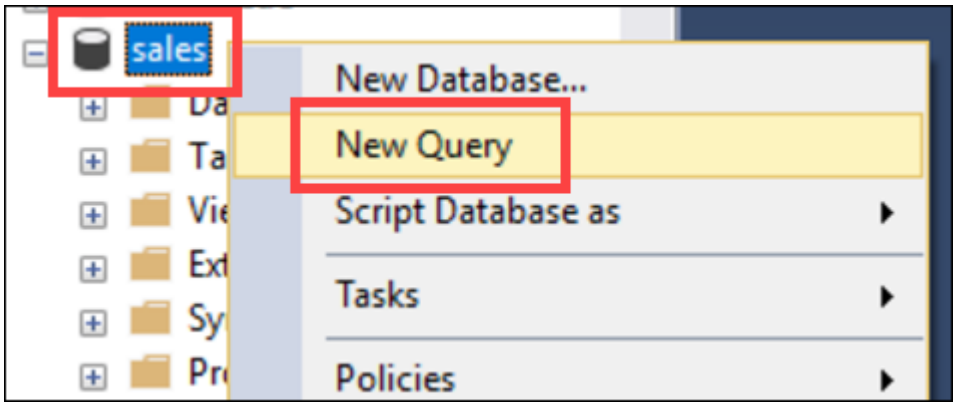
Task 4: Fix compliance issues with dynamic data masking

Some of the columns identified by the Data Discovery & Classification tool as containing sensitive (PII/GDPR) information include phone numbers, email addresses, billing addresses, and credit card numbers. One way to ensure compliance with various rules and regulations that enforce policies to protect such sensitive data is to prevent those who are not authorized from seeing it. An example would be displaying `XXX-XXX-XX95` instead of `123-555-2695` when outputting a phone number within a SQL query result, report, web page, etc. This is commonly called data masking. Traditionally, modifying systems and applications to implement data masking can be challenging. This is especially true when the masking has to apply all the way down to the data source level. Fortunately, SQL Server and its cloud-related product, Azure SQL Database, provides a feature named Dynamic Data Masking (DDM) to automatically protect this sensitive data from non-privileged users.

DDM helps prevent unauthorized access to sensitive data by enabling customers to designate how much of the sensitive data to reveal with minimal impact on the application layer. DDM can be configured on the database to hide sensitive data in the result sets of queries over designated database fields, while the data in the database is not changed. Dynamic Data Masking is easy to use with existing applications, since masking rules are applied in the query results. Many applications can mask sensitive data without modifying existing queries.

In this task, you will apply DDM to one of the database fields so you can see how to address the reported compliance issues. To test the data mask, you will create a test user and query the field as that user.

1. To get started, expand databases in the SQL Server Management Studio (SSMS) Object Explorer, right-click the `sales_XXXXX` database (where XXXXX is the unique identifier assigned to you for this workshop), and then select **New Query**.

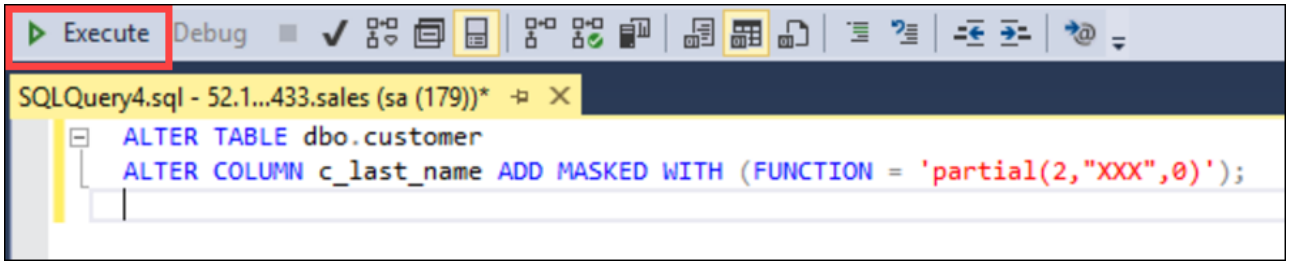


2. Add a dynamic data mask to the existing `dbo.customer.c_last_name` field by pasting the below query into the new query window:

```
ALTER TABLE dbo.customer
ALTER COLUMN c_last_name ADD MASKED WITH (FUNCTION = 'partial(2,"XXX",0)');
```

> The `partial` custom string masking method above exposes the first two characters and adds a custom padding string after for the remaining characters. The parameters are: `prefix,[padding],suffix`

3. Execute the query by selecting the **Execute** button in the SSMS toolbar, or pressing the *F5* key on your keyboard.



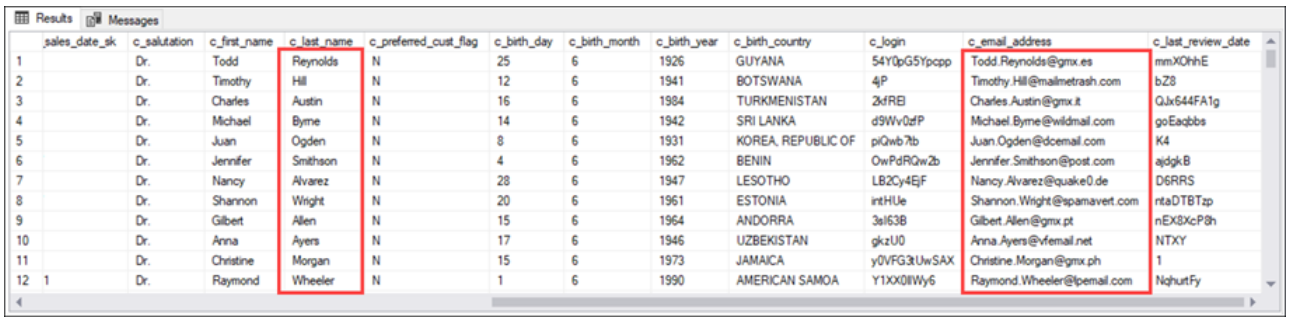
4. Clear the query window and replace the previous query with the following to add a dynamic data mask to the `dbo.customer.c_email_address` field:

```
ALTER TABLE dbo.customer
ALTER COLUMN c_email_address ADD MASKED WITH (FUNCTION = 'email()');
```

> The `email` masking method exposes the first letter of an email address and the constant suffix ".com", in the form of an email address: `aXXX@XXXX.com`.

5. Clear the query window and replace the previous query with the following, selecting all rows from the customer table:

```
SELECT * FROM dbo.customer
```



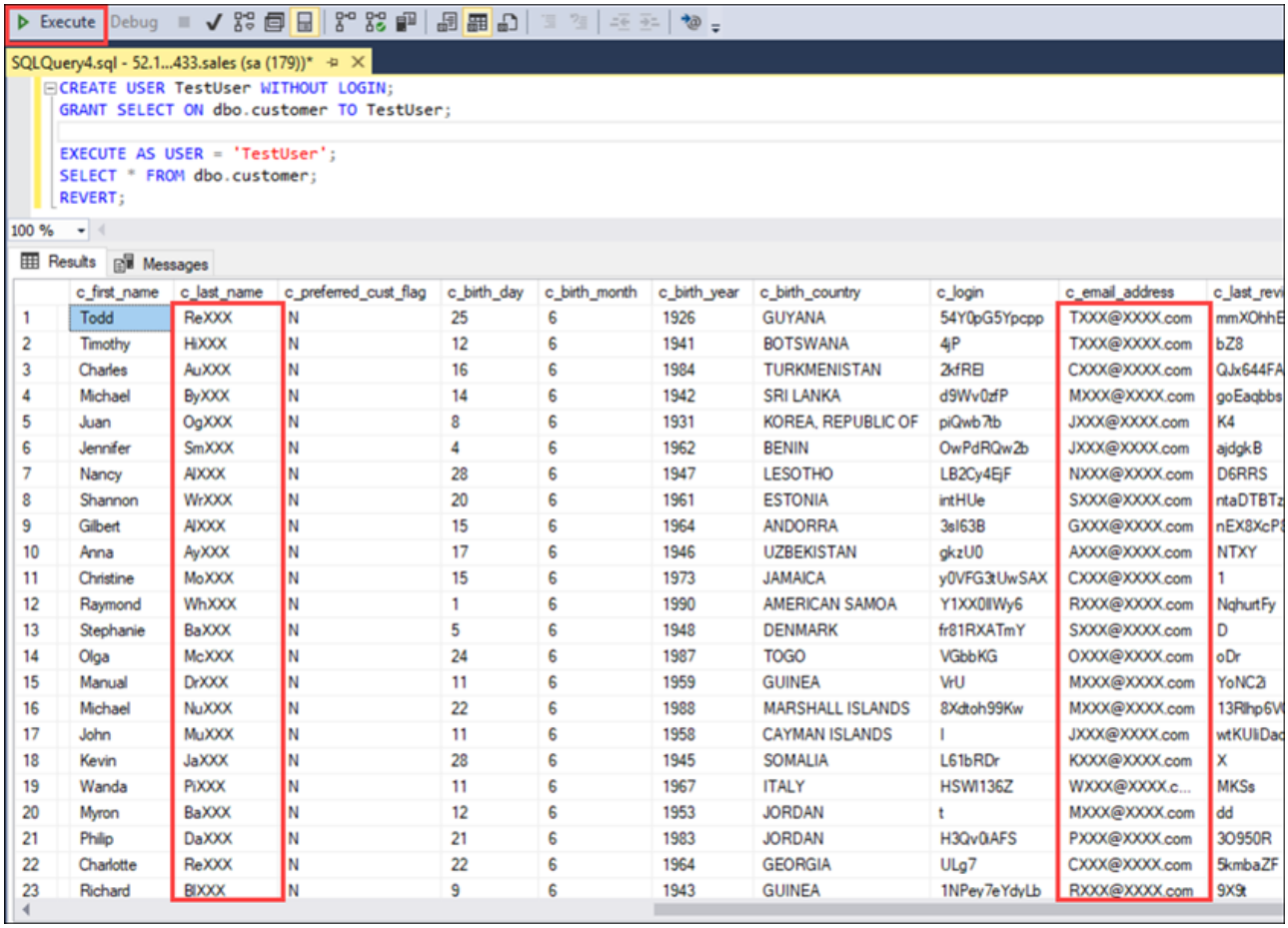
	sales_date_sk	c_salutation	c_first_name	c_last_name	c_preferred_cust_flag	c_birth_day	c_birth_month	c_birth_year	c_birth_country	c_login	c_email_address	c_last_review_date
1		Dr.	Todd	Reynolds	N	25	6	1926	GUYANA	54Y0pG5Ypcpp	Todd.Reynolds@gmx.es	mmXOhhE
2		Dr.	Timothy	Hill	N	12	6	1941	BOTSWANA	4jP	Timothy.Hill@mailmetrash.com	bZ8
3		Dr.	Charles	Austin	N	16	6	1984	TURKMENISTAN	2kfREl	Charles.Austin@gmx.it	QJx644FA1g
4		Dr.	Michael	Byrne	N	14	6	1942	SRI LANKA	d9Wv0zfP	Michael.Byrne@wildmail.com	goEaqbbs
5		Dr.	Juan	Ogden	N	8	6	1931	KOREA, REPUBLIC OF	piQwb7tb	Juan.Ogden@dcemail.com	K4
6		Dr.	Jennifer	Smithson	N	4	6	1962	BENIN	OwPdRQw2b	Jennifer.Smithson@post.com	ajdgkB
7		Dr.	Nancy	Alvarez	N	28	6	1947	LESOTHO	LB2Cy4EjF	Nancy.Alvarez@quake0.de	D6RRS
8		Dr.	Shannon	Wright	N	20	6	1961	ESTONIA	intHUe	Shannon.Wright@spamavert.com	ntaDTBTzp
9		Dr.	Gilbert	Allen	N	15	6	1964	ANDORRA	3sl63B	Gilbert.Allen@gmx.pt	nEX8XcP8h
10		Dr.	Anna	Ayers	N	17	6	1946	UZBEKISTAN	gkzU0	Anna.Ayers@vfemail.net	NTXY
11		Dr.	Christine	Morgan	N	15	6	1973	JAMAICA	y0VFG3tUwSAX	Christine.Morgan@gmx.ph	1
12	1	Dr.	Raymond	Wheeler	N	1	6	1990	AMERICAN SAMOA	Y1XX0lWy6	Raymond.Wheeler@pemail.com	NqhurtFy

6. Notice that the full last name and email address values are visible. That is because the user you are logged in as is a privileged user. Let's create a new user and execute the query again:

```
CREATE USER TestUser WITHOUT LOGIN;
GRANT SELECT ON dbo.customer TO TestUser;
```

```
EXECUTE AS USER = 'TestUser';
SELECT * FROM dbo.customer;
REVERT;
```

7. Run the query by selecting the **Execute** button. Observe that with the the `c_last_name` and `c_email_address` filed values are now masked, hiding the full value contained within the field and preventing expose of sensitive information.



	c_first_name	c_last_name	c_preferred_cust_flag	c_birth_day	c_birth_month	c_birth_year	c_birth_country	c_login	c_email_address	c_last_revi
1	Todd	ReXXX	N	25	6	1926	GUYANA	54Y0pG5Ypcpp	TXXX@XXXX.com	mmXOhhE
2	Timothy	HiXXX	N	12	6	1941	BOTSWANA	4jP	TXXX@XXXX.com	bZ8
3	Charles	AuXXX	N	16	6	1984	TURKMENISTAN	2kfREl	CXXX@XXXX.com	QJx644FA
4	Michael	ByXXX	N	14	6	1942	SRI LANKA	d9Wv0zfP	MXXX@XXXX.com	goEaqbbs
5	Juan	OgXXX	N	8	6	1931	KOREA, REPUBLIC OF	piQwb7tb	JXXX@XXXX.com	K4
6	Jennifer	SmXXX	N	4	6	1962	BENIN	OwPdRQw2b	JXXX@XXXX.com	ajdgkB
7	Nancy	AlXXX	N	28	6	1947	LESOTHO	LB2Cy4EjF	NXXX@XXXX.com	D6RRS
8	Shannon	WrXXX	N	20	6	1961	ESTONIA	intHUe	SXXX@XXXX.com	ntaDTBTz
9	Gilbert	AlXXX	N	15	6	1964	ANDORRA	3sl63B	GXXX@XXXX.com	nEX8XcP
10	Anna	AyXXX	N	17	6	1946	UZBEKISTAN	gkzU0	AXXX@XXXX.com	NTXY
11	Christine	MoXXX	N	15	6	1973	JAMAICA	y0VFG3tUwSAX	CXXX@XXXX.com	1
12	Raymond	WhXXX	N	1	6	1990	AMERICAN SAMOA	Y1XX0llWy6	RXXX@XXXX.com	NqhurtFy
13	Stephanie	BaXXX	N	5	6	1948	DENMARK	fr81RXATmY	SXXX@XXXX.com	D
14	Olga	McXXX	N	24	6	1987	TOGO	VGbbKG	OXXX@XXXX.com	oDr
15	Manual	DrXXX	N	11	6	1959	GUINEA	VrU	MXXX@XXXX.com	YoNC2i
16	Michael	NuXXX	N	22	6	1988	MARSHALL ISLANDS	8Xdtoh99Kw	MXXX@XXXX.com	13Rlhp6V
17	John	MuXXX	N	11	6	1958	CAYMAN ISLANDS	I	JXXX@XXXX.com	wtKUliDac
18	Kevin	JaXXX	N	28	6	1945	SOMALIA	L61bRDr	KXXX@XXXX.com	X
19	Wanda	PiXXX	N	11	6	1967	ITALY	HSWI136Z	WXXX@XXXX.c...	MKSs
20	Myron	BaXXX	N	12	6	1953	JORDAN	t	MXXX@XXXX.com	dd
21	Philip	DaXXX	N	21	6	1983	JORDAN	H3Qv0iAFS	PXXX@XXXX.com	3O950R
22	Charlotte	ReXXX	N	22	6	1964	GEORGIA	ULg7	CXXX@XXXX.com	5kmbaZF
23	Richard	BlXXX	N	9	6	1943	GUINEA	1NPey7eYdyLb	RXXX@XXXX.com	9X9t

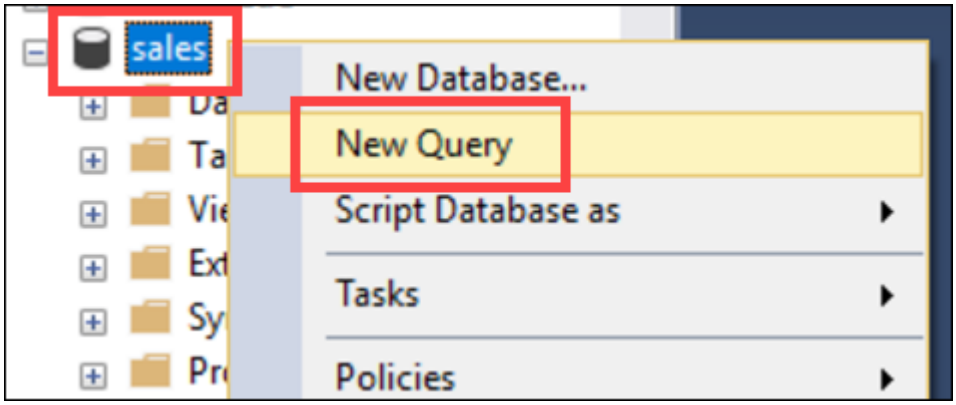
Task 5: Restrict data access with Row-level security

Row-Level Security (RLS) enables you to use group membership or execution context to control access to rows in a database table. RLS simplifies the design and coding of security in your application by helping you implement restrictions on data row access. For example, a hospital can create a security policy that allows doctors to view data rows for their patients only. Another example is a multi-tenant application can create a policy to enforce a logical separation of each tenant's data rows from every other tenant's rows. Efficiencies are achieved by the storage of data for many tenants in a single table. Each tenant can see only its data rows.

The access restriction logic is located in the database tier rather than away from the data in another application tier. The database system applies the access restrictions every time that data access is attempted from any tier. This makes your security system more reliable and robust by reducing the surface area of your security system.

In this task, you will apply row-level security to one of the database tables and run some queries using multiple user contexts to see how data access is restricted.

1. To get started, expand databases in the SQL Server Management Studio (SSMS) Object Explorer, right-click the `sales_XXXXX` database (where XXXXX is the unique identifier assigned to you for this workshop), and then select **New Query**.



2. Within the `store` table, there are three stores located in Kentucky. Using RLS, we will restrict access to sales records so that user can only see sales from their own store, while the regional manager for Kentucky will be able to see all sales data for the state. In the new query window, enter the following SQL commands to create users that will be used for querying data secured by RLS with different contexts, and then grant them `SELECT` permissions on the `dbo.store_sales` table. Run the query by selecting **Execute** on the SSMS toolbar.

```
CREATE USER ManagerKY WITHOUT LOGIN;
CREATE USER Elmwood WITHOUT LOGIN;
CREATE USER Egypt WITHOUT LOGIN;
CREATE USER Bedford WITHOUT LOGIN;
GO

GRANT SELECT ON dbo.store_sales TO ManagerKY;
GRANT SELECT ON dbo.store_sales TO Elmwood;
GRANT SELECT ON dbo.store_sales TO Egypt;
GRANT SELECT ON dbo.store_sales TO Bedford;
GO
```

> Row-level security will be applied to the `dbo.store_sales` table, and you will run queries against that to observe how RLS affects the results displayed for users with different contexts.

3. Next, you will create a new schema, and an inline table-valued function that will be used to apply the RLS policy. The function returns 1 when a row in the `ss_store_sk` column is the same as the store for the user executing the query (`s.s_store_sk = @StoreId`) and the store name is the same of the user's store (`s.s_store_name = USER_NAME()`), or if the user executing the query is the Manager user

(`USER_NAME() = 'ManagerKY'`). Create a new query window and paste the following SQL script into the window, and then select **Execute**.

```
CREATE SCHEMA Security;
GO

CREATE FUNCTION Security.fn_securitypredicate(@StoreId AS INT)
  RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN (SELECT 1 AS fn_securitypredicate_result
   FROM dbo.store_sales ss
      INNER JOIN dbo.store s ON ss.ss_store_sk = s.s_store_sk
   WHERE s.s_state = 'KY'
   AND s.s_store_sk = @StoreId
   AND (s.s_store_name = USER_NAME() OR USER_NAME() = 'ManagerKY'));
GO
```

> RLS is implement by using the `CREATE SECURITY POLICY` Transact-SQL statement, and predicates created as inline table-valued functions.

4. The next step is to create a security policy by adding the function as a filter predicate. The state must be set to ON to enable the policy. Create a new query window and paste the following SQL script into the window, and then select **Execute**.

```
CREATE SECURITY POLICY SalesFilter
ADD FILTER PREDICATE Security.fn_securitypredicate(ss_store_sk)
ON dbo.store_sales
WITH (STATE = ON);
GO
```

> RLS filter predicates are functionally equivalent to appending a `WHERE` clause to a query against the target table. The predicate can be as sophisticated as business practices dictate, or the clause can be as simple as `WHERE TenantId = 42`.

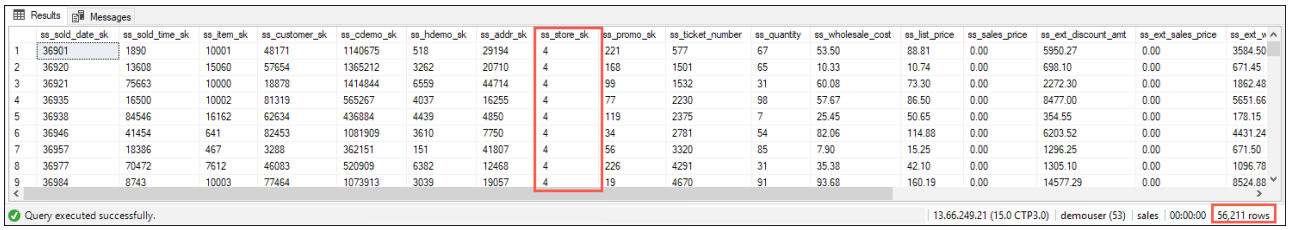
5. Allow `SELECT` permissions to the `fn_securitypredicate` function by executing the following query in a new query window.

```
GRANT SELECT ON security.fn_securitypredicate TO ManagerKY;
GRANT SELECT ON security.fn_securitypredicate TO Elmwood;
GRANT SELECT ON security.fn_securitypredicate TO Egypt;
GRANT SELECT ON security.fn_securitypredicate TO Bedford;
GO
```

6. Now test the filtering predicate, by selecting from the `store_sales` table as each user. Paste each of the queries below into a new query window and run each the commands. Observe the count of records

returned, as well as the data in the `ss_store_sk` column.

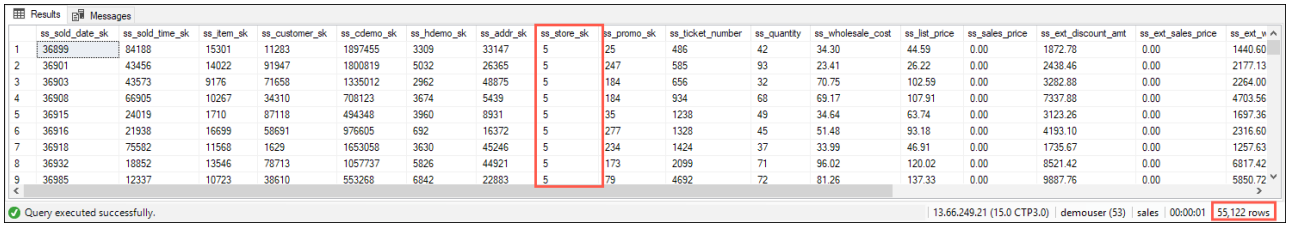
```
EXECUTE AS USER = 'Elmwood';
SELECT * FROM dbo.store_sales;
REVERT;
```



	ss_sold_date_sk	ss_sold_time_sk	ss_item_sk	ss_customer_sk	ss_cdemo_sk	ss_hdemo_sk	ss_addr_sk	ss_store_sk	ss_promo_sk	ss_ticket_number	ss_quantity	ss_wholesale_cost	ss_list_price	ss_sales_price	ss_ext_discount_amt	ss_ext_sales_price	ss_ext_w
1	36901	1890	10001	48171	1140675	518	29194	4	221	577	67	53.50	88.81	0.00	5950.27	0.00	3584.50
2	36920	13608	15060	57654	1365212	3262	20710	4	168	1501	65	10.33	10.74	0.00	698.10	0.00	671.45
3	36921	75663	10000	18878	1414844	6559	44714	4	99	1532	31	60.08	73.30	0.00	2272.30	0.00	1862.48
4	36935	16500	10002	81319	565267	4037	16255	4	77	2230	98	57.67	86.50	0.00	8477.00	0.00	5651.66
5	36938	84546	16162	62634	436884	4439	4850	4	119	2375	7	25.45	50.65	0.00	354.55	0.00	178.15
6	36946	41454	641	82453	1081909	3610	7750	4	34	2781	54	82.06	114.88	0.00	6203.52	0.00	4431.24
7	36957	18386	467	3288	362151	151	41807	4	56	3320	85	7.90	15.25	0.00	1296.25	0.00	671.50
8	36977	70472	7612	46083	520909	6382	12468	4	226	4291	31	35.38	42.10	0.00	1305.10	0.00	1096.78
9	36984	8743	10003	77464	1073913	3039	19057	4	19	4670	91	93.68	160.19	0.00	14577.29	0.00	8524.88

Query executed successfully. | 13.66.249.21 (15.0 CTP3.0) | demouser (53) | sales | 00:00:00 | 56,211 rows

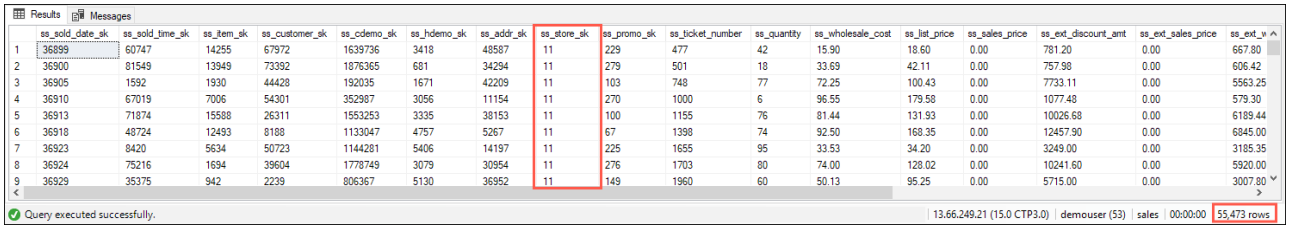
```
EXECUTE AS USER = 'Egypt';
SELECT * FROM dbo.store_sales;
REVERT;
```



	ss_sold_date_sk	ss_sold_time_sk	ss_item_sk	ss_customer_sk	ss_cdemo_sk	ss_hdemo_sk	ss_addr_sk	ss_store_sk	ss_promo_sk	ss_ticket_number	ss_quantity	ss_wholesale_cost	ss_list_price	ss_sales_price	ss_ext_discount_amt	ss_ext_sales_price	ss_ext_w
1	36899	84188	15301	11283	1897455	3309	33147	5	25	486	42	34.30	44.59	0.00	1872.78	0.00	1440.60
2	36901	43456	14022	91947	1800819	5032	26365	5	247	585	93	23.41	26.22	0.00	2438.46	0.00	2177.13
3	36903	43573	9176	71658	1335012	2962	48875	5	184	656	32	70.75	102.59	0.00	3282.88	0.00	2264.00
4	36908	66905	10267	34310	708123	3674	5439	5	184	934	68	69.17	107.91	0.00	7337.88	0.00	4703.56
5	36915	24019	1710	87118	494348	3960	8931	5	35	1238	49	34.64	63.74	0.00	3123.26	0.00	1697.36
6	36916	21938	16699	58691	976605	692	16372	5	277	1328	45	51.48	93.18	0.00	4193.10	0.00	2316.60
7	36918	75582	11568	1629	1653058	3630	45246	5	234	1424	37	33.99	46.91	0.00	1735.67	0.00	1257.63
8	36932	18852	13546	78713	1057737	5826	44921	5	173	2099	71	96.02	120.02	0.00	8521.42	0.00	6817.42
9	36985	12337	10723	38610	553268	6842	22883	5	79	4692	72	81.26	137.33	0.00	9887.76	0.00	5850.72

Query executed successfully. | 13.66.249.21 (15.0 CTP3.0) | demouser (53) | sales | 00:00:01 | 55,122 rows

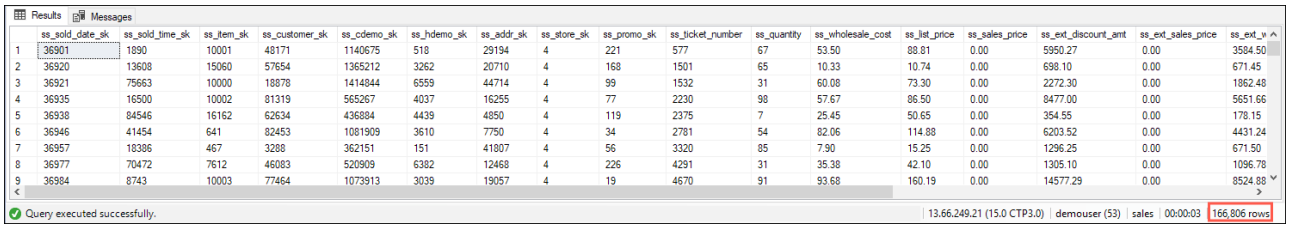
```
EXECUTE AS USER = 'Bedford';
SELECT * FROM dbo.store_sales;
REVERT;
```



	ss_sold_date_sk	ss_sold_time_sk	ss_item_sk	ss_customer_sk	ss_cdemo_sk	ss_hdemo_sk	ss_addr_sk	ss_store_sk	ss_promo_sk	ss_ticket_number	ss_quantity	ss_wholesale_cost	ss_list_price	ss_sales_price	ss_ext_discount_amt	ss_ext_sales_price	ss_ext_w
1	36899	60747	14255	67972	1639736	3418	48587	11	229	477	42	15.90	18.60	0.00	781.20	0.00	667.80
2	36900	81549	13949	73392	1876365	681	34294	11	279	501	18	33.69	42.11	0.00	757.98	0.00	606.42
3	36905	1592	1930	44428	192035	1671	42209	11	103	748	77	72.25	100.43	0.00	7733.11	0.00	5563.25
4	36910	67019	7006	54301	352987	3056	11154	11	270	1000	6	96.55	179.58	0.00	1077.48	0.00	579.30
5	36913	71874	15588	26311	1553253	3335	38153	11	100	1155	76	81.44	131.93	0.00	10026.68	0.00	6189.44
6	36918	48724	12493	8188	1133047	4757	5267	11	67	1398	74	92.50	168.35	0.00	12457.90	0.00	6845.00
7	36923	8420	5634	50723	1144281	5406	14197	11	225	1655	95	33.53	34.20	0.00	3249.00	0.00	3185.35
8	36924	75216	1694	39604	1778749	3079	30954	11	276	1703	80	74.00	128.02	0.00	10241.60	0.00	5920.00
9	36929	35375	942	2239	806367	5130	36952	11	149	1960	60	50.13	95.25	0.00	5715.00	0.00	3007.80

Query executed successfully. | 13.66.249.21 (15.0 CTP3.0) | demouser (53) | sales | 00:00:00 | 55,473 rows

```
EXECUTE AS USER = 'ManagerKY';
SELECT * FROM dbo.store_sales;
REVERT;
```



	ss_sold_date_sk	ss_sold_time_sk	ss_item_sk	ss_customer_sk	ss_cdemo_sk	ss_hdemo_sk	ss_addr_sk	ss_store_sk	ss_promo_sk	ss_ticket_number	ss_quantity	ss_wholesale_cost	ss_list_price	ss_sales_price	ss_ext_discount_amt	ss_ext_sales_price	ss_ext_w
1	36901	1890	10001	48171	1140675	518	29194	4	221	577	67	53.50	88.81	0.00	5950.27	0.00	3584.50
2	36920	13608	15060	57654	1365212	3262	20710	4	168	1501	65	10.33	10.74	0.00	698.10	0.00	671.45
3	36921	75663	10000	18878	1414844	6559	44714	4	99	1532	31	60.08	73.30	0.00	2272.30	0.00	1862.48
4	36935	16500	10002	81319	565267	4037	16255	4	77	2230	98	57.67	86.50	0.00	8477.00	0.00	5651.66
5	36938	84546	16162	62634	436884	4439	4850	4	119	2375	7	25.45	50.65	0.00	354.55	0.00	178.15
6	36946	41454	641	82453	1081909	3610	7750	4	34	2781	54	82.06	114.88	0.00	6203.52	0.00	4431.24
7	36957	18386	467	3288	362151	151	41807	4	56	3320	85	7.90	15.25	0.00	1296.25	0.00	671.50
8	36977	70472	7612	46083	520909	6382	12468	4	226	4291	31	35.38	42.10	0.00	1305.10	0.00	1096.78
9	36984	8743	10003	77464	1073913	3039	19057	4	19	4670	91	93.68	160.19	0.00	14577.29	0.00	8524.88

Query executed successfully. | 13.66.249.21 (15.0 CTP3.0) | demouser (53) | sales | 00:00:03 | 166,806 rows

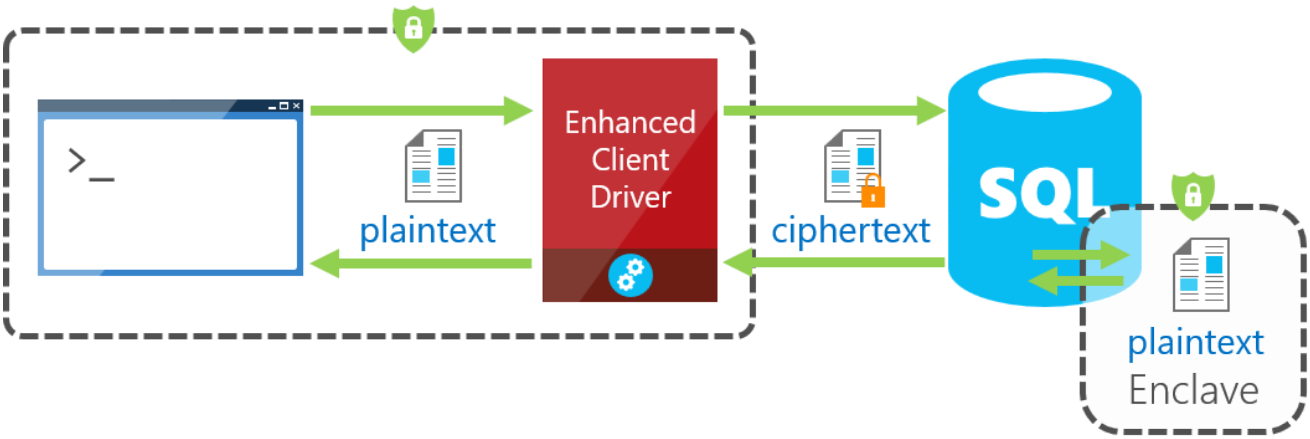
> As you can see within each set of results, users in a Store context see only the data for their store, while queries run within the Manager context see data for all three stores. Row-Level Security makes it easy to implement filtering to restrict access to data within SQL Server tables.

Task 6: Always Encrypted with secure enclaves

Always Encrypted is a feature designed to protect sensitive data, such as credit card numbers or national identification numbers (for example, U.S. social security numbers), stored in SQL Server databases. It allows clients to encrypt sensitive data inside client applications and never reveal the encryption keys to the Database Engine. As a result, Always Encrypted provides a separation between those who own the data (and can view it) and those who manage the data (but should have no access). By ensuring on-premises database administrators, cloud database operators, or other high-privileged, but unauthorized users, cannot access the encrypted data, Always Encrypted enables customers to confidently store sensitive data outside of their direct control.

Always Encrypted protects data by encrypting it on the client side and never allowing the data or the corresponding cryptographic keys to appear in plaintext inside the SQL Server Engine. As a result, the functionality on encrypted columns inside the database is severely restricted. The only operations SQL Server could perform on encrypted data were equality comparisons (and equality comparisons were only available with deterministic encryption). All other operations, including cryptographic operations (initial data encryption or key rotation), or rich computations (for example, pattern matching) were not supported inside the database. Users needed to move the data outside of the database to perform these operations on the client-side.

Always Encrypted with secure enclaves addresses these limitations by allowing computations on plaintext data inside a secure enclave on the server side. A secure enclave is a protected region of memory within the SQL Server process, and acts as a trusted execution environment for processing sensitive data inside the SQL Server engine. A secure enclave appears as a black box to the rest of the SQL Server and other processes on the hosting machine. There is no way to view any data or code inside the enclave from the outside, even with a debugger.



When parsing an application's query, the SQL Server Engine determines if the query contains any operations on encrypted data that require the use of the secure enclave. For queries where the secure enclave needs to be accessed:

- The client driver sends the column encryption keys required for the operations to the secure enclave (over a secure channel).

- Then, the client driver submits the query for execution along with the encrypted query parameters.

During query processing, the data or the column encryption keys are not exposed in plaintext in the SQL Server Engine outside of the secure enclave. The SQL Server engine delegates cryptographic operations and computations on encrypted columns to the secure enclave. If needed, the secure enclave decrypts the query parameters and/or the data stored in encrypted columns and performs the requested operations.

Configure a secure enclave

In this task, you will use SSMS to verify that Always Encrypted with secure enclaves is loaded on the SQL Server instance.

> **NOTE**: When you use SSMS to configure Always Encrypted, SSMS handles both Always Encrypted keys and sensitive data, so both the keys and the data appear in plaintext inside the SSMS process. As the primary goal of Always Encrypted is to ensure encrypted sensitive data is safe even if the database system gets compromised, this approach is recommended only for development and non-production environments.

1. Return to SSMS, open a new query window, and verify Always Encrypted with secure enclaves is loaded by running the following query:

```
SELECT [name], [value], [value_in_use] FROM sys.configurations
WHERE [name] = 'column encryption enclave type';
```

2. The query should return the following result:

name	value	value_in_use
column encryption enclave type	1	1

3. Next, enable rich computations on encrypted columns by running the following query:

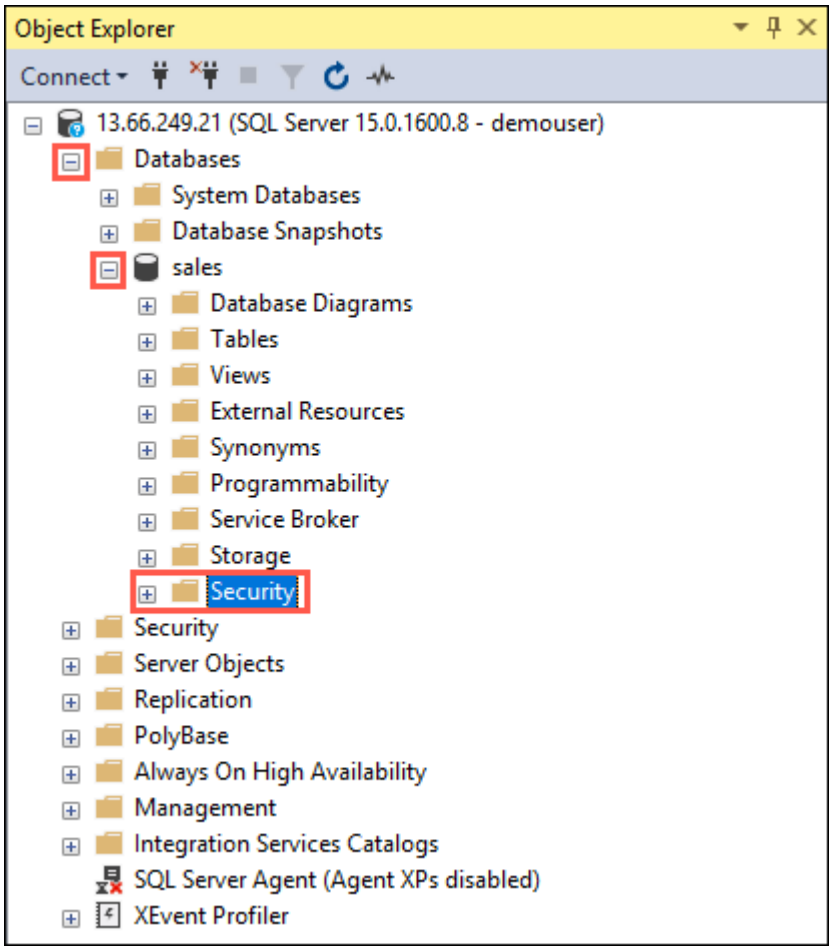
```
DBCC traceon(127,-1)
```

> **NOTE**: Rich computations are disabled by default in SQL Server 2019 preview. They need to be enabled using the above statement after each restart of your SQL Server instance.

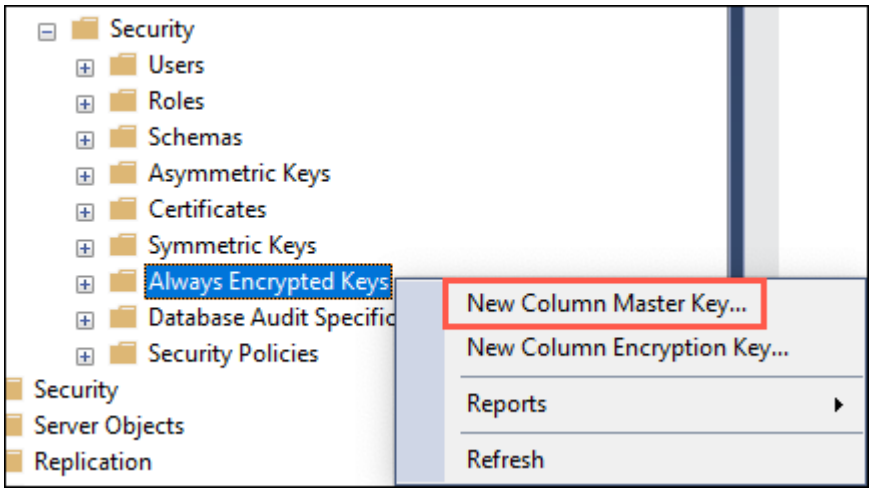
Provision enclave-enabled keys

In this step, you will create a column master key and a column encryption key that allow enclave computations.

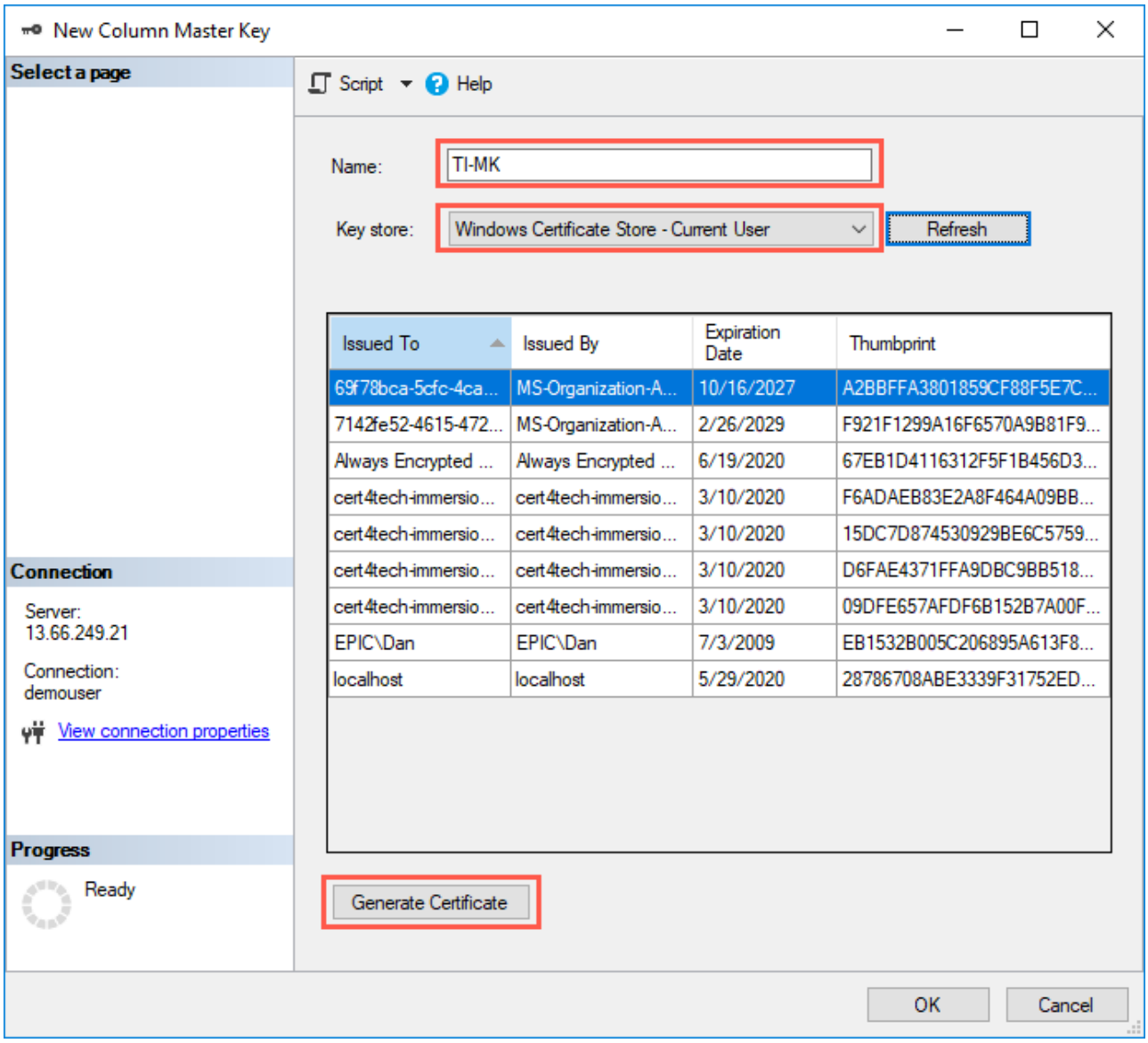
1. Expand databases in the SQL Server Management Studio (SSMS) Object Explorer, then expand the `sales_XXXXX` database (where XXXXX is the unique identifier assigned to you for this workshop), and locate the **Security** folder under the database.



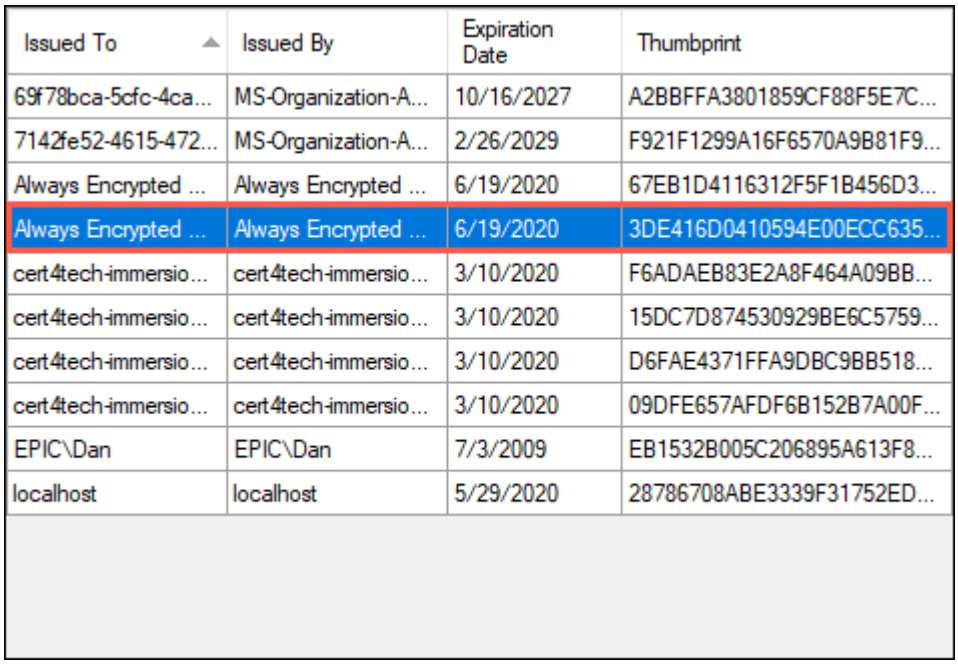
2. Expand the **Security** folder, right-click **Always Encrypted Keys**, and then select **New Column Master Key...**"



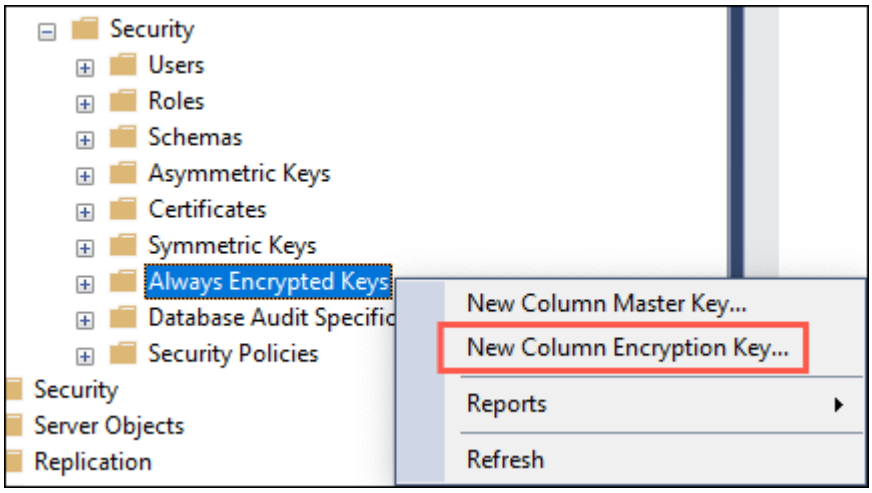
3. In the New Column Master Key dialog, enter **TI-MK** as the name, ensure the Key store is set to **Windows Certificate Store - Current User**, and then select **Generate Certificate**.



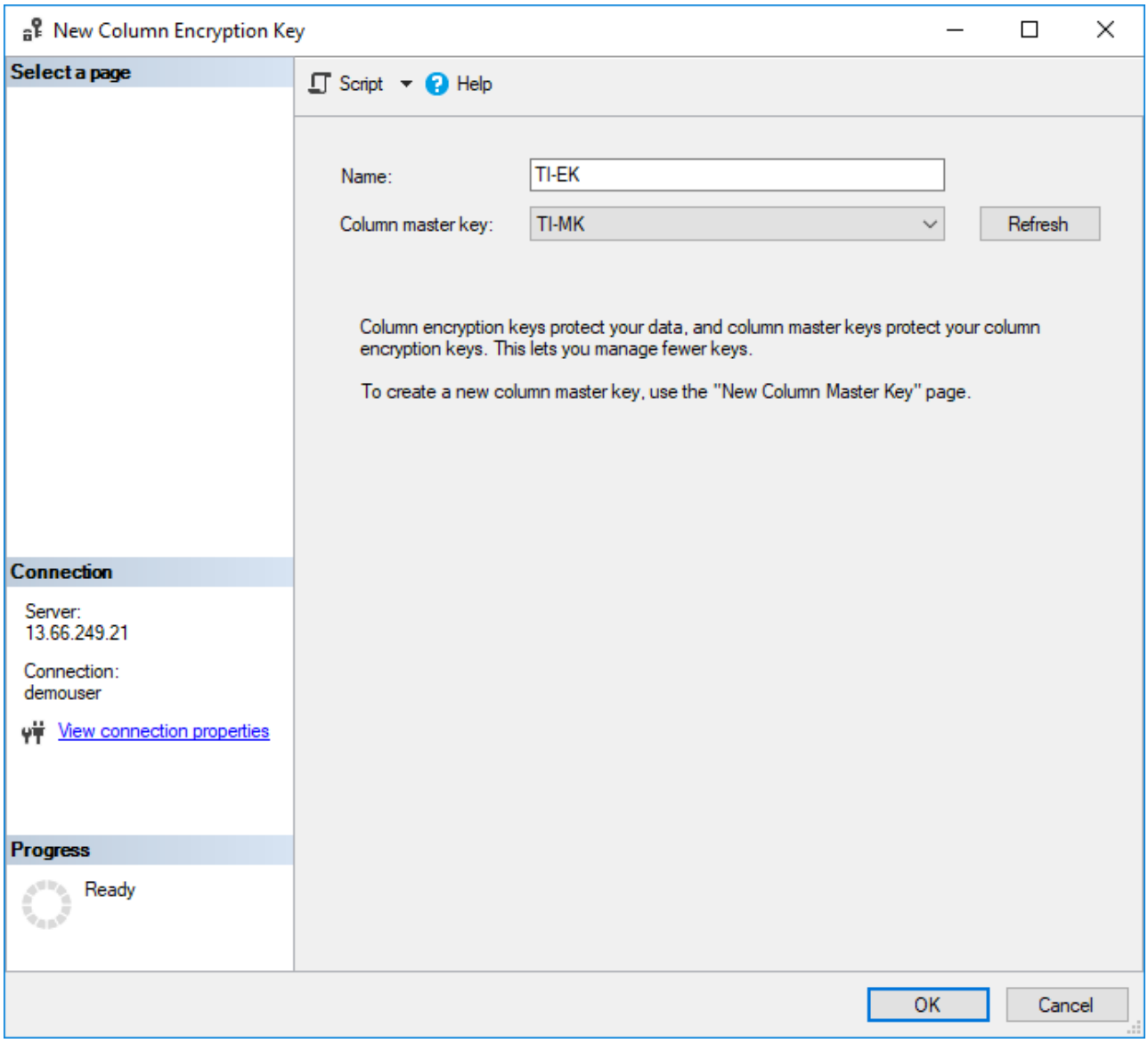
4. Ensure the generated key is highlighted in the list of keys, and then select **OK** in the New Column Master Key dialog.



5. Right-click on **Always Encrypted Keys** again, and this time select **New Column Encryption Key...**



6. In the **New Column Encryption Key** dialog, enter **TI-EK** for the name, select **TI-MK** for the Column master key, and select **OK**.



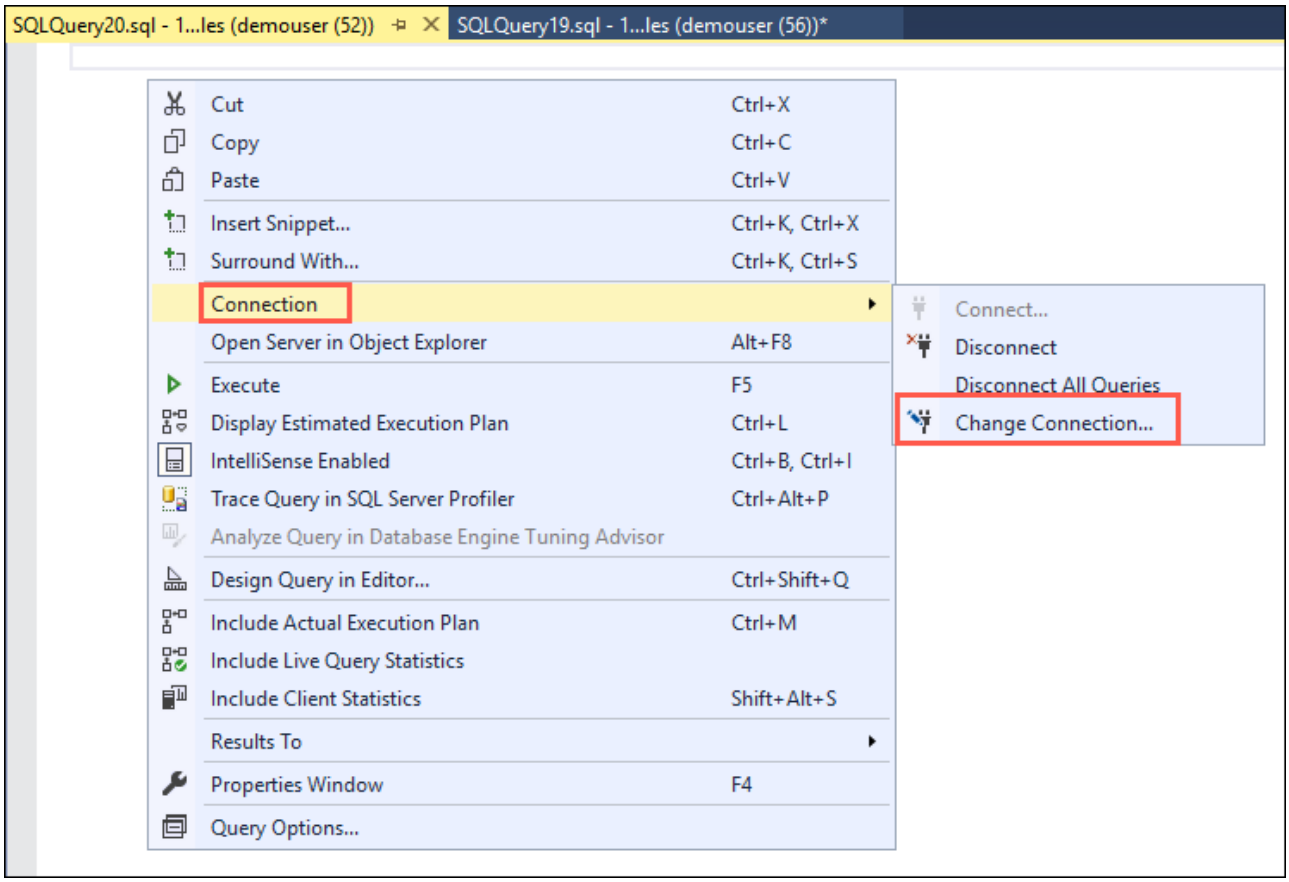
Encrypt customer email column

In this step, you will encrypt the data stored in the email column inside the server-side enclave, and then test a SELECT query of the data. For testing purposes, you will open two new query windows. One of the query

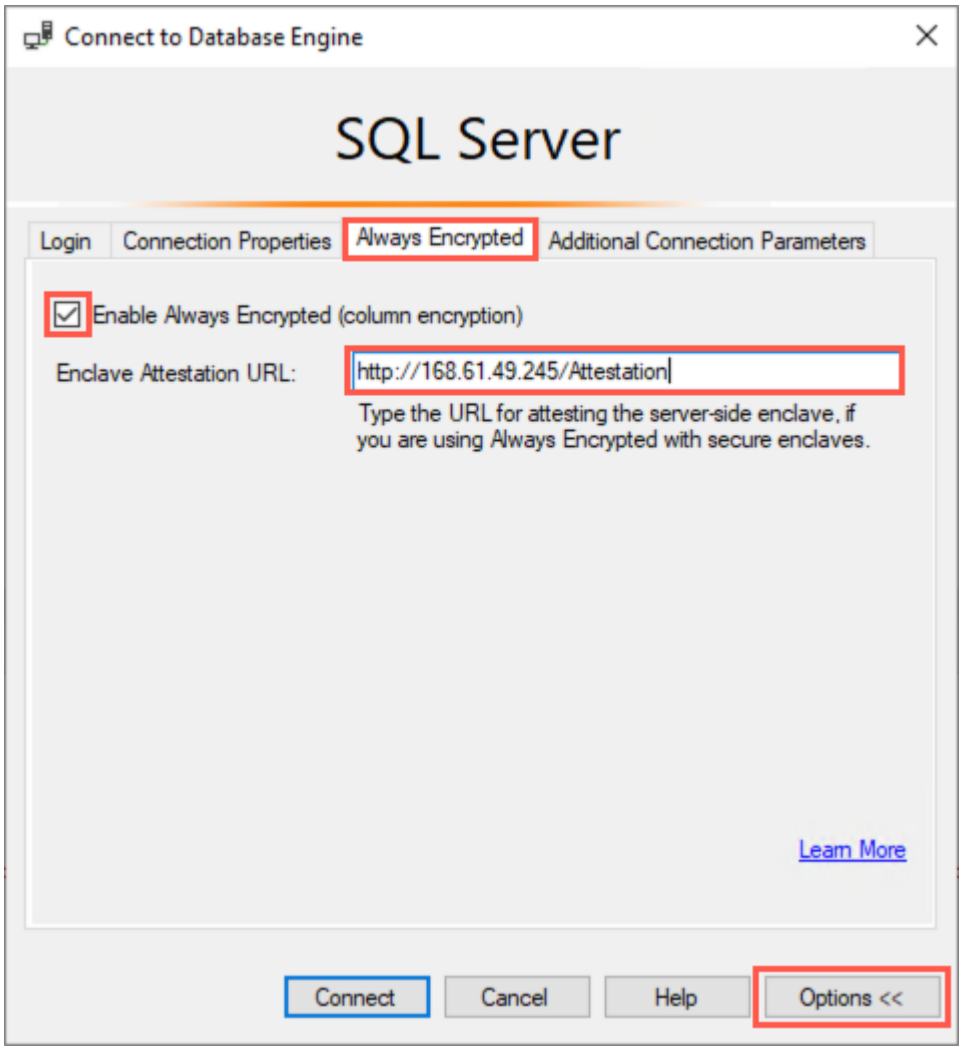
windows will be configured with Always Encrypted enabled and the other will have it disabled, so you can see the results of running with Always Encrypted with secure enclaves.

1. In SSMS, select the `sales` database in the Object Explorer, and then select **New query** in the SSMS toolbar to open a new query window.

2. Right-click anywhere in the new query window, and select **Connection > Change Connection**.

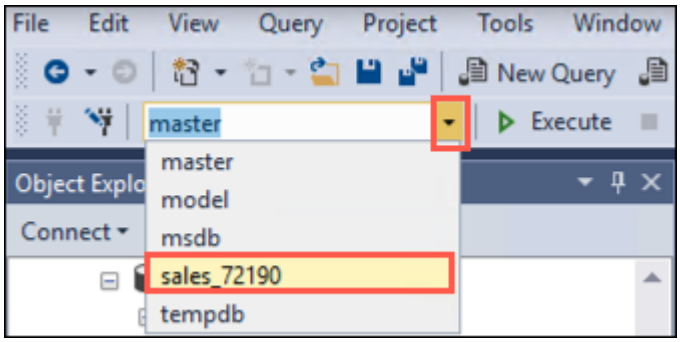


3. In the Connect to Database Engine dialog, select **Options**, navigate to the **Always Encrypted** tab, then check the **Enable Always Encrypted** box and enter the `AttestationServerUrl` from the value sheet you were provided for this lab into the Enclave Attestation URL box. For example, `http://168.61.49.245/Attestation`.



4. Select **Connect**.

5. Now, on the SSMS toolbar, change the database back to your **sales_XXXXX** database by selecting the drop down arrow next to **master** and selecting the **sales_XXXXX** database ((where XXXX is the unique identifier assigned to you for this lab).



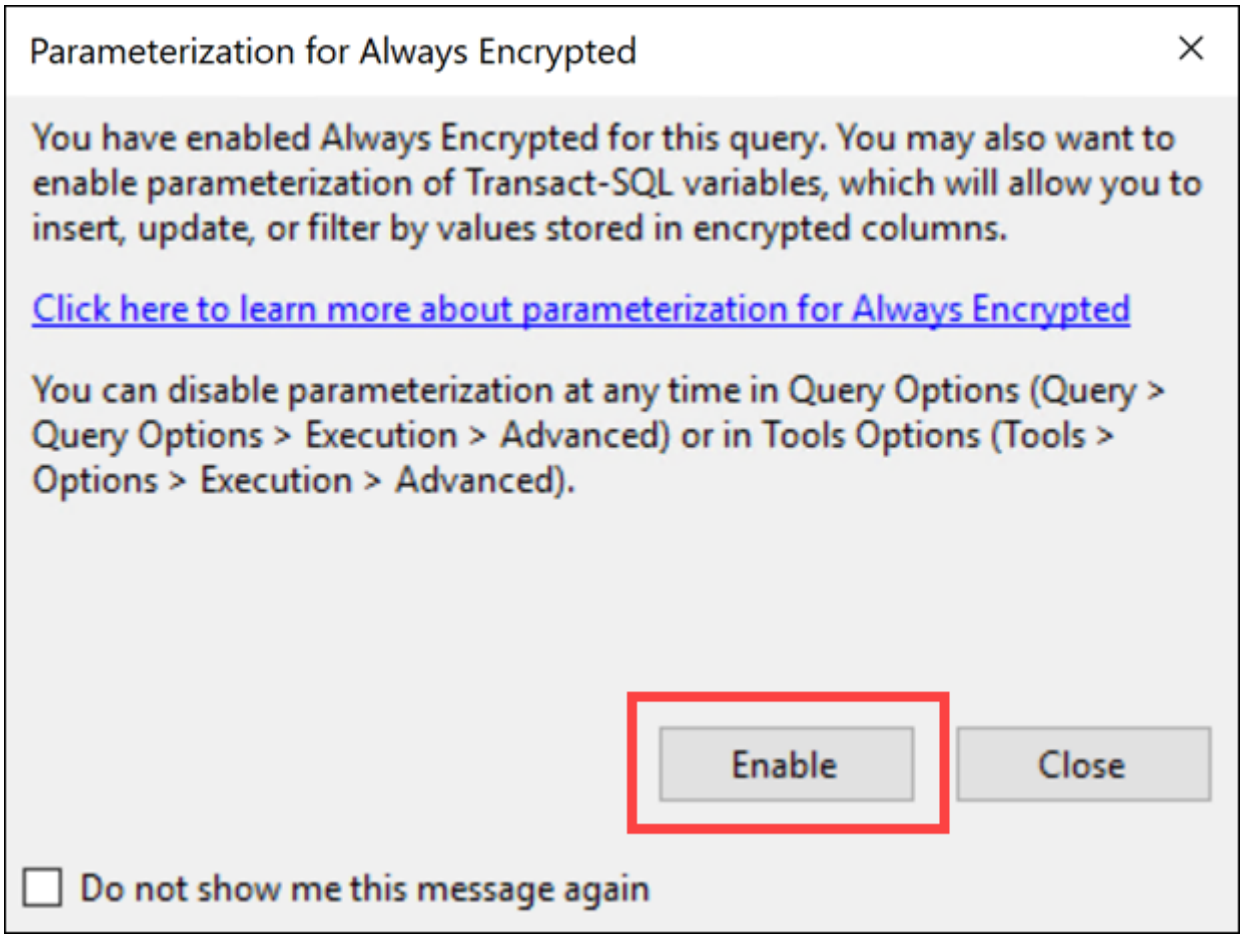
6. In the query window, copy and paste the following query to encrypt the `c_email_address` column in the `dbo.customer` table.

```
DROP INDEX [cci_customer] ON [dbo].[customer]
GO

ALTER TABLE [dbo].[customer]
ALTER COLUMN [c_email_address] [char](50) COLLATE Latin1_General_BIN2
```

```
ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = [TI-EK], ENCRYPTION_TYPE =
Randomized, ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256') NULL
WITH (ONLINE = ON);

ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;

CREATE CLUSTERED COLUMNSTORE INDEX [cci_customer] ON [dbo].[customer] WITH
(DROP_EXISTING = OFF, COMPRESSION_DELAY = 0) ON [PRIMARY]
GO
```

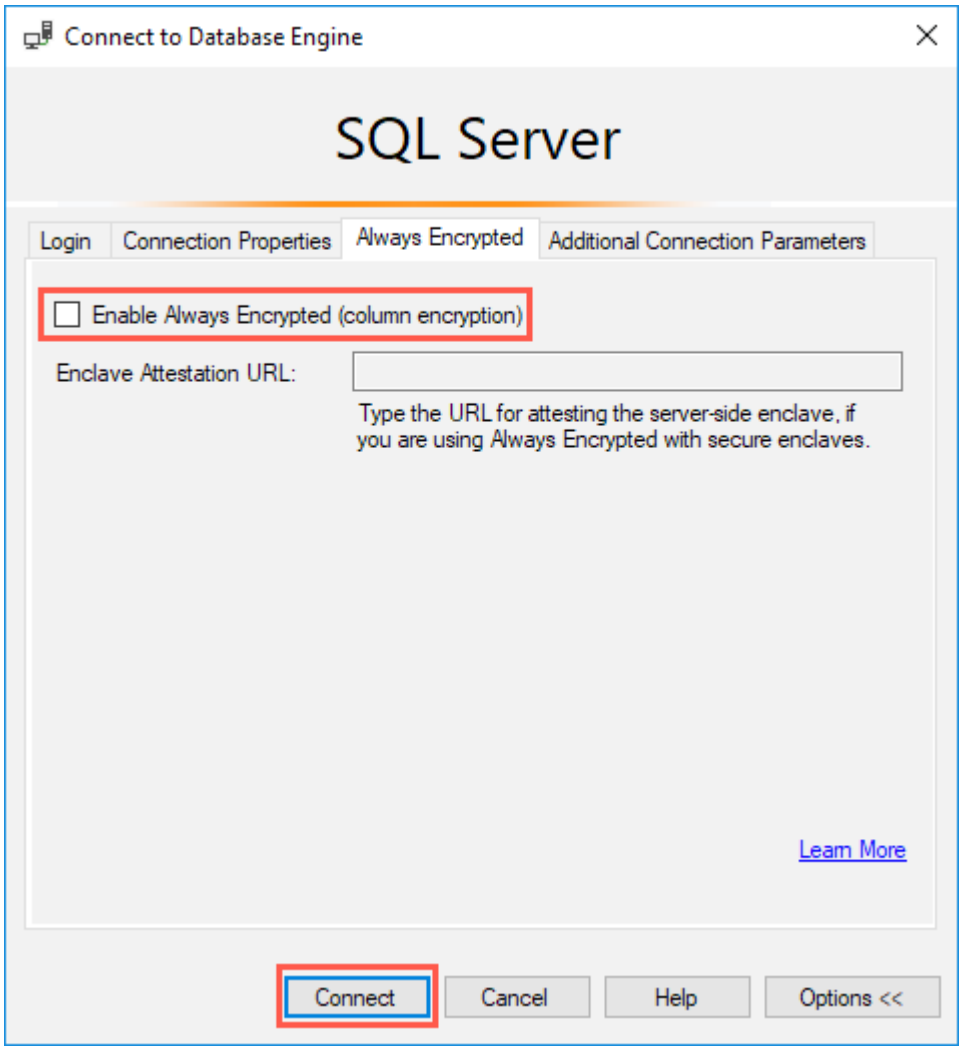
> **NOTE**: There are two additional aspects of the query above you should take note of: 1.) The `DROP INDEX` statement listed first will drop the Clustered ColumnStore index on the `customer` table, as the encryption cannot be applied while that is in place. The final statement is a `CREATE INDEX` command to recreate it. 2.) Notice the `ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE` statement to clear the query plan cache for the database in the above script. After you have altered the table, you need to clear the plans for all batches and stored procedures that access the table, to refresh parameters encryption information.

7. If you see a dialog appear after executing the above script, asking whether to enable parameterization for Always Encrypted, click **Enable**.

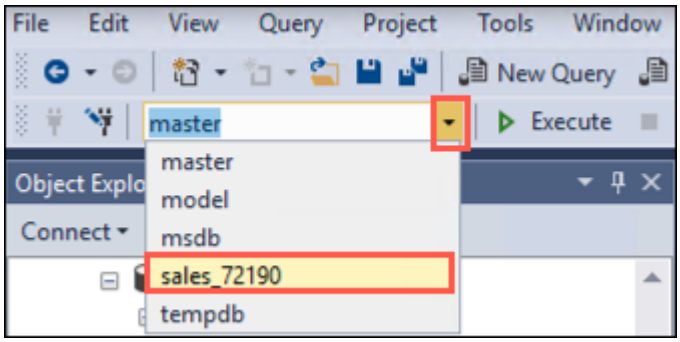


8. Open another new query window in SSMS, and right-click anywhere in the new query window, then select **Connection > Change Connection**.

9. In the Connect to Database Engine dialog, select **Options**, navigate to the **Always Encrypted** tab and make sure **Enable Always Encrypted** is not checked, and then select **Connect**.

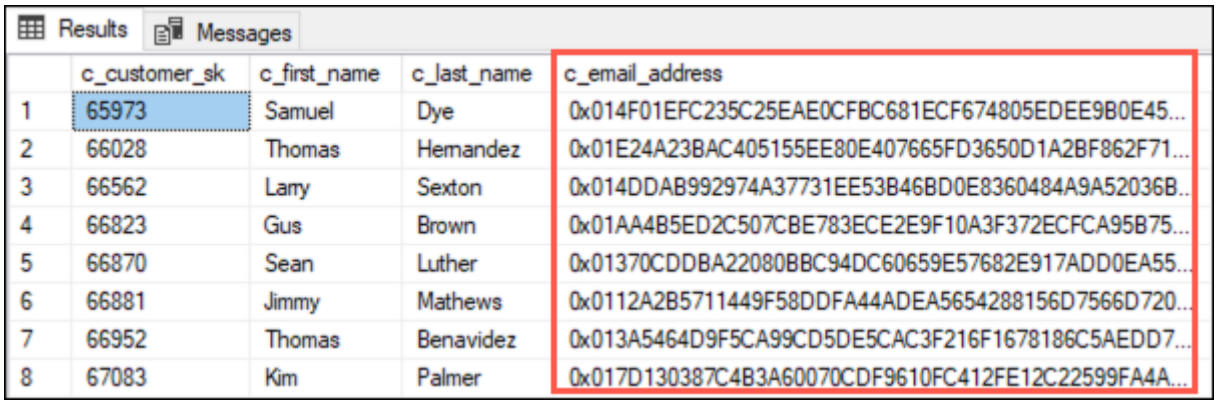


10. As you did previously, on the SSMS toolbar, change the database back to your **sales_XXXXX** database by selecting the drop down arrow next to **master** and selecting the **sales_XXXXX** database ((where XXXX is the unique identifier assigned to you for this lab).



11. To verify the `c_email_address` column is now encrypted, paste in and execute the below statement in the query window with Always Encrypted disabled. The query window should return encrypted values in the SSN and Salary columns.

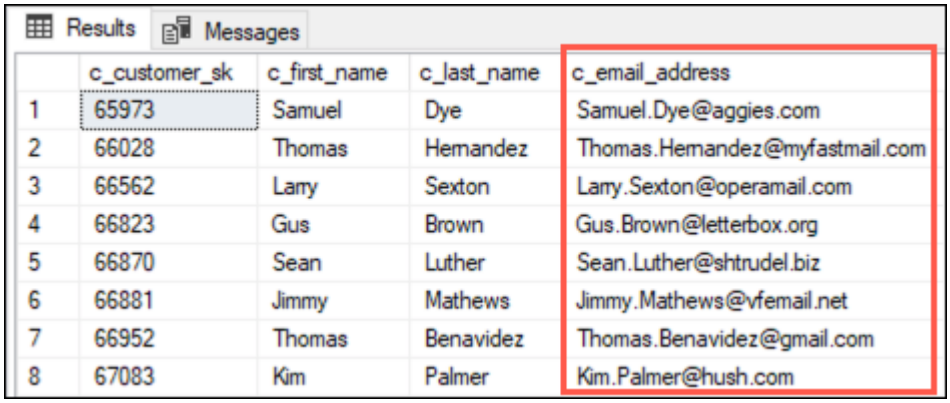
```
SELECT c_customer_sk, c_first_name, c_last_name, c_email_address FROM [dbo].
[customer]
```



Results | Messages

	c_customer_sk	c_first_name	c_last_name	c_email_address
1	65973	Samuel	Dye	0x014F01EFC235C25EAE0CFBC681ECF674805EDEE9B0E45...
2	66028	Thomas	Hernandez	0x01E24A23BAC405155EE80E407665FD3650D1A2BF862F71...
3	66562	Larry	Sexton	0x014DDAB992974A37731EE53B46BD0E8360484A9A52036B...
4	66823	Gus	Brown	0x01AA4B5ED2C507CBE783ECE2E9F10A3F372ECFCA95B75...
5	66870	Sean	Luther	0x01370CDDBA22080BBC94DC60659E57682E917ADD0EA55...
6	66881	Jimmy	Mathews	0x0112A2B5711449F58DDFA44ADEA5654288156D7566D720...
7	66952	Thomas	Benavidez	0x013A5464D9F5CA99CD5DE5CAC3F216F1678186C5AEDD7...
8	67083	Kim	Palmer	0x017D130387C4B3A60070CDF9610FC412FE12C22599FA4A...

12. Now, run the same query in the query window with the Always Encrypted enabled. The returned results should contain the data decrypted.



Results | Messages

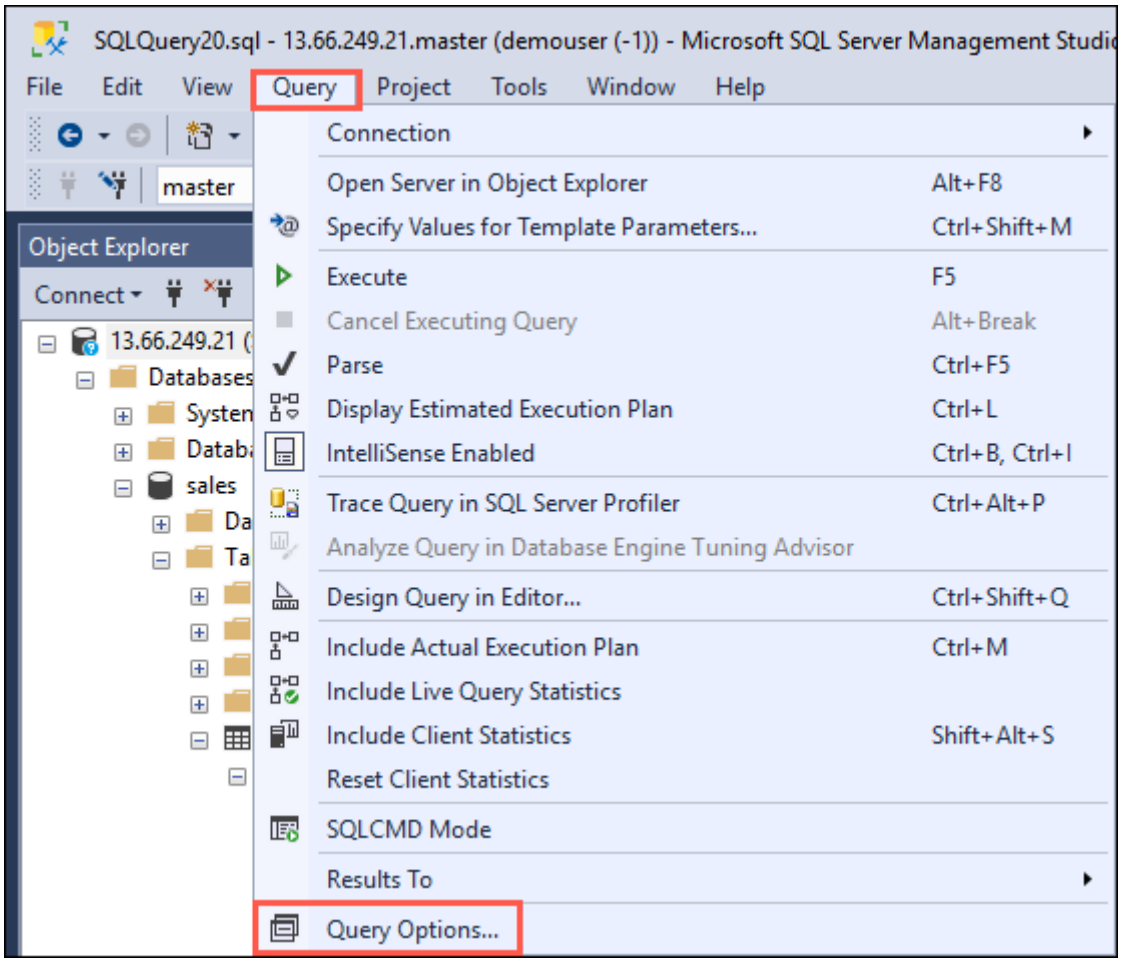
	c_customer_sk	c_first_name	c_last_name	c_email_address
1	65973	Samuel	Dye	Samuel.Dye@aggies.com
2	66028	Thomas	Hernandez	Thomas.Hernandez@myfastmail.com
3	66562	Larry	Sexton	Larry.Sexton@operamail.com
4	66823	Gus	Brown	Gus.Brown@letterbox.org
5	66870	Sean	Luther	Sean.Luther@shtrudel.biz
6	66881	Jimmy	Mathews	Jimmy.Mathews@vfemail.net
7	66952	Thomas	Benavidez	Thomas.Benavidez@gmail.com
8	67083	Kim	Palmer	Kim.Palmer@hush.com

> Using a secure enclave, you are now able to run queries against columns using Always Encrypted, and view decrypted results in SSMS.

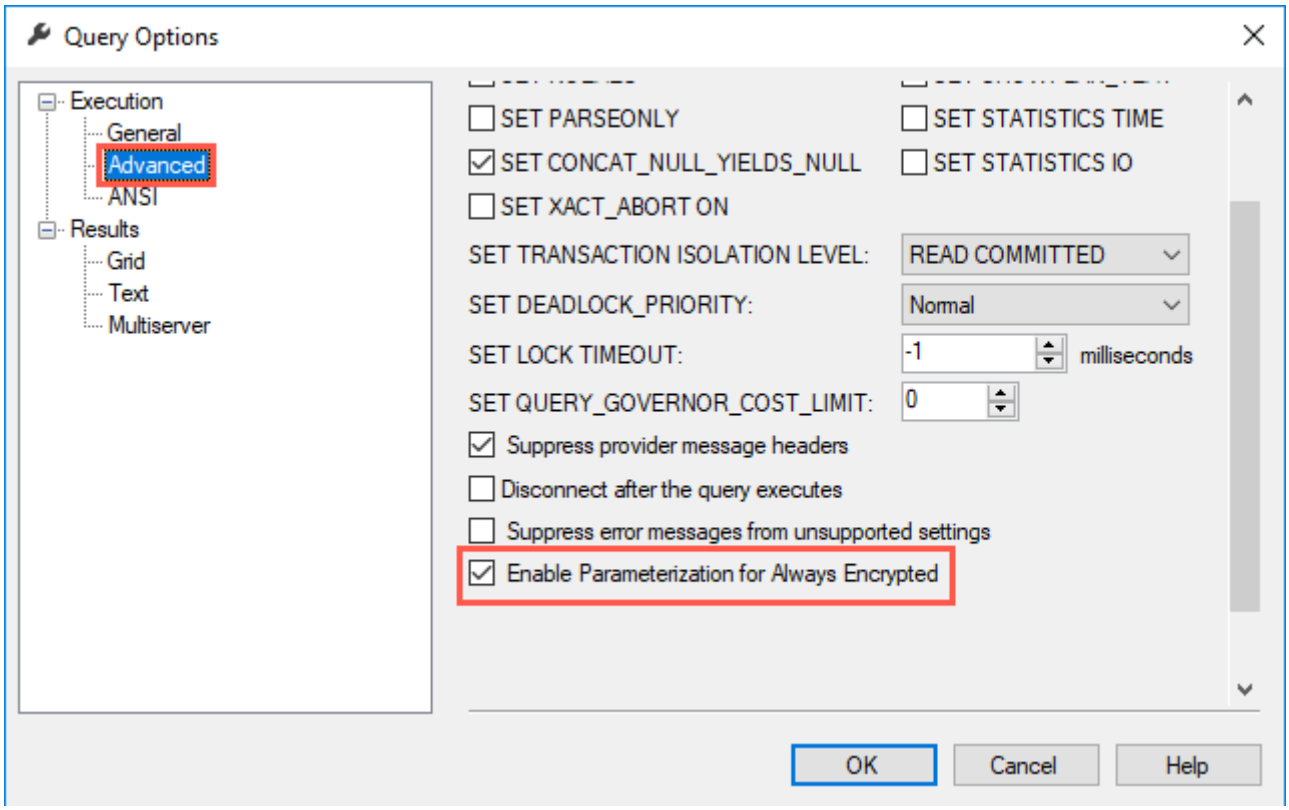
Run rich queries against an encrypted column

Now, you can run rich queries against the encrypted columns. Some query processing will be performed inside your server-side enclave.

1. Ensure that Parameterization for Always Encrypted is enabled, by selecting **Query** from the main menu of SSMS, and then selecting **Query Option...**.



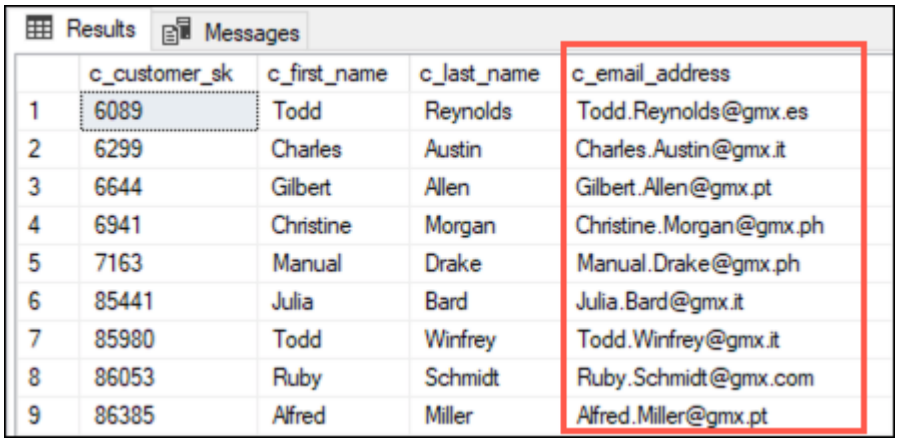
2. On the Query Options dialog, select **Advanced** under Execution, and then check the box for **Enable Parameterization for Always Encrypted**.



3. Select **OK**.

4. In the query window with Always Encrypted enabled, paste in and execute the below query. The query should return plaintext values and rows meeting the specified search criteria, which are records with email addresses with `@gmx.` in them.

```
DECLARE @EmailPattern [char](50) = '%@gmx.%';

SELECT c_customer_sk, c_first_name, c_last_name, c_email_address FROM [dbo].
[customer]
WHERE c_email_address LIKE @EmailPattern;
```



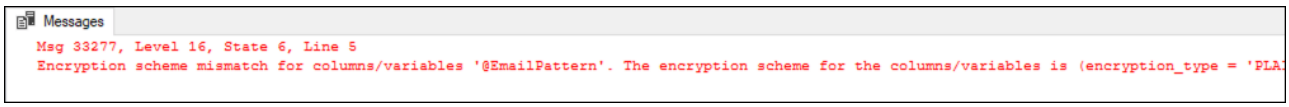
Results | Messages

	c_customer_sk	c_first_name	c_last_name	c_email_address
1	6089	Todd	Reynolds	Todd.Reynolds@gmx.es
2	6299	Charles	Austin	Charles.Austin@gmx.it
3	6644	Gilbert	Allen	Gilbert.Allen@gmx.pt
4	6941	Christine	Morgan	Christine.Morgan@gmx.ph
5	7163	Manual	Drake	Manual.Drake@gmx.ph
6	85441	Julia	Bard	Julia.Bard@gmx.it
7	85980	Todd	Winfrey	Todd.Winfrey@gmx.it
8	86053	Ruby	Schmidt	Ruby.Schmidt@gmx.com
9	86385	Alfred	Miller	Alfred.Miller@gmx.pt

> In the results, notices that all email address have a domain of `@gmx`.

5. Try the same query again in the query window that does not have Always Encrypted enabled, and note the failure that occurs. You will see an error similar to the following:

```
Msg 33277, Level 16, State 6, Line 5
Encryption scheme mismatch for columns/variables '@EmailPattern'. The
encryption scheme for the columns/variables is (encryption_type =
'PLAINTEXT') and the expression near line '4' expects it to be Randomized, ,
a BIN2 collation for string data types, and an enclave-enabled column
encryption key, or PLAINTEXT.
```



Wrap-up

Thank you for participating in the SQL Server 2019 DBA experience! We hope you are excited about the new capabilities, and will refer back to this experience to learn more about these features.

To recap, you experienced:

1. Using migration tools to evaluate and perform a migration from SQL Server 2008 R2 to a newer version of SQL Server.

2. Intelligent Query Processing (QP) performance improvements with SQL Server 2019's new database compatibility level: `150`.
3. Using the SQL Data Discovery & Classification tool to identify and tag PII and GDPR-related compliance issues.
4. Used dynamic data masking to automatically protect sensitive data from unauthorized users.
5. Enabled Row-Level Security to restrict access to data rows using user contexts.
6. Using Always Encrypted with secure enclaves to perform rich queries on encrypted data columns.

Additional resources and more information

- What's new in SQL Server 2019 preview
- Security Center for SQL Server Database Engine and Azure SQL Database
- SQL Data Discovery and Classification tool documentation
- Intelligent query processing in SQL databases
- What's new in SQL Server Machine Learning Services
- Learning content in GitHub: SQL Server Workshops
- SQL Server Samples Repository in GitHub. Feature demos, code samples etc.
- Always Encrypted with secure enclaves