

This is the last topic that will be covered in this subject. In this topic we will be going through another important database object within SQL Server – this is the use of VIEWS.

The first question that we need to answer is: What is a view?

- A view can be considered as virtual/logical table whose content is obtained through the use of a query (which may obtain data from one single column or from multiple columns) or sometimes even another view
- A view is said to be virtual/logical as it contains no data but is like a window through which data can be viewed or changed
- Each view will have one or more base tables. Base tables are tables on which the view is actually based

The second question that needs to be answered is the following: What are the advantages of using a view?

- It simplifies the internal workings of how the database works to users – the code which is required is much simpler
- It offer better security mechanisms as the users will be given access to particular data according to their needs. Also users will not have direct access to the underlying tables.
- If also offers data independence as one view can be used to retrieve data from several tables within the database

Views are commonly categorised into two: SIMPLE and COMPLEX views.

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through views (INSERT, UPDATE, DELETE)	Yes	Not always

While creating a view the below syntax needs to be followed:

```
CREATE VIEW [ schema_name . ] viewname [ (column [ ,...n ] ) ]  
[WITH [ENCRYPTION | SCHEMABINDING | VIEW_METADATA] [,...n ] ]  
AS select_statement  
[WITH CHECK OPTION]
```

where:

- **schema_name**: this is the name of the schema to which this view will pertain,
- **viewname**: the name of the view
- **column**: the column name within the view. This is an optional part unless the column is being obtained from an arithmetic operation/function/constant. If the column name is not specified then the name of the column in the SELECT statement is taken.
- **ENCRYPTION**: encrypts the text of the CREATE VIEW statement and prevents the view from being published through SQL Server replication
- **SCHEMABINDING**: binds the view to the underlying table/s. This means that the base table cannot be modified such that it affects the view definition – the view definition is to be modified/dropped first and then the underlying table. When this option is used the select_statement must include two-part names (schema.object) of tables/views/functions
- **VIEW_METADATA**: specifies that the SQL Server instance returns the meta data information about the view instead of the base table/s.
- **select_statement**: this needs to be replaced by a complete SELECT statement that defines the view. The SELECT clause cannot include an ORDER BY, INTO, or OPTION clause
- **CHECK OPTION**: forces all the modification statements passed through the view to follow the criteria within the select_statement. When data is modified through a view that has the CHECK OPTION, it makes sure that the data changed remains visible through the view.

VIEWS

Example 1: Write a query that will create a view named empvu80, that will allow users to view the employee_id, surname and salary of all the employees in department 80.

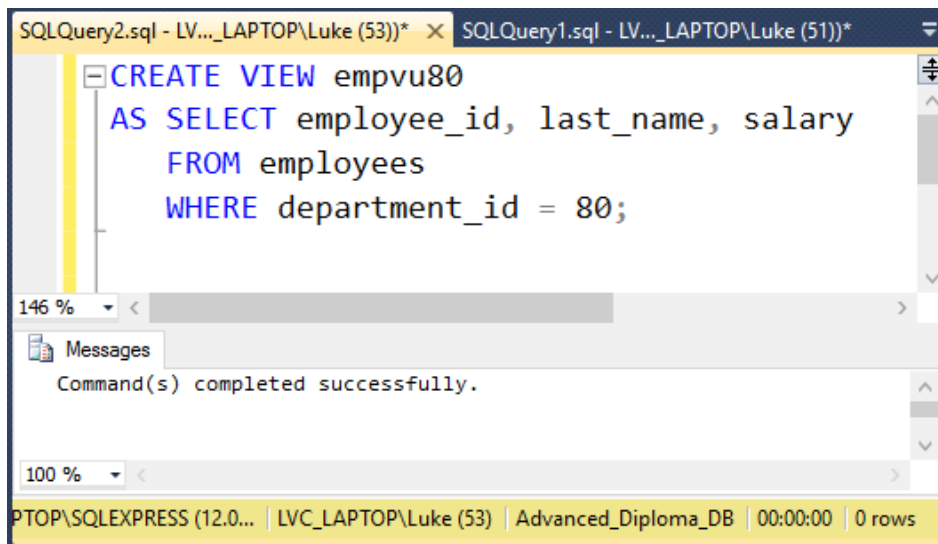
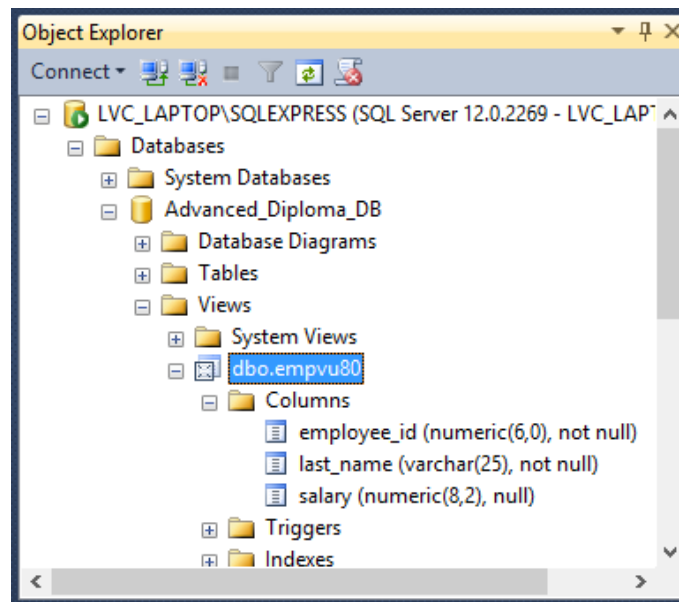


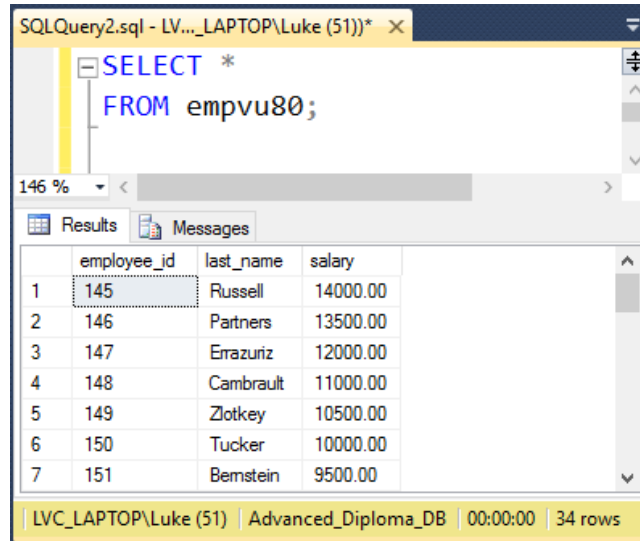
Figure 1 - Result of example 1: view of all employees in department 80

NOTE: The above query has successfully create the empvu80 and this can be verified by expanding the Views folder (in Object Explorer) in the database as shown below:



VIEWS

Once that a view has been created it should be used. In order to be able to use a view a query needs to be executed but the FROM clause should refer to the view instead of the table. In the below screenshot we will retrieve all the data that is found in the newly created view.



The screenshot shows a SQL query window with the following text:

```
SELECT *
FROM empvu80;
```

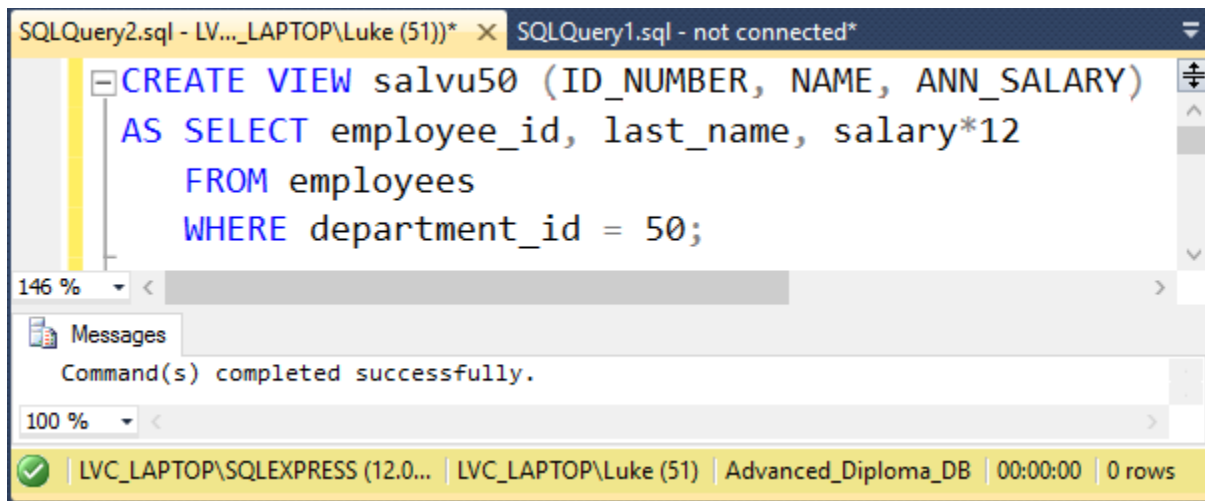
Below the query, the 'Results' tab is active, displaying a table with 7 rows and 4 columns: employee_id, last_name, salary, and an unnamed column. The data is as follows:

	employee_id	last_name	salary	
1	145	Russell	14000.00	
2	146	Partners	13500.00	
3	147	Errazuriz	12000.00	
4	148	Cambrault	11000.00	
5	149	Zlotkey	10500.00	
6	150	Tucker	10000.00	
7	151	Bernstein	9500.00	

The status bar at the bottom indicates: LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 34 rows

Figure 2 - Query to retrieve information from view named empvu80

Example 2: Write a query that will create a view named salvu50 which will have 3 columns named (ID_NUMBER, NAME, ANN_SALARY). The view should display the employee_id, surname and annual salary (which is the salary*12) for all those employees working in department 50



The screenshot shows a SQL query window with the following text:

```
CREATE VIEW salvu50 (ID_NUMBER, NAME, ANN_SALARY)
AS SELECT employee_id, last_name, salary*12
FROM employees
WHERE department_id = 50;
```

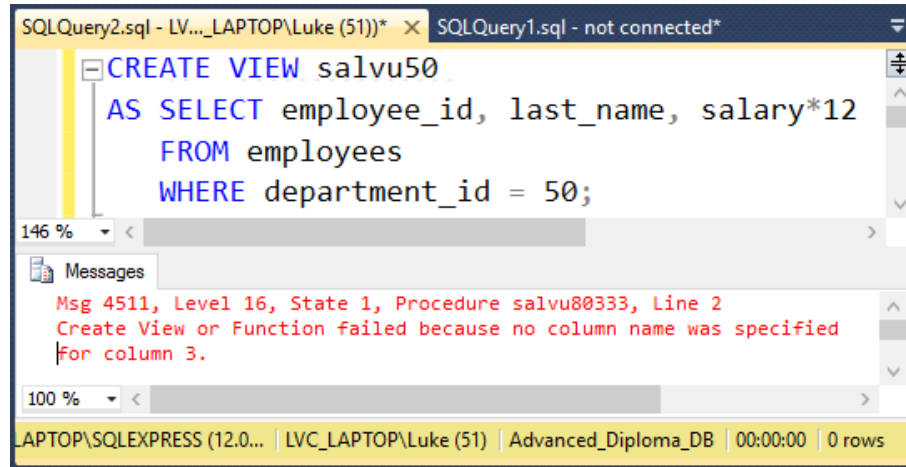
Below the query, the 'Messages' tab is active, displaying the message: "Command(s) completed successfully."

The status bar at the bottom indicates: LVC_LAPTOP\SQLEXPRESS (12.0...) | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows

Figure 3 - Result of example 2: salvu50 created successfully

VIEWS

NOTE: The syntax used in example 2 works correctly. It is important to notice that since the last column in the SELECT clause is an expression, it is really important that the column names after the view name are included as otherwise the query returns an error as shown below:



Example 3: Write a query that will retrieve all the columns and rows that are accessible via the salvu50 view.

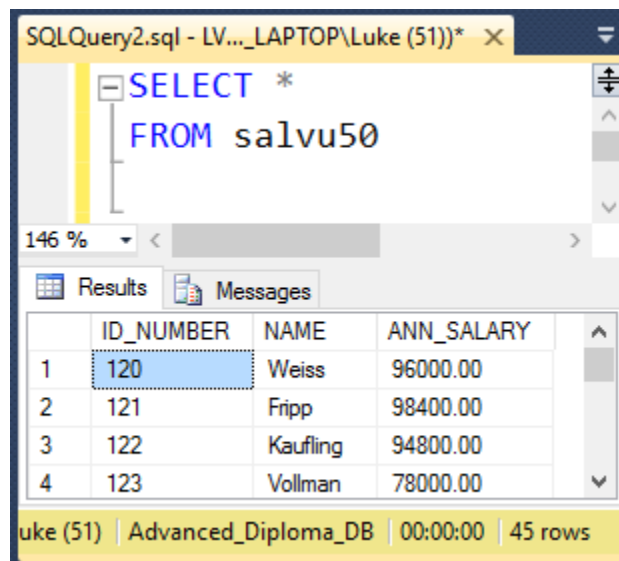


Figure 4 - Result of example 3 - Retrieving data from salvu50

NOTE: The result returned in example 3 shows 3 columns all of which have a column name which is the same as that which was specified in example 2, while creating the view.

VIEWS

Example 4: Write a query that create a view with the same characteristics as the in example 2, but this time you are to make sure that the view uses Encryption.

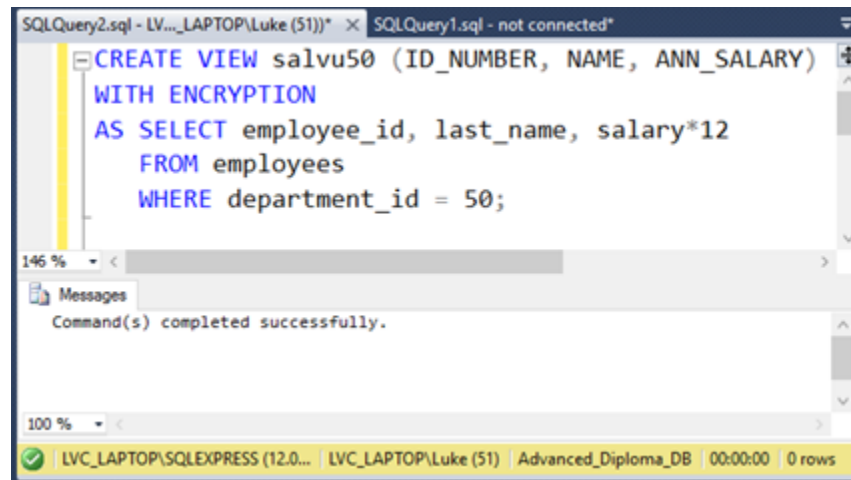
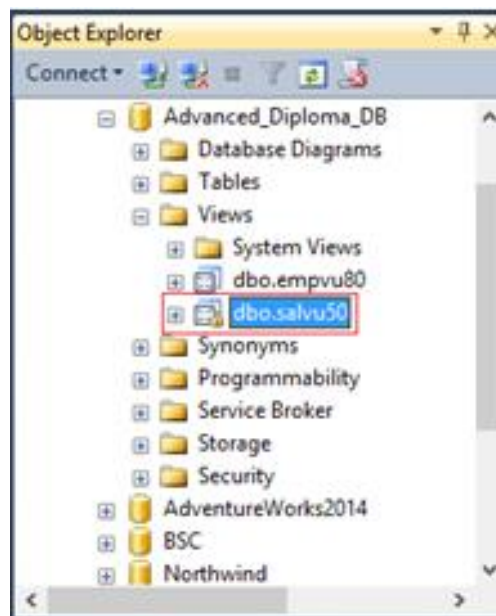
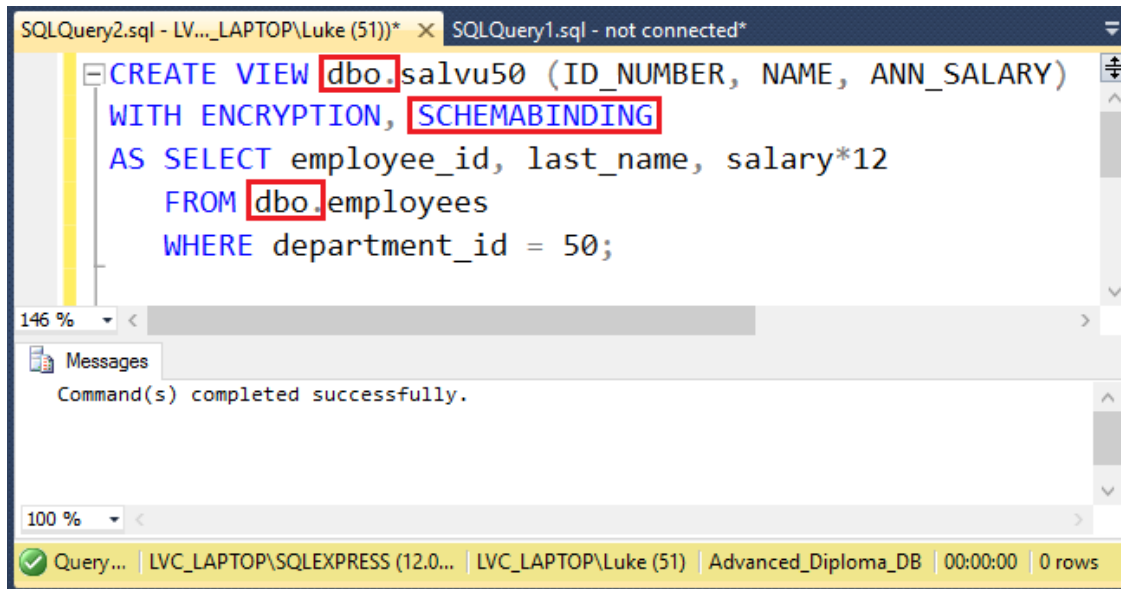


Figure 5 -Result of example 4 - view with encryption



NOTE: As can be seen in the second screen shot, in the Object Explorer the newly created view salvu50 has a padlock on the icon. This signifies that the view's statement has been encrypted

Example 5: Modify the query that was used to create the view in example 4, such that the view has SCHEMABINDING as well as ENCRYPTION



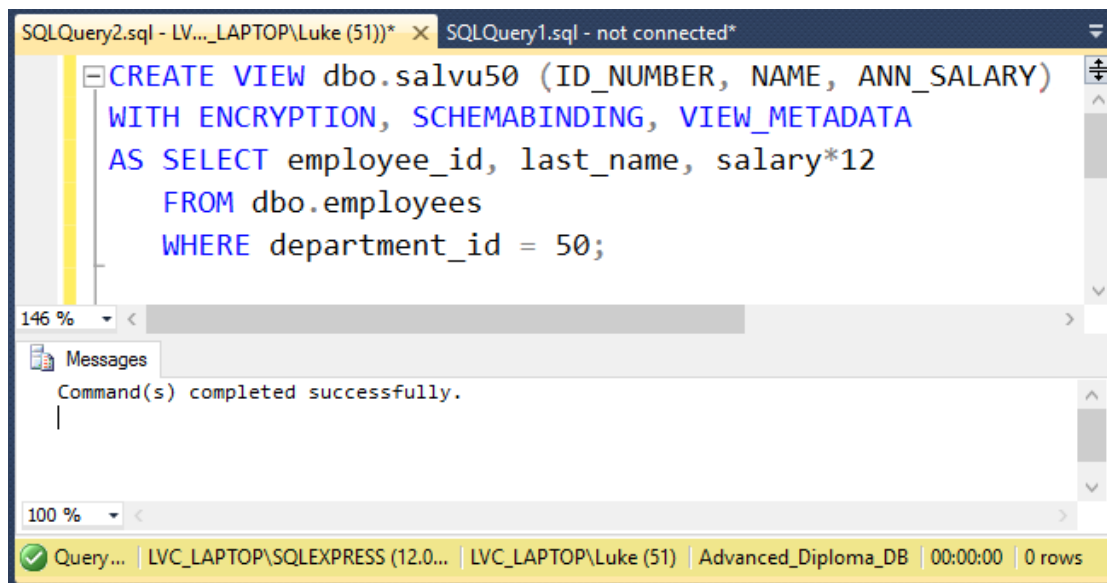
The screenshot shows a SQL query window with the following code:

```
CREATE VIEW dbo.salvu50 (ID_NUMBER, NAME, ANN_SALARY)
WITH ENCRYPTION, SCHEMABINDING
AS SELECT employee_id, last_name, salary*12
FROM dbo.employees
WHERE department_id = 50;
```

The code is displayed in a window titled "SQLQuery2.sql - LV..._LAPTOP\Luke (51))*" and "SQLQuery1.sql - not connected*". The status bar at the bottom indicates "Query..." and "LVC_LAPTOP\SQLEXPRESS (12.0...) | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows". A message pane below the query shows "Command(s) completed successfully."

Figure 6 - Result of example 5 - View with SCHEMABINDING

NOTE: In the above code it is important to notice that when you introduce the SCHEMABINDING option, the table name and view name should be preceded by the schema in which they reside. This is highlighted by red squares in Figure 6. In the below screenshot we have modified the same query such that now the view has the VIEW_METADATA option as well

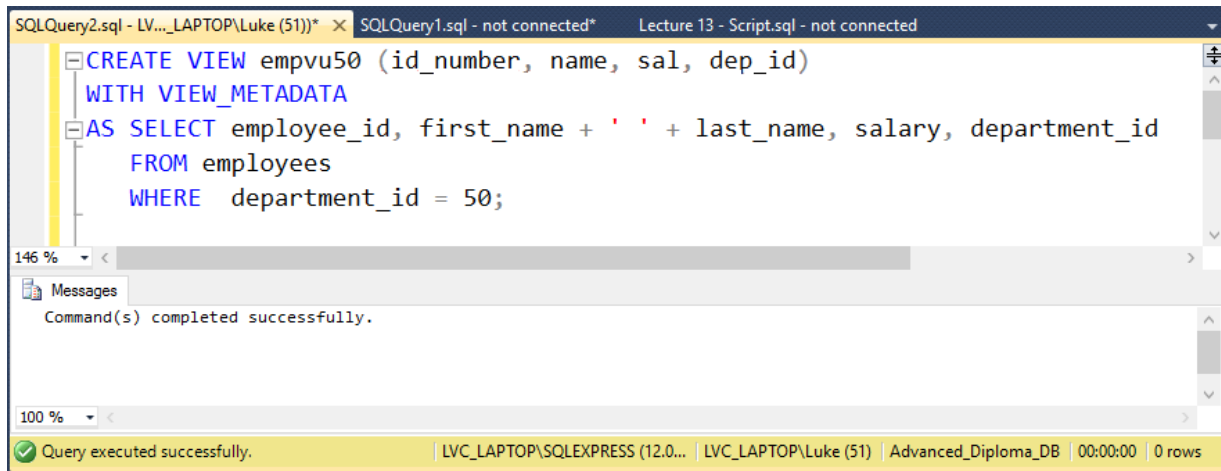


The screenshot shows a SQL query window with the following code:

```
CREATE VIEW dbo.salvu50 (ID_NUMBER, NAME, ANN_SALARY)
WITH ENCRYPTION, SCHEMABINDING, VIEW_METADATA
AS SELECT employee_id, last_name, salary*12
FROM dbo.employees
WHERE department_id = 50;
```

The code is displayed in a window titled "SQLQuery2.sql - LV..._LAPTOP\Luke (51))*" and "SQLQuery1.sql - not connected*". The status bar at the bottom indicates "Query..." and "LVC_LAPTOP\SQLEXPRESS (12.0...) | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows". A message pane below the query shows "Command(s) completed successfully."

Example 6: Write a query named empvu50 which will have four columns (id_number, name, sal and dep_id). It is important that this view includes only those employees who work in department 50 and the second column should have the name and surname separated by a space. Also the view should cater for viewing of meta data.



```

CREATE VIEW empvu50 (id_number, name, sal, dep_id)
WITH VIEW_METADATA
AS SELECT employee_id, first_name + ' ' + last_name, salary, department_id
FROM employees
WHERE department_id = 50;

```

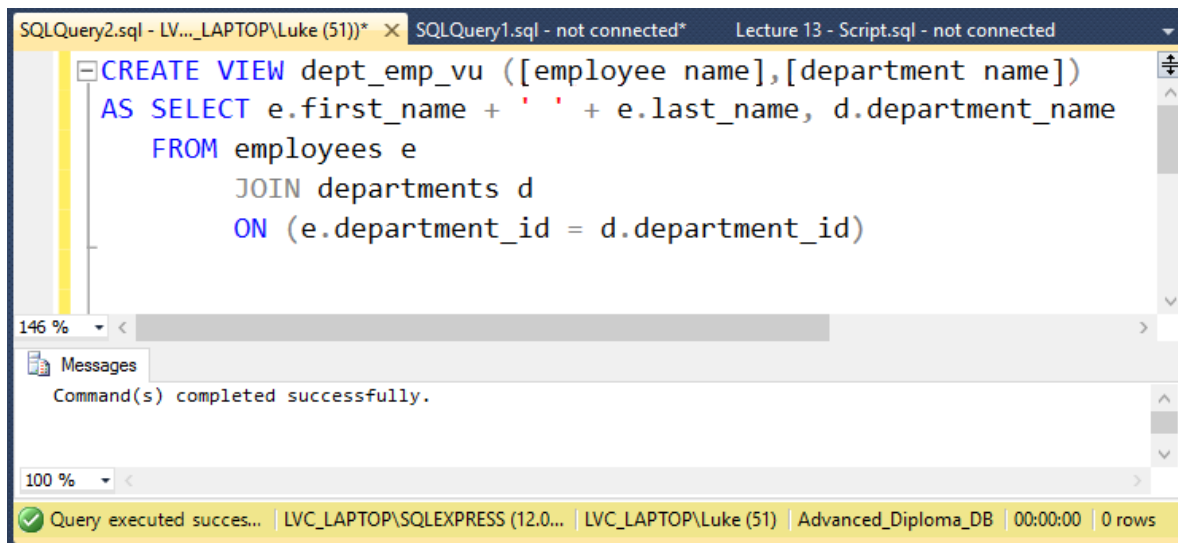
Messages
Command(s) completed successfully.

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0...) | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows

Figure 7 - Result of example 6: empvu50 with concatenation in the SELECT statement

NOTE: All example that have been created so far are said to be SIMPLE Views as the SELECT statement included in the AS clause obtains data from one single table and does not involve any aggregate functions.

Example7: Write a query named dept_emp_vu which has 2 columns named employee name, and department name. This view should display the employee name and surname in one column and the department in which they work in another column.



```

CREATE VIEW dept_emp_vu ([employee name],[department name])
AS SELECT e.first_name + ' ' + e.last_name, d.department_name
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)

```

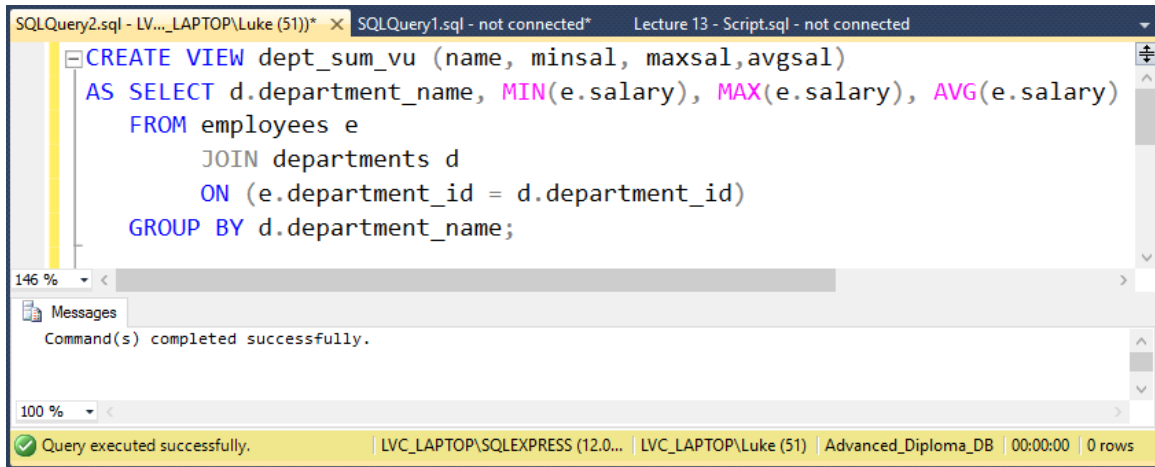
Messages
Command(s) completed successfully.

Query executed succes... | LVC_LAPTOP\SQLEXPRESS (12.0...) | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows

Figure 8 – Result of example 7: Dept_emp_vu which is a complex view

NOTE: the view in example 6 is a COMPLEX VIEW as it is obtaining data from two different tables (employees and departments).

Example 8: Write a query named dept_sum_vu which has 4 columns named name, minsal, maxsal, and avgsal. This view should display the minimum salary, maximum salary and average salary of all the different departments. It is very important that the department name is displayed.



```

CREATE VIEW dept_sum_vu (name, minsal, maxsal, avgsal)
AS SELECT d.department_name, MIN(e.salary), MAX(e.salary), AVG(e.salary)
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)
GROUP BY d.department_name;

```

Messages
Command(s) completed successfully.

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows

Figure 9 - Result of example 8 – complex view with data from multiple tables

NOTE: similar to example 7, the view in example 8 is a COMPLEX VIEW as it is obtaining data from two different tables. Also in this query is the use of aggregate functions such as MIN, MAX and AVG.

The WITH CHECK OPTION

The main aim of this option is to make sure that DML operations performed on the view stay in the domain of the view. This means that this option specifies that INSERT and UPDATE statements performed through the view cannot create/modify rows that the view will not be able to select. This enables integrity constraints and data validation checks.

VIEWS

Example 9: Write a query that creates a view that displays all the employees in department 20. The name of the view should be empvu20 and all the columns should be included

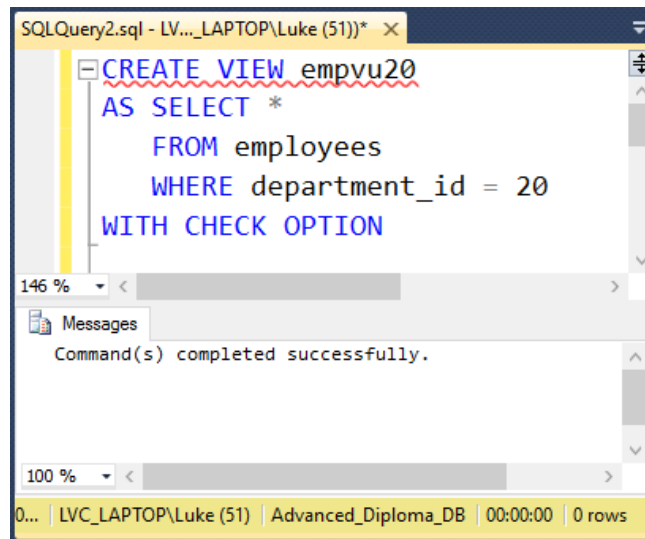
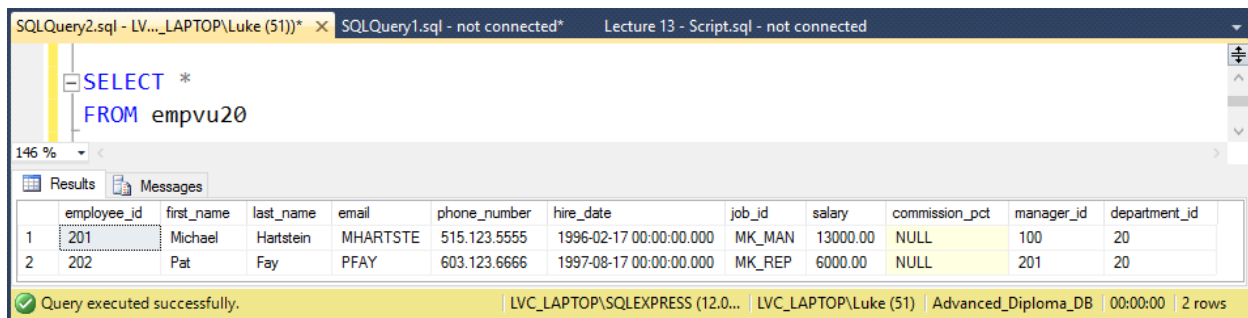


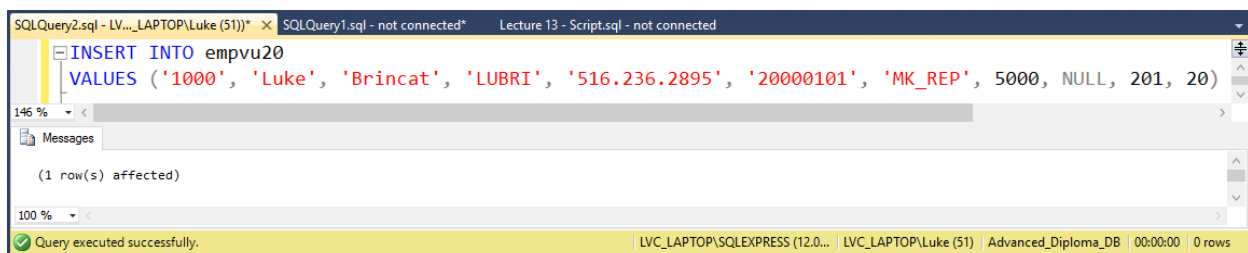
Figure 10 - Result of example 9 - empvu20 that has the WITH CHECK OPTION constraint

NOTE: The user will be able to view the data obtained by this view, and create INSERT and UPDATE statements that affect rows in department 20. It is important that these statements leave the data in the domain of the view.

The below screenshot shows all the data that can be retrieved by empvu20:

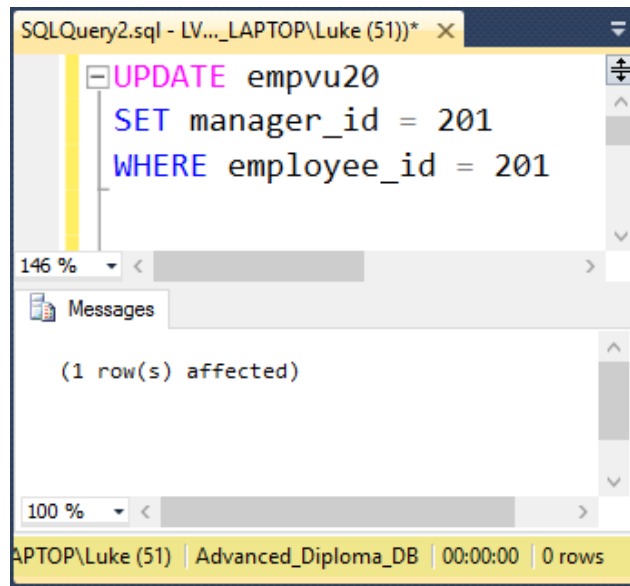


The below query adds employee 1000 to the employees table via the empvu20 view. Important to note that the employee is placed in department 20 (the last value in the VALUES part)



VIEWS

The below query will execute without problems as the statement does not change the domain of employee 201. It is only modifying the manager id of employee 201 from 100 to 201

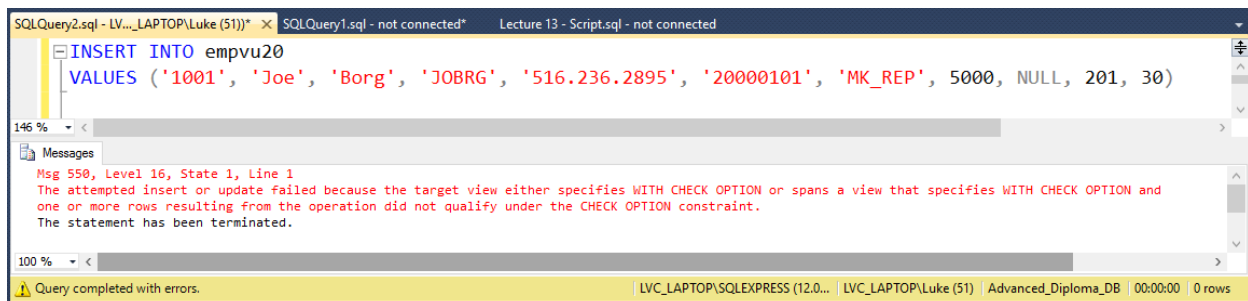


The screenshot shows a SQL query window titled 'SQLQuery2.sql - LV..._LAPTOP\Luke (51))' with the following SQL code:

```
UPDATE empvu20
SET manager_id = 201
WHERE employee_id = 201
```

Below the query, the 'Messages' pane displays '(1 row(s) affected)'. The status bar at the bottom indicates 'LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows'.

The below statement does not execute and returns an error. The reason behind this error is that the empvu20 view has a WITH CHECK OPTION constraint. The INSERT statement below attempts to add an employee in department 30 (last value) through the use of empvu20. This is not possible as the scope of empvu20 is department 20 and therefore if this INSERT is allowed it will not be accessible through the same view.



The screenshot shows a SQL query window titled 'SQLQuery2.sql - LV..._LAPTOP\Luke (51))' with the following SQL code:

```
INSERT INTO empvu20
VALUES ('1001', 'Joe', 'Borg', 'JOBRG', '516.236.2895', '20000101', 'MK_REP', 5000, NULL, 201, 30)
```

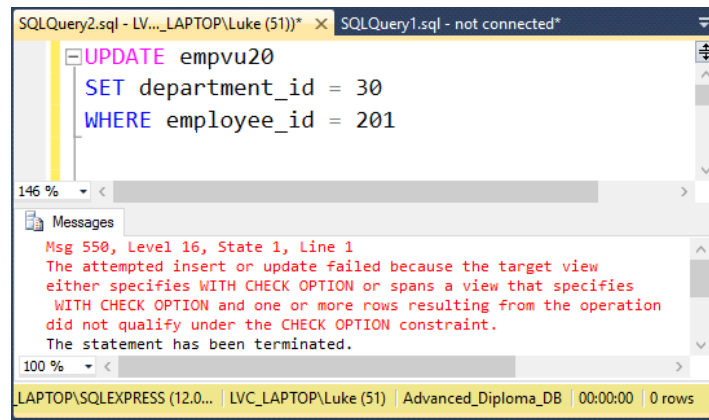
Below the query, the 'Messages' pane displays an error message:

```
Msg 550, Level 16, State 1, Line 1
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
The statement has been terminated.
```

The status bar at the bottom indicates 'Query completed with errors.' and 'LVC_LAPTOP\SQLEXPRESS (12.0...) | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows'.

VIEWS

The below statement attempts to change the department_id of employee 201 from 20 to 30 through empvu20. This is not allowed as the scope of view empvu20 is department 20.



```
SQLQuery2.sql - LV..._LAPTOP\Luke (51))* x SQLQuery1.sql - not connected*
UPDATE empvu20
SET department_id = 30
WHERE employee_id = 201
```

146 %

Messages

Msg 550, Level 16, State 1, Line 1
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
The statement has been terminated.

100 %

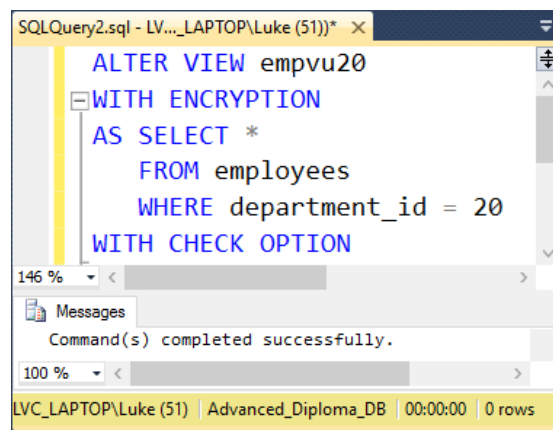
LAPTOP\SQLLEXPRESS (12.0... | LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows

Modifying an existing view.

In order to modify an existing view the ALTER VIEW statement must be used. The syntax and options available in ALTER VIEW are identical to those of CREATE VIEW, in fact the syntax is the same:

```
ALTER VIEW [ schema_name . ] viewname [ (column [ ,...n ] ) ]  
[WITH [ENCRYPTION | SCHEMABINDING | VIEW_METADATA] [,...n ] ]  
AS select_statement  
[WITH CHECK OPTION]
```

Example 10: Modify the view in example 9 such that it becomes encrypted



```
SQLQuery2.sql - LV..._LAPTOP\Luke (51))* x
ALTER VIEW empvu20
WITH ENCRYPTION
AS SELECT *
FROM employees
WHERE department_id = 20
WITH CHECK OPTION
```

146 %

Messages

Command(s) completed successfully.

100 %

LVC_LAPTOP\Luke (51) | Advanced_Diploma_DB | 00:00:00 | 0 rows

Figure 11 - Result of example 10 - Changing empvu20 so that it has encryption

Removing an existing view.

To remove a view from the database the DROP VIEW statement should be used. The DROP VIEW keywords should be followed by the view name.

Example 11: Remove the view named empvu20

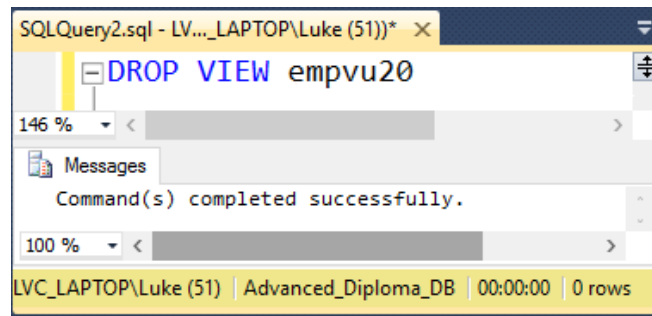


Figure 12 - Result of example 11 - removing the empvu20 view