Document: Mandatory Assignment 1, Blog DB.

Student: Brian Munksgaard

Data modelling notation: Crow's Foot

Video Subject: Stored Procedures and View

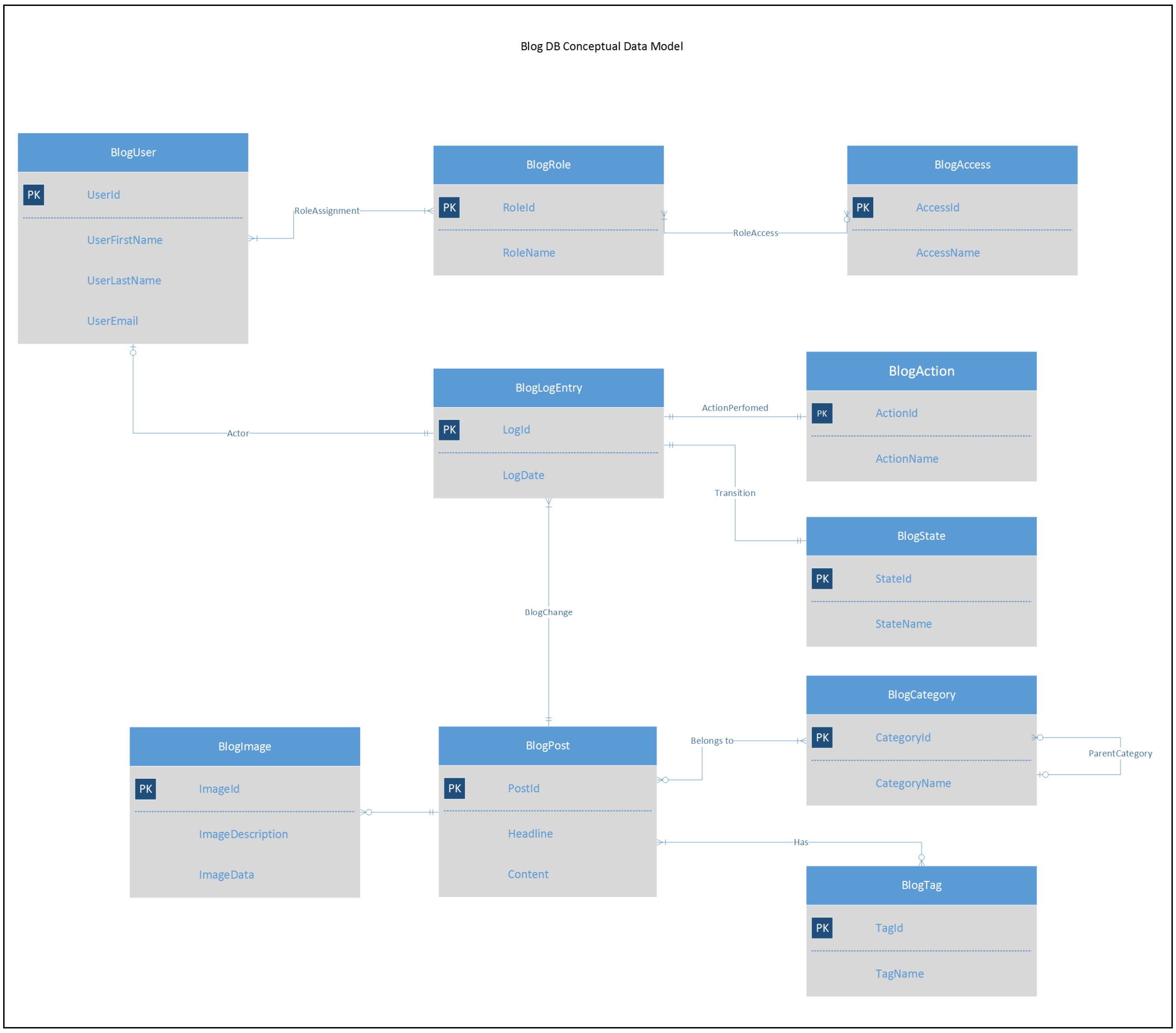
Video Location: https://youtu.be/tuamEblknew

1. Start of the blog

1.1 Data Model

Conceptual Model

Below you can see the conceptual/logical data model for the Blog DB. Since this database is rather simple there are no distinctions between the conceptual and the logical data model.



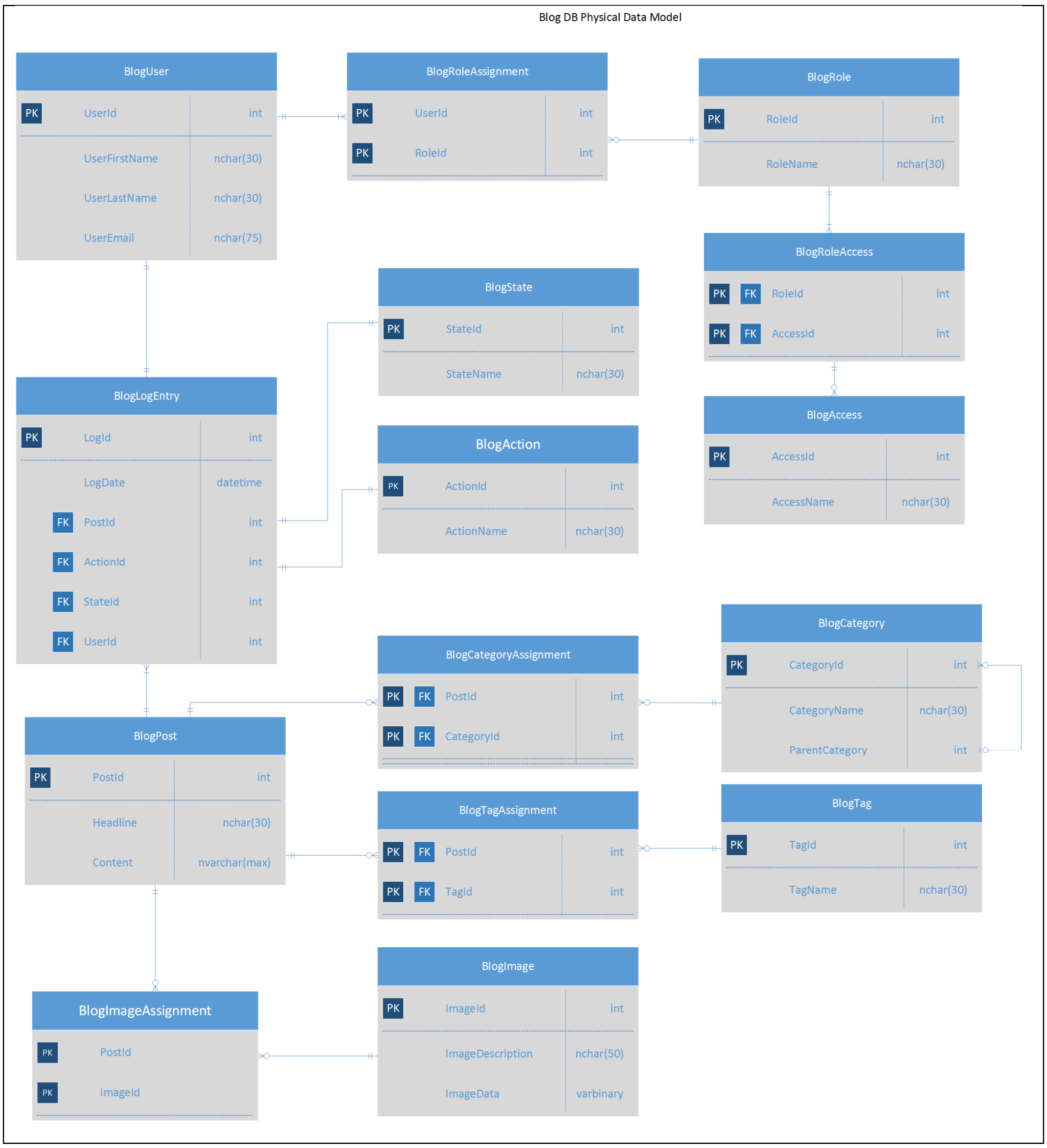
Given this data model we can:

- Register users and assign one or more roles to the user.
- Assign access levels to roles.
- Create blog posts that belongs to multiple categories, have multiple tags and contains multiple images. A blog post must belong to at least one category.
- Support a hierarchy of categories, a category may have sub-categories.
- Perform actions (like create, edit, publish, delete etc.) on the blog post.
- Log blog actions by user and date.
- Track (using the log) the blog state transitions caused by actions.

One might argue that the BlogLogEntry table should not be part of the conceptual model. However, since transitions between states are considered important information, it is modelled here. Assigning or removing tags or categories need not be logged. The model can be extended to support this if needed.

Physical Model

Knowing that the Blog DB should be implemented in Microsoft SQL Server, I have created a physical data model as well. The physical data model displays further details such as attributes, corresponding data types and junction tables used to implement many-to-many relations.



The junction tables are:

- BlogRoleAssignment (users assigned to roles).
- BlogRoleAccess (access level(s) for roles).
- BlogCategoryAssignment (categories assigned to a blog post).
- BlogTagAssignment (tags assigned to a blog post).
- BlogImageAssigment (images used in a blog post).

1.2 SQL

Q1 - Add Blog Post

Assuming we have the following tags and categories in our Blog DB:

TagId	TagName
1	Mast
2	Rig
3	Fortøjring
4	Påhængsmotor

CategoryId	CategoryName	ParentId
1	Bådtyper	NULL
2	Kølbåd	1
3	Jolle	1
4	Motorbåd	1
5	Kajak	1
6	Beklædning	NULL
7	Sejlområder	NULL
8	Klubber	7

And the users:

UserId	UserFirstName	UserLastName	UserEmail
1	Brian	Munksgaard	brian.munksgaard@gmail.com
2	Emil	Munksgaard	emil.munksgaard@gmail.com
3	Lucas	Munksgaard	lucas.munksgaard@gmail.com
4	Mikkel	Munksgaard	mikkelm.munksgaard@gmail.com

And the possible blog states:

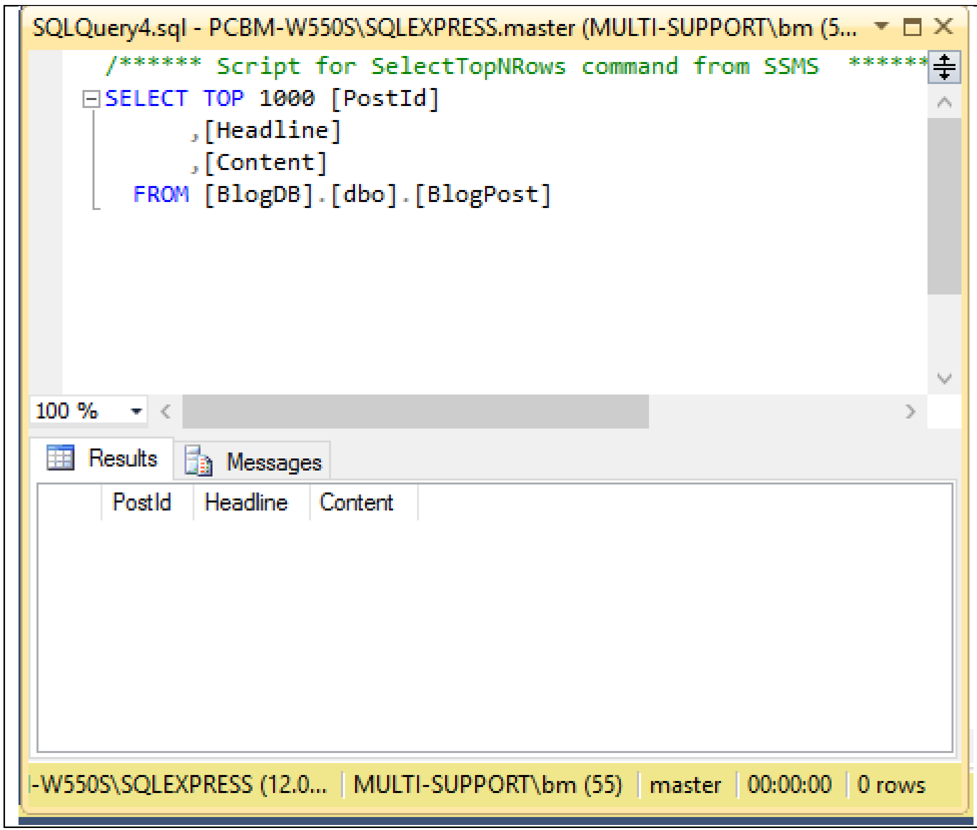
StateId	StateName
1	Draft
2	Published
3	Archived
4	Hidden

We can then establish a blog post with the following SQL statements:

```sql
USE BlogDB;

-- Insert blog post.
INSERT INTO dbo.BlogPost (Headline, Content)
VALUES ('Optimist', 'Optimistjolle, også kaldet optimisten eller optien, er bla bla
bla .....');

-- Assign category.
INSERT INTO dbo.BlogCategoryAssignment (PostId, CategoryId)
VALUES (1, 3); -- Første blog post er om joller.
```

```
-- Assign tags.
INSERT INTO dbo.BlogTagAssignment (PostId, TagId)
VALUES (1, 1), -- Mast
       (1, 2)  -- Rig

-- Add log entry.
INSERT INTO dbo.BlogLogEntry (LogDate, PostId, ActionId, StateId, UserId)
VALUES (
   GETDATE(),
   1, /* PostId */
   1, /* Action: Create */
   1, /* State: Draft */
   1  /* User: Brian Munksgaard */
);
```

Before executing the statements the blog post table is empty:



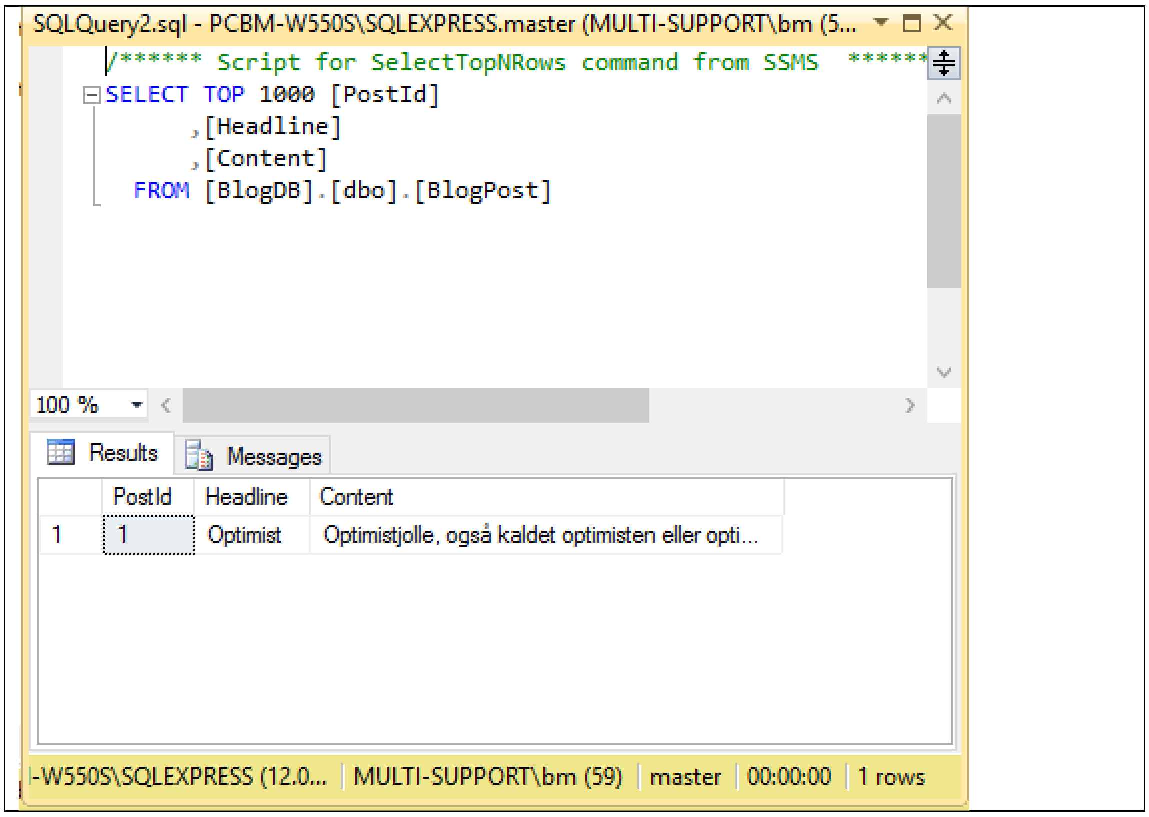
SQLQuery4.sql - PCBM-W550S\SQLEXPRESS.master (MULTI-SUPPORT\bm (5...
100 %
Results
Messages

```
/****** Script for SelectTopNRows command from SSMS ******
SELECT TOP 1000 [PostId]
      ,[Headline]
      ,[Content]
  FROM [BlogDB].[dbo].[BlogPost]
```

PostId	Headline	Content

|-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (55) | master | 00:00:00 | 0 rows

After executing the statements the blog post table looks like this:



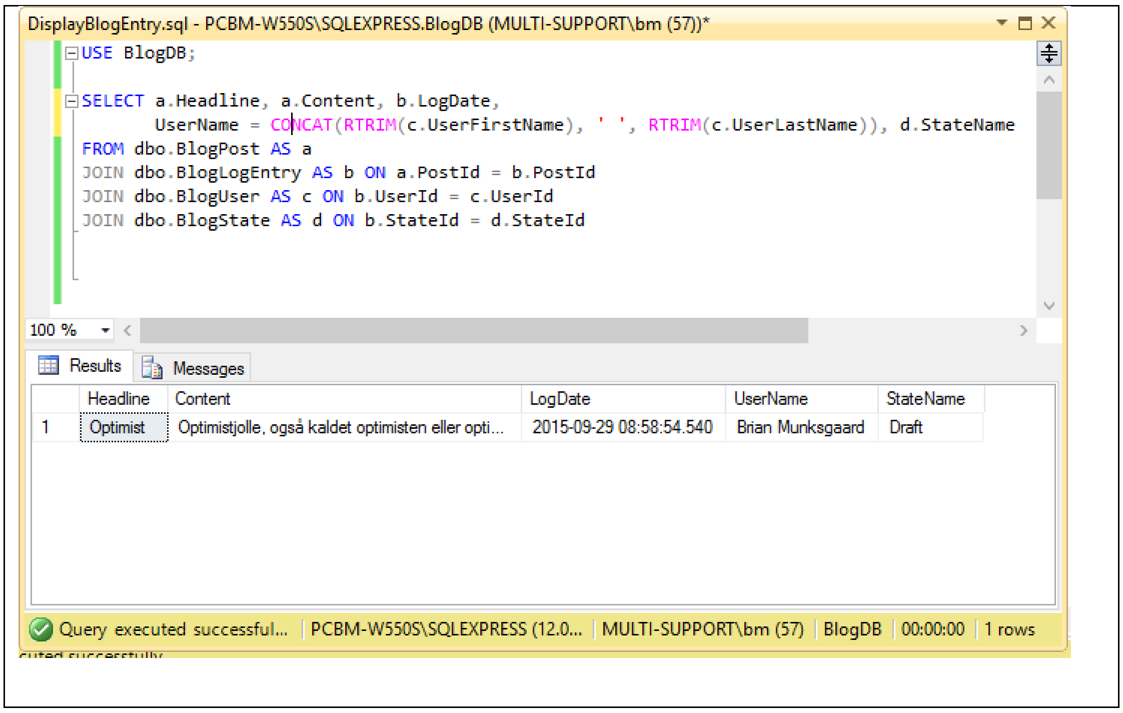
SQLQuery2.sql - PCBM-W550S\SQLEXPRESS.master (MULTI-SUPPORT\bm (5...

```
/****** Script for SelectTopNRows command from SSMS  ******
SELECT TOP 1000 [PostId]
      ,[Headline]
      ,[Content]
  FROM [BlogDB].[dbo].[BlogPost]
```

	PostId	Headline	Content
1	1	Optimist	Optimistjolle, også kaldet optimisten eller opti...

```
|-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (59) | master | 00:00:00 | 1 rows
```

Joining the relevant tables you can see the user Brian created a blog entry with the headline Optimist and that the blog entry is a draft:



DisplayBlogEntry.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (57))*

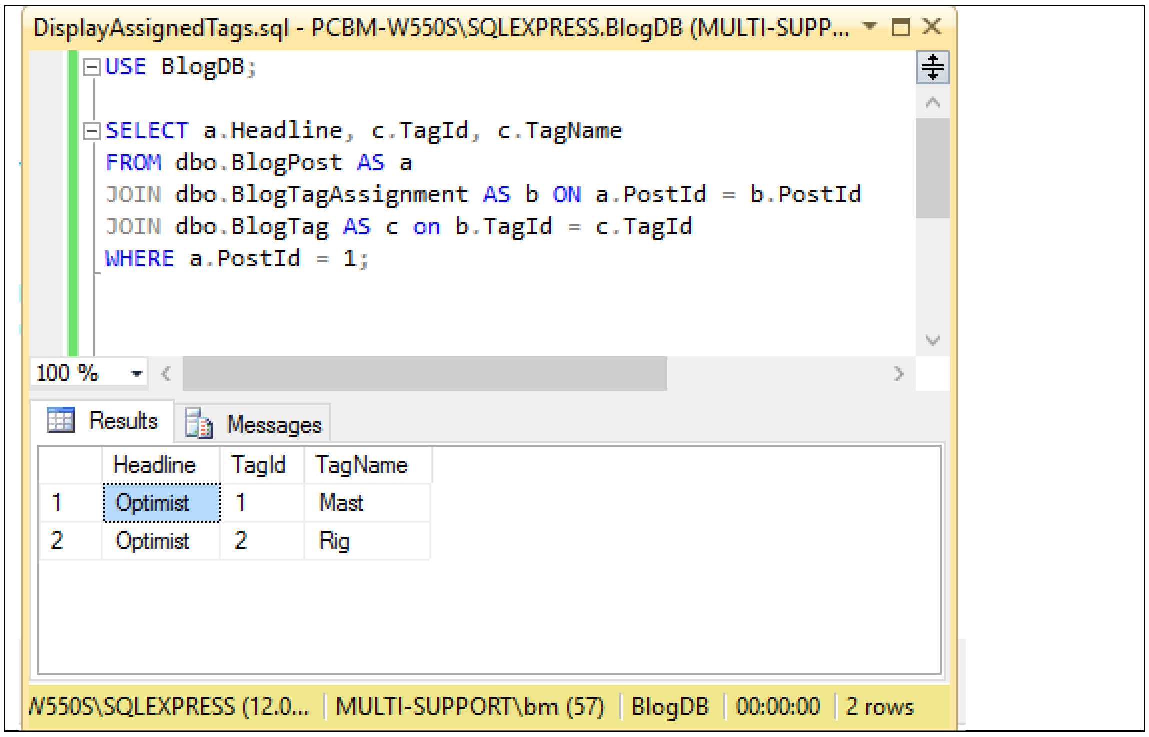
```
USE BlogDB;

SELECT a.Headline, a.Content, b.LogDate,
       UserName = CONCAT(RTRIM(c.UserFirstName), ' ', RTRIM(c.UserLastName)), d.StateName
FROM dbo.BlogPost AS a
JOIN dbo.BlogLogEntry AS b ON a.PostId = b.PostId
JOIN dbo.BlogUser AS c ON b.UserId = c.UserId
JOIN dbo.BlogState AS d ON b.StateId = d.StateId
```

	Headline	Content	LogDate	UserName	StateName
1	Optimist	Optimistjolle, også kaldet optimisten eller opti...	2015-09-29 08:58:54.540	Brian Munksgaard	Draft

```
Query executed successful... | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (57) | BlogDB | 00:00:00 | 1 rows
```

Q2 – Display assigned tags

Using the following SQL, we can display all tags associated with first blog post (PostId = 1):


DisplayAssignedTags.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPP...

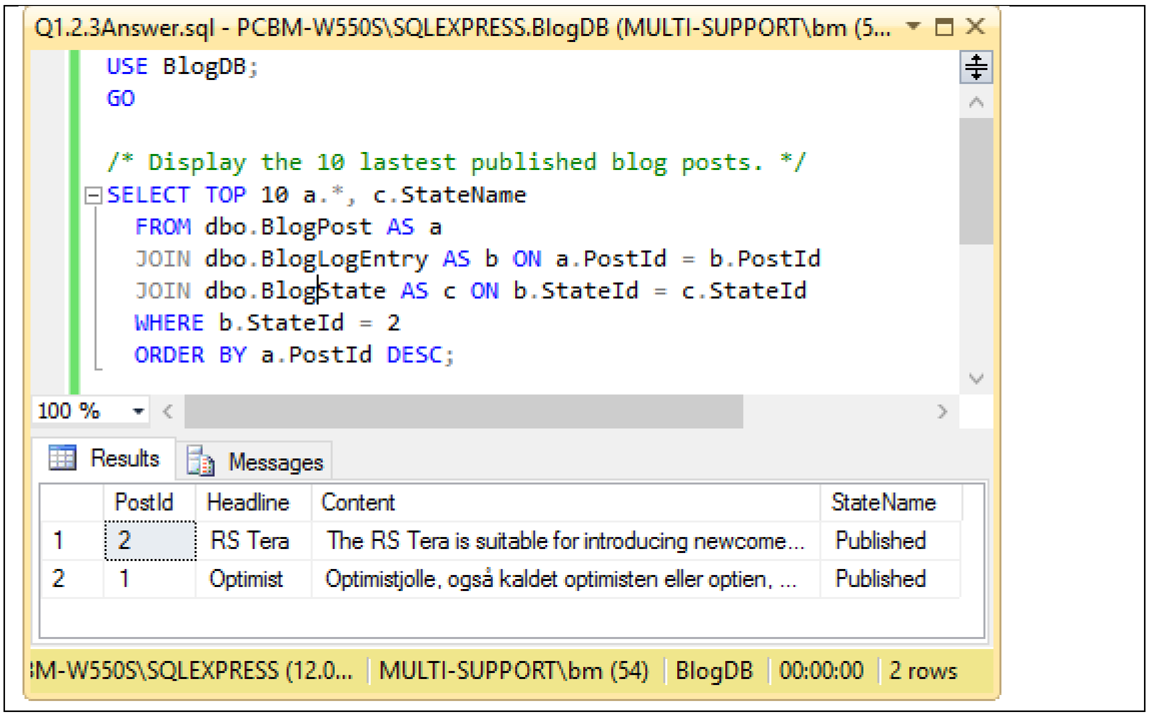
```
USE BlogDB;

SELECT a.Headline, c.TagId, c.TagName
FROM dbo.BlogPost AS a
JOIN dbo.BlogTagAssignment AS b ON a.PostId = b.PostId
JOIN dbo.BlogTag AS c on b.TagId = c.TagId
WHERE a.PostId = 1;
```

	Headline	TagId	TagName
1	Optimist	1	Mast
2	Optimist	2	Rig

N550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (57) | BlogDB | 00:00:00 | 2 rows

Q3 – Display 10 latest published posts

Using the following SQL, we can display the 10 latest published blog posts:


Q1.2.3Answer.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (5...

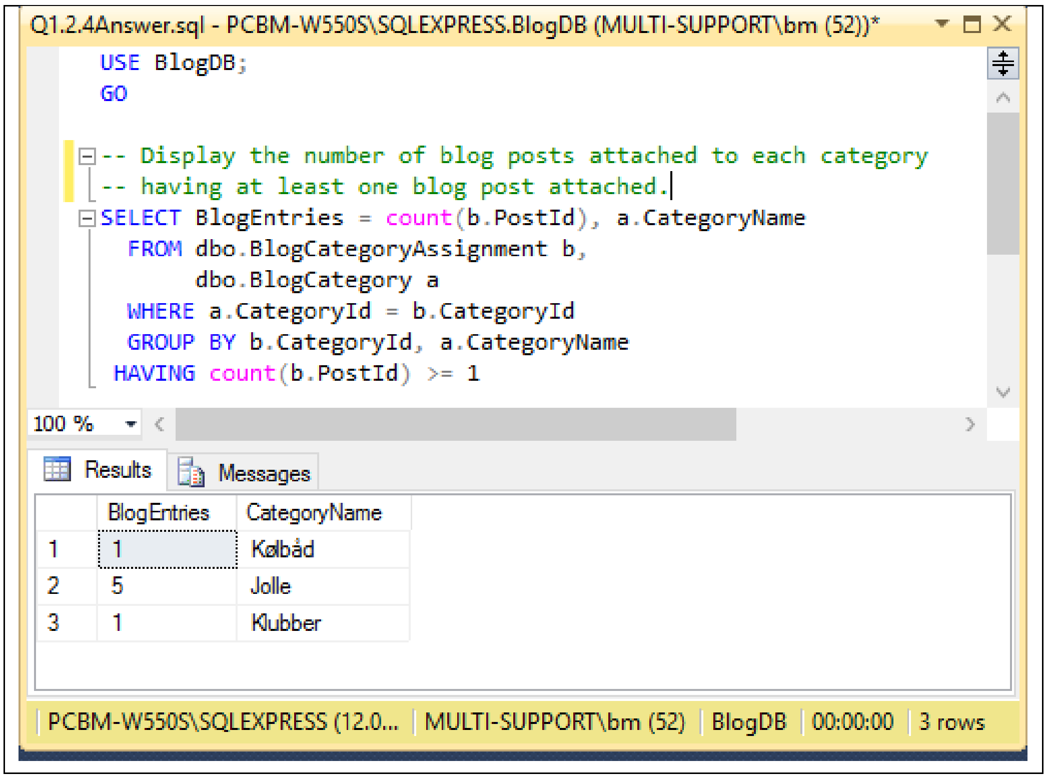
```
USE BlogDB;
GO

/* Display the 10 lastest published blog posts. */
SELECT TOP 10 a.*, c.StateName
  FROM dbo.BlogPost AS a
  JOIN dbo.BlogLogEntry AS b ON a.PostId = b.PostId
  JOIN dbo.BlogState AS c ON b.StateId = c.StateId
  WHERE b.StateId = 2
  ORDER BY a.PostId DESC;
```

	PostId	Headline	Content	StateName
1	2	RS Tera	The RS Tera is suitable for introducing newcome...	Published
2	1	Optimist	Optimistjolle, også kaldet optimisten eller optien, ...	Published

M-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (54) | BlogDB | 00:00:00 | 2 rows

Q4 – Display the number of blog posts attached to each category

Using the following SQL we can display the number of blog posts attached to each category having at least one blog post attached.



Q1.2.4Answer.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (52))*

```
USE BlogDB;
GO

-- Display the number of blog posts attached to each category
-- having at least one blog post attached.
SELECT BlogEntries = count(b.PostId), a.CategoryName
  FROM dbo.BlogCategoryAssignment b,
       dbo.BlogCategory a
  WHERE a.CategoryId = b.CategoryId
  GROUP BY b.CategoryId, a.CategoryName
 HAVING count(b.PostId) >= 1
```

100 %

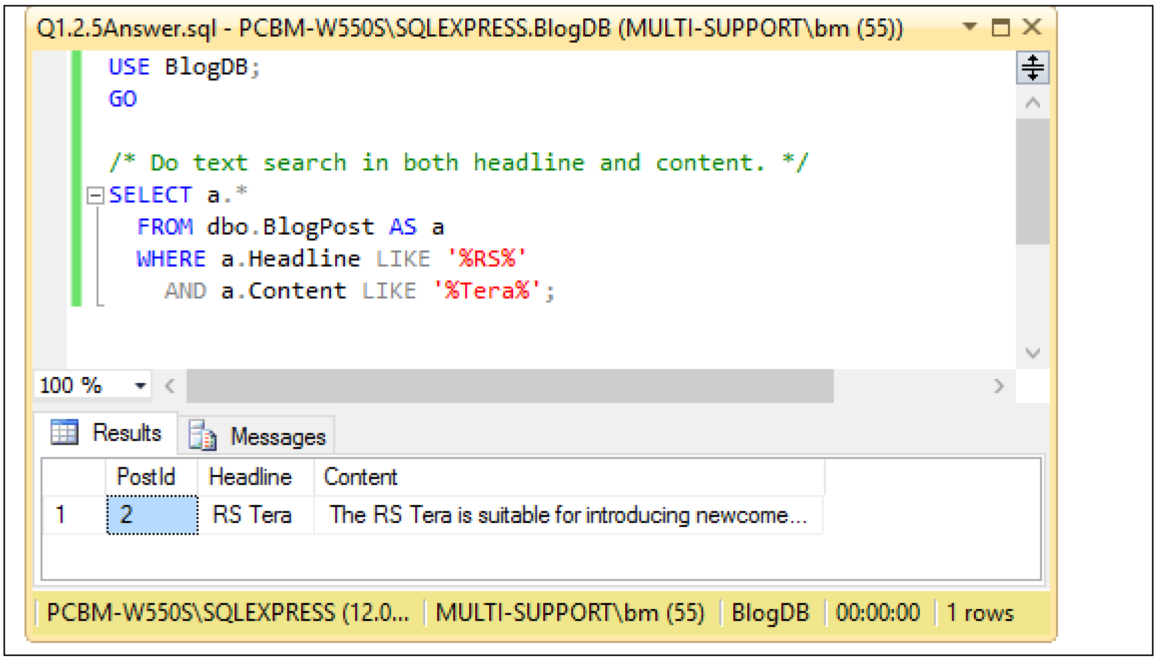
Results | Messages

	BlogEntries	CategoryName
1	1	Kølbåd
2	5	Jolle
3	1	Klubber

PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (52) | BlogDB | 00:00:00 | 3 rows

Q5 – Perform a text search in both headline and content of a blog post

Below is an example of a blog post text search. The headline has to contain 'RS' and the content has to contain 'Tera'.



Q1.2.5Answer.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (55))

```
USE BlogDB;
GO

/* Do text search in both headline and content. */
SELECT a.*
  FROM dbo.BlogPost AS a
  WHERE a.Headline LIKE '%RS%'
    AND a.Content LIKE '%Tera%';
```

100 %

Results | Messages

	PostId	Headline	Content
1	2	RS Tera	The RS Tera is suitable for introducing newcome...

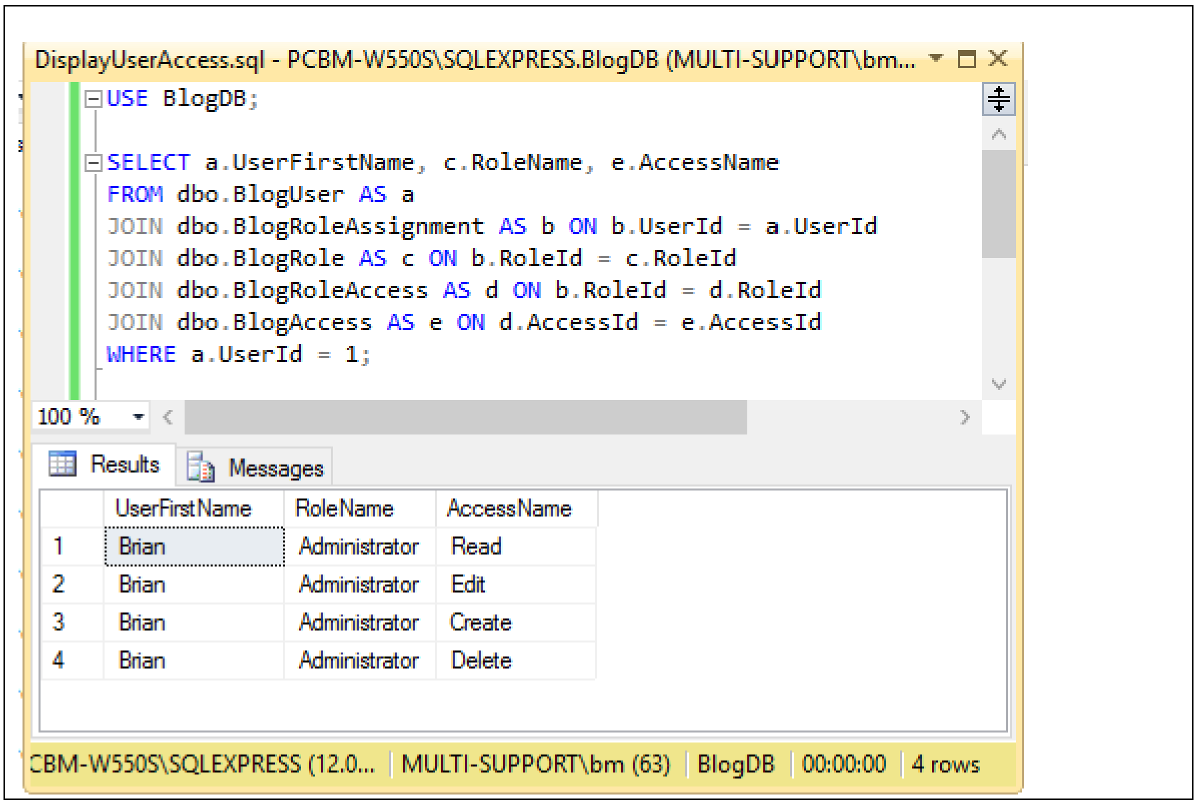
PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (55) | BlogDB | 00:00:00 | 1 rows

2. Expansion of blog

2.1 Expansion of the data model

The data model for the blog already handles user rights. As you can see in the E/R diagram, a user can be assigned one or more roles and a role can have one or more modes of access.

Querying the database like below, we can see that the user Brian is assigned to the Administrator role and that the administrator role has Read, Edit, Create and Delete access.


DisplayUserAccess.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm...
100 %
Results
Messages

```
USE BlogDB;

SELECT a.UserFirstName, c.RoleName, e.AccessName
FROM dbo.BlogUser AS a
JOIN dbo.BlogRoleAssignment AS b ON b.UserId = a.UserId
JOIN dbo.BlogRole AS c ON b.RoleId = c.RoleId
JOIN dbo.BlogRoleAccess AS d ON b.RoleId = d.RoleId
JOIN dbo.BlogAccess AS e ON d.AccessId = e.AccessId
WHERE a.UserId = 1;
```

	UserFirstName	RoleName	AccessName
1	Brian	Administrator	Read
2	Brian	Administrator	Edit
3	Brian	Administrator	Create
4	Brian	Administrator	Delete

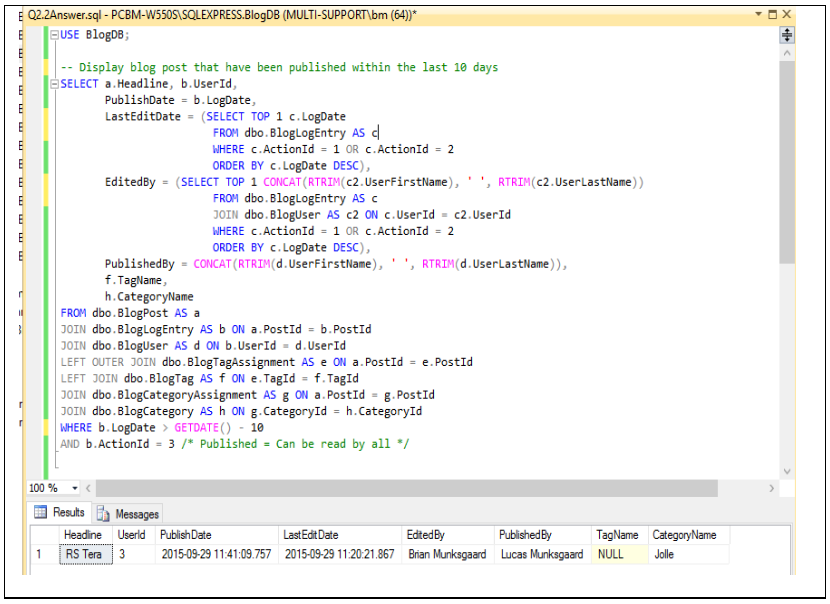
CBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (63) | BlogDB | 00:00:00 | 4 rows

Using role based access control is more convenient than controlling access for each user. Say that commentators should be able to delete blog post. Using role based access we need only assign delete rights to the commentator role. Using user based access control each user registered as a commentator needs to be updated.

2.2 Date

Using the BlogEntryLog table we can determine when various actions have been performed on a blog post.

In order to retrieve blog post from the lasts 10 days that are allowed to be read by all (read by all = published) we can use the following monster SQL statement:



Q2.2Answer.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (64))*

```sql
USE BlogDB;

-- Display blog post that have been published within the last 10 days
SELECT a.Headline, b.UserId,
       PublishDate = b.LogDate,
       LastEditDate = (SELECT TOP 1 c.LogDate
                       FROM dbo.BlogLogEntry AS c
                       WHERE c.ActionId = 1 OR c.ActionId = 2
                       ORDER BY c.LogDate DESC),
       EditedBy = (SELECT TOP 1 CONCAT(RTRIM(c2.UserFirstName), ' ', RTRIM(c2.UserLastName))
                       FROM dbo.BlogLogEntry AS c
                       JOIN dbo.BlogUser AS c2 ON c.UserId = c2.UserId
                       WHERE c.ActionId = 1 OR c.ActionId = 2
                       ORDER BY c.LogDate DESC),
       PublishedBy = CONCAT(RTRIM(d.UserFirstName), ' ', RTRIM(d.UserLastName)),
       f.TagName,
       h.CategoryName
FROM dbo.BlogPost AS a
JOIN dbo.BlogLogEntry AS b ON a.PostId = b.PostId
JOIN dbo.BlogUser AS d ON b.UserId = d.UserId
LEFT OUTER JOIN dbo.BlogTagAssignment AS e ON a.PostId = e.PostId
LEFT JOIN dbo.BlogTag AS f ON e.TagId = f.TagId
JOIN dbo.BlogCategoryAssignment AS g ON a.PostId = g.PostId
JOIN dbo.BlogCategory AS h ON g.CategoryId = h.CategoryId
WHERE b.LogDate > GETDATE() - 10
AND b.ActionId = 3 /* Published = Can be read by all */
```

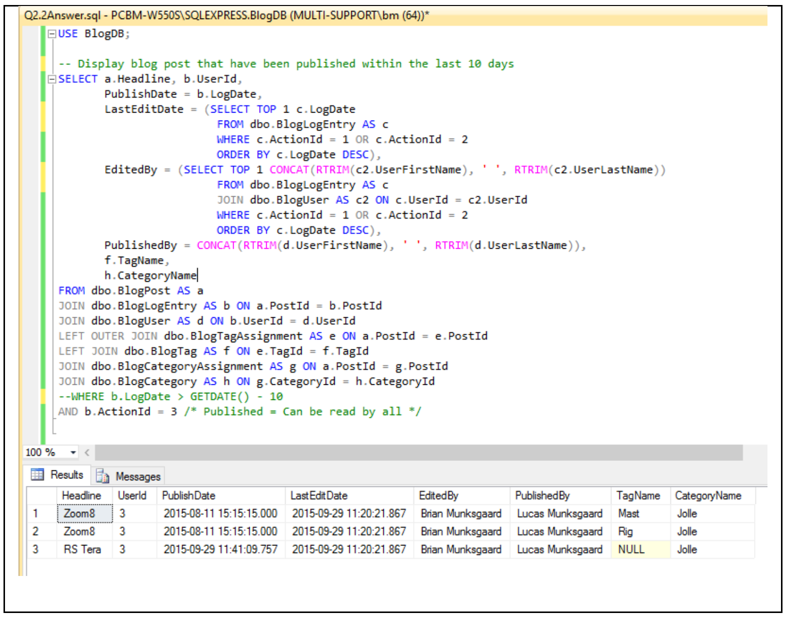
100 %

Results | Messages

	Headline	UserId	PublishDate	LastEditDate	EditedBy	PublishedBy	TagName	CategoryName
1	RS Tera	3	2015-09-29 11:41:09.757	2015-09-29 11:20:21.867	Brian Munksgaard	Lucas Munksgaard	NULL	Jolle

The LEFT OUTER JOIN on the BlogTagAssignment is needed because a blog post need not have a tag. A blog post must be in a category so no OUTER JOIN is needed here.

Ignoring the publish date, so we retrieve all published blog posts we get the Zoom8 post. This post has tags assigned which are also displayed:



Q2.2Answer.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (64))*

```sql
USE BlogDB;

-- Display blog post that have been published within the last 10 days
SELECT a.Headline, b.UserId,
       PublishDate = b.LogDate,
       LastEditDate = (SELECT TOP 1 c.LogDate
                       FROM dbo.BlogLogEntry AS c
                       WHERE c.ActionId = 1 OR c.ActionId = 2
                       ORDER BY c.LogDate DESC),
       EditedBy = (SELECT TOP 1 CONCAT(RTRIM(c2.UserFirstName), ' ', RTRIM(c2.UserLastName))
                       FROM dbo.BlogLogEntry AS c
                       JOIN dbo.BlogUser AS c2 ON c.UserId = c2.UserId
                       WHERE c.ActionId = 1 OR c.ActionId = 2
                       ORDER BY c.LogDate DESC),
       PublishedBy = CONCAT(RTRIM(d.UserFirstName), ' ', RTRIM(d.UserLastName)),
       f.TagName,
       h.CategoryName
FROM dbo.BlogPost AS a
JOIN dbo.BlogLogEntry AS b ON a.PostId = b.PostId
JOIN dbo.BlogUser AS d ON b.UserId = d.UserId
LEFT OUTER JOIN dbo.BlogTagAssignment AS e ON a.PostId = e.PostId
LEFT JOIN dbo.BlogTag AS f ON e.TagId = f.TagId
JOIN dbo.BlogCategoryAssignment AS g ON a.PostId = g.PostId
JOIN dbo.BlogCategory AS h ON g.CategoryId = h.CategoryId
--WHERE b.LogDate > GETDATE() - 10
AND b.ActionId = 3 /* Published = Can be read by all */
```

100 %

Results | Messages

	Headline	UserId	PublishDate	LastEditDate	EditedBy	PublishedBy	TagName	CategoryName
1	Zoom8	3	2015-08-11 15:15:15.000	2015-09-29 11:20:21.867	Brian Munksgaard	Lucas Munksgaard	Mast	Jolle
2	Zoom8	3	2015-08-11 15:15:15.000	2015-09-29 11:20:21.867	Brian Munksgaard	Lucas Munksgaard	Rig	Jolle
3	RS Tera	3	2015-09-29 11:41:09.757	2015-09-29 11:20:21.867	Brian Munksgaard	Lucas Munksgaard	NULL	Jolle

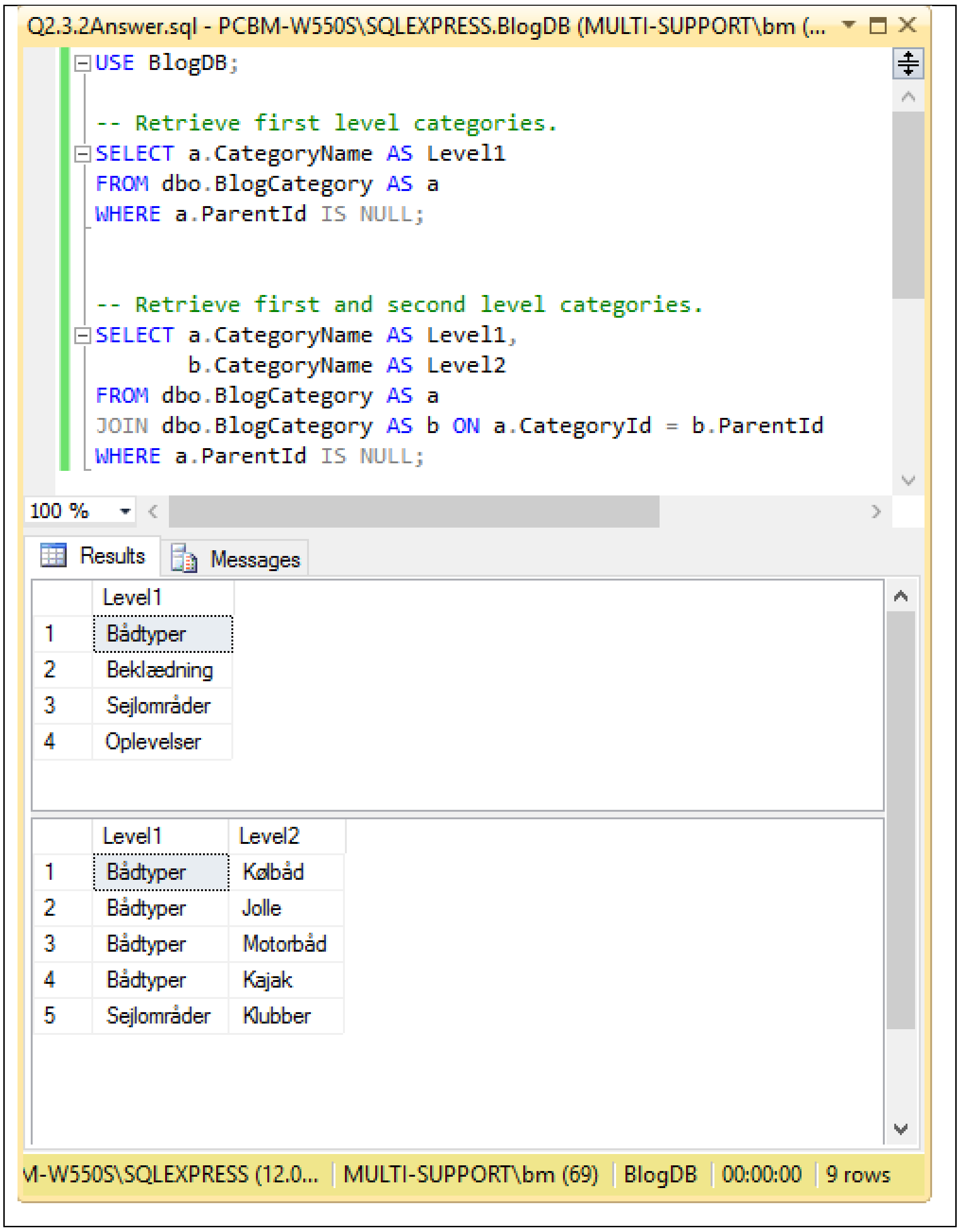
2.3 Recursive Categories

Q1 – Data model with self join

The data model already supports recursive categories. In the physical model this is implemented by adding an extra attribute ParentCategory to the BlogCategory table.

Q2/Q3 – First and second level categories

In order to retrieve all first level categories we use the first statement shown below. In order to retrieve all first and second level categories we use the second statement shown below:



Q2.3.2Answer.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (...

```sql
USE BlogDB;

-- Retrieve first level categories.
SELECT a.CategoryName AS Level1
FROM dbo.BlogCategory AS a
WHERE a.ParentId IS NULL;


-- Retrieve first and second level categories.
SELECT a.CategoryName AS Level1,
       b.CategoryName AS Level2
FROM dbo.BlogCategory AS a
JOIN dbo.BlogCategory AS b ON a.CategoryId = b.ParentId
WHERE a.ParentId IS NULL;
```

	Level1
1	Bådtyper
2	Beklædning
3	Sejlområder
4	Oplevelser

	Level1	Level2
1	Bådtyper	Kølbåd
2	Bådtyper	Jolle
3	Bådtyper	Motorbåd
4	Bådtyper	Kajak
5	Sejlområder	Klubber

M-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (69) | BlogDB | 00:00:00 | 9 rows

3 Automation of SQL in blog

3.1 What goes where

As a rule of thumb, stored procedures handles:

- Table creation
- Database updates including complex, multiple statements, updates to the database.
- Complex queries that requires conditional logic (if/then/else).
- Parameterized units of work that is to be executed over and over again.

Views, on the other hand, are simpler and are only used for retrieving data:

- No parameters.
- One SQL statement only.
- Views are queries, not updates. However, under certain conditions you can perform updates to the underlying tables in a view. In my humble opinion this is a no-go.
- Good for often used (complex) joins.

In the table below we can see which tasks got where, that is which task is done in either a stored procedure or a view.

Question	Function	SP or View
Q1.2.1	Add blog post	Stored procedure
Q1.2.2	Retrieve tags associated with a blog post	View
Q1.2.3	Retrieve published blog posts	View
Q1.2.4	Retrieve the number of blog post for the categories that have at least one blog post attached.	View
Q1.2.5	Perform text search in both blog headline and blog content.	Stored Procedure
Q2.2	Retrieve published blog posts with relevant meta data.	View
Q2.3	Retrieve category levels	View

3.2 Create Table Script

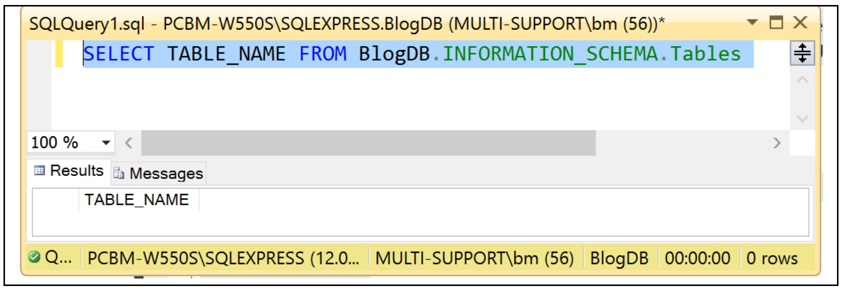
The following script creates the stored procedure DropCreateTables which creates all tables, primary keys and foreign key references used in the blog database.

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.DropCreateTables') IS NOT NULL DROP PROCEDURE dbo.DropCreateTables;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 17-09-2015
-- Description: This stored procedure is used to
--              delete and create the tables
--              used in the BlogDB.
-- =============================================
CREATE PROCEDURE dbo.DropCreateTables
AS
BEGIN
```

```
              /* First remove junction tables */
              IF OBJECT_ID('dbo.BlogLogEntry', 'U') IS NOT NULL DROP TABLE
dbo.BlogLogEntry;
              IF OBJECT_ID('dbo.BlogRoleAssignment', 'U') IS NOT NULL DROP TABLE
dbo.BlogRoleAssignment;
              IF OBJECT_ID('dbo.BlogImageAssignment', 'U') IS NOT NULL DROP TABLE
dbo.BlogImageAssignment;
              IF OBJECT_ID('dbo.BlogTagAssignment', 'U') IS NOT NULL DROP TABLE
dbo.BlogTagAssignment;
              IF OBJECT_ID('dbo.BlogCategoryAssignment', 'U') IS NOT NULL DROP TABLE
dbo.BlogCategoryAssignment;
              IF OBJECT_ID('dbo.BlogRoleAccess', 'U') IS NOT NULL DROP TABLE
dbo.BlogRoleAccess;

              /* Remove rest of the tables */
              IF OBJECT_ID('dbo.BlogAction', 'U') IS NOT NULL DROP TABLE dbo.BlogAction;
              IF OBJECT_ID('dbo.BlogRole', 'U') IS NOT NULL DROP TABLE dbo.BlogRole;
              IF OBJECT_ID('dbo.BlogState', 'U') IS NOT NULL DROP TABLE dbo.BlogState;
              IF OBJECT_ID('dbo.BlogCategory', 'U') IS NOT NULL DROP TABLE
dbo.BlogCategory;
              IF OBJECT_ID('dbo.BlogTag', 'U') IS NOT NULL DROP TABLE dbo.BlogTag;
              IF OBJECT_ID('dbo.BlogImage', 'U') IS NOT NULL DROP TABLE dbo.BlogImage;
              IF OBJECT_ID('dbo.BlogAccess', 'U') IS NOT NULL DROP TABLE dbo.BlogAccess;

              IF OBJECT_ID('dbo.BlogPost', 'U') IS NOT NULL DROP TABLE dbo.BlogPost;
              IF OBJECT_ID('dbo.BlogUser', 'U') IS NOT NULL DROP TABLE dbo.BlogUser;

              BEGIN TRANSACTION CreateTransaction;

              SET ANSI_NULLS ON;
              SET QUOTED_IDENTIFIER ON;

              CREATE TABLE dbo.BlogRole (
                            RoleId int IDENTITY(1,1) PRIMARY KEY,
                            RoleName nchar(30) NOT NULL
              ) ON [PRIMARY];

              CREATE TABLE dbo.BlogState(
                            StateId int IDENTITY(1,1) PRIMARY KEY,
                            StateName nchar(30) NOT NULL
              ) ON [PRIMARY];

              CREATE TABLE dbo.BlogUser(
                            UserId int IDENTITY(1,1) PRIMARY KEY,
                            UserFirstName nchar(30) NOT NULL,
                            UserLastName nchar(30) NOT NULL,
                            UserEmail nchar(75) NOT NULL
              ) ON [PRIMARY];

              CREATE TABLE dbo.BlogCategory (
                            CategoryId int IDENTITY(1,1) PRIMARY KEY,
                            CategoryName nchar(30) NOT NULL,
                            ParentId int FOREIGN KEY REFERENCES BlogCategory(CategoryId)
              ) ON [PRIMARY];

              CREATE TABLE dbo.BlogTag (
                            TagId int IDENTITY(1,1) PRIMARY KEY,
                            TagName nchar(30)
              ) ON [PRIMARY];

              CREATE TABLE dbo.BlogAccess (
                            AccessId int IDENTITY(1,1) PRIMARY KEY,
                            AccessName nchar(30) NOT NULL
              ) ON [PRIMARY];

              CREATE TABLE dbo.BlogRoleAccess (
                            RoleId int FOREIGN KEY REFERENCES BlogRole(RoleId),
                            AccessId int FOREIGN KEY REFERENCES BlogAccess(AccessId)
                            PRIMARY KEY (RoleId, AccessId)
              ) ON [PRIMARY];

              CREATE TABLE dbo.BlogRoleAssignment (
                            UserId int FOREIGN KEY REFERENCES BlogUser(UserId),
                            RoleId int FOREIGN KEY REFERENCES BlogRole(RoleId)
```

```
                        PRIMARY KEY (UserId, RoleId)
            ) ON [PRIMARY];

            CREATE TABLE dbo.BlogImage (
                        ImageId int IDENTITY(1,1) PRIMARY KEY,
                        ImageDescription nchar(50) NOT NULL,
                        ImageData varbinary NOT NULL
            ) ON [PRIMARY];

            CREATE TABLE dbo.BlogPost (
                        PostId int IDENTITY(1,1) PRIMARY KEY,
                        Headline nchar(30) NOT NULL,
                        Content nvarchar(MAX) NOT NULL
            ) ON [PRIMARY];

            CREATE TABLE dbo.BlogImageAssignment (
                        PostId int FOREIGN KEY REFERENCES BlogPost(PostId),
                        ImageId int FOREIGN KEY REFERENCES BlogImage(ImageId)
                        PRIMARY KEY (PostId, ImageId)
            ) ON [PRIMARY];

            CREATE TABLE dbo.BlogTagAssignment (
                        PostId int FOREIGN KEY REFERENCES BlogPost(PostId),
                        TagId int FOREIGN KEY REFERENCES BlogTag(TagId)
                        PRIMARY KEY (PostId, TagId)
            ) ON [PRIMARY];

            CREATE TABLE dbo.BlogCategoryAssignment (
                        PostId int FOREIGN KEY REFERENCES BlogPost(PostId),
                        CategoryId int FOREIGN KEY REFERENCES BlogCategory(CategoryId)
                        PRIMARY KEY (PostId, CategoryId)
            ) ON [PRIMARY];

            CREATE TABLE dbo.BlogAction (
                        ActionId int IDENTITY(1,1) PRIMARY KEY,
                        ActionName nchar(30) NOT NULL
            ) ON [PRIMARY];

            CREATE TABLE dbo.BlogLogEntry (
                        LogId int IDENTITY(1,1) PRIMARY KEY,
                        LogDate datetime,
                        PostId int FOREIGN KEY REFERENCES BlogPost(PostId),
                        ActionId int FOREIGN KEY REFERENCES BlogAction(ActionId),
                        StateId int FOREIGN KEY REFERENCES BlogState(StateId),
                        UserId int FOREIGN KEY REFERENCES BlogUser(UserId),
            ) ON [PRIMARY];

            IF @@ERROR <> 0
                        BEGIN
                                    ROLLBACK TRANSACTION CreateTransaction;
                        END
            ELSE
                        BEGIN
                                    COMMIT TRANSACTION CreateTransaction;
                        END;
END
```

Using the SQL below, we can see that we have a blog database without any tables:



SQLQuery1.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (56))*

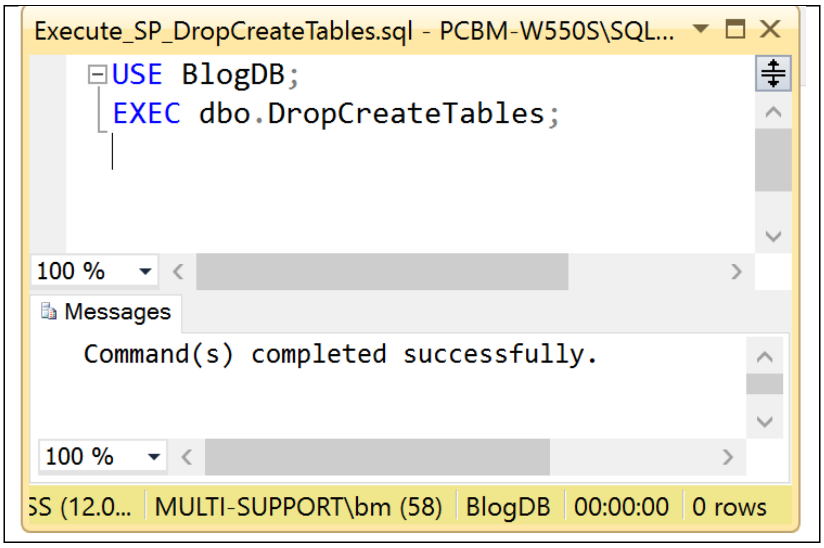
```
SELECT TABLE_NAME FROM BlogDB.INFORMATION_SCHEMA.Tables
```

Results | Messages

```
TABLE_NAME
```

Q... | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (56) | BlogDB | 00:00:00 | 0 rows

After executing the stored procedure like:



Execute_SP_DropCreateTables.sql - PCBM-W550S\SQL...

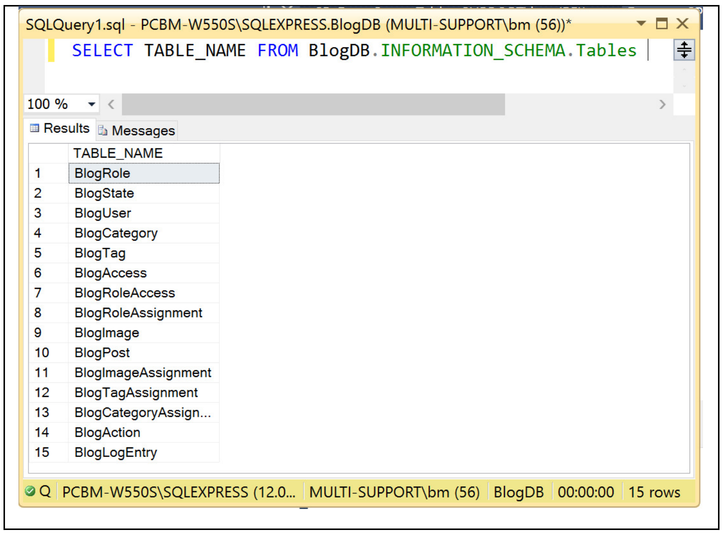
```
USE BlogDB;
EXEC dbo.DropCreateTables;
```

Messages

```
Command(s) completed successfully.
```

SS (12.0... | MULTI-SUPPORT\bm (58) | BlogDB | 00:00:00 | 0 rows

We can list that tables in the blog database again:



SQLQuery1.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (56))*

```
SELECT TABLE_NAME FROM BlogDB.INFORMATION_SCHEMA.Tables
```

Results | Messages

	TABLE_NAME
1	BlogRole
2	BlogState
3	BlogUser
4	BlogCategory
5	BlogTag
6	BlogAccess
7	BlogRoleAccess
8	BlogRoleAssignment
9	BlogImage
10	BlogPost
11	BlogImageAssignment
12	BlogTagAssignment
13	BlogCategoryAssign...
14	BlogAction
15	BlogLogEntry

Q | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (56) | BlogDB | 00:00:00 | 15 rows

And we can now confirm that the tables have been created.

3.3 Create Foreign Keys

The foreign key are created during table creation.

3.4 Create Test Data

Below we can see the SQL script, that creates the stored procedures used to populate the blog database with test data. Three stored procedures are created:

Stored Procedure	Purpose
CleanupDB	Drops and creates all database tables.
PopulateDB	Populates entities.
PopulateDBJunctions	Populates junctions/relations.

SQL Script for creating stored procedures used to populate the DB:

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.CleanupDB') IS NOT NULL DROP PROCEDURE dbo.CleanupDB;
IF OBJECT_ID('dbo.PopulateDB') IS NOT NULL DROP PROCEDURE dbo.PopulateDB;
IF OBJECT_ID('dbo.PopulateDBJunctions') IS NOT NULL DROP PROCEDURE
dbo.PopulateDBJunctions;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 01-10-2015
-- Description: This stored procedure is used to
--              cleanup the BlogDB tables by
--              dropping and creating them.
-- =============================================
CREATE PROCEDURE dbo.CleanupDB
AS
BEGIN
              EXEC dbo.DropCreateTables;
END;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 01-10-2015
-- Description: This stored procedure is used to
--              populate the BlogDB with test
--              data.
-- =============================================
CREATE PROCEDURE dbo.PopulateDB
AS
BEGIN

              BEGIN TRANSACTION PopulateTransaction;

              /* Insert actions */
              INSERT INTO dbo.BlogAction (ActionName)
              VALUES
                            ('Create'),
                            ('Edit'),
                            ('Publish'),
                            ('Hide'),
                            ('Delete')
```

```
            /* Insert roles */
            INSERT INTO dbo.BlogRole (RoleName)
            VALUES
                                        ('Author'),
                                        ('Administrator'),
                                        ('Commentator');

            /* Insert blog access modes */
            INSERT INTO dbo.BlogAccess (AccessName)
            VALUES
                          ('Read'),
                          ('Edit'),
                          ('Create'),
                          ('Delete');

            /* Insert categories */
            INSERT INTO dbo.BlogCategory (CategoryName, ParentId)
            VALUES
                          ('Bådtyper', NULL),     -- Top level category
                          ('Kølbåd', 1),          -- Parent -> Bådtyper
                          ('Jolle', 1),           -- Parent -> Bådtyper
                          ('Motorbåd', 1),        -- Parent -> Bådtyper
                          ('Kajak', 1),           -- Parent -> Bådtyper
                          ('Beklædning', NULL),   -- Top level category
                          ('Sejlområder', NULL), -- Top level category
                          ('Klubber', 7),         -- Parent -> Sejlområder
                          ('Oplevelser', NULL);   -- Top level category

            /* Insert blog states */
            INSERT INTO dbo.BlogState (StateName)
            VALUES
                          ('Draft'),
                          ('Published'),
                          ('Archived'),
                          ('Hidden');

            /* Insert blog tags */
            INSERT INTO dbo.BlogTag (TagName)
            VALUES
                          ('Mast'),
                          ('Rig'),
                          ('Fortøjring'),
                          ('Påhængsmotor');

            /* Insert blog users */
            INSERT INTO dbo.BlogUser (UserFirstName, UserLastName, UserEmail)
            VALUES
                          ('Brian', 'Munksgaard', 'brian.munksgaard@gmail.com'),
                          ('Emil', 'Munksgaard', 'emil.munksgaard@gmail.com'),
                          ('Lucas', 'Munksgaard', 'lucas.munksgaard@gmail.com'),
                          ('Mikkel', 'Munksgaard', 'mikkelm.munksgaard@gmail.com');

            IF @@ERROR <> 0
                          BEGIN
                                        ROLLBACK TRANSACTION PopulateTransaction;
                                        RETURN 1;
                          END
            ELSE
                          BEGIN
                                        COMMIT TRANSACTION PopulateTransaction;
                                        RETURN 0;
                          END;

END;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 01-10-2015
-- Description: This stored procedure is used to
--              populate the BlogDB junction
--              tables with test data.
-- =============================================
CREATE PROCEDURE dbo.PopulateDBJunctions
AS
BEGIN

            BEGIN TRANSACTION PopulateTransaction;

            /* Assign role access */
            INSERT INTO dbo.BlogRoleAccess (RoleId, AccessId)
            VALUES
                          (1, 1), /* Author read */
                          (1, 2), /* Author edit */
                          (1, 3), /* Author create */
                          (2, 1), /* Administrator read */
                          (2, 2), /* Administrator edit */
                          (2, 3), /* Administrator create */
                          (2, 4), /* Administrator delete */
                          (3, 1), /* Commentator read */
                          (3, 2); /* Commentator edit */

            /* Assign roles to users */
            INSERT INTO dbo.BlogRoleAssignment (UserId, RoleId)
            VALUES
                          (1, 2), /* Brian som Administrator */
                          (2, 3), /* Emil som Commentator. */
                          (3, 1), /* Lucas som Author */
                          (4, 3); /* Mikkel som Commentator */

            IF @@ERROR <> 0
                          BEGIN
                                        ROLLBACK TRANSACTION PopulateTransaction;
                                        RETURN 1;
                          END
            ELSE
                          BEGIN
                                        COMMIT TRANSACTION PopulateTransaction;
                                        RETURN 0;
                          END;
END;
```

The stored procedures are executed like below:

```
USE BlogDB;

-- Clean database.
EXEC dbo.CleanupDB;

DECLARE @Status AS int;
SET @Status = 0;

-- Populate entities.
EXEC @Status = dbo.PopulateDB;

-- Populate junctions/relations.
IF @Status = 0
BEGIN
              EXEC @Status = dbo.PopulateDBJunctions;
END;

-- Any errors? Cleanup database.
IF @Status <> 0
BEGIN
              PRINT 'Errors have occurred.';
              EXEC dbo.CleanupDB;
END;
```

3.5 Create Relevant Views

Previously the following view candidates where identified:

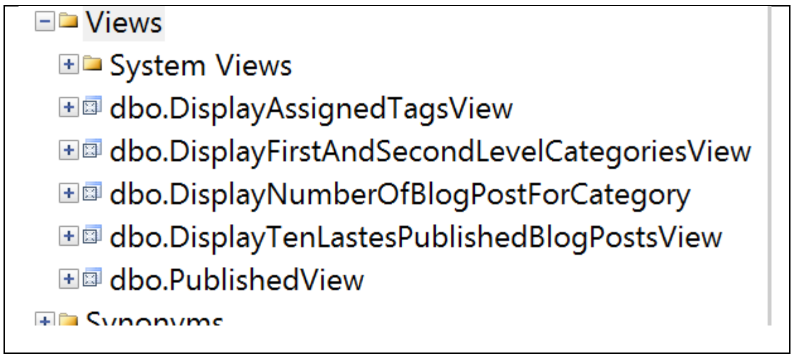
Question	Function	SP or View
Q1.2.2	Retrieve tags associated with a blog post	View
Q1.2.3	Retrieve published blog posts	View
Q1.2.4	Retrieve the number of blog post for the categories that have at least one blog post attached.	View
Q2.2	Retrieve published blog posts with relevant meta data.	View
Q2.3	Retrieve category levels	View

The views are created with the SQL script shown below:

```
-- Retrieve tags associated with a blog post.
CREATE VIEW DisplayAssignedTagsView AS
SELECT a.Headline, c.TagId, c.TagName
FROM dbo.BlogPost AS a
JOIN dbo.BlogTagAssignment AS b ON a.PostId = b.PostId
JOIN dbo.BlogTag AS c on b.TagId = c.TagId;
GO

-- Retrieve latest 10 published blog posts.
CREATE VIEW DisplayTenLastesPublishedBlogPostsView
AS
SELECT TOP 10 a.*, c.StateName
  FROM dbo.BlogPost AS a
  JOIN dbo.BlogLogEntry AS b ON a.PostId = b.PostId
  JOIN dbo.BlogState AS c ON b.StateId = c.StateId
  WHERE b.StateId = 2
  ORDER BY a.PostId DESC;
GO

-- Retrieve the number of blog post for the categories.
-- that have at least one blog post attached.
CREATE VIEW DisplayNumberOfBlogPostForCategory
AS
SELECT BlogEntries = count(b.PostId), a.CategoryName
  FROM dbo.BlogCategoryAssignment b,
       dbo.BlogCategory a
  WHERE a.CategoryId = b.CategoryId
  GROUP BY b.CategoryId, a.CategoryName
 HAVING count(b.PostId) >= 1
GO

-- Retrieve published blog posts with relevant meta data.
CREATE VIEW dbo.PublishedView AS
SELECT a.Headline, b.UserId,
       PublishDate = b.LogDate,
       LastEditDate = (SELECT TOP 1 c.LogDate
                                  FROM dbo.BlogLogEntry AS c
                                                                        WHERE c.ActionId =
1 OR c.ActionId = 2
                                                                                  ORDER BY
c.LogDate DESC),
       EditedBy = (SELECT TOP 1 CONCAT(RTRIM(c2.UserFirstName), ' ',
RTRIM(c2.UserLastName))
                                                                                  FROM
dbo.BlogLogEntry AS c
                                                                                  JOIN
dbo.BlogUser AS c2 ON c.UserId = c2.UserId
                                                                        WHERE c.ActionId =
1 OR c.ActionId = 2
```

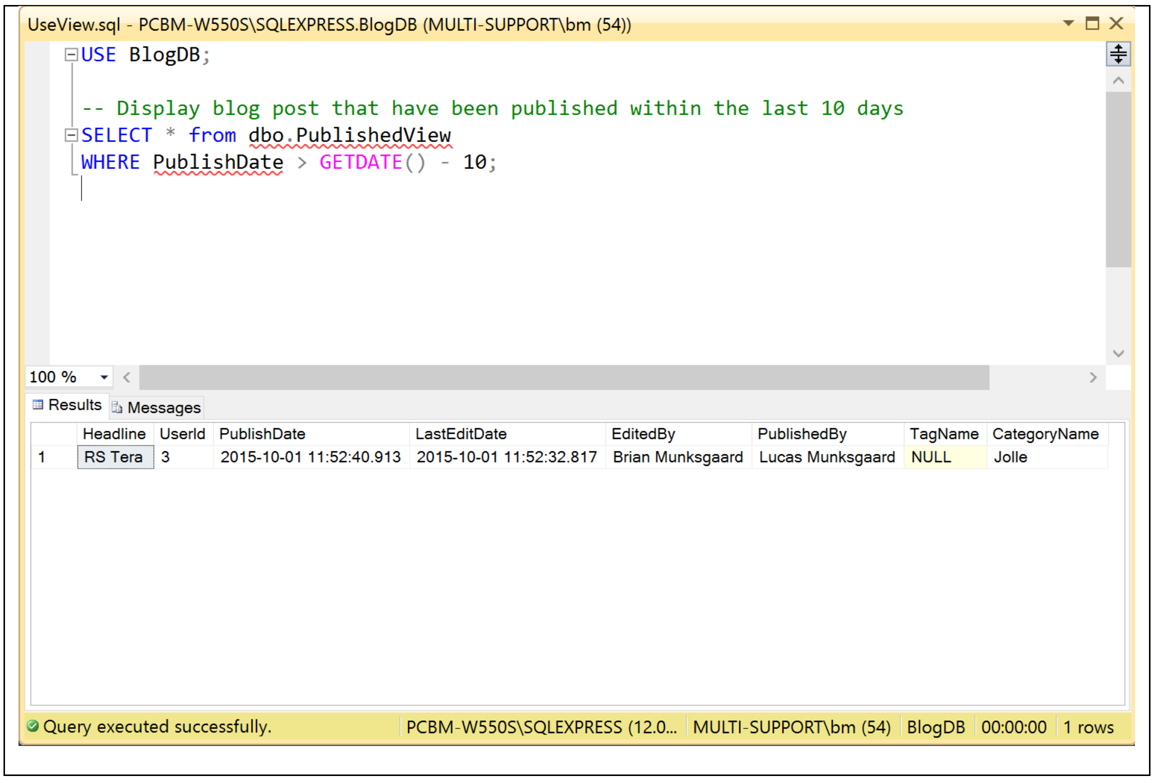
```
                                                                                       ORDER BY
c.LogDate DESC),
                PublishedBy = CONCAT(RTRIM(d.UserFirstName), ' ', RTRIM(d.UserLastName)),
                f.TagName,
                h.CategoryName
FROM dbo.BlogPost AS a
JOIN dbo.BlogLogEntry AS b ON a.PostId = b.PostId
JOIN dbo.BlogUser AS d ON b.UserId = d.UserId
LEFT OUTER JOIN dbo.BlogTagAssignment AS e ON a.PostId = e.PostId
LEFT JOIN dbo.BlogTag AS f ON e.TagId = f.TagId
JOIN dbo.BlogCategoryAssignment AS g ON a.PostId = g.PostId
JOIN dbo.BlogCategory AS h ON g.CategoryId = h.CategoryId
AND b.ActionId = 3 /* Published = Can be read by all */
;
GO

-- Display first and second level categories.
CREATE VIEW DisplayFirstAndSecondLevelCategoriesView AS
SELECT a.CategoryName AS Level1,
       b.CategoryName AS Level2
FROM dbo.BlogCategory AS a
JOIN dbo.BlogCategory AS b ON a.CategoryId = b.ParentId
WHERE a.ParentId IS NULL;
GO
```

After running the script we can see that the views have been created in the database:



```
Views
  System Views
  dbo.DisplayAssignedTagsView
  dbo.DisplayFirstAndSecondLevelCategoriesView
  dbo.DisplayNumberOfBlogPostForCategory
  dbo.DisplayTenLastesPublishedBlogPostsView
  dbo.PublishedView
Synonyms
```

In order to display the blog posts published within the last 10 days along with relevant meta data we use the PublishedView view:



UseView.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (54))

```
USE BlogDB;

-- Display blog post that have been published within the last 10 days
SELECT * from dbo.PublishedView
WHERE PublishDate > GETDATE() - 10;
```

	Headline	UserId	PublishDate	LastEditDate	EditedBy	PublishedBy	TagName	CategoryName
1	RS Tera	3	2015-10-01 11:52:40.913	2015-10-01 11:52:32.817	Brian Munksgaard	Lucas Munksgaard	NULL	Jolle

Query executed successfully. | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (54) | BlogDB | 00:00:00 | 1 rows

3.6 Create Relevant Stored Procedures

Previously the following stored procedure candidates where identified:

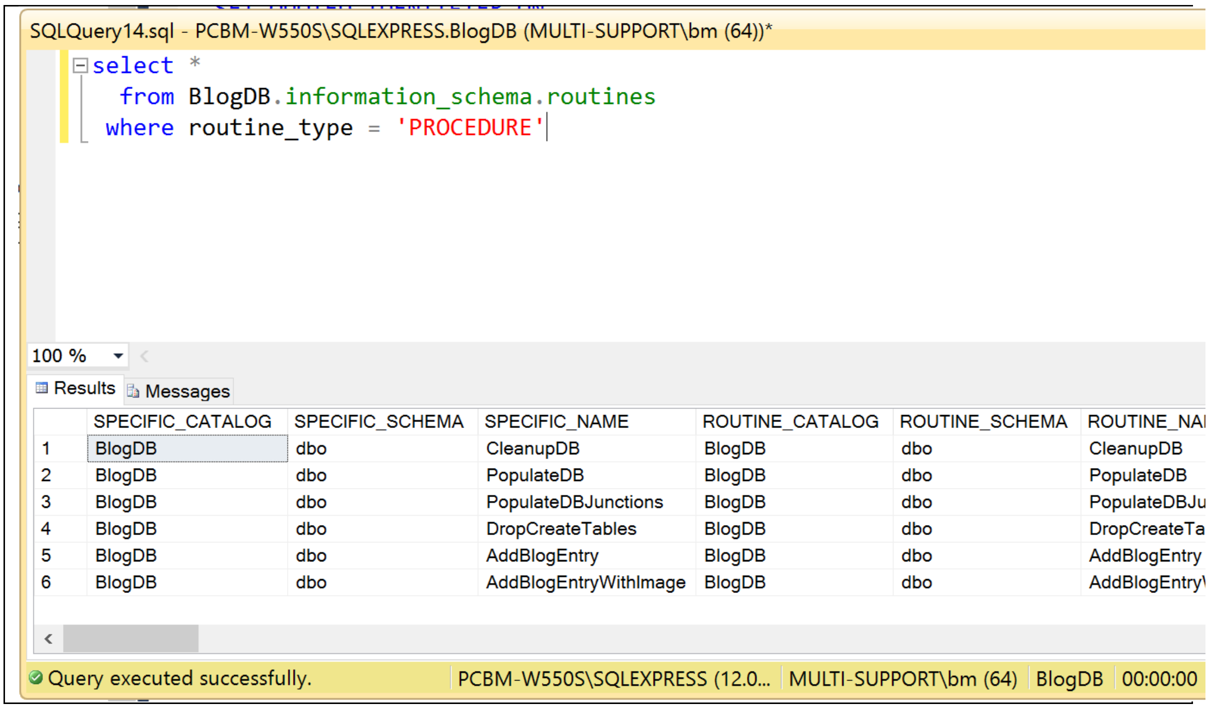
Question	Function	SP or View
Q1.2.1	Add blog post (AddBlogEntry)	Stored procedure
Q1.2.5	Perform text search in both blog headline and blog content (SearchBlogEntry).	Stored Procedure

AddBlogEntry

The procedure AddBlogEntry is created with the statements below:

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.AddBlogEntry') IS NOT NULL DROP PROCEDURE dbo.AddBlogEntry;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 30-09-2015
-- Description:             This stored procedure is used to
--                   add a blog entry to the blog.
-- =============================================
CREATE PROCEDURE dbo.AddBlogEntry
              @UserId int,
              @CategoryId int,
              @Headline nchar(30),
              @Content nvarchar(MAX)
AS
BEGIN

              BEGIN TRANSACTION AddBlogEntryTransaction;

              -- Add the post.
              INSERT INTO dbo.BlogPost (Headline, Content)
              VALUES(@Headline, @Content);

              DECLARE @PostId AS int;
              SET @PostId = SCOPE_IDENTITY();

              -- Assign the category.
              INSERT INTO dbo.BlogCategoryAssignment (PostId, CategoryId)
              VALUES (@PostId, @CategoryId);

              -- Update the log.
              INSERT INTO dbo.BlogLogEntry (LogDate, PostId, ActionId, StateId, UserId)
              VALUES (
                            GETDATE(),
                            @PostId,
                            1, -- Action: Create.
                            1, -- State: Draft.
                            @UserId
              );

              IF @@ERROR <> 0
                            BEGIN
                                          ROLLBACK TRANSACTION AddBlogEntryTransaction;
                            END
              ELSE
                            BEGIN
                                          COMMIT TRANSACTION AddBlogEntryTransaction;
                            END;
END;
```

After running the script above, we now have the following stored procedures.


SQLQuery14.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (64))*

```sql
select *
  from BlogDB.information_schema.routines
 where routine_type = 'PROCEDURE'
```

	SPECIFIC_CATALOG	SPECIFIC_SCHEMA	SPECIFIC_NAME	ROUTINE_CATALOG	ROUTINE_SCHEMA	ROUTINE_NAI
1	BlogDB	dbo	CleanupDB	BlogDB	dbo	CleanupDB
2	BlogDB	dbo	PopulateDB	BlogDB	dbo	PopulateDB
3	BlogDB	dbo	PopulateDBJunctions	BlogDB	dbo	PopulateDBJu
4	BlogDB	dbo	DropCreateTables	BlogDB	dbo	DropCreateTa
5	BlogDB	dbo	AddBlogEntry	BlogDB	dbo	AddBlogEntry
6	BlogDB	dbo	AddBlogEntryWithImage	BlogDB	dbo	AddBlogEntry\

Query executed successfully. | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (64) | BlogDB | 00:00:00

We can then add blog entries by executing the stored procedure:

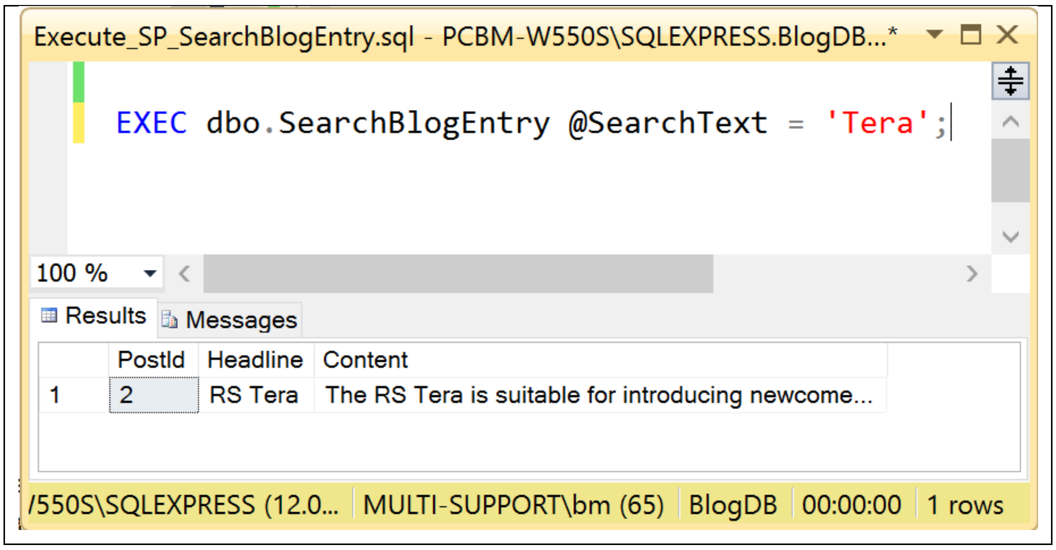
```sql
USE BlogDB;

EXEC dbo.AddBlogEntry
              @UserId = 1, -- Brian
              @CategoryId = 3, -- Jolle
              @Headline = 'Optimist',
              @Content = 'Optimistjolle, også kaldet optimisten eller optien, er
standarden for enmandsjoller, og er generelt den mest udbredte og hurtigst voksende
jolletype. Oftest starter man med at sejle den, omkring 7-14 års alderen. Jollen styres og
trimmes udelukkende med roret, sværdet, skødet, sprydstagen, bomnedhalet og sejlerens
placering i båden. Optimistsejlerne blive delt op i A-, B-, og C-sejlere, hvor c-sejlere
er de mindst øvede og a-sejlere er mest øvede.';

EXEC dbo.AddBlogEntry
              @UserId = 1, -- Brian
              @CategoryId = 3, -- Jolle
              @Headline = 'RS Tera',
              @Content = 'The RS Tera is suitable for introducing newcomers to the sport
of sailing, but is also a good boat to race.[2] The boat is highly robust, and it is built
with a self draining cockpit and is easy to right after a capsize, in addition to which it
has a floating daggerboard. The boat is fairly small and light, meaning it is possible to
transport on a roof rack, and that it is manageable on the water by younger children. The
mast comes in two pieces, and the boom is padded. Furthermore, the RS Tera can be rowed
and has oarlocks.[3] Built with a Comptec PE3 hull, the RS Tera has been described to have
a modern look.';

EXEC dbo.AddBlogEntry
              @UserId = 1, -- Brian
              @CategoryId = 2, -- Kølbåd
              @Headline = 'J/70',
              @Content = 'The J/70 Speedster (22.75 feet) is J/Boats first ramp-launchable
keelboat - designed to fulfill the growing need for an easy-to-own, high performance one-
design that is exciting to sail, stable enough sailboat for the family, and built to last.
With fleet discussions underway around the world, J/70 is on-track to take the world by
storm.';

EXEC dbo.AddBlogEntry
              @UserId = 1, -- Brian
              @CategoryId = 3, -- Jolle
              @Headline = 'RS Feva',
              @Content = 'The RS Feva is a two-person sailing dinghy designed by Paul
Handley in 2002. It is manufactured and distributed by RS Sailing.The RS Feva is an
International Sailing Federation (ISAF) International Class, a Royal Yachting Association
(RYA) Supported Junior Class, and has been selected by the Dansk Sejlunion (Danish Sailing
Association) and Norges Seilforbund (Norwegian Sailing Federation) for major sailing
growth projects.';

EXEC dbo.AddBlogEntry
              @UserId = 1, -- Brian
              @CategoryId = 3, -- Jolle
              @Headline = 'Laser',
              @Content = 'The Laser is one of the most popular single-handed dinghies in
the world. As of 2012, there are more than 200,000 boats worldwide. A commonly cited
reason for its popularity is that it is robust and simple to rig and sail in addition to
its durability. The Laser also provides very competitive racing due to the very tight
class association controls which eliminate differences in hull, sails and equipment.';

EXEC dbo.AddBlogEntry
              @UserId = 1, -- Brian
              @CategoryId = 8, -- Klub
              @Headline = 'Silkeborg Sejlklub',
              @Content = 'Klubben er en hyggelig lille sejlklub, hvor vi nyder
sejlerlivets mange glæder på Silkeborgsøerne. Vi byder velkommen til enkeltpersoner,
voksne som børn og familier. ';
```

SearchBlogEntry

The stored procedure SearchBlogEntry is created using the script below:

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.SearchBlogEntry') IS NOT NULL DROP PROCEDURE dbo.SearchBlogEntry;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 01-10-2015
-- Description:             This stored procedure is used to
--              perform a text search in the blog
--                                               entries. Both Headline and Content
--              attributes are searched. If either
--              one contains the search text,
--              the blog entry is returned.
-- =============================================
CREATE PROCEDURE dbo.SearchBlogEntry
              @SearchText varchar(50)
AS
BEGIN

              DECLARE @_SearchText AS varchar(50);
              SET @_SearchText = '%' + LTRIM(RTRIM(@SearchText)) + '%';

              PRINT @_SearchText;

              SELECT *
              FROM dbo.BlogPost AS a
              WHERE a.Headline LIKE @_SearchText
    OR a.Content LIKE @_SearchText;

END;
```

We can then do a search by calling the stored procedure:



Execute_SP_SearchBlogEntry.sql - PCBM-W550S\SQLEXPRESS.BlogDB...*
100 %
Results
Messages
/550S\SQLEXPRESS (12.0... MULTI-SUPPORT\bm (65) BlogDB 00:00:00 1 rows

```
EXEC dbo.SearchBlogEntry @SearchText = 'Tera';
```

	PostId	Headline	Content
1	2	RS Tera	The RS Tera is suitable for introducing newcome...

The search is a contains-search, but since the search is done in a stored procedure, the developer need not know how this is done in SQL.

3.7 Create new Views and Stored Procedures

The following stored procedures and views may assist developers in their daily work with the database:

Type	Name	Description
SP	PublishBlogEntry	Like adding an entry to the blog, publishing an entry should also be handled in a stored procedure. The same reasons apply: One unit of work that either fails or succeeds and is executed as close to database as possible. Also, future changes to the publishing procedure are handled in the procedure. Influence on the application developer is at a minimum.
SP	AddDefaultUser	Adding a user to the database involves multiple tables that needs to updated and/or inserted into (Adding the user, Assigning a role). Considering this as a unit of work that either fails or succeeds, the ideal solution is to use a stored procedure.
View	DisplayDraftsView	Managing a blog also mean keeping information on the blog and in the blog database in a good state. As a blog administrator you might want the possibility to identify drafts older than x days and not being worked on, so proper action (delete, notify user etc.) can be taken.
View	DisplayRoleAssignmentsView	In a production system it is not advisable to have too many administrators. Using this view you can quickly list the number users assigned to each role.

PublishBlogEntry

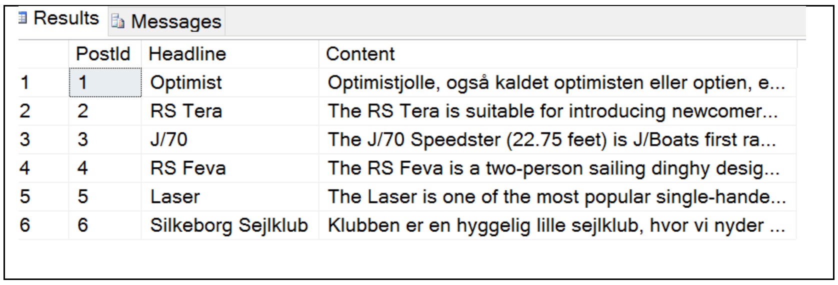
The script for creating the PublishBlogEntry stored procedure is shown below:

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.PublishBlogEntry') IS NOT NULL DROP PROCEDURE dbo.PublishBlogEntry;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 01-10-2015
-- Description:             This stored procedure is used to
--              publish a blog entry to the blog.
-- =============================================
CREATE PROCEDURE dbo.PublishBlogEntry
              @UserId int,
              @PostId int
AS
BEGIN

              BEGIN TRANSACTION PostBlogEntryTransaction;

              -- First determine whether or not the user has the required
              -- access rights.
              DECLARE @CanEdit AS int;
              EXEC @CanEdit = dbo.CanEdit @UserId;

              -- If the user has edit rights mark the blog entry as published.
              IF @CanEdit = 0
              BEGIN
                            INSERT INTO dbo.BlogLogEntry (LogDate, PostId, ActionId,
StateId, UserId)
                            VALUES (
                                          GETDATE(),
                                          @PostId,
                                          3, /* Action: Publish */
                                          2, /* State: Published */
                                          @UserId);
              END

              IF @@ERROR <> 0
                            BEGIN
                                          ROLLBACK TRANSACTION PostBlogEntryTransaction;
                            END
              ELSE
                            BEGIN
                                          COMMIT TRANSACTION PostBlogEntryTransaction;
                            END;
END;
```

The stored procdure CanEdit looks like this:

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.CanEdit') IS NOT NULL DROP PROCEDURE dbo.CanEdit;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 01-10-2015
-- Description:             This stored procedure is used to
--              determine whether or not a user
--              has edit rights.
-- =============================================
CREATE PROCEDURE dbo.CanEdit
              @UserId int
AS
BEGIN

              IF EXISTS (
                            SELECT e.AccessName
                            FROM dbo.BlogUser AS a
                            JOIN dbo.BlogRoleAssignment AS b ON b.UserId = a.UserId
                            JOIN dbo.BlogRole AS c ON b.RoleId = c.RoleId
                            JOIN dbo.BlogRoleAccess AS d ON b.RoleId = d.RoleId
                            JOIN dbo.BlogAccess AS e ON d.AccessId = e.AccessId
                            WHERE a.UserId = @UserId
                              AND e.AccessName = 'Edit'
                )
              BEGIN
                            RETURN 0; -- Can Edit.
              END
              ELSE
              BEGIN
                            RETURN 1; -- No can do.
              END;
END;
```

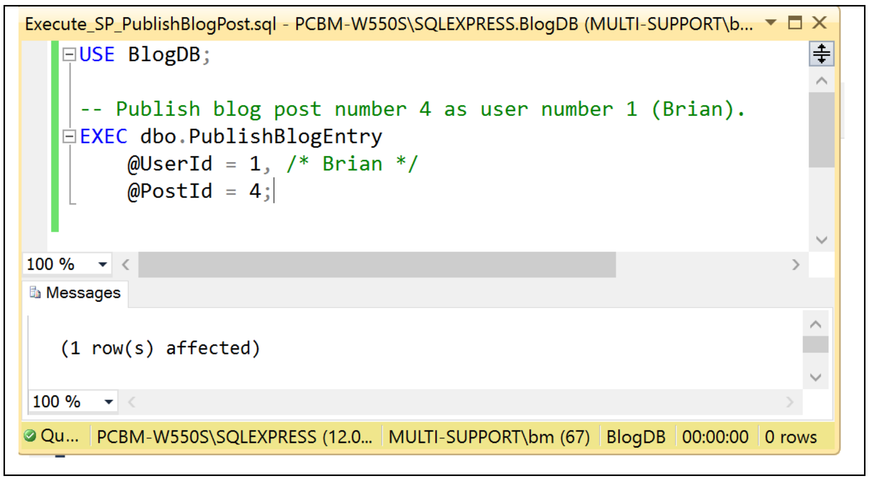
Having the following posts:



Results | Messages

	PostId	Headline	Content
1	1	Optimist	Optimistjolle, også kaldet optimisten eller optien, e...
2	2	RS Tera	The RS Tera is suitable for introducing newcomer...
3	3	J/70	The J/70 Speedster (22.75 feet) is J/Boats first ra...
4	4	RS Feva	The RS Feva is a two-person sailing dinghy desig...
5	5	Laser	The Laser is one of the most popular single-hande...
6	6	Silkeborg Sejlklub	Klubben er en hyggelig lille sejlklub, hvor vi nyder ...

We will now publish the post with PostId number 4.



Execute_SP_PublishBlogPost.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\b...

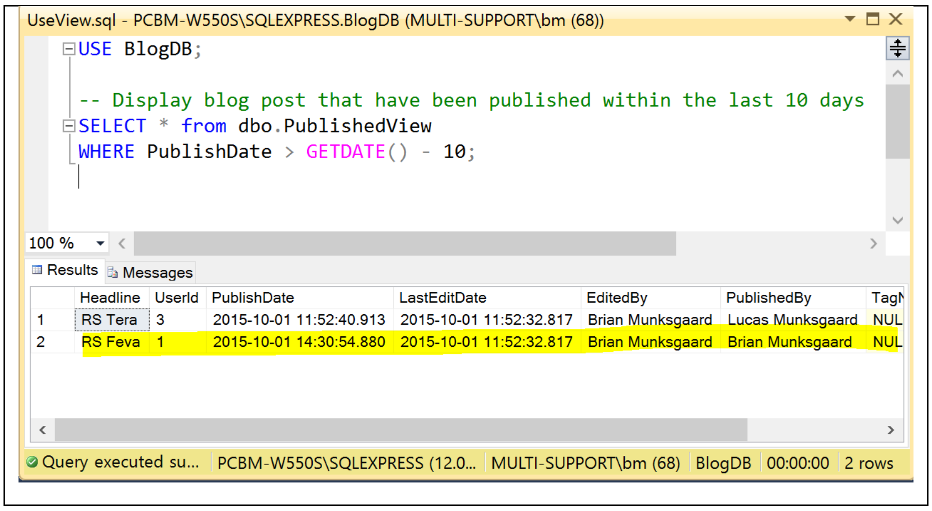
```
USE BlogDB;

-- Publish blog post number 4 as user number 1 (Brian).
EXEC dbo.PublishBlogEntry
    @UserId = 1, /* Brian */
    @PostId = 4;
```

Messages

```
(1 row(s) affected)
```

Qu... | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (67) | BlogDB | 00:00:00 | 0 rows

And using the view created earlier we can see that we have just published a post:



UseView.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (68))

```
USE BlogDB;

-- Display blog post that have been published within the last 10 days
SELECT * from dbo.PublishedView
WHERE PublishDate > GETDATE() - 10;
```

100 %

Results | Messages

	Headline	UserId	PublishDate	LastEditDate	EditedBy	PublishedBy	TagN
1	RS Tera	3	2015-10-01 11:52:40.913	2015-10-01 11:52:32.817	Brian Munksgaard	Lucas Munksgaard	NUL
2	RS Feva	1	2015-10-01 14:30:54.880	2015-10-01 11:52:32.817	Brian Munksgaard	Brian Munksgaard	NUL

Query executed su... | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (68) | BlogDB | 00:00:00 | 2 rows

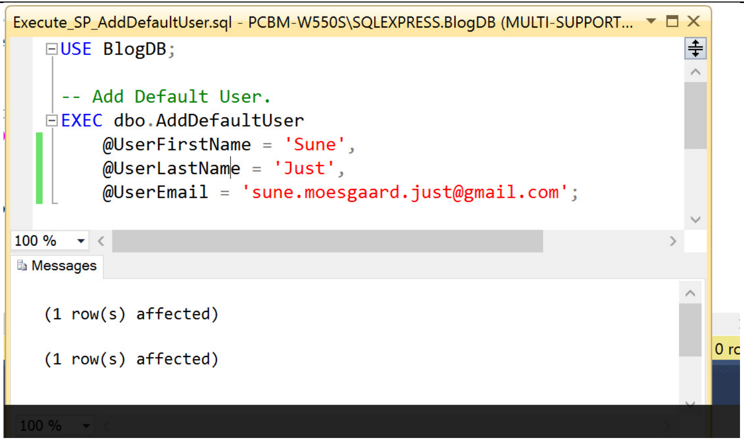
AddDefaultUser

Below, we can see the stored procedure for adding a user to the database:

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.AddDefaultUser') IS NOT NULL DROP PROCEDURE dbo.AddDefaultUser;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 04-10-2015
-- Description:             This stored procedure is used to
--              add a default user to the blog.
--              A default user is assigned the
--              commentator role.
-- =============================================
CREATE PROCEDURE dbo.AddDefaultUser
             @UserFirstName nchar(30),
             @UserLastName nchar(30),
             @UserEmail nchar(75)
AS
BEGIN

             BEGIN TRANSACTION AddDefaultUserTransaction;

             -- Add the user.
             INSERT INTO dbo.BlogUser (UserFirstName, UserLastName, UserEmail)
             VALUES (@UserFirstName, @UserLastName, @UserEmail);

             -- Retrieve user id.
             DECLARE @UserId AS int;
             SET @UserId = SCOPE_IDENTITY();

             -- Assign the role.
             INSERT INTO dbo.BlogRoleAssignment (UserId, RoleId)
             VALUES (@UserId, 3 /* Commentator */)

             IF @@ERROR <> 0
                          BEGIN
                                       ROLLBACK TRANSACTION AddDefaultUserTransaction;
                          END
             ELSE
```

```
                                    BEGIN
                                                  COMMIT TRANSACTION AddDefaultUserTransaction;
                                    END;
END;
```

A user is added by executing the stored procedure like:



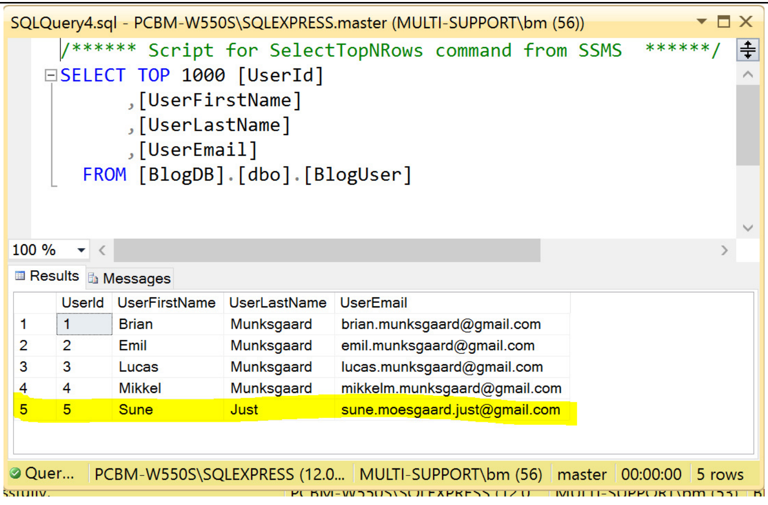
Execute_SP_AddDefaultUser.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT...

```
USE BlogDB;

-- Add Default User.
EXEC dbo.AddDefaultUser
    @UserFirstName = 'Sune',
    @UserLastName = 'Just',
    @UserEmail = 'sune.moesgaard.just@gmail.com';
```

Messages

```
(1 row(s) affected)

(1 row(s) affected)
```

And we can see that the user has been added:



SQLQuery4.sql - PCBM-W550S\SQLEXPRESS.master (MULTI-SUPPORT\bm (56))

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [UserId]
      ,[UserFirstName]
      ,[UserLastName]
      ,[UserEmail]
  FROM [BlogDB].[dbo].[BlogUser]
```

Results | Messages

	UserId	UserFirstName	UserLastName	UserEmail
1	1	Brian	Munksgaard	brian.munksgaard@gmail.com
2	2	Emil	Munksgaard	emil.munksgaard@gmail.com
3	3	Lucas	Munksgaard	lucas.munksgaard@gmail.com
4	4	Mikkel	Munksgaard	mikkelm.munksgaard@gmail.com
5	5	Sune	Just	sune.moesgaard.just@gmail.com

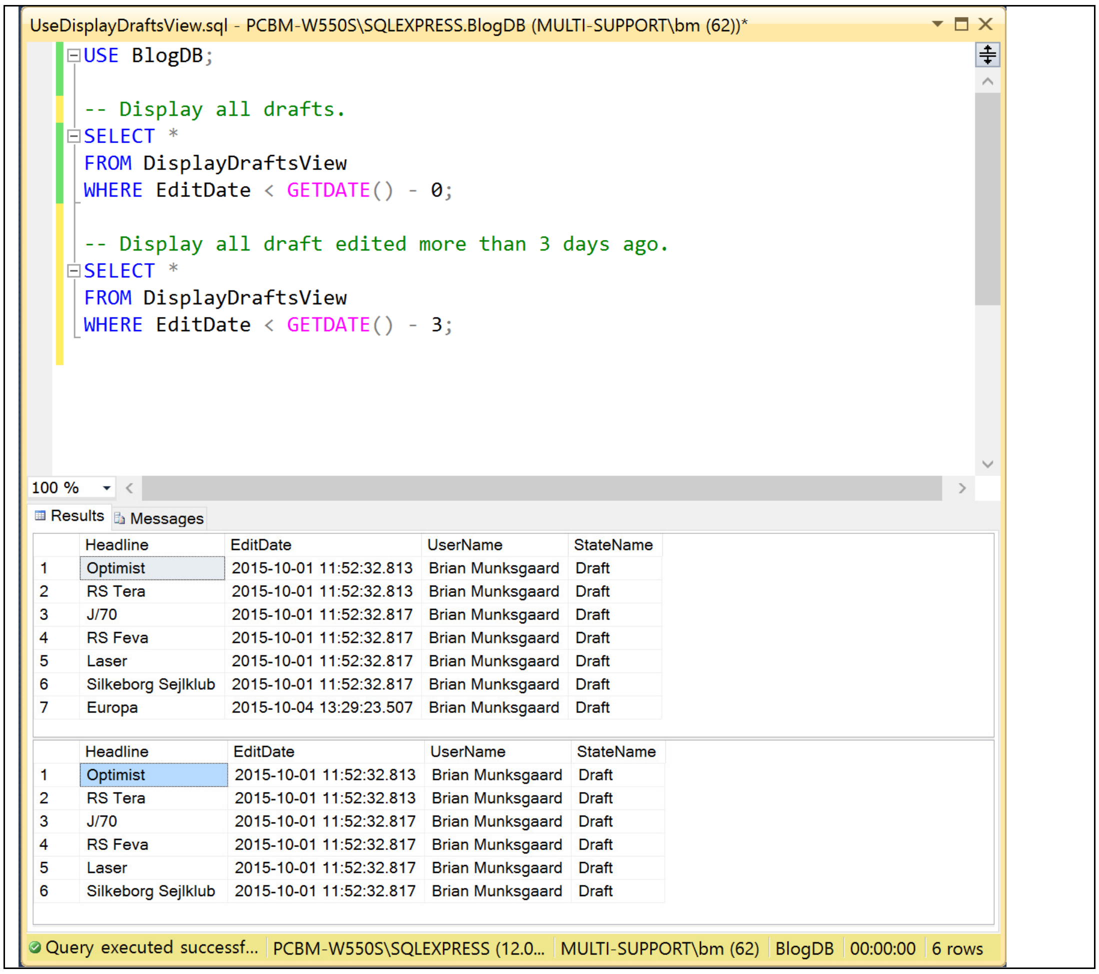
Quer... PCBM-W550S\SQLEXPRESS (12.0... MULTI-SUPPORT\bm (56) master 00:00:00 5 rows

DisplayDraftsView

This view is used to display drafts that have not been edited for a number of days. First the view is created:

```
USE BlogDB;

IF OBJECT_ID('dbo.DisplayDraftsView', 'V') IS NOT NULL DROP VIEW dbo.DisplayDraftsView;
GO

-- Display blog post that are still drafts.
CREATE VIEW DisplayDraftsView AS
SELECT a.Headline,
                 EditDate = b.LogDate,
                 UserName = CONCAT(RTRIM(c.UserFirstName), ' ', RTRIM(c.UserLastName)),
                 d.StateName
FROM dbo.BlogPost AS a
JOIN dbo.BlogLogEntry AS b ON a.PostId = b.PostId
JOIN dbo.BlogUser AS c ON b.UserId = c.UserId
JOIN dbo.BlogState AS d ON b.StateId = d.StateId
AND b.StateId = 1;
```

And then we use the view to display draft blog entries:



UseDisplayDraftsView.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (62))*

```
USE BlogDB;

-- Display all drafts.
SELECT *
FROM DisplayDraftsView
WHERE EditDate < GETDATE() - 0;

-- Display all draft edited more than 3 days ago.
SELECT *
FROM DisplayDraftsView
WHERE EditDate < GETDATE() - 3;
```

	Headline	EditDate	UserName	StateName
1	Optimist	2015-10-01 11:52:32.813	Brian Munksgaard	Draft
2	RS Tera	2015-10-01 11:52:32.813	Brian Munksgaard	Draft
3	J/70	2015-10-01 11:52:32.817	Brian Munksgaard	Draft
4	RS Feva	2015-10-01 11:52:32.817	Brian Munksgaard	Draft
5	Laser	2015-10-01 11:52:32.817	Brian Munksgaard	Draft
6	Silkeborg Sejlklub	2015-10-01 11:52:32.817	Brian Munksgaard	Draft
7	Europa	2015-10-04 13:29:23.507	Brian Munksgaard	Draft

	Headline	EditDate	UserName	StateName
1	Optimist	2015-10-01 11:52:32.813	Brian Munksgaard	Draft
2	RS Tera	2015-10-01 11:52:32.813	Brian Munksgaard	Draft
3	J/70	2015-10-01 11:52:32.817	Brian Munksgaard	Draft
4	RS Feva	2015-10-01 11:52:32.817	Brian Munksgaard	Draft
5	Laser	2015-10-01 11:52:32.817	Brian Munksgaard	Draft
6	Silkeborg Sejlklub	2015-10-01 11:52:32.817	Brian Munksgaard	Draft

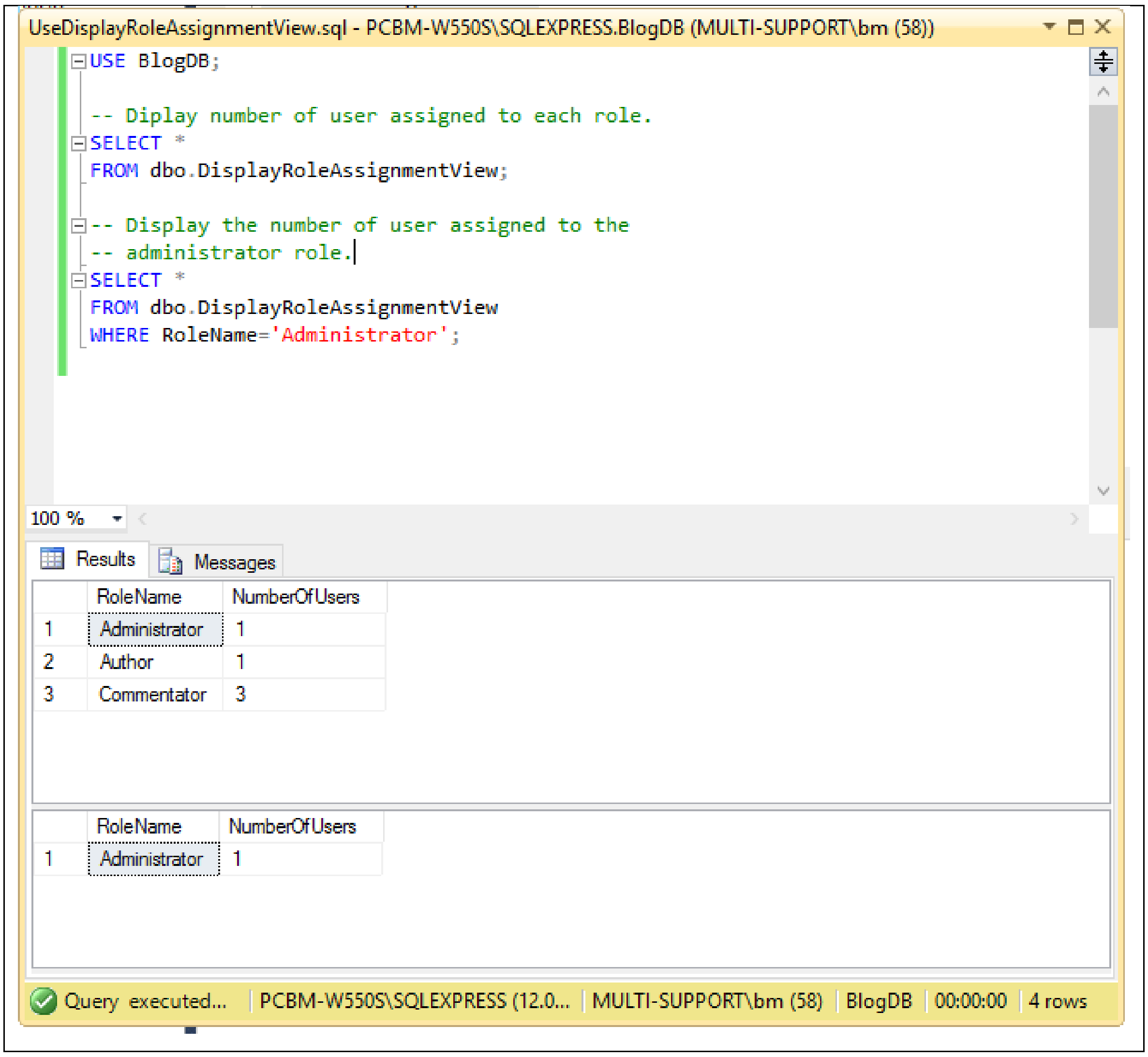
Query executed successf... | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (62) | BlogDB | 00:00:00 | 6 rows

DisplayRoleAssignmentsView

This view is used to diplay the number of users assigned to each view. The view is created with the following SQL statement:

```
USE BlogDB;

IF OBJECT_ID('dbo.DisplayRoleAssignmentView', 'V') IS NOT NULL DROP VIEW
dbo.DisplayRoleAssignmentView;
GO

-- Display the number of users assigned to each role.
CREATE VIEW dbo.DisplayRoleAssignmentView AS
SELECT a.RoleName, COUNT(b.RoleId) AS NumberOfUsers
FROM dbo.BlogRole AS a
JOIN dbo.BlogRoleAssignment AS b ON a.RoleId = b.RoleId
GROUP BY a.RoleName;
```

The view can then be used like this:



UseDisplayRoleAssignmentView.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\bm (58))

```
USE BlogDB;

-- Diplay number of user assigned to each role.
SELECT *
FROM dbo.DisplayRoleAssignmentView;

-- Display the number of user assigned to the
-- administrator role.
SELECT *
FROM dbo.DisplayRoleAssignmentView
WHERE RoleName='Administrator';
```

	RoleName	NumberOfUsers
1	Administrator	1
2	Author	1
3	Commentator	3

	RoleName	NumberOfUsers
1	Administrator	1

Query executed... | PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (58) | BlogDB | 00:00:00 | 4 rows

4 Transaction and Triggers

4.1 Create Transactions

As a rule of thumb, you can say that, transactions should be used whenever:

- We are doing multiple inserts or updates to the database.
- The inserts and updates are logically considered one unit of work that either succeeds or fails (The Atomicity part of the ACID principles).

Following those two rules, almost all the stored procedures (like AddBlogEntry or AddDefaultUser) in the Blog DB already contains transactions. Stored procedures in the blog DB only doing selects (like CanEdit and SearchBlogEntry) are not using transactions.

4.2 Create new Transactions

Since most stored procedures in the Blog DB already uses transaction I have create only one new stored procedure, EditBlogEntry, which is used to update the content of an existing blog post. The code for the stored procedure is shown below:

```
USE BlogDB;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF OBJECT_ID('dbo.EditBlogEntry') IS NOT NULL DROP PROCEDURE dbo.EditBlogEntry;
GO

-- =============================================
-- Author:                  Brian Munksgaard
-- Create date: 07-10-2015
-- Description:             This stored procedure is used to
--              update an existing blog entry.
--              When a blog post is edited it
--              changes state to a draft.
-- =============================================
CREATE PROCEDURE dbo.EditBlogEntry
              @PostId int,
              @UserId int,
              @Headline nchar(30),
              @Content nvarchar(MAX)
AS
BEGIN

              BEGIN TRANSACTION EditBlogEntryTransaction;

              -- Update Headline and Content.
              UPDATE dbo.BlogPost
              SET Headline = @Headline,
                            Content = @Content
              WHERE PostId = @PostId;

              -- Update the log.
              INSERT INTO dbo.BlogLogEntry (LogDate, PostId, ActionId, StateId, UserId)
              VALUES (
                            GETDATE(),
                            @PostId,
                            2, -- Action: Edit.
                            1, -- State: Draft.
                            @UserId
              );

              IF @@ERROR <> 0
                            BEGIN
```

```
                                                    ROLLBACK TRANSACTION EditBlogEntryTransaction;
                                      END
                         ELSE
                                      BEGIN
                                                    COMMIT TRANSACTION EditBlogEntryTransaction;
                                      END;
END;
```

I have not shown an example of use here, because the stored procedure is close tied to the next task.

4.3 Create new Triggers

This task is about creating a trigger that should be used to update a BlogLog. In my blog DB logging is already done in the stored procedures. Instead I have created a new table, BlogPostRev, to hold blog post revisions. Whenever a blog post is updated, preferably through the EditBlogEntry SP, a trigger is fired. The trigger writes the previous blog data to the revision table.

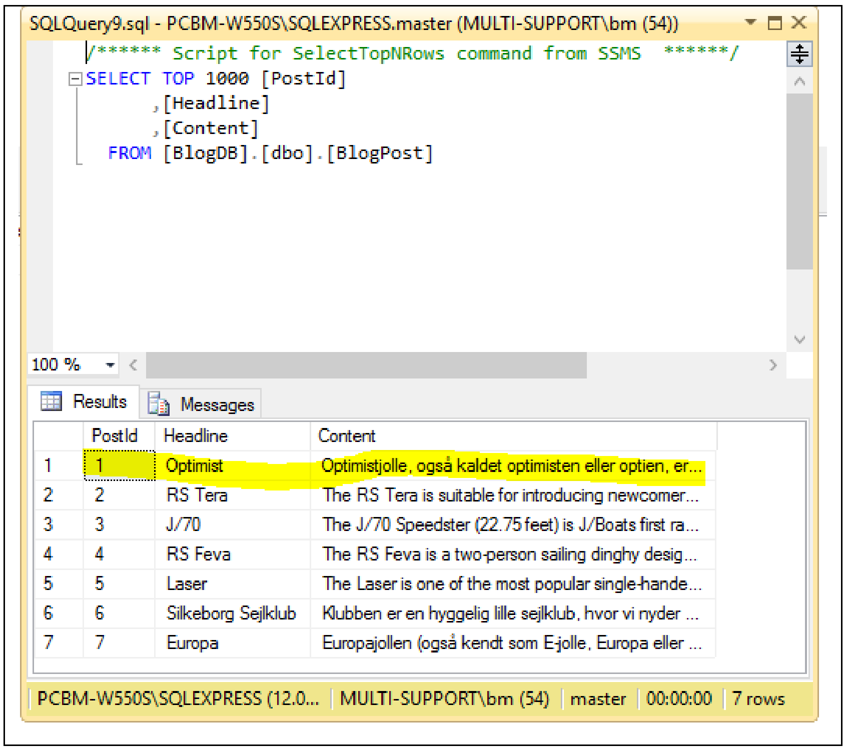
The BlogPostRev table looks like this:

```
USE BlogDB;
GO

-- Delete table if it already exists.
IF OBJECT_ID('dbo.BlogPostRev', 'U') IS NOT NULL DROP TABLE dbo.BlogPostRev;

-- Blog post revisions
CREATE TABLE dbo.BlogPostRev (
             RevisionId int IDENTITY(1,1) PRIMARY KEY,
             PostId int FOREIGN KEY REFERENCES BlogPost(PostId),
             RevisionNumber int NOT NULL,
             Headline nchar(30) NOT NULL,
             Content nvarchar(MAX) NOT NULL
) ON [PRIMARY];
```

The trigger, UpdateBlogPostRevTrigger, is created with the following script:

```
USE BlogDB;
GO

-- This trigger is used to update the BlogPostRev table
-- whenever a blog post is changed.
CREATE TRIGGER UpdateBlogPostRevTrigger ON dbo.BlogPost
AFTER UPDATE
AS
BEGIN

             DECLARE @PostId as int;
             SET @PostId = (SELECT PostId FROM deleted);

             DECLARE @NextRev as int;
             SELECT @NextRev = (SELECT MAX(RevisionNumber) FROM dbo.BlogPostRev AS a
WHERE a.PostId = @PostId) + 1;

             INSERT INTO dbo.BlogPostRev(PostId, RevisionNumber, Headline, Content)
             VALUES(@PostId, @NextRev, (SELECT Headline FROM deleted), (SELECT Content
FROM deleted));

END;
```

Now, let us change the content of a blog post. Currently we have the following posts:



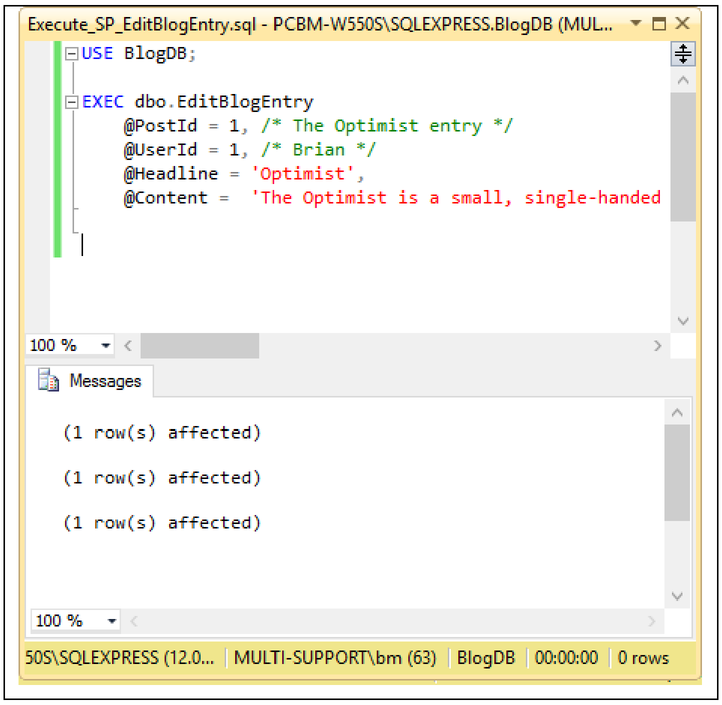
SQLQuery9.sql - PCBM-W550S\SQLEXPRESS.master (MULTI-SUPPORT\bm (54))

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [PostId]
      ,[Headline]
      ,[Content]
  FROM [BlogDB].[dbo].[BlogPost]
```

	PostId	Headline	Content
1	1	Optimist	Optimistjolle, også kaldet optimisten eller optien, er...
2	2	RS Tera	The RS Tera is suitable for introducing newcomer...
3	3	J/70	The J/70 Speedster (22.75 feet) is J/Boats first ra...
4	4	RS Feva	The RS Feva is a two-person sailing dinghy desig...
5	5	Laser	The Laser is one of the most popular single-hande...
6	6	Silkeborg Sejlklub	Klubben er en hyggelig lille sejlklub, hvor vi nyder ...
7	7	Europa	Europajollen (også kendt som E-jolle, Europa eller ...

PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (54) | master | 00:00:00 | 7 rows

Executing the EditBlogEntry SP like:



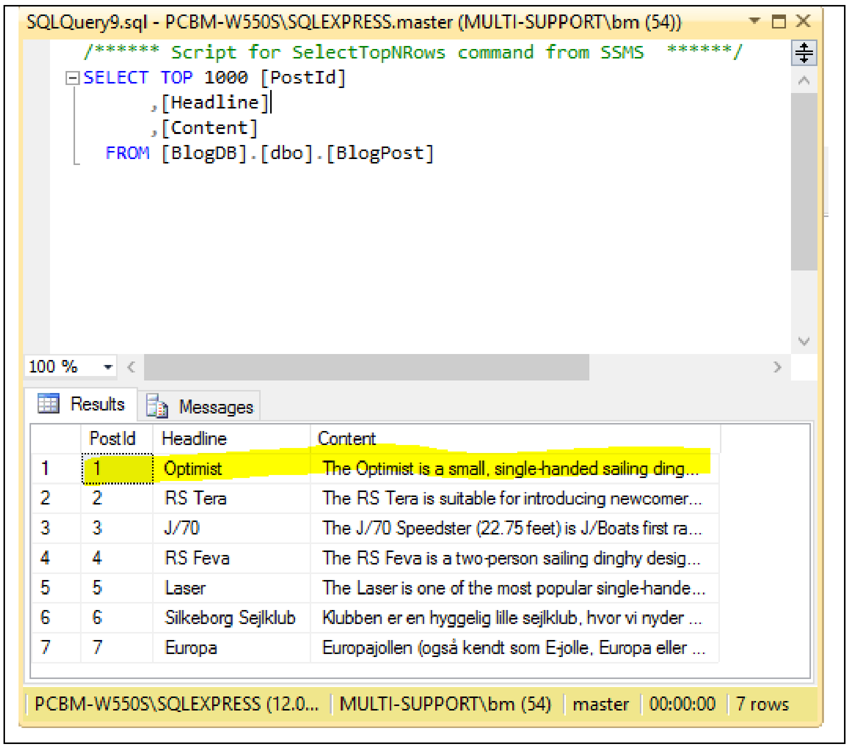
Execute_SP_EditBlogEntry.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MUL...

```
USE BlogDB;

EXEC dbo.EditBlogEntry
     @PostId = 1, /* The Optimist entry */
     @UserId = 1, /* Brian */
     @Headline = 'Optimist',
     @Content =  'The Optimist is a small, single-handed
```

```
(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)
```

50S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (63) | BlogDB | 00:00:00 | 0 rows

We can see that the blog content for PostId 1 is now in English:



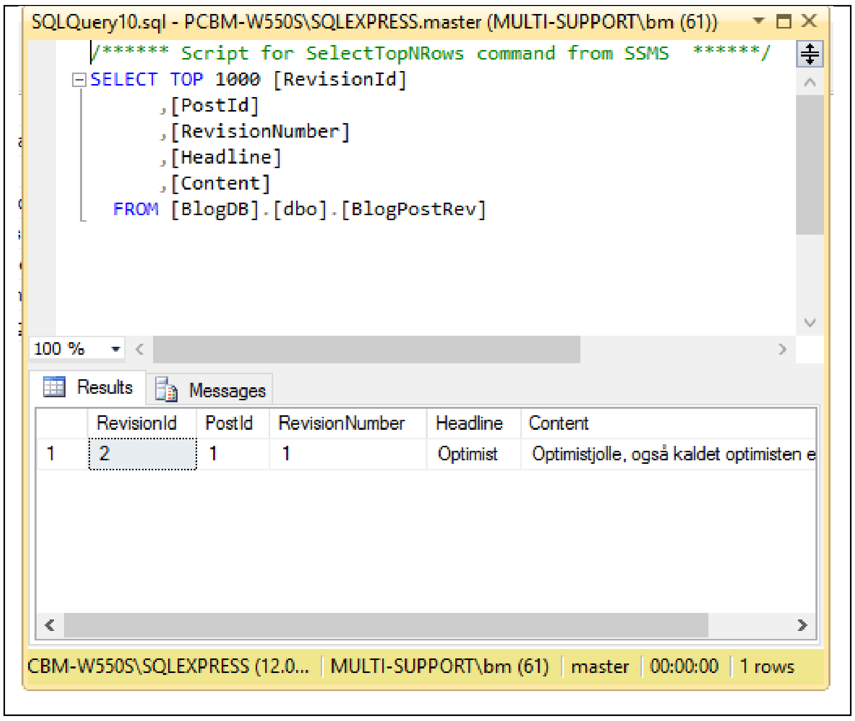
SQLQuery9.sql - PCBM-W550S\SQLEXPRESS.master (MULTI-SUPPORT\bm (54))

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [PostId]
      ,[Headline]
      ,[Content]
  FROM [BlogDB].[dbo].[BlogPost]
```

	PostId	Headline	Content
1	1	Optimist	The Optimist is a small, single-handed sailing ding...
2	2	RS Tera	The RS Tera is suitable for introducing newcomer...
3	3	J/70	The J/70 Speedster (22.75 feet) is J/Boats first ra...
4	4	RS Feva	The RS Feva is a two-person sailing dinghy desig...
5	5	Laser	The Laser is one of the most popular single-hande...
6	6	Silkeborg Sejlklub	Klubben er en hyggelig lille sejlklub, hvor vi nyder ...
7	7	Europa	Europajollen (også kendt som E-jolle, Europa eller ...

PCBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (54) | master | 00:00:00 | 7 rows

And looking at the revision table we see that the previous content of the blog post has been stored:



SQLQuery10.sql - PCBM-W550S\SQLEXPRESS.master (MULTI-SUPPORT\bm (61))

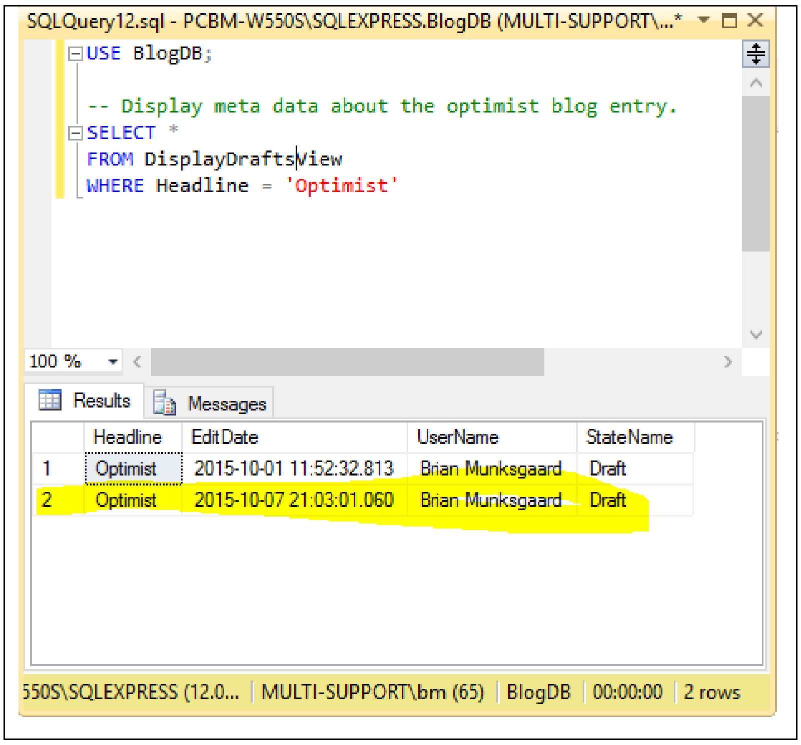
```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [RevisionId]
      ,[PostId]
      ,[RevisionNumber]
      ,[Headline]
      ,[Content]
  FROM [BlogDB].[dbo].[BlogPostRev]
```

	RevisionId	PostId	RevisionNumber	Headline	Content
1	2	1	1	Optimist	Optimistjolle, også kaldet optimisten e

CBM-W550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (61) | master | 00:00:00 | 1 rows

Notice the content of the Content attribute is in Danish.

And in the log we can see that we have just edited the blog entry:


SQLQuery12.sql - PCBM-W550S\SQLEXPRESS.BlogDB (MULTI-SUPPORT\...*

```
USE BlogDB;

-- Display meta data about the optimist blog entry.
SELECT *
FROM DisplayDraftsView
WHERE Headline = 'Optimist'
```

	Headline	EditDate	UserName	StateName
1	Optimist	2015-10-01 11:52:32.813	Brian Munksgaard	Draft
2	Optimist	2015-10-07 21:03:01.060	Brian Munksgaard	Draft

550S\SQLEXPRESS (12.0... | MULTI-SUPPORT\bm (65) | BlogDB | 00:00:00 | 2 rows