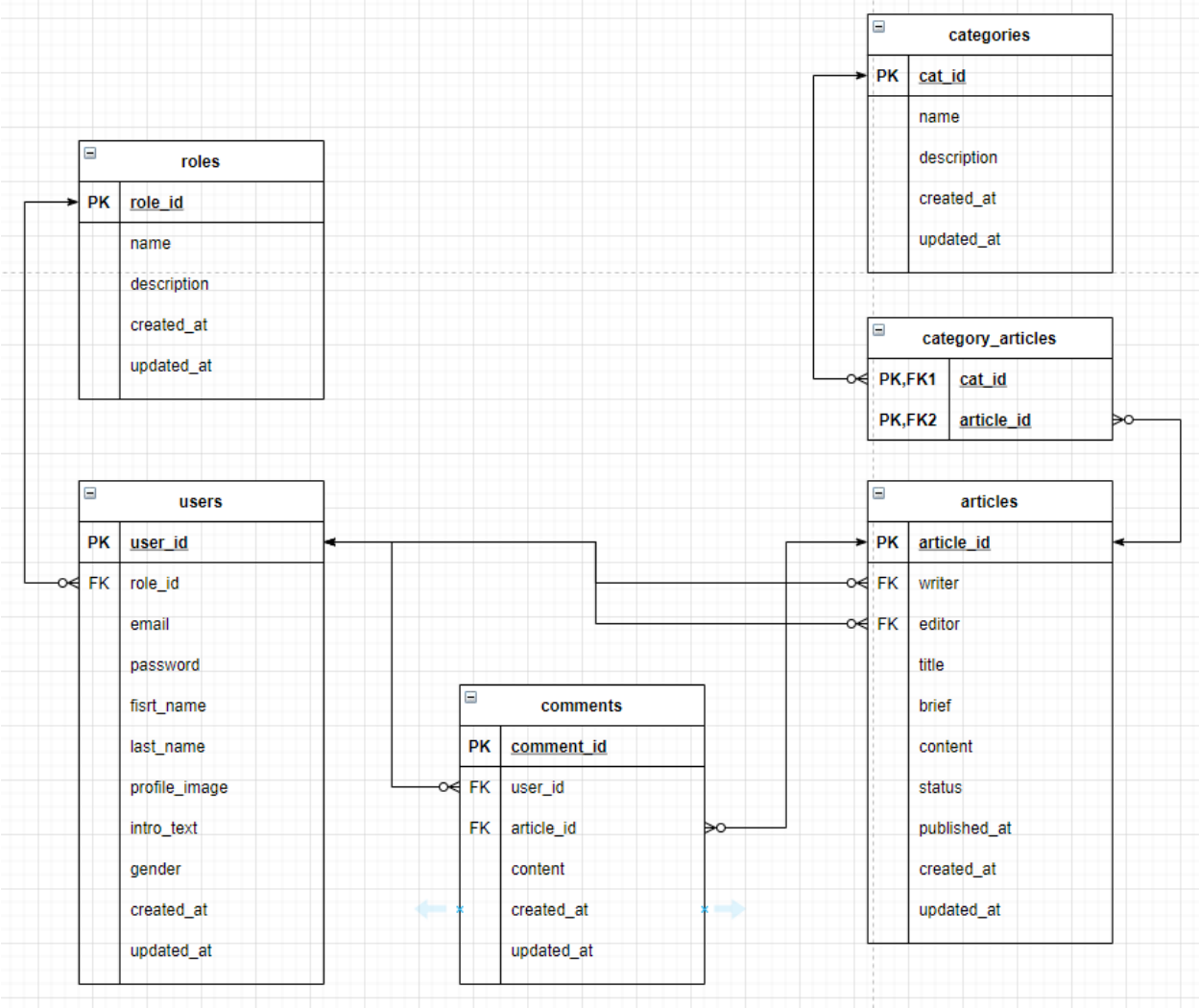
DBI202x_02-A_VN

Assignment 02

Phạm Minh Hùng - SE00234x

From assignment 01, we have:



Note: This asm pdf file comes with a sql file with the same name.

We start by creating new a database named: “newspaper”

```
CREATE DATABASE newspaper;
```

Change the current context to use “newspaper”

```
USE newspaper;
```

Now we can start creating tables.

I. Creating tables

1. “roles” table

Create table with below script:

```
CREATE TABLE roles
(
   role_id     VARCHAR(10) PRIMARY KEY,
   name        NVARCHAR(30),
   description NVARCHAR(200),
   created_at  DATETIME DEFAULT GETUTCDATE() NOT NULL,
   updated_at  DATETIME DEFAULT GETUTCDATE() NOT NULL
);
```

Explanation:

- ***role_id***: it does not need to contain UTF-16 characters, and I want it to be expressive, so the length 10 is more than enough. That’s why I chose VARCHAR(10).
- ***name***: it should support UTF-16, and length 30 is good enough => NVARCHAR(30).
- ***description***: UTF-16 and long enough => NVARCHAR(200).
- ***created_at*** and ***updated_at***: default to the time the row is recorded.

I want it to update the “updated_at” every time the role is changed, so I need to create a trigger for that:

```
CREATE TRIGGER tg_RoleUpdateTime
   ON roles
   AFTER UPDATE AS
BEGIN
   SET NOCOUNT ON
   DECLARE @Id VARCHAR(10);
   SELECT @Id = role_id FROM inserted;
   IF TRIGGER_NESTLEVEL(OBJECT_ID('dbo.tg_RoleUpdateTime')) > 1 RETURN
   UPDATE dbo.roles SET updated_at = GETUTCDATE() WHERE role_id = @Id;
END
```

Add data to Role table:

```
INSERT INTO roles(role_id, name, description)
VALUES ('sa', 'Super Admin', 'S/HE can do all the configurations'),
      ('admin', 'Admin', 'S/HE can manage all editors and all business
aspects'),
      ('editor', 'Editor', 'S/HE can manage all writers and their articles'),
      ('writer', 'Writer', 'S/HE can manage all of his/her own articles'),
      ('viewer', 'Viewer', 'S/HE can reads articles'),
      ('viewer1', 'Viewer1', 'S/HE can reads articles'), -- duplicate
      ('viewer2', 'Viewer2', 'S/HE can reads articles'), -- duplicate
      ('viewer3', 'Viewer3', 'S/HE can reads articles'), -- duplicate
      ('viewer4', 'Viewer4', 'S/HE can reads articles'), -- duplicate
      ('viewer5', 'Viewer5', 'S/HE can reads articles'); -- duplicate
```

Note: for duplicate values, I just make it to achieve 10 records in each table (one criteria in requirement)

2. **“users” table**

Create table with below script:

```
CREATE TABLE users
(
   user_id       INT IDENTITY (1,1) PRIMARY KEY,
   role_id       VARCHAR(10)                    NOT NULL,
   email         NVARCHAR(320) UNIQUE           NOT NULL,
   password      NVARCHAR(64),
   first_name    NVARCHAR(30),
   last_name     NVARCHAR(30),
   profile_image NVARCHAR(100),
   intro_text    NVARCHAR(200),
   gender        CHAR(1),
   created_at    DATETIME DEFAULT GETUTCDATE() NOT NULL,
   updated_at    DATETIME DEFAULT GETUTCDATE() NOT NULL,

   CONSTRAINT FK_USER_ROLE FOREIGN KEY (role_id) REFERENCES roles (role_id)
);
```

Explanation:

- email: "local part": 64 + "@": 1 + "domain part": 255 which sums to 320
  Ref: tools.ietf.org/html/rfc3696 => NVARCHAR(320)
- password: since we can store it as plain string or hashed string, the hashed string will be 64 in length => NVARCHAR(64)
- gender: for now, we just store “m” for male, and “f” for female => CHAR(1).
- FK_USER_ROLE constraint to reference the “roles” table.

I will create the same trigger to update the updated_at:

```
CREATE TRIGGER tg_UserUpdateTime
   ON users
   AFTER UPDATE AS
BEGIN
   SET NOCOUNT ON
   DECLARE @UserId INT;
   SELECT @UserId = user_id FROM inserted;
   IF TRIGGER_NESTLEVEL(OBJECT_ID('dbo.tg_UserUpdateTime')) > 1 RETURN
   UPDATE users SET updated_at = GETUTCDATE() WHERE user_id = @UserId;
END
```

Insert data for it:

```
SET IDENTITY_INSERT users ON;
INSERT INTO users(user_id, role_id, email, first_name, last_name, gender)
VALUES (1, 'sa', 'hungpmse00234x@funix.edu.vn', 'Hung', 'Pham', 'm'),
      (2, 'admin', 'jen@gmail.com', 'Jennifer', 'Pham', 'f'),
      (3, 'editor', 'james@gmail.com', 'James', 'Potter', 'm'),
      (4, 'editor', 'evelyn@gmail.com', 'Evelyn', 'Truong', 'f'),
      (5, 'writer', 'jackson@gmail.com', 'Jackson', 'Stewart', 'm'),
      (6, 'writer', 'ellie@gmail.com', 'Ellie', 'Lopez', 'f'),
      (7, 'writer', 'dorian@gmail.com', 'Dorian', 'Bell', 'm'),
      (8, 'viewer', 'tasha@gmail.com', 'Tashia', 'Thompson', 'f'),
      (9, 'viewer', 'norris@gmail.com', 'Norris', 'Howard', 'm'),
      (10, 'viewer', 'vince@gmail.com', 'Vince', 'Blake', 'm'),
      (11, 'viewer', 'ned@gmail.com', 'Ned', 'Alexander', 'm'),
      (12, 'viewer', 'jessica@gmail.com', 'Jessica', 'Valdez', 'f');
SET IDENTITY_INSERT users OFF;
```

3. “articles” table

Create table:

```
CREATE TABLE articles
(
   article_id   INT IDENTITY (1,1) PRIMARY KEY,
   writer       INT                            NOT NULL,
   editor       INT,
   title        NVARCHAR(200),
   brief        NVARCHAR(500),
   content      NVARCHAR(MAX),
   status       VARCHAR(10) DEFAULT 'draft',
   published_at DATETIME,
   created_at   DATETIME    DEFAULT GETUTCDATE() NOT NULL,
   updated_at   DATETIME    DEFAULT GETUTCDATE() NOT NULL,
```

```
   CONSTRAINT FK_ARTICLE_WRITER FOREIGN KEY (writer) REFERENCES users
(user_id),
   CONSTRAINT FK_ARTICLE_EDITOR FOREIGN KEY (editor) REFERENCES users
(user_id)
);
```

Explanation:

- **status**: as in assignment 1, the status will have: draft, pending, denied, published => VARCHAR(10) is enough

Since we want to search latest news, we add nonclustered index to “published_at”:

```
CREATE NONCLUSTERED INDEX idx_ArticlePublishedAt
   ON articles (published_at DESC);
```

And, add trigger for updating:

```
CREATE TRIGGER tg_ArticleUpdateTime
   ON articles
   AFTER UPDATE AS
BEGIN
   SET NOCOUNT ON
   DECLARE @Id INT;
   SELECT @Id = article_id FROM inserted;
   IF TRIGGER_NESTLEVEL(OBJECT_ID('dbo.tg_ArticleUpdateTime')) > 1 RETURN
   UPDATE articles SET updated_at = GETUTCDATE() WHERE article_id = @Id;
END
```

And data for “articles” table:

```
SET IDENTITY_INSERT articles ON;
INSERT INTO articles(article_id, writer, editor, title, status, published_at)
VALUES (1, 5, NULL, 'article 1', 'draft', NULL),
      (2, 6, NULL, 'article 2', 'draft', NULL),
      (3, 7, 3, 'article 3', 'draft', DATEFROMPARTS(2021, 9, 1)),
      (4, 5, 4, 'article 4', 'published', DATEFROMPARTS(2021, 8, 1)),
      (5, 6, 3, 'article 5', 'published', DATEFROMPARTS(2021, 6, 1)),
      (6, 5, 3, 'article 6', 'pending', DATEFROMPARTS(2021, 6, 1)),
      (7, 7, 4, 'article 7', 'pending', DATEFROMPARTS(2021, 6, 1)),
      (8, 7, 4, 'article 8', 'denied', DATEFROMPARTS(2021, 6, 1)),
      (9, 6, 4, 'article 9', 'published', DATEFROMPARTS(2021, 6, 1)),
      (10, 5, 3, 'article 10', 'published', DATEFROMPARTS(2021, 6, 1));
SET IDENTITY_INSERT articles OFF;
```

4. **“comments” table**

Create table:

```
CREATE TABLE comments
(
   comment_id INT IDENTITY (1,1) PRIMARY KEY,
   user_id    INT                          NOT NULL,
   article_id INT                          NOT NULL,
   content    NVARCHAR(500)                NOT NULL,
   created_at DATETIME DEFAULT GETUTCDATE() NOT NULL,
   updated_at DATETIME DEFAULT GETUTCDATE() NOT NULL,

   CONSTRAINT FK_COMMENT_USER FOREIGN KEY (user_id) REFERENCES users
(user_id),
   CONSTRAINT FK_COMMENT_ARTICLE FOREIGN KEY (article_id) REFERENCES users
(user_id),
);
```

We also want to create an index for the “comments” table, we want to search for all comments in one article in DESC order for “created_at”:

```
CREATE NONCLUSTERED INDEX idx_CommentArticle
   ON comments (article_id, created_at DESC);
```

Trigger for updating updated_at:

```
CREATE TRIGGER tg_CommentUpdateTime
   ON comments
   AFTER UPDATE AS
BEGIN
   SET NOCOUNT ON
   DECLARE @Id INT;
   SELECT @Id = comment_id FROM inserted;
   IF TRIGGER_NESTLEVEL(OBJECT_ID('dbo.tg_CommentUpdateTime')) > 1 RETURN
   UPDATE comments SET updated_at = GETUTCDATE() WHERE comment_id = @Id;
END
```

Insert data to “comments” table:

```
SET IDENTITY_INSERT comments ON;
INSERT INTO comments(comment_id, user_id, article_id, content, created_at)
VALUES (1, 8, 3, 'some comment', DATEFROMPARTS(2021, 10, 01)),
       (2, 11, 3, 'some comment', DATEFROMPARTS(2021, 10, 02)),
       (3, 12, 4, 'some comment', DATEFROMPARTS(2021, 10, 03)),
       (4, 9, 3, 'some comment', DATEFROMPARTS(2021, 10, 04)),
       (5, 10, 4, 'some comment', DATEFROMPARTS(2021, 10, 05)),
       (6, 8, 3, 'some comment', DATEFROMPARTS(2021, 10, 06)),
```

```
       (7, 12, 5, 'some comment', DATEFROMPARTS(2021, 10, 07)),
       (8, 9, 5, 'some comment', DATEFROMPARTS(2021, 10, 08)),
       (9, 12, 3, 'some comment', DATEFROMPARTS(2021, 10, 09)),
       (10, 8, 3, 'some comment', DATEFROMPARTS(2021, 10, 10)),
       (11, 10, 4, 'some comment', DATEFROMPARTS(2021, 10, 11)),
       (12, 9, 3, 'some comment', DATEFROMPARTS(2021, 10, 12)),
       (13, 11, 5, 'some comment', DATEFROMPARTS(2021, 10, 13)),
       (14, 10, 3, 'some comment', DATEFROMPARTS(2021, 10, 14)),
       (15, 8, 4, 'some comment', DATEFROMPARTS(2021, 10, 15)),
       (16, 10, 3, 'some comment', DATEFROMPARTS(2021, 10, 16));
SET IDENTITY_INSERT comments OFF;
```

5. **“categories” table**

Create table:

```
CREATE TABLE categories
(
   cat_id      INT IDENTITY (1,1) PRIMARY KEY,
   name        NVARCHAR(20),
   description NVARCHAR(100),
   created_at  DATETIME DEFAULT GETUTCDATE() NOT NULL,
   updated_at  DATETIME DEFAULT GETUTCDATE() NOT NULL
);
```

Trigger for updating updated_at prop:

```
CREATE TRIGGER tg_CategoryUpdateTime
   ON categories
   AFTER UPDATE AS
BEGIN
   SET NOCOUNT ON
   DECLARE @Id INT;
   SELECT @Id = cat_id FROM inserted;
   IF TRIGGER_NESTLEVEL(OBJECT_ID('dbo.tg_CategoryUpdateTime')) > 1 RETURN
   UPDATE categories SET updated_at = GETUTCDATE() WHERE cat_id = @Id;
END
```

Insert data:

```
SET IDENTITY_INSERT categories ON;
INSERT INTO categories(cat_id, name, description)
VALUES (1, 'POV', 'Point Of View'),
       (2, 'World', 'World news'),
       (3, 'Business', 'Business news'),
       (4, 'Health', 'Health news'),
       (5, 'Science', 'Science news'),
       (6, 'Education', 'Education news'),
```

```
      (7, 'Travel', 'Travel news'),
      (8, 'Digital', 'Digital news'),
      (9, 'Comedy', 'Comedy news'),
      (10, 'Entertainment', 'Entertainment news');
SET IDENTITY_INSERT categories OFF;
```

6. “category_articles” table

Create table:

```
CREATE TABLE category_articles
(
   cat_id     INT NOT NULL,
   article_id INT NOT NULL,

   CONSTRAINT PK_CategoryArticle PRIMARY KEY (cat_id, article_id),
   CONSTRAINT FK_CategoryArticle_Category FOREIGN KEY (cat_id) REFERENCES
categories (cat_id),
   CONSTRAINT FK_CategoryArticle_Article FOREIGN KEY (article_id) REFERENCES
articles (article_id),
);
```

Add index so we can search for articles in one category:

```
CREATE NONCLUSTERED INDEX idx_CategoryArticle_Category
   ON category_articles (cat_id);
```

Insert data into table:

```
INSERT INTO category_articles(cat_id, article_id)
VALUES (2, 1),
      (2, 2),
      (3, 3),
      (3, 4),
      (3, 5),
      (4, 6),
      (4, 7),
      (4, 8),
      (5, 9),
      (5, 10);
```

II. Requirements' queries

This will come with Vietnamese requirements:

- Truy vấn dữ liệu trên một bảng

```
SELECT * FROM users;
```

- Truy vấn có sử dụng Order by

```
SELECT * FROM comments ORDER BY created_at DESC;
```

- Truy vấn dữ liệu từ nhiều bảng sử dụng Inner join

```
SELECT *
FROM users U
        INNER JOIN comments C
                   ON U.user_id = C.user_id
WHERE U.user_id = 8;
```

- Truy vấn thống kê sử dụng Group by và Having

```
SELECT U.user_id, count(*) AS comment_num
FROM users U
        INNER JOIN comments C
                   ON U.user_id = C.user_id
GROUP BY U.user_id
HAVING count(*) >= 3;
```

- Truy vấn sử dụng truy vấn con.

```
SELECT U.user_id,
      ISNULL((SELECT count(*) FROM comments C WHERE U.user_id = C.user_id
GROUP BY C.user_id), 0) as comment_num
FROM users U;
```

- Truy vấn sử dụng toán tử Like và các so sánh xâu ký tự.

```
SELECT *
FROM roles
WHERE name LIKE '%admin%';
```

- Truy vấn liên quan tới điều kiện về thời gian

```
SELECT *
FROM articles
WHERE published_at >= DATEFROMPARTS(2021, 8, 1);
```

- Truy vấn sử dụng Self join, Outer join.

```
--- Self Join: ghép cặp cùng role
SELECT A.role_id, A.first_name, B.first_name
FROM users A,
    users B
WHERE A.user_id < B.user_id
 AND A.role_id = B.role_id
ORDER BY A.role_id;
GO

-- Outer Join
SELECT U.user_id, U.first_name, C.article_id, C.content
FROM users U
        LEFT JOIN comments C ON U.user_id = C.user_id;
```

- Truy vấn sử dụng With.

```
WITH tblWriterArticleCount AS (
   SELECT U.user_id,
          U.first_name,
          U.last_name,
          (SELECT count(*) FROM articles A WHERE U.user_id = A.writer GROUP BY
A.writer) as article_count
   FROM users U
   WHERE U.role_id = 'writer'
)

SELECT TOP (2) user_id, first_name, last_name, article_count
FROM tblWriterArticleCount
ORDER BY article_count DESC
```

- Truy vấn sử dụng function (hàm) đã viết trong bước trước.

```
CREATE FUNCTION GetWriterPublishedArticleCountTable()
   RETURNS TABLE AS RETURN
           (
               SELECT writer, COUNT(*) AS published_article_count
               FROM articles
               WHERE status = 'published'
               GROUP BY writer
           );
GO

CREATE FUNCTION GetWriterwWithMostPublishedArticles()
   RETURNS INT AS
BEGIN
```

```
    DECLARE @id INT;

    SELECT @id = writer
    FROM GetWriterPublishedArticleCountTable()
    WHERE published_article_count = (
        SELECT MAX(published_article_count) FROM
GetWriterPublishedArticleCountTable()
    )

    RETURN @id
END
```