**Introduction to Databases**

**Programming Project**

**Phase 2**

# Project Name: Post ITT

# Group

MM

# Contributors

Helen O’Brien

Katie Griffiths

Anthony Bruhl

Aoife Sayers

# Date of Submission

Phase 2: Thursday 10th November 30%

* SQL script corresponding to relational schema
* Implement database in parallel - Amazon Web Services, SQL server
* Populate tables with data
* Triggers - 3 triggers each
* Identify need for & implement views, indexes & sequences
* Show triggers working
* Group & individual reports

Table of Contents

[Project Name: Post ITT 1](#_Toc467001006)

[Group 1](#_Toc467001007)

[MM 1](#_Toc467001008)

[Contributors 1](#_Toc467001009)

[Date of Submission 1](#_Toc467001010)

[Introduction 3](#_Toc467001011)

[Database Schema Adjustments 4](#_Toc467001012)

[Drop Table & Sequences 5](#_Toc467001013)

[Sequences 6](#_Toc467001014)

[Need for sequences 8](#_Toc467001015)

[Create table statements ordered properly & domain constraints were added 9](#_Toc467001016)

[Populated the script with data 12](#_Toc467001017)

[Implementing the database in parallel 17](#_Toc467001018)

[Implementing the Business Rules 19](#_Toc467001019)

[Business Rules Altered 20](#_Toc467001020)

[Triggers 21](#_Toc467001021)

[Procedures 21](#_Toc467001022)

[Views 22](#_Toc467001023)

[Index 23](#_Toc467001024)

[Procedures 24](#_Toc467001025)

[Triggers 48](#_Toc467001026)

[Views 61](#_Toc467001027)

[Index’s 69](#_Toc467001028)

[References 71](#_Toc467001029)

# Introduction

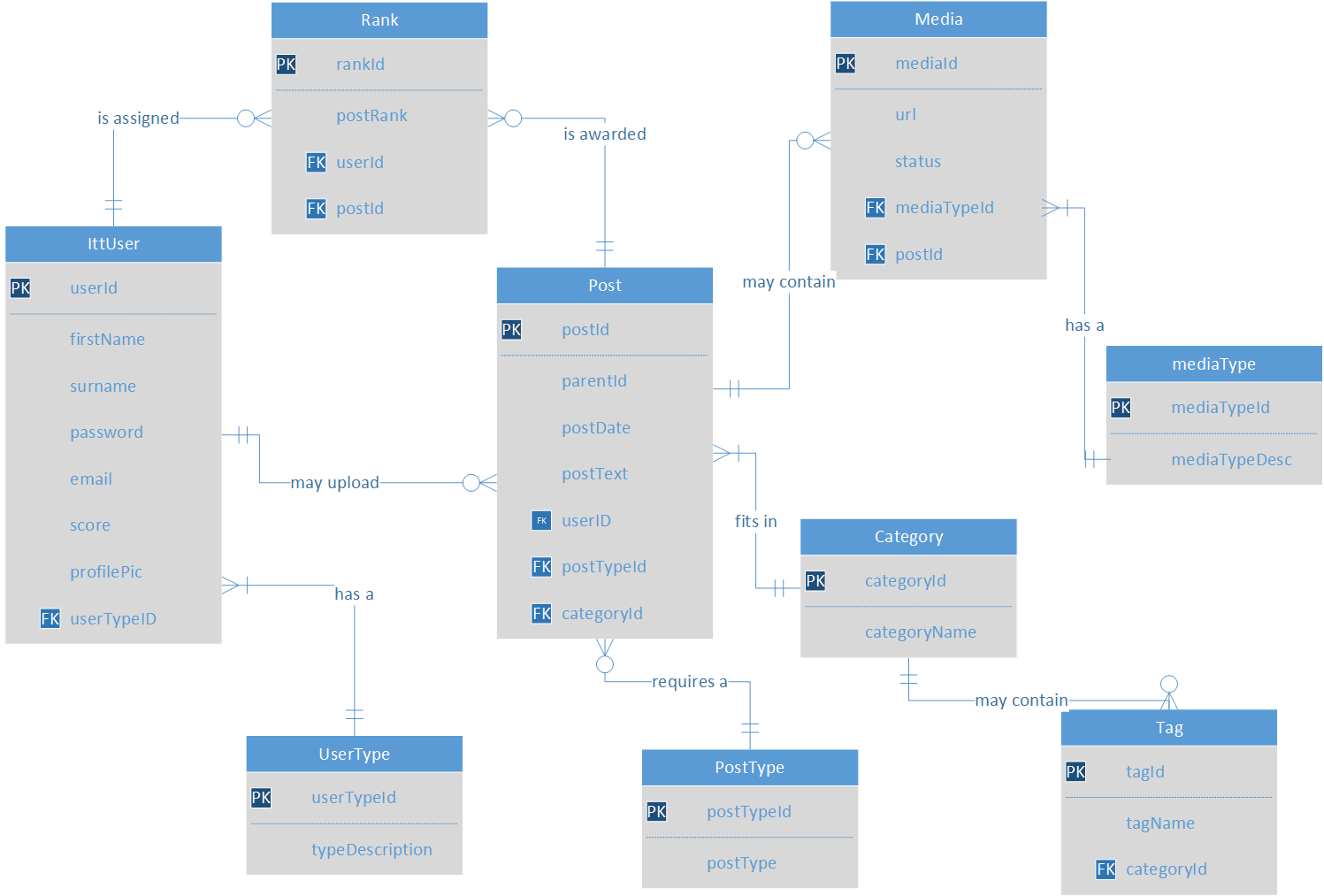
For this section of the project, we were required to produce SQL script corresponding to our relational schema, Triggers, Procedures, Sequences, Views and Indexes as well as all of our tables and alterations to the tables. We all worked as a team to put thought into how we would separate each part of the project equally, we found this hard at the beginning and saw that we needed to change our tactics quickly. With group meetings during college, we realised we each needed to choose a table and work with the procedures and triggers that were needed for them to work. We found that this was a better way in which to work and gave us all a chance to work using triggers, procedures, views and indexes.

We were also able to work from home by implementing the database in parallel on an Oracle instance on Amazon Web Services.

There is one script that creates the tables and sequences and another script with populates the tables with dummy information.

A later alteration to the project was a thought of using a Blob and Clob procedure for handling the users profile picture and for the handling of any document files that maybe loaded into the database. As a group, we thought to leave the screen shots of our earlier tasks to show the progression of working through this project using new skills and techniques that we learned.

# Database Schema Adjustments

****

Changes

* User Table changed to ITTUser - User is a reserved word in oracle
* Post Table - added a userID foreign key constraint & a mediaId foreign key constraint
* Media Table - added a status field to check p','a','f','d', -- pending, approved, flagged, disabled

# Drop Table & Sequences

DROP TABLE Media;

DROP TABLE MediaType;

DROP TABLE Rank;

DROP TABLE Post;

DROP TABLE Tag;

DROP TABLE Category;

DROP TABLE PostType;

DROP TABLE IttUser;

DROP TABLE UserType;

DROP SEQUENCE Media\_seq;

DROP SEQUENCE MediaType\_seq;

DROP SEQUENCE Rank\_seq;

DROP SEQUENCE Post\_seq;

DROP SEQUENCE Tag\_seq;

DROP SEQUENCE Category\_seq;

DROP SEQUENCE PostType\_seq;

DROP SEQUENCE IttUser\_seq;

DROP SEQUENCE UserType\_seq;

# Sequences

Using these instead of auto-increment for support with Oracle

CACHE clause keeps 20 sequence values in cache memory so they can be accessed faster

CREATE SEQUENCE Media\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE UserType\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE IttUser\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE PostType\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE Category\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE MediaType\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE Rank\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE Post\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

CREATE SEQUENCE Tag\_seq

MINVALUE 1

MAXVALUE 99999

START WITH 1

INCREMENT BY 1

CACHE 20;

# Need for sequences

* Oracle does not have support for the AUTO\_INCREMENT (MySQL Syntax)
  + Auto-increment allows a unique integer to be generated when a new record is inserted into a table
  + Auto\_increment used in MySQL, SQLServer & Access
* Oracle uses sequences
  + Generate unique integers on primary key values
  + Sequence number is incremented each time
  + CREATE SEQUENCE supplier\_seq  
     MINVALUE 1  
     START WITH 1  
     INCREMENT BY 1  
     CACHE 20;
  + INSERT INTO suppliers(supplier\_id, supplier\_name)  
    VALUES(supplier\_seq.NEXTVAL, 'Kraft Foods');
* Use sequences to generate a unique integer on a primary key entity
* No chance of clashing primary keys on inserting into the database while using sequences - maintains Entity integrity

# Create table statements ordered properly & domain constraints were added

CREATE TABLE UserType (

userTypeId      NUMBER(5),

typeDescription VARCHAR2(1) NOT NULL UNIQUE CONSTRAINT typeDesc\_chk

CHECK(typeDescription IN('A','U')),

CONSTRAINT pk\_uType\_ID PRIMARY KEY (userTypeId));

CREATE TABLE IttUser (

   userId     NUMBER(5),

   firstName  VARCHAR2(20) NOT NULL CONSTRAINT fname\_chk

CHECK(LENGTH(firstName)>0 AND LENGTH(firstName)<=20),

   surname    VARCHAR2(20) NOT NULL CONSTRAINT sname\_chk

CHECK(LENGTH(surname)>0 AND LENGTH(surname)<=20),

   password   VARCHAR2(30) NOT NULL,

   email      VARCHAR2(30) NOT NULL UNIQUE ,

   score      NUMBER(5) CHECK(score>0 AND score<99999),

   profilePic VARCHAR2(50),

   userTypeId   NUMBER(5),

   CONSTRAINT pk\_firstName PRIMARY KEY (userId),

   CONSTRAINT fk\_userTypeId\_UD FOREIGN KEY (userTypeId) REFERENCES

UserType(userTypeId));

CREATE TABLE PostType (

   postTypeId NUMBER(5) CONSTRAINT postType\_chk CHECK(postTypeId >0

AND postTypeId<=99999),

  postType   VARCHAR2(20) NOT NULL CONSTRAINT postTypetbl\_chk

CHECK(postType IN('Informative','Reply','Question','Answer')),

   CONSTRAINT pk\_postTypeId PRIMARY KEY (postTypeId));

CREATE TABLE Category (

   categoryId   NUMBER(5) CONSTRAINT catid\_chk CHECK(categoryId >0 AND

categoryId<=99999),

   categoryName VARCHAR2(20) NOT NULL UNIQUE CONSTRAINT catName\_chk

CHECK(categoryName IN('Computing','Business','Creative

Media','Engineering','Nursing','Social Science')),

   CONSTRAINT pk\_categoryId PRIMARY KEY (categoryId));

CREATE TABLE Tag (

   tagId   NUMBER(5) CONSTRAINT tagid\_chk CHECK(tagId >0 AND

tagId<=99999),

   tagName VARCHAR2(20) NOT NULL UNIQUE,

   categoryId   NUMBER(5) NOT NULL CONSTRAINT catidtag\_chk

CHECK(categoryId >0 AND categoryId<=99999),

   CONSTRAINT pk\_tagId PRIMARY KEY (tagId),

   CONSTRAINT fk\_categoryId\_T FOREIGN KEY (categoryId) REFERENCES

Category(categoryId));

CREATE TABLE Post (

   postId     NUMBER(5)  CONSTRAINT postid\_chk CHECK(postId >0 AND

postId<=99999),

   parentId   NUMBER(5) CONSTRAINT parentId\_chk CHECK(parentId >0 AND

parentId<=99999),

   postdate   DATE DEFAULT SYSDATE,

   postText   VARCHAR2(1000) NOT NULL,

   userId     NUMBER(5),

   postTypeId NUMBER(5) CONSTRAINT postType\_chk1 CHECK(postTypeId >0

AND postTypeId<=99999),

   categoryId NUMBER(5) CONSTRAINT cat\_chk1 CHECK(categoryId >0 AND

categoryId<=99999),

   mediaId    NUMBER(5),

   CONSTRAINT pk\_postId PRIMARY KEY (postId),

   CONSTRAINT fk\_parentid\_P FOREIGN KEY (parentId) REFERENCES

Post(postId),

   CONSTRAINT fk\_userId\_P FOREIGN KEY (userId) REFERENCES

IttUser(userId),

   CONSTRAINT fk\_postTypeId\_P FOREIGN KEY (postTypeId) REFERENCES

PostType(postTypeId),

   CONSTRAINT fk\_categoryId\_P FOREIGN KEY (categoryId) REFERENCES

Category(categoryId));

CREATE TABLE Rank (

   rankId   NUMBER(5) CONSTRAINT rankid\_chk CHECK(rankId >0 AND

rankId<=99999),

   postRank NUMBER(5) NOT NULL  CONSTRAINT postRank\_chk\_R

CHECK(postRank >0 AND postRank<=99999),

   userId   NUMBER(5) NOT NULL  CONSTRAINT userId\_chk\_R CHECK(userId >0

AND userId<=99999),

  postId   NUMBER(5) CONSTRAINT postId\_chk\_R CHECK(postId >0 AND

postId<=99999),

   CONSTRAINT pk\_rankId PRIMARY KEY (rankId),

   CONSTRAINT fk\_userId\_R FOREIGN KEY (userId) REFERENCES

IttUser(userId),

   CONSTRAINT fk\_postId\_R FOREIGN KEY (postId) REFERENCES Post(postId));

CREATE TABLE MediaType

   mediaTypeId   NUMBER(5) CONSTRAINT mediaTypeid\_chk

CHECK(mediaTypeId >0 AND mediaTypeId<=99999),

   MediaTypeDesc VARCHAR2(20) NOT NULL UNIQUE CONSTRAINT

mediaTypeDesc\_chk CHECK(MediaTypeDesc IN ('.jpg','.jpeg','.png','.gif', '.docx',

'.doc', '.pdf', '.java', '.html', '.txt', '.htm')),

   CONSTRAINT pk\_mediaTypeId PRIMARY KEY (mediaTypeId));

CREATE TABLE Media (

   mediaId        NUMBER(5) CONSTRAINT mediaid\_chk CHECK(mediaId >0 AND

mediaId<=99999),

   mediaTypeId    NUMBER(5),

   url            VARCHAR2(40) NOT NULL,

   postId         NUMBER(5),

   CONSTRAINT pk\_mediaId PRIMARY KEY (mediaId),

   CONSTRAINT fk\_mediaTypeId\_M FOREIGN KEY (mediaTypeId ) REFERENCES

MediaType(mediaTypeId),

   CONSTRAINT fk\_postId\_M FOREIGN KEY (postId) REFERENCES Post(postId ));

COMMIT;

# Populated the script with data

* **UserType**

INSERT INTO UserType(userTypeId, typeDescription)

VALUES(UserType\_seq.NEXTVAL,'U');

INSERT INTO UserType(userTypeId, typeDescription)

VALUES(UserType\_seq.NEXTVAL,'A');

* **IttUser Table**

INSERT INTO IttUser (userId, firstName, surname, password, email, score, profilePic, userTypeId)

VALUES(IttUser\_seq.NEXTVAL, 'Aoife','Sayers - USER', 'myPassword123','aoifesayers@gmail.com',34,'mypic.jpg',1);

INSERT INTO IttUser (userId, firstName, surname, password, email, score, profilePic, userTypeId)

VALUES(IttUser\_seq.NEXTVAL, 'Helen','O Brien - Admin', 'myPassword123','h3l3n@gmail.com',322,'mypic.png',2);

INSERT INTO ITTUser (userId, firstName, surname, password, email, score, profilePic, userTypeId)

VALUES(IttUser\_seq.NEXTVAL, 'Katie', 'Griffiths', 'myPass!', 'katie@gmail.com', 30, 'mypicKatie.jpg', 1);

INSERT INTO ITTUser (userId, firstName, surname, password, email, score, profilePic, userTypeId)

VALUES(ittUser\_seq.nextval, 'Katie', 'Griffiths', 'myPass!', 'katie@gmail', 30, 'mypicKatie.jpg', 1);

* **PostType**

INSERT INTO PostType(postTypeId,postType)

VALUES(PostType\_seq.NEXTVAL,'Informative');

INSERT INTO PostType(postTypeId,postType)

VALUES(PostType\_seq.NEXTVAL,'Reply');

INSERT INTO PostType(postTypeId,postType)

VALUES(PostType\_seq.NEXTVAL,'Question');

INSERT INTO PostType(postTypeId,postType)

VALUES(PostType\_seq.NEXTVAL,'Answer');

* **Category**

INSERT INTO Category(categoryId,categoryName)

VALUES(Category\_seq.NEXTVAL,'Computing');

INSERT INTO Category(categoryId,categoryName)

VALUES(Category\_seq.NEXTVAL,'Business');

INSERT INTO Category(categoryId,categoryName)

VALUES(Category\_seq.NEXTVAL,'Creative Media');

INSERT INTO Category(categoryId,categoryName)

VALUES(Category\_seq.NEXTVAL,'Engineering');

INSERT INTO Category(categoryId,categoryName)

VALUES(Category\_seq.NEXTVAL,'Nursing');

INSERT INTO Category(categoryId,categoryName)

VALUES(Category\_seq.NEXTVAL,'Social Science');

* **Tag**

INSERT INTO Tag (tagId, tagName, categoryId)

VALUES(Tag\_seq.NEXTVAL,'HTML',1); --Testing computing category

INSERT INTO Tag (tagId,tagName, categoryId)

VALUES(Tag\_seq.NEXTVAL,'Java',1); -- Testing Computing category again

INSERT INTO Tag (tagId,tagName, categoryId)

VALUES(Tag\_seq.NEXTVAL,'Accounting',2); -- Testing business category again

INSERT INTO Tag (tagId, tagName, categoryId)

VALUES(Tag\_seq.NEXTVAL, 'Budgeting', 2);

INSERT INTO Tag (tagId, tagName, categoryId)

VALUES(Tag\_seq.NEXTVAL, 'Photoshop', 1);

INSERT INTO Tag (tagId, tagName, categoryId)

VALUES(Tag\_seq.NEXTVAL, 'Nursing Practice', 4);

* **Post**

INSERT INTO Post (postId, parentId, postDate, postText, userId, postTypeId, categoryId)

VALUES(Post\_seq.NEXTVAL,1,SYSDATE,'Hello World! Informative post here', 1, 2, 1);

INSERT INTO Post (postId, parentId, postDate, postText, userId, postTypeId, categoryId)

VALUES(Post\_seq.NEXTVAL,1,SYSDATE,'Hi there - reply post here', 2, 1, 2);

INSERT INTO Post (postId, parentId, postDate, postText, userId, postTypeId, categoryId)

VALUES(Post\_seq.NEXTVAL,1,SYSDATE,'Who is this? - question post here', 1, 2, 3);

INSERT INTO Post (postId, parentId, postDate, postText, userId, postTypeId, categoryId)

VALUES(Post\_seq.NEXTVAL,1,SYSDATE,'It is me - answer post here', 1, 2,4);

INSERT INTO Post (postId, parentId, postDate, postText, userId, postTypeId, categoryId)

VALUES(Post\_seq.NEXTVAL, 1, SYSDATE, 'This is an informative post about HTML', 1, 1, 1);

INSERT INTO Post (postId, parentId, postDate, postText, userId, postTypeId, categoryId)

VALUES(Post\_seq.NEXTVAL, 2, SYSDATE, 'Question post about HTML again', 1, 2, 1);

INSERT INTO Post (postId, parentId, postDate, postText, userId, postTypeId, categoryId)

VALUES(Post\_seq.NEXTVAL, 2, SYSDATE, 'Reply post about HTML question above', 2, 3, 1);

* **Rank**

INSERT INTO Rank (rankId, postRank, userId, postId)

VALUES(Rank\_seq.NEXTVAL, 1, 1, 1); -- Aoife's Hello World Informative post has rank number 1

INSERT INTO Rank (rankId, postRank, userId, postId)

VALUES(Rank\_seq.NEXTVAL, 2, 2, 2);

* **Media Type**

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.jpg');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.jpeg');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.png');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.gif');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.docx');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.doc');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.txt');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.pdf');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.java');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.html');

INSERT INTO MediaType (mediaTypeId,MediaTypeDesc)

VALUES(MediaType\_seq.NEXTVAL,'.htm');

* **Media**

INSERT INTO MEDIA(mediaId, mediaTypeId, url, postId)

VALUES(media\_seq.NEXTVAL, 1, '\documents\myPhotos\',1);

--adding .jpg file to Hello World post - file stored in \documents\myPhotos

INSERT INTO MEDIA(mediaId, mediaTypeId, url, postId)

VALUES(media\_seq.NEXTVAL, 8, '\documents\javaBits\',1);

-- adding java file to hello World post

INSERT INTO Media (mediaId, mediaTypeId, url, postId)

VALUES(media\_seq.NEXTVAL, 1, 'images\mypic.jpg', 2);

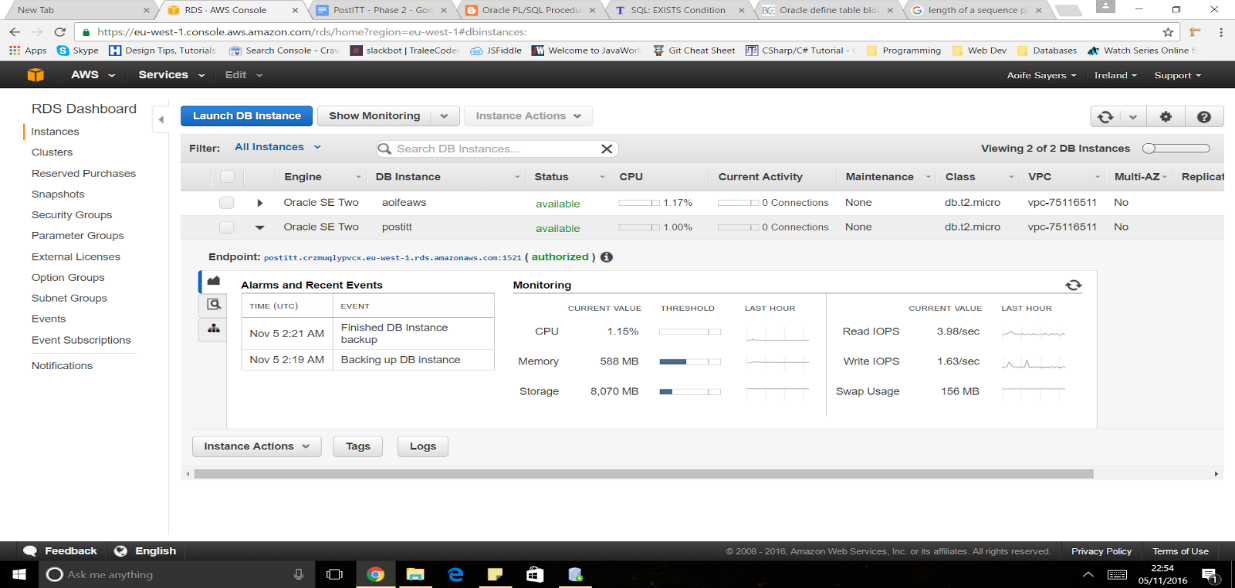
COMMIT;

# Implementing the database in parallel

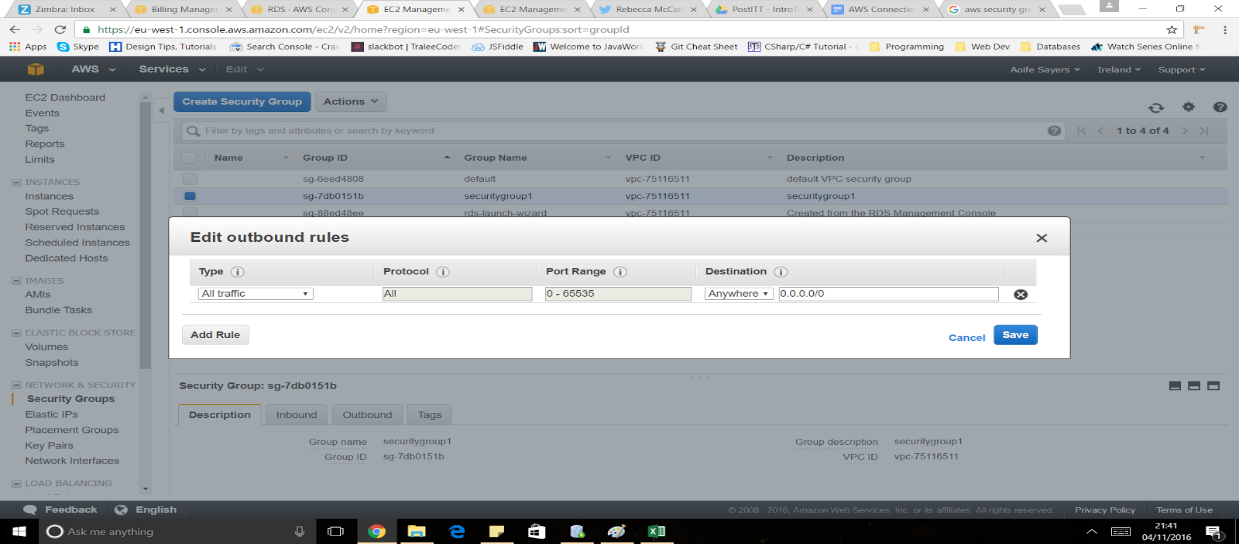
We decided to use Amazon Web Services for implementing our database in parallel. Each team member had access to the postitt instance on AWS.

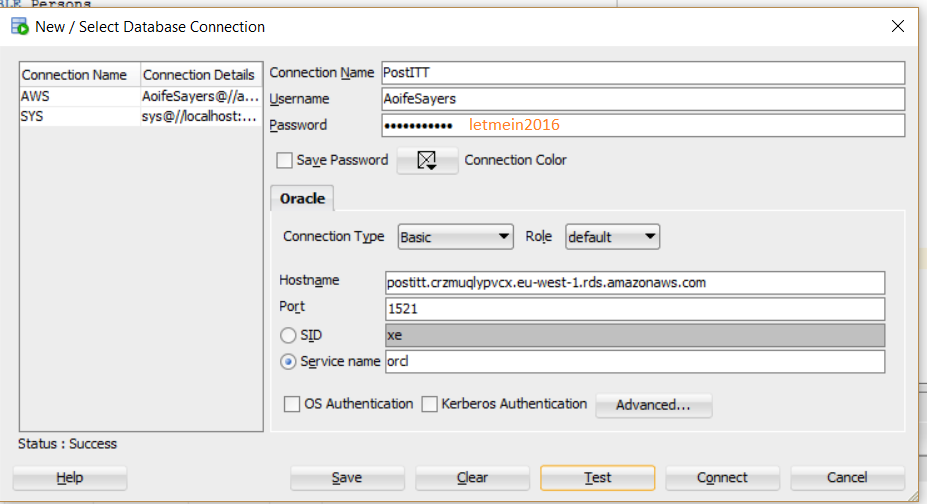
RDS

Oracle SE TWO DB instance

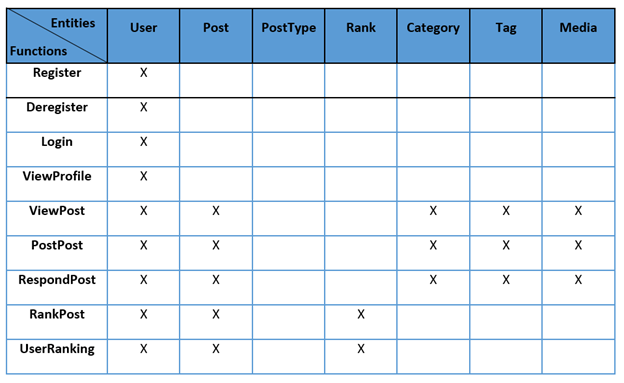


We discovered that to connect to SQLDeveloper, we had to create a new security group and edit the Inbound and Outbound rules to all traffic





# Implementing the Business Rules



# Business Rules Altered

**Structural**

* User’s must use valid email in order to register for the website
* User’s first & last names must be valid, containing only letters
* A user’s password must be at least 5 letters long, be mixed case and include at least 1 number
* Category names must be set up by the administrator & must be unique names
* Tag names can be entered by the user & must be unique names
* User must be registered to use application
* The Modulator cannot participate in posting
* A user must choose if an initial post is an informative post or a question post
* A users score is calculated by the number of posts X(times) the sum of their post ranks. The score is recalculated & updated each time they post a new post
* A Post type must be Informative, Question, Answer or reply post
* A User type must be User ‘U’ or Administrator/Moderator ‘A’
* The postText cannot contain any profanities
* The valid media types are: .docx, .doc, .pdf, .java, .html, .txt, .htm, .jpg, .jpeg, .png, .gif

**Constraint on a field**

* Registration details cannot be null
* Login username must be valid
* User email must be valid
* Implemented using email checking trigger. Email must contain an @ symbol
* Login password must be valid
* Implemented using the password trigger on insert
* A post cannot be null
* Implemented using domain Constraints
* A post must have a category
* A post can only have one category
* A post can have no more than 10 tags
* Users can only vote once per post
* No more than 1 pieces of media per post
* Media file can be no larger than 5 Mb

# Triggers

* First Name check on ITTUser - created using a trigger userITTNameCheck
* Surname Name check on ITTUser - created using a trigger userITTSurnameCheck
* Profile pic on IttUser – created using a trigger profilePic
* Profile pic on IttUser – created using a trigger with Blob profilePicTrigger
* Update score - created using a procedure updateScore & a trigger INSPOSTUPDATESCORE
* Password Strength password - created using a validate password trigger
* Validate an email - created a trigger called ValidateEmail
* Validate media type - created a trigger called mediaType trigger
* Increment posts rank - created using a postRanking trigger
* Moderator unable to post – created using trigger modulatorNoPosting
* Profanities in post Text - created using a trigger called SwearWords and postDecrement function
* Duplicate tags – created using trigger TAGTRIGGER
* Tag Name Duplication check - created using a trigger tagNameDupeTrigger

# Procedures

* Register a user - created using createITTUser procedure
* Deregister user - created using  DeregisterITTUser procedure
* Create a User UserType - created using procedure createUserType
* Create an Administrator UserType - created using procedure createAdminstratorType
* Create a User UserType – created using procedure createUserType
* Login user - created using login procedure
* Load a blob – created using procedure LoadABlobProcedure
* Load a clob – created using procedure LoadAClobProcedure
* Post a post – created using procedure insertPost
* Initial Post – created using procedure InitialPost
* Creating reply post  - created using procedure ReplyPostProc
* Creating question post - created using procedure QuestionPost
* Creating reply post – created using procedure ReplyPost
* Creating Answer Post  - created using a procedure AnswerPost
* Creating Informative Post - created using a procedure InformativePost
* Creating a category - created using a procedure CreateCategory
* Creating a tag - created using a procedure CreateTag
* Media Approval - created using a media\_approved procedure
* Media Flagged - created using a media\_flagged procedure
* Media Disabled - created using a media\_disabled procedure
* Update score - created using a procedure updateScore & a trigger

# Views

* ViewUserTypePosts
* ViewUserTypeRankCategoryPosts
* ViewUserPostsTypeRankCatTagRank
* View User Profile - created using a view called ViewUser
* View Posts - created using a view called ViewPosts
* View Profile - created using ViewUserProfile

**Need for Views**

* A virtual table that does not physically exist - just displays selected data
* Takes up no storage space
* Created by a query [joining one or more tables](https://www.techonthenet.com/oracle/joins.php).
* Reducing complex queries
* No DML operations - UPDATE or INSERT

CREATE or REPLACE VIEW sup\_orders AS  
 SELECT suppliers.supplier\_id, orders.quantity, orders.price  
 FROM suppliers  
 INNER JOIN orders  
 ON suppliers.supplier\_id = orders.supplier\_id  
 WHERE suppliers.supplier\_name = 'Apple';

# Index

* Index to View Name and Score
* Index for Posts
* Index for Rank
* Index for Category
* Index for Tag

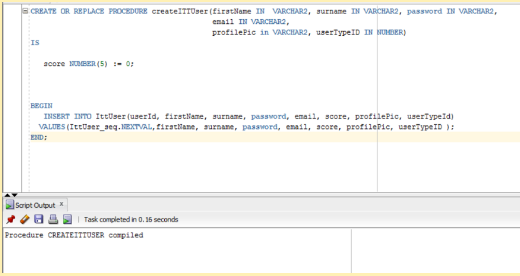
**Need For Index**

* Allow for faster retrieval & performance of queries by reducing the number of database data pages that have to be visited/scanned
  + Speed up SELECT queries and WHERE clauses
  + Slows down UPDATE and INSERT statements
* An Index is a pointer to data in a table.
* Use indexes on large tables - 50 rows +
* Indexes can be unique - duplicate entries in column are not allowed
  + Similar to the UNIQUE constraint
* CREATE INDEX index\_name  
  ON table\_name (column\_name);
* Avoid using indexes if the tables:
  + Small tables
  + Have a lot of UPDATE & INSERT Queries
  + Have a lot of NULL operations

# Procedures

**Register a User**

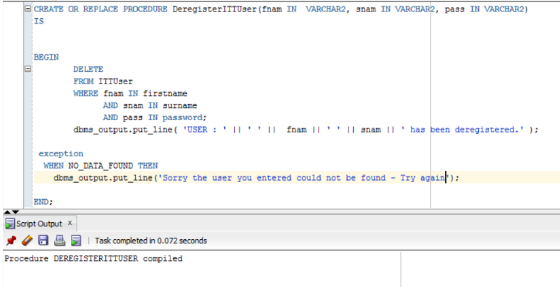
The create IttUser procedure requires parameters such as firstname, surname, password, email, profilePic and the userTypeId. By default the score is set to 0, the userId is set using the sequences next value. This procedure takes the parameters passed into it and inserts a new row in IttUser table.



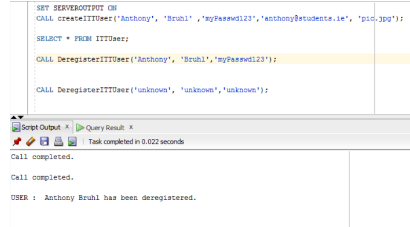
**Deregister a user**

The Deregister ITTUser procedure requires the firstname, surname and password of an ITT user to be passed into the procedure. PostITT sytem them finds the row where the firstname, surname and password entered exist & deletes the row from the table. The system confirms if the user has been deregistered. If no data was found - the NO\_DATA\_FOUND exception will output the error that occurred.

Data found & user Deregistered Data found & user Deregistered

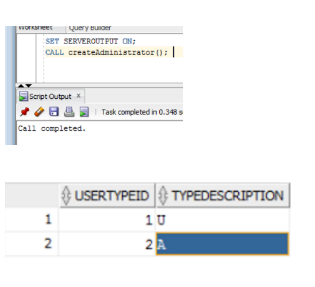
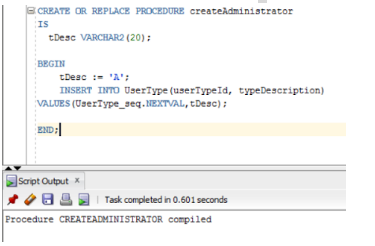


Data found & user Deregistered



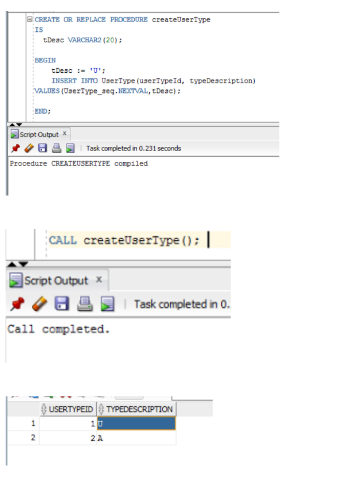
**Create an Administrator UserType Procedure**

Setting up the user type as an Administrator – ‘A’ for the ITT User. An Administrator user type cannot participate in posting



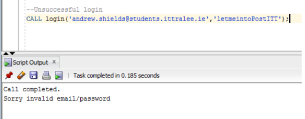
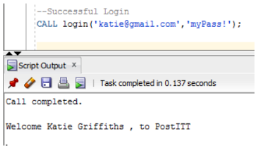
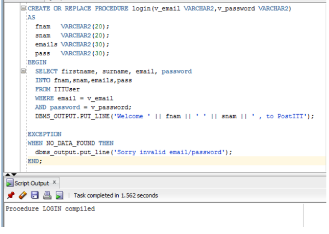
**Create UserType procedure**

Setting up the user type as a regular user who can participate in posting – ‘U’ for the ITT User



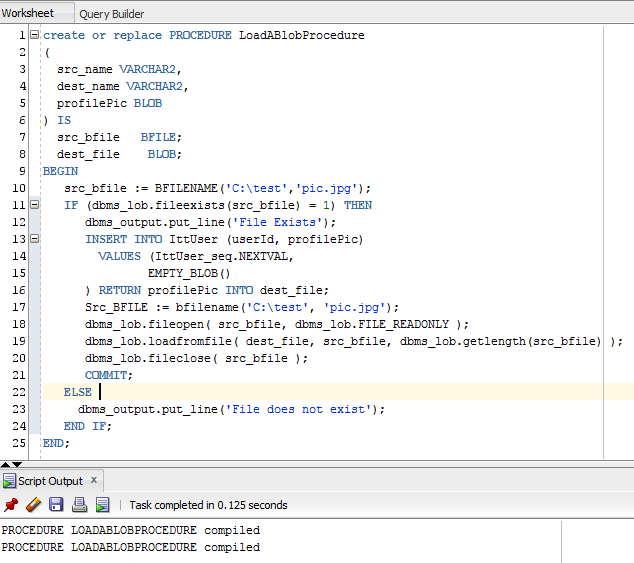
**Login**

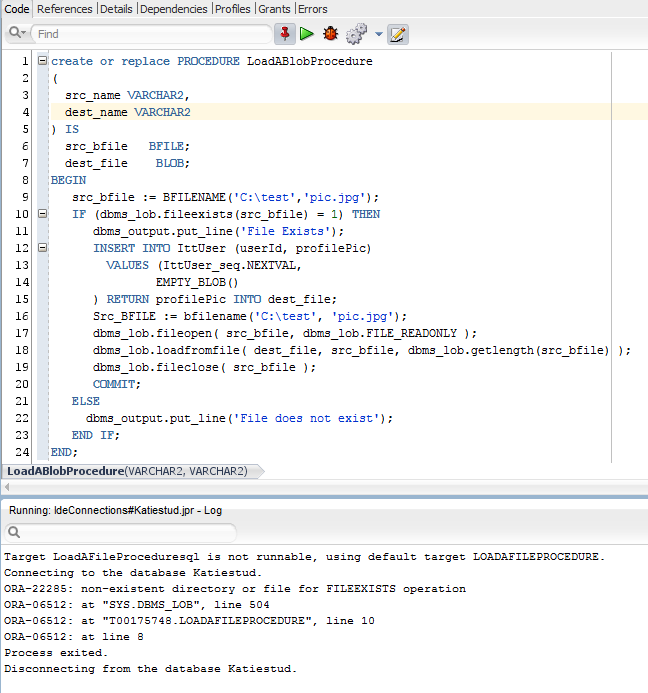
The login procedure requires the user to pass in their email and password. PostITT finds the email and corresponding password, name and surname from the ITTUser table. If the login is successful PostITT prints out a line welcoming the user. If data wasn’t found,  PostITT prints out a line saying the email/password was invalid.



**Load A Blob Procedure**

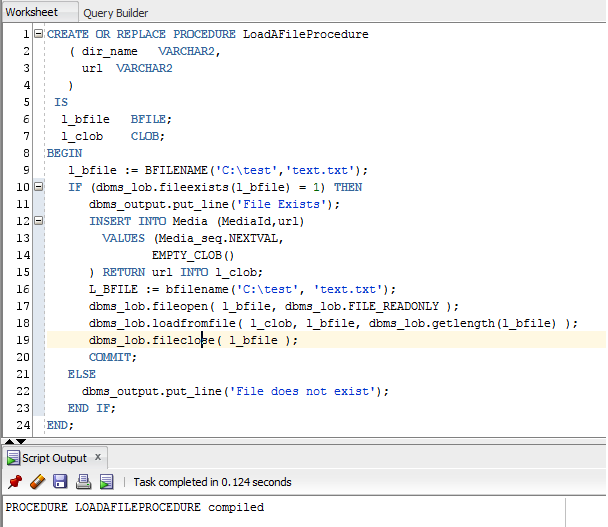
This procedure will allow the user to load a picture from there computer and use it.

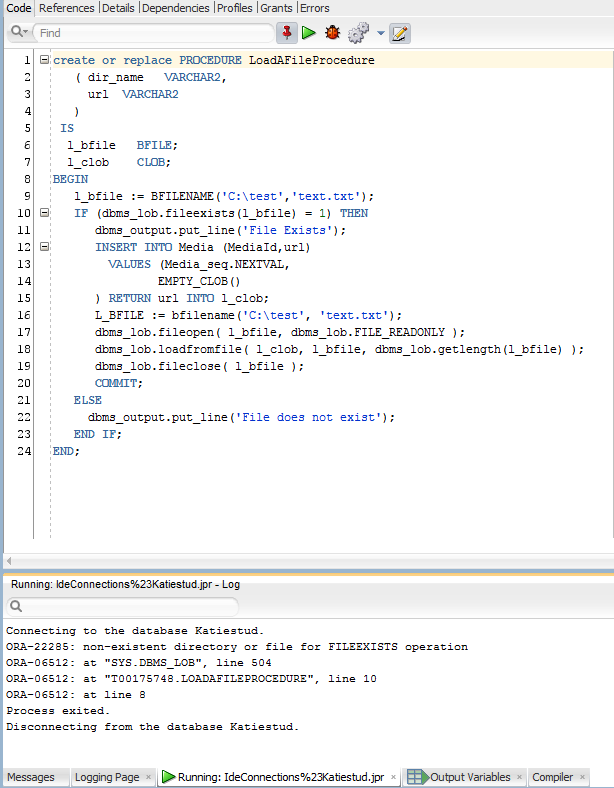




**Load a Clob Procedure**

This allows the user to load text info.

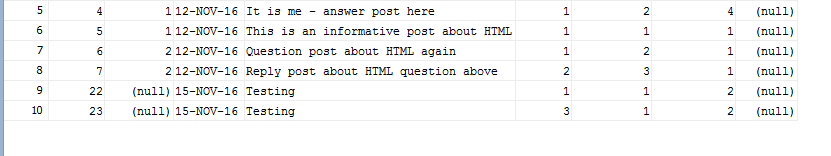




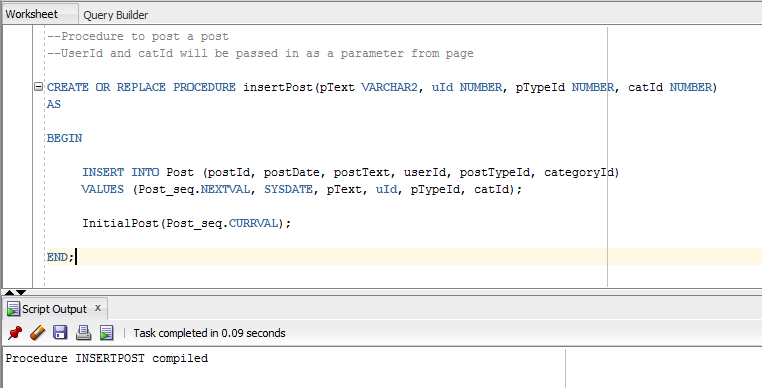
**Procedure to Insert a Post**

This procedure insert a post, it inserts postId by sequence, System Date, Post Text, User Id, Post Type Id and Category Id. The procedure also calls the Initial Post Procedure and inserts the Post Id into the Parent Id.

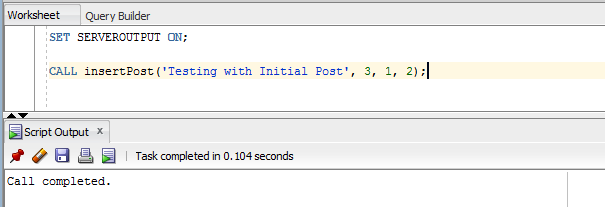
Post Table before insert



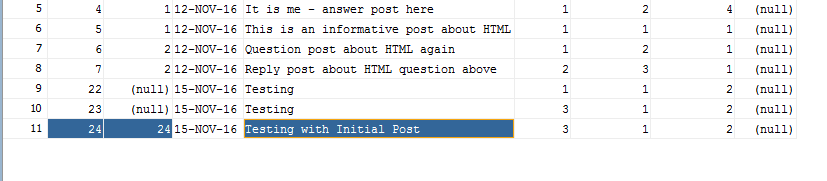
Insert Post Procedure



Call the Procedure

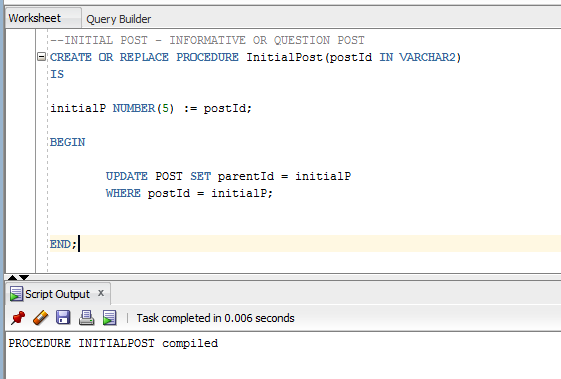


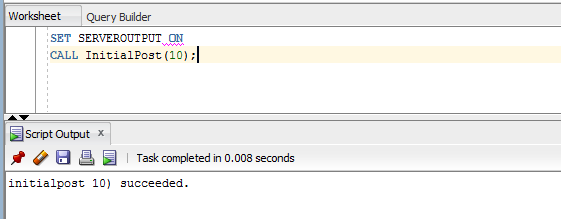
Result – Post Inserted



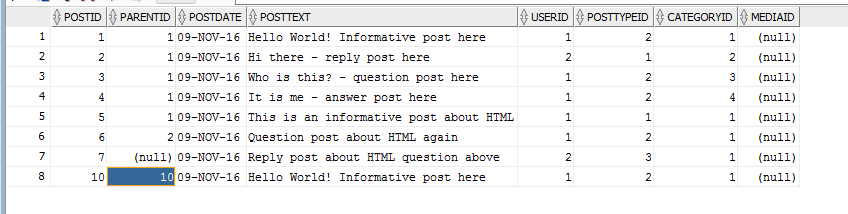
**Initial Post**

The initial post procedure takes the postId and inserts it into the parentId, when these are the same it indicates an initial or parent post. This procedure is called by the button used to submit the post.



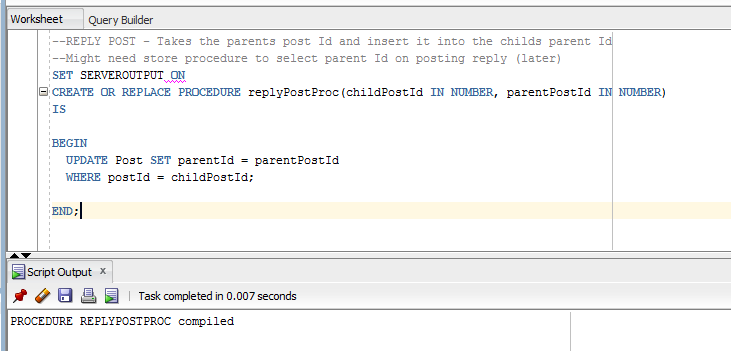


Result – Post Id Inserted into Parent Id

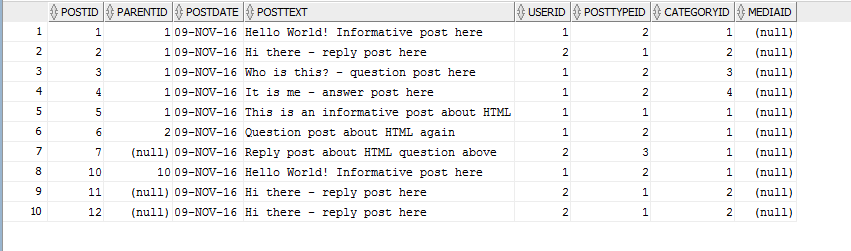


**Reply Post**

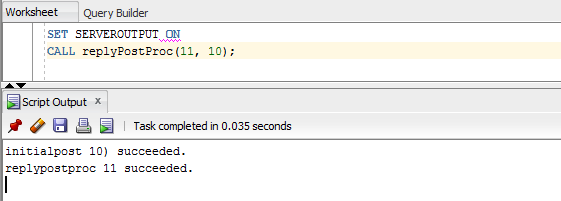
The reply post procedure takes the parents postId and inserts it into the parentId of the child post, when postId and parentId are different it indicates a child post. This procedure is called by the button used to submit a reply post.

****

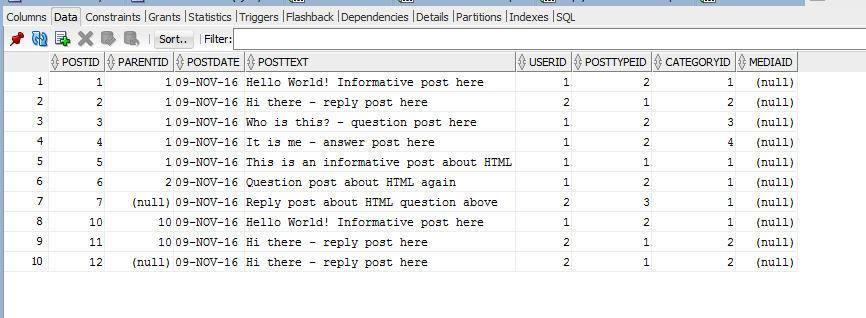
Before Reply Post Procedure

****

**Procedure Called**

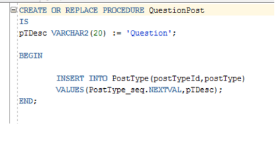
****

After Reply Post Procedure – Parent Post Inserted

****

**Question post type Procedure**

This procedure sets up an initial Question post.



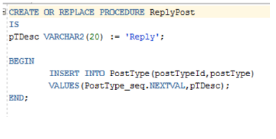
**Answer Post Procedure**

This procedure sets up a response Answer post



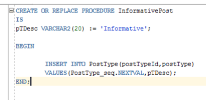
**Reply Post Procedure**

This procedure sets up a response reply post.



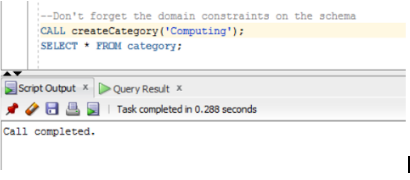
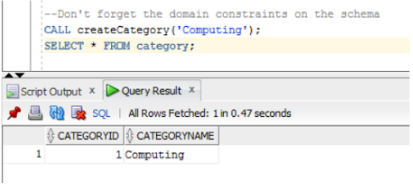
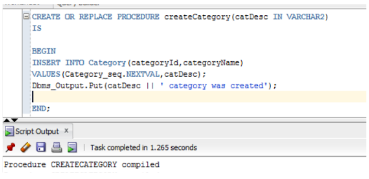
**Informative Post Procedure**

This procedure sets up an initial Informative post.



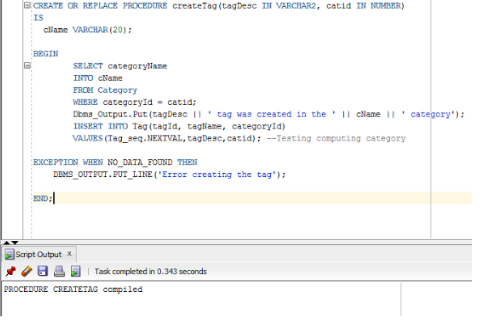
**Create Category**

This procedure creates a new category. The user passes in the category name and a sequence assigns the next category ID.

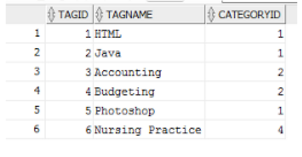


**Create Tag**

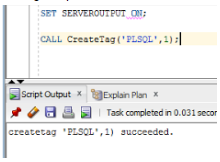
This procedure creates a new tag. The user passes in the tag name, cat id. A sequence assigns the next tag ID.



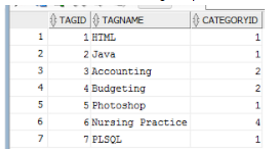
Before Testing. This image shows the tags already in the database.



Calling the Procedure

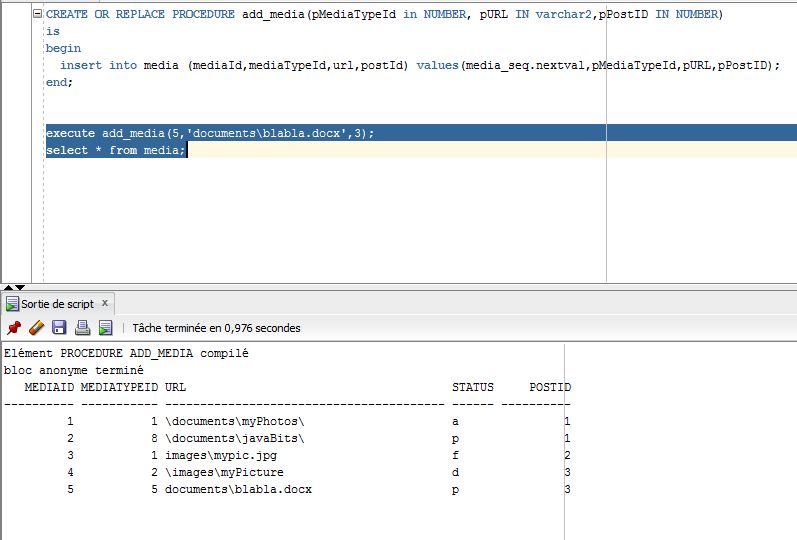


Data in the table after calling the procedure.



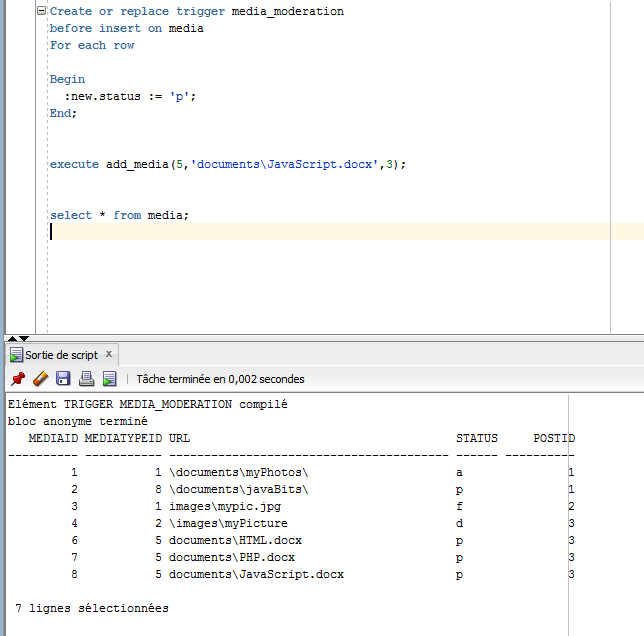
**Create media procedure**

This procedure creates a media according to the parameters

****

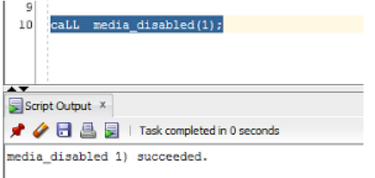
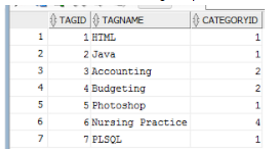
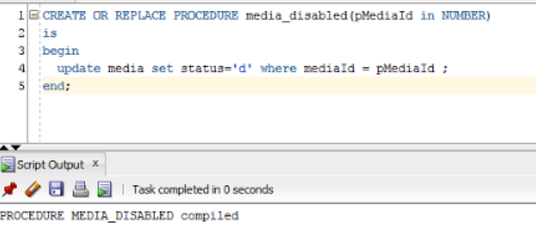
**Media\_moderation\_trigger**

When media is created, the media\_moderation\_trigger is triggered to change the media status. The media status will be “Pending”

****

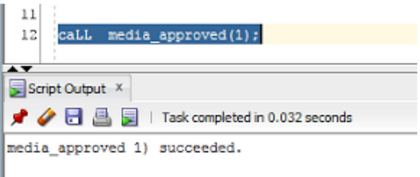
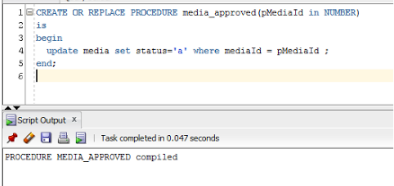
**Media Disabled Procedure**

This procedure allows the administrator to change the media status to “Disabled”. This means the media can’t be display on the website after this change.



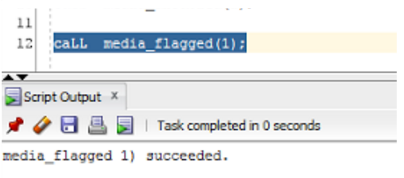
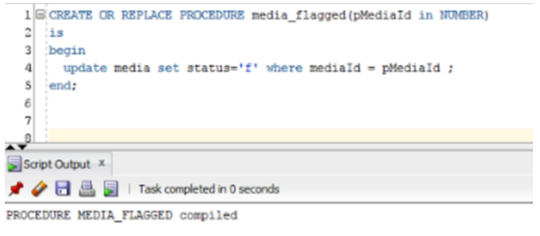
**Media Approved Procedure**

This procedure allows the administrator to change the media status to “Approved”. This means that the media can be displayed on the website after this change has happened. The moderator certifies that the media is suitable.



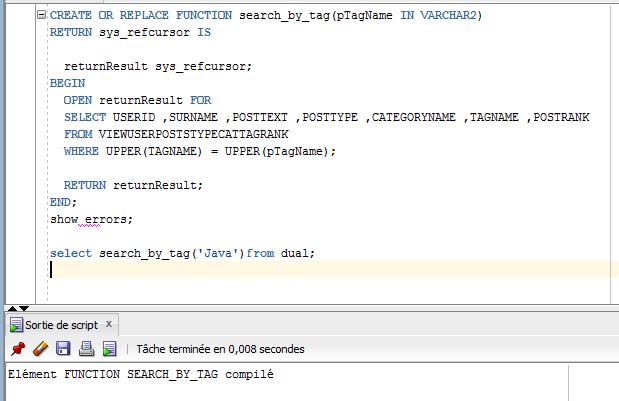
**Media Flagged Procedure**

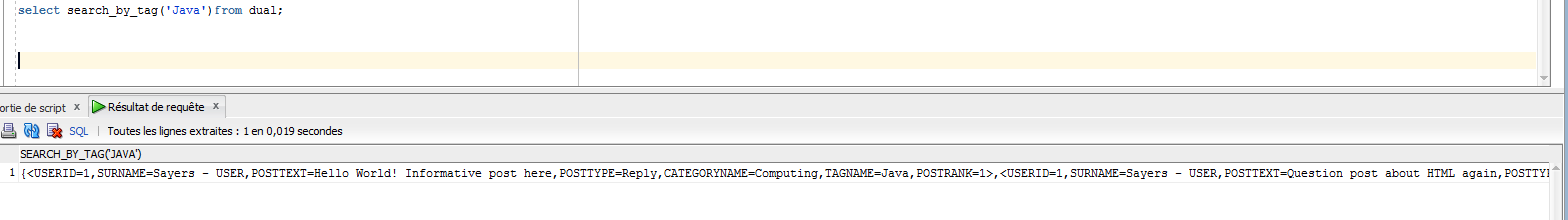
This procedure allows the users to change the media status to “Flagged”. This means the media will not be displayed on the website after this change has happened. The moderator must check that the media is then “Approved” or “Disabled” it.



**Search\_by\_tag procedure**

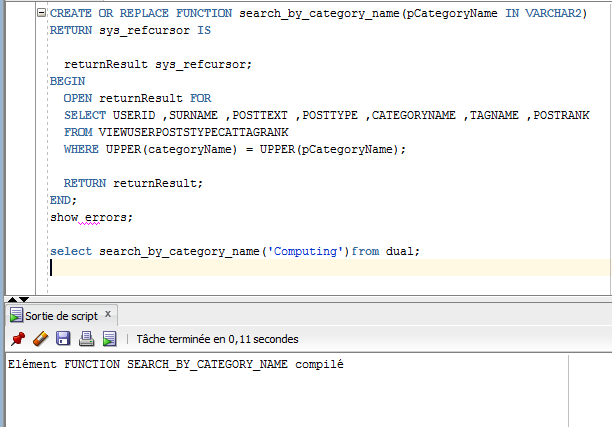
This procedure allows the users to search all of the posts which have a tag.

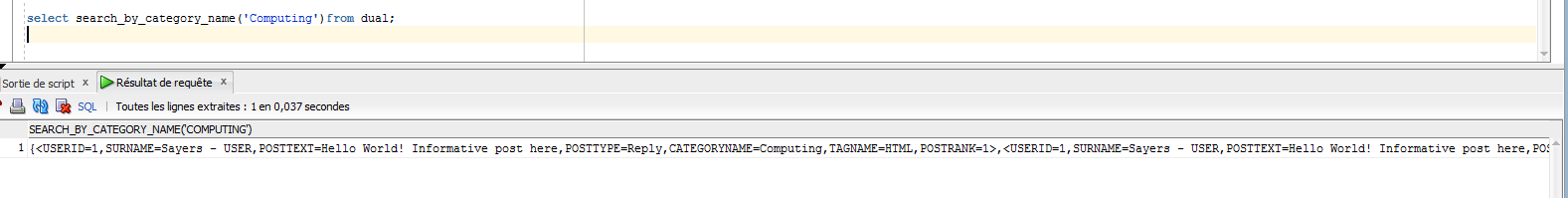
****

****

**Search\_by\_category\_name procedure**

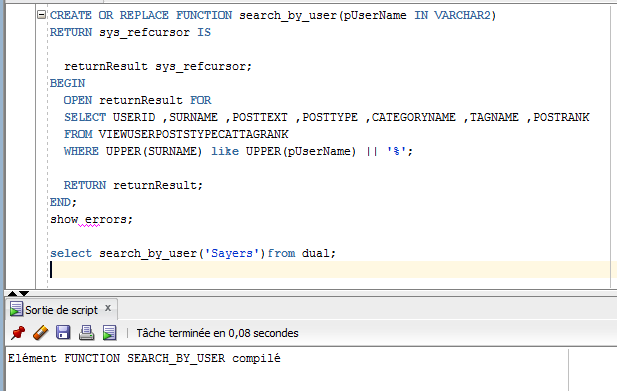
This procedure allows the users to search all of the posts which have a category assigned to it.

****

****

**Search\_by\_user procedure**

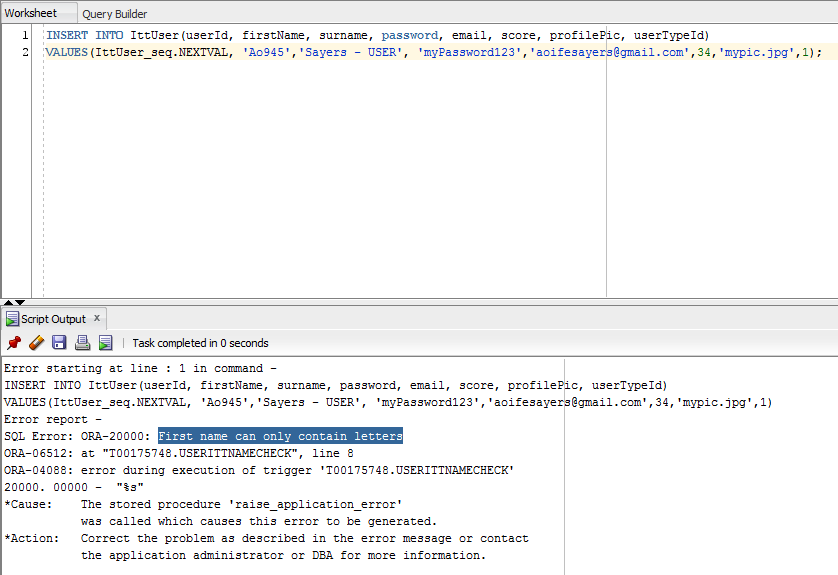
This procedure allows the users to search all posts which have a certain users name attached to it.

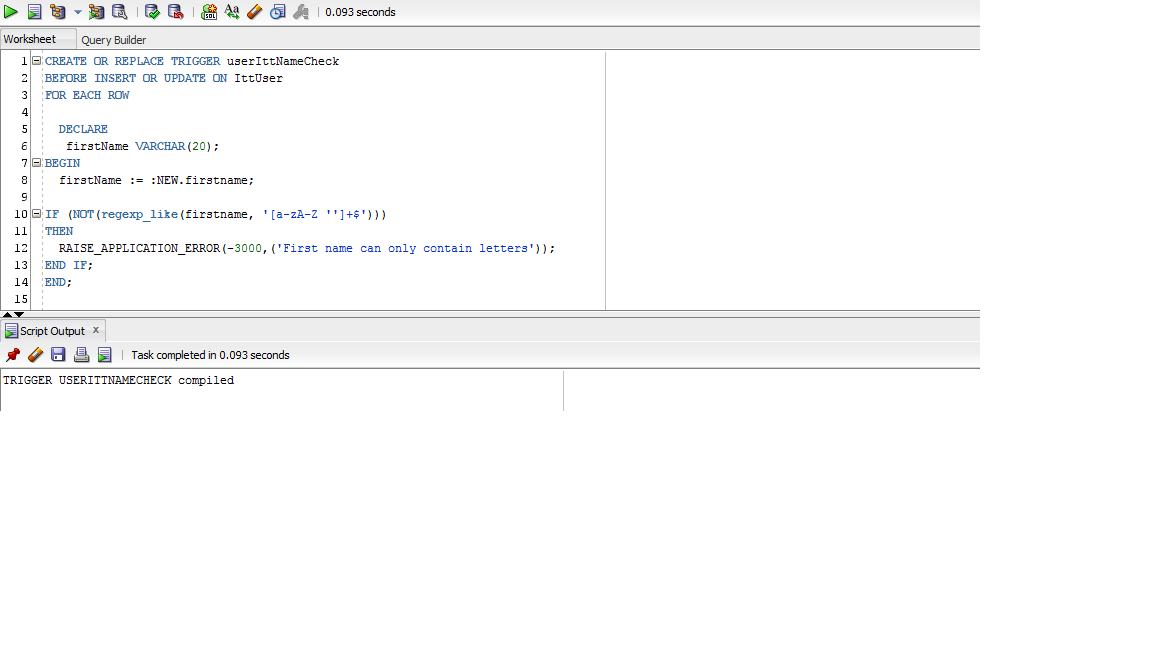
****

# Triggers

**First Name check**

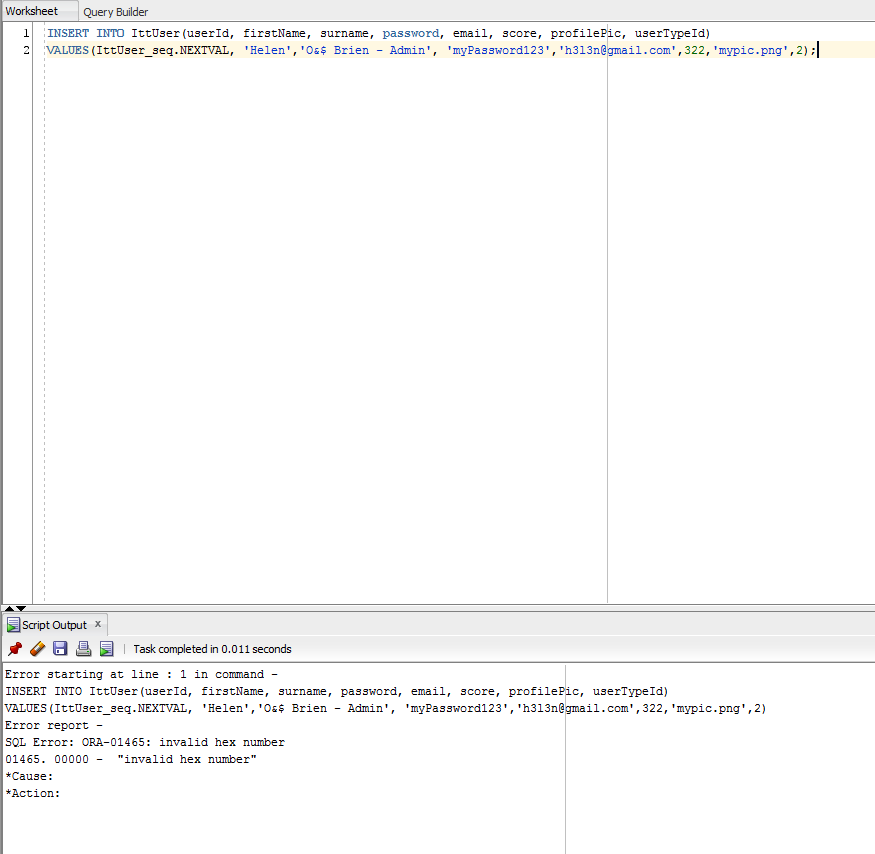
This is a trigger in which checks to see if the first name of the user is a valid name containing only letters.

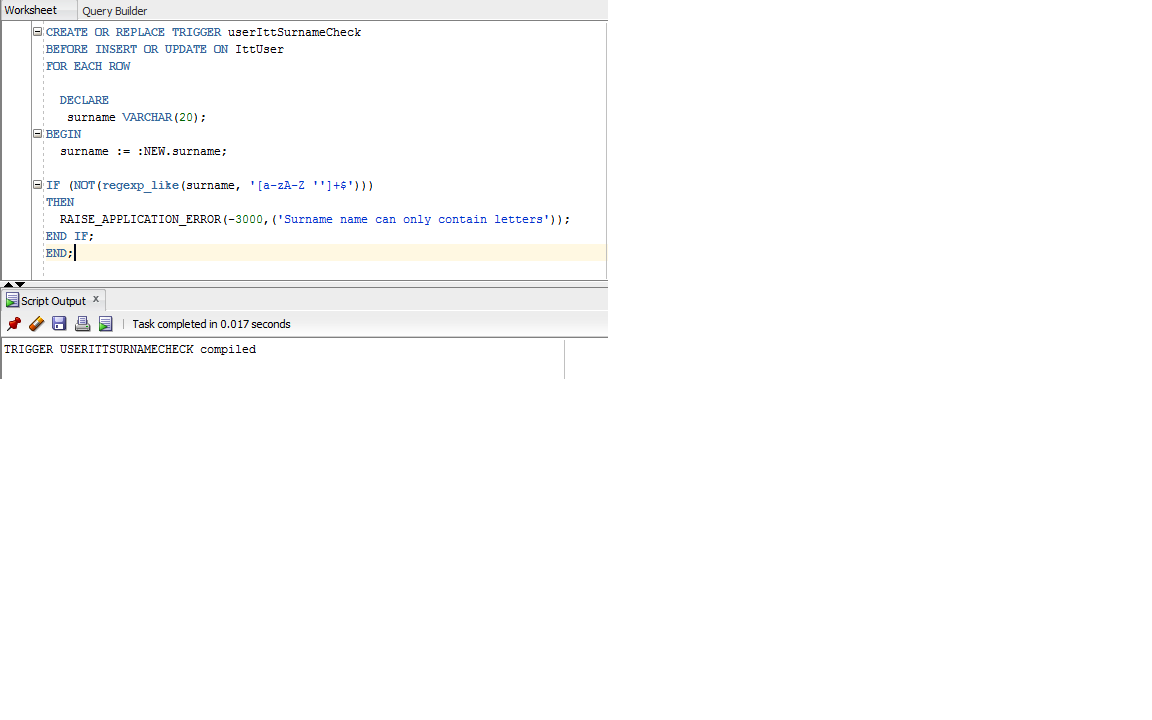




**Last Name Check trigger**

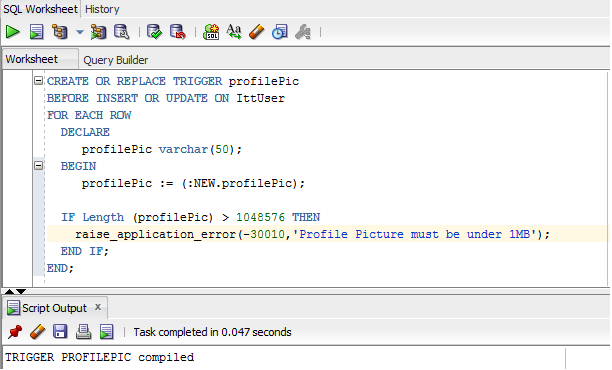
This is a trigger that checks that the users last name is a valid name and only contains letters.





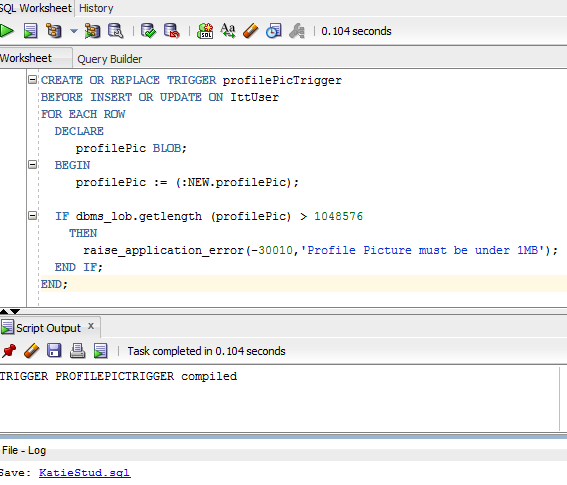
**Profile Pic Trigger**

This is a trigger to check that the user’s picture for their profile is not bigger than 5mb

****

**Blob Trigger ProfilePic**

This trigger was brought in to use instead of the previous trigger done above. It was decided to try to use a blob in the database.

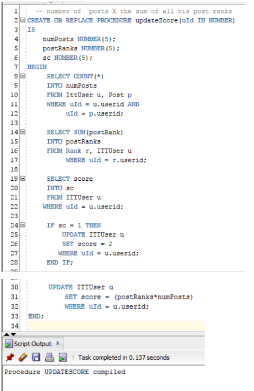
****

**Trigger & Procedure to Update Score**

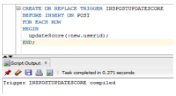
This procedure updates the score of a user, the score is calculated by:-

Number of posts for a userId   X   Total of all posts rank for a user = User Score

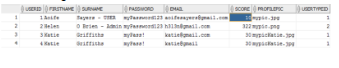
The trigger triggers when a post is inserted & calls the update score procedure.



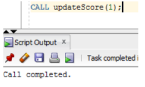
Trigger



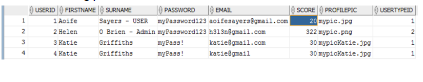
Before Calling



Call Procedure



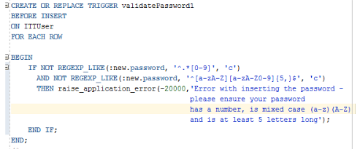
Result after calling procedure



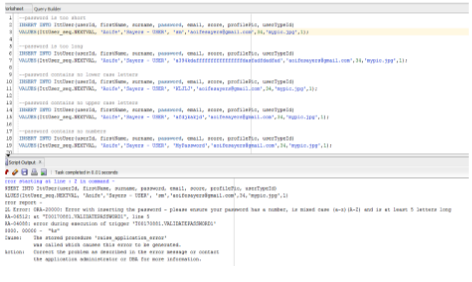
**Password Strength Trigger**

This trigger tests the password strength. Ensures the password has:

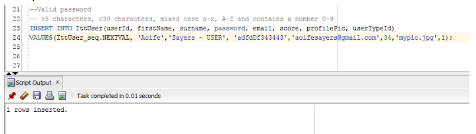
* At least 1 digit (0-9)
* Mixed case letters (a-z) (A-Z)
* 5 or more characters long



Failing inserts demonstration here..



Insert Passes



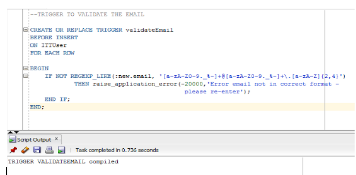
**Conclusion:** Decided to use regular expressions instead of

:new.password NOT LIKE '%[a-z]%' because it wasn’t working & more condense in code

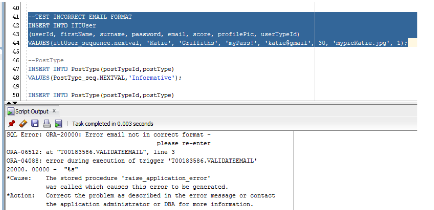
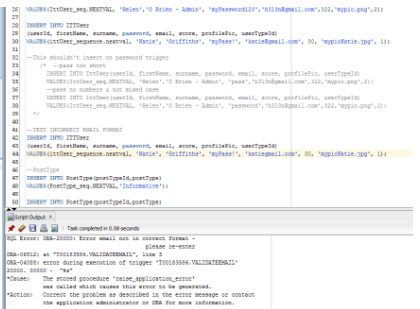
**Trigger to Validate Email**

This trigger tests the email validity, and ensures the email has:

* A digit (0-9) and/or Mixed case letters (a-z) (A-Z) at least 2 characters long for username
* Followed by @
* A digit (0-9) and/or Mixed case letters (a-z) (A-Z) at least 4 characters long for domain



Incorrect Inputs



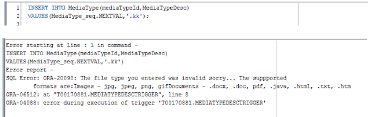
**Trigger to Validate Media Type**

This trigger tests the media types validity matches the business rules

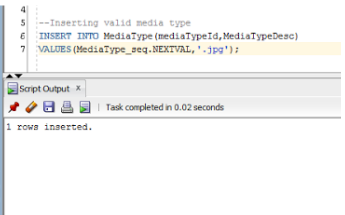
* Valid File Types
* Images - jpg, jpeg, png, gif
* Documents - .docx, .doc, pdf, .java, .html, .txt, .htm



Testing the trigger with invalid values

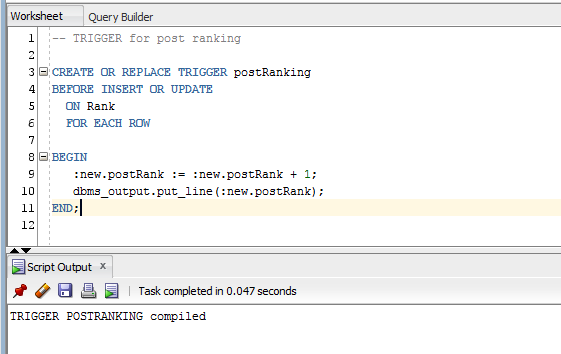


Testing trigger with valid values

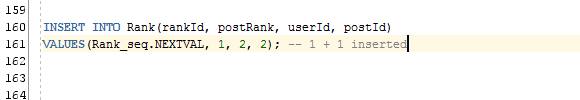


**Trigger to Increment Posts Rank**

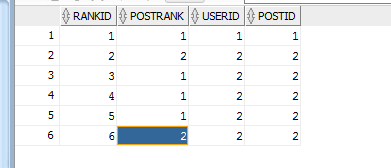
This trigger increments the postRank by 1 on insert or update of the rank



Insert

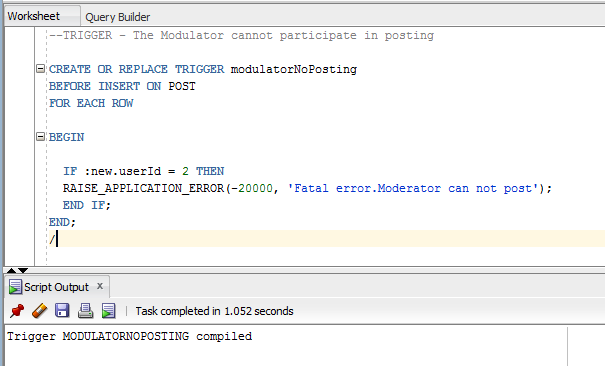
****

Result

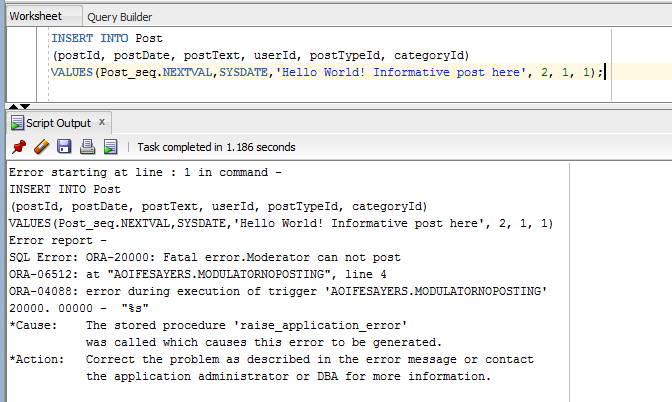
****

**Trigger - Moderator Unable to Post**

The moderator is not able to participate in posting, this trigger will raise an application error.

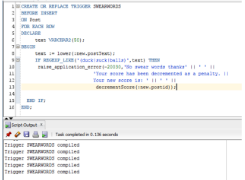


If the moderator tries to post an error occurs

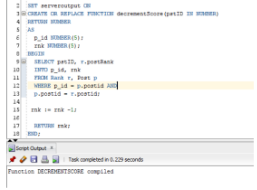
****

**Trigger & Function for profanities in Post Text - post rank is decremented**

If a postTexts contains the words ‘duck’, ‘suck’ or ‘balls’, a trigger will be fired on insert and the decrement score function will be called which reduces the user’s score by 1.

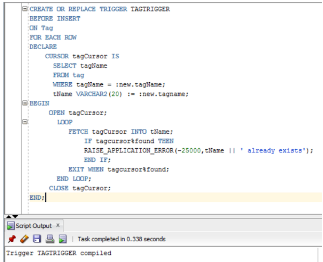


decrementScore Function – This function reduces the posts rank by one as a penalty for incorporating a swear word in the post text



**Trigger for duplicate tags**

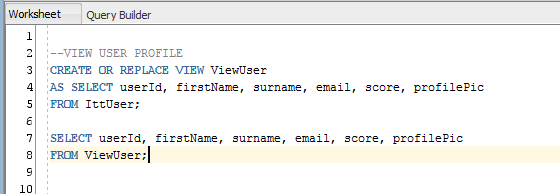
This trigger uses a cursor to check if the tag name being inserted already exists in the database by using a cursor and while loop to iterate through each row. If the tag name being inserted does not exist in the database, the trigger does not throw any application errors and successfully inserts into the database.



# Views

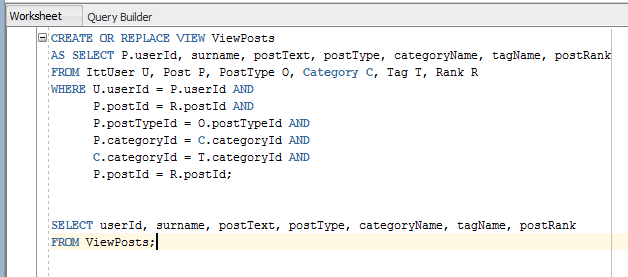
**View User Profile**

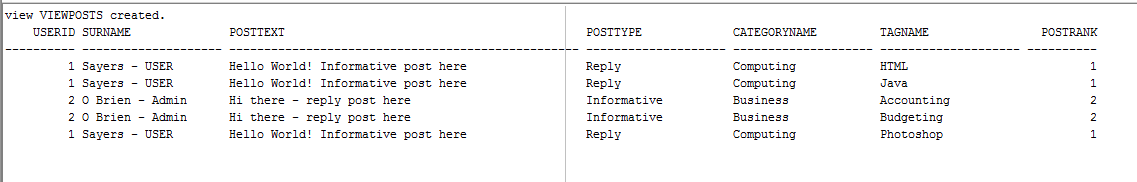
User profile is the result set of a stored query to display a user and their profile

****

**View Post**

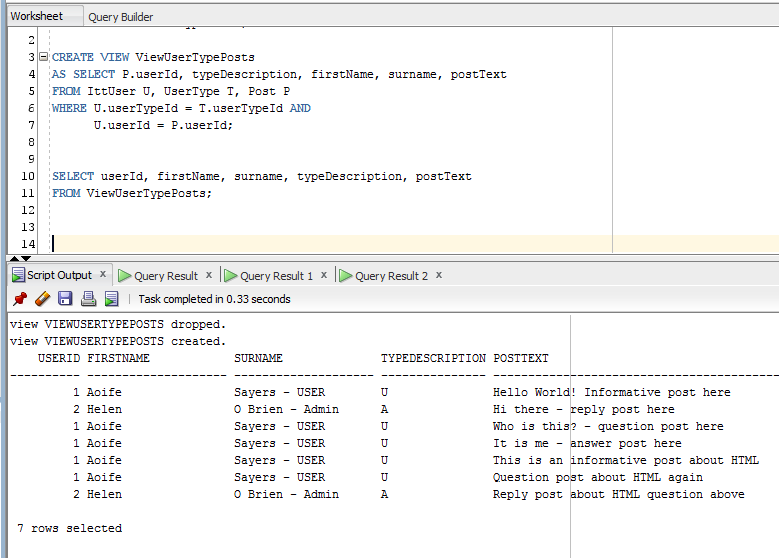
View post is the result set of a stored query to display a post and the information

****

****

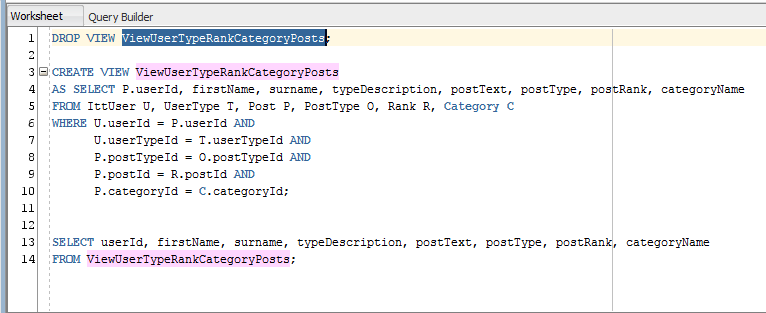
**ViewUserTypePosts**

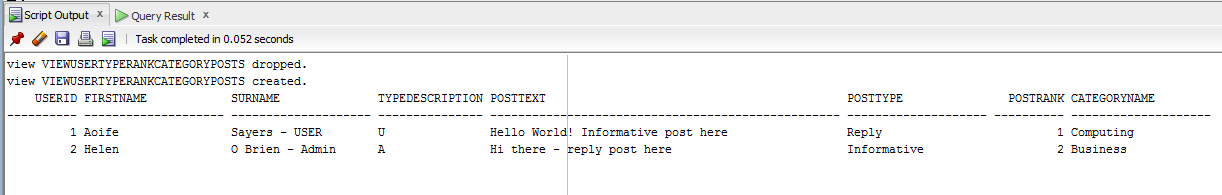
View user type post is the result set of a stored query to display a user’s post and the information

****

**ViewUserTypeRankCategoryPosts**

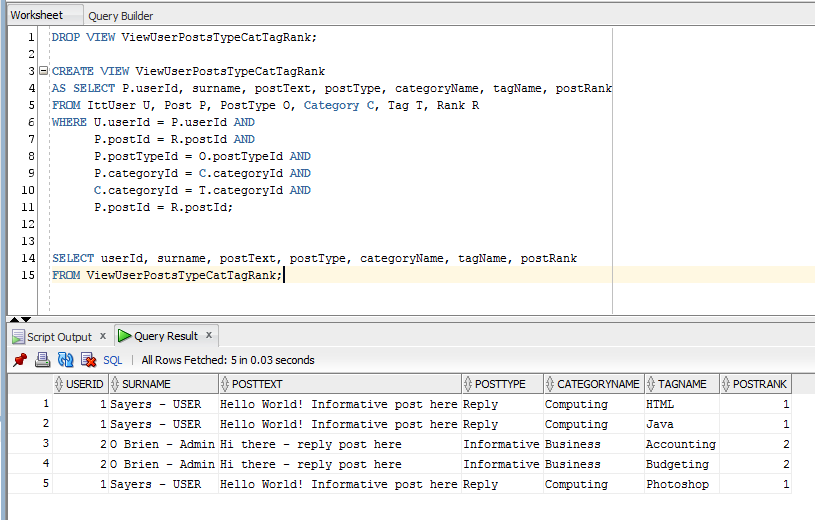
View user type rank category posts is the result set of a stored query to display a user, post and the posts information.

****

****

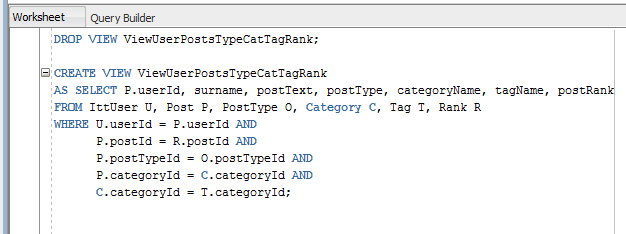
**ViewUserPostsTypeRankCatTagRank**

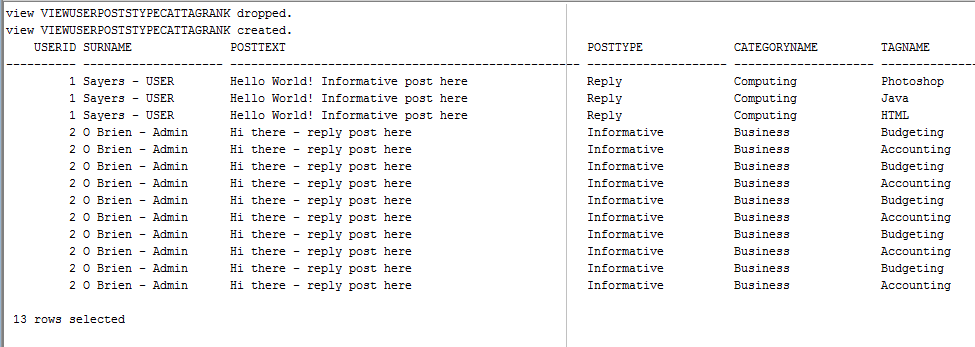
View user post type rank category tag rank is the result set of a stored query to display a user, post, post type, category name, tag name and the post rank information.

****

**View User Posts Type Cat Tag Rank**

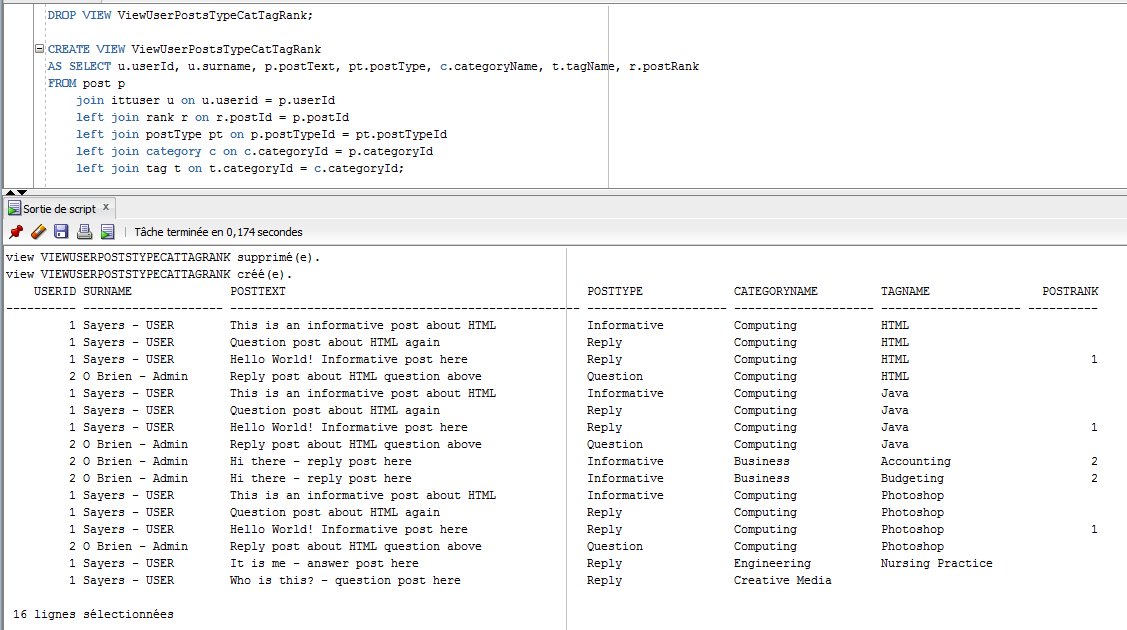
View user post type cat tag rank is the result set of a stored query to display a user, post, post type, category name, tag name information.

****

****

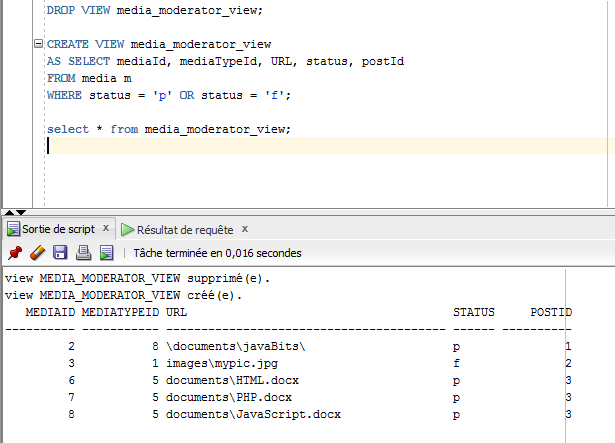
**ViewUserPostsTypeCatTagRank**

This view shows all the relationships with the post table. This view is used in search a function.

****

**Media\_moderator\_view**

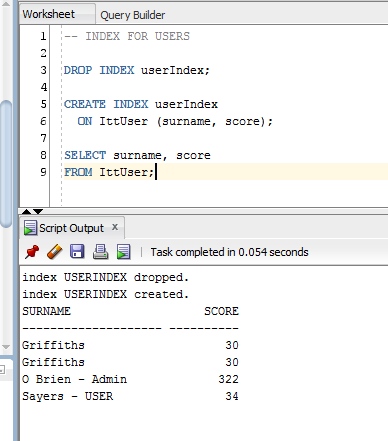
This view allows the moderator to see all of the media that is “Pending” or in a “flagged” state.

****

# Index’s

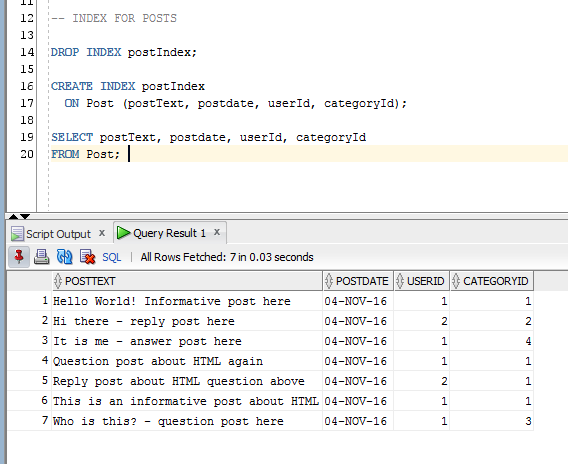
**User Index**

User Index is a pointer to data in what will be a large table for Users, for faster retrieval & performance of queries.

****

**Post Index**

Post Index is a pointer to data in what will be a large table for Posts, for faster retrieval & performance of queries.

****

# References

"Oracle PL/SQL: Verify Password - A Function Useful In Password Verification And Maintenance.". Psoug.org. N.p., 2016. Web. 29 Oct. 2016.

"Oracle / PLSQL: Sequences (Autonumber)". Techonthenet.com. N.p., 2016. Web. 29 Oct. 2016.

"Loading Documents And Other File Data Into The Oracle Database (Oracle Search - Technical Tips)". *Blogs.oracle.com*. N.p., 2016. Web. 15 Nov. 2016.

"Oracle / PLSQL: Sequences (Autonumber)". *Techonthenet.com*. N.p., 2016. Web. 15 Nov. 2016.

"Oracle / PLSQL: VIEW". *Techonthenet.com*. N.p., 2016. Web. 15 Nov. 2016.

"Oracle PL/SQL: Verify Password - A Function Useful In Password Verification And Maintenance.". *Psoug.org*. N.p., 2016. Web. 15 Nov. 2016.

"SQL Indexes". *www.tutorialspoint.com*. N.p., 2016. Web. 15 Nov. 2016.

sql/oracle, validation. "Validation On Email / Postcode Fields In Sql/Oracle". *Stackoverflow.com*. N.p., 2016. Web. 15 Nov. 2016.

Tan, Bryian. "Regular Expression To Validate File Path And Extension - Codeproject". *Codeproject.com*. N.p., 2016. Web. 15 Nov. 2016.

Views, SQL. "SQL Views". *Sql-plsql.blogspot.ie*. N.p., 2016. Web. 15 Nov. 2016.

"JDBC Callablestatement – Stored Procedure CURSOR Example". Mkyong.com. N.p., 2016. Web. 15 Nov. 2016.