import java.sql.\*;

public class CPTWQuery

{

// JDBC driver name and database URL

//change dbname

static final String DB\_URL = "jdbc:mysql://localhost/Plugins";

// Database credentials

static final String USER = "";

static final String PASS = "";

private static Connection conn = null;

private static Statement statement = null;

private static ResultSet rs = null;

public static void main(String[] args) throws SQLException

{

try

{

// Class.forName(JDBC\_DRIVER); //Register JDBC Driver

openConnection();

/\*\* Begin with 9 select queries\*/

/\*\*select 1

this select query will find developers who make both fx plugins and generator plugins and find

total plugins each of these selected developers create.\*/

selectDeveloperCreateBothPlugins();

/\*\* select 2

this select query will find the type of plugin that is downloaded the most by users\*/

selectPluinsDownloadedMost();

/\*\* select 3

this query will find users who downloaded both fx and generator plugins\*/

selectUserDownloadBothPlugins();

// select 4

averageFx();

// select 5

pluginsAfter2020();

// select 6

vst3Plugins();

// select 7

equalDownloads();

// select 8

notfreeDownload();

// select 9

specializedDevs();

/\*\* Begin with 2 update queries\*/

// update 1

function10();

// update 2

function11();

/\*\* Begin with 2 delete queries\*/

/\*\* delete 1 : work with trigger 2: triggerdeleteDownloadRecord()

delete the uid =7, and plugin = ‘Haas’ and downloadDate=’2021-09-26’ will delete the record

from the downloaded table as well as update the user table for the downloaded times for uid 7

\*/

deleteFromDownloaded();

// delete 2

expiration();

/\*\* Begin with 2 insert queries\*/

/\*\* insert 1: work with trigger 1

for insert into downloaded table where user download new plugins at current time\*/

insertDownloaded();

// insert 2

insertUser() ;

/\*\* begin with 1 procedure\*/

/\*\* stored procedure table archive will store old plugins that its lastUpdate was 2020 and earlier.

this procedure will first insert into archive using data from plugins join fx and plugins join

generator using common attribute plugin name and the given parameter ‘last\_update’ . Then

delete old plugs info from plugins, fx and generator table using common attributes and given

parameters. Archive table also include user who has downloaded that plugins \*\*/

archive();

/\*\* begin with 2 triggers\*/

/\*\* trigger 1:

create trigger validateUserForDownload before insert into downloaded table, need to check

if user is an existing user, and will add 1 to total download times for that user in the user table.\*\*/

triggervalidateUserForDownload();

/\*\* trigger 2:

create trigger validateUserForDownload before insert into downloaded table, need to check

if user is an existing user, and will add 1 to total download times for that user in the user table.\*\*/

triggerdeleteDownloadRecord();

/\*\* begin with 5 key constraints will show errors message \*/

/\*\* key constraint for plugins table\*/

keyConstraintForPlugins();

/\*\* key constraint for Generator table\*/

keyConstraintForGenerator();

/\*\* key constraint for Fx table\*/

keyConstraintForFx();

/\*\* key constraint for User table\*/

keyConstraintForUser();

/\*\* key constraint for Downloaded table\*/

keyConstraintForDownloaded();

/\*\* Foreign key constraint for Downloaded table update uid\*/

foreignkeyConstraintForDownloaded();

/\*\* Foreign key constraint for Downloaded table update plugin\*/

foreignkeyConstraintForDownloaded2();

}

catch(SQLIntegrityConstraintViolationException se) {

se.printStackTrace();

}

catch(SQLException se){

// handle errors for JDBC

se.printStackTrace();

}catch(Exception e){

// handle errors class.forname

e.printStackTrace();

}finally

{

try{ if(statement!=null)

statement.close();

}catch(SQLException se2){

}// nothing we can do

try{

if(conn!=null)

conn.close();

}catch(SQLException se){

se.printStackTrace();

}// end finally

}// end try

System.out.println("Goodbye!");

}//end main

private static void openConnection() throws SQLException

{

// Open a connection

System.out.println("Connecting to database...");

conn = DriverManager.getConnection(DB\_URL+"?serverTimezone=UTC", USER, PASS);

// testing for connection

//STEP 4: Execute a query

System.out.println("Creating a statement...");

statement = conn.createStatement();

rs = statement.executeQuery("select uID, name from User");

//STEP 5: Process the results

while(rs.next()){

System.out.println("User ID="+rs.getInt("uID")+", Name="+rs.getString("name"));

}

}

/\*\*select 1

this select query will find developers who make both fx plugins and generator plugins and find

total plugins each of these selected developers create.\*/

private static void selectDeveloperCreateBothPlugins() throws SQLException

{

rs = statement.executeQuery(" select developer, count(developer) as numOfPlugins from plugins old \r\n"

+ " where type !=any (select type from plugins where developer = old.developer) \r\n"

+ " group by (developer);");

while(rs.next()) {

String developer = rs.getString("developer");

int count = rs.getInt("numOfPlugins");

System.out.println("Deverloper: " + developer + " numOfPlugins: " + count);

}

System.out.println();

}

/\*\* select 2

this select query will find the type of plugin that is downloaded the most by users\*/

private static void selectPluinsDownloadedMost() throws SQLException

{

rs = statement.executeQuery("select type from plugins join (\r\n"

+ " select plugin, count from (\r\n"

+ " select plugin, count(plugin) as count from downloaded group by plugin) as d1 \r\n"

+ " where d1.count >= all (\r\n"

+ " select count(plugin) from downloaded group by plugin))d2 \r\n"

+ " where d2.plugin = plugins.plugin;\r\n"

+ "");

while(rs.next()) {

String type = rs.getString("type");

System.out.println("Plugin: " + type);

}

System.out.println();

}

/\*\* select 3

this query will find users who downloaded both fx and generator plugins\*/

private static void selectUserDownloadBothPlugins() throws SQLException

{

rs = statement.executeQuery("select name from user where uid = (select uID from\r\n"

+ " (select uID, plugin from downloaded natural join plugins\r\n"

+ " where plugin != all (select name from fx))t1 join\r\n"

+ " (select uID, plugin from downloaded natural join plugins \r\n"

+ " where plugin = any (select name from fx))t2 using(uID)); \r\n"

+ "");

while(rs.next()) {

String name = rs.getString("name");

System.out.println("username: " + name);

}

System.out.println();

}

//4. query that selects name of plugins that have been download from Fx and Generator after year 2020

private static void pluginsAfter2020() throws SQLException

{

rs = statement.executeQuery("select distinct generator.name from generator \r\n"

+ " left outer join downloaded on generator.name = downloaded.plugin \r\n"

+ " where generator.lastUpdate > '2019-12-31' and downloaded.downloadDate > '2019-12-31' \r\n"

+ " union all \r\n"

+ " select distinct fx.name from fx \r\n"

+ " right outer join downloaded on fx.name = downloaded.plugin \r\n"

+ " where fx.lastUpdate > '2019-12-31' and downloaded.downloadDate > '2019-12-31';");

System.out.println("Plugin Name downloaded after 2020:");

while(rs.next())

{

String after20 = rs.getString("name");

System.out.println(after20);

}

}

//5. query selects number of plugins in VST3 format

private static void vst3Plugins() throws SQLException

{

rs = statement.executeQuery("select count(distinct generator.name) as genCount, count(distinct fx.name) as fxCount \r\n"

+ " from generator, FX \r\n"

+ " where generator.format = 'VST3' and fx.format = 'VST3';");

System.out.println("Count of Generators and FX Plugins in VST3 format:");

while(rs.next())

{

String gen = rs.getString("genCount");

String fx = rs.getString("fxCount");

System.out.println("Generators: " + gen);

System.out.println("FX: " + fx);

}

}

//6. this query will find the average number of plugins in fx table that are free

private static void averageFx() throws SQLException

{

rs = statement.executeQuery("select avg(stcounts.ct) \r\n"

+ " from(select count(subtype) as ct \r\n"

+ " from fx \r\n"

+ " where fx.isFree = 1 \r\n"

+ " group by subtype \r\n"

+ " having count(subtype) > 1 \r\n"

+ " ) as stcounts; \r\n"

+ "");

while(rs.next())

{

String average = rs.getString("avg(stcounts.ct)");

System.out.println("Average Plugins: " + average);

}

System.out.println();

}

//7. query selecting all users with an equal number of downloads

private static void equalDownloads() throws SQLException

{

rs = statement.executeQuery("select distinct u1.name as User1, u2.name as User2 \r\n"

+ " from User u1, User u2 \r\n "

+ " where u1.name < u2.name and u1.downLoaded = u2.downLoaded \r\n"

+ " group by u1.uID;");

while(rs.next())

{

String u1 = rs.getString("User1");

String u2 = rs.getString("User2");

System.out.println("Users: " + u1 + ", " + u2);

}

}

//8. find developers made only one type of plugins and not the other and the total the

// plugins that developer made

private static void specializedDevs() throws SQLException

{

rs = statement.executeQuery("select developer, type, count(developer) as numOfPlugins \r\n"

+ " from plugins where developer != any (select developer from plugins old where type != any \r\n"

+ " (select type from plugins where old.developer = developer) group by developer) group by developer;");

System.out.println("Developer: Type: Number of Plugins");

System.out.println("Number of Plugins:");

while(rs.next())

{

String dev = rs.getString("developer");

String typ = rs.getString("type");

String count = rs.getString("numofPlugins");

System.out.println(dev + " " + typ + "Count: " + count);

}

}

//9. query selects user whose download is not free but has license

private static void notfreeDownload() throws SQLException

{

rs = statement.executeQuery("select user.name from (select uid from (select \* from downloaded left outer join generator on \r\n"

+ " plugin = name union select \* from downloaded left outer join fx on plugin = name) t where "

+ " t.name is not null and isFree = 0 and hasLicense = 1 group by(uID)) s natural join user;");

System.out.println("Licensed Users with non free download: ");

while(rs.next())

{

String user = rs.getString("user.name");

System.out.println(user);

}

}

/\*\* delete 1 : work with trigger 2

delete the uid =7, and plugin = ‘Haas’ and downloadDate=’2021-09-26’ will delete the record

from the downloaded table as well as update the user table for the downloaded times for uid 7

\*/

private static void deleteFromDownloaded() throws SQLException

{

statement.executeUpdate("delete from downloaded where uid = 7 and plugin ='Haas' and downloadDate='2021-09-26'");

rs = statement.executeQuery("select \* from downloaded");

while(rs.next()) {

int uID = rs.getInt("uID");

String plugin = rs.getString("plugin");

String downloadDate = rs.getString("downloadDate");

System.out.println("username: " + uID + " plugin: " + plugin + " downloadDate: " + downloadDate);

}

System.out.println();

}

// delete 2 // expiration after 4 months

//delete plugins that have not been updated in 4 months

private static void expiration() throws SQLException

{

statement.executeUpdate("delete from fx where now() > DATE\_ADD(fx.lastUpdate, INTERVAL 4 MONTH)");

}

/\*\* insert 1: work with trigger 1

for insert into downloaded table where user download new plugins at current time\*/

private static void insertDownloaded() throws SQLException

{

statement.executeUpdate("insert into downloaded values(1, 'Haas', now())");

}

// insert 2: insert a new user

private static void insertUser() throws SQLException

{

statement.executeUpdate("insert into user(name,age,downloaded) values('Anna',20,0)");

}

/\*\* stored procedure table archive will store old plugins that its lastUpdate was 2020 and earlier.

this procedure will first insert into archive using data from plugins join fx and plugins join

generator using common attribute plugin name and the given parameter ‘last\_update’ . Then

delete old plugs info from plugins, fx and generator table using common attributes and given

parameters. Archive table also include user who has downloaded that plugins \*\*/

private static void archive() throws SQLException

{

String queryDrop = "drop procedure if exists archive;";

Statement drop = conn.createStatement();

drop.execute(queryDrop);

String createProcedure = "create procedure archive(IN last\_update timestamp) "

+ "begin "

+ "insert into archive (name, format, developer, type, subtype, isFree, hasLicense, lastUpdate, uID, downloadDate) select name, format, developer, type, subtype, isFree, hasLicense, lastUpdate, uID, downloadDate from plugins join fx on fx.name = plugins.plugin and fx.lastUpdate < last\_update left outer join downloaded on downloaded.plugin = plugins.plugin;"

+ "insert into archive (name, format, developer, type, subtype, isFree, hasLicense, lastUpdate, uID, downloadDate) select name, format, developer, type, subtype, isFree, hasLicense, lastUpdate, uID, downloadDate from plugins join generator on generator.name = plugins.plugin and generator.lastUpdate < last\_update left outer join downloaded on downloaded.plugin = plugins.plugin;"

+ "delete from plugins where plugins.plugin in (select name from fx where lastUpdate < last\_update);"

+ "delete from plugins where plugins.plugin in (select name from generator where lastUpdate < last\_update);"

+ "delete from fx where lastUpdate < last\_update;"

+ "delete from generator where lastUpdate < last\_update;"

+ "end";

statement.executeUpdate(createProcedure);

boolean hasResults = statement.execute("{CALL archive('2020-01-01')}");

rs = statement.executeQuery("Select \* from archive");

while(rs.next()) {

String name = rs.getString("name");

String format = rs.getString("format");

String developer = rs.getString("developer");

String type = rs.getString("type");

String subtype = rs.getString("subtype");

boolean isFree = rs.getBoolean("isFree");

boolean hasLicense = rs.getBoolean("hasLicense");

String lastUpdate = rs.getString("name");

int uID = rs.getInt("uID");

String downloadDate = rs.getString("downloadDate");

System.out.println("Plugins: " + name + " Format: " + format + "Developer: " + developer

+ "type: " + type + "subtype: " + subtype + "isFree: " + isFree + "hasLicense: " + hasLicense

+ "lastUpdate: " + lastUpdate + " uID: " + uID + " DownloadDate: " + downloadDate );

}

System.out.println();

}

/\*\* trigger 1:

create trigger validateUserForDownload before insert into downloaded table, need to check

if user is an existing user, and will add 1 to total download times for that user in the user table.\*\*/

public static void triggervalidateUserForDownload() throws SQLException

{

String queryDrop = "drop trigger if exists validateUserForDownload";

Statement drop = conn.createStatement();

drop.execute(queryDrop);

statement.execute("create trigger validateUserForDownload\r\n"

+ "before insert on downloaded\r\n"

+ "for each row\r\n"

+ "if new.uID in (select uID from user) \r\n"

+ "then update User set downLoaded = downLoaded +1 where uID = new.uID;\r\n"

+ "end if");

}

/\*\* trigger 2:

delete record from downloaded table, the user table downloaded plugin’s time will be reduced

as well, this trigger will not check delete record input for validation. So it is allow delete

multiple record, as long as the value in the where clause exist in the downloaded table.\*\*/

public static void triggerdeleteDownloadRecord() throws SQLException

{

String queryDrop = "drop trigger if exists deleteDownloadRecord";

Statement drop = conn.createStatement();

drop.execute(queryDrop);

statement.execute("create trigger deleteDownloadRecord\r\n"

+ "before delete on downloaded \r\n"

+ "for each row\r\n"

+ "begin\r\n"

+ "update User set downloaded = downloaded -1 where uid = old.uid;\r\n"

+ "end");

}

/\*\* key constraint for plugins table\*/

public static void keyConstraintForPlugins() throws SQLException

{

statement.executeUpdate("insert into plugins values(\"Chorus\", \"Slate Digital\", \"Generator\");");

}

/\*\* key constraint for Generator table\*/

public static void keyConstraintForGenerator() throws SQLException

{

statement.executeUpdate("insert into generator values(\"Vital\", \"VST2\", \"softsynth\", 1,0,now())");

}

/\*\* key constraint for Fx table\*/

public static void keyConstraintForFx() throws SQLException

{

statement.executeUpdate("insert into fx values(\"Delay\", \"VST2\", \"softsynth\", 1,0,now());");

}

/\*\* key constraint for User table\*/

public static void keyConstraintForUser() throws SQLException

{

statement.executeUpdate("insert into user values(1, \"jason\", 20, 0);");

}

/\*\* key constraint for Downloaded table\*/

public static void keyConstraintForDownloaded() throws SQLException

{

statement.executeUpdate("insert into downloaded values(2, \"Gate\", \"2021-10-02\");");

}

/\*\* Foreign key constraint for Downloaded table update uid\*/

public static void foreignkeyConstraintForDownloaded() throws SQLException{

statement.executeUpdate(" update downloaded set uid = 8 where uid =1;");

}

/\*\* Foreign key constraint for Downloaded table update plugin\*/

public static void foreignkeyConstraintForDownloaded2() throws SQLException{

statement.executeUpdate(" update downloaded set plugin = \"Pro\" where plugin = \"Haas\";");

}

}//end CPTWQuery