Intermediate Report

CPTW Software Group

Faculty Sponsor:

Suneuy Kim - San Jose State University Professor

San Jose State University - College of Engineering

Principal Investigators:

Diego Cruz

Jared Park

Andrew Tedijanto

Shuti Wang

1 - Table of Contents

2 - Abstract

Our project is a representation of plugin management for audio processing software commonly used in audio engineering applications and music production. The data set is inspired by the current software inside of the King Library 4th Floor Sound Studio managed by Student Computing Services. The data set structure is partially inspired by the plugin databases that exist for Digital Audio Workstation softwares such as Image-Line FL Studio and Cockos REAPER. The group sought to improve upon these already-established plugin databases by introducing new attributes as well as introducing a modified structure expressed through tables.

This intermediate report will provide our current stage of designing a Java based software for retrieving data about these plugins from an SQL database, including the design of our database schema, the queries that our program currently offers, and the background behind our project. Further testing and design of our Java program should yield a finished software project by the time of our final report.

3- Background

The design of this project is in the context of a desktop machine geared for audio production that exists on the 4th floor of the King Library, inside of the sound studio. There exists a clear and present need to manage the plugins (and their relevant licensing) inside of the sound studio for internal use among the Student Computing Services and IT employees.

Plugins are software tools that are used in the audio production process. The most common formats of plugins are VST/VST3 (Virtual Studio Technology- Windows and Mac), AU (Audio Unit - Mac Only), and AAX (Pro Tools). They are commonly divided into effects and generators, which then have several different subtypes. There exist both free and paid plugins, as well as plugins that do not require a license to use and those that do. Interestingly some free plugins do require a license to be able to use them, and just about every paid plugin requires a license to fully use the plugin.

FX plugins take an already-generated sound and manipulate it using various methods. For instance, an equalizer can influence the overall sound by applying boosts/cuts/ dynamic modulations with certain frequencies, a reverb can give a sound the impression that it is playing in a certain acoustic environment, and a delay can give a sound the impression that it is playing in a space where reverberations can bounce back and give echo feedbacks. There are many different types of effects, but the aforementioned are some of the most commonly used effects.

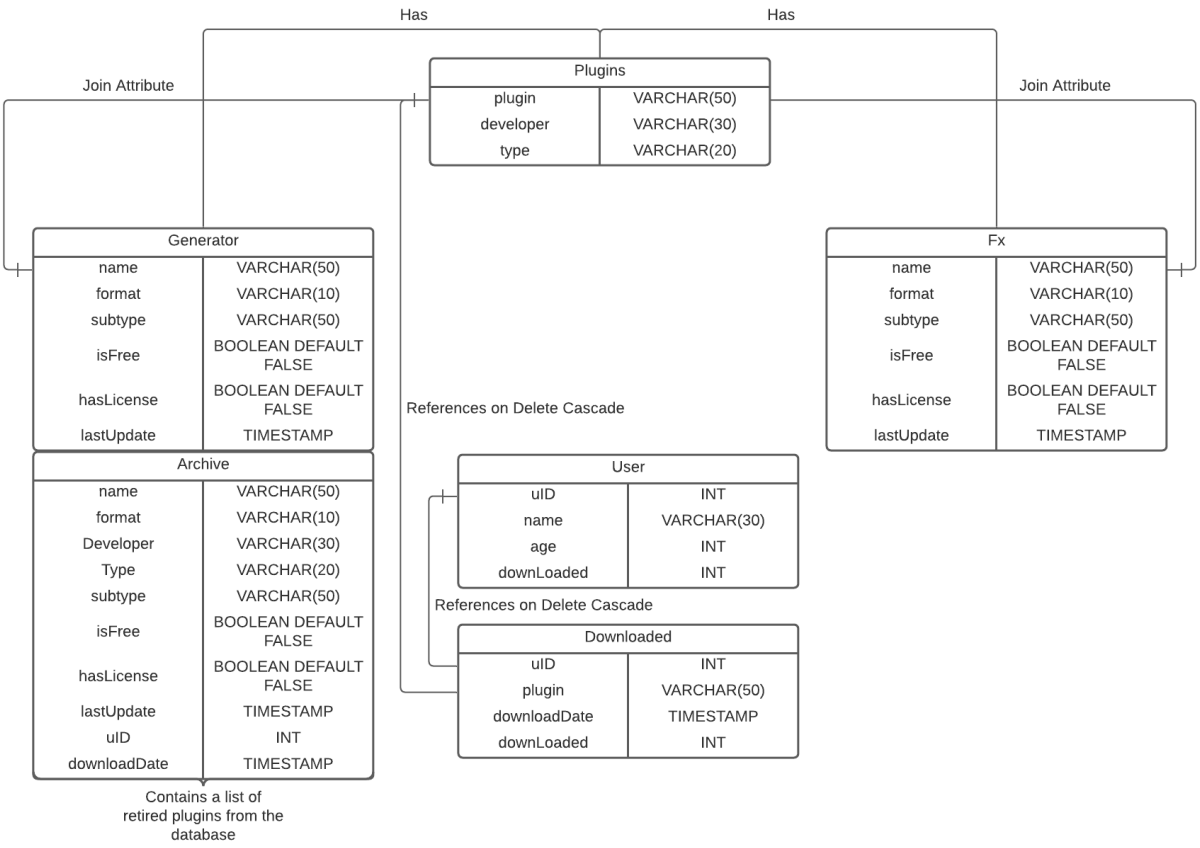
Generator plugins are able to generate a sound, often resampling them or synthesizing them. There are two common types of generators that cover 97% of all generator plugins that exist. There exists ROMplers, which use a multisample library of a sound, and add parameters where you can add some effects onto them. These do not generate the sound from scratch, so there is

little that can be done to change the fundamentals of the sound. On the other hand, softsynths are synthesizer plugins that can generate a sound from scratch, often using basic wave shapes, wavetables, or a few other methods to create a sound from the ground up.

4 - Complete Database Schema

Here is a visual representation of the Database Schema:

Figure 4.0.1 - Entity Relationship Diagram



Here is the related mySQL Schema:

==========

```
DROP DATABASE IF EXISTS PLUGINS;
CREATE DATABASE PLUGINS;
USE PLUGINS;

DROP TABLE IF EXISTS Plugins;
CREATE TABLE Plugins(
```

```
plugin VARCHAR(50), # plugin uses as join attributes(name) in generator and fx table
developer VARCHAR(30),
type VARCHAR(20), # fx or generator
PRIMARY KEY(plugin)
);

DROP TABLE IF EXISTS Generator;
CREATE TABLE Generator(
name VARCHAR(50) PRIMARY KEY, # name of the plugin
format VARCHAR(10),
subtype VARCHAR(50),
isFree BOOLEAN DEFAULT FALSE,
hasLicense BOOLEAN DEFAULT FALSE,
lastUpdate timestamp not null on update current_timestamp  # 0000-00-00 00:00:00
);

DROP TABLE IF EXISTS Fx;
CREATE TABLE Fx(
name VARCHAR(50) PRIMARY KEY,  # name of the plugin
format VARCHAR(10),
subtype VARCHAR(50),
isFree BOOLEAN DEFAULT FALSE,
hasLicense BOOLEAN DEFAULT FALSE,
lastUpdate  timestamp not null on update current_timestamp
);


DROP TABLE IF EXISTS User;
CREATE TABLE User(
uID INT AUTO_INCREMENT,
name VARCHAR(30),
age INT,
downLoaded INT,
PRIMARY KEY(uID)
);

DROP TABLE IF EXISTS Downloaded;
CREATE TABLE Downloaded(
uID INT,
```

```
plugin VARCHAR(50),
downloadDate timestamp not null on update current_timestamp,
PRIMARY KEY(uID, plugin, downloadDate),
FOREIGN KEY(uID) references User (uID) on delete cascade,
FOREIGN KEY(plugin) references Plugins (plugin) on delete cascade
);

DROP TABLE IF EXISTS Archive;
CREATE TABLE Archive(
name VARCHAR(50) primary key,  # name of the plugin
format VARCHAR(10),
developer VARCHAR(30),
type VARCHAR(20), # fx or generator
subtype VARCHAR(50),
isFree BOOLEAN DEFAULT FALSE,
hasLicense BOOLEAN DEFAULT FALSE,
lastUpdate  timestamp not null on update current_timestamp,
uID INT,
downloadDate timestamp on update current_timestamp
);
==========
```

4.1 - Constraints

Several constraints have been implemented into the Plugin Database to manage the number of entries/tuples in the database, establish connections between schemas and their data for specific modifications, and prevent irregular data from being modified in the database. An example of irregular modifications includes the fact that by design, plugins seldom change names, exchange developers, or completely morph into a different type/subtype of plugin. The furthest that a plugin can change is in the formats it is offered in (VST/AAX/AU/RACK), and whether a new version is released (in which the new one is installed as another, discrete plugin e.g. FabFilter Pro-Q 2 vs. FabFilter Pro-Q 3).

The constraints stored in the database include: primary and foreign keys, cascading attributes, and triggers. Each table contains a primary key in the form of a name or plugin name attribute, indicating that these attributes must be unique and not null. Various keys are referenced as well, as foreign keys in other tables like the Downloaded table. The primary key, uID in the User table, is referenced to the uID in the Downloaded table as a foreign key. Additionally, the lastUpdate attribute, which is found in most of the tables, is constantly changed to the current time whenever an update is executed on the table.

There are currently 2 triggers implemented into the database: validateUserForDownload and deleteDownloadRecord. The validateUserForDownload trigger activates before the insertion of the Downloaded table, checking if the User's uID is a valid identification, thereby allowing them to download a plugin and increment their number of downloads by 1. The deleteDownloadRecord trigger handles deletions in the Downloaded table. Before a deletion can be executed, the number of downloads of the respective User is decremented.

5 - Requirements

The following are the requirements of our Java program:

5.1 - Functional Requirements

Functional Requirement No.	Functional Requirement Description
1	Load data from SQL database
2	Take user input in text boxes
4	User inputs are analyzed by the program
5	Data from the SQL database is retrieved
6	Analyzing the inputs returns an output through the form of a table
7	User can quit the program

6 - Results

6.1 - Associated SQL Programs

Please highlight in bold the five significantly different types of SQL (one correlated subquery, group by and having, aggregation, outer join, and mathematical set operation)

- List all SQL select statements: 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.
    select developer, count(developer) as numOfPlugins from plugins old
    where type !=any (select type from plugins where developer = old.developer)
    group by (developer);
```

```
// select 2
// this select query will find the type of plugin that is downloaded the most by users
      select type from plugins join (
      select plugin, count from (
      select plugin, count(plugin) as count from downloaded group by plugin) as d1
      where d1.count >= all (
      select count(plugin) from downloaded group by plugin))d2
      where d2.plugin = plugins.plugin;

// select 3
// this query will find user who downloaded both fx and generator plugins
      select name from user where uid = (select uID from
      (select uID, plugin from downloaded natural join plugins
      where plugin != all (select name from fx)) t1 join
      (select uID, plugin from downloaded natural join plugins
      where plugin = any (select name from fx)) t2 using(uID));

//select 4
select the names of the plugins that have been downloaded
from Fx and Generator after the year 2020

select distinct generator.name from generator
left outer join downloaded on generator.name = downloaded.plugin
where generator.lastUpdate > '2019-12-31' and downloaded.downloadDate > '2019-12-31'
union all
select distinct fx.name from fx
right outer join downloaded on fx.name = downloaded.plugin
where fx.lastUpdate > '2019-12-31' and downloaded.downloadDate > '2019-12-31';

//select 5
//select number of generator plugins and fx plugins in VST3 format
select count(distinct generator.name) as genCount, count(distinct fx.name) as fxCount
from generator, FX
where generator.format = 'VST3' and fx.format = 'VST3';

//select 6
//select average number of plugins in fx table that are free
select avg(stcounts.ct)
from(
select count(subtype) as ct
from fx
where fx.isFree = 1
```

```
group by subtype
having count(subtype) > 1
) as stcounts;

//select 7

//select all users that have the same number of downloads
select distinct u1.name as User1, u2.name as User2
from User u1, User u2
where u1.name < u2.name and u1.downLoaded = u2.downLoaded
group by u1.uID;


//select 8  find developers made only one type of plugins and not the other and the total the
//plugins that developer made
select developer, type, count(developer) as numOfPlugins from plugins where developer != any
(select developer from plugins old where type != any (
select type from plugins where old.developer = developer) group by developer) group by
developer;

// select 9 : select user whose download is not free but has license
select user.name from (select uid from (select * from downloaded left outer join generator on
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;
```

- List all SQL update statements: 2 update queries

```
// update 1: update the plugin 'Delay Lama' with the current time.
update generator set lastUpdate = now() where name = 'Delay Lama';

// update 2 : update generator by inverting the isFree variable for plugins with the subtype
'ROMpler'

update generator set isFree = not isFree where subtype = 'ROMpler';
```

- List all SQL delete statements: 2 delete queries

```
// 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
delete from downloaded where uid = 7 and plugin ='Haas' and downloadDate='2021-09-26';

// delete 2 // expiration after 4 months
```

```
//delete plugins that have not been updated in 4 months
delete from fx
where NOW() > DATE_ADD(fx.lastUpdate, INTERVAL 4 MONTH);
```

- List all SQL insert statements: 2 insert queries

```
// insert 1: work with trigger 1
// for insert into downloaded table where user download new plugins at current time
insert into downloaded values(1, 'Haas', now());

// insert 2: insert a new user
 insert into user(name, age, downloaded) values('Anna', 20, 0);
```

6.1a - Stored Procedures

```
// 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

drop procedure if exists archive;
delimiter //
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 //
delimiter ;
```

```
call archive('2020-01-01');
```

6.1b - Triggers : 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.
delimiter //
drop trigger if exists validateUserForDownload;
create trigger validateUserForDownload
before insert on downloaded
for each row
if new.uID in (select uID from user)
then update User set downLoaded = downLoaded +1 where uID = new.uID;
end if;//
delimiter ;

// 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
delimiter //
drop trigger if exists deleteDownloadRecord;
create trigger deleteDownloadRecord
before delete on downloaded
for each row
begin
update User set downloaded = downloaded -1 where uid = old.uid;
end;//
delimiter ;
```

6.2 - Screenshots

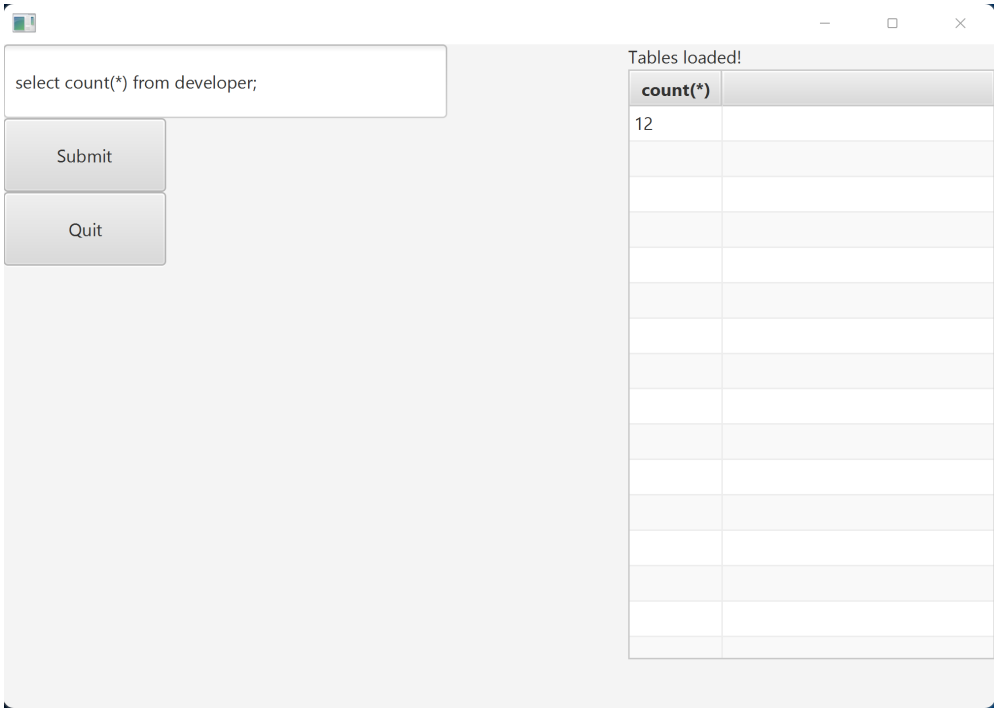


Figure 1: Number of developers in database

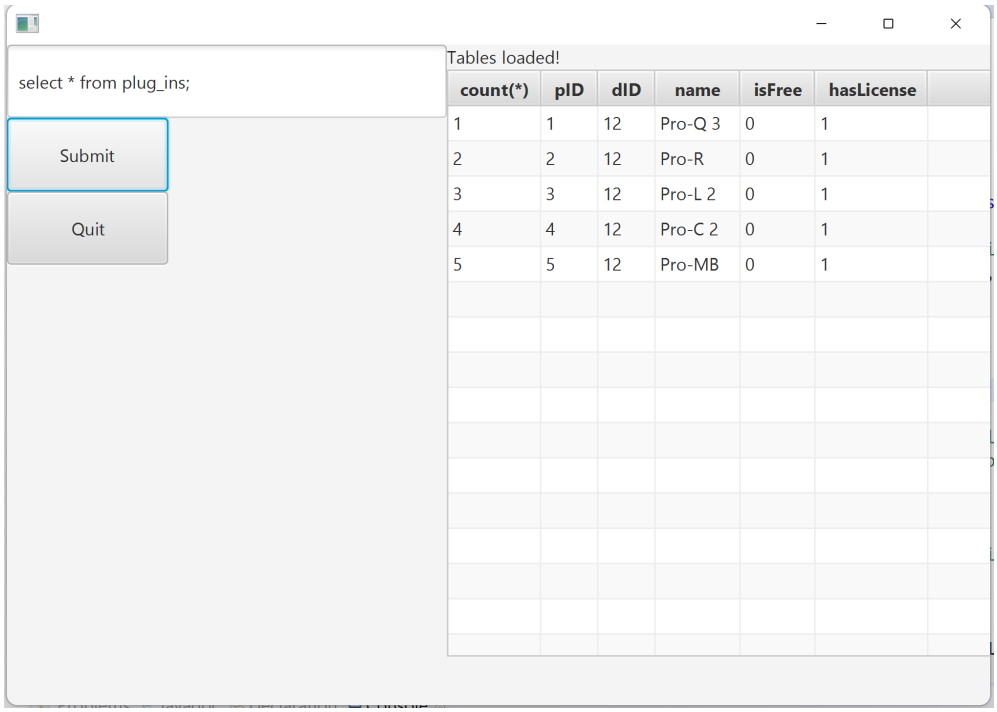


Figure 2: List of plugins in database

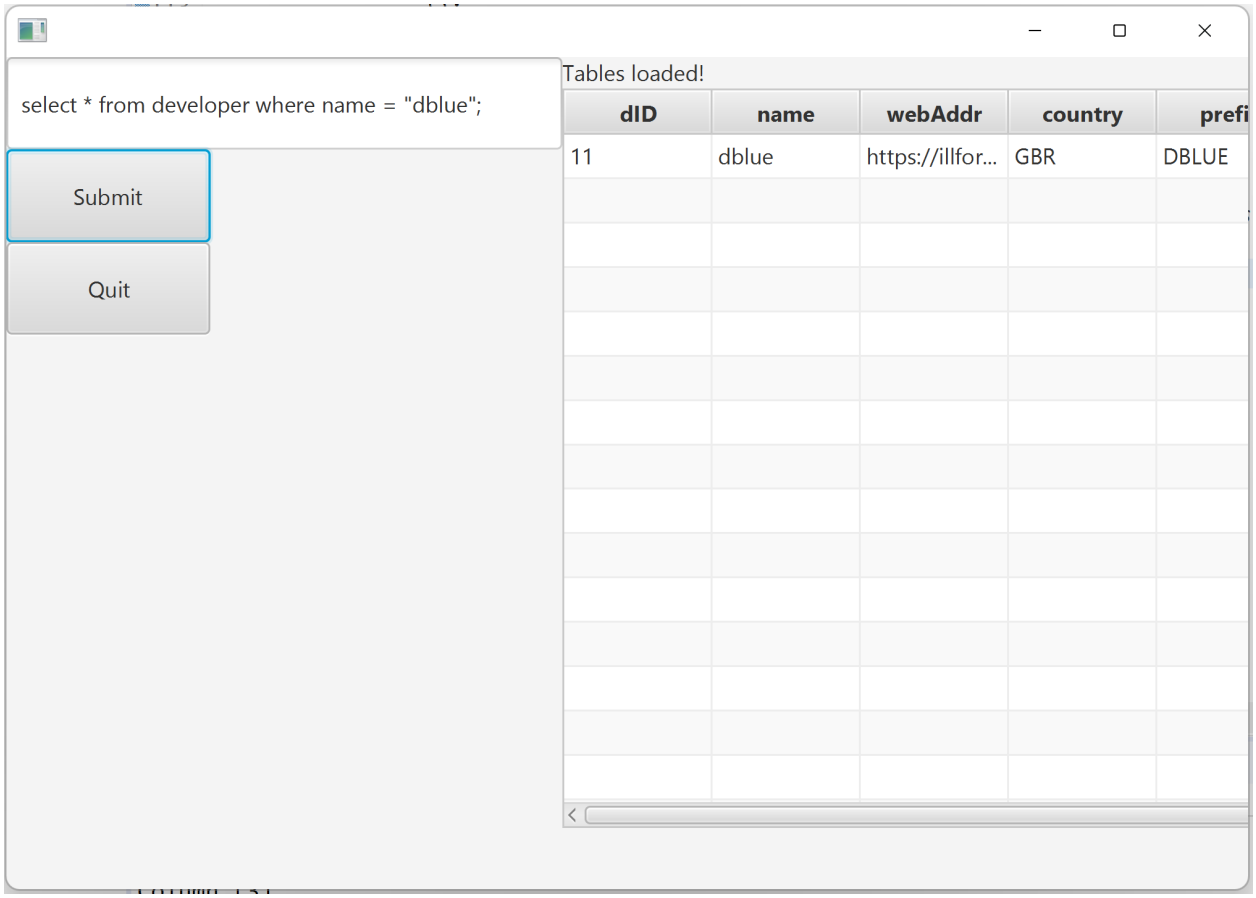


Figure 3: Searching for a row in Developer table