Database Difference Checker (DBC)

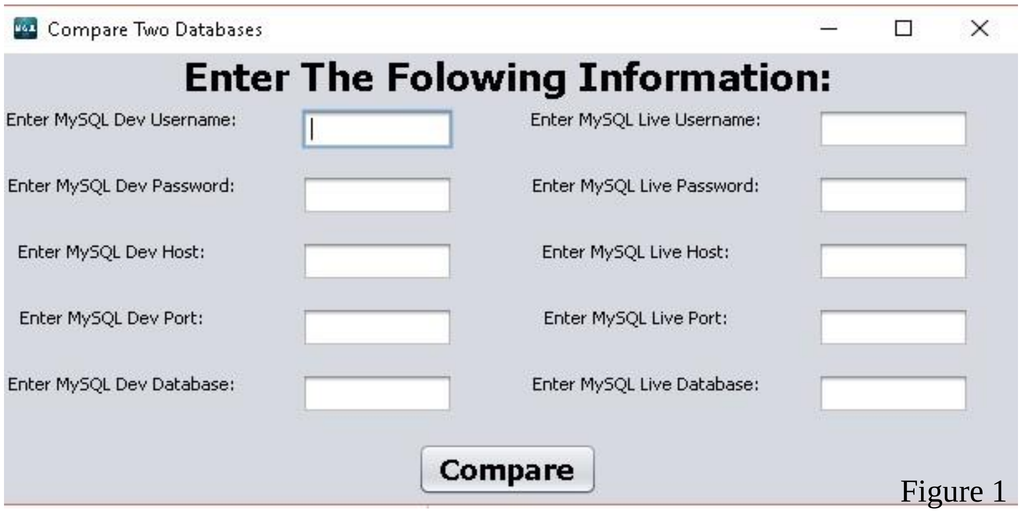
By Peter Kaufman

Abstract

Many organizations collect data for later use. This data can be stored in many different places, one of which is a MySQL database. There is often one that is used for a development (dev) environment and one for a live environment. As time goes on, the dev environment and database change. As a result, the live database needs to be updated for the environment to run the same as the dev environment. In order to make the live database the same as the dev database, DBC was created to compare the two databases and determine the SQL statements to make them the same. The database comparison works with column, table, view, and index differences. The DBC can also work with updating a database when the dev database cannot be connected to. This is done by taking a database (DB) snapshot, which stores the structure of a database schema in a serialized file (Aaron).

Introduction

The DBC consists of fifteen Java classes and JFrames and determines which columns, indices, and tables need to be added and dropped. The DBC can compare two databases using the MySQL username, password, host, port, and database name (Figure 1); take a DB snapshot (Figure 2); compare a database to a DB snapshot (Aaron)(Figure 3); catches and displays errors that occur (Figure 4); display the SQL statement(s) that was/were run last time the DBC ran (Figure 5); display the DBC run log (Figure 6); and display the DBC error log(Figure 7).

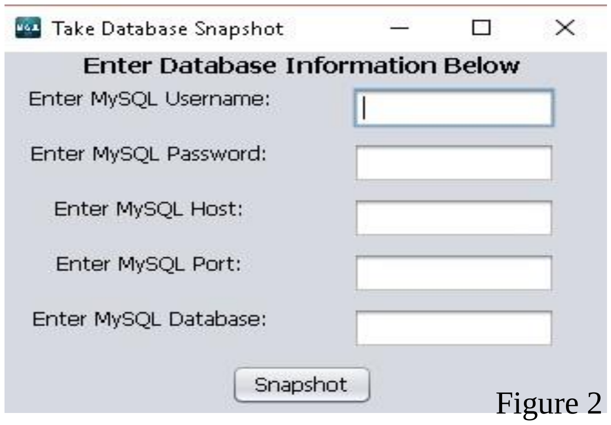


Compare Two Databases

Enter The Folowing Information:

Enter MySQL Dev Username:		Enter MySQL Live Username:	
Enter MySQL Dev Password:		Enter MySQL Live Password:	
Enter MySQL Dev Host:		Enter MySQL Live Host:	
Enter MySQL Dev Port:		Enter MySQL Live Port:	
Enter MySQL Dev Database:		Enter MySQL Live Database:	

Figure 1



Take Database Snapshot

Enter Database Information Below

Enter MySQL Username:

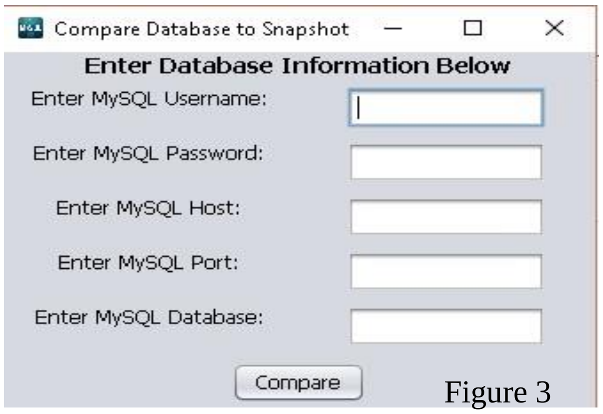
Enter MySQL Password:

Enter MySQL Host:

Enter MySQL Port:

Enter MySQL Database:

Figure 2



Compare Database to Snapshot

Enter Database Information Below

Enter MySQL Username:

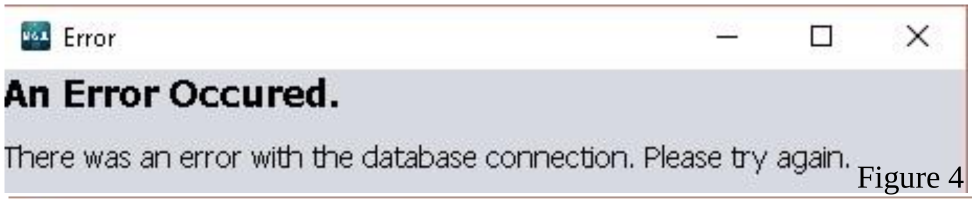
Enter MySQL Password:

Enter MySQL Host:

Enter MySQL Port:

Enter MySQL Database:

Figure 3

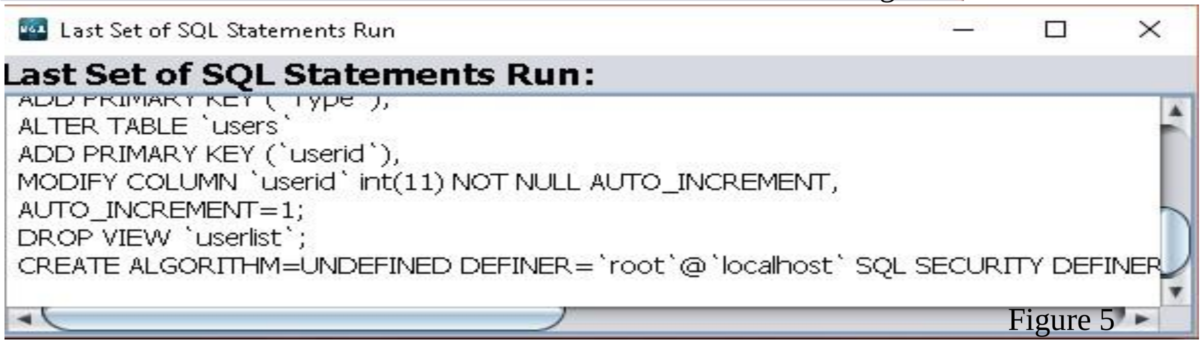


Error

An Error Occured.

There was an error with the database connection. Please try again.

Figure 4

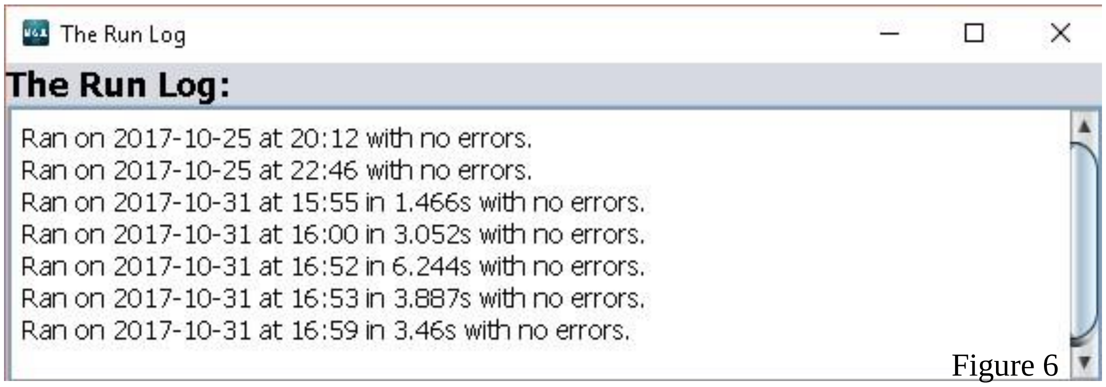


Last Set of SQL Statements Run

Last Set of SQL Statements Run:

```
ADD PRIMARY KEY (`type`),
ALTER TABLE `users`
ADD PRIMARY KEY (`userid`),
MODIFY COLUMN `userid` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=1;
DROP VIEW `userlist`;
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER
```

Figure 5

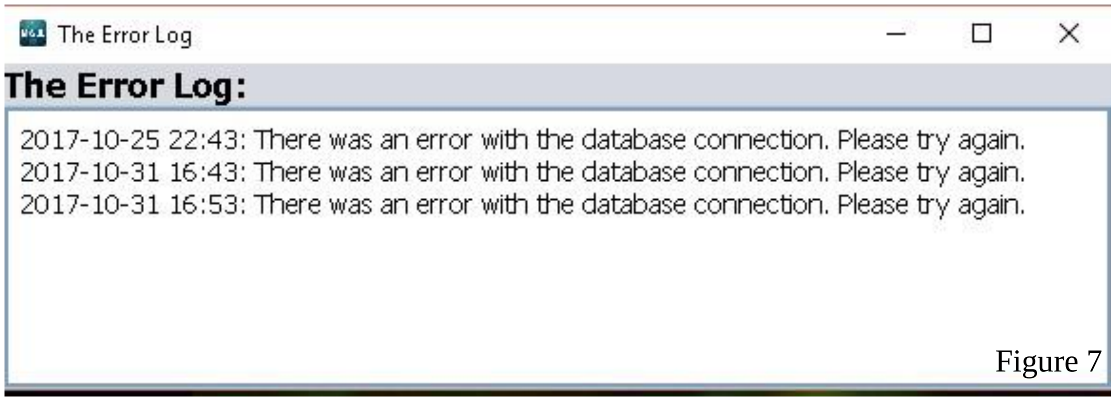


The Run Log

The Run Log:

```
Ran on 2017-10-25 at 20:12 with no errors.
Ran on 2017-10-25 at 22:46 with no errors.
Ran on 2017-10-31 at 15:55 in 1.466s with no errors.
Ran on 2017-10-31 at 16:00 in 3.052s with no errors.
Ran on 2017-10-31 at 16:52 in 6.244s with no errors.
Ran on 2017-10-31 at 16:53 in 3.887s with no errors.
Ran on 2017-10-31 at 16:59 in 3.46s with no errors.
```

Figure 6



The Error Log

The Error Log:

```
2017-10-25 22:43: There was an error with the database connection. Please try again.
2017-10-31 16:43: There was an error with the database connection. Please try again.
2017-10-31 16:53: There was an error with the database connection. Please try again.
```

Figure 7

Methodology

The DBC consist of fifteen Java classes, uses the JDBC connector library, and uses a serialized file. The DBC uses multiple JFrames to allow the JFrames to be closed and opened when needed (The Use of Multiple JFrames: Good or Bad Practice?). This allows for a more user friendly experience. When a user chooses to compare two databases using two database connections, a new window pops for the user to input information to connect to the dev and live databases (Figure 1). Once the user inputs the necessary information, a database connection is made for each of the databases. For each database, the connection is tested. If the connection is unable to be established, then an error window will pop up and inform the user (Figure 4). If the connection is established, a query is run to collect all column, table, view, and index information from the database. Collecting the index, table, view, and column information in one shot makes the program faster because there is no need to run several smaller queries to do what one larger query can do in one shot. Next, all of the live database's primary keys are dropped and all auto increment columns are changed to remove the auto increment from the column. After that, all of dev database's primary keys and auto increment columns are added. If these two steps are not done, the SQL statement(s) will not run without error if the dev and live have large differences in their database schemas. Once the Database objects have been initialized, the two Database objects are compared. First off, an ArrayList of Table objects from each Database object is compared to look for missing and extra tables. If any extra or missing tables are found, the appropriate SQL statement(s) is/are generated to make the table list the same. These tables are added to a list of tables to exclude from the rest of the comparisons. After the exclusion list has been made, the live database's primary key and auto increment drop and modification statements are added to the SQL list. Third, a list of tables, which are to be updated, are compiled based on whether or not the create statements of a table with the same name in each database are the same. If any difference in the create statements is found, then the table name is added to the list. Fourth, this set of tables is then used to update these tables. The table information is compared column by column and index by index. If any are found to be extra or missing, the appropriate SQL statements are generated. For each table, all of the SQL statements needed to make the table the same in the live and dev database is put into one SQL statement to speed up the running the SQL statements. Fifth, the dev database's primary key and auto increment add and modification statements are added to the SQL list if the table name of where the column(s) and the primary key is not found in the exclusion list. Last, all of the views in the live database have their drop statements generated, and all the views in the dev database have their create statements generated. After each of these four comparisons, the SQL statements are added to a "master list." This list is displayed in a new JFrame where the user can copy the code in order to run it elsewhere or run it from the GUI application itself. When the user clicks the button to run the SQL statements, the "master list" is written to a file for later use. It will show the progress of the database update as it runs the SQL statements. If the user chooses to do a database comparison with a DB snapshot, the process is the same except that a serialized file must be converted back into a Database object before the comparisons occur. Furthermore, if the user chooses to take a DB snapshot, a Database object is initialized, and then it is converted to a serialized file, which can be converted back to a Database object as desired later. If an error occurs at any time, a new JFrame will appear with an error message related to why the error occurred. This error is written to an external file for later use. If the user would like to see the SQL that was used to update the database last; check when a user last ran the DBC; or check to see any error that occurred, the appropriate information can be displayed by reading text from the appropriate file. Listed below are the most important methods of the DBC.

Class	Method	Description
DB_Diff_Checker_GUI	continueBtnMouseClicked	Determines which method the user has selected and opens the appropriate JFrame
DB_Diff_Checker_GUI	displayLog	Opens a JFrame with either the last set of SQL statements run or the run log depending on what file name is passed to it
DB_Diff_Checker_GUI/DBCompare/Error/Result	initComponents	Sets up the GUI Layout, sets up all action events, and initializes instance variables
DBCompare	compareDatabases	Compares two databases and determines their differences and how to make them the same
DBCompare	databaseConnection1btnActionPerformed	Determines if the user has put in the appropriate information and either takes a database snapshot or compares a two databases (one can be a snapshot)
DBCompare	getSequelStatementsInBackground	Compares two databases based on user input (one can be a snapshot) in the background to keep from freezing up the GUI

DBCompare	takeSnapshot	Takes a snapshot of a database by converting a Database object to a serialized file
JFrameV2	displayResult	Opens a JFrame with the result of the comparison
JFrameV2	error	Opens a new JFrame which displays the error that occurred
Database	tablesDiffs	Updates the list of tables which are not the same in dev
Table	equals	Takes in a Table and compares it to the current one, the result is the SQL statements to be run to make the two tables the same
FileConversion	serializeDatabase	Turns a Database object into a serialized file
FileConversion	deserailizDatabase	Turns a serialized file into a Database object/Returns an ArrayList Strings which were read from a file
FileConversion	writeToFile	Write to either the logs file or the last run file based on the type of input (string or ArrayList)
FileConversion	fileExists	Takes a file path and determines whether it exists or not
Db_conn	getTableList	Gets the tables, columns, and indices of the db
Db_conn	getViews	Gets the views of the db
Db_conn	runSequelStatement	Takes a SQL statement/statement list and runs it

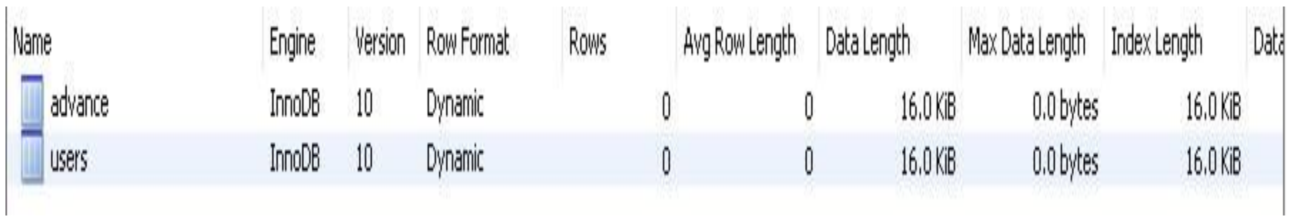
Result & Discussion

Say there are two databases one called live and one called dev as shown below (Dev, Live). If the DBC is run to make the live database the same as the dev database, the result is shown below (DBC Result). When run, these SQL statements make the live database the same as the dev database. After running the above code, the result of running the DBC again is shown below (DBC Result2).

The views are the only ones that show up because any views from the live database are automatically dropped and added regardless of whether or not they are different. The primary keys and auto increment columns are drooped/added and modified regardless of whether or not they are different.

Dev:

Tables



Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length	Data
advance	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	16.0 KiB	
users	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	16.0 KiB	

Columns

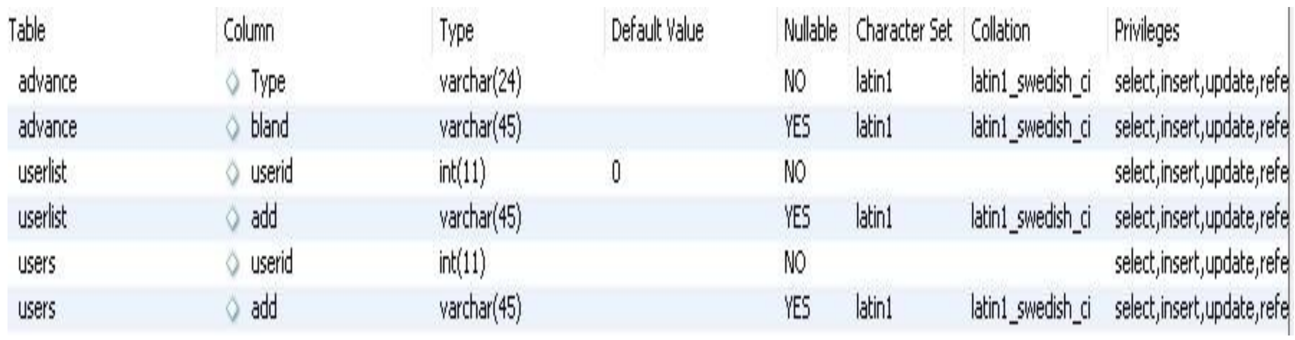


Table	Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
advance	Type	varchar(24)		NO	latin1	latin1_swedish_ci	select,insert,update,refe
advance	bland	varchar(45)		YES	latin1	latin1_swedish_ci	select,insert,update,refe
userlist	userid	int(11)	0	NO			select,insert,update,refe
userlist	add	varchar(45)		YES	latin1	latin1_swedish_ci	select,insert,update,refe
users	userid	int(11)		NO			select,insert,update,refe
users	add	varchar(45)		YES	latin1	latin1_swedish_ci	select,insert,update,refe

Indices

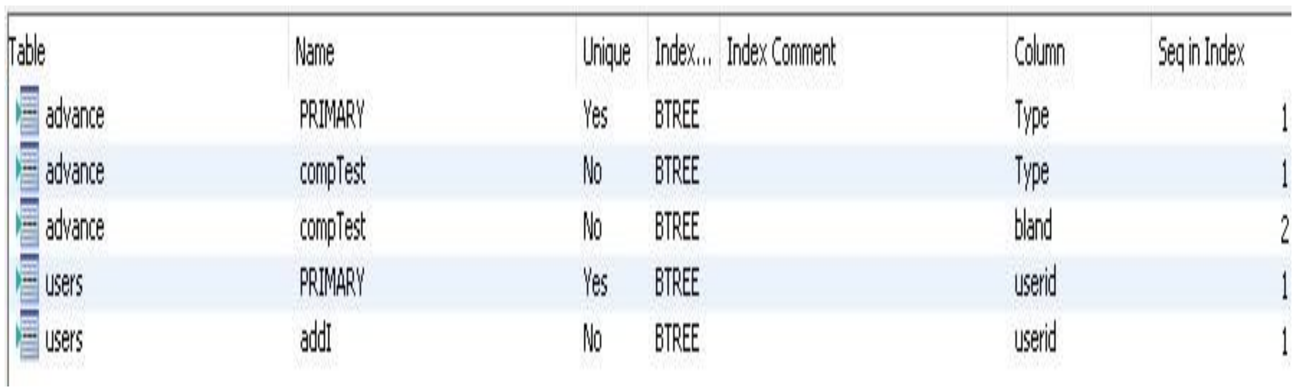
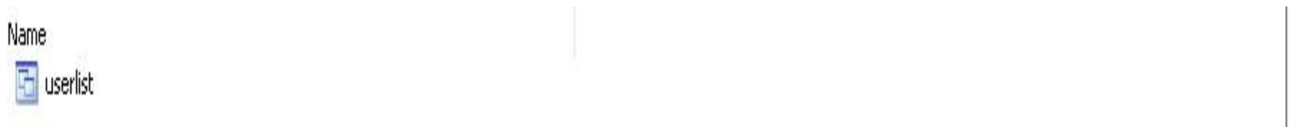


Table	Name	Unique	Index...	Index Comment	Column	Seq in Index
advance	PRIMARY	Yes	BTREE		Type	1
advance	compTest	No	BTREE		Type	1
advance	compTest	No	BTREE		bland	2
users	PRIMARY	Yes	BTREE		userid	1
users	addI	No	BTREE		userid	1

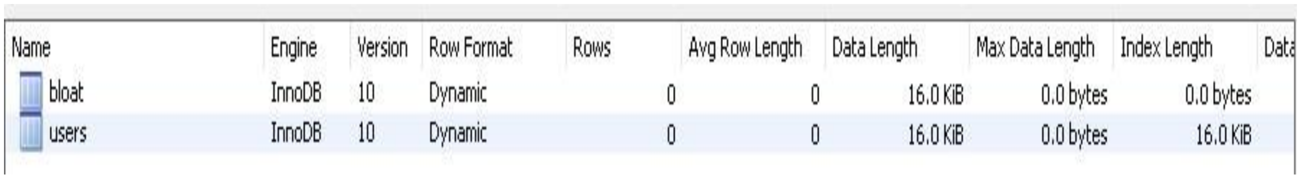
Views



Name
userlist

Live:

Tables



Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length	Data
bloat	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	0.0 bytes	
users	InnoDB	10	Dynamic	0	0	16.0 KiB	0.0 bytes	16.0 KiB	

Columns

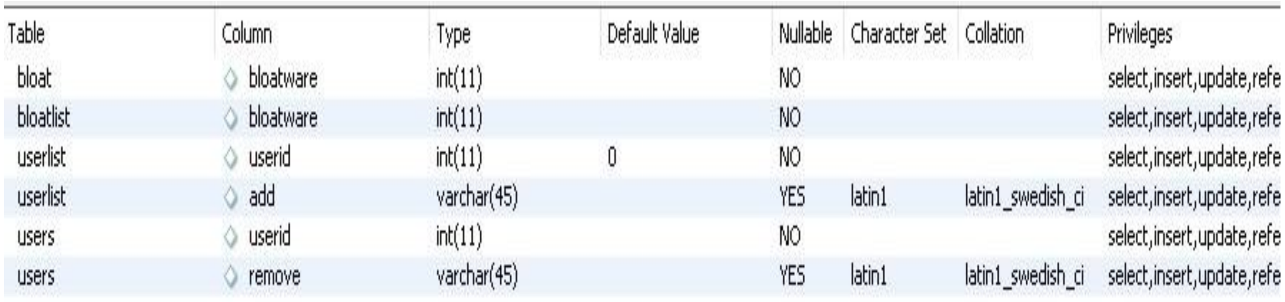


Table	Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
bloat	bloatware	int(11)		NO			select,insert,update,refe
bloatlist	bloatware	int(11)		NO			select,insert,update,refe
userlist	userid	int(11)	0	NO			select,insert,update,refe
userlist	add	varchar(45)		YES	latin1	latin1_swedish_ci	select,insert,update,refe
users	userid	int(11)		NO			select,insert,update,refe
users	remove	varchar(45)		YES	latin1	latin1_swedish_ci	select,insert,update,refe

Indices

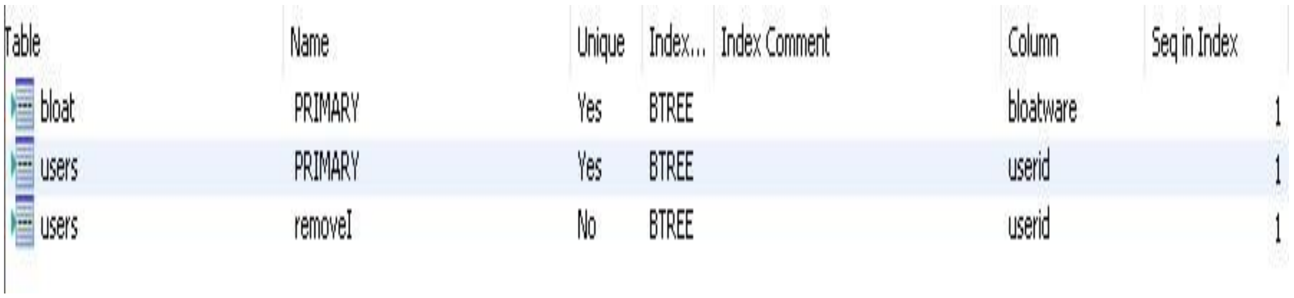
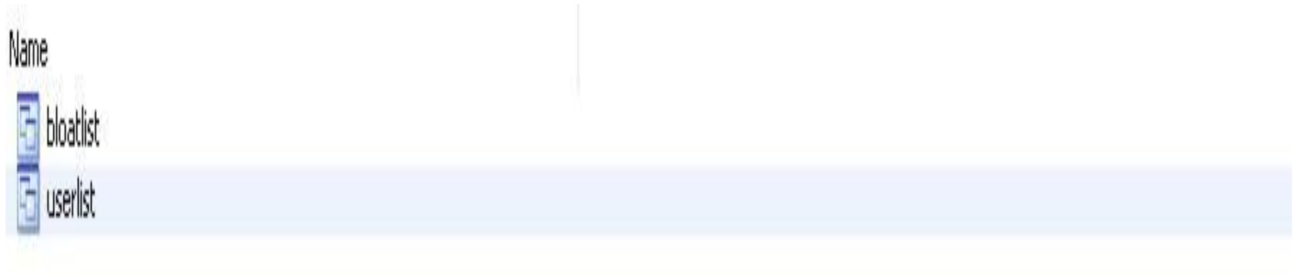


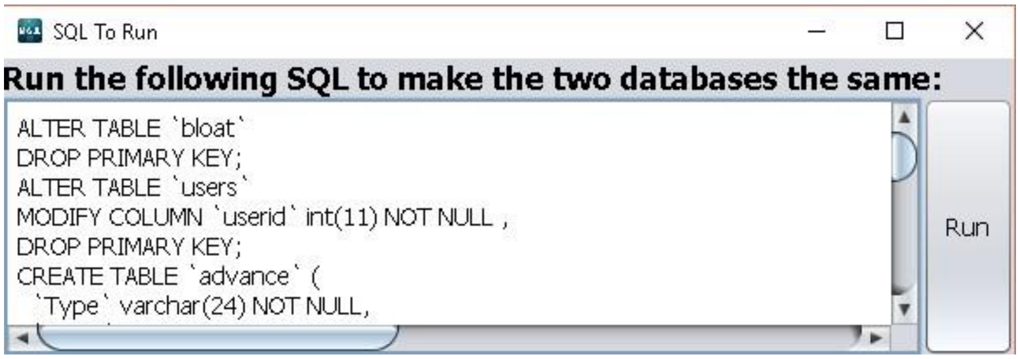
Table	Name	Unique	Index...	Index Comment	Column	Seq in Index
bloat	PRIMARY	Yes	BTREE		bloatware	1
users	PRIMARY	Yes	BTREE		userid	1
users	removeI	No	BTREE		userid	1

Views



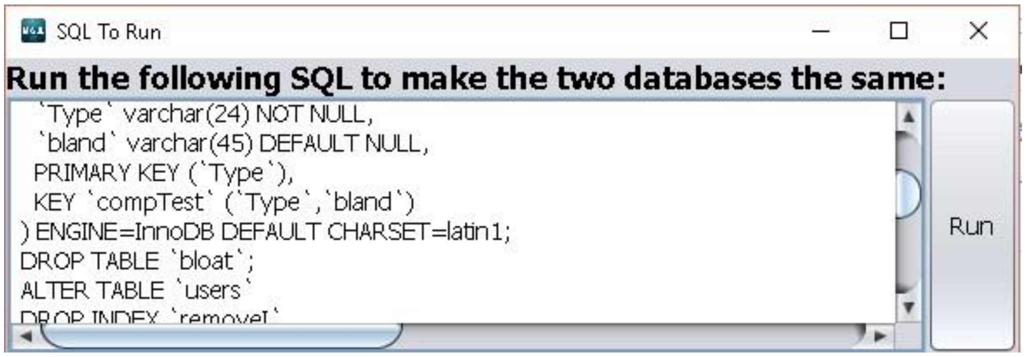
Name
bloatlist
userlist

DBC Result (Default Result):



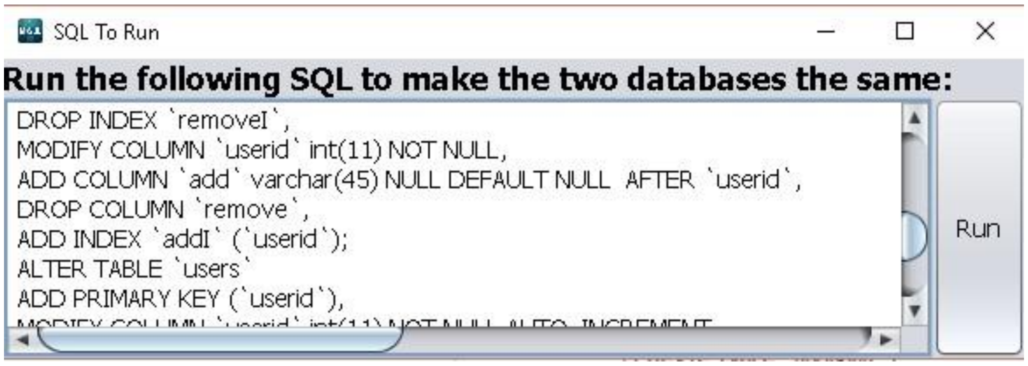
Run the following SQL to make the two databases the same:

```
ALTER TABLE `bloat`
DROP PRIMARY KEY;
ALTER TABLE `users`
MODIFY COLUMN `userid` int(11) NOT NULL ,
DROP PRIMARY KEY;
CREATE TABLE `advance` (
  `Type` varchar(24) NOT NULL,
```


SQL To Run

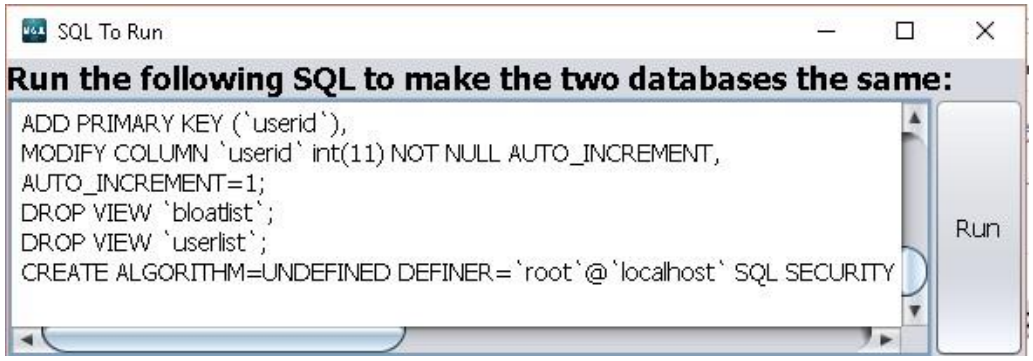
Run the following SQL to make the two databases the same:

```
  `Type` varchar(24) NOT NULL,
  `bland` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`Type`),
  KEY `compTest` (`Type`,`bland`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
DROP TABLE `bloat`;
ALTER TABLE `users`
```


SQL To Run

Run the following SQL to make the two databases the same:

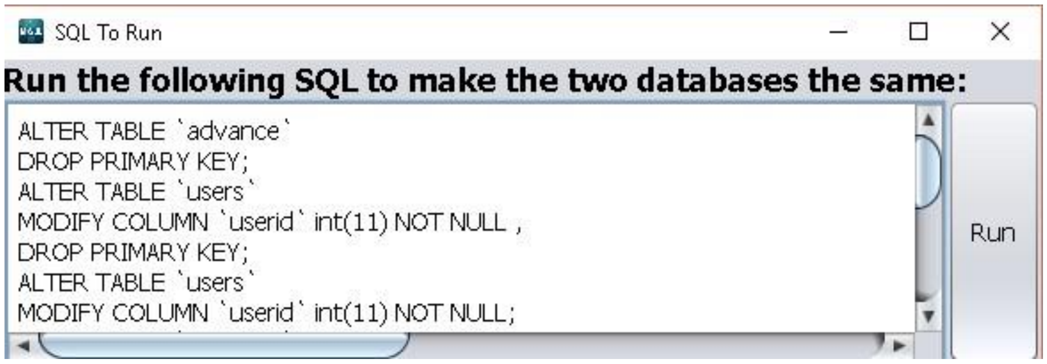
```
DROP INDEX `removeI`,
MODIFY COLUMN `userid` int(11) NOT NULL,
ADD COLUMN `add` varchar(45) NULL DEFAULT NULL  AFTER `userid`,
DROP COLUMN `remove`,
ADD INDEX `addI` (`userid`);
ALTER TABLE `users`
ADD PRIMARY KEY (`userid`),
```


SQL To Run

Run the following SQL to make the two databases the same:

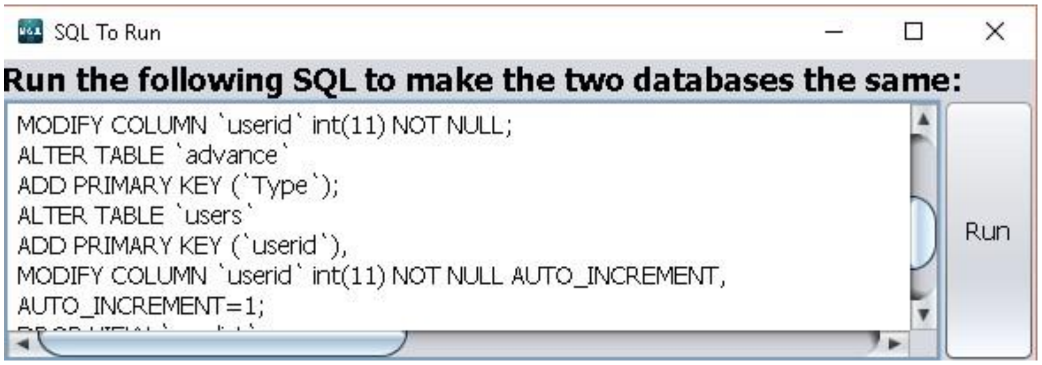
```
ADD PRIMARY KEY (`userid`),
MODIFY COLUMN `userid` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=1;
DROP VIEW `bloatlist`;
DROP VIEW `userlist`;
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
```

DBC Result2:


SQL To Run

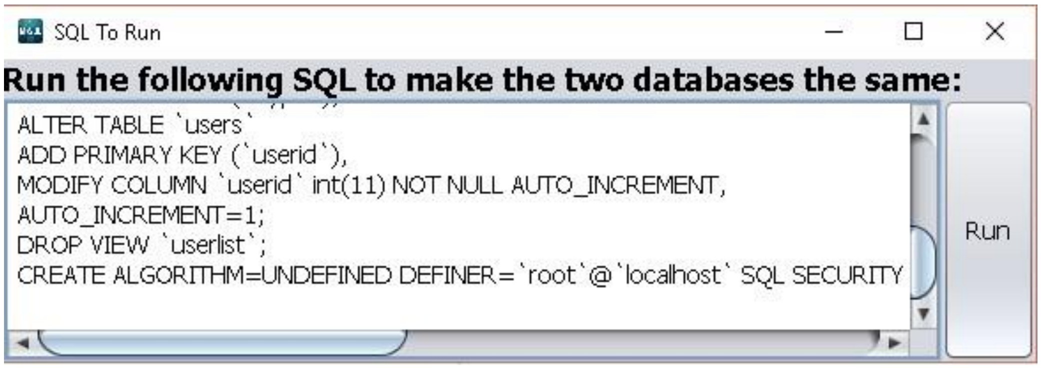
Run the following SQL to make the two databases the same:

```
ALTER TABLE `advance`
DROP PRIMARY KEY;
ALTER TABLE `users`
MODIFY COLUMN `userid` int(11) NOT NULL ,
DROP PRIMARY KEY;
ALTER TABLE `users`
MODIFY COLUMN `userid` int(11) NOT NULL;
```



Run the following SQL to make the two databases the same:

```
MODIFY COLUMN `userid` int(11) NOT NULL;
ALTER TABLE `advance`
ADD PRIMARY KEY (`Type`);
ALTER TABLE `users`
ADD PRIMARY KEY (`userid`),
MODIFY COLUMN `userid` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=1;
```



Run the following SQL to make the two databases the same:

```
ALTER TABLE `users`
ADD PRIMARY KEY (`userid`),
MODIFY COLUMN `userid` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=1;
DROP VIEW `userlist`;
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
```

Conclusion

The DBC makes comparing two databases' tables, columns, indices, and views easy regardless of whether or not the comparison is made using a DB snapshot or two database connections (Aaron). Running the DBC generates the SQL statements, which make two databases the same. The DBC can be used when updating software and bringing a live database up to speed with a dev database.

Works Cited:

Aaron, Rance. Personal Interview. 1 June 2017 – 11 Aug. 2017.

“The Use of Multiple JFrames: Good or Bad Practice?,” *Stack Overflow,* Stack Exchange Inc, 4 Mar. 2012, stackoverflow.com/questions/9554636/the-use-of-multiple-jframes-good-orbad-practice.