

James Schallert

Databases

Final Project

## **Mockbuster: Creating a Database with Front End Accessibility**

### **Introduction:**

The goal of this project is to acquire and reinforce the skills needed to implement a database accessible via a web interface. MySQL was chosen for the database because it was voted most popular database in a StackOverflow survey in 2018 [1]. Python was chosen for the back end as it is the most popular programming language for data science [2]. Django was chosen for the front-end CRUD application as it was the top recommended front-end framework for Python [3].

The first step of the project was to assess the needs of a mock company. The mock company was a chain of video game rental stores called Mockbuster. Mockbuster needed a system they could use to track video game inventory, customers, employees, rentals, vendors, store locations, and purchases as well as collect data on rental history. The information would be stored in a database accessible by managers and employees via login to a company intranet, as well as directly queryable for data analysis. The intranet needed to include functionality in the front-end for checking out and returning games, reviewing lists of currently checked out games, signing up customers, and membership cancellation. In addition, an administrative page was required to be accessible by managers to include functionality for adding and deleting employees, buying new games, and adding store locations. The company also wanted to collect customer's rental history so that they could later implement a recommendation system as well as making better informed decisions on what types of games to buy and how many.

## Requirements:

To model the database, entities (objects based on the Mockbuster rental stores) were defined to later become tables in the database. The entities that we needed to track were video game inventory (Appendix 1.1: VIDEO\_GAME), video game details (Appendix 1.10: VIDEO\_DETAIL), rentals (Appendix 1.8: RENTAL), customers (Appendix 1.6: CUSTOMER), employees (Appendix 1.3: EMPLOYEE), managers (Appendix 1.7: MANAGER), store branches (Appendix 1.4: BRANCH), rental history (Appendix 1.9: RENTAL\_HIST), purchase history (Appendix 1.5: PURCHASE\_HIST), and vendors (Appendix 1.2: VENDOR). The Entity Relationship Diagram can be viewed in Appendix 2.

In the VIDEO\_GAME table (Appendix 1.1), each video game inventory entry needs to contain the attributes video game details (VD\_ID), copies owned (VG\_COPIES\_OWNED), copies rented (VG\_COPIES\_RENTED), and which store they are located in (BR\_ID). The branch store attribute is a non-null foreign key connecting to the branch store entity table in a one (BRANCH) to many (VIDEO\_GAME) relationship. VD\_ID is a non-null foreign key connecting to the video game details entity in a one (VIDEO\_DETAIL) to many (VIDEO\_GAME) relationship. The attribute VG\_COPIES\_RENTED will be updated automatically via triggers when either rentals (Appendix 3.1) or returns are made (Appendix 3.2). The attribute VG\_COPIES\_OWNED will be updated automatically via triggers when purchases are made (Appendix 3.3). Each entry also contains an auto-incremented, non-null primary key (VG\_ID) and has a uniqueness constraint placed on each unique combination of video game and branch location. The video game entity is also able to be modified manually through the front-end admin page.

In the VIDEO\_DETAIL table (Appendix 1.10), each video game detail entry contains the name (VD\_NAME), genre (VD\_GENRE), and publisher (VD\_PUBLISHER). Each entry also contains an auto-incremented, non-null primary key (VD\_ID). The video game detail entity is also modifiable through the front-end admin page.

In the RENTAL table (Appendix 1.8), each rental entry contains the attributes rental date (RE\_DATE), what game was rented (VG\_ID), which employee rented it (EM\_ID), which branch store the game was being rented from (BR\_ID), which customer the game was rented to (CU\_ID), and the date that the game is due back (DUE\_DATE). VG\_ID, EM\_ID, BR\_ID, and CU\_ID are all non-null foreign keys to the VIDEO\_GAME, EMPLOYEE, BRANCH, and CUSTOMER entities, respectively. These are all one (other) to many (RENTAL) relationships. Each entry also contains an auto-incremented, non-null primary key (RE\_ID). The rental date and due date are created automatically in the front-end when a game is rented. Whenever an entry is added or removed, a trigger updates the VIDEO\_GAME entity for the video game rented, increasing or decreasing the number of copies currently rented (Appendix 3.1,3.2). Also, whenever an entry is added, an entry is created in the RENTAL\_HIST entity logging the transaction (Appendix 3.1). This entity will be pivotal in the front-end. The data of its entries will be displayed on the current rentals page. When a game is rented or returned in the front end, this is the entity that will be getting modified.

In the RENTAL\_HIST table (Appendix 1.9), each rental has a log written to the rental history containing the attributes date (RH\_DATE), what game was rented (VG\_ID), who it was rented to (CU\_ID), which employee made the transaction (EM\_ID), and which branch it was rented at (BR\_ID). VG\_ID, EM\_ID, BR\_ID, and CU\_ID are all non-null foreign keys to the VIDEO\_GAME, EMPLOYEE, BRANCH, and CUSTOMER entities, respectively. These are all

one (other entity) to many (RENTAL\_HIST) relationships. Each entry also contains an auto-incremented, non-null primary key (RH\_ID). This entity is written to automatically whenever a video game is rented via trigger (Appendix 3.1).

In the CUSTOMER table (Appendix 1.6), each customer entry contains the attributes first name (CU\_FIRST\_NAME), last name (CU\_LAST\_NAME), phone number (CU\_PHONE\_NUM), and address (CU\_ADDRESS). Each entry also contains an auto-incremented, non-null primary key (CU\_ID). There is also a uniqueness constraint for every combination of last name and phone number so that customers can't accidentally be signed up multiple times. This entity will have its contents displayed in the front-end as well as being able to add or delete customers.

In the EMPLOYEE table (Appendix 1.3), each employee entry contains the attributes first name (EM\_FIRST\_NAME), last name (EM\_LAST\_NAME), pay rate (EM\_PAY\_RATE), phone number (EM\_PHONE\_NUM), date of birth (EM\_DOB), social security number (EM\_SS\_NUM), email address (EM\_ADDRESS), and branch store location (BR\_ID). Each entry also contains an auto-incremented, non-null primary key (EM\_ID). BR\_ID is a one (BRANCH) to many (EMPLOYEE) non-null foreign key. There is a uniqueness constraint placed on the social security attribute (EM\_SS\_NUM) so that an employee can't be accidentally added multiple times. The employee entity will be able to be viewed, modified, added to or deleted from through the front-end admin page.

In the MANAGER table (Appendix 1.7), each manager entry is comprised of two one-to-one, non-null foreign keys, the employee (EM\_ID) and branch store (BR\_ID). Each entry also contains an auto-incremented, non-null primary key (MA\_ID). This entity simply links an

employee to a branch to show that they are the manager. This entity is modifiable via the front-end admin page.

In the BRANCH table (Appendix 1.4), each branch entry is comprised of the address of the store (BR\_ADDRESS), the store's phone number (BR\_PHONE\_NUM), and the manager (MA\_ID). Each entry also contains an auto-incremented, non-null primary key (BR\_ID).

MA\_ID is a one-to-one non-null foreign key connecting BRANCH to MANAGER. Branches can be added and removed through the front-end admin page.

In the PURCHASE\_HIST table (Appendix 1.5), each purchase entry is comprised of the attributes date of purchase (PH\_DATE), game purchased (VG\_ID), the branch store purchasing the game (BR\_ID), the number of games purchased (NUM\_PURCHASED), the cost of each game (IND\_COST), the total cost (TOT\_COST), who the game was purchased from (VE\_ID), and who the manager that purchased the game was (MA\_ID). VG\_ID, BR\_ID, VE\_ID, and MA\_ID are all one (other) to many (PURCHASE\_HIST) non-null foreign keys connecting to the VIDEO\_GAME, BRANCH, VENDOR, and MANAGER entities. Each entry also contains an auto-incremented, non-null primary key (PH\_ID). Whenever an entry is added to PURCHASE\_HIST, the VIDEO\_GAME entity is automatically updated with a trigger, adding the number of games purchased to the number of copies owned of that game (Appendix 3.3). When a game is purchased, the transaction will be entered through the front-end admin page.

In the VENDOR table (Appendix 1.2), each vendor entry is comprised of the attributes name (VE\_NAME), the vendor's phone number (VE\_PHONE\_NUM), and the vendor's address (VE\_ADDRESS). Each entry also contains an auto-incremented, non-null primary key (VE\_ID). Vendors can be added and deleted through the front-end admin page.

Intuitively, for all names, addresses, genres, and publishers, the data is stored as the varchar data-type of varying lengths. All attributes pertaining to money, such as pay rate and game costs, are stored as the datatype decimal. All keys and discretely counted variables, such as copies of a game owned, are stored as integers. Most obviously, dates are stored using the date data-type. Unintuitively, phone numbers and social security numbers are stored as varchars, as they include non-numeric characters in addition to numerals. The specifics can be found in the Entity Relationship Diagram (Appendix 2).

The database was built with normalization in mind. Each row contains a single entry and uniqueness constraints are in place to achieve first normal form: non-repeating groups of data. Each entity has a single non-null, auto-incrementing primary key that all the other attributes are solely dependent on with no transitive dependency, achieving second and third normal form. When designing the database, whenever a non-key dependency or transitive dependency was found, a new entity would be created. An example of this was the original video game inventory entity. It originally also contained all the information that is now stored in the video game detail entity.

### **Implementation:**

This project consists of three main pieces: the database, front-end, and back-end. After the requirements of the database were established and the ERD diagram was laid out, the database was built using MySQL Workbench and then populated with data using SQL statements. After the database was built and populated, the front-end was built through Python using the Django framework.

The first step in building the database was creating the server where the application would be stored and accessed. As this was a sample project, it was simply run on the local host (Appendix 4.1). Once the server was created, the structure of the database could then be built using the MySQL Workbench scripting interface. Using SQL queries, first the database itself was created, then each table to represent each entity (Appendix 4.2). After the tables were created, the foreign keys were assigned (Appendix 4.3). Next, the triggers were added (Appendix 4.4). Now that the structure of the database was built, the database was populated with mock data using SQL statements (Appendix 4.5).

Having built and populated the database, next was building the front end. Django's web framework made this process much easier. After installing Django, the first step was creating a new project. Projects in Django are a repository for all the python code, html templates, and other files needed by the front-end. The next step is to create an app for your database. When a new app is created, Django populates it with many pre-set views and templates to make building the front-end much easier. Once the app is created, it must be added into the project's settings.py file under `INSTALLED_APPS`. In order to connect Django to our database, we also need to change the database parameters in the settings.py file to match our database's (Appendix 4.6).

Next, the models.py file must be created in the app's directory. Model is Django's moniker to reference the entities of the database. To import the models from the already built database, the database's information needs to be gathered in a form that Django can interpret. Conveniently, Django has this functionality built in and can gather the information it needs through the command line [4]. Once the information is gathered, it can be added to the models.py file (Appendix 4.7). Next, the data must be migrated. This can be done with a

*makemigrations* command followed by a *migrate* command at the terminal through the `manage.py` file.

Django has a built-in admin page that allows you to perform create, retrieve, update, and delete functions on any model you connect to it, given that you're logged in as a user with sufficient permissions. It is accessible by creating an `admin.py` file in the app's directory and importing the necessary Django sub-libraries, then telling it which models you want it to have access to (Appendix 4.8). It also has the ability to add, remove, and modify users as well as their permissions.

To create the other pages of the front-end, a file called `views.py` must be created in the app's directory. This file includes an object for each page being made. Each object is passed a generic view object from the Django library that serves as a base and gives a lot of functionality. The model that each view is focused on must also be designated, as well as fields (Django's moniker for attributes) and success URLs for views that need them (Appendix 4.9).

After the views are created, the URLs that direct to them must be defined. There needs to be a `urls.py` file in the app's directory as well as a `urls.py` file in the project directory that references the first (Appendix 4.10). Django uses a special URL dispatcher that simplifies navigation between pages as well as passing parameters between them [5]. Once the URLs have been set, HTML templates for each view must be created in the app's directory (Appendix 4.11). All that is left is to create a superuser (Admin) at the command line, run the server, and then log in.



## **Functional Description**

There are two users for this system defined by their role. Employees have access to the base CRUD application, where they can sign up or modify customer memberships and rent or return games. Managers are granted administrator access and have access to all the tables in the database through the admin pages. This allows them to purchase more games, hire and remove employees, and create branch stores on top of the basic employee functionality. A scenario utilizing this system would be an employee helping a new customer that wants to rent a game. The customer comes up to the counter at the store with the game that they want to rent. The customer is new, so they must be entered into the system before they can rent the game. The employee opens the webpage (Appendix 4.12) and logs in (Appendix 4.13) to the system (Appendix 4.14) and navigates to the customer page (Appendix 4.15). The employee presses the New Customer button and fills out the form (Appendix 4.16). After completing it, he is redirected and sees the new customer's entry has appeared (Appendix 4.17). He then returns to the homepage, then navigates to the rental page (Appendix 4.18). This page shows all the currently rented games. The employee clicks the New Rental link and is brought to the New Rental form to fill out (Appendix 4.19). Currently, the form is populated by drop-down menus with entries corresponding to primary keys. These are placeholders that would be replaced with a scanned-in barcode on the game, as well as from a customer's membership card. It would also be helpful to automatically fill out the branch and employee currently signed in, but it has not been implemented yet. After clicking submit, the employee is redirected back to the rental page where he sees the new entry (Appendix 4.20).

Another example is a manager with administrator privileges needing to add a new employee. Logging in as a user with admin privileges gives a slightly different homepage (Appendix 4.21), which can be used to navigate to the admin page (Appendix 4.22). From the admin page, the manager can navigate to the add employee page (Appendix 4.23). After adding the employee to the database, the manager must navigate to the add user page (Appendix 4.24) to set the employees login and permissions (Appendix 4.25).

## **Testing**

It was stated in the requirements that the company was interested in using rental history for business intelligence, particularly recommendation systems. The company can test the server database with SQL statements that may be of use for this purpose. The obvious would be querying what movies all users have seen with a select statement on the rental history table (Appendix 4.26). The query in the appendix only returns pairs of video game and customer primary keys, but it is ideal to feed into a recommendation system. The recommendation system would return primary keys as recommendations, which could be reintroduced to the database system easily. If the user preferred having the names, the tables could be joined with the video game table and customer table and pull that data from those tables (Appendix 4.27). A simple test would be, what movies does a customer currently have checked out (Appendix 4.28)? This query shows that the customer with ID 4 has one movie checked out.

## **Conclusion**

The goal was to develop the skills necessary to build a database with front end access. The demonstration and testing shown in the functional description prove that this goal was a success. The database is relatively complex without sacrificing usability or normalization. The

mock company's goal to collect data that could be used in the future for business intelligence applications was also a success. The rental history table was the result of that goal, which could easily be mined to create a recommendation system for customers. In order to implement the recommendation system, the data would need to be piped from MySQL into a form that can be used with one of the many Python data science libraries, such as Orange or Scikit-Learn. The most difficult part was creating the app, views and templates for Django. The CRUD application isn't visually appealing, but all the required functionality is there. The next step would be implementing CSS to make the site more visually appealing.

Concerning Mockbuster, all their needs were met as laid out in the introduction. The system currently requires some interaction from the user that could be automated out. Two examples are the case of having to select your employee id and location id when renting out a game. Both could be set when logging in to the web application. The system would also need to integrate some sort of physical scanning device to scan video game bar codes as well as membership ID's. It could also be helpful to integrate more views into the front-end with functionality like seeing what games a customer has checked out. Most importantly, a system for accepting payment would need to be integrated somehow. However, that is beyond the scope of this project and the goal of implementing a database with a front-end was achieved.

## Bibliography

- [1] T. Shay, "Most popular databases in 2018 according to StackOverflow survey". *EverSQL*, March 13, 2018. [Online], Available: <https://www.eversql.com/most-popular-databases-in-2018-according-to-stackoverflow-survey/>. [Accessed: 10-May-2019].
- [2] A. Babu, "Top 8 programming languages every data scientist should master in 2019". *Big Data Made Simple*, Jan 24, 2019. [Online], Available: <https://bigdata-madesimple.com/top-8-programming-languages-every-data-scientist-should-master-in-2019/>. [Accessed: 10-May-2019].
- [3] SteelKiwi, "Top 10 Python Web Frameworks to Learn in 2018". *Hackernoon*, Mar 6, 2018. [Online], Available: <https://hackernoon.com/top-10-python-web-frameworks-to-learn-in-2018-b2ebab969d1a>. [Accessed: 10-May-2019].
- [4] "Integrating Django with a legacy database". *Django*. [Online], Available: <https://docs.djangoproject.com/en/2.2/howto/legacy-databases/>. [Accessed: 10-May-2019].
- [5] "URL dispatcher". *Django*. [Online], Available: <https://docs.djangoproject.com/en/2.2/topics/http/urls/>. [Accessed: 10-May-2019].
- [6] "Create Trigger in MySQL". *MySQLTutorial*. [Online], Available: <http://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx>. [Accessed: 10-May-2019].
- [7] "Using MySQL ALTER TABLE To Change Table Structure". *MySQLTutorial*. [Online], Available: <http://www.mysqltutorial.org/mysql-alter-table.aspx>. [Accessed: 10-May-2019].
- [8] "MySQL CURDATE() function". *W3Resource*, Apr 4, 2019. [Online], Available: <https://www.w3resource.com/mysql/date-and-time-functions/mysql-curdate-function.php>. [Accessed: 10-May-2019].

- [9] BlueApple Courses, “How to connect Django to MySQL”. *Youtube*, Sep 16, 2015. [Video File], Available: <https://www.youtube.com/watch?v=E-lhwAW4vs4>. [Accessed: 10-May-2019].
- [10] [Online], Available: <https://www.digitalocean.com/community/tutorials/how-to-create-a-django-app-and-connect-it-to-a-database>. [Accessed: 10-May-2019].
- [11] J. Morris, “How To Create a Django App and Connect it to a Database”. *Digital Ocean*, Apr 9, 2018. [Online], Available: <https://django-book.readthedocs.io/en/latest/chapter18.html>. [Accessed: 10-May-2019].
- [12] “Python interface to MySQL”. *Anaconda Cloud*. [Online], Available: <https://anaconda.org/bioconda/mysqlclient>. [Accessed: 10-May-2019].
- [13] “Managing static files (e.g. images, JavaScript, CSS)” *Django*. [Online], Available: <https://docs.djangoproject.com/en/dev/howto/static-files/>. [Accessed: 10-May-2019].
- [14] S. Wambler, “Introduction to Data Normalization: A Database "Best" Practice”. *Agile Data*. [Online], Available: <http://agiledata.org/essays/dataNormalization.html>. [Accessed: 10-May-2019].
- [15] W. Vincent, “Django Login/Logout Tutorial”. *William Vincent's Blog*, Oct 22, 2018. [Online], Available: <https://wsvincent.com/django-user-authentication-tutorial-login-and-logout/>. [Accessed: 10-May-2019].
- [16] “How do you join two tables on a foreign key field using django ORM?”. *Stack Overflow*, Oct 26, 2012. [Online], Available: <https://stackoverflow.com/questions/13092268/how-do-you-join-two-tables-on-a-foreign-key-field-using-django-orm>. [Accessed: 10-May-2019].

## APPENDIX

Appendix 0.1: All code can be found at <https://github.com/JamesSchallert/MockbusterDatabase/>

Appendix 0.2: Demo video can be found at <https://www.youtube.com/watch?v=v1Cs9jejT54>

### Appendix 1: Entities

#### Appendix 1.1: VIDEO\_GAME table

	VG_ID	VD_ID	VG_COPIES_OWNED	VG_COPIES_RENTED	BR_ID
▶	1	1	5	2	1
	2	1	5	1	2
	3	2	5	1	1
	4	2	5	1	2
	5	3	5	1	1
	6	3	5	1	2
	7	4	5	1	1
	8	4	5	1	2
	9	5	5	1	1
	10	5	5	1	2
	11	6	5	0	1
	12	6	5	0	2
	13	7	5	0	1
	14	7	5	0	2
	15	8	5	0	1
	16	8	5	0	2
	17	9	5	0	1
	18	9	5	0	2
	19	10	5	0	1
	20	10	7	0	2
	21	11	5	0	1
	22	11	7	0	2
	23	12	5	0	1

#### Appendix 1.2: VENDOR table

	VE_ID	VE_NAME	VE_PHONE_NUM	VE_ADDRESS
▶	1	Wholesale Video Game Sales	607-654-6674	10 Wherever Ave, Troy, NY, 12180
	2	LameStop	607-613-3333	15 Laurence Lane, Maryland, NY, 12116
	3	Electronic Larks	603-999-8071	5000 Hullabaloo Way, Nashua, NH, 03063
✱	NULL	NULL	NULL	NULL

### Appendix 1.3: EMPLOYEE table

	EM_ID	EM_FIRST_NAME	EM_LAST_NAME	EM_PAY_RATE	EM_PHONE_NUM	EM_DOB	EM_SS_NUM	EM_ADDRESS	BR_ID
▶	1	John	Westerson	25.00	607-607-6677	1985-04-25	111-111-1111	106 Pheonix Rd, Maryland, NY, 12116	1
	2	Jane	Easterson	10.00	511-611-6457	1995-02-15	222-222-2222	107 Tiger Rd, Maryland, NY, 12116	1
	3	Elon	Musket	10.00	123-132-1677	1975-03-17	333-333-3333	123 Tesla St, Maryland, NY, 12116	1
	4	Jason	Boberts	25.00	518-518-5188	1905-07-15	110-110-1110	5 Proto Rd, Troy, NY, 12180	2
	5	Janet	Weiss	10.00	518-621-6777	1990-12-25	232-232-2213	107 Rocky Rd, Troy, NY, 12180	2
	6	Bronn	Guiermo	10.00	115-162-1669	1960-01-01	353-353-3553	123 Castle St, Troy, NY, 12180	2
	7	Tobias	Manne	10.00	654-654-6666	2000-05-03	605-88-8512	105 Whest La, Rooberville, OH, 11111	1
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### Appendix 1.4: BRANCH table

	BR_ID	BR_ADDRESS	BR_PHONE_NUM	MA_ID
▶	1	1776 Freedom Rd, Maryland, NY, 12116	456-456-4567	1
	2	1920 Roarin Rd, Troy, NY, 12180	120-120-1203	2
*	NULL	NULL	NULL	NULL

### Appendix 1.5: PURCHASE\_HIST table

	PH_ID	PH_DATE	VG_ID	BR_ID	NUM_PURCHASED	IND_COST	TOT_COST	VE_ID	MA_ID
▶	1	2019-05-03	1	1	5	50.00	NULL	1	1
	2	2019-05-03	3	1	5	50.00	NULL	1	1
	3	2019-05-03	5	1	5	50.00	NULL	1	1
	4	2019-05-03	7	1	5	50.00	NULL	1	1
	5	2019-05-03	9	1	5	50.00	NULL	1	1
	6	2019-05-03	11	1	5	50.00	NULL	1	1
	7	2019-05-03	13	1	5	50.00	NULL	1	1
	8	2019-05-03	15	1	5	50.00	NULL	1	1
	9	2019-05-03	17	1	5	50.00	NULL	1	1
	10	2019-05-03	19	1	5	50.00	NULL	1	1
	11	2019-05-03	21	1	5	50.00	NULL	1	1
	12	2019-05-03	23	1	5	50.00	NULL	1	1
	13	2019-05-03	25	1	5	50.00	NULL	1	1
	14	2019-05-03	27	1	5	50.00	NULL	1	1
	15	2019-05-03	29	1	5	50.00	NULL	1	1
	16	2019-05-03	31	1	5	50.00	NULL	1	1
	17	2019-05-03	33	1	5	50.00	NULL	1	1
	18	2019-05-03	35	1	5	50.00	NULL	1	1
	19	2019-05-03	37	1	5	50.00	NULL	1	1
	20	2019-05-03	39	1	5	50.00	NULL	1	1
	21	2019-05-03	2	2	5	50.00	NULL	2	2
	22	2019-05-03	4	2	5	50.00	NULL	2	2
	23	2019-05-03	6	2	5	50.00	NULL	2	2
	24	2019-05-03	8	2	5	50.00	NULL	2	2





## Appendix 1.6: CUSTOMER table

	CU_ID	CU_FIRST_NAME	CU_LAST_NAME	CU_PHONE_NUM	CU_ADDRESS
▶	1	Greg	Bowman	888-888-8888	106 Yeetville Rd, Troy, NY, 12180
	2	Rita	Skita	887-887-8887	107 Yetterville Ave, Troy, NY, 12180
	3	Bill	Bo	877-877-8877	207 Teryville Ave, Troy, NY, 12180
	4	Sarah	Silva	777-777-7777	257 Everyville st, Troy, NY, 12180
	5	Banya	Oster	778-778-7778	27 Villa rd, Troy, NY, 12180
	6	Barry	Stein	788-788-7788	71 Vanilla Way, Troy, NY, 12180
	7	Lu	Bu	686-768-7768	664 Manilla St, Troy, NY, 12180
	8	Dong	Zhuou	486-468-7448	463 Chin Yang St, Troy, NY, 12180
	9	Zhuge	Liang	386-438-7433	43 Tsu Yang St, Troy, NY, 12180
	10	Arnold	Ang	383-638-6433	63 Yang St, Troy, NY, 12180
	11	Gerold	Tuling	333-338-3433	3 Bang St, Maryland, NY, 12116
	12	Ronald	Gungnir	363-638-3463	6 Bang St, Maryland, NY, 12116
	13	Ron	Nirzche	663-636-6466	36 Old Rd, Maryland, NY, 12116
	14	Alejando	Moon	605-104-3333	123 Wherever
⊛	NULL	NULL	NULL	NULL	NULL

## Appendix 1.7: MANAGER table

	MA_ID	EM_ID	BR_ID
▶	1	1	1
	2	4	2
*	NULL	NULL	NULL

### Appendix 1.8: RENTAL table

[illegible]

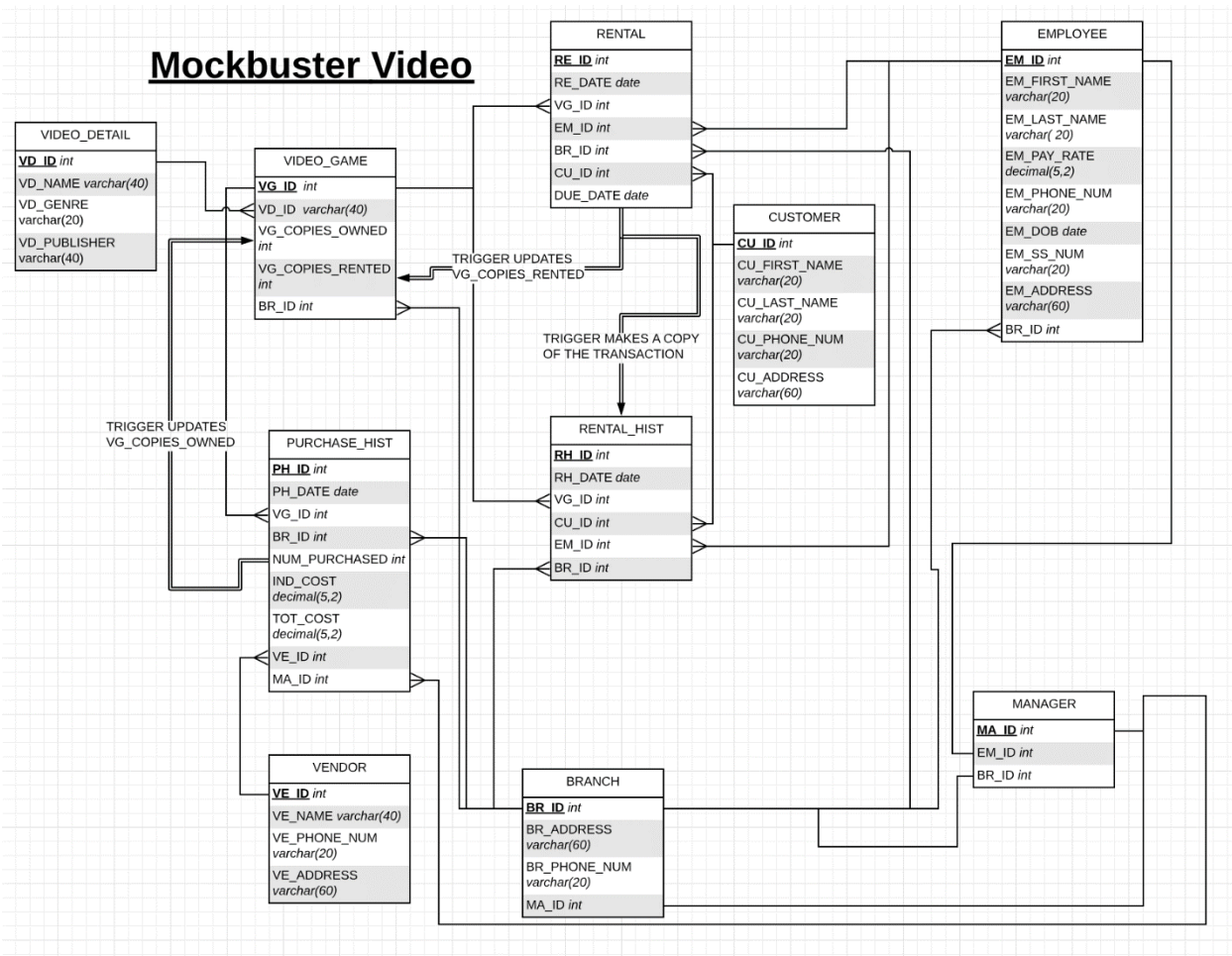
## Appendix 1.9: RENTAL\_HIST table

	RH_ID	RH_DATE	VG_ID	CU_ID	EM_ID	BR_ID
▶	1	2019-05-03	1	1	1	1
	2	2019-05-03	2	2	1	1
	3	2019-05-03	3	3	1	1
	4	2019-05-03	4	4	1	1
	5	2019-05-03	5	5	1	1
	6	2019-05-03	6	6	1	1
	7	2019-05-03	7	7	4	2
	8	2019-05-03	8	8	4	2
	9	2019-05-03	9	9	4	2
	10	2019-05-03	10	10	4	2
	11	2019-05-03	1	3	1	1
	12	2019-05-03	2	4	1	1
	13	2019-05-03	3	5	1	1
	14	2019-05-03	4	6	1	1
	15	2019-05-03	5	7	1	1
	16	2019-05-03	6	8	1	1
	17	2019-05-03	7	9	4	2
	18	2019-05-03	8	10	4	2
	19	2019-05-03	9	11	4	2
	20	2019-05-03	10	12	4	2
	21	2019-05-03	1	1	1	1
	22	2019-05-03	1	1	1	1

## Appendix 1.10: VIDEO\_DETAIL table

	VD_ID	VD_NAME	VD_GENRE	VD_PUBLISHER
▶	1	God of Doors	Action	SOMY Entertainment
	2	Civilized Discourse	Simulation	Mid Seier
	3	Dark Soles	Action RPG	Fromhard
	4	Big Snek	Action	Fromhard
	5	Pretty Intense Soccer 2019	Sports	Electric Arts
	6	Pretty Intense Football 2019	Sports	Electric Arts
	7	FIFER 2019	Sports	Electric Arts
	8	Grandma Fighting Z	Fighting	Electric Arts
	9	Definitely not a clone of Pod Racing	Racing	Not Lucas Arts
	10	Last Fantasy 15	RPG	Trianglesoft
	11	Lord of the Things	Action RPG	Trianglesoft
	12	Gaussian Elimination	Action	Normalsoft
	13	Pro Leet Golf	Sports	Normalsoft
	14	Werewolf Island	Action	Black Peninsula
	15	Robots	Fighting	Turing Co.
	16	Heroes of the Galaxy	Action	Marvelous
	17	World of Battlecraft	RPG	Hurricane
	18	Make it Dead	Horror	Mocpac
	19	Make it Dead 2: Redeadening	Horror	Mocpac
	20	Managing a Zoo: Harambe Edition	Simulation	Mid Seier
*	NULL	NULL	NULL	NULL

## Appendix 2: Entity Relationship Diagram



## Appendix 3: Triggers

### Appendix 3.1: AFTER\_RENTAL\_INSERT

```

192 • CREATE TRIGGER AFTER_RENTAL_INSERT
193     AFTER INSERT ON RENTAL
194     FOR EACH ROW
195     BEGIN
196         INSERT INTO RENTAL_HIST (RH_DATE, VG_ID, CU_ID, EM_ID, BR_ID)
197             Values (NEW.RE_DATE, NEW.VG_ID, NEW.CU_ID, NEW.EM_ID, NEW.BR_ID);
198
199         UPDATE VIDEO_GAME
200             SET VG_COPIES_RENTED = VG_COPIES_RENTED + 1 WHERE VG_ID = NEW.VG_ID;
201     END$$
202     DELIMITER ;
  
```

### Appendix 3.2: BEFORE\_RENTAL\_DELETE

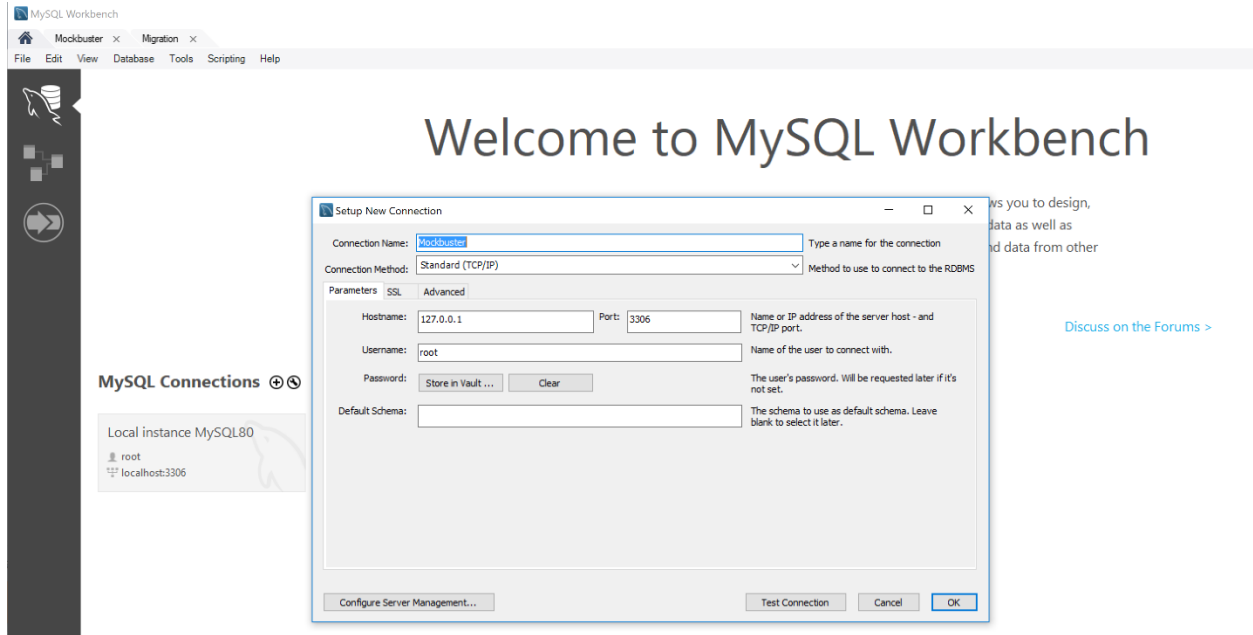
```
204     DELIMITER $$
205 •   CREATE TRIGGER BEFORE_RENTAL_DELETE
206         BEFORE DELETE ON RENTAL
207         FOR EACH ROW
208     BEGIN
209         UPDATE VIDEO_GAME
210         SET VG_COPIES_RENTED = VG_COPIES_RENTED - 1 WHERE VG_ID = OLD.VG_ID;
211     END$$
212     DELIMITER ;
```

### Appendix 3.3: AFTER\_PURCHASE\_HIST\_INSERT

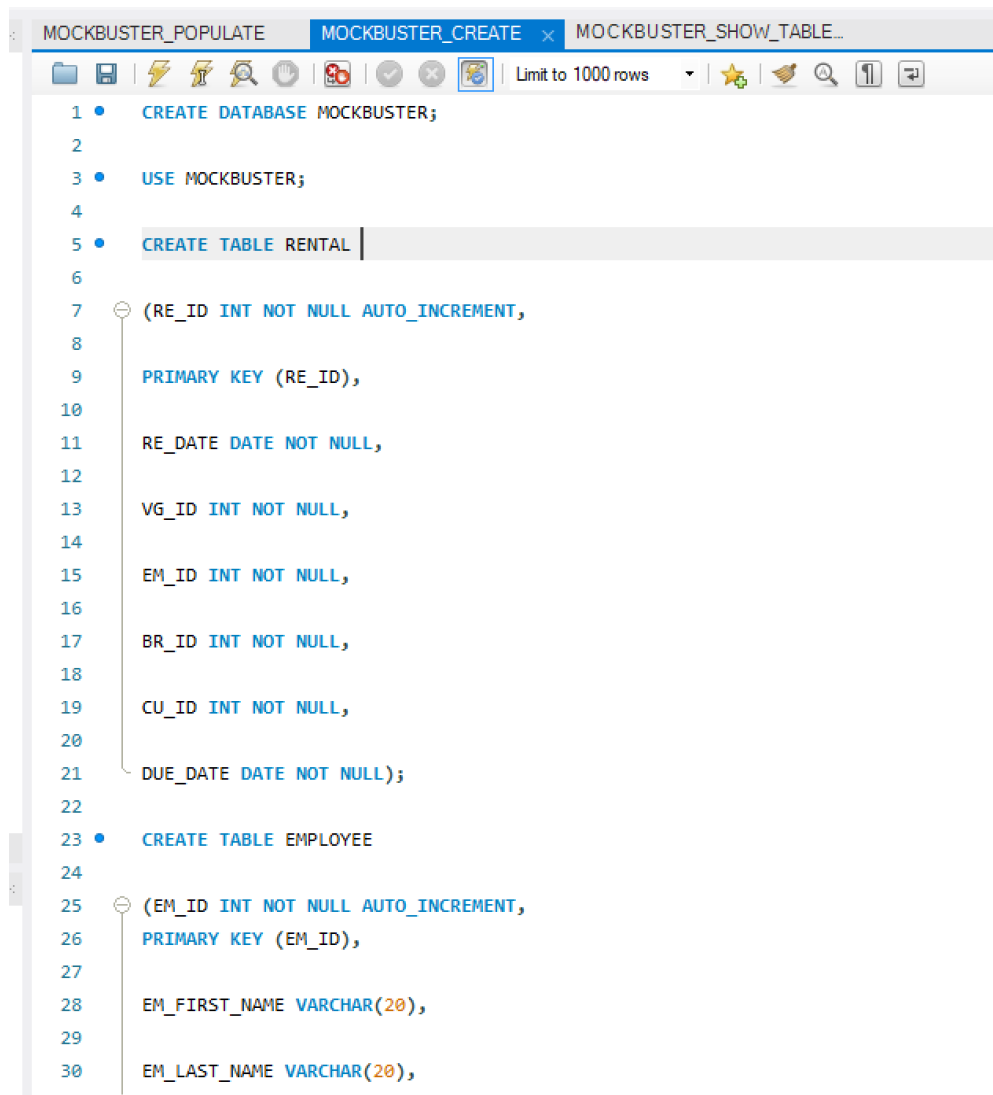
```
214     DELIMITER $$
215 •   CREATE TRIGGER AFTER_PURCHASE_HIST_INSERT
216         AFTER INSERT ON PURCHASE_HIST
217         FOR EACH ROW
218     BEGIN
219         UPDATE VIDEO_GAME
220         SET VG_COPIES_OWNED = VG_COPIES_OWNED + new.NUM_PURCHASED WHERE VG_ID = NEW.VG_ID;
221     END$$
222     DELIMITER ;
```

## Appendix 4: Screen captures

### Appendix 4.1:



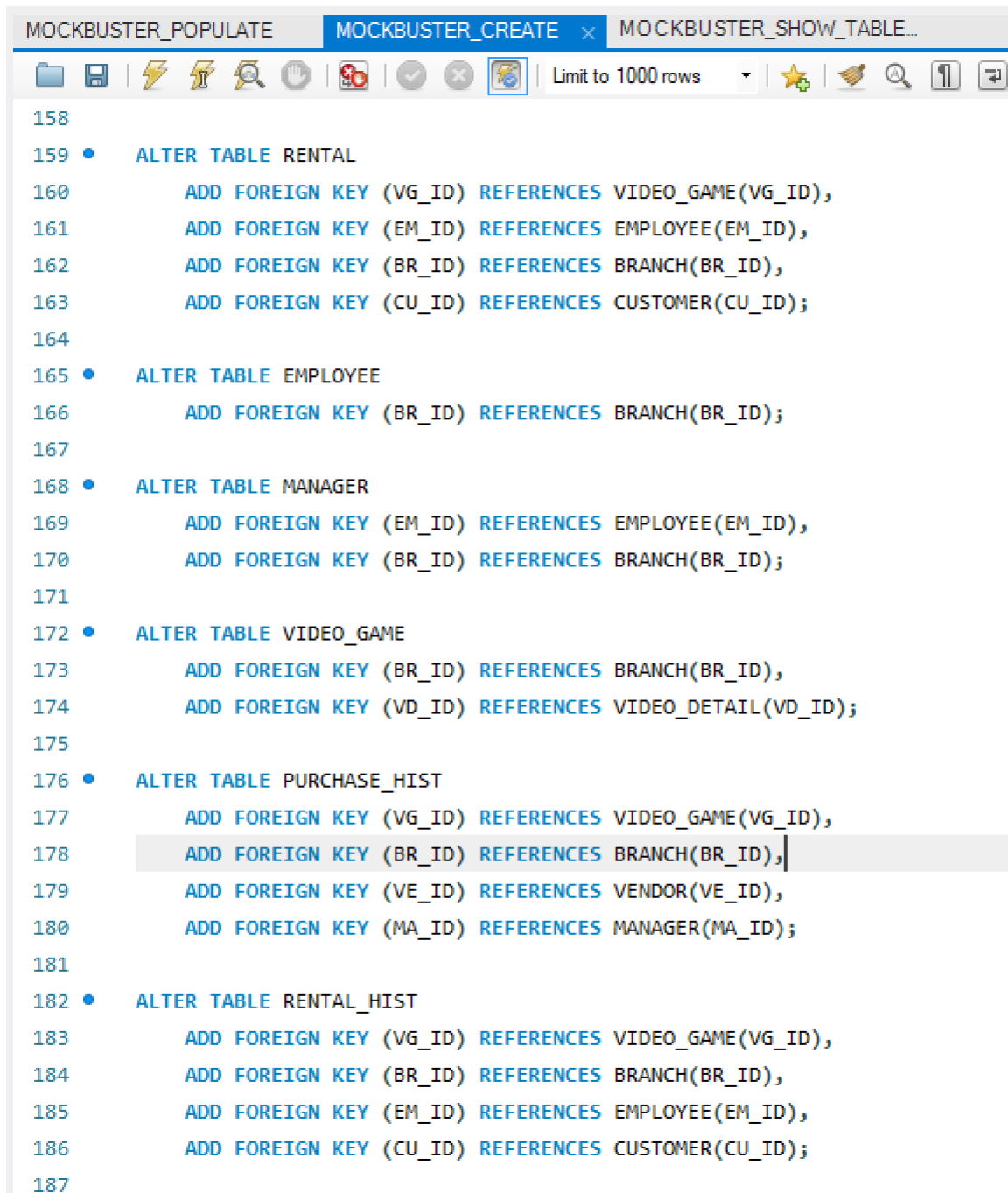
## Appendix 4.2:



The screenshot shows a SQL IDE window with three tabs: 'MOCKBUSTER\_POPULATE', 'MOCKBUSTER\_CREATE' (active), and 'MOCKBUSTER\_SHOW\_TABLE...'. The active tab contains SQL code for creating a database and two tables. The code is as follows:

```
1 • CREATE DATABASE MOCKBUSTER;  
2  
3 • USE MOCKBUSTER;  
4  
5 • CREATE TABLE RENTAL |  
6  
7 (RE_ID INT NOT NULL AUTO_INCREMENT,  
8  
9 PRIMARY KEY (RE_ID),  
10  
11 RE_DATE DATE NOT NULL,  
12  
13 VG_ID INT NOT NULL,  
14  
15 EM_ID INT NOT NULL,  
16  
17 BR_ID INT NOT NULL,  
18  
19 CU_ID INT NOT NULL,  
20  
21 DUE_DATE DATE NOT NULL);  
22  
23 • CREATE TABLE EMPLOYEE  
24  
25 (EM_ID INT NOT NULL AUTO_INCREMENT,  
26 PRIMARY KEY (EM_ID),  
27  
28 EM_FIRST_NAME VARCHAR(20),  
29  
30 EM_LAST_NAME VARCHAR(20),
```

### Appendix 4.3:



The screenshot shows a SQL IDE window with three tabs: 'MOCKBUSTER\_POPULATE', 'MOCKBUSTER\_CREATE' (active), and 'MOCKBUSTER\_SHOW\_TABLE...'. The toolbar includes icons for file operations, execution, and search, along with a 'Limit to 1000 rows' dropdown. The SQL code is as follows:

```
158
159 • ALTER TABLE RENTAL
160     ADD FOREIGN KEY (VG_ID) REFERENCES VIDEO_GAME(VG_ID),
161     ADD FOREIGN KEY (EM_ID) REFERENCES EMPLOYEE(EM_ID),
162     ADD FOREIGN KEY (BR_ID) REFERENCES BRANCH(BR_ID),
163     ADD FOREIGN KEY (CU_ID) REFERENCES CUSTOMER(CU_ID);
164
165 • ALTER TABLE EMPLOYEE
166     ADD FOREIGN KEY (BR_ID) REFERENCES BRANCH(BR_ID);
167
168 • ALTER TABLE MANAGER
169     ADD FOREIGN KEY (EM_ID) REFERENCES EMPLOYEE(EM_ID),
170     ADD FOREIGN KEY (BR_ID) REFERENCES BRANCH(BR_ID);
171
172 • ALTER TABLE VIDEO_GAME
173     ADD FOREIGN KEY (BR_ID) REFERENCES BRANCH(BR_ID),
174     ADD FOREIGN KEY (VD_ID) REFERENCES VIDEO_DETAIL(VD_ID);
175
176 • ALTER TABLE PURCHASE_HIST
177     ADD FOREIGN KEY (VG_ID) REFERENCES VIDEO_GAME(VG_ID),
178     ADD FOREIGN KEY (BR_ID) REFERENCES BRANCH(BR_ID),
179     ADD FOREIGN KEY (VE_ID) REFERENCES VENDOR(VE_ID),
180     ADD FOREIGN KEY (MA_ID) REFERENCES MANAGER(MA_ID);
181
182 • ALTER TABLE RENTAL_HIST
183     ADD FOREIGN KEY (VG_ID) REFERENCES VIDEO_GAME(VG_ID),
184     ADD FOREIGN KEY (BR_ID) REFERENCES BRANCH(BR_ID),
185     ADD FOREIGN KEY (EM_ID) REFERENCES EMPLOYEE(EM_ID),
186     ADD FOREIGN KEY (CU_ID) REFERENCES CUSTOMER(CU_ID);
187
```

#### Appendix 4.4:

```
MOCKBUSTER_POPULATE  MOCKBUSTER_CREATE  MOCKBUSTER_SHOW_TABLE...
Limit to 1000 rows

191  DELIMITER $$
192  • CREATE TRIGGER AFTER_RENTAL_INSERT
193      AFTER INSERT ON RENTAL
194      FOR EACH ROW
195  BEGIN
196      INSERT INTO RENTAL_HIST (RH_DATE,VG_ID,CU_ID,EM_ID,BR_ID)
197      Values (NEW.RE_DATE,NEW.VG_ID,NEW.CU_ID,NEW.EM_ID,NEW.BR_ID);
198
199      UPDATE VIDEO_GAME
200      SET VG_COPIES_RENTED = VG_COPIES_RENTED + 1 WHERE VG_ID = NEW.VG_ID;
201  END$$
202  DELIMITER ;
203
204  DELIMITER $$
205  • CREATE TRIGGER BEFORE_RENTAL_DELETE
206      BEFORE DELETE ON RENTAL
207      FOR EACH ROW
208  BEGIN
209      UPDATE VIDEO_GAME
210      SET VG_COPIES_RENTED = VG_COPIES_RENTED -1 WHERE VG_ID = OLD.VG_ID;
211  END$$
212  DELIMITER ;
213
214  DELIMITER $$
215  • CREATE TRIGGER AFTER_PURCHASE_HIST_INSERT
216      AFTER INSERT ON PURCHASE_HIST
217      FOR EACH ROW
218  BEGIN
219      UPDATE VIDEO_GAME
220      SET VG_COPIES_OWNED = VG_COPIES_OWNED + new.NUM_PURCHASED WHERE VG_ID = NEW.VG_ID;
```



## Appendix 4.5:

```
MOCKBUSTER_POPULATE x MOCKBUSTER_CREATE MOCKBUSTER_SHOW_TABLE...
Limit to 1000 rows

1 • USE MOCKBUSTER;
2
3
4 # Create 3 Vendors to purchase games from:
5
6
7 • INSERT INTO VENDOR (VE_NAME,VE_PHONE_NUM,VE_ADDRESS) VALUES ('Wholesale Video Game Sales','607-654-6674','10 Wherever Ave, Troy, NY, 12180');
8
9 • INSERT INTO VENDOR (VE_NAME,VE_PHONE_NUM,VE_ADDRESS) VALUES ('LameStop','607-613-3333','15 Laurence Lane, Maryland, NY, 12116');
10
11 • INSERT INTO VENDOR (VE_NAME,VE_PHONE_NUM,VE_ADDRESS) VALUES ('Electronic Larks','603-999-8071','5000 Hullabaloo Way, Nashua, NH, 03063');
12
13
14 # Create the first branch store, 3 employees, and assign a manager:
15
16
17 • Insert INTO BRANCH (BR_ADDRESS, BR_PHONE_NUM) VALUES ('1776 Freedom Rd,Maryland, NY, 12116','456-456-4567');
18
19 • Insert INTO EMPLOYEE (EM_FIRST_NAME, EM_LAST_NAME, EM_PAY_RATE, EM_PHONE_NUM, EM_DOB, EM_SS_NUM, EM_ADDRESS, BR_ID)
20   VALUES ('John','Westerson',25.00,'607-607-6677','1985-04-25','111-111-1111','106 Pheonix Rd, Maryland, NY, 12116',1);
21
22 • INSERT INTO MANAGER (EM_ID, BR_ID) VALUE (1,1);
23
24 • UPDATE BRANCH SET MA_ID = 1 WHERE BR_ID = 1;
25
26 • Insert INTO EMPLOYEE (EM_FIRST_NAME, EM_LAST_NAME, EM_PAY_RATE, EM_PHONE_NUM, EM_DOB, EM_SS_NUM, EM_ADDRESS, BR_ID)
27   VALUES ('Jane','Easterson',10.00,'511-611-6457','1995-02-15','222-222-2222','107 Tiger Rd, Maryland, NY, 12116',1);
28
29 • Insert INTO EMPLOYEE (EM_FIRST_NAME, EM_LAST_NAME, EM_PAY_RATE, EM_PHONE_NUM, EM_DOB, EM_SS_NUM, EM_ADDRESS, BR_ID)
30   VALUES ('Elon','Muskett',10.00,'123-132-1677','1975-03-17','333-333-3333','123 Tesla St, Maryland, NY, 12116',1);
```

## Appendix 4.6:

```
72
73
74 # Database
75 # https://docs.djangoproject.com/en/2.2/ref/settings/#databases
76
77 DATABASES = {
78     'default': {
79         'ENGINE': 'django.db.backends.mysql',
80         'NAME': 'MOCKBUSTER',
81         'USER': 'MockbusterAdmin',
82         'PASSWORD': 'Mockbuster',
83         'HOST': 'localhost',
84         'PORT': '3306'
85     }
86 }
87
```

## Appendix 4.7:

```
1 from django.db import models
2
3 # Create your models here.
4 from django.urls import reverse
5 import datetime
6
7 class Mockbuster(models.Model):
8     name = models.CharField(max_length=200)
9     pages = models.IntegerField()
10
11     def __str__(self):
12         return self.name
13
14     def get_absolute_url(self):
15         return reverse('mockbuster_edit', kwargs={'pk': self.pk})
16
17
18 class Branch(models.Model):
19     br_id = models.AutoField(db_column='BR_ID', primary_key=True) # Field name made lowercase.
20     br_address = models.CharField(db_column='BR_ADDRESS', max_length=60, blank=True, null=True) # Field name made lowercase.
21     br_phone_num = models.CharField(db_column='BR_PHONE_NUM', max_length=20, blank=True, null=True) # Field name made lowercase.
22     ma = models.ForeignKey('Manager', models.DO_NOTHING, db_column='MA_ID', blank=True, null=True) # Field name made lowercase.
23
24     class Meta:
25         managed = False
26         db_table = 'branch'
27
28
29 class Customer(models.Model):
30     cu_id = models.AutoField(db_column='CU_ID', primary_key=True) # Field name made lowercase.
31     cu_first_name = models.CharField(db_column='CU_FIRST_NAME', max_length=20, blank=True, null=True) # Field name made lowercase.
32     cu_last_name = models.CharField(db_column='CU_LAST_NAME', max_length=20, blank=True, null=True) # Field name made lowercase.
33     cu_phone_num = models.CharField(db_column='CU_PHONE_NUM', max_length=20, blank=True, null=True) # Field name made lowercase.
34     cu_address = models.CharField(db_column='CU_ADDRESS', max_length=60, blank=True, null=True) # Field name made lowercase.
35
36     class Meta:
37         managed = False
38         db_table = 'customer'
```

## Appendix 4.8:

```
1 from django.contrib import admin
2 from mockbusterApp.models import Branch, Customer, Employee, PurchaseHist, Manager, Rental, RentalHist,
3
4
5 admin.site.register(Branch)
6 admin.site.register(Customer)
7 admin.site.register(Employee)
8 admin.site.register(PurchaseHist)
9 admin.site.register(Manager)
10 admin.site.register(Rental)
11 admin.site.register(RentalHist)
12 admin.site.register(Vendor)
13 admin.site.register(VideoGame)
14 admin.site.register(VideoDetail)
15 # Register your models here.
16
```

## Appendix 4.9:

E:\drive\cs410\mockbuster\mockbusterApp\views.py - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

```
1 from django.shortcuts import render
2
3 # Create your views here.
4 from django.http import HttpResponse
5 from django.views.generic import ListView, DetailView, TemplateView
6 from django.views.generic.edit import CreateView, UpdateView, DeleteView
7 from django.urls import reverse_lazy
8
9 from mockbusterApp.models import Branch, VideoGame, Rental, Customer as CustomerModel
10
11 class Rent(ListView):
12     model = Rental
13     template_name = 'mockbusterApp/Rent.html'
14
15 class Return(DeleteView):
16     model = Rental
17     template_name = 'mockbusterApp/Return.html'
18     success_url = reverse_lazy('Rent')
19
20 class NewRental(CreateView):
21     model = Rental
22     template_name = 'mockbusterApp/NewRental.html'
23     fields = ['vg', 'cu', 'br', 'em']
24     success_url = reverse_lazy('Rent')
25
26 class Customer(ListView):
27     model = CustomerModel
28     template_name = 'mockbusterApp/Customer.html'
29
30 class DeleteCustomer(DeleteView):
31     model = CustomerModel
32     template_name = 'mockbusterApp/DeleteCustomer.html'
33     success_url = reverse_lazy('Customer')
34
35 class NewCustomer(CreateView):
36     model = CustomerModel
37     template_name = 'mockbusterApp/NewCustomer.html'
38     fields = ['cu_first_name', 'cu_last_name', 'cu_phone_num', 'cu_address']
39     success_url = reverse_lazy('Customer')
```

Other files: length: 1076, lines: 65, ln: 32, col: 51, sel: 0:10, Windows (CRLF)

## Appendix 4.10:

```
1 """mockbuster URL Configuration
2
3 The 'urlpatterns' list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/2.2/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10    1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 """
16 from django.contrib import admin
17 from django.urls import path
18 from django.urls import include
19 from django.views.generic.base import TemplateView
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('accounts/', include('django.contrib.auth.urls')),
24     path('', TemplateView.as_view(template_name='home.html'), name='home'),
25     path('mockbuster/', include('mockbusterApp.urls')),
26 ]
27
28
29 from django.urls import path
30 from . import views
31
32 urlpatterns = [
33     path('videogame_inventory/', views.VideoGameList.as_view(), name='videogame_inventory'),
34     path('Rent/', views.Rent.as_view(), name='Rent'),
35     path('Return/<int:pk>', views.Return.as_view(), name='Return'),
36     path('NewRental/', views.NewRental.as_view(), name='NewRental'),
37     path('Customer/', views.Customer.as_view(), name='Customer'),
38     path('NewCustomer/', views.NewCustomer.as_view(), name='NewCustomer'),
39     path('DeleteCustomer/<int:pk>', views.DeleteCustomer.as_view(), name='DeleteCustomer'),
40     path('UpdateCustomer/<int:pk>', views.UpdateCustomer.as_view(), name='UpdateCustomer'),
41     #path('view/<int:pk>', views.MockbusterView.as_view(), name='mockbuster_view'),
42     #path('new', views.MockbusterCreate.as_view(), name='mockbuster_new'),
43     #path('view/<int:pk>', views.MockbusterView.as_view(), name='mockbuster_view'),
44     #path('edit/<int:pk>', views.MockbusterUpdate.as_view(), name='mockbuster_edit'),
45     #path('delete/<int:pk>', views.MockbusterDelete.as_view(), name='mockbuster_delete'),
46 ]
```

## Appendix 4.11:

```
urls.py | urls.py | views.py | home.html | Rent.html | Return.html | NewRental.html | models.py | Customer.html | base.html | admin.py | v
1  {% block content %}
2  {% if user.is_authenticated %}
3
4  <h1>Customers</h1>
5
6  <p><a href="{% url 'home' %}">Home</a></p>
7  <p><a href="{% url 'logout' %}">Logout</a></p>
8  <p></p>
9
10 <p><a href="{% url 'NewCustomer' %}">New Customer</a></p>
11 <table border="1">
12 <thead>
13 <tr>
14 <th>Customer</th>
15 <th>Phone Number</th>
16 <th>Address</th>
17 <th>Update</th>
18 <th>Delete</th>
19
20 </tr>
21 </thead>
22 <tbody>
23 {% for customer in object_list %}
24 <tr>
25 <td>{{ customer.cu_first_name }} {{ customer.cu_last_name }}</td>
26 <td>{{ customer.cu_phone_num }}</td>
27
28 <td>{{ customer.cu_address }}</td>
29 <td><a href="{% url 'UpdateCustomer' customer.cu_id %}">Update</a></td>
30 <td><a href="{% url 'DeleteCustomer' customer.cu_id %}">Delete</a></td>
31 </tr>
32 {% endfor %}
33 </tbody>
34 </table>
35 {% else %}
36 <p>You are not logged in</p>
37 <a href="{% url 'login' %}">login</a>
38
39
```

Hyper Text Markup Language file      length : 932    lines : 41      Ln : 21    Col : 10    Sel : 0 | 0      Windows (CR LF)    UT

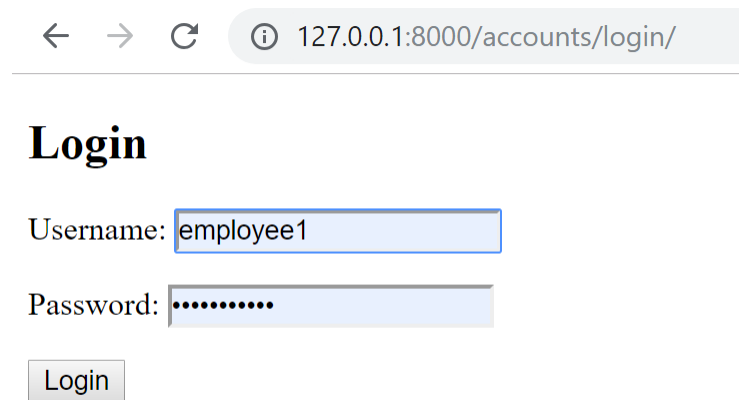
## Appendix 4.12:



You are not logged in

[login](#)

#### Appendix 4.13:



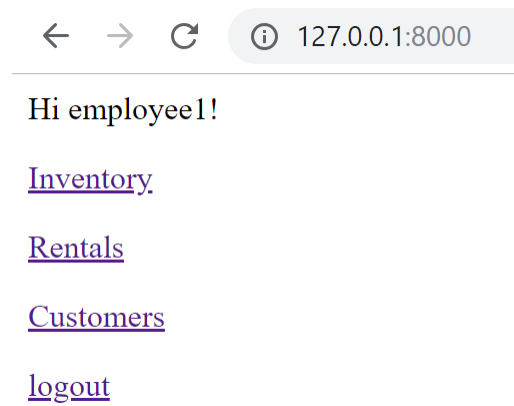
← → ↻ ⓘ 127.0.0.1:8000/accounts/login/

## Login

Username:

Password:

#### Appendix 4.14:



← → ↻ ⓘ 127.0.0.1:8000

Hi employee1!

[Inventory.](#)

[Rentals](#)

[Customers](#)

[logout](#)

Appendix 4.15:

← → ↻ ⓘ 127.0.0.1:8000/mockbuster/Customer/

# Customers

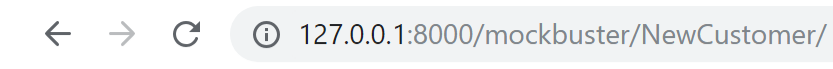
[Home](#)

[Logout](#)

[New Customer](#)

Customer	Phone Number	Address	Update	Delete
Greg Bowman	888-888-8888	106 Yeetville Rd, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Rita Skita	887-887-8887	107 Yetterville Ave, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Bill Bo	877-877-8877	207 Teryville Ave, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Sarah Silva	777-777-7777	257 Everyville st, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Banya Oster	778-778-7778	27 Villa rd, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Barry Stein	788-788-7788	71 Vanilla Way, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Lu Bu	686-768-7768	664 Manilla St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Dong Zhuou	486-468-7448	463 Chin Yang St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Zhuge Liang	386-438-7433	43 Tsu Yang St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Arnold Ang	383-638-6433	63 Yang St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Gerold Tulng	333-338-3433	3 Bang St, Maryland, NY, 12116	<a href="#">Update</a>	<a href="#">Delete</a>
Ronald Gungnir	363-638-3463	6 Bang St, Maryland, NY, 12116	<a href="#">Update</a>	<a href="#">Delete</a>
Ron Nirzche	663-636-6466	36 Old Rd, Maryland, NY, 12116	<a href="#">Update</a>	<a href="#">Delete</a>

Appendix 4.16:



# New Customer

Cu first name:

Cu last name:

Cu phone num:

Cu address:

[Cancel](#)



Appendix 4.17:

← → ↻ ⓘ 127.0.0.1:8000/mockbuster/Customer/

# Customers

[Home](#)

[Logout](#)

[New Customer](#)

Customer	Phone Number	Address	Update	Delete
Greg Bowman	888-888-8888	106 Yeetville Rd, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Rita Skita	887-887-8887	107 Yetterville Ave, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Bill Bo	877-877-8877	207 Teryville Ave, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Sarah Silva	777-777-7777	257 Everyville st, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Banya Oster	778-778-7778	27 Villa rd, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Barry Stein	788-788-7788	71 Vanilla Way, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Lu Bu	686-768-7768	664 Manilla St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Dong Zhuou	486-468-7448	463 Chin Yang St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Zhuge Liang	386-438-7433	43 Tsu Yang St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Arnold Ang	383-638-6433	63 Yang St, Troy, NY, 12180	<a href="#">Update</a>	<a href="#">Delete</a>
Gerold Tulng	333-338-3433	3 Bang St, Maryland, NY, 12116	<a href="#">Update</a>	<a href="#">Delete</a>
Ronald Gungnir	363-638-3463	6 Bang St, Maryland, NY, 12116	<a href="#">Update</a>	<a href="#">Delete</a>
Ron Nirzche	663-636-6466	36 Old Rd, Maryland, NY, 12116	<a href="#">Update</a>	<a href="#">Delete</a>
Alejandro Moon	605-104-3333	123 Wherever	<a href="#">Update</a>	<a href="#">Delete</a>

Appendix 4.18:

← → ↻ ⓘ 127.0.0.1:8000/mockbuster/Rent/

# Rentals

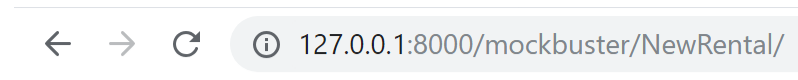
[Home](#)

[Logout](#)

[New Rental](#)

Customer	Date Rented	Date Due	Game	Branch	Return
Bill Bo	May 3, 2019	May 8, 2019	God of Doors	1	<a href="#">Return</a>
Sarah Silva	May 3, 2019	May 8, 2019	God of Doors	1	<a href="#">Return</a>
Banya Oster	May 3, 2019	May 8, 2019	Civilized Discourse	1	<a href="#">Return</a>
Barry Stein	May 3, 2019	May 8, 2019	Civilized Discourse	1	<a href="#">Return</a>
Lu Bu	May 3, 2019	May 8, 2019	Dark Soles	1	<a href="#">Return</a>
Dong Zhuou	May 3, 2019	May 8, 2019	Dark Soles	1	<a href="#">Return</a>
Zhuge Liang	May 3, 2019	May 8, 2019	Big Snek	2	<a href="#">Return</a>
Arnold Ang	May 3, 2019	May 8, 2019	Big Snek	2	<a href="#">Return</a>
Gerold Tulng	May 3, 2019	May 8, 2019	Pretty Intense Soccer 2019	2	<a href="#">Return</a>
Ronald Gungnir	May 3, 2019	May 8, 2019	Pretty Intense Soccer 2019	2	<a href="#">Return</a>

Appendix 4.19:



# New Rental

Vg:

Cu:

Br:

Em:

[Cancel](#)

Appendix 4.20:

← → ↻ ⓘ 127.0.0.1:8000/mockbuster/Rent/

# Rentals

[Home](#)

[Logout](#)

[New Rental](#)

Customer	Date Rented	Date Due	Game	Branch	Return
Bill Bo	May 3, 2019	May 8, 2019	God of Doors	1	<a href="#">Return</a>
Sarah Silva	May 3, 2019	May 8, 2019	God of Doors	1	<a href="#">Return</a>
Banya Oster	May 3, 2019	May 8, 2019	Civilized Discourse	1	<a href="#">Return</a>
Barry Stein	May 3, 2019	May 8, 2019	Civilized Discourse	1	<a href="#">Return</a>
Lu Bu	May 3, 2019	May 8, 2019	Dark Soles	1	<a href="#">Return</a>
Dong Zhuou	May 3, 2019	May 8, 2019	Dark Soles	1	<a href="#">Return</a>
Zhuge Liang	May 3, 2019	May 8, 2019	Big Snek	2	<a href="#">Return</a>
Arnold Ang	May 3, 2019	May 8, 2019	Big Snek	2	<a href="#">Return</a>
Gerold Tulng	May 3, 2019	May 8, 2019	Pretty Intense Soccer 2019	2	<a href="#">Return</a>
Ronald Gungnir	May 3, 2019	May 8, 2019	Pretty Intense Soccer 2019	2	<a href="#">Return</a>
Greg Bowman	May 3, 2019	May 8, 2019	God of Doors	1	<a href="#">Return</a>

## Appendix 4.21:



Hi james!

[Inventory](#).

[Rentals](#)

[Customers](#)

[Admin Page](#)

[logout](#)

## Appendix 4.22:

A screenshot of the Django administration interface. At the top, a browser address bar shows the URL '127.0.0.1:8000/admin/'. Below this is a dark blue header bar with the text 'Django administration' in yellow. The main content area is titled 'Site administration' and contains two primary sections. The first section, 'AUTHENTICATION AND AUTHORIZATION', lists 'Groups' and 'Users', each with a green '+ Add' button and a yellow pencil 'Change' button. The second section, 'MOCKBUSTERAPP', lists various models: 'Branchs', 'Customers', 'Employees', 'Managers', 'Purchase hists', 'Rental hists', 'Rentals', 'Vendors', 'Video details', and 'Video games'. Each model entry also has a green '+ Add' button and a yellow pencil 'Change' button. On the right side of the interface, there is a sidebar titled 'Recent actions'. It contains a sub-header 'My actions' followed by two entries, each consisting of a yellow pencil icon, the text 'employee1', and the word 'User' below it.

## Appendix 4.23:

Django administration

WELCOME, JAMES. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home › Mockbusterapp › Employees › Add employee

### Add employee

Em first name:	<input type="text" value="Tobias"/>
Em last name:	<input type="text" value="Manne"/>
Em pay rate:	<input type="text" value="10.00"/>
Em phone num:	<input type="text" value="654-654-6666"/>
Em dob:	<input type="text" value="2000-05-03"/> Today
Note: You are 4 hours behind server time.	
Em ss num:	<input type="text" value="605-88-8512"/>
Em address:	<input type="text" value="105 Whest La, Rooberville, OH, 11111"/>
Br:	<div>Branch object (1)   </div>

Save and add another

Save and continue editing

SAVE

## Appendix 4.24:

Django administration

WELCOME, JAMES. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home › Authentication and Authorization › Users › Add user

### Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Username:

tobias

Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Password:

.....

Your password can't be too similar to your other personal information.  
Your password must contain at least 8 characters.  
Your password can't be a commonly used password.  
Your password can't be entirely numeric.

Password confirmation:

.....

Enter the same password as before, for verification.

Save and add another

Save and continue editing

SAVE

Appendix 4.25:

☐ Superuser status

Designates that this user has all permissions without explicitly assigning them.

Groups:

Available groups

Q

Filter

Choose all

Chosen groups

Remove all

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

User permissions:

Available user permissions

Q

Filter

admin | log entry | Can add log entry

admin | log entry | Can change log entry

admin | log entry | Can delete log entry

admin | log entry | Can view log entry

auth | group | Can add group

Choose all

Chosen user permissions

mockbusterApp | customer | Can add customer

mockbusterApp | customer | Can change customer

mockbusterApp | customer | Can delete customer

mockbusterApp | customer | Can view customer

mockbusterApp | rental | Can add rental

mockbusterApp | rental | Can change rental

mockbusterApp | rental | Can delete rental

mockbusterApp | rental | Can view rental

Remove all

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

Important dates

Last login:

Date:

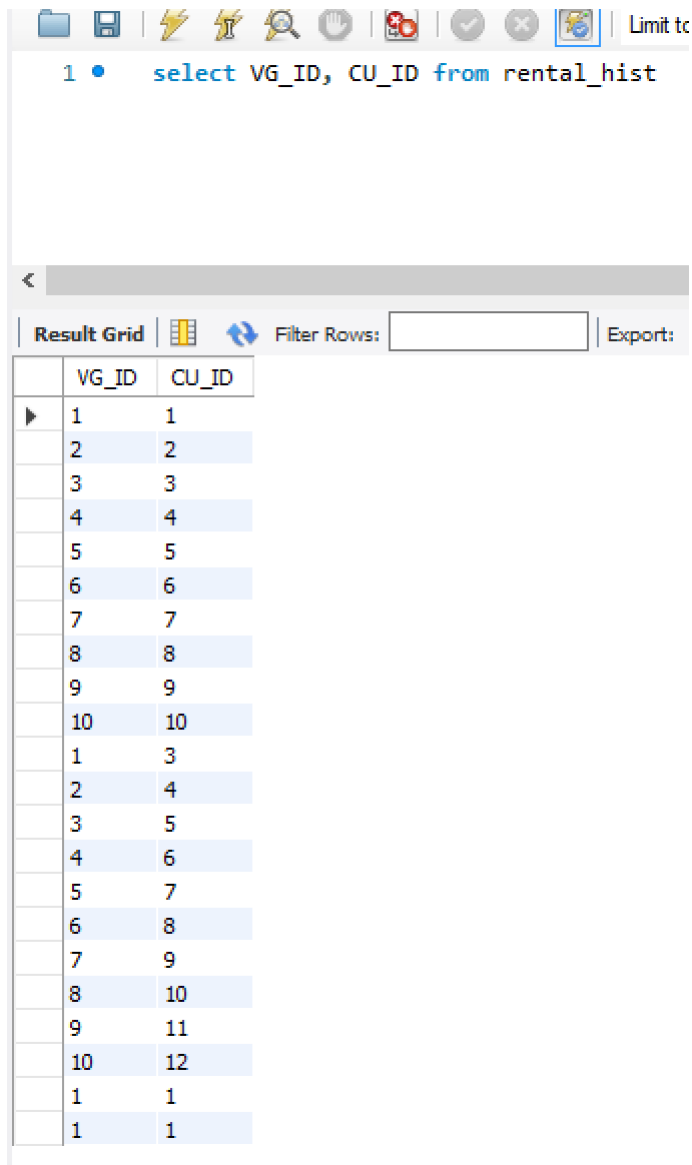
Today

Time:

Now



Appendix 4.26:



The screenshot shows a SQL query editor interface. At the top, there is a toolbar with various icons for file operations, execution, and navigation. Below the toolbar, a SQL query is entered in a text area:

```
1 • select VG_ID, CU_ID from rental_hist
```

Below the query area, there is a section for the results. It includes a "Result Grid" tab, a "Filter Rows:" input field, and an "Export:" button. The results are displayed in a table with two columns: "VG\_ID" and "CU\_ID".

	VG_ID	CU_ID
▶	1	1
	2	2
	3	3
	4	4
	5	5
	6	6
	7	7
	8	8
	9	9
	10	10
	1	3
	2	4
	3	5
	4	6
	5	7
	6	8
	7	9
	8	10
	9	11
	10	12
	1	1
	1	1

Appendix 4.27:

```

1 • SELECT t1.CU_ID,t4.CU_FIRST_NAME,t4.CU_LAST_NAME, t1.vg_id, t3.vd_name
2       FROM rental_hist as t1
3       left JOIN video_game as t2 ON t1.vg_id = t2.vg_id
4       left join video_detail as t3 on t2.vd_id = t3.vd_id
5       left join customer as t4 on t1.cu_id = t4.cu_id;

```

<div> <div>&lt;</div> <div>Result Grid</div> <div> <div></div> <div></div> </div> <div>Filter Rows: <input type="text"/></div> <div>Export: <div></div></div> <div>Wrap Cell Content: <div></div></div> </div>					
	CU_ID	CU_FIRST_NAME	CU_LAST_NAME	vg_id	vd_name
▶	1	Greg	Bowman	1	God of Doors
	2	Rita	Skita	2	God of Doors
	3	Bill	Bo	3	Civilized Discourse
	4	Sarah	Silva	4	Civilized Discourse
	5	Banya	Oster	5	Dark Soles
	6	Barry	Stein	6	Dark Soles
	7	Lu	Bu	7	Big Snek
	8	Dong	Zhuou	8	Big Snek
	9	Zhuge	Liang	9	Pretty Intense Soccer 2019
	10	Arnold	Ang	10	Pretty Intense Soccer 2019
	3	Bill	Bo	1	God of Doors
	4	Sarah	Silva	2	God of Doors
	5	Banya	Oster	3	Civilized Discourse
	6	Barry	Stein	4	Civilized Discourse
	7	Lu	Bu	5	Dark Soles
	8	Dong	Zhuou	6	Dark Soles
	9	Zhuge	Liang	7	Big Snek
	10	Arnold	Ang	8	Big Snek
	11	Gerold	Tuling	9	Pretty Intense Soccer 2019
	12	Ronald	Gungnir	10	Pretty Intense Soccer 2019
	1	Greg	Bowman	1	God of Doors
	1	Greg	Bowman	1	God of Doors

## Appendix 4.28:

The screenshot shows a SQL editor interface. At the top, there is a toolbar with various icons for file operations, search, and execution. Below the toolbar, the SQL query is entered in a text area: `select * from rental where cu_id = 4;`. The query is highlighted in blue. To the right of the query, there is a dropdown menu set to "Limit to 1000 rows".