Inventory Control/Order Fulfillment Database

Emily Foley, Cole Nelson, Debsankar Mukhopadhyay

University of Missouri – Kansas City

Author Note

Group Project Final Report for CS470: Introduction to Database Management Systems

Abstract

A database schema has been developed for a fictitious inventory control and order fulfillment database. A step by step study has been performed to develop the Entity Relationship Diagram, Schema Diagram, Data Definition Language (DDL), Data Manipulation Language (DML), and a simple user interface based on a set of well-defined requirements and constraints. A simple approach to database security has also been presented.

Keywords: Database Management System; Entity Relationship Diagram; Schema Diagram, Requirements; Database Security, DDL; DML.

Introduction

The database system chosen by our group is an inventory pickup/management system. The three-tiered database system we are creating will aid inventory/pickup system employees by keeping records of product inventory quantities and their locations. This MySQL database will guide employees throughout the process of packing totes and shipping to the correct location and time, while also keeping track of inventory as the product is picked and shipped from the warehouse.

Requirements

The employee/order picker clocks in and logs into the tablet with their employee id. The employee has a name, id number, start date, and performance percentage score. Each tablet also has a number. On the tablet, they select their sled number and then scan in the labels for the assignments they will be picking (usually 4 at a time but can do 1-6 assignments). (The sled is what is used to pull the totes along on rollers as you pick from the shelves. Looks a little bit like this picture but with a tablet attached and smaller plastic totes:)

The system requirements can be stated as follows:

1.   An employee should be identified by a unique employee ID.

2.   An employee must sign on one device.

3.   A device may or may not be assigned to an employee.

4.   An employee should select one sled.

5.   A sled may or may not be assigned to any employee.

6.   There must be 1-6 assignments per sled/per employee. But a sled may or may not have any assignments.

7.   Each assignment can have 1 to N number of totes.

8.   Each tote can contain 1 to N number of products.

9.   Each assignment should contain a product, number of items for that product, start and destination location for the product.

10.   Each tote is uniquely identified by a label.

The labels show the batch number, store number, current tote number and total number of totes for that assignment, lane number (only used for shipping), # of items in current tote,

department and beginning and ending location number of items in current tote, and label number. Batch and store number are the same for all labels in that assignment. Other numbers are different for each tote/label (one label per tote).

Department refers to the section/aisle in the store where the items will be placed, such as: First Aid, Candy, Cosmetics, Snacks, Batteries, Gift Cards, Hair Care, etc. etc.

Once assignments are scanned in and started, the tablet displays the location number and number of items to pick for each product. The picker is able to decrease the number or zero it out if there is limited or no inventory available (hasn't been restocked).

Products are stored on shelves with 4-6 rows. Shelves are labeled with location stickers (one per product). Each location has the location number which is formatted like this example: A2-44-55-5. The first (A2) refers to the section. There are 3 "mods": AB, CD, and EF. Each mod has 4 floors. Each floor has two sides (for example, the A side and the B side). So A2 refers to the AB mod, 2nd floor, A side. The second number is the bay number. I believe each section has 44 bays (so 01-44 is the range for bays). The third number is the location from left to right/right to left (kind of like the column). It goes by 5's and not every number is present on the shelf (it just depends on how big the cases are and lots of other factors like things being removed and added) but the lowest will be 5 and the highest will be 80 (usually not that high unless the items are all really small like cosmetics). The last number is the shelf number (most bays have 5 rows of shelves; some have 4 and some 6. The shelf lowest to the ground is 1 and goes up.

Each location also has the product name, item number, number of pieces per case, weight (in oz) (of one piece), expiration date (if applicable), and UPC. As pieces are picked from the case, the inventory decrements, and as cases are added to the shelf, inventory increments by the number of cases x the number of pieces per case. The system determines the number of items to go in each

tote and the total number of totes needed for the assignment by the overall weight and volume of all the items ordered. So along with weight each item also has length, width, and height in inches. The totes are 19"L x 13.5"W x 8.5"H. The weight limit is about 20lbs.

Each assignment has a "standard" amount of time it is predicted to take to finish based on how many items, number of totes, and many other calculated factors. The actual amount of time the employee takes to complete all assignments compared with the sum of the predicted times provides the employee's performance percentage for the day. So, employee start date would only be relevant if we want to figure out performance other than daily, like weekly, monthly, year-to-date, etc.

A hypothetical real-world system of this kind can be envisioned as shown in Fig 1.

Figure 1: The real-world scenario of the inventory pickup/placement management system.
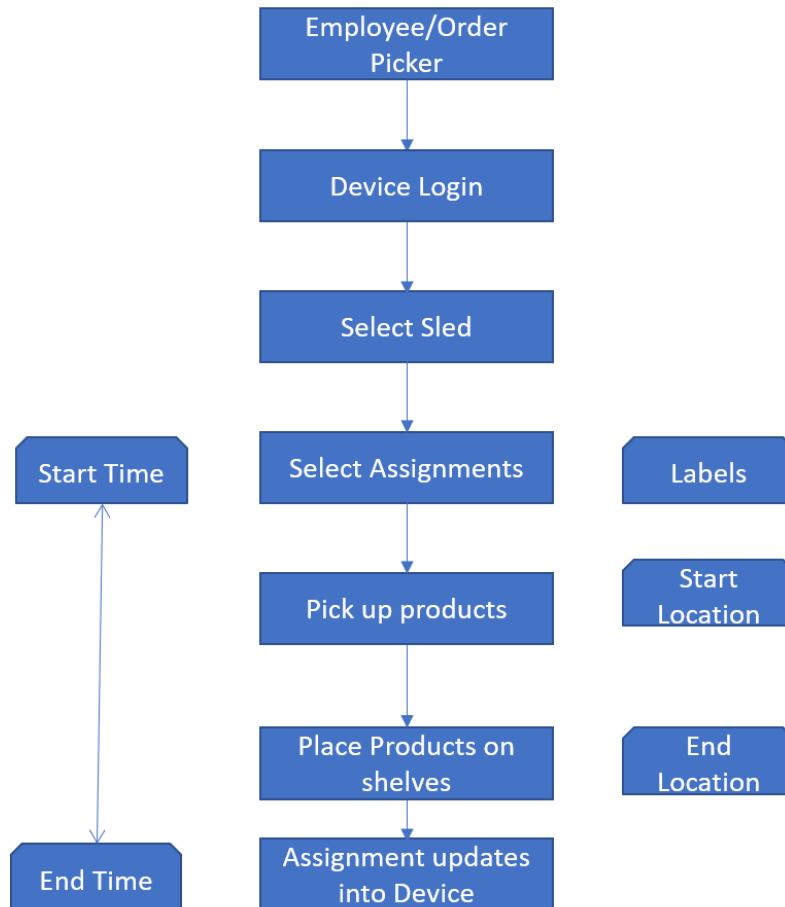
A basic flow of the activities shown in Fig 2.

Figure 2: A simplified workflow of the system.

Architecture

We will follow a simple three-tier Architecture (Fig 3) with a Database Server, Application
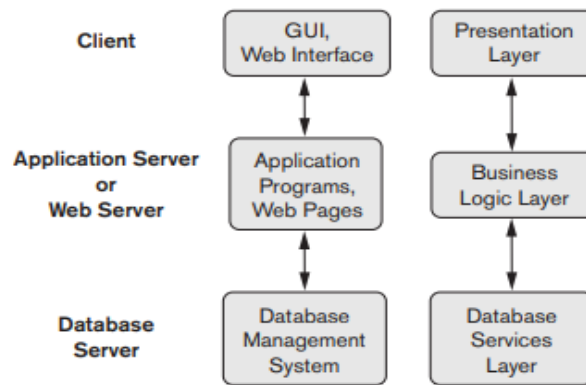
Server and Client tier.

Figure 3: System Architecture

The presentation layer is a simple python program that interacts with the underlying database through an appropriate driver. The business logic is embedded in the python code as a service. The database server is the main focus of this project. We have used a SqlLite (DQLite Home Page, n.d.) database server that contains the interbnal and conceptual level or schema. The external schema, however, shows the entire mini-world of the order pick up management system. By using a three-schema structure we are ensuring both logical and physical data independence.

Scopes and Constraints

Our project is restricted to the mini-world of the order pickup management system to meet the above requirements. We used a relational database management system to solve the problems in this project. We utilize DBMS Software specific utilities (like sequence generators etc.) to specify the primary keys. The constraints on the entities and relationships are enforced in the Data Definition Language Script that creates the physical schema. Our effort on the presentation and middle tiers are minimal. We have not addressed any system level performance or scaling issues. Our performance tuning is limited to basic RDBMS indexing, efficient joins, and optimized queries. The constraints are summarized in Table 1.

Table 1: List of constraints

EMPLOYEE	ey	ull	Type	Notes	Constraints
Employee_ID			int (7)		
First_Name			char (30)		
Last_Name			char (30)		
DEVICE					
Device_ID			XNNN		
SLED					
Sled_ID			int (2)		
ASSIGNMENT					
Assignment_ID			int (10)		
TOTE					
Label			composite		
Batch_Num			int (3)		100 to 999
Tote_Num			int (2)		1 to 99
N_Totes			int (2)	should be derived attribute of relationship CONTAINS rather than of TOTE? (because derived	1 to 99

				from number of TOTES that ASSIGNMENT CONTAINS)	
Store_Num			int (5)		10000 to 99999
Lane_Num			int (4)	can remove, I'm not really sure how this one works anyway	
Start_Locati on			XN-NN-NN-N		letter is A-F, N1 is 1-4, N2&N3 is 01-44, N4&N5 is 05-80 by 5's, N6 is 1-6
End_Locatio n			XN-NN-NN-N		letter is A-F, N1 is 1-4, N2&N3 is 01-44, N4&N5 is 05-80 by 5's, N6 is 1-6
PRODUCT					
N_Products			int (3)	derived attribute of relationship TOTE_HAS_PROD T (number of products of that type that TOTE has)	1-999
PRODUCT_ TYPE					
Product_ID			int (6)		100000-999999

Name			char (30)		
Location			XN-NN-NN-N		letter is A-F, N1 is 1-4, N2&N3 is 01-44, N4&N5 is 05-80 by 5's, N6 is 1-6
Weight			int (3) (ounces)	optional but if we want to do weight and dimension constraints--then we should also have derived attributes tote weight and tote dimensions for relationship TOTE_HAS	Total tote weight limit is 30lbs aka 480oz
Dimensions			NNxNNxNN (inches)	or split as length, width, height	Total tote dimension limits are 19"L x 13.5"W x 8.5"H

Entity Relationship Diagram

We have defined following entities:

1. Device,

2. Employee,

3. Sled,

4. Assignment,

5.   Tote,

6.   Product_Type.

We have simplified our mini-world in many ways compared to an actual inventory management system. For example, we did not include any cost information, employee's historic performance records, and any shipment related information.

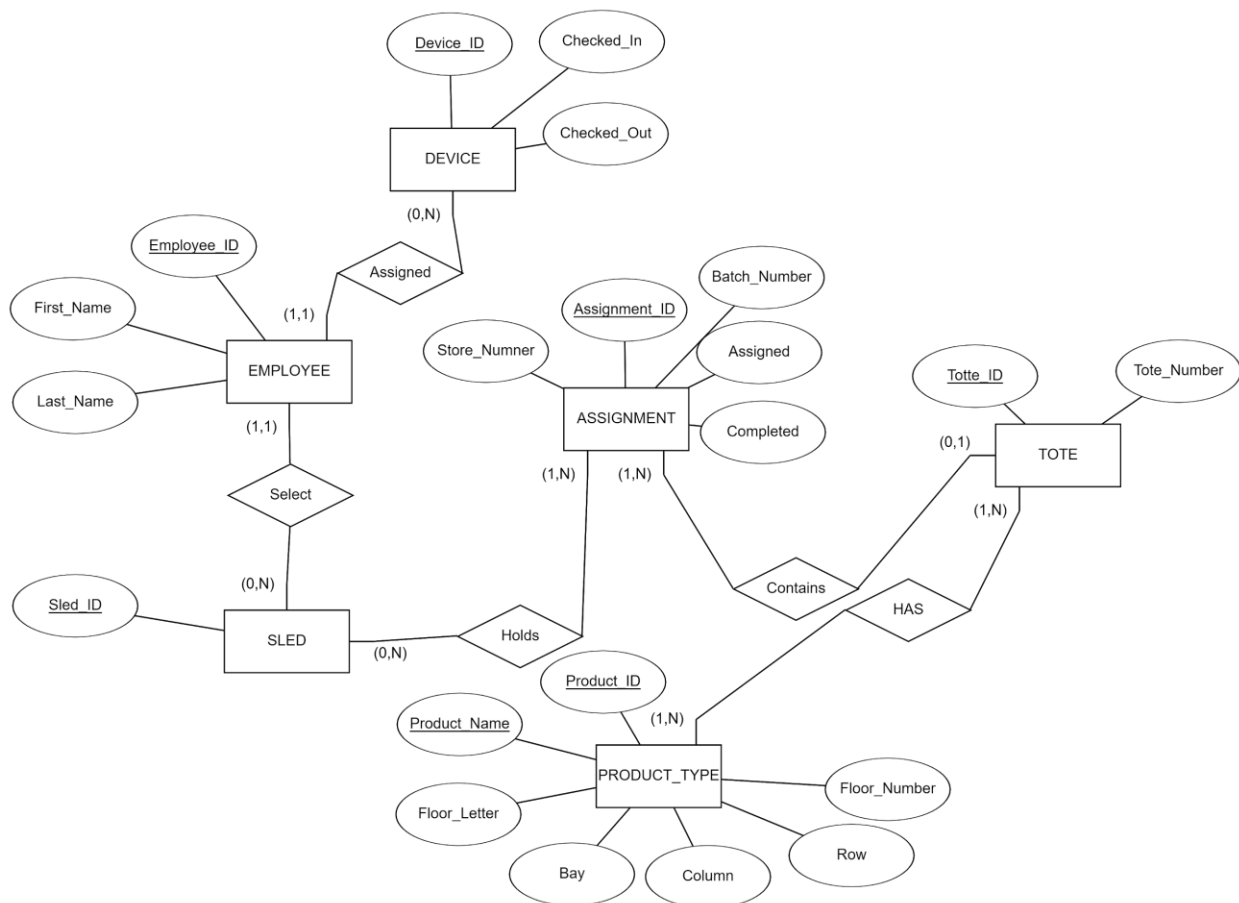The Entity-Relationship diagram is shown in Fig 4.

Figure 4: Entity-Relationship Diagram for an order pick-up/inventory management system.

     The Entity-Relationship diagram has been evolved from our initial set of diagrams through a requirement adjustment, and database normalization process.

We converted multi-valued and composite attributes either into separate schemas or included or mapped them into functionally appropriate entities. As a result, our schema passed first, second, and third normalization tests seamlessly.

Schema Diagram

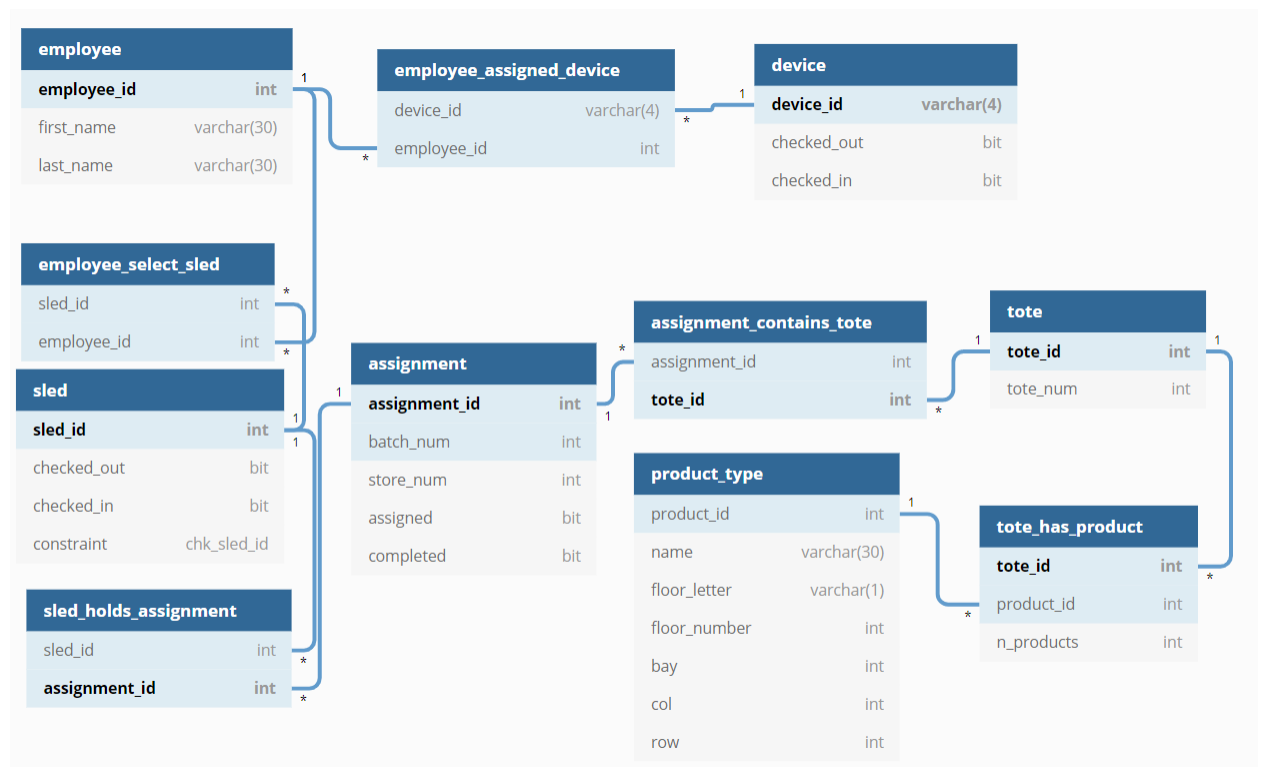The schema diagram is displayed in Fig 5.

Figure 5: Schema Diagram.

As a next step to create the physical schema using the DDLs, we worked out relational algebra for the core queries (Listing 1).

Listing 1: Relational Algebra for core queries

        Before step 1: all employee_ID's are inputted into EMPLOYEE

    1. A. Check if the employee_ID entered is found in EMPLOYEE

σ_Employee_ID = NNNNNNNN (EMPLOYEE)

B. If not found, an error message is displayed

C. Check if the ID entered, isn't already signed into a device

σ_Employee_ID = NNNNNNNN (ASSIGNED)

D. If employee is already signed into a device, an error message is

displayed.

E. If they aren't already signed into a device, then Emplyee_ID and

Device_ID are inserted into ASSIGNED

Before step 2: All Sled_ID's are inserted into SLED

2. A. Check if the ID entered is found in SLED
    σ_Sled_ID = <current sled> (SLED)

    B. if not found, error message is displayed

    Before step 3: will have inserted into ASSIGNMENT

3. A. Check if the ID entered, is found in ASSIGNMENT
    σ_Assignment_ID = <current assignment> (ASSIGNMENT)

    B. If not found, an error message is displayed.

    C. Check if the ID entered isn't already assigned to another sled

    σ_Assignment_ID = <current assignment> (HAS)

    D. if found, error message is displayed

    E. Check if the ID entered is in progress or completed

    F. σ_Assignment_ID = <current assignment> (σ_Completed = True

(ASSIGNMENT))

    G. If found, an error message is displayed.

H. for displaying the number Totes Required, sum of N_Totes for each

assignment on the current sled (I'm struggling with the relational algebra on this

one, it is fairly complex)

Π_cnt(

      I. After each assignment is entered, insert Assignment_ID and Label into

CONTAINS. If CLEAR is pressed, delete from CONTAINS (this will update the

number Totes Required as assignments are entered or cleared).

Before step 4: all Product_IDs are inserted into PRODUCT

4. Each part of the pick display will come from the database
      A. Each sled hold 1-6 assignments, the pick page displays each product

and its quantity

      B. matching TOTE_HAS_PRODT that each ASSIGNMENT CONTAINS by

LOCATION.

      C. If there are less than 6 assignments on a sled, the remaining assigned

slots are filled with empty boxes

      D. The number in each box is derived from the count of PRODUCTS of the

same Product_Type that TOTE has (TOTE_HAS_PROD)

      E. After the pick is complete assignments are automatically updated and

dropped from SLED (deleted from HAS relationship between sled and

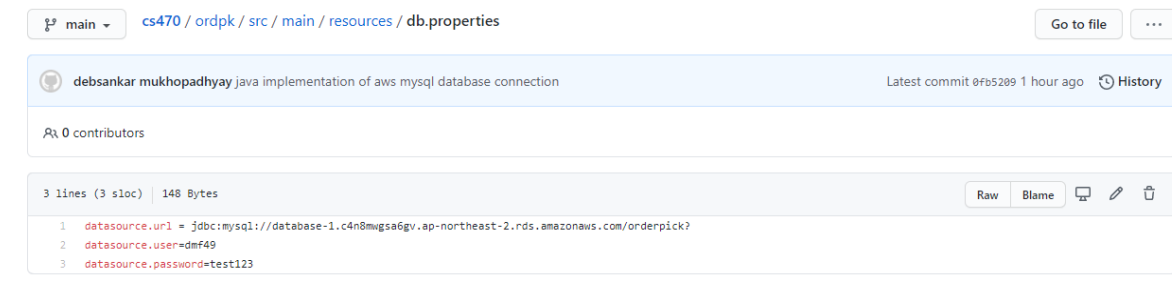assignment)

Database Security

The most basic front security measure used is an employee ID. The ID consists of 7 numbers. The ID is needed in order for an employee to log in to a device. The log in security measures are as follows, the employee cannot access a device if the employee's ID is not already in the system, the employee cannot log in to a device if they are already logged into another device. Employees will not be able to access the company database unless they are on a local company server. Once employee ID's are entered, the ID's will be encrypted and hashed in order to prevent data decryption and plain text storage in our database. All employee ID's will be encrypted using the **aes_encrypt('employee_ID', 'key1234')** function. The usage of this Advanced Encryption Standard (AES) (Akash & Beniwal, 2020) ensures employee_ID's cannot be displayed in our database. The function will change the input into random numbers, letters, and symbols. For Sql Injection defense, our database uses access limitations to only allow certain employees to access employee tables and employee data. The database also uses escape code to prevent malicious input**: Employee_id = 'something' Employee_id = Employee_id.replace("'", "\\'").** Our database also uses a secure backup to make sure no data is lost and use connectors to mitigate any damage done if the application is breached. Using a connector can help limit the access to data if there was a malicious data breach. This way we can store important data outside of our breached application and prevent further damage.

Login Input	Login Output
346366	"Employee_ID must be 7-digit number. Please try again."
45256A2	"Employee_ID must be 7-digit number. Please try again."

*Employee is logged into another device	"Error: This Employee_ID is already associated with another device. Please log off other device and try again."
3915663	*Employee successfully enters system and proceed to the next step of sled selection
SQL injection: 3915663' Escape code: 3915663//'	"Employee_ID must be 7-digit number. Please try again."

Employee ID Input	Employee ID Input After Encryption
3463674	A@e5$20%4fv$gb4#

In our AWS/Java implementation of the MySql/JDBC based implementation, preparedstatement interface has been used to prevent SQL Injection attacks and the user credentials are retrieved from a configuration file (Fig 6).


```
main ▾     cs470 / ordpk / src / main / resources / db.properties                    Go to file   ⋯

⬤ debsankar mukhopadhyay java implementation of aws mysql database connection        Latest commit 0fb5209 1 hour ago  🕐 History

👥 0 contributors

3 lines (3 sloc)   148 Bytes                                                          Raw   Blame   💻 ✎ 🗑

1   datasource.url = jdbc:mysql://database-1.c4n8mwgsa6gv.ap-northeast-2.rds.amazonaws.com/orderpick?
2   datasource.user=dmf49
3   datasource.password=test123
```

Figure 6: Java/JDBC/AWS MySql implementation configuration file is storing the credentials.

Deficiencies

Due to the learning organization structure in our project, we have designed a system in parallel with our educational evolution in the course. So, the first deficiency, is that our project execution did not structurally include the serialized processes deemed as best practices to avoid poor designs (Mmekut, 2018). The database design did not include any purchase and sell accounting system. There was no way to define a purchase system. The purchase system, in theory,

could be accomplished by extending the product_type entity. The inventory placement could be categorized by departments and zones of the stores. In the original requirements, that was specified through the labels. In our design, we simplified it with an assumption that the products are placed on certain bays, floors, rows and columns. So, this design may not be scalable for a big store or warehouses. We have not included any option to shipments and receiving processes. That will prevent our system to be easily integrated with a logistic system.

<center>**Future Scope**</center>

AWS Implementation

We have used a free AWS tier for doing a proof of concept for a mysql database server implementation (Fig 7).
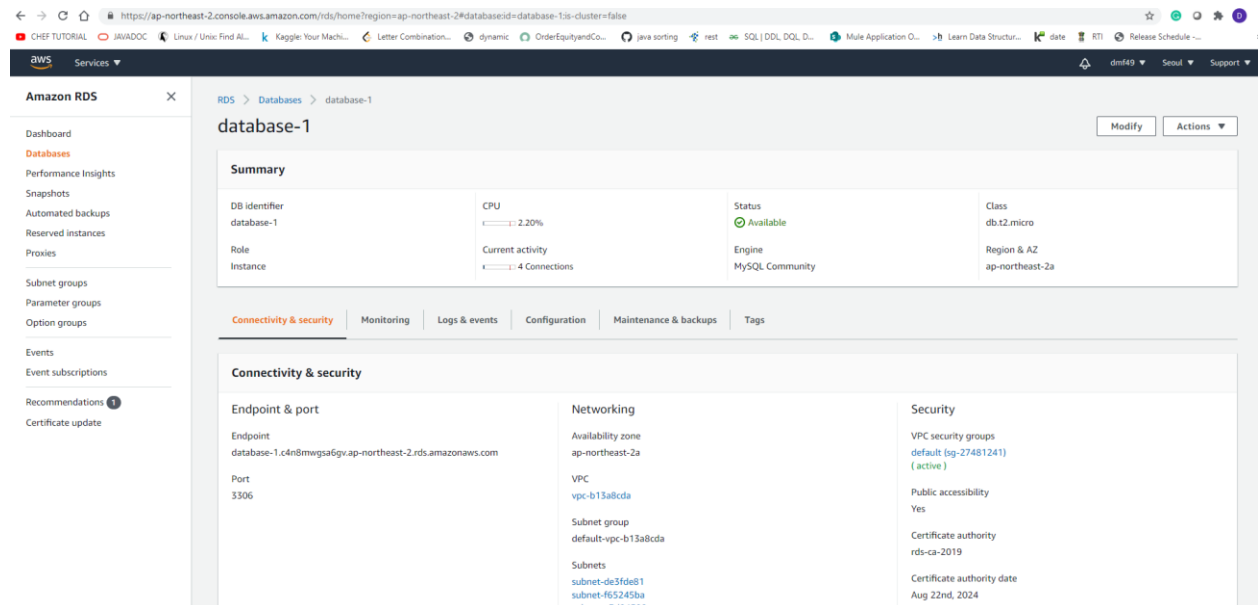

<center>Figure 7: AWS Free Tier RDS dashboard for MySql instance</center>

We have connected to the AWS Mysql RDS instance using a local Sql Developer IDE using following connection parameters:

Endpoint & port

Endpoint: database-1.c4n8mwgsa6gv.ap-northeast-2.rds.amazonaws.com

Port: 3306

Once connected, we have implemented a Java code to connect to that database and utilize JDBC's "PreparedStatement" interface to determine the number of a given product on a Tote with information of the store number and whether that has been assigned to anyone. A reference implementation can be found in our project's GitHub repository (Foley, Nelson, & Mukhopadhyauy, 2020).

It is to emphasize that we have used a PreparedStatement interface in our Java code to pass the parameters to the database. This process is adapted to prevent any sql injection (Fig 8).

```
try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        //String mysqlConnServ = "database-1.c4n8mwgsa6gv.ap-northeast-2.rds.amazonaws.com";
        connection = DriverManager.getConnection(prop.getProperty("datasource.url")+ "user="+ prop.getProperty("datasource.user") + "&password=" + prop.getProp

        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setString(1, productName);
    resultSet = ps.executeQuery();

        System.out.println("Searching the product, store_num, and tote number...." );

        while (resultSet.next()) {
                int storeNum = resultSet.getInt("store_num");
                int toteNum = resultSet.getInt("tote_num");
                int count = resultSet.getInt("CNT");
                Boolean assigned = resultSet.getBoolean("assigned");

                System.out.println("storeNum = " + storeNum + " tote Number=" + toteNum + ", count =  " + count + ", is assigned = " + assigned.booleanValue())
        }
```

Figure 8: Code snippet showing jdbc connection handling to prevent SQL Injection.

RDS utility in AWS comes with features to create regular backup of our database. AWS identifies backups as snapshots. We have created a manual data backup or manual snapshot as shown in the screenshot in Fig 9-10.
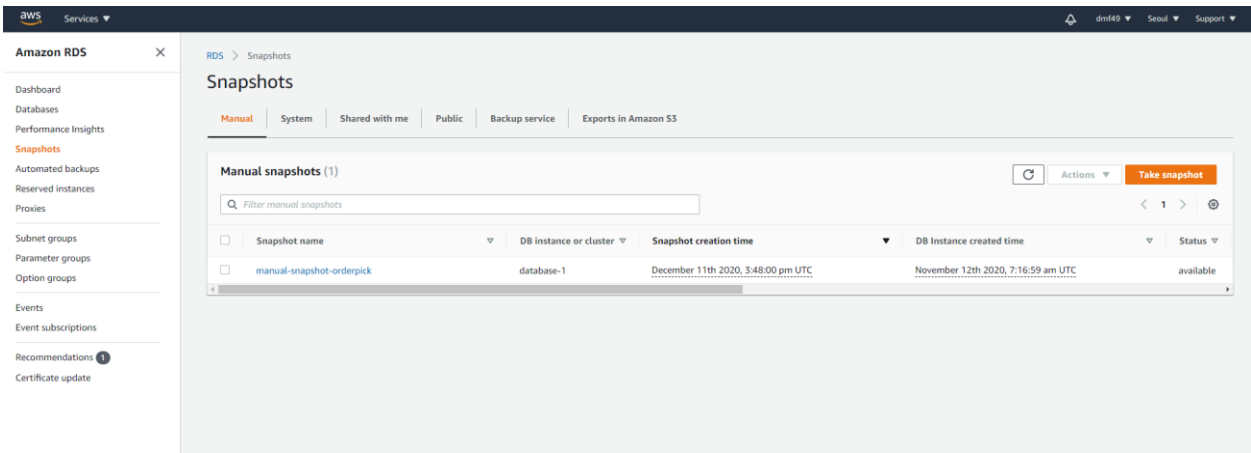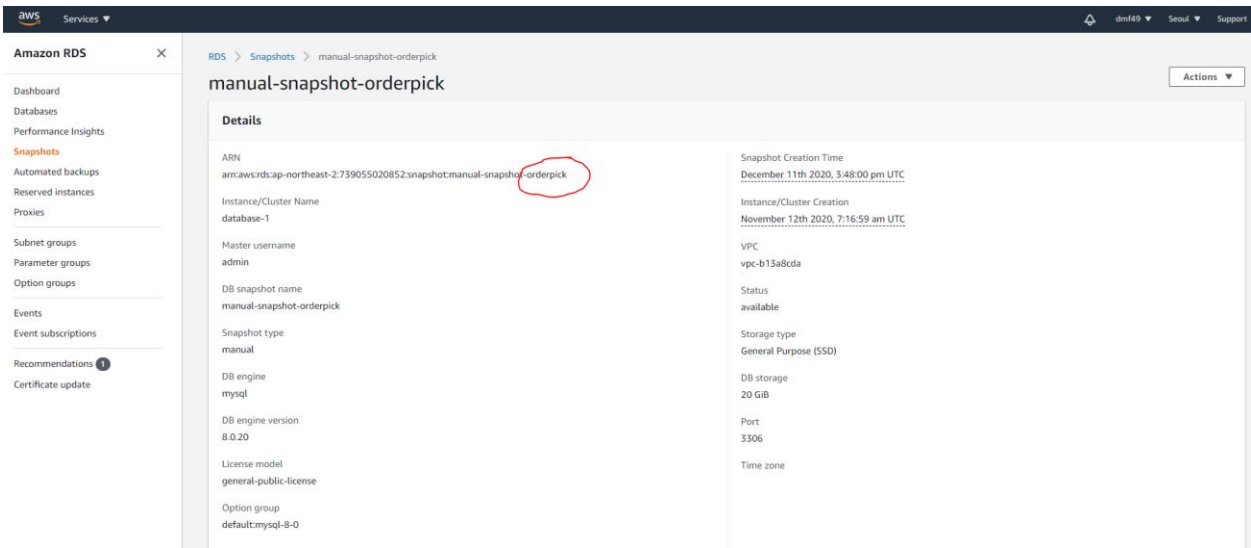
Figure 9: Database backup at the AWS.

Figure 10: Backup details

Functional Opportunities

We have opportunities to extend our current design to include departments, register locations, inventory purchase and sell systems. We will need to identify the possible relational aspects with those entities with the currently included entities. It may be meaningful to expand the integrability of this schema by defining possible relation tables with potential entities available for

integration in future. We will have opportunity to extend the current schema to NoSQL type of design to accommodate scalable analytics development.

References

Akash, & Beniwal, D. (2020). ENHANCEMENT IN AES ALGORITHM. *International*

   Research Journal of Engineering and Technology (IRJET), 7(4), 1435 - 1439.

DQLite Home Page. (n.d.). Retrieved from https://www.sqlite.org/index.html

Foley, E., Nelson, C., & Mukhopadhyauy, D. (2020, December). Retrieved from

   https://github.com/dmf49/cs470

Mmekut, E. (2018, October 8). *How To Avoid Poor Design /Planning In Database Design*.

   Retrieved from https://medium.com/@mmekutmfonedet/how-to-avoid-poor-design-

   planning-in-database-design-798fa606fb31