CS/DSA-4513-001 – Fall 2018

Dr. Le Gruenwald

Clayton Glenn

113375641

Glen5641@ou.edu

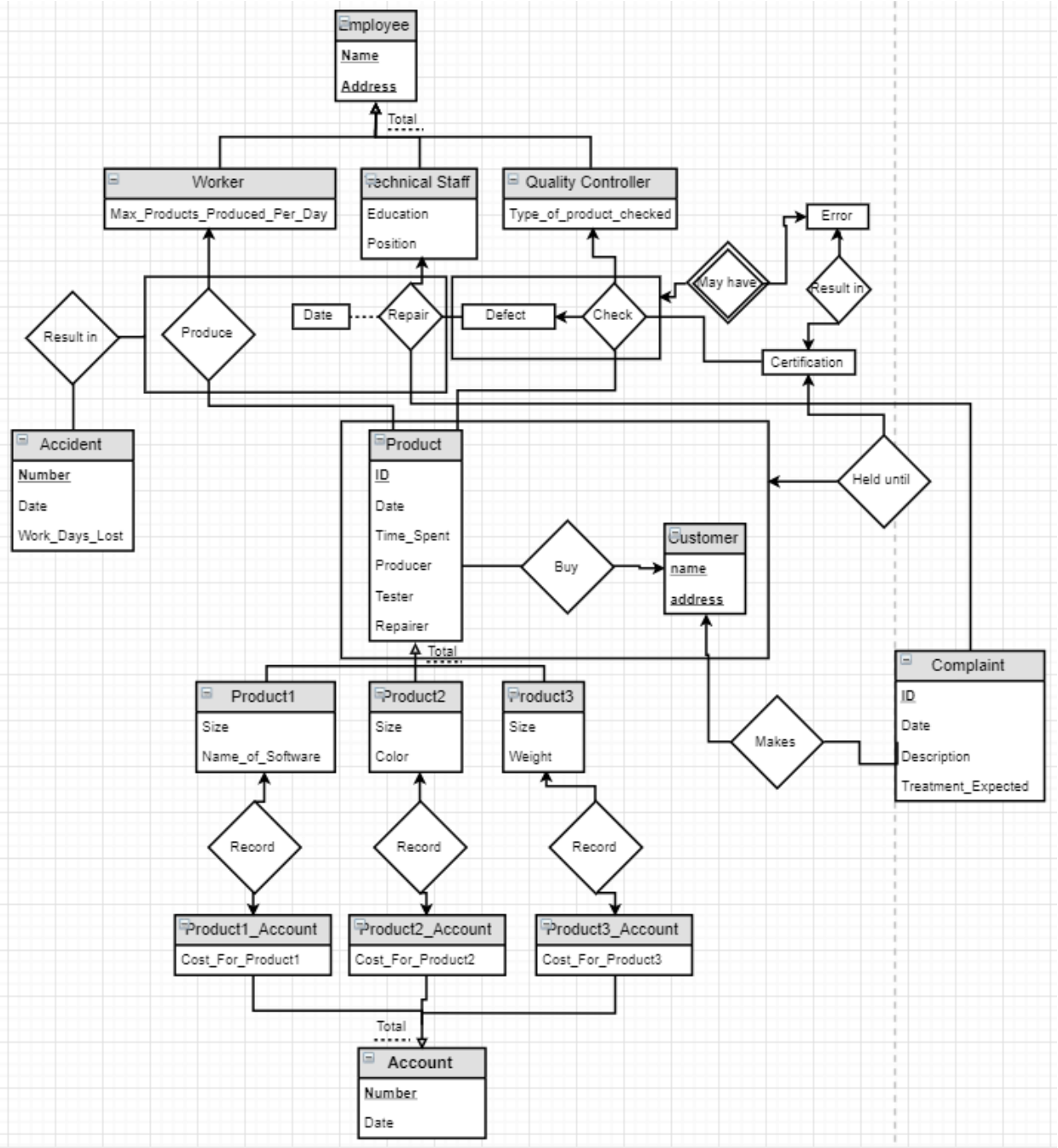
Future, INC Systems Database

Individual Project

Table of Contents

Task 1.

1.1) ER Diagram



1.2. Relational Database Schema

1.2.1. Description of Future, Inc Database System

The FUTURE, Inc. has three different types of employees: technical staff, quality controller and worker.
Each employee has a unique name, and address.
A technical staff has an education record indicating the degrees he/she obtained (BS, MS, Ph.D) and technical position.
For each quality controller, the company records the type of the product he/she will check.
Each controller can check only one type of product.
The company records the maximum number of products a worker can produce per day.
Workers are responsible for making the products.
Quality controllers are responsible for testing the quality of the products.
Technical staffs will fix the problems on products.
The products will not be allowed to get out of the company without the certification of a quality controller.
However, a defected product may get out due to an error made by a quality controller.
Each product is produced, checked and repaired by one worker, technical staff, and quality controller, respectively.
Each product is assigned a unique product ID.
The following information about a product should also be known:
Date the product is produced;
Time spent to make the product;
The person who produced the product;
The person who tested the product and the person who repaired the product if the product has been repaired.
There are three different types of products: product1, product2, product3.
For product1, the size (small, medium, large) of the product and name of the major software used will be recorded.
For product2, the size and color of the product will be recorded.
For product3, the size and weight of the product will be recorded.
If a product1 has any problem, only a technical staff who has graduate education can repair it.
For the other products, any technical staff can repair it.
An account is maintained by the company to keep track of cost for each product.
For each account, the database stores its unique account number and the date the account established, and the cost for the product.
Three types of accounts are maintained:
product1-account to record cost for product 1;
product2-account to record cost for product 2;
product3-account to record cost for product 3.
A customer has a unique name, and an address.
A customer can purchase one or more products.
If a product purchased by a customer is defected due to an error made by a quality controller, the customer will make a formal complaint to the company with the following information:
Date of the complaint;
Detailed description of the complaint;
Treatment expected (get money back or exchange for another product).
Each complaint is identified uniquely by its own id.

A product can be repaired by a technical staff either because it got a complaint or because the repair was requested by a quality controller.
The date of repair will be recorded.
When workers produce products or technical staffs repair products, there may be an accident.
A unique accident number, accident date and number of work days lost due to the accident will be recorded for each accident.

1.2.2. Breaking Down the Description of Future, Inc.

The first technique is associating objects together in each sentence such as:

The FUTURE, Inc. has three different types of employees: technical staff, quality controller and worker.

CAN BE WRITTEN AS

Employee{technical staff, quality controller, worker}

Since employee has 3 different types and can be stated as the 3 types as the employee's set. In doing so, we create a list of linked objects and attributes. We can also group the Set owners together to see the relations such as:

Employee{technical staff, quality controller, worker}
Employee{Unique name, address}
Technical Staff{education{BS, MS, Ph.D}, technical position}
Technical Staff{Fixes problems on products}
Technical Staff{Repair Product by complaint or request by Quality Controller}
Worker ∪ Technical Staff{Can have accident while making or repairing}
Worker{Maximum Number of Products Produced}
Worker{Makes Products}
Quality Controller{Type of Product Checked}
Quality Controller{Check One type of product}
Quality Controller{Tests quality of products}
Product{Needs certification by quality controller to be sold}
Product{Can have error and still get certified by Quality Controller}
Product{Produced by Worker, Checked by Quality Controller, Repaired by Technical Staff}
Product{Unique ID}
Product{Date, Time Spent, Person who produced, Person who tested, Person who repaired if and only if the product was repaired}
Product{Product1, Product2, Product3}
Product1{size{small, medium, large}, Name of major software used}
Product1{If has problem, only Graduate Technical Staff can repair it}
Product2{size{small, medium, large}, Color}
Product2 ∪ Product3{Any Technical Staff can repair it}
Product3{size{small, medium, large}, Weight}
Account{Keep track of each Product Cost}
Account{Unique Account Number, Date Established, Cost}
Account1{Cost of each product1}

Account2{Cost of each product2}
Account3{Cost of each product3}
Customer{Unique name, Address}
Customer{CAN Purchase more than one product}
Customer{File complaint if product is defective}
Complaint{Date, Detailed Description, Treatment Expected, Unique ID}
Repair{Date}
Accident{Unique Accident Number, Date, Number of work days lost}

We now have sufficient information and a more optimal view to build the relational database.

1.2.3. Building the Relational Database

We can work from a top-down process with the Set List to find the Entities that are crucial to the database. First, we have Employee. From the list, we see that Employee has specializations of Technical Staff, Quality Controller, and Worker. Employee also has a Unique Name and Address. From this, we can deduce that Employee will be an entity with the primary key as a Super Key of Name and Address and there will be foreign keys for each specialization to Employee also.

Technical Staff has an Unique Employee Name and Address and can have an education of the 3 options; BS, MS, and Ph.D. Technical Staff will also have a position within the company. Technical Staff will not have an entity in association with type of product they can repair since the education of technical staff can be checked as a custom constraint. Design complexity stays at a minimum with this option. Quality Controllers only have a recorded type of product that particular Quality Controller can check. The Quality Controller will also have a foreign key associated with Employee. Workers have an employee name and address and a recorded max for products produced per day. Name and Address can be the Primary Key for each of these types of employees.

Products have a Unique ID, which is the primary key, and time spent. Every product also records size, so we can generalize size into product instead of product 1, 2, or 3. Each product also has a particular worker, quality controller, and technical staff that contributed hours to it and can be defective and certified.

Product 1 has a product ID that is a key in products and also has the name of the major software used to make it. Product 2 and 3 also have a product ID that is a key in products, but product 2 has a recorded color and product 3 has a recorded weight. Product ID is the primary key in this case for all types of products for generalization.

Account can be a generalization of all product accounts since each type of account only records the cost of the product and the date the account was established. Each account also has a product ID that is associated with Products and a Unique Account Number. Since account number is unique, we can say it is the primary key of account.

Customer simply has a Unique name and Address, which will be the primary key. The customer can file a complaint and that particular customer's name will be associated to the complaint along with a Unique (Primary key) ID. The complaint will also have a date, description, and expected treatment.

The major “Design” split in the road is whether to have a Make, Check, and Repair entity. Repair is the only choice that has an attribute associated with it and product records who touches it, so I chose to only have repair as an entity with an associated date along with the quality controller and technical staff and product ID.

Last but not least, we have Accidents. Accidents have a unique (again Primary key) Accident Number. The employee and product associated are also recorded. The date and how many days the employee lost are recorded.

In addition to these, a purchase entity that was not listed should be implemented due to the fact that a customer can purchase multiple items. Purchase will have the customer and product associated with it.

There is one more special criteria to address. Employee, Product, and Customer are heavily used for searching and updating other table and we have a very good option that we can implement to make the database more efficient. Since each primary key is actually what is referenced the most, we can have dense indexes on each of these as a unique index.

1.2.4. Making Mock Tables with the Information

employee (<u>name, address</u>)
Unique Index on Primary Key

technical_staff (<u>name, address</u>, education, position)
technical_staff(name, address) ← *Foreign Key* →employee(name, address)

quality_controller (<u>name, address,</u> type_of_product_checked)
quality_controller(name, address) ← *Foreign Key* →employee(name, address)

worker (<u>name, address,</u> max_produced_per_day)
worker(name, address) ← *Foreign Key* →employee(name, address)

product (<u>ID</u>, time_spent, worker_name, worker_address, quality_controller_name, quality_controller_address, technical_staff_name, technical_staff_address, size, defect, certified)
product(worker_name, worker_address) ← *Foreign Key* →worker(name, address)
product(quality_controller_name, address) ← *Foreign Key* →quality_controller(name, address)
product(technical_staff_name, technical_staff_address) ← *Foreign Key* →technical_staff(name, address)
Unique Index on Primary Key

accident (<u>accident_number</u>, employee_name, employee_address, product_ID, date, num_work_days_lost)
accident (product_ID) ← *Foreign Key* →product(ID)
accident(employee_name, employee_address) ← *Foreign Key* →employee(name, address)

repair (<u>product_ID</u>, quality_controller_name, quality_controller_address, technical_staff_name, technical_staff_address, date)
repair(product_ID) ← *Foreign Key* → product(ID),

repair(quality_controller_name, quality_controller_address) ← *Foreign Key* → quality_controller(name, address)
repair(technical_staff_name, technical_staff_address) ← *Foreign Key* → technical_staff(name, address), Use Check for Technical_staff_name, technical_staff_address qualification

product_1 (product_ID, NAME_OF_MAJOR_SOFTWARE)
product_1(product_ID) ← *Foreign Key* → product(ID)

product_2 (product_ID, color)
product_2(product_ID) ← *Foreign Key* → product(ID)

product_3 (product_ID, weight)
product_3(product_ID) ← *Foreign Key* → product(ID)

account (account_number, product_ID, date_established, cost_to_make)
account(product_ID) ← *Foreign Key* → product(ID)

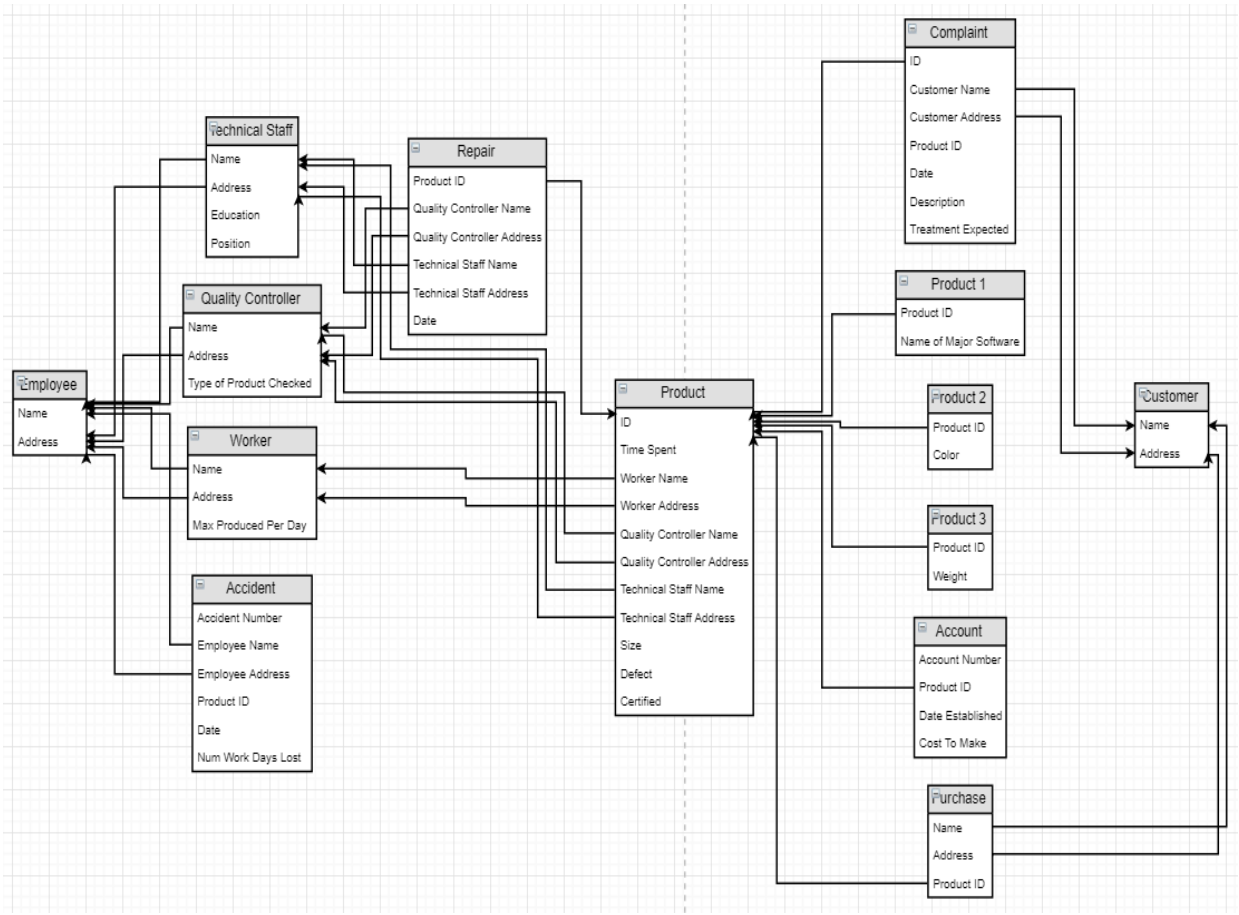
customer (name, address)
Unique Index on Primary Key

purchase (name, address, product_ID)
purchase(name, address) ← *Foreign Key* → customer(name, address),
purchase(product_ID) ← *Foreign Key* → product(ID)

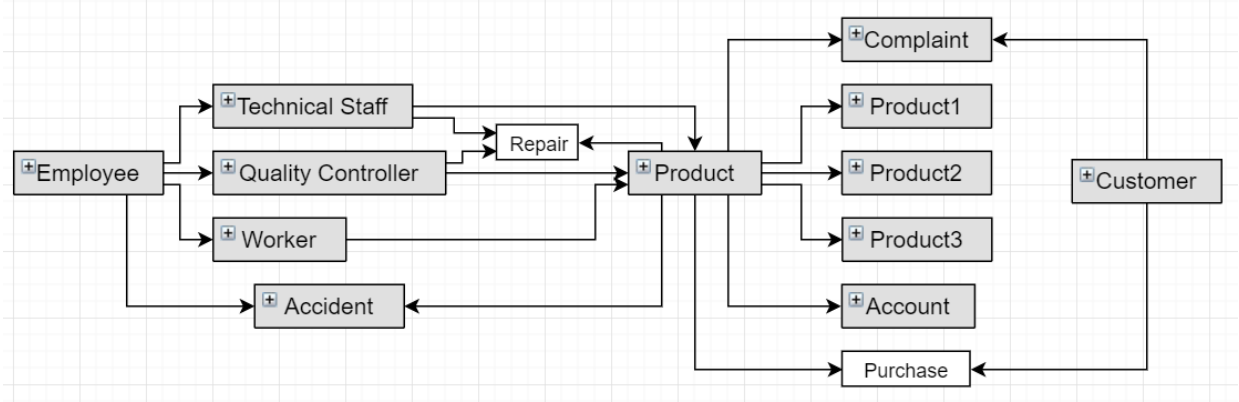
complaint (ID, customer_name, customer_address, product_ID, date, description, treatment_expected)
complaint(product_ID) ← *Foreign Key* → product(ID),
complaint(customer_name, customer_address) ← *Foreign Key* → customer(name, address)

1.2.5. Creating the Schema Diagram

Using the Information from the above mock tables, we can create the following Schema Diagram:



To show the referencing of each table to the other as a whole, a Reference Diagram is also produced.



Task 2. Data Dictionary

Table Name	Attribute Name	Type	Size	Contraints
Employee	Name	VARCHAR	64	Primary Key, Indexed
	Address	VARCHAR	64	
Technical Staff	Name	VARCHAR	64	Primary Key, Employee Table Foreign Key
	Address	VARCHAR	64	
	Education	VARCHAR	4	Null, or either BS, MS, or Ph.D
	Position	VARCHAR	32	Can be Null
Quality Controller	Name	VARCHAR	64	Primary Key, Employee Table Foreign Key
	Address	VARCHAR	64	
	Type_of_Product_Checked	INT	4	Not Null, either 1,2, or 3
Worker	Name	VARCHAR	64	Primary Key, Employee Table Foreign Key
	Address	VARCHAR	64	
	Max_Produced_Per_Day	INT	4	DEFAULT is 1000
Product	ID	INT	4	Primary Key, Indexed
	Time_Spent	INT	4	DEFAULT is 10
	Worker_Name	VARCHAR	64	Not Null, Worker Table Foreign Key
	Worker_Address	VARCHAR	64	
	Quality_Controller_Name	VARCHAR	64	Not Null, Quality Controller Table Foreign Key
	Quality_Controller_Address	VARCHAR	64	
	Technical_Staff_Name	VARCHAR	64	Can be Null, Technical Staff Table Foreign Key
	Technical_Staff_Address	VARCHAR	64	
	Size	VARCHAR	7	DEFAULT is MEDIUM
	Defect	VARCHAR	3	DEFAULT is NO
	Certified	VARCHAR	3	DEFAULT is NO
Accident	Accident_Number	INT	4	Primary Key
	Employee_Name	VARCHAR	64	Not Null, Employee Table Foreign Key
	Employee_Address	VARCHAR	64	
	Product_ID	INT	4	Not Null, Product Table Foreign Key
	Date	DATE	8	DEFAULT Current Date, YYYY-MM-DD
	Num_Work_Days_Lost	INT	4	DEFAULT is 5
	Product_ID	INT	4	Primary Key, Product Table Foreign Key

Repair	Quality_Controller_Name	VARCHAR	64	Not Null, Quality Controller Table Foreign Key
	Quality_Controller_Address	VARCHAR	64	
	Technical_Staff_Name	VARCHAR	64	Not Null, Technical Staff Table Foreign Key
	Technical_Staff_Address	VARCHAR	64	
	Date	DATE	8	DEFAULT Current Date, YYYY-MM-DD
Product_1	Product_ID	INT	4	Primary Key and Foreign Key to Product
	Name_Of_Major_Software	VARCHAR	64	Not Null
Product_2	Product_ID	INT	4	Primary Key and Foreign Key to Product
	Color	VARCHAR	32	Not Null
Product_3	Product_ID	INT	4	Primary Key and Foreign Key to Product
	Weight	NUMERIC	6	DEFAULT is 10
Account	Account_Number	INT	4	Primary Key
	Product_ID	INT	4	Not Null, Product Table Foreign Key
	Date_Established	DATE	8	DEFAULT Current Date, YYYY-MM-DD
	Cost_To_Make	NUMERIC	6	DEFAULT is 100
Customer	Name	VARCHAR	64	Primary Key, Indexed
	Address	VARCHAR	64	
Purchase	Name	VARCHAR	64	Primary Key, Customer Table Foreign Key and Product Table Foreign Key
	Address	VARCHAR	64	
	Product_ID	INT	4	
Complaint	ID	INT	4	Primary Key
	Customer_Name	VARCHAR	64	Not Null, Customer Table Foreign Key
	Customer_Address	VARCHAR	64	
	Product_ID	INT	4	Not Null, Product Table Foreign Key
	Date	DATE	8	DEFAULT Current Date, YYYY-MM-DD
	Description	VARCHAR	200	Can be Null
	Treatment_Expected	VARCHAR	200	Can be Null

Task 3.

3.1. Discussion of storage structures for tables

3.1.1. Choices of File Structures

In discussing storage structures for tables, we have the choices between:

- Heap File
    - Insertion is O(1)
    - Deletion is O(N)
    - Lookup is also O(N)
    - Records are inserted at the bottom of the file and lookup is sequential for finding a record, which also makes deletion the same time complexity since the record needs to be found.
- Sequential File
    - Insertion is O(N)
    - Deletion is O(N)
    - Lookup is O(N)
    - Records are inserted in order in the file and insertion has to find the correct spot for the record, along with deletion and lookup having to sequentially look for the correct record.
- Indexed Sequential File
    - Insertion is O(N)
    - Deletion is O(1)
    - Lookup is O(1)
    - Records are inserted just like the regular sequential file, but each record is indexed. Assuming the index is dense, then each record will have its own index, so lookup is immediate and deletion is immediate.
- B Tree File
    - Insertion is O(log(N))
    - Deletion is O(log(N))
    - Lookup is O(log(N))
    - The file is indexed in tree format, so each file can be inserted, deleted, and looked up in time = to depth of tree + the number of records in each leaf.
- B+ Tree File
    - Insertion is O(log(N))
    - Deletion is O(log(N))
    - Lookup is O(log(N))
    - The file is also indexed in tree format and insertion, deletion, and lookup are also the same as B trees, but the unique feature the B+ tree has is, if the pointer is already at the depth of the tree and needs to access another leaf, each leaf is attached and can be accessed in a sequential order.

And certain tables can have options such as static and dynamic hashing, and indexing on these files whether the index be sparse or dense. The factors of that determine how we store our files include;

Where and how frequent we are searching through the table, how many times we manipulate the table such as insertion and deletion, Frequencies of queries, Time Complexity, and Space Complexity.

3.1.2. Known Queries in Database

The queries we have for the database are as follows;

- Enter a new employee (2/month).
- Enter a new product associated with the person who made the product, repaired the product if it is repaired, or checked the product (400/day).
- Enter a customer associated with some products (50/day).
- Create a new account associated with a product (40/day).
- Enter a complaint associated with a customer and product (30/day).
- Enter an accident associated with appropriate employee and product (1/week).
- Retrieve the date produced and time spent to produce a particular product (100/day).
- Retrieve all products made by a particular worker (2000/day).
- Retrieve the total number of errors a particular quality controller made. This is the total number of products certified by this controller and got some complaints (400/day).
- Retrieve the total costs of the products in the product3 category which were repaired at the request of a particular quality controller (40/day).
- Retrieve all customers who purchased all products of a particular color (5/month).
- Retrieve the total number of work days lost due to accidents in repairing the products which got complaints (1/month).
- Retrieve all customers who are also workers (10/month).
- Retrieve all the customers who have purchased the products made or certified or repaired by themselves (5/day).
- Retrieve the average cost of all products made in a particular year (5/day).
- Switch the position between a technical staff and a quality controller (1/ 3 months).
- Delete all accidents whose dates are in some range (1/day).

3.1.3. Average Maximum Operations of Queries

We can generalize the frequencies of these queries as operation/day to get the maximum average operations/day as:

- Employee table
  - .06 insertions/day
  - 1 queries/day
- Technical Staff table
  - .08 insertions/day
  - 405 queries/day
  - .01 deletions/day
- Quality Controller table
  - .08 insertions/day
  - 845 queries/day

    - .01 deletions/day
- Worker table
    - .07 insertions/day
    - 2415 queries/day
- Product table
    - 400 insertions/day
    - 2625.34 queries/day
- Accident table
    - .14 insertions/day
    - .03 queries/day
    - 1 Range deletion/day
- Repair table
    - 400 insertions/day
    - 40 queries/day
- Product 1 table
    - 400 insertions/day
- Product 2 table
    - 400 insertions/day
    - .17 queries/day
- Product 3 table
    - 400 insertions/day
    - 40 queries/day
- Account table
    - 40 insertions/day
    - 5 queries/day
- Customer table
    - 50 insertions/day
    - 35.5 queries/day
- Purchase table
    - 50 insertions/day
    - 5.17 queries/day
- Complaint table
    - 30 insertions/day
    - 400.03 queries/day

3.1.4. File Structure Decisions on Tables

With the Employee table, we have minimal insertions and references per day, but references in this case beats the insertion of employees. That being said, we can place the table in a indexed sequential file, with a dense index on the employees name and address since the combination is unique for every record.

With the Technical Staff table, we have very minimal insertions and deletions, but we have pretty heavy referencing per day. Referencing is a major factor in deciding on a storage structure for the table. Indexed Sequential files seem to fit the structure, since we do not need to worry about insertion as much.

With the Quality Controller table, we have very minimal insertions and deletions, but we have pretty heavy referencing per day. Referencing is a major factor in deciding on a storage structure for this table also. Indexed Sequential files seem to also fit.

With the Worker table, we have very minimal insertions and no deletions, but we have extremely heavy referencing per day. Referencing is a massive factor in deciding on a storage structure for the worker table. Indexed Sequential files seem to also fit.

With the Product table, we have heavy insertions and referencing per day. There are no deletions with the product table, so we can exclude certain tables. Since insertion and referencing is heavy, a B Tree is needed to keep these operations at a medium minimum.

With the Accident table, we can see that queries are minimal, but insertion and deletion are heavy. We can place this in a Heap File structure to maximize the efficiency of insertion and deletion. Although queries will be done in linear time, the frequency of queries on this table is so miniscule, the heap pile will be best for the table.

With the Repair table, we have heavy insertion rates and average query rates. With this table, the best option we can use is Hashing because insertion and lookup will be exceptional compared to other structures.

With the Product 1 table, only we have heavy insertion rates. This being the case, the table is very unused in the database and may be queried later in future versions, but for now we can place this table in a heap pile.

With the Product 2 table and Product 3 table, we have heavy insertion rates, and average query rates. We can put hashes on these to create fast and efficient insertion and lookup.

With the Account table, Customer table, Purchase table, Complaint table, we have medium insertion and query rates. This being said, we can place each of these in a B tree file structure to maximize efficiency in insertion and lookup without compromising the tables.

3.2. Discussion of storage structures for tables (Azure SQL Database)

With the storage structures of Azure, we have choices of indexes such as;

- Clustered Index
- Non-Clustered Index
- Unique Index
- Hash Index
- Single Index
- Composite Index

A clustered and non-clustered indices store the tables in B trees, but clustered indices sort the data before it is stored. Tables can have multiple non-clustered indices. Unique indices have different values and a dense indices on clustered and non-clustered indices. A hash index stores a different structure that hashes the table for easy lookup and has a specified bucket size for the table. Single and Composite indices are different as in the single index is an index on one attribute and a composite index is an index on multiple attributes.

The Employee Table has a Unique Name and Address, which is referenced by the worker, technical staff, quality controller, and accident tables, so this can be a composite, non-clustered, unique index on the primary key for referencing. This table will be stored in a B tree.

The Technical Staff, Quality Controller, and Worker Tables also have the exact same name and address of the employee table, which deems the employee table primary key as the foreign key to each of these tables. These tables will also have an index that is composite on name and address, non-clustered, and unique. Each of these tables will be stored in B trees.

The product table will have an index on ID where it will be unique, single, and clustered because the ID is referenced by accident, repair, product 1, product 2, product 3, account, purchase, and complaint tables.

The Repair table will have a single hash index on product ID that can have multiple records in a bucket, so each product, while very rare, can be repaired more than once.

The Customer Table will have a composite, non-clustered, unique index on the name and address for easy referencing, where the complaint and purchase tables can reference more efficiently.

Task 4. SQL and text files showing the creation of tables in Azure SQL Database

```sql
--Drop Previous Tables and Reinitialize Empty Tables
drop table complaint
drop table purchase
drop table product_3
drop table product_2
drop table product_1
drop table customer
drop table account
drop table repair
drop table accident
drop table product
drop table worker
drop table technical_staff
drop table quality_controller
drop table employee


--Create Employee Table With Unique Index on the
--Primary Key for Derived Table References
CREATE TABLE employee (
    name VARCHAR(64) NOT NULL,
    address VARCHAR(64) NOT NULL,
    primary key(name, address))
CREATE UNIQUE NONCLUSTERED INDEX employee_info
ON employee (name, address);


--Create a Technical_Staff Table that is Specialized From the
--Employee Table and adds a foreign key for Syncronization.
CREATE TABLE technical_staff (
    name VARCHAR(64) NOT NULL,
    address VARCHAR(64) NOT NULL,
    FOREIGN key(name, address) REFERENCES employee(name, address),
    education VARCHAR(5) CHECK (education IN ('BS', 'MS', 'PH.D')),
    position VARCHAR(32)
    primary key(name, address))
CREATE UNIQUE NONCLUSTERED INDEX ts_info
ON technical_staff (name, address);

--Create a Quality Controller Table that is Specialized from the
--Employee Table and adds a foreign key for Syncronization
CREATE TABLE quality_controller (
    name VARCHAR(64) NOT NULL,
    address VARCHAR(64) NOT NULL,
    FOREIGN key(name, address) REFERENCES employee(name, address),
    type_of_product_checked INT NOT NULL CHECK (type_of_product_checked IN (1, 2, 3))
    primary key(name, address))
CREATE UNIQUE NONCLUSTERED INDEX qc_info
ON quality_controller (name, address);

--Create a Worker Table that is Specialized from the
--Employee Table and adds a foreign key for Syncronization
CREATE TABLE worker (
    name VARCHAR(64) NOT NULL,
    address VARCHAR(64) NOT NULL,
    FOREIGN key(name, address) REFERENCES employee(name, address),
    max_produced_per_day INT DEFAULT 1000
    primary key(name, address))
CREATE UNIQUE NONCLUSTERED INDEX w_info
ON worker (name, address);
```

```
--Create a Product Table that Uses Multiple Foreign Keys
--to stay in sync with specialized Employee Tables and
--creates a unique index on product ID for multiple tables
--To reference
CREATE TABLE product (
    ID INT NOT NULL,
    time_spent INT DEFAULT 10,
    worker_name VARCHAR(64) NOT NULL,
    worker_address VARCHAR(64) NOT NULL,
    FOREIGN key(worker_name, worker_address) REFERENCES worker(name, address),
    quality_controller_name VARCHAR(64) NOT NULL,
    quality_controller_address VARCHAR(64) NOT NULL,
    FOREIGN key(quality_controller_name, quality_controller_address) REFERENCES quality_controller(name, address),
    technical_staff_name VARCHAR(64),
    technical_staff_address VARCHAR(64),
    FOREIGN key(technical_staff_name, technical_staff_address) REFERENCES technical_staff(name, address),
    size VARCHAR(7) CHECK (size IN ('SMALL', 'MEDIUM', 'LARGE')) DEFAULT 'MEDIUM',
    defect VARCHAR(3) CHECK (defect IN ('YES', 'NO')) DEFAULT 'NO',
    certified VARCHAR(3) CHECK (certified IN ('YES', 'NO')) DEFAULT 'NO'
    primary key(ID))
CREATE UNIQUE NONCLUSTERED INDEX product_index
ON product (ID);


--Create an accident Table that Has a foreign key
--on the employee and product tables for syncronization
CREATE TABLE accident (
    accident_number INT NOT NULL,
    employee_name VARCHAR(64) NOT NULL,
    employee_address VARCHAR(64) NOT NULL,
    FOREIGN key(employee_name, employee_address) REFERENCES employee(name, address),
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    date DATE DEFAULT getdate(),
    num_work_days_lost INT DEFAULT 5
    primary key(accident_number))


--Create repair table that has quality controller
--and technical staff foreign keys to keep Sync.
--Added custom constraint to check if technical
--staff member has qualifying education.
--Hashing allows for a year and half of repairs on record.
CREATE TABLE repair (
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    quality_controller_name VARCHAR(64) NOT NULL,
    quality_controller_address VARCHAR(64) NOT NULL,
    FOREIGN key(quality_controller_name, quality_controller_address) REFERENCES quality_controller(name, address),
    technical_staff_name VARCHAR(64) NOT NULL,
    technical_staff_address VARCHAR(64) NOT NULL,
    FOREIGN key(technical_staff_name, technical_staff_address) REFERENCES technical_staff(name, address),
    date DATE DEFAULT getdate(),
    primary key(product_ID))
ALTER TABLE repair
ADD CONSTRAINT check_if_qualified
CHECK (dbo.technical_staff_qualify(product_ID, technical_staff_name, technical_staff_address) = 0);

--Create product 1 table with foreign key on product
CREATE TABLE product_1 (
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    NAME_OF_MAJOR_SOFTWARE VARCHAR(64) NOT NULL
    primary key(product_ID))


--Create product 2 table with foreign key on product
CREATE TABLE product_2 (
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    color VARCHAR(32) NOT NULL
    primary key(product_ID))
```

```
--Create product 3 table with foreign key on product
CREATE TABLE product_3 (
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    weight NUMERIC(6,2) DEFAULT 10
    primary key(product_ID))


--Create account Table with foreign key on product
CREATE TABLE account (
    account_number INT NOT NULL,
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    date_established DATE DEFAULT getdate(),
    cost_to_make NUMERIC(6,2) DEFAULT 100,
    primary key(account_number))


--Create customer table with unique index on the customer
--information that allows for speedy searching and referencing.
CREATE TABLE customer (
    name VARCHAR(64) NOT NULL,
    address VARCHAR(64) NOT NULL,
    primary key(name, address))
CREATE UNIQUE INDEX custome_index
ON customer (name, address);


--Create purchase table with foreign keys on
--customer and product tables
CREATE TABLE purchase (
    name VARCHAR(64) NOT NULL,
    address VARCHAR(64) NOT NULL,
    FOREIGN key(name, address) REFERENCES customer(name, address),
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    primary key(name, address, product_ID),)


--Create complaint Table with foreign keys on
--customer and product tables
CREATE TABLE complaint (
    ID int not null,
    customer_name VARCHAR(64) not null,
    customer_address VARCHAR(64) not null,
    FOREIGN key(customer_name, customer_address) REFERENCES customer(name, address),
    product_ID INT NOT NULL FOREIGN key REFERENCES product(ID),
    date DATE DEFAULT getdate(),
    description VARCHAR(200),
    treatment_expected VARCHAR(200)
    primary key(ID))
```

```sql
--Function to check whether or not a technical_staff member is
--qualified to repair a product in the category product_1.
--Returns 0 if the technical staff qualifies
--and 1 if the technical staff does not.
CREATE FUNCTION dbo.technical_staff_qualify(@product_ID INT,
@technical_staff_name VARCHAR(64), @technical_staff_address VARCHAR(64))
    RETURNS INT
AS
    BEGIN
        DECLARE @val int
        SET @val = 0;
        IF EXISTS (
            SELECT 1
            FROM technical_staff
            WHERE technical_staff.name = @technical_staff_name
                AND technical_staff.address = @technical_staff_address
                AND technical_staff.education IS NULL)
        AND EXISTS(
            SELECT 1
            FROM product_1
            WHERE product_1.product_ID = @product_ID)
        BEGIN
            SET @val = 1
        END
        RETURN @val
    END
GO

--Drop previously made procedure
DROP PROCEDURE sel7Func;
DROP PROCEDURE sel8Func;
DROP PROCEDURE sel9Func;
DROP PROCEDURE sel10Func;
DROP PROCEDURE sel11Func;
DROP PROCEDURE sel12Func;
DROP PROCEDURE sel13Func;
DROP PROCEDURE sel14Func;
DROP PROCEDURE sel15Func;


--Stored Procedure for selection 7
--Retrieves the date produced and time spent
--to produce a particular product (100/day).
GO
CREATE PROCEDURE sel7Func
    @product_ID INT
AS
    BEGIN
        SELECT account.date_established, product.time_spent
        FROM product, account
        WHERE product.ID = @product_ID
            AND account.product_ID = @product_ID;
    END
GO
```

```
--Stored Procedure for selection 8
--Retrieves all products made by a particular worker (2000/day).
GO
CREATE PROCEDURE sel8Func
    @worker_name VARCHAR(64),
    @worker_address VARCHAR(64)
AS
    BEGIN
        SELECT ID
        FROM product
        WHERE worker_name = @worker_name
            AND worker_address = @worker_address;
    END
GO

--Stored Procedure for selection 9
--Retrieves the total number of errors a particular quality controller made. This is the total
--number of products certified by this controller and got some complaints (400/day).
GO
CREATE PROCEDURE sel9Func
    @quality_controller_name VARCHAR(64),
    @quality_controller_address VARCHAR(64)
AS
    BEGIN
        SELECT COUNT(product.ID)
        FROM product
        WHERE quality_controller_name = @quality_controller_name
            AND quality_controller_address = @quality_controller_address
            AND defect = 'yes'
            AND certified = 'yes'
    END
GO
```

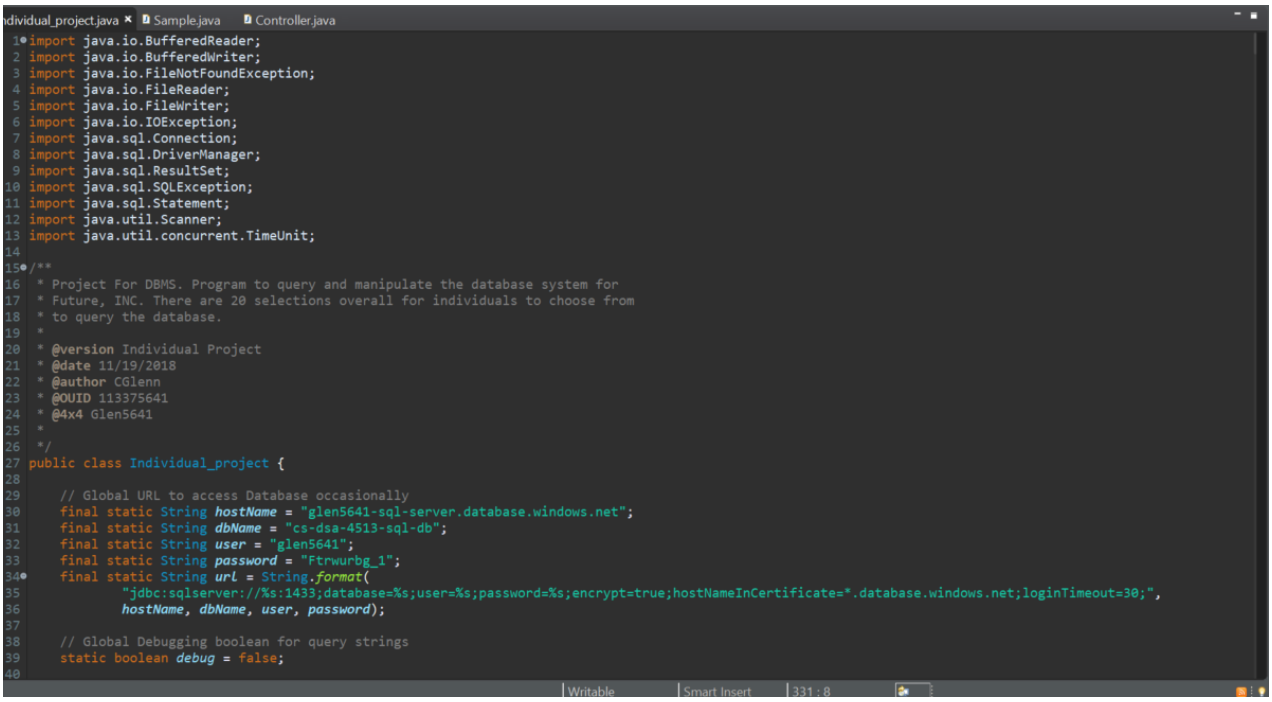
```
--Stored Procedure for selection 10
--Retrieve the total costs of the products in the product3 category which
--were repaired at the request of a particular quality controller (40/day).
GO
CREATE PROCEDURE sel10Func
    @quality_controller_name VARCHAR(64),
    @quality_controller_address VARCHAR(64)
AS
    BEGIN
        SELECT SUM(account.cost_to_make)
        FROM product, product_3, repair, account
        WHERE product.ID = product_3.product_ID
            AND product.quality_controller_name = @quality_controller_name
            AND product.quality_controller_address = @quality_controller_address
            AND product.ID = repair.product_ID
            AND account.product_ID = product.ID;
    END
GO

--Stored Procedure for selection 11
--Retrieve all customers who purchased all products of a particular color (5/month).
GO
CREATE PROCEDURE sel11Func
    @color VARCHAR(20)
AS
    BEGIN
        SELECT purchase.name, purchase.address
        FROM purchase, product_2
        WHERE product_2.color = @color
            AND purchase.product_ID = product_2.product_ID;
    END
GO
```

```
--Stored Procedure for selection 12
--Retrieve the total number of work days lost due to accidents
--in repairing the products which got complaints (1/month).
GO
CREATE PROCEDURE sel12Func
AS
    BEGIN
        SELECT SUM(accident.num_work_days_lost)
        FROM accident, repair, complaint
        WHERE accident.product_ID = repair.product_ID
            AND accident.product_ID = complaint.product_ID;
    END
GO

--Stored Procedure for selection 13
--Retrieve all customers who are also workers (10/month).
GO
CREATE PROCEDURE sel13Func
AS
    BEGIN
        SELECT customer.name, customer.address
        FROM customer, worker
        WHERE customer.name = worker.name
            AND customer.address = worker.address;
    END
GO
```

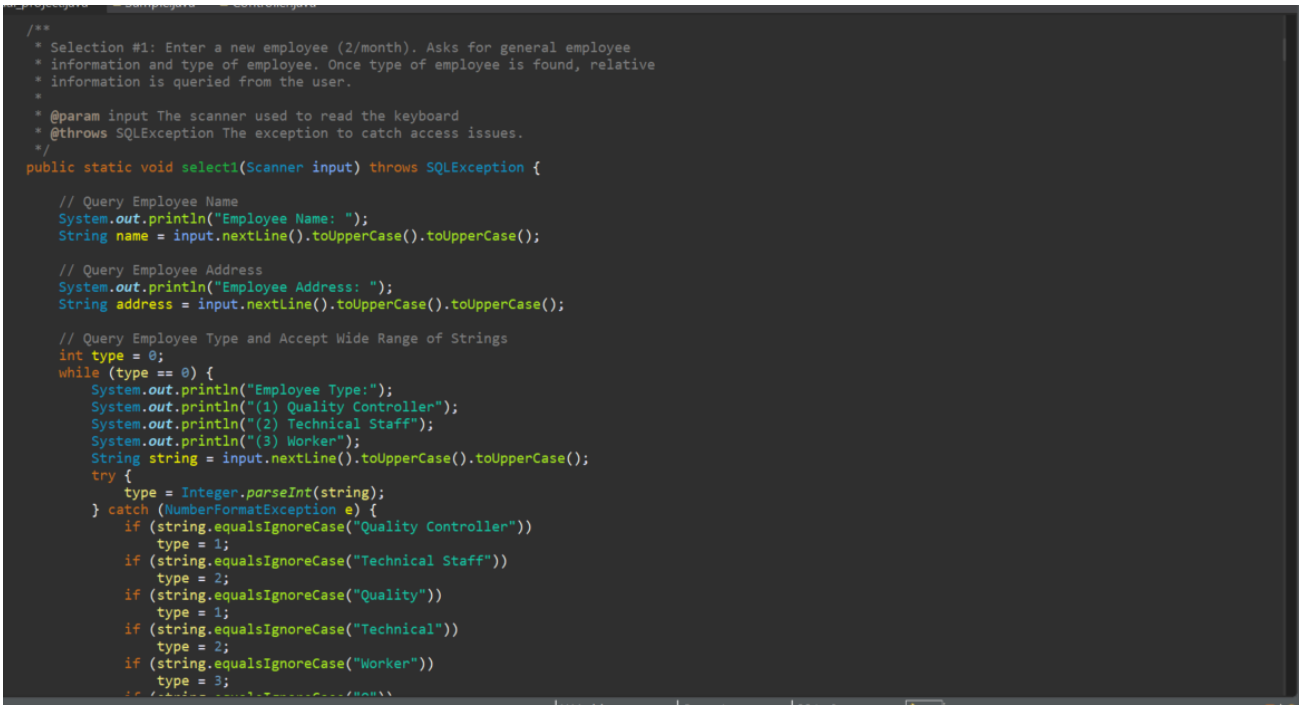
```
--Stored Procedure for selection 14
--Retrieve all the customers who have purchased the products
--made or certified or repaired by themselves (5/day).
GO
CREATE PROCEDURE sel14Func
AS
    BEGIN
        SELECT purchase.name, purchase.address
        FROM purchase, product
        WHERE
            (purchase.name = product.worker_name
            AND purchase.address = product.worker_address
            AND purchase.product_ID = product.ID)
        OR
            (purchase.name = product.quality_controller_name
            AND purchase.address = product.quality_controller_address
            AND purchase.product_ID = product.ID)
        OR
            (purchase.name = product.technical_staff_name
            AND purchase.address = product.technical_staff_address
            AND purchase.product_ID = product.ID)
    END
GO

--Stored Procedure for selection 15
--Retrieve the average cost of all products made in a particular year (5/day).
GO
CREATE PROCEDURE sel15Func
@year INT
AS
    BEGIN
        SELECT AVG(account.cost_to_make)
        FROM account
        WHERE year(account.date_established) = @year;

    END
GO
```

Task 5. Screenshots showing entire Java program and its successful compilation

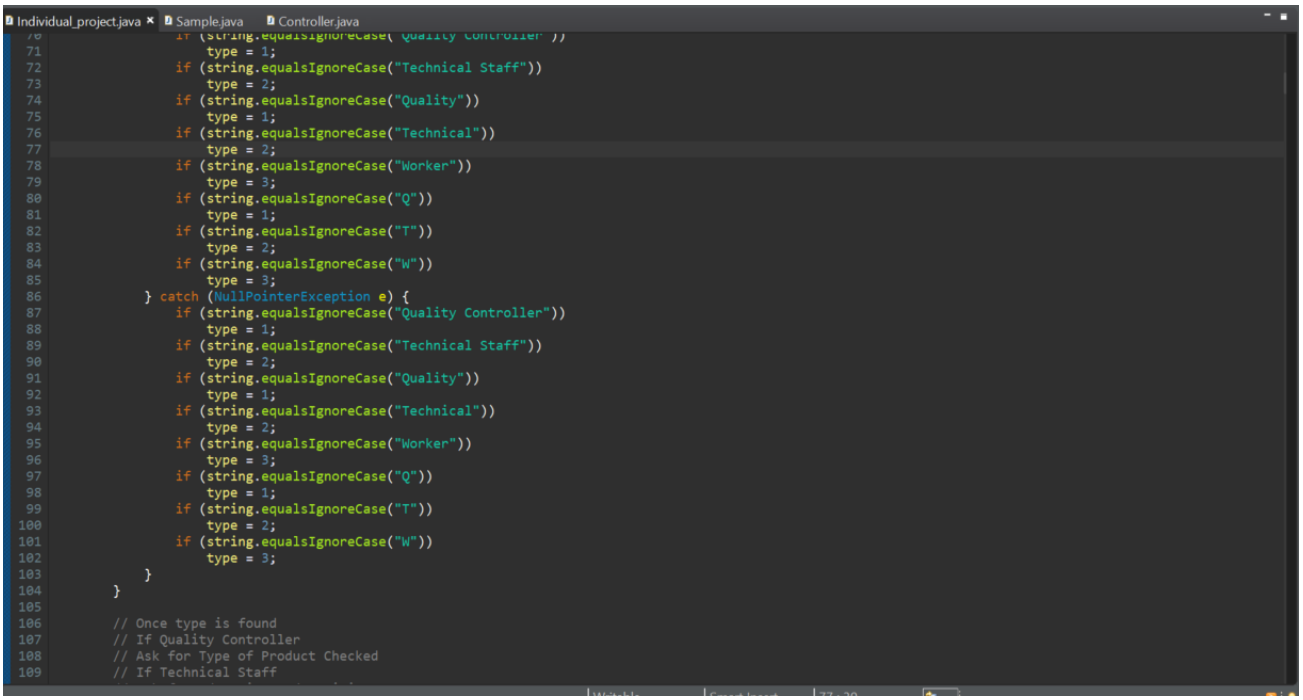
5.1) Screenshots of Java Code



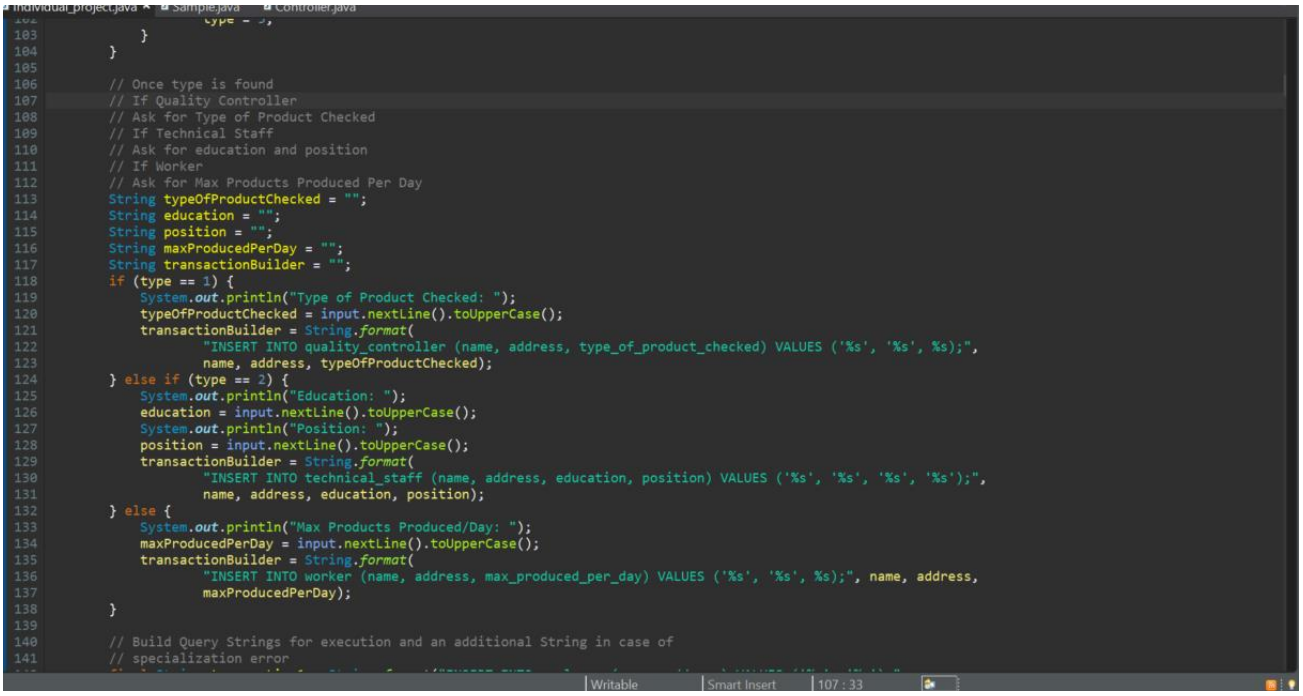
```
dividual_project.java ×  Sample.java  Controller.java
 1 import java.io.BufferedReader;
 2 import java.io.BufferedWriter;
 3 import java.io.FileNotFoundException;
 4 import java.io.FileReader;
 5 import java.io.FileWriter;
 6 import java.io.IOException;
 7 import java.sql.Connection;
 8 import java.sql.DriverManager;
 9 import java.sql.ResultSet;
10 import java.sql.SQLException;
11 import java.sql.Statement;
12 import java.util.Scanner;
13 import java.util.concurrent.TimeUnit;
14
15 /**
16  * Project For DBMS. Program to query and manipulate the database system for
17  * Future, INC. There are 20 selections overall for individuals to choose from
18  * to query the database.
19  *
20  * @version Individual Project
21  * @date 11/19/2018
22  * @author CGlenn
23  * @OUID 113375641
24  * @4x4 Glen5641
25  *
26  */
27 public class Individual_project {
28
29     // Global URL to access Database occasionally
30     final static String hostName = "glen5641-sql-server.database.windows.net";
31     final static String dbName = "cs-dsa-4513-sql-db";
32     final static String user = "glen5641";
33     final static String password = "Ftrwurbg_1";
34     final static String url = String.format(
35             "jdbc:sqlserver://%s:1433;database=%s;user=%s;password=%s;encrypt=true;hostNameInCertificate=*.database.windows.net;loginTimeout=30;",
36             hostName, dbName, user, password);
37
38     // Global Debugging boolean for query strings
39     static boolean debug = false;
40
                                  Writable   Smart Insert   331 : 8
```



```
/**
 * Selection #1: Enter a new employee (2/month). Asks for general employee
 * information and type of employee. Once type of employee is found, relative
 * information is queried from the user.
 *
 * @param input The scanner used to read the keyboard
 * @throws SQLException The exception to catch access issues.
 */
public static void select1(Scanner input) throws SQLException {

    // Query Employee Name
    System.out.println("Employee Name: ");
    String name = input.nextLine().toUpperCase().toUpperCase();

    // Query Employee Address
    System.out.println("Employee Address: ");
    String address = input.nextLine().toUpperCase().toUpperCase();

    // Query Employee Type and Accept Wide Range of Strings
    int type = 0;
    while (type == 0) {
        System.out.println("Employee Type:");
        System.out.println("(1) Quality Controller");
        System.out.println("(2) Technical Staff");
        System.out.println("(3) Worker");
        String string = input.nextLine().toUpperCase().toUpperCase();
        try {
            type = Integer.parseInt(string);
        } catch (NumberFormatException e) {
            if (string.equalsIgnoreCase("Quality Controller"))
                type = 1;
            if (string.equalsIgnoreCase("Technical Staff"))
                type = 2;
            if (string.equalsIgnoreCase("Quality"))
                type = 1;
            if (string.equalsIgnoreCase("Technical"))
                type = 2;
            if (string.equalsIgnoreCase("Worker"))
                type = 3;
```



```
Individual_project.java × Sample.java Controller.java
71                  type = 1;
72              if (string.equalsIgnoreCase("Technical Staff"))
73                  type = 2;
74              if (string.equalsIgnoreCase("Quality"))
75                  type = 1;
76              if (string.equalsIgnoreCase("Technical"))
77                  type = 2;
78              if (string.equalsIgnoreCase("Worker"))
79                  type = 3;
80              if (string.equalsIgnoreCase("Q"))
81                  type = 1;
82              if (string.equalsIgnoreCase("T"))
83                  type = 2;
84              if (string.equalsIgnoreCase("W"))
85                  type = 3;
86          } catch (NullPointerException e) {
87              if (string.equalsIgnoreCase("Quality Controller"))
88                  type = 1;
89              if (string.equalsIgnoreCase("Technical Staff"))
90                  type = 2;
91              if (string.equalsIgnoreCase("Quality"))
92                  type = 1;
93              if (string.equalsIgnoreCase("Technical"))
94                  type = 2;
95              if (string.equalsIgnoreCase("Worker"))
96                  type = 3;
97              if (string.equalsIgnoreCase("Q"))
98                  type = 1;
99              if (string.equalsIgnoreCase("T"))
100                 type = 2;
101             if (string.equalsIgnoreCase("W"))
102                 type = 3;
103         }
104     }
105
106     // Once type is found
107     // If Quality Controller
108     // Ask for Type of Product Checked
109     // If Technical Staff
```



```
Individual_project.java × Sample.java Controller.java
103         }
104     }
105
106     // Once type is found
107     // If Quality Controller
108     // Ask for Type of Product Checked
109     // If Technical Staff
110     // Ask for education and position
111     // If Worker
112     // Ask for Max Products Produced Per Day
113     String typeOfProductChecked = "";
114     String education = "";
115     String position = "";
116     String maxProducedPerDay = "";
117     String transactionBuilder = "";
118     if (type == 1) {
119         System.out.println("Type of Product Checked: ");
120         typeOfProductChecked = input.nextLine().toUpperCase();
121         transactionBuilder = String.format(
122                 "INSERT INTO quality_controller (name, address, type_of_product_checked) VALUES ('%s', '%s', %s);",
123                 name, address, typeOfProductChecked);
124     } else if (type == 2) {
125         System.out.println("Education: ");
126         education = input.nextLine().toUpperCase();
127         System.out.println("Position: ");
128         position = input.nextLine().toUpperCase();
129         transactionBuilder = String.format(
130                 "INSERT INTO technical_staff (name, address, education, position) VALUES ('%s', '%s', '%s', '%s');",
131                 name, address, education, position);
132     } else {
133         System.out.println("Max Products Produced/Day: ");
134         maxProducedPerDay = input.nextLine().toUpperCase();
135         transactionBuilder = String.format(
136                 "INSERT INTO worker (name, address, max_produced_per_day) VALUES ('%s', '%s', %s);", name, address,
137                 maxProducedPerDay);
138     }
139
140     // Build Query Strings for execution and an additional String in case of
141     // specialization error
Writable  Smart Insert  107 : 33
```

```java
        // Build Query Strings for execution and an additional String in case of
        // specialization error
        final String transaction1 = String.format("INSERT INTO employee (name, address) VALUES ('%s', '%s');", name,
                address);
        final String transaction2 = transactionBuilder;
        final String transaction1Rewind = String.format("DELETE FROM employee WHERE name='%s' AND address='%s';", name,
                address);

        // Print strings if debug
        if (debug) {
            System.out.println(transaction1);
            System.out.println(transaction2);
            System.out.println(transaction1Rewind);
        }

        // Access the database via url
        try (final Connection connection = DriverManager.getConnection(url)) {
            try {
                // Try to execute statement 1. If 1 fails return
                final Statement statement1 = connection.createStatement();
                statement1.execute(transaction1);
            } catch (SQLException e) {
                System.err.println("Entry already Exists.");
                return;
            }
            try {
                // If statement 1 passes, try statement 2. If 2 fails, remove the entry from
                // employee table
                final Statement statement2 = connection.createStatement();
                statement2.execute(transaction2);
            } catch (SQLException e) {
                System.err.println("Invalid education or type of product checked");
                try {
                    // If rewind fails, database is out of sync
                    final Statement statement3 = connection.createStatement();
                    statement3.execute(transaction1Rewind);
                } catch (SQLException s) {
                    System.err.println("Critical, Unable to correct Database.");
                }
```

Writable | Smart Insert | 107 : 33

Individual_project.java | Sample.java | Controller.java

```java
                final Statement statement2 = connection.createStatement();
                statement2.execute(transaction2);
            } catch (SQLException e) {
                System.err.println("Invalid education or type of product checked");
                try {
                    // If rewind fails, database is out of sync
                    final Statement statement3 = connection.createStatement();
                    statement3.execute(transaction1Rewind);
                } catch (SQLException s) {
                    System.err.println("Critical, Unable to correct Database.");
                }
            }
        }
    }

    /**
     * Selection #2: Enter a new product associated with the person who made the
     * product, repaired the product if it is repaired, or checked the product
     * (400/day). Build the correct specialized product and product # and add it to
     * the database with the information queried from the user.
     *
     * @param input The scanner used to read the keyboard
     * @throws SQLException The exception to catch access issues.
     */
    public static void select2(Scanner input) throws SQLException {

        // Query Product ID
        System.out.println("Product ID: ");
        String productID = input.nextLine().toUpperCase();

        // Query Time Spent
        System.out.println("Time Spent making the Product: ");
        String timeSpent = input.nextLine().toUpperCase();

        // Query Worker Name
        System.out.println("Worker Name: ");
        String workerName = input.nextLine().toUpperCase();

        // Query Worker Address
```

Writable | Smart Insert | 205 : 1

```java
        // Query Product ID
        System.out.println("Product ID: ");
        String productID = input.nextLine().toUpperCase();

        // Query Time Spent
        System.out.println("Time Spent making the Product: ");
        String timeSpent = input.nextLine().toUpperCase();

        // Query Worker Name
        System.out.println("Worker Name: ");
        String workerName = input.nextLine().toUpperCase();

        // Query Worker Address
        System.out.println("Worker Address: ");
        String workerAddress = input.nextLine().toUpperCase();

        // Query Controller Name
        System.out.println("Quality Controller Name: ");
        String qualityControllerName = input.nextLine().toUpperCase();

        // Query Controller Address
        System.out.println("Quality Controller Address: ");
        String qualityControllerAddress = input.nextLine().toUpperCase();

        // Query isRepaired
        System.out.println("Did the product need repaired?(yes/no)");
        String isRepaired = input.nextLine().toUpperCase();

        // Query for Staff name and address
        // and date for product and repair tables
        String technicalStaffName = "";
        String technicalStaffAddress = "";
        String date = "";
        if (isRepaired.equalsIgnoreCase("yes") || isRepaired.equalsIgnoreCase("ye")
                || isRepaired.equalsIgnoreCase("y")) {
            System.out.println("Technical Staff Name: ");
            technicalStaffName = input.nextLine().toUpperCase();

            System.out.println("Technical Staff Address: ");
```

```java
        // and date for product and repair tables
        String technicalStaffName = "";
        String technicalStaffAddress = "";
        String date = "";
        if (isRepaired.equalsIgnoreCase("yes") || isRepaired.equalsIgnoreCase("ye")
                || isRepaired.equalsIgnoreCase("y")) {
            System.out.println("Technical Staff Name: ");
            technicalStaffName = input.nextLine().toUpperCase();

            System.out.println("Technical Staff Address: ");
            technicalStaffAddress = input.nextLine().toUpperCase();

            System.out.println("Date Repaired(yyyy-MM-dd): ");
            date = input.nextLine().toUpperCase();
        }

        // Query size of product
        System.out.println("Size of Product: ");
        String size = input.nextLine().toUpperCase();

        String defect = "";
        if (isRepaired.equalsIgnoreCase("yes") || isRepaired.equalsIgnoreCase("ye")
                || isRepaired.equalsIgnoreCase("y")) {
            defect = "yes";
        } else {
            defect = "no";
        }

        // Query isCertified
        String certified = "";
        System.out.println("Is the Product certified?(yes/no)");
        String isCertified = input.nextLine().toUpperCase();
        if (isCertified.equalsIgnoreCase("yes") || isCertified.equalsIgnoreCase("ye")
                || isCertified.equalsIgnoreCase("y")) {
            certified = "yes";
        } else {
            certified = "no";
        }

        // Query Type of product for future queries
```

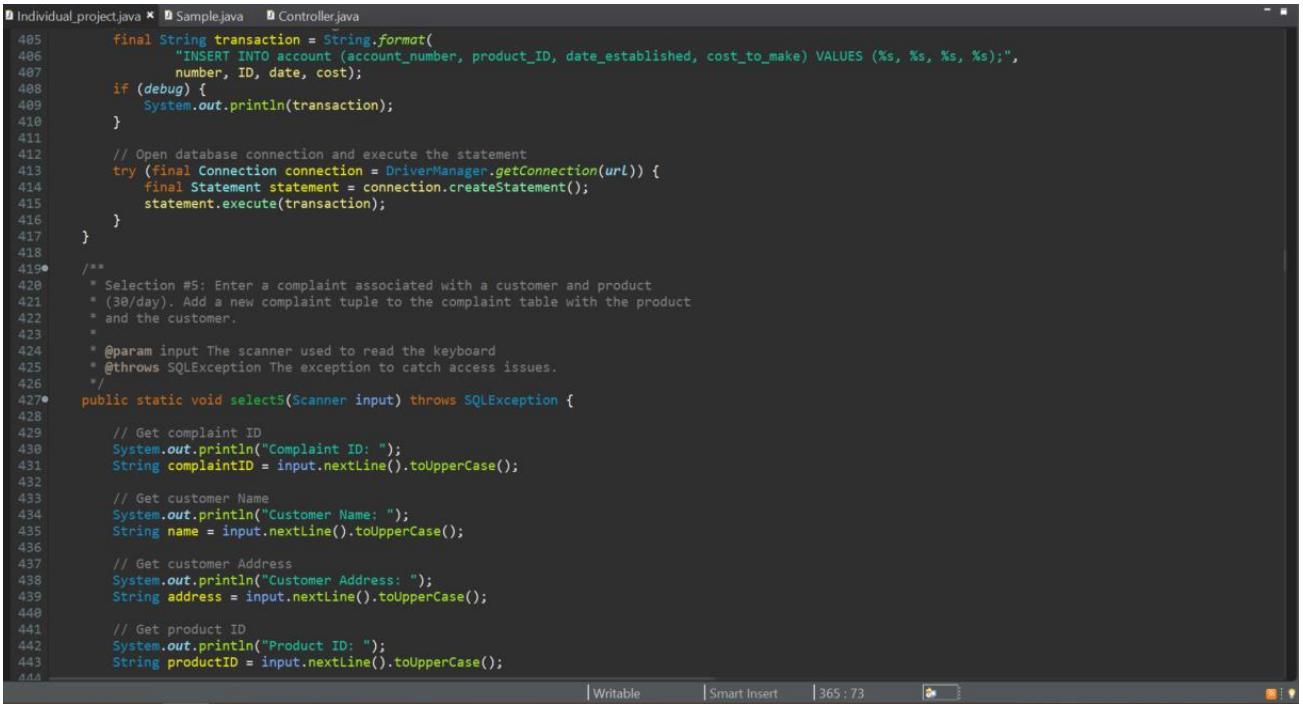
```
261
262         // Query Type of product for future queries
263         System.out.println("Type of Product (1,2, or 3): ");
264         String type = input.nextLine().toUpperCase();
265
266         // Query Get the specific attributes of the records for the corresponding
267         // product#
268         String nameOfMajorSoftware = "";
269         String color = "";
270         String weight = "";
271         String transactionBuilder = "";
272         if (type.equals("1")) {
273             System.out.println("Name of Major Software: ");
274             nameOfMajorSoftware = input.nextLine().toUpperCase();
275             transactionBuilder = String.format(
276                     "INSERT INTO product_1 (product_ID, NAME_OF_MAJOR_SOFTWARE) VALUES (%s, '%s');", productID,
277                     nameOfMajorSoftware);
278         } else if (type.equals("2")) {
279             System.out.println("Color: ");
280             color = input.nextLine().toUpperCase();
281             transactionBuilder = String.format("INSERT INTO product_2 (product_ID, color) VALUES (%s, '%s');",
282                     productID, color);
283         } else {
284             System.out.println("Weight: ");
285             weight = input.nextLine().toUpperCase();
286             transactionBuilder = String.format("INSERT INTO product_3 (product_ID, weight) VALUES (%s, %s);", productID,
287                     weight);
288         }
289
290         // Build Product table insertion Statement
291         final String transaction1 = String.format(
292                 "INSERT INTO product (ID, date, time_spent, worker_name, worker_address, quality_controller_name, quality_controller_address, technical_staff_
293                 productID, timeSpent, workerName, workerAddress, qualityControllerName, qualityControllerAddress,
294                 technicalStaffName, technicalStaffAddress, size, defect, certified);
295
296         // Build Product # table insertion statement
297         final String transaction2 = transactionBuilder;
298
299         // Open connection to database
300         try (final Connection connection = DriverManager.getConnection(url)) {
```

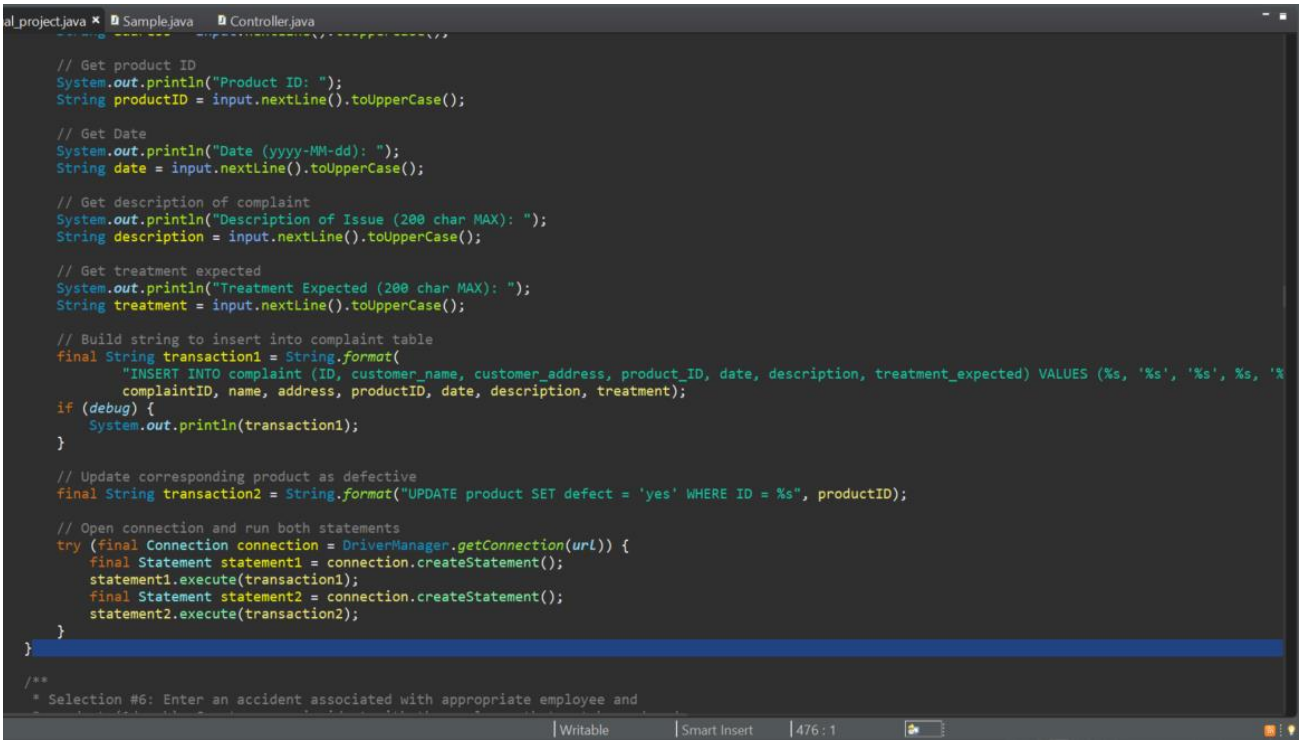
Individual_project.java × Sample.java Controller.java

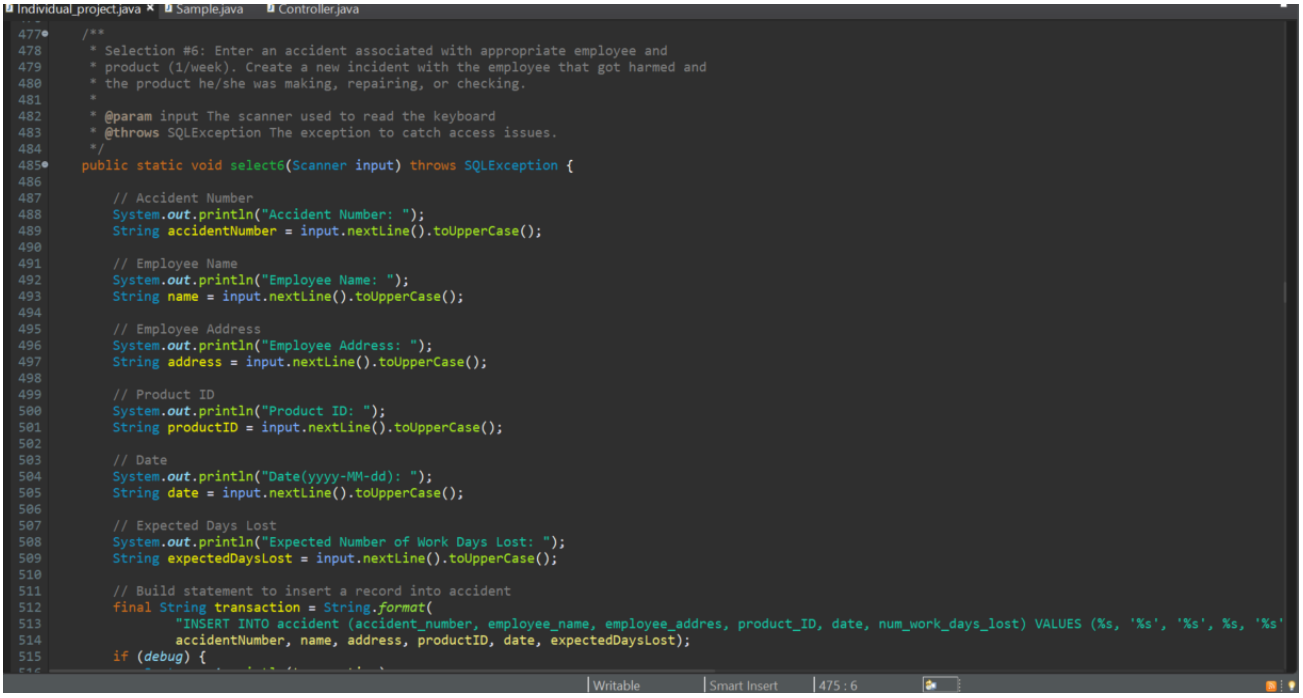
```
298
299         // Open connection to database
300         try (final Connection connection = DriverManager.getConnection(url)) {
301
302             // Execute first statement
303             final Statement statement1 = connection.createStatement();
304             statement1.execute(transaction1);
305             // Execute second statement
306             final Statement statement2 = connection.createStatement();
307             statement2.execute(transaction2);
308
309             // If is repaired, execute third statement updating repaired table
310             if (isRepaired.equalsIgnoreCase("yes") || isRepaired.equalsIgnoreCase("ye")
311                     || isRepaired.equalsIgnoreCase("y")) {
312                 final String transaction3 = String.format(
313                         "INSERT INTO repair (product_ID, quality_controller_name, quality_controller_address, technical_staff_name, technical_staff_address, d
314                         productID, qualityControllerName, qualityControllerAddress, technicalStaffName,
315                         technicalStaffAddress, date);
316                 final Statement statement3 = connection.createStatement();
317                 statement3.execute(transaction3);
318                 if (debug)
319                     System.out.println(transaction3);
320
321             }
322         }
323
324         if (debug) {
325             System.out.println(transaction1);
326             System.out.println(transaction2);
327         }
328
329     }
330
331     /**
332      * Selection #3: Enter a customer associated with some products (50/day). Add a
333      * customer and associate them with the products they bought in the database by
334      * the purchase table.
335      *
336      * @param input The scanner used to read the keyboard
337      * @throws SQLException The exception to catch access issues.
```

```java
    /**
     * Selection #3: Enter a customer associated with some products (50/day). Add a
     * customer and associate them with the products they bought in the database by
     * the purchase table.
     *
     * @param input The scanner used to read the keyboard
     * @throws SQLException The exception to catch access issues.
     */
    public static void select3(Scanner input) throws SQLException {

        // Get customer name
        System.out.println("Customer Name: ");
        String name = input.nextLine().toUpperCase();

        // Get customer address
        System.out.println("Customer Address: ");
        String address = input.nextLine().toUpperCase();

        // Query all products associated with customer for priming statement
        System.out.println("Which products is the customer associated?");
        System.out.println("Enter Product ID or leave blank to finish.");
        String products = input.nextLine().toUpperCase();

        // Build first query
        final String transaction1 = String.format("INSERT INTO customer (name, address) VALUES ('%s', '%s');", name,
                address);

        // Loop through all products entered with built query to insert into purchase
        // table while keeping database connection open
        try (final Connection connection = DriverManager.getConnection(url)) {
            final Statement statement1 = connection.createStatement();
            statement1.execute(transaction1);
            while (!products.equals("") || !products.isEmpty()) {
                final String transaction2 = String.format(
                        "INSERT INTO purchase (name, address, product_ID) VALUES ('%s', '%s', %s);", name, address,
                        products);
                if (debug) {
                    System.out.println(transaction2);
                }
```

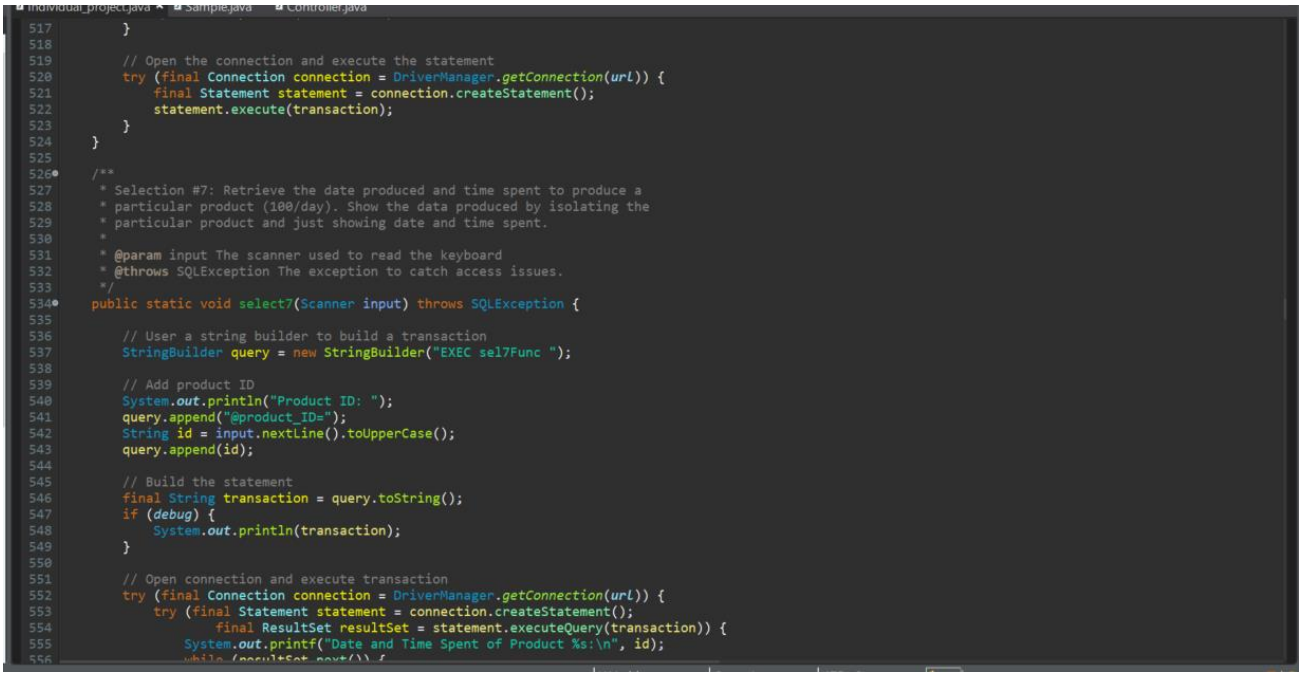
```java
                }
                final Statement statement2 = connection.createStatement();
                statement2.execute(transaction2);

                System.out.println("Enter Product ID or leave blank to finish.");
                products = input.nextLine().toUpperCase();
            }
        }
    }

    /**
     * Selection #4: Create a new account associated with a product (40/day). Create
     * a new account for a product and add it to the account table.
     *
     * @param input The scanner used to read the keyboard
     * @throws SQLException The exception to catch access issues.
     */
    public static void select4(Scanner input) throws SQLException {

        // Get account number
        System.out.println("Account Number: ");
        String number = input.nextLine().toUpperCase();

        // Get product ID
        System.out.println("Product ID: ");
        String ID = input.nextLine().toUpperCase();

        // Get date
        System.out.println("Date Established(yyyy-MM-dd): ");
        String date = input.nextLine().toUpperCase();

        // Get cost
        System.out.println("Cost of Product: ");
        String cost = input.nextLine().toUpperCase();

        // Build the statement string
        final String transaction = String.format(
                "INSERT INTO account (account_number, product_ID, date_established, cost_to_make) VALUES (%s, %s, %s, %s);",
                number, ID, date, cost);
```


Individual_project.java
Sample.java
Controller.java
final String transaction = String.format(
"INSERT INTO account (account_number, product_ID, date_established, cost_to_make) VALUES (%s, %s, %s, %s);",
number, ID, date, cost);
if (debug) {
System.out.println(transaction);
}
// Open database connection and execute the statement
try (final Connection connection = DriverManager.getConnection(url)) {
final Statement statement = connection.createStatement();
statement.execute(transaction);
}
}
/**
* Selection #5: Enter a complaint associated with a customer and product
* (30/day). Add a new complaint tuple to the complaint table with the product
* and the customer.
*
* @param input The scanner used to read the keyboard
* @throws SQLException The exception to catch access issues.
*/
public static void select5(Scanner input) throws SQLException {
// Get complaint ID
System.out.println("Complaint ID: ");
String complaintID = input.nextLine().toUpperCase();
// Get customer Name
System.out.println("Customer Name: ");
String name = input.nextLine().toUpperCase();
// Get customer Address
System.out.println("Customer Address: ");
String address = input.nextLine().toUpperCase();
// Get product ID
System.out.println("Product ID: ");
String productID = input.nextLine().toUpperCase();
Writable
Smart Insert
365 : 73

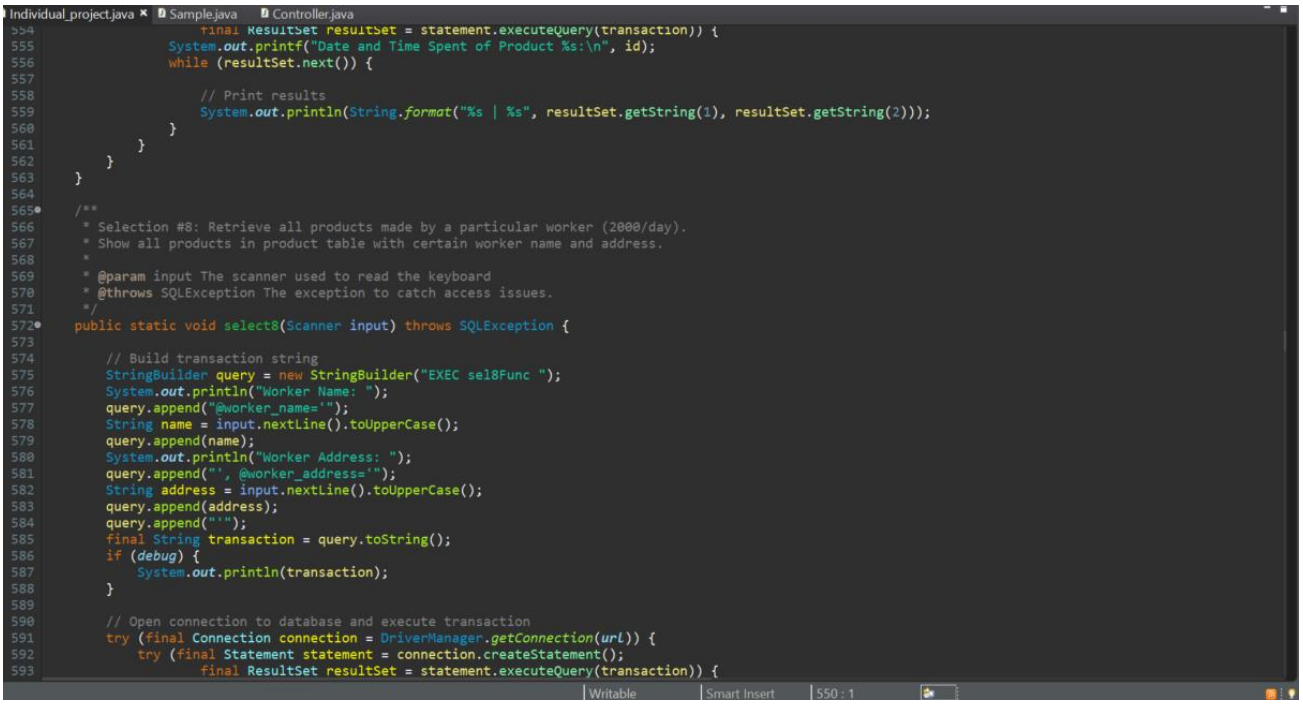

al_project.java
Sample.java
Controller.java
// Get product ID
System.out.println("Product ID: ");
String productID = input.nextLine().toUpperCase();
// Get Date
System.out.println("Date (yyyy-MM-dd): ");
String date = input.nextLine().toUpperCase();
// Get description of complaint
System.out.println("Description of Issue (200 char MAX): ");
String description = input.nextLine().toUpperCase();
// Get treatment expected
System.out.println("Treatment Expected (200 char MAX): ");
String treatment = input.nextLine().toUpperCase();
// Build string to insert into complaint table
final String transaction1 = String.format(
"INSERT INTO complaint (ID, customer_name, customer_address, product_ID, date, description, treatment_expected) VALUES (%s, '%s', '%s', %s, '%
complaintID, name, address, productID, date, description, treatment);
if (debug) {
System.out.println(transaction1);
}
// Update corresponding product as defective
final String transaction2 = String.format("UPDATE product SET defect = 'yes' WHERE ID = %s", productID);
// Open connection and run both statements
try (final Connection connection = DriverManager.getConnection(url)) {
final Statement statement1 = connection.createStatement();
statement1.execute(transaction1);
final Statement statement2 = connection.createStatement();
statement2.execute(transaction2);
}
}
/**
* Selection #6: Enter an accident associated with appropriate employee and
Writable
Smart Insert
476 : 1



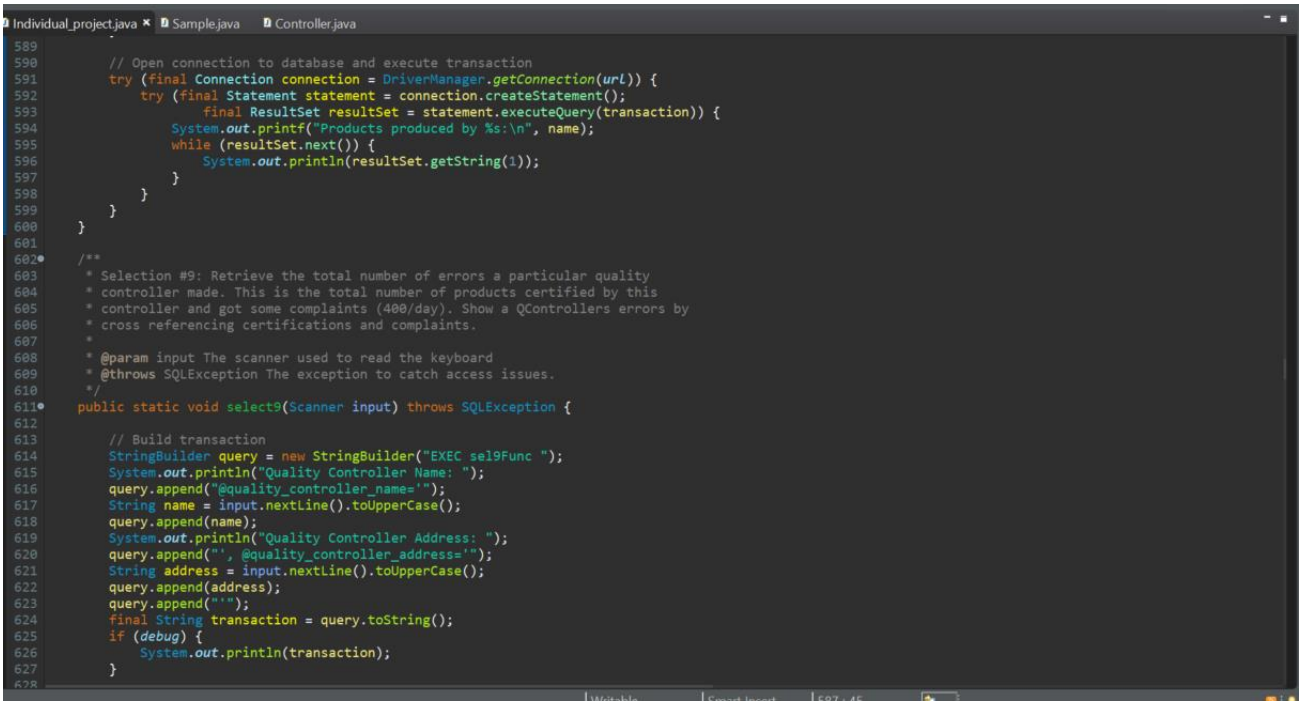
```java
/**
 * Selection #6: Enter an accident associated with appropriate employee and
 * product (1/week). Create a new incident with the employee that got harmed and
 * the product he/she was making, repairing, or checking.
 *
 * @param input The scanner used to read the keyboard
 * @throws SQLException The exception to catch access issues.
 */
public static void select6(Scanner input) throws SQLException {

    // Accident Number
    System.out.println("Accident Number: ");
    String accidentNumber = input.nextLine().toUpperCase();

    // Employee Name
    System.out.println("Employee Name: ");
    String name = input.nextLine().toUpperCase();

    // Employee Address
    System.out.println("Employee Address: ");
    String address = input.nextLine().toUpperCase();

    // Product ID
    System.out.println("Product ID: ");
    String productID = input.nextLine().toUpperCase();

    // Date
    System.out.println("Date(yyyy-MM-dd): ");
    String date = input.nextLine().toUpperCase();

    // Expected Days Lost
    System.out.println("Expected Number of Work Days Lost: ");
    String expectedDaysLost = input.nextLine().toUpperCase();

    // Build statement to insert a record into accident
    final String transaction = String.format(
            "INSERT INTO accident (accident_number, employee_name, employee_addres, product_ID, date, num_work_days_lost) VALUES (%s, '%s', '%s', %s, '%s'
            accidentNumber, name, address, productID, date, expectedDaysLost);
    if (debug) {
```



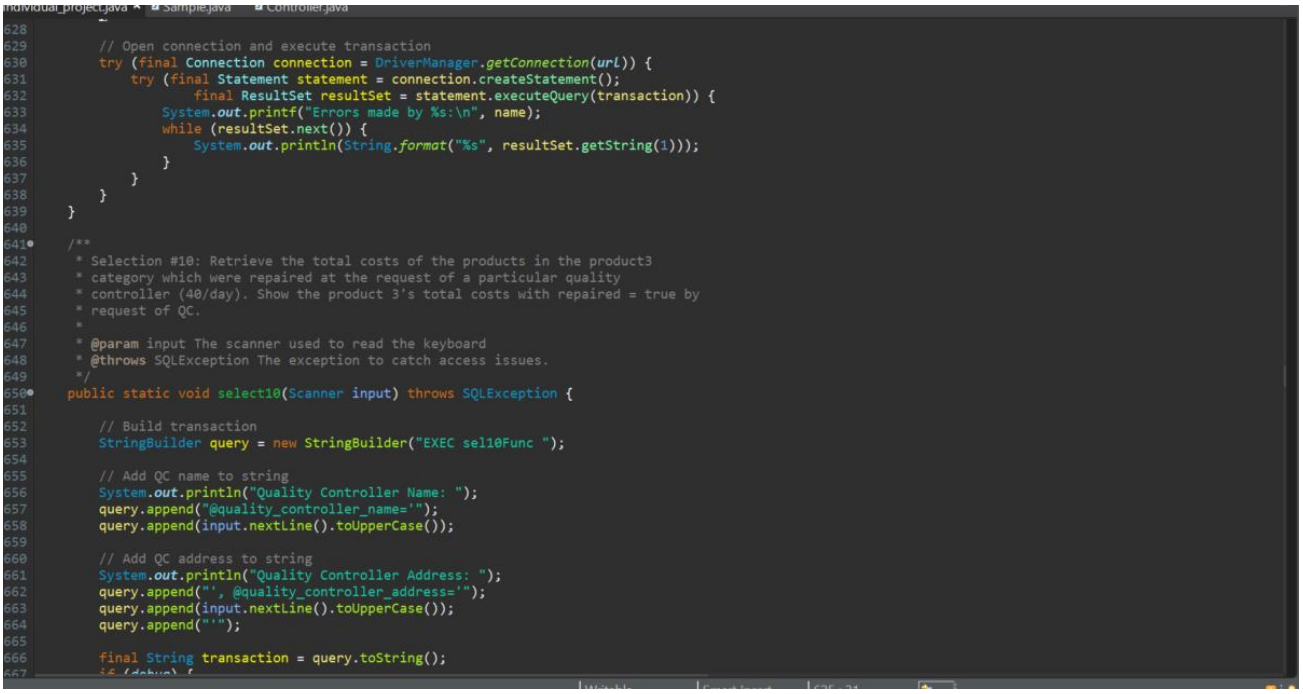
```java
    }

    // Open the connection and execute the statement
    try (final Connection connection = DriverManager.getConnection(url)) {
        final Statement statement = connection.createStatement();
        statement.execute(transaction);
    }
}

/**
 * Selection #7: Retrieve the date produced and time spent to produce a
 * particular product (100/day). Show the data produced by isolating the
 * particular product and just showing date and time spent.
 *
 * @param input The scanner used to read the keyboard
 * @throws SQLException The exception to catch access issues.
 */
public static void select7(Scanner input) throws SQLException {

    // User a string builder to build a transaction
    StringBuilder query = new StringBuilder("EXEC sel7Func ");

    // Add product ID
    System.out.println("Product ID: ");
    query.append("@product_ID=");
    String id = input.nextLine().toUpperCase();
    query.append(id);

    // Build the statement
    final String transaction = query.toString();
    if (debug) {
        System.out.println(transaction);
    }

    // Open connection and execute transaction
    try (final Connection connection = DriverManager.getConnection(url)) {
        try (final Statement statement = connection.createStatement();
                final ResultSet resultSet = statement.executeQuery(transaction)) {
            System.out.printf("Date and Time Spent of Product %s:\n", id);
```



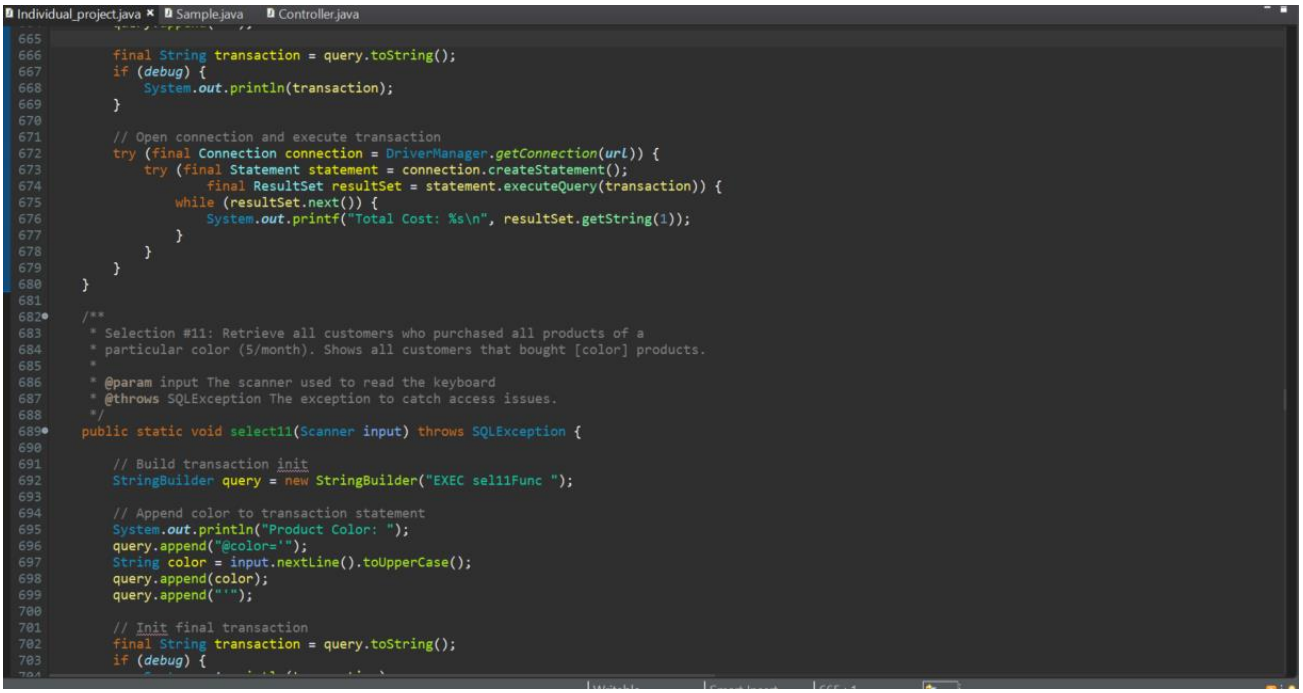
```
Individual_project.java × | Sample.java | Controller.java
554                 final ResultSet resultSet = statement.executeQuery(transaction)) {
555                 System.out.printf("Date and Time Spent of Product %s:\n", id);
556                 while (resultSet.next()) {
557
558                     // Print results
559                     System.out.println(String.format("%s | %s", resultSet.getString(1), resultSet.getString(2)));
560                 }
561             }
562         }
563     }
564
565     /**
566      * Selection #8: Retrieve all products made by a particular worker (2000/day).
567      * Show all products in product table with certain worker name and address.
568      *
569      * @param input The scanner used to read the keyboard
570      * @throws SQLException The exception to catch access issues.
571      */
572     public static void select8(Scanner input) throws SQLException {
573
574         // Build transaction string
575         StringBuilder query = new StringBuilder("EXEC sel8Func ");
576         System.out.println("Worker Name: ");
577         query.append("@worker_name='");
578         String name = input.nextLine().toUpperCase();
579         query.append(name);
580         System.out.println("Worker Address: ");
581         query.append("', @worker_address='");
582         String address = input.nextLine().toUpperCase();
583         query.append(address);
584         query.append("'");
585         final String transaction = query.toString();
586         if (debug) {
587             System.out.println(transaction);
588         }
589
590         // Open connection to database and execute transaction
591         try (final Connection connection = DriverManager.getConnection(url)) {
592             try (final Statement statement = connection.createStatement();
593                     final ResultSet resultSet = statement.executeQuery(transaction)) {
                                                    Writable | Smart Insert | 550 : 1
```



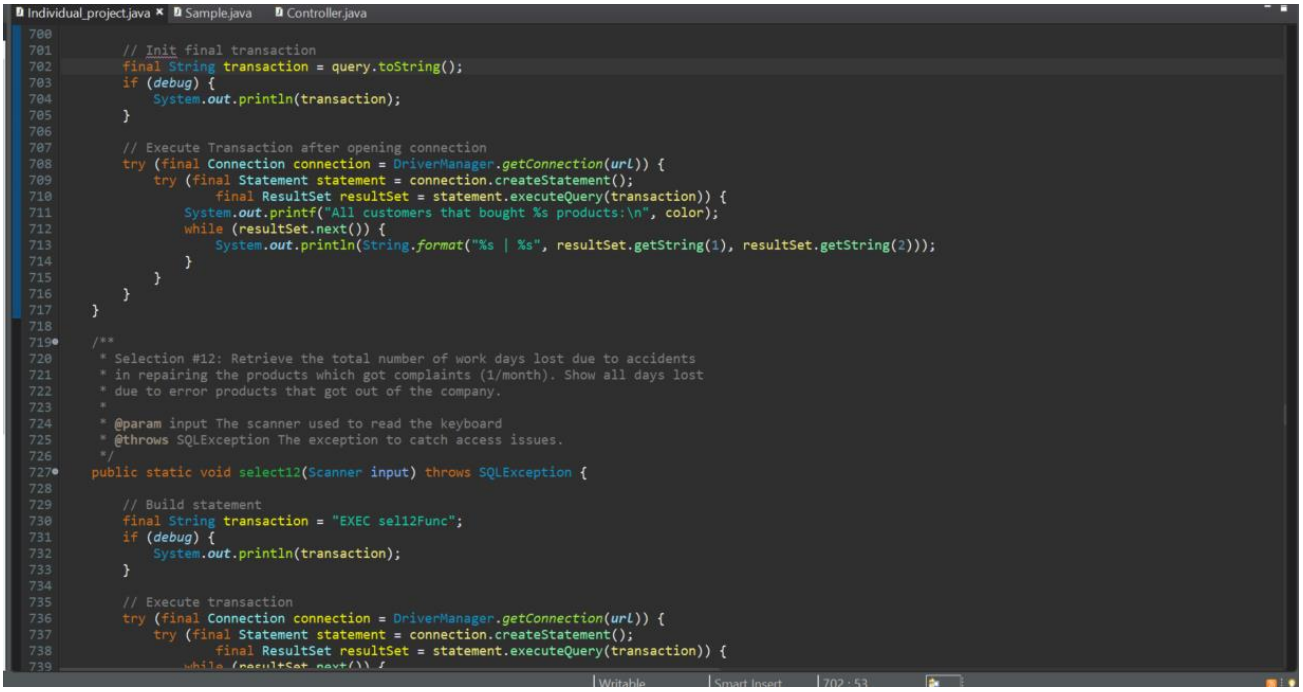
```
Individual_project.java × | Sample.java | Controller.java
589
590         // Open connection to database and execute transaction
591         try (final Connection connection = DriverManager.getConnection(url)) {
592             try (final Statement statement = connection.createStatement();
593                     final ResultSet resultSet = statement.executeQuery(transaction)) {
594                 System.out.printf("Products produced by %s:\n", name);
595                 while (resultSet.next()) {
596                     System.out.println(resultSet.getString(1));
597                 }
598             }
599         }
600     }
601
602     /**
603      * Selection #9: Retrieve the total number of errors a particular quality
604      * controller made. This is the total number of products certified by this
605      * controller and got some complaints (400/day). Show a QControllers errors by
606      * cross referencing certifications and complaints.
607      *
608      * @param input The scanner used to read the keyboard
609      * @throws SQLException The exception to catch access issues.
610      */
611     public static void select9(Scanner input) throws SQLException {
612
613         // Build transaction
614         StringBuilder query = new StringBuilder("EXEC sel9Func ");
615         System.out.println("Quality Controller Name: ");
616         query.append("@quality_controller_name='");
617         String name = input.nextLine().toUpperCase();
618         query.append(name);
619         System.out.println("Quality Controller Address: ");
620         query.append("', @quality_controller_address='");
621         String address = input.nextLine().toUpperCase();
622         query.append(address);
623         query.append("'");
624         final String transaction = query.toString();
625         if (debug) {
626             System.out.println(transaction);
627         }
628
```



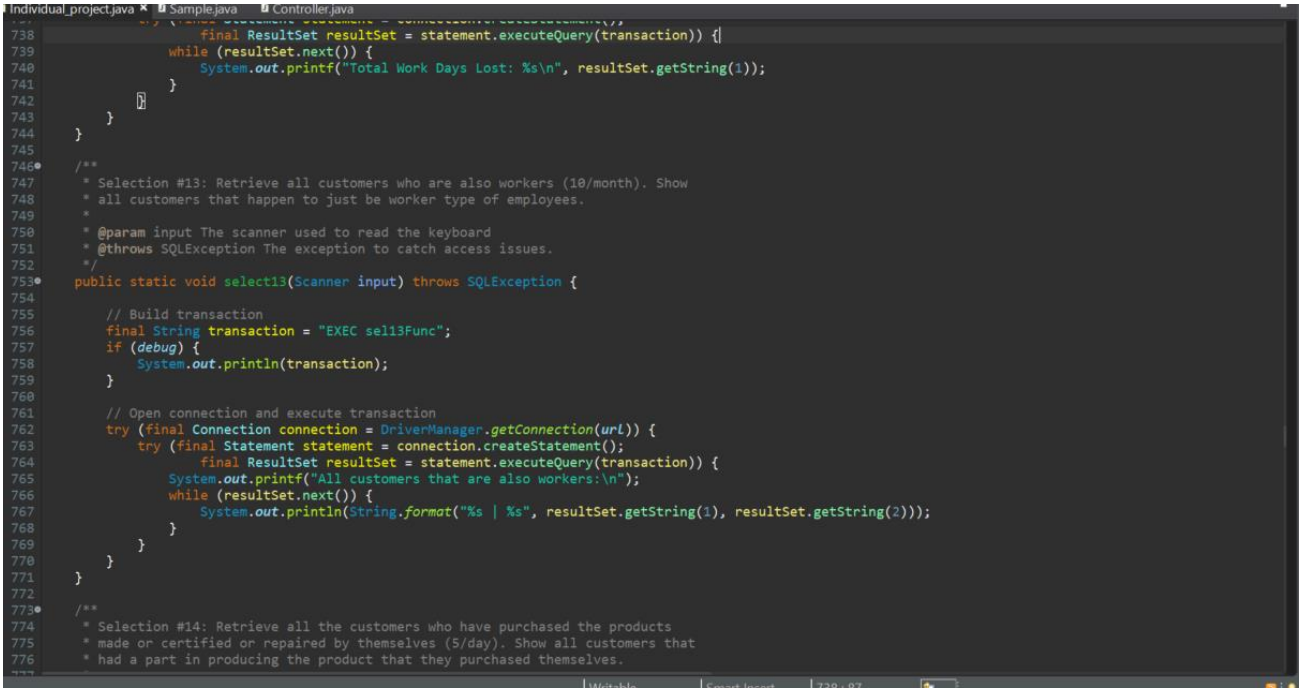
```
628
629         // Open connection and execute transaction
630         try (final Connection connection = DriverManager.getConnection(url)) {
631             try (final Statement statement = connection.createStatement();
632                     final ResultSet resultSet = statement.executeQuery(transaction)) {
633                 System.out.printf("Errors made by %s:\n", name);
634                 while (resultSet.next()) {
635                     System.out.println(String.format("%s", resultSet.getString(1)));
636                 }
637             }
638         }
639     }
640
641     /**
642      * Selection #10: Retrieve the total costs of the products in the product3
643      * category which were repaired at the request of a particular quality
644      * controller (40/day). Show the product 3's total costs with repaired = true by
645      * request of QC.
646      *
647      * @param input The scanner used to read the keyboard
648      * @throws SQLException The exception to catch access issues.
649      */
650     public static void select10(Scanner input) throws SQLException {
651
652         // Build transaction
653         StringBuilder query = new StringBuilder("EXEC sel10Func ");
654
655         // Add QC name to string
656         System.out.println("Quality Controller Name: ");
657         query.append("@quality_controller_name='");
658         query.append(input.nextLine().toUpperCase());
659
660         // Add QC address to string
661         System.out.println("Quality Controller Address: ");
662         query.append("', @quality_controller_address='");
663         query.append(input.nextLine().toUpperCase());
664         query.append("'");
665
666         final String transaction = query.toString();
667         if (debug) {
```



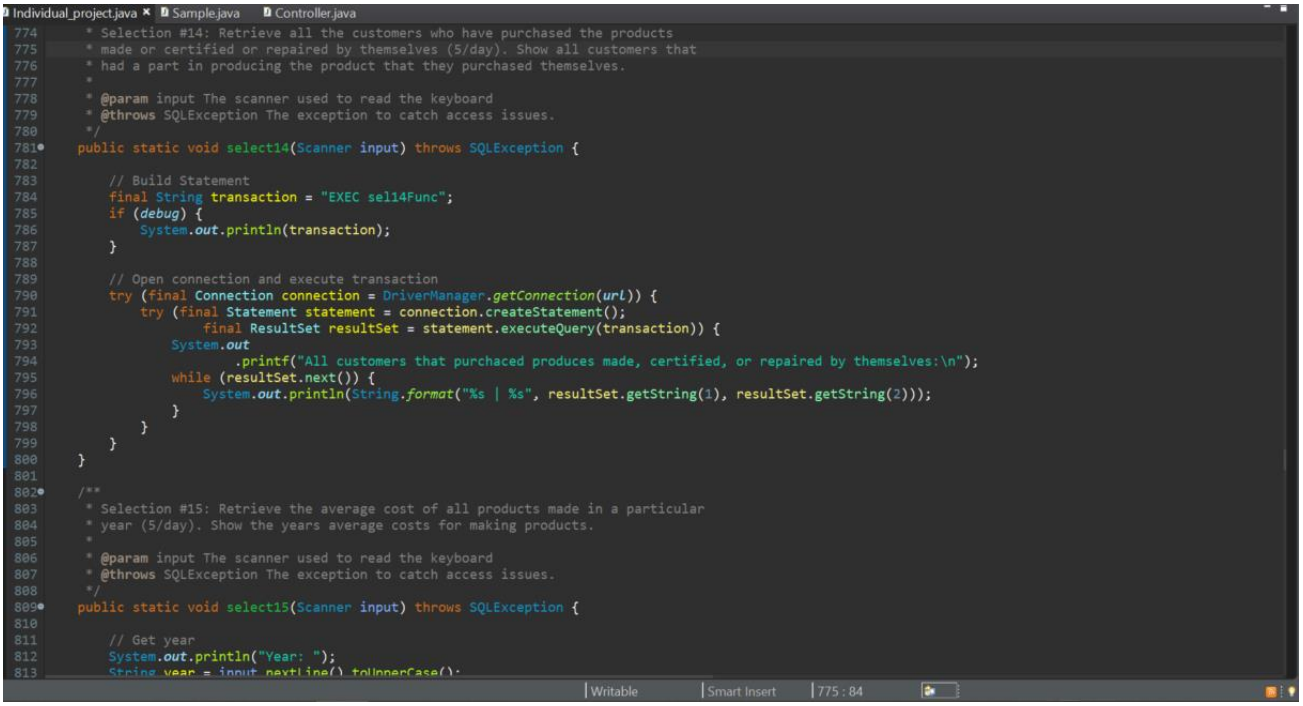
```
Individual_project.java   Sample.java   Controller.java
665
666         final String transaction = query.toString();
667         if (debug) {
668             System.out.println(transaction);
669         }
670
671         // Open connection and execute transaction
672         try (final Connection connection = DriverManager.getConnection(url)) {
673             try (final Statement statement = connection.createStatement();
674                     final ResultSet resultSet = statement.executeQuery(transaction)) {
675                 while (resultSet.next()) {
676                     System.out.printf("Total Cost: %s\n", resultSet.getString(1));
677                 }
678             }
679         }
680     }
681
682     /**
683      * Selection #11: Retrieve all customers who purchased all products of a
684      * particular color (5/month). Shows all customers that bought [color] products.
685      *
686      * @param input The scanner used to read the keyboard
687      * @throws SQLException The exception to catch access issues.
688      */
689     public static void select11(Scanner input) throws SQLException {
690
691         // Build transaction init
692         StringBuilder query = new StringBuilder("EXEC sel11Func ");
693
694         // Append color to transaction statement
695         System.out.println("Product Color: ");
696         query.append("@color='");
697         String color = input.nextLine().toUpperCase();
698         query.append(color);
699         query.append("'");
700
701         // Init final transaction
702         final String transaction = query.toString();
703         if (debug) {
```



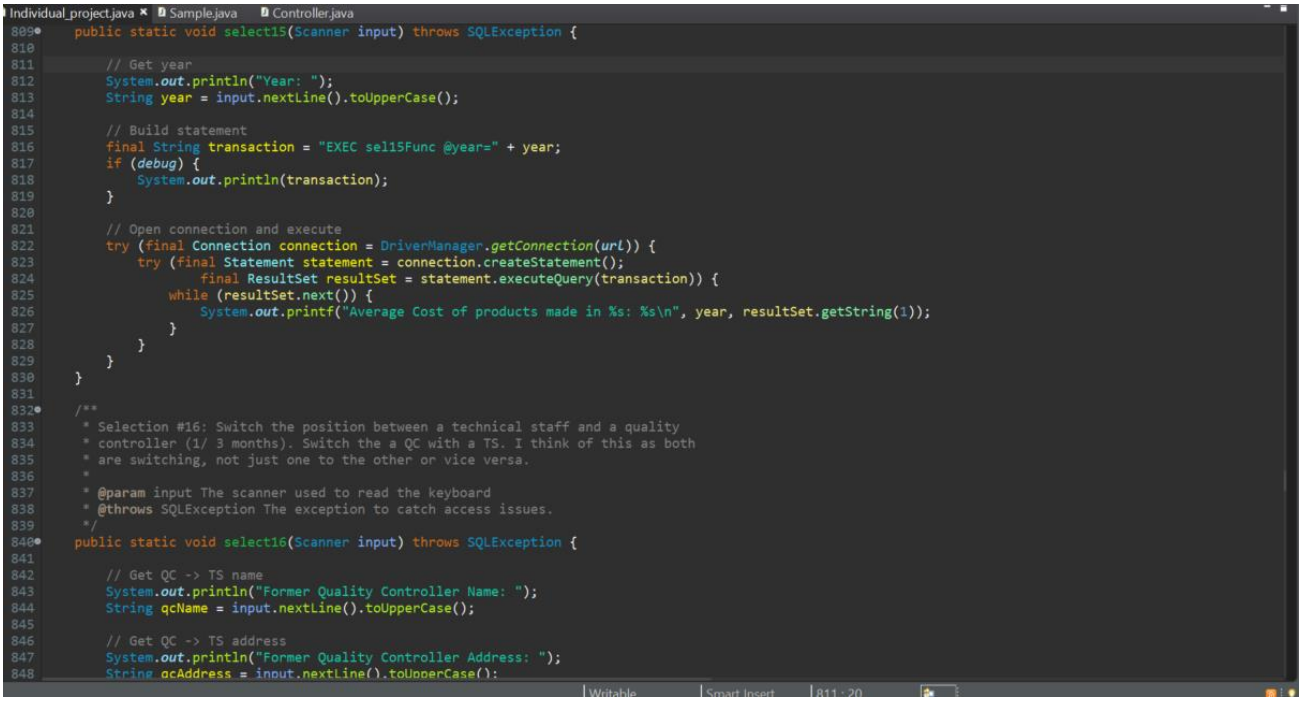
```
Individual_project.java × Sample.java Controller.java
700
701         // Init final transaction
702         final String transaction = query.toString();
703         if (debug) {
704             System.out.println(transaction);
705         }
706
707         // Execute Transaction after opening connection
708         try (final Connection connection = DriverManager.getConnection(url)) {
709             try (final Statement statement = connection.createStatement();
710                  final ResultSet resultSet = statement.executeQuery(transaction)) {
711                 System.out.printf("All customers that bought %s products:\n", color);
712                 while (resultSet.next()) {
713                     System.out.println(String.format("%s | %s", resultSet.getString(1), resultSet.getString(2)));
714                 }
715             }
716         }
717     }
718
719     /**
720      * Selection #12: Retrieve the total number of work days lost due to accidents
721      * in repairing the products which got complaints (1/month). Show all days lost
722      * due to error products that got out of the company.
723      *
724      * @param input The scanner used to read the keyboard
725      * @throws SQLException The exception to catch access issues.
726      */
727     public static void select12(Scanner input) throws SQLException {
728
729         // Build statement
730         final String transaction = "EXEC sel12Func";
731         if (debug) {
732             System.out.println(transaction);
733         }
734
735         // Execute transaction
736         try (final Connection connection = DriverManager.getConnection(url)) {
737             try (final Statement statement = connection.createStatement();
738                  final ResultSet resultSet = statement.executeQuery(transaction)) {
739                 while (resultSet.next()) {
```



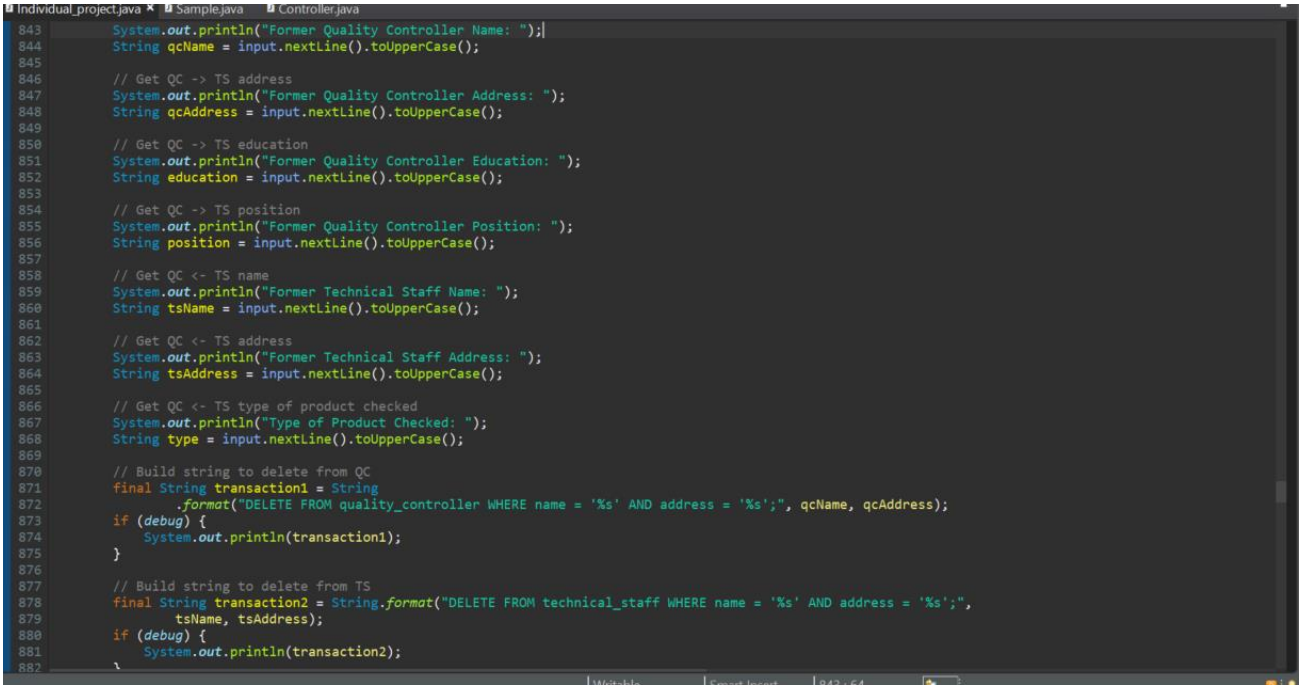
```
Individual_project.java × Sample.java Controller.java
738                  final ResultSet resultSet = statement.executeQuery(transaction)) {
739                 while (resultSet.next()) {
740                     System.out.printf("Total Work Days Lost: %s\n", resultSet.getString(1));
741                 }
742             }
743         }
744     }
745
746     /**
747      * Selection #13: Retrieve all customers who are also workers (10/month). Show
748      * all customers that happen to just be worker type of employees.
749      *
750      * @param input The scanner used to read the keyboard
751      * @throws SQLException The exception to catch access issues.
752      */
753     public static void select13(Scanner input) throws SQLException {
754
755         // Build transaction
756         final String transaction = "EXEC sel13Func";
757         if (debug) {
758             System.out.println(transaction);
759         }
760
761         // Open connection and execute transaction
762         try (final Connection connection = DriverManager.getConnection(url)) {
763             try (final Statement statement = connection.createStatement();
764                  final ResultSet resultSet = statement.executeQuery(transaction)) {
765                 System.out.printf("All customers that are also workers:\n");
766                 while (resultSet.next()) {
767                     System.out.println(String.format("%s | %s", resultSet.getString(1), resultSet.getString(2)));
768                 }
769             }
770         }
771     }
772
773     /**
774      * Selection #14: Retrieve all the customers who have purchased the products
775      * made or certified or repaired by themselves (5/day). Show all customers that
776      * had a part in producing the product that they purchased themselves.
```



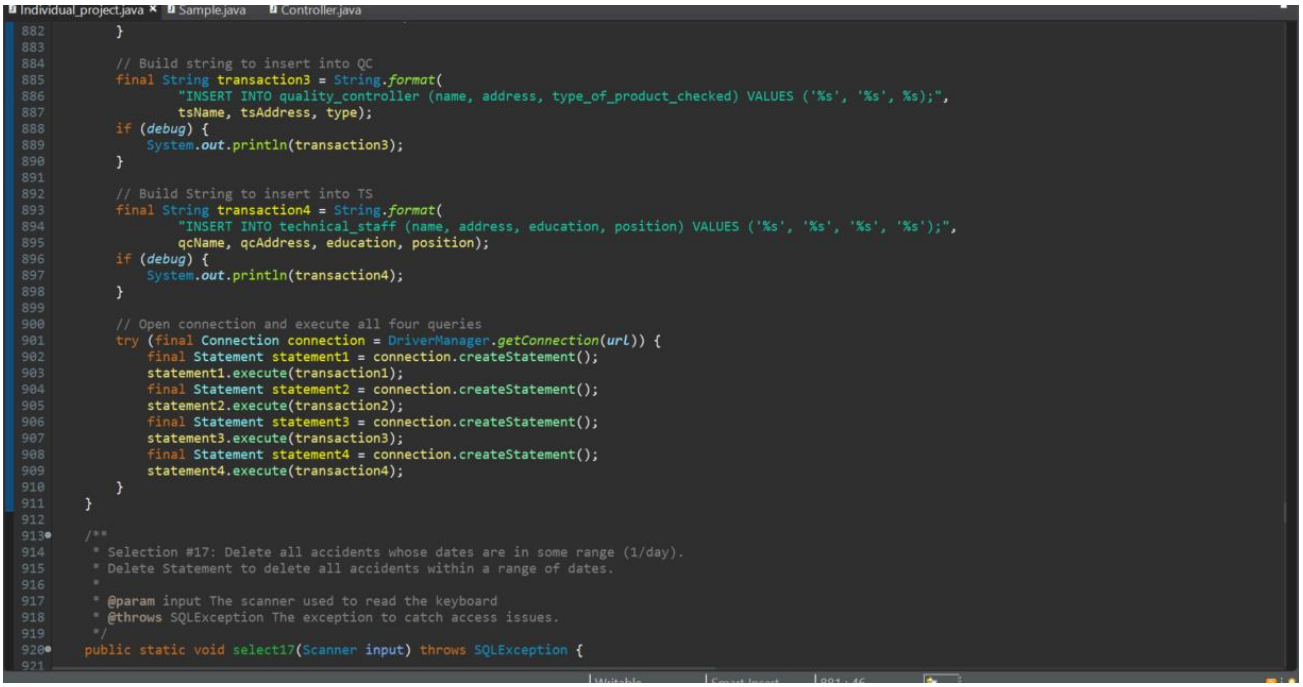
```
Individual_project.java × Sample.java Controller.java
774      * Selection #14: Retrieve all the customers who have purchased the products
775      * made or certified or repaired by themselves (5/day). Show all customers that
776      * had a part in producing the product that they purchased themselves.
777      *
778      * @param input The scanner used to read the keyboard
779      * @throws SQLException The exception to catch access issues.
780      */
781     public static void select14(Scanner input) throws SQLException {
782
783         // Build Statement
784         final String transaction = "EXEC sel14Func";
785         if (debug) {
786             System.out.println(transaction);
787         }
788
789         // Open connection and execute transaction
790         try (final Connection connection = DriverManager.getConnection(url)) {
791             try (final Statement statement = connection.createStatement();
792                     final ResultSet resultSet = statement.executeQuery(transaction)) {
793                 System.out
794                         .printf("All customers that purchaced produces made, certified, or repaired by themselves:\n");
795                 while (resultSet.next()) {
796                     System.out.println(String.format("%s | %s", resultSet.getString(1), resultSet.getString(2)));
797                 }
798             }
799         }
800     }
801
802     /**
803      * Selection #15: Retrieve the average cost of all products made in a particular
804      * year (5/day). Show the years average costs for making products.
805      *
806      * @param input The scanner used to read the keyboard
807      * @throws SQLException The exception to catch access issues.
808      */
809     public static void select15(Scanner input) throws SQLException {
810
811         // Get year
812         System.out.println("Year: ");
813         String year = input.nextLine().toUpperCase();
                                  Writable   Smart Insert   775 : 84
```



```
Individual_project.java × Sample.java Controller.java
809     public static void select15(Scanner input) throws SQLException {
810
811         // Get year
812         System.out.println("Year: ");
813         String year = input.nextLine().toUpperCase();
814
815         // Build statement
816         final String transaction = "EXEC sel15Func @year=" + year;
817         if (debug) {
818             System.out.println(transaction);
819         }
820
821         // Open connection and execute
822         try (final Connection connection = DriverManager.getConnection(url)) {
823             try (final Statement statement = connection.createStatement();
824                     final ResultSet resultSet = statement.executeQuery(transaction)) {
825                 while (resultSet.next()) {
826                     System.out.printf("Average Cost of products made in %s: %s\n", year, resultSet.getString(1));
827                 }
828             }
829         }
830     }
831
832     /**
833      * Selection #16: Switch the position between a technical staff and a quality
834      * controller (1/ 3 months). Switch the a QC with a TS. I think of this as both
835      * are switching, not just one to the other or vice versa.
836      *
837      * @param input The scanner used to read the keyboard
838      * @throws SQLException The exception to catch access issues.
839      */
840     public static void select16(Scanner input) throws SQLException {
841
842         // Get QC -> TS name
843         System.out.println("Former Quality Controller Name: ");
844         String qcName = input.nextLine().toUpperCase();
845
846         // Get QC -> TS address
847         System.out.println("Former Quality Controller Address: ");
848         String qcAddress = input.nextLine().toUpperCase();
                                  Writable   Smart Insert   811 : 20
```



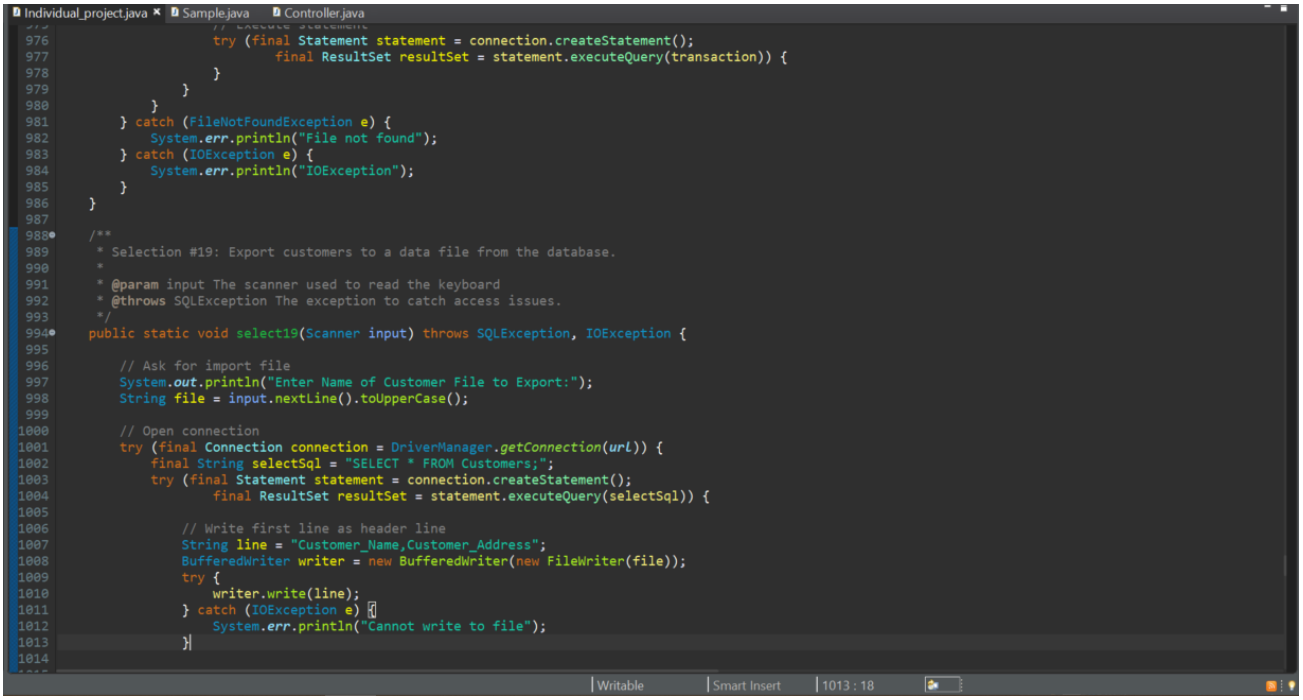
```
Individual_project.java × Sample.java Controller.java
843         System.out.println("Former Quality Controller Name: ");
844         String qcName = input.nextLine().toUpperCase();
845
846         // Get QC -> TS address
847         System.out.println("Former Quality Controller Address: ");
848         String qcAddress = input.nextLine().toUpperCase();
849
850         // Get QC -> TS education
851         System.out.println("Former Quality Controller Education: ");
852         String education = input.nextLine().toUpperCase();
853
854         // Get QC -> TS position
855         System.out.println("Former Quality Controller Position: ");
856         String position = input.nextLine().toUpperCase();
857
858         // Get QC <- TS name
859         System.out.println("Former Technical Staff Name: ");
860         String tsName = input.nextLine().toUpperCase();
861
862         // Get QC <- TS address
863         System.out.println("Former Technical Staff Address: ");
864         String tsAddress = input.nextLine().toUpperCase();
865
866         // Get QC <- TS type of product checked
867         System.out.println("Type of Product Checked: ");
868         String type = input.nextLine().toUpperCase();
869
870         // Build string to delete from QC
871         final String transaction1 = String
872                 .format("DELETE FROM quality_controller WHERE name = '%s' AND address = '%s';", qcName, qcAddress);
873         if (debug) {
874             System.out.println(transaction1);
875         }
876
877         // Build string to delete from TS
878         final String transaction2 = String.format("DELETE FROM technical_staff WHERE name = '%s' AND address = '%s';",
879                 tsName, tsAddress);
880         if (debug) {
881             System.out.println(transaction2);
882         }
```



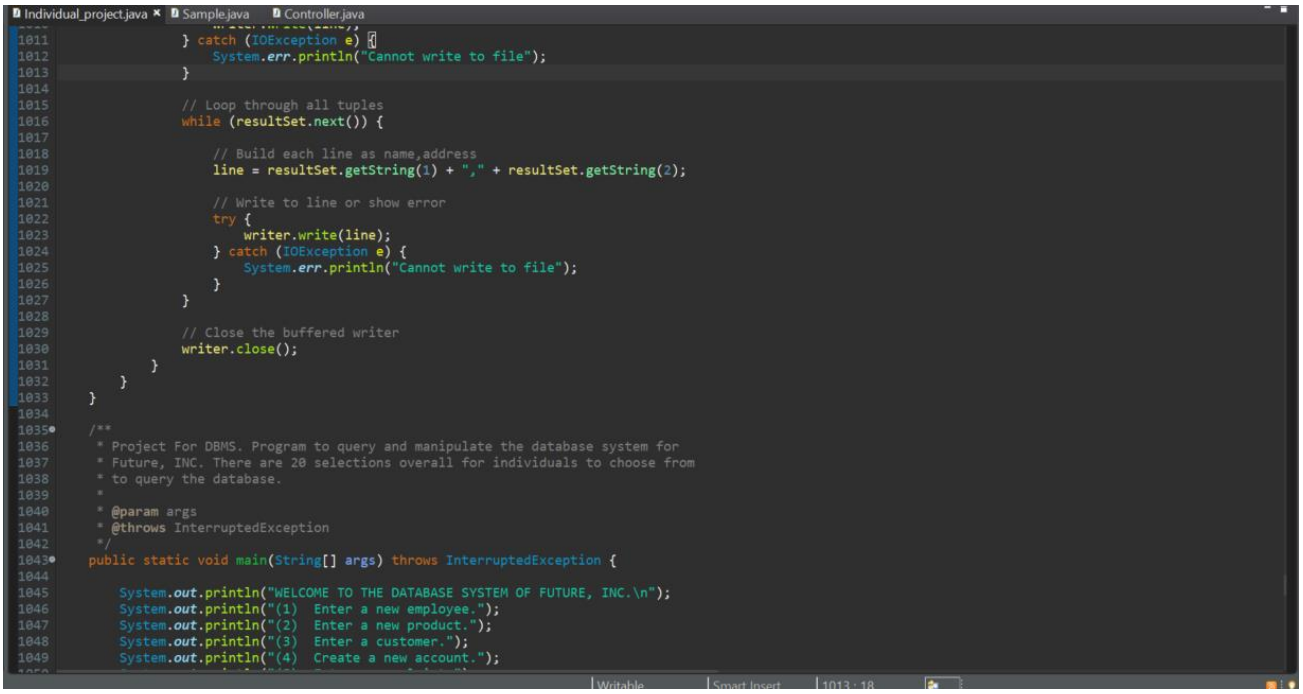
```
Individual_project.java × Sample.java Controller.java
882         }
883
884         // Build string to insert into QC
885         final String transaction3 = String.format(
886                 "INSERT INTO quality_controller (name, address, type_of_product_checked) VALUES ('%s', '%s', %s);",
887                 tsName, tsAddress, type);
888         if (debug) {
889             System.out.println(transaction3);
890         }
891
892         // Build String to insert into TS
893         final String transaction4 = String.format(
894                 "INSERT INTO technical_staff (name, address, education, position) VALUES ('%s', '%s', '%s', '%s');",
895                 qcName, qcAddress, education, position);
896         if (debug) {
897             System.out.println(transaction4);
898         }
899
900         // Open connection and execute all four queries
901         try (final Connection connection = DriverManager.getConnection(url)) {
902             final Statement statement1 = connection.createStatement();
903             statement1.execute(transaction1);
904             final Statement statement2 = connection.createStatement();
905             statement2.execute(transaction2);
906             final Statement statement3 = connection.createStatement();
907             statement3.execute(transaction3);
908             final Statement statement4 = connection.createStatement();
909             statement4.execute(transaction4);
910         }
911     }
912
913     /**
914      * Selection #17: Delete all accidents whose dates are in some range (1/day).
915      * Delete Statement to delete all accidents within a range of dates.
916      *
917      * @param input The scanner used to read the keyboard
918      * @throws SQLException The exception to catch access issues.
919      */
920     public static void select17(Scanner input) throws SQLException {
921
```

```java
Individual_project.java × Sample.java Controller.java
913●    /**
914     * Selection #17: Delete all accidents whose dates are in some range (1/day).
915     * Delete Statement to delete all accidents within a range of dates.
916     *
917     * @param input The scanner used to read the keyboard
918     * @throws SQLException The exception to catch access issues.
919     */
920●   public static void select17(Scanner input) throws SQLException {
921
922         // Get start date
923         System.out.println("Start Date(yyyy-MM-dd): ");
924         String startDate = input.nextLine().toUpperCase();
925
926         // Get end date
927         System.out.println("End Date(yyyy-MM-dd): ");
928         String endDate = input.nextLine().toUpperCase();
929
930         // Build manipulation statement
931         final String transaction = String.format("DELETE FROM accident WHERE date > '%s' AND date < '%s';", startDate,
932                 endDate);
933         if (debug) {
934             System.out.println(transaction);
935         }
936
937         // Execute
938         try (final Connection connection = DriverManager.getConnection(url)) {
939             final Statement statement = connection.createStatement();
940             statement.execute(transaction);
941         }
942     }
943
944●    /**
945     * Selection #18: Import customers from a data file to the database.
946     *
947     * @param input The scanner used to read the keyboard
948     * @throws SQLException The exception to catch access issues.
949     */
950●   public static void select18(Scanner input) throws SQLException {
951
952         // Query file for import
                                                    Writable   Smart Insert   881 : 46
```

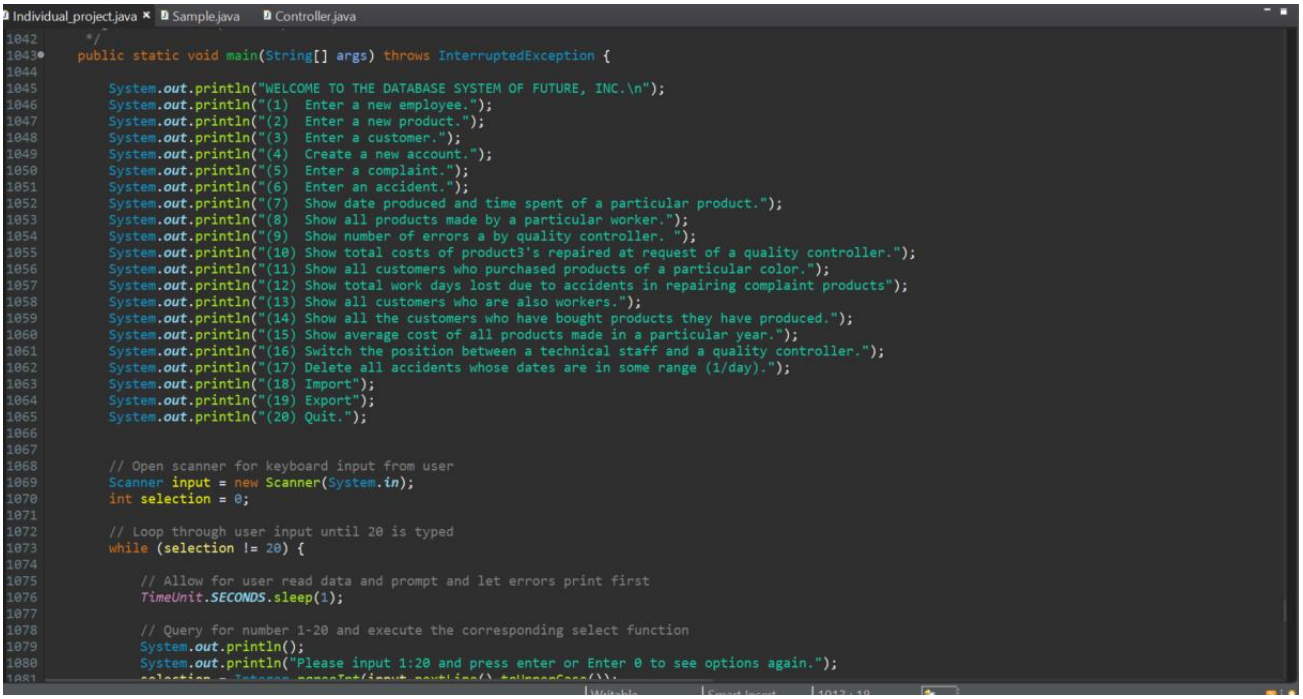
```java
Individual_project.java × Sample.java Controller.java
941         }
942     }
943
944●    /**
945     * Selection #18: Import customers from a data file to the database.
946     *
947     * @param input The scanner used to read the keyboard
948     * @throws SQLException The exception to catch access issues.
949     */
950●   public static void select18(Scanner input) throws SQLException {
951
952         // Query file for import
953         System.out.println("Enter the name of Customer File as Import: ");
954         String file = input.nextLine().toUpperCase();
955
956         // Read from file and skip header line
957         try (BufferedReader br = new BufferedReader(new FileReader(file))) {
958             String line;
959             br.readLine();
960
961             // Loop until end of file and open connection
962             try (final Connection connection = DriverManager.getConnection(url)) {
963                 while ((line = br.readLine()) != null) {
964
965                     // Tokenize the whole line by , delimitors
966                     String[] attributes = line.split(",");
967
968                     // Build the transaction to add to customer table
969                     final String transaction = String.format("INSERT INTO customer (name, address) VALUES ('%s', '%s')",
970                             attributes[0], attributes[1]);
971                     if (debug) {
972                         System.out.println(transaction);
973                     }
974
975                     // Execute statement
976                     try (final Statement statement = connection.createStatement();
977                             final ResultSet resultSet = statement.executeQuery(transaction)) {
978                     }
979                 }
```



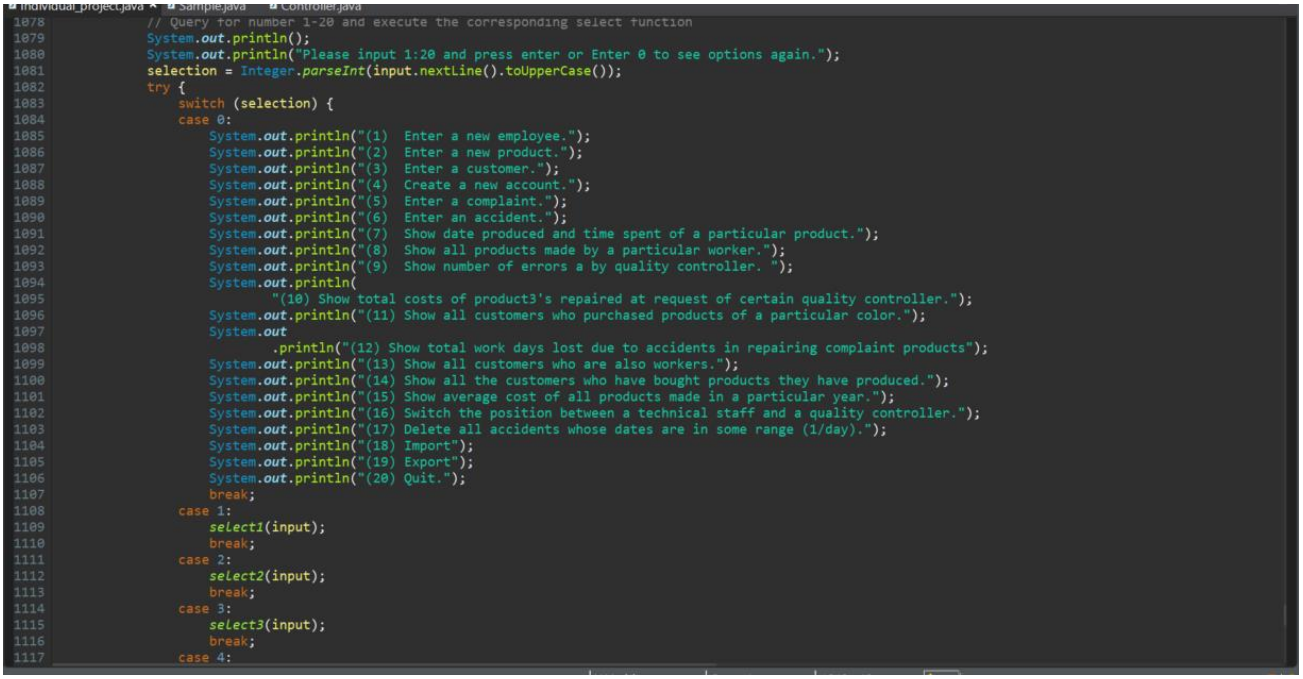
```
Individual_project.java  Sample.java  Controller.java

                try (final Statement statement = connection.createStatement();
                        final ResultSet resultSet = statement.executeQuery(transaction)) {
                }
            }
        }
    } catch (FileNotFoundException e) {
        System.err.println("File not found");
    } catch (IOException e) {
        System.err.println("IOException");
    }
}

/**
 * Selection #19: Export customers to a data file from the database.
 *
 * @param input The scanner used to read the keyboard
 * @throws SQLException The exception to catch access issues.
 */
public static void select19(Scanner input) throws SQLException, IOException {

    // Ask for import file
    System.out.println("Enter Name of Customer File to Export:");
    String file = input.nextLine().toUpperCase();

    // Open connection
    try (final Connection connection = DriverManager.getConnection(url)) {
        final String selectSql = "SELECT * FROM Customers;";
        try (final Statement statement = connection.createStatement();
                final ResultSet resultSet = statement.executeQuery(selectSql)) {

            // Write first line as header line
            String line = "Customer_Name,Customer_Address";
            BufferedWriter writer = new BufferedWriter(new FileWriter(file));
            try {
                writer.write(line);
            } catch (IOException e) {
                System.err.println("Cannot write to file");
            }

Writable   Smart Insert   1013 : 18
```



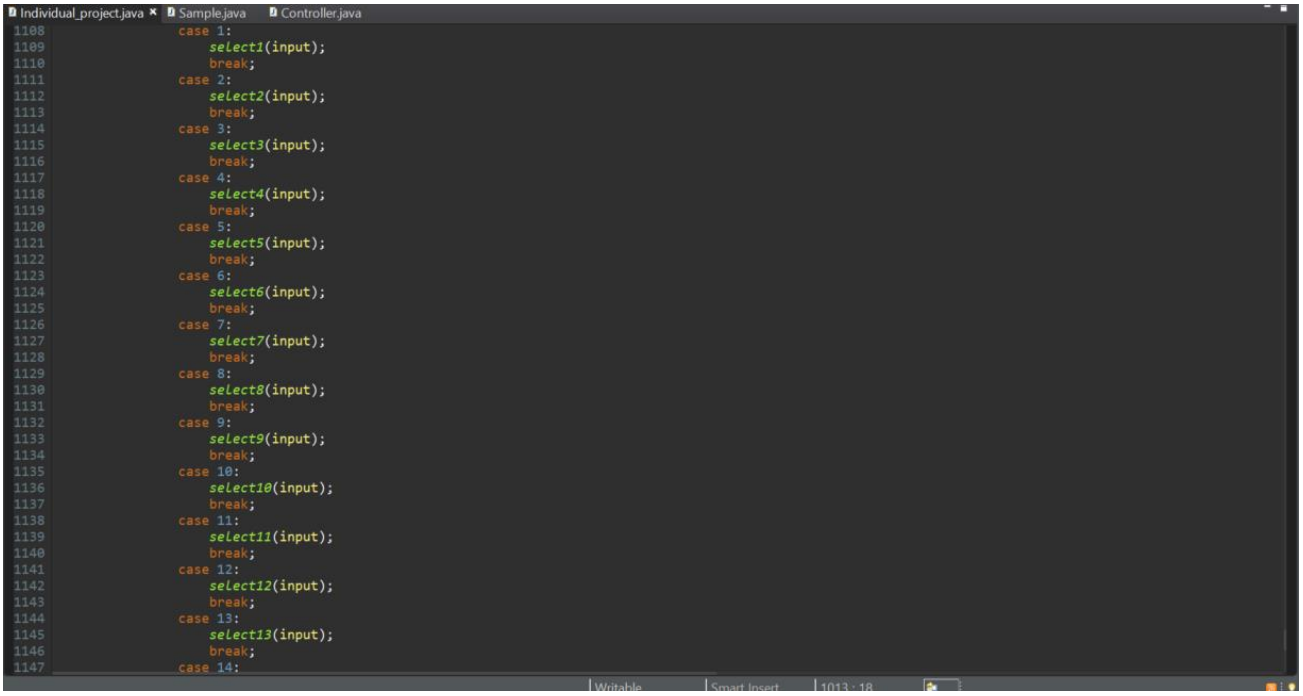
```
Individual_project.java  Sample.java  Controller.java

            } catch (IOException e) {
                System.err.println("Cannot write to file");
            }

            // Loop through all tuples
            while (resultSet.next()) {

                // Build each line as name,address
                line = resultSet.getString(1) + "," + resultSet.getString(2);

                // Write to line or show error
                try {
                    writer.write(line);
                } catch (IOException e) {
                    System.err.println("Cannot write to file");
                }
            }

            // Close the buffered writer
            writer.close();
        }
    }
}

/**
 * Project For DBMS. Program to query and manipulate the database system for
 * Future, INC. There are 20 selections overall for individuals to choose from
 * to query the database.
 *
 * @param args
 * @throws InterruptedException
 */
public static void main(String[] args) throws InterruptedException {

    System.out.println("WELCOME TO THE DATABASE SYSTEM OF FUTURE, INC.\n");
    System.out.println("(1)  Enter a new employee.");
    System.out.println("(2)  Enter a new product.");
    System.out.println("(3)  Enter a customer.");
    System.out.println("(4)  Create a new account.");

Writable   Smart Insert   1013 : 18
```



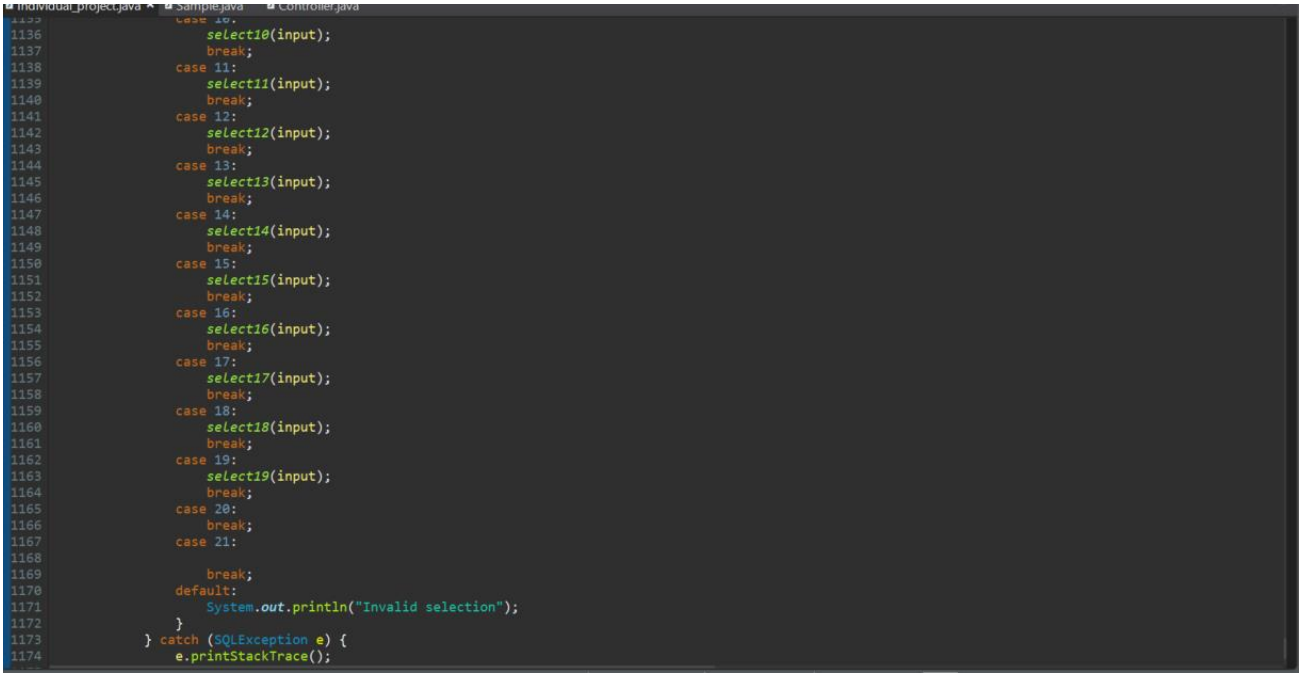
```
Individual_project.java ×  Sample.java  Controller.java
1042     */
1043     public static void main(String[] args) throws InterruptedException {
1044
1045         System.out.println("WELCOME TO THE DATABASE SYSTEM OF FUTURE, INC.\n");
1046         System.out.println("(1)  Enter a new employee.");
1047         System.out.println("(2)  Enter a new product.");
1048         System.out.println("(3)  Enter a customer.");
1049         System.out.println("(4)  Create a new account.");
1050         System.out.println("(5)  Enter a complaint.");
1051         System.out.println("(6)  Enter an accident.");
1052         System.out.println("(7)  Show date produced and time spent of a particular product.");
1053         System.out.println("(8)  Show all products made by a particular worker.");
1054         System.out.println("(9)  Show number of errors a by quality controller. ");
1055         System.out.println("(10) Show total costs of product3's repaired at request of a quality controller.");
1056         System.out.println("(11) Show all customers who purchased products of a particular color.");
1057         System.out.println("(12) Show total work days lost due to accidents in repairing complaint products");
1058         System.out.println("(13) Show all customers who are also workers.");
1059         System.out.println("(14) Show all the customers who have bought products they have produced.");
1060         System.out.println("(15) Show average cost of all products made in a particular year.");
1061         System.out.println("(16) Switch the position between a technical staff and a quality controller.");
1062         System.out.println("(17) Delete all accidents whose dates are in some range (1/day).");
1063         System.out.println("(18) Import");
1064         System.out.println("(19) Export");
1065         System.out.println("(20) Quit.");
1066
1067
1068         // Open scanner for keyboard input from user
1069         Scanner input = new Scanner(System.in);
1070         int selection = 0;
1071
1072         // Loop through user input until 20 is typed
1073         while (selection != 20) {
1074
1075             // Allow for user read data and prompt and let errors print first
1076             TimeUnit.SECONDS.sleep(1);
1077
1078             // Query for number 1-20 and execute the corresponding select function
1079             System.out.println();
1080             System.out.println("Please input 1:20 and press enter or Enter 0 to see options again.");
1081             selection = Integer.parseInt(input.nextLine().toUpperCase());
```



```
Individual_project.java ×  Sample.java  Controller.java
1078             // Query for number 1-20 and execute the corresponding select function
1079             System.out.println();
1080             System.out.println("Please input 1:20 and press enter or Enter 0 to see options again.");
1081             selection = Integer.parseInt(input.nextLine().toUpperCase());
1082             try {
1083                 switch (selection) {
1084                 case 0:
1085                     System.out.println("(1)  Enter a new employee.");
1086                     System.out.println("(2)  Enter a new product.");
1087                     System.out.println("(3)  Enter a customer.");
1088                     System.out.println("(4)  Create a new account.");
1089                     System.out.println("(5)  Enter a complaint.");
1090                     System.out.println("(6)  Enter an accident.");
1091                     System.out.println("(7)  Show date produced and time spent of a particular product.");
1092                     System.out.println("(8)  Show all products made by a particular worker.");
1093                     System.out.println("(9)  Show number of errors a by quality controller. ");
1094                     System.out.println(
1095                             "(10) Show total costs of product3's repaired at request of certain quality controller.");
1096                     System.out.println("(11) Show all customers who purchased products of a particular color.");
1097                     System.out
1098                             .println("(12) Show total work days lost due to accidents in repairing complaint products");
1099                     System.out.println("(13) Show all customers who are also workers.");
1100                     System.out.println("(14) Show all the customers who have bought products they have produced.");
1101                     System.out.println("(15) Show average cost of all products made in a particular year.");
1102                     System.out.println("(16) Switch the position between a technical staff and a quality controller.");
1103                     System.out.println("(17) Delete all accidents whose dates are in some range (1/day).");
1104                     System.out.println("(18) Import");
1105                     System.out.println("(19) Export");
1106                     System.out.println("(20) Quit.");
1107                     break;
1108                 case 1:
1109                     select1(input);
1110                     break;
1111                 case 2:
1112                     select2(input);
1113                     break;
1114                 case 3:
1115                     select3(input);
1116                     break;
1117                 case 4:
```



```
Individual_project.java × Sample.java Controller.java
1108                case 1:
1109                    select1(input);
1110                    break;
1111                case 2:
1112                    select2(input);
1113                    break;
1114                case 3:
1115                    select3(input);
1116                    break;
1117                case 4:
1118                    select4(input);
1119                    break;
1120                case 5:
1121                    select5(input);
1122                    break;
1123                case 6:
1124                    select6(input);
1125                    break;
1126                case 7:
1127                    select7(input);
1128                    break;
1129                case 8:
1130                    select8(input);
1131                    break;
1132                case 9:
1133                    select9(input);
1134                    break;
1135                case 10:
1136                    select10(input);
1137                    break;
1138                case 11:
1139                    select11(input);
1140                    break;
1141                case 12:
1142                    select12(input);
1143                    break;
1144                case 13:
1145                    select13(input);
1146                    break;
1147                case 14:
```



```
Individual_project.java × Sample.java Controller.java
1136                    select10(input);
1137                    break;
1138                case 11:
1139                    select11(input);
1140                    break;
1141                case 12:
1142                    select12(input);
1143                    break;
1144                case 13:
1145                    select13(input);
1146                    break;
1147                case 14:
1148                    select14(input);
1149                    break;
1150                case 15:
1151                    select15(input);
1152                    break;
1153                case 16:
1154                    select16(input);
1155                    break;
1156                case 17:
1157                    select17(input);
1158                    break;
1159                case 18:
1160                    select18(input);
1161                    break;
1162                case 19:
1163                    select19(input);
1164                    break;
1165                case 20:
1166                    break;
1167                case 21:
1168
1169                    break;
1170                default:
1171                    System.out.println("Invalid selection");
1172                }
1173            } catch (SQLException e) {
1174                e.printStackTrace();
```

```
Individual_project.java   Sample.java   Controller.java
                    break;
                case 14:
                    select14(input);
                    break;
                case 15:
                    select15(input);
                    break;
                case 16:
                    select16(input);
                    break;
                case 17:
                    select17(input);
                    break;
                case 18:
                    select18(input);
                    break;
                case 19:
                    select19(input);
                    break;
                case 20:
                    break;
                case 21:

                    break;
                default:
                    System.out.println("Invalid selection");
                }
            } catch (SQLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                System.err.println("Cannot read/write to file");
            }
        }

        // Close down
        System.out.println("THANK YOU FOR USING DATABASE SYSTEM OF FUTURE, INC.");
        input.close();
    }
}
```

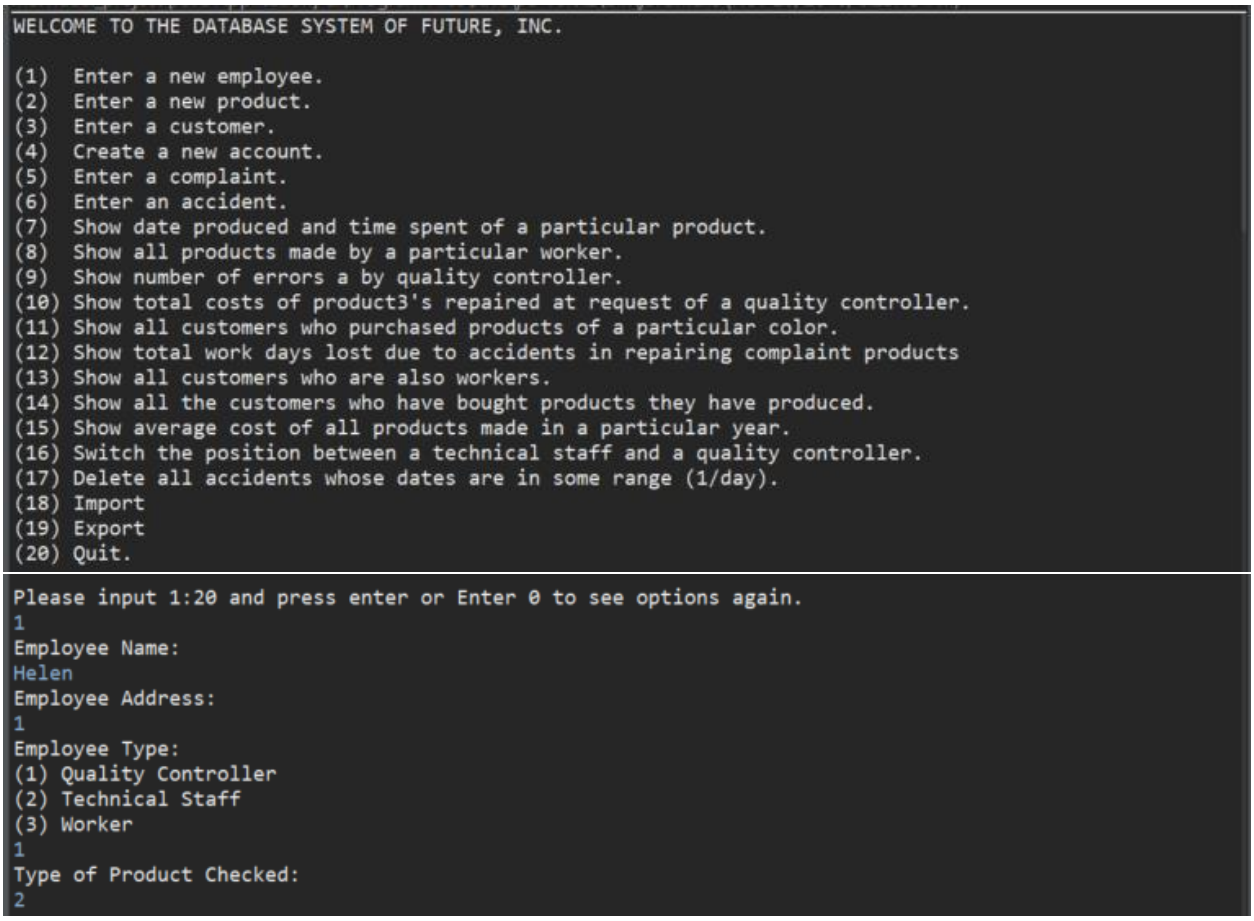
5.2. Java Program Compilation and Initial Run

```
Individual_project [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (Nov 20, 2018, 6:23:15 PM)
WELCOME TO THE DATABASE SYSTEM OF FUTURE, INC.

(1)  Enter a new employee.
(2)  Enter a new product.
(3)  Enter a customer.
(4)  Create a new account.
(5)  Enter a complaint.
(6)  Enter an accident.
(7)  Show date produced and time spent of a particular product.
(8)  Show all products made by a particular worker.
(9)  Show number of errors a by quality controller.
(10) Show total costs of product3's repaired at request of a quality controller.
(11) Show all customers who purchased products of a particular color.
(12) Show total work days lost due to accidents in repairing complaint products
(13) Show all customers who are also workers.
(14) Show all the customers who have bought products they have produced.
(15) Show average cost of all products made in a particular year.
(16) Switch the position between a technical staff and a quality controller.
(17) Delete all accidents whose dates are in some range (1/day).
(18) Import
(19) Export
(20) Quit.

Please input 1:20 and press enter or Enter 0 to see options again.
```

Task 6. Java program Execution

6.1. Screenshots showing the testing of query 1

Query 1 Output



```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Mike
Employee Address:
2
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
2
Education:
BS
Position:
Lead
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Marry
Employee Address:
3
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
2
Education:
MS
Position:
New
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Mitch
Employee Address:
4
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
3
Max Products Produced/Day:
4000
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Phil
Employee Address:
5
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
3
Max Products Produced/Day:
2000
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Leslie
Employee Address:
6
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
1
Type of Product Checked:
3
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Chase
Employee Address:
7
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
1
Type of Product Checked:
1
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Kyle
Employee Address:
8
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
2
Education:
ph.d
Position:
Something
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Cassidy
Employee Address:
9
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
3
Max Products Produced/Day:
5000
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
Eric
Employee Address:
10
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
1
Type of Product Checked:
2

Please input 1:20 and press enter or Enter 0 to see options again.
```

Updated Employee Table

	name	address
1	CASSIDY	9
2	CHASE	7
3	ERIC	10
4	HELEN	1
5	KYLE	8
6	LESLIE	6
7	MARRY	3
8	MIKE	2
9	MITCH	4
10	PHIL	5

Updated Quality Controller Table

	name	address	type_of_product...
1	CHASE	7	1
2	ERIC	10	2
3	HELEN	1	2
4	LESLIE	6	3

Updated Technical Staff Table

	name	address	education	position
1	KYLE	8	PH.D	SOMETHING
2	MARRY	3	MS	NEW
3	MIKE	2	BS	LEAD

Updated Worker Table

	name	address	max_produced_...
1	CASSIDY	9	5000
2	MITCH	4	4000
3	PHIL	5	2000

6.2. Screenshots showing the testing of query 2

Query 2 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
1
Time Spent making the Product:
1
Worker Name:
phil
Worker Address:
5
Quality Controller Name:
chase
Quality Controller Address:
7
Did the product need repaired?(yes/no)
yes
Technical Staff Name:
kyle
Technical Staff Address:
8
Date Repaired(yyyy-MM-dd):
2000-01-01
Size of Product:
small
Is the Product certified?(yes/no)
yes
Type of Product (1,2, or 3):
1
Name of Major Software:
Java
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
2
Time Spent making the Product:
1
Worker Name:
phil
Worker Address:
5
Quality Controller Name:
chase
Quality Controller Address:
7
Did the product need repaired?(yes/no)
no
Size of Product:
large
Is the Product certified?(yes/no)
no
Type of Product (1,2, or 3):
1
Name of Major Software:
C
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
3
Time Spent making the Product:
1
Worker Name:
phil
Worker Address:
5
Quality Controller Name:
eric
Quality Controller Address:
10
Did the product need repaired?(yes/no)
yes
Technical Staff Name:
kyle
Technical Staff Address:
8
Date Repaired(yyyy-MM-dd):
2000-01-01
Size of Product:
large
Is the Product certified?(yes/no)
yes
Type of Product (1,2, or 3):
2
Color:
blue
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
4
Time Spent making the Product:
1
Worker Name:
phil
Worker Address:
5
Quality Controller Name:
eric
Quality Controller Address:
10
Did the product need repaired?(yes/no)
no
Size of Product:
medium
Is the Product certified?(yes/no)
no
Type of Product (1,2, or 3):
2
Color:
yellow
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
5
Time Spent making the Product:
1
Worker Name:
phil
Worker Address:
5
Quality Controller Name:
leslie
Quality Controller Address:
6
Did the product need repaired?(yes/no)
yes
Technical Staff Name:
marry
Technical Staff Address:
3
Date Repaired(yyyy-MM-dd):
2000-01-01
Size of Product:
small
Is the Product certified?(yes/no)
yes
Type of Product (1,2, or 3):
3
Weight:
245.54
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
6
Time Spent making the Product:
1
Worker Name:
mitch
Worker Address:
4
Quality Controller Name:
leslie
Quality Controller Address:
6
Did the product need repaired?(yes/no)
no
Size of Product:
medium
Is the Product certified?(yes/no)
no
Type of Product (1,2, or 3):
3
Weight:
100
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
7
Time Spent making the Product:
1
Worker Name:
cassidy
Worker Address:
9
Quality Controller Name:
helen
Quality Controller Address:
1
Did the product need repaired?(yes/no)
yes
Technical Staff Name:
mike
Technical Staff Address:
2
Date Repaired(yyyy-MM-dd):
2000-01-01
Size of Product:
small
Is the Product certified?(yes/no)
yes
Type of Product (1,2, or 3):
2
Color:
green
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
8
Time Spent making the Product:
1
Worker Name:
cassidy
Worker Address:
9
Quality Controller Name:
chase
Quality Controller Address:
7
Did the product need repaired?(yes/no)
no
Size of Product:
large
Is the Product certified?(yes/no)
yes
Type of Product (1,2, or 3):
1
Name of Major Software:
SQL
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
9
Time Spent making the Product:
1
Worker Name:
phil
Worker Address:
5
Quality Controller Name:
leslie
Quality Controller Address:
6
Did the product need repaired?(yes/no)
no
Size of Product:
small
Is the Product certified?(yes/no)
yes
Type of Product (1,2, or 3):
3
Weight:
209
```

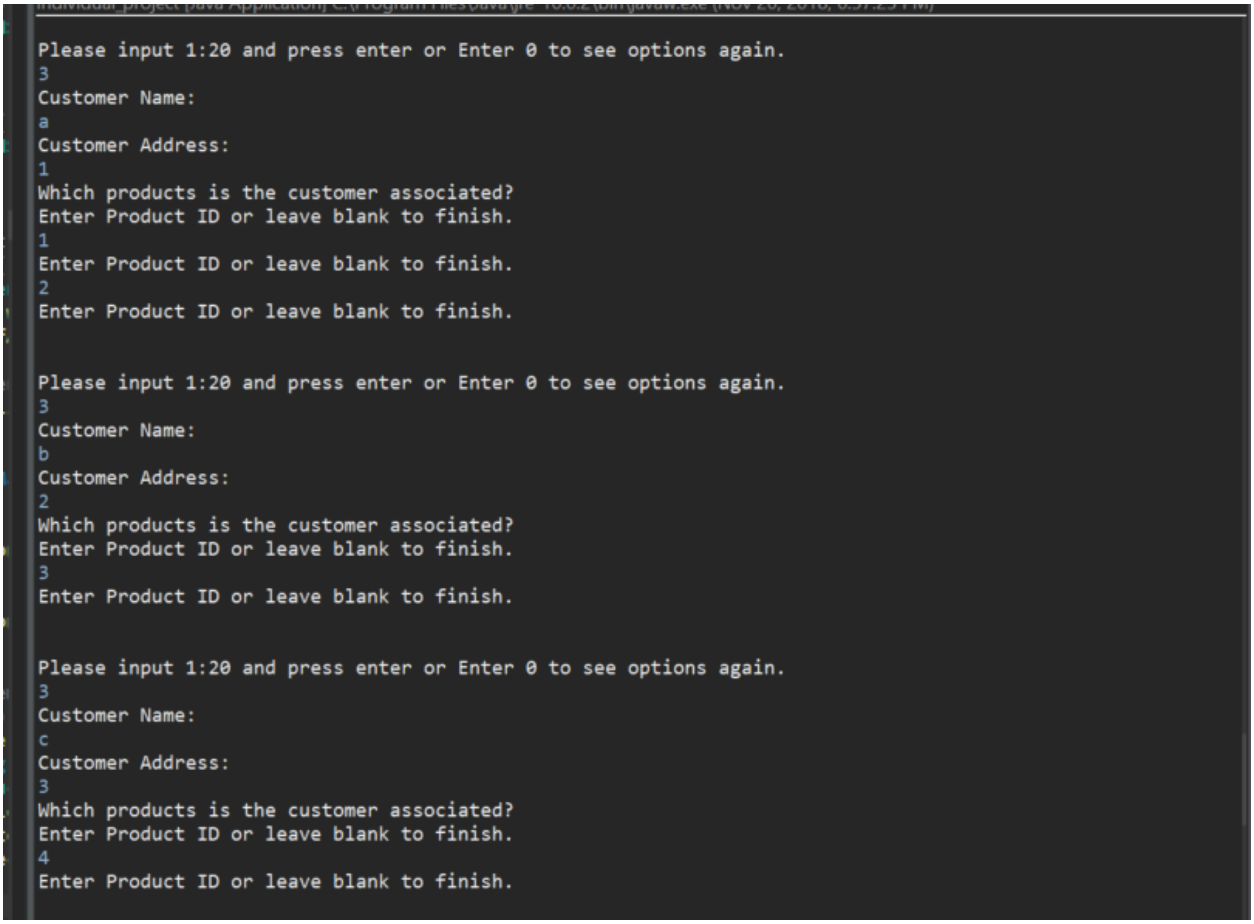
```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
10
Time Spent making the Product:
1
Worker Name:
mitch
Worker Address:
4
Quality Controller Name:
eric
Quality Controller Address:
10
Did the product need repaired?(yes/no)
no
Size of Product:
small
Is the Product certified?(yes/no)
no
Type of Product (1,2, or 3):
2
Color:
Blue

Please input 1:20 and press enter or Enter 0 to see options again.
```

Product Table

	ID	time_spent	worker_name	worker_address	quality_controll...	quality_controll...	technical_staff_...	technical_staff_a...	size	defect	certified
1	1	1	PHIL	5	CHASE	7	KYLE	8	SMALL	yes	yes
2	2	1	PHIL	5	CHASE	7			LARGE	no	no
3	3	1	PHIL	5	ERIC	10	KYLE	8	LARGE	yes	yes
4	4	1	PHIL	5	ERIC	10			MEDIUM	no	no
5	5	1	PHIL	5	LESLIE	6	MARRY	3	SMALL	yes	yes
6	6	1	MITCH	4	LESLIE	6			MEDIUM	no	no
7	7	1	CASSIDY	9	HELEN	1	MIKE	2	SMALL	yes	yes
8	8	1	CASSIDY	9	CHASE	7			LARGE	no	yes
9	9	1	PHIL	5	LESLIE	6			SMALL	no	yes
10	10	1	MITCH	4	ERIC	10			SMALL	no	no

6.3. Screenshots showing the testing of query 3

With query 3, a customer can be associated with as many products as needed, so this is a looping query with the same customer name and address, but the product ID changes. However, there are only 10 queries for products, so I am limited to only 10 loops, not 10 queries.



```
Please input 1:20 and press enter or Enter 0 to see options again.
3
Customer Name:
d
Customer Address:
4
Which products is the customer associated?
Enter Product ID or leave blank to finish.
5
Enter Product ID or leave blank to finish.
6
Enter Product ID or leave blank to finish.
7
Enter Product ID or leave blank to finish.


Please input 1:20 and press enter or Enter 0 to see options again.
3
Customer Name:
e
Customer Address:
5
Which products is the customer associated?
Enter Product ID or leave blank to finish.
8
Enter Product ID or leave blank to finish.
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
3
Customer Name:
f
Customer Address:
6
Which products is the customer associated?
Enter Product ID or leave blank to finish.
9
Enter Product ID or leave blank to finish.
10
Enter Product ID or leave blank to finish.


Please input 1:20 and press enter or Enter 0 to see options again.
```

Customer Table

	name	address
1	A	1
2	B	2
3	C	3
4	D	4
5	E	5
6	F	6

Purchase Table

	name	address	product_ID
1	A	1	1
2	A	1	2
3	B	2	3
4	C	3	4
5	D	4	5
6	D	4	6
7	D	4	7
8	E	5	8
9	F	6	9
10	F	6	10

6.4. Screenshots showing the testing of query 4

Query 4 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
1
Product ID:
1
Date Established(yyyy-MM-dd):
2000-01-01
Cost of Product:
1

Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
2
Product ID:
2
Date Established(yyyy-MM-dd):
2000-01-02
Cost of Product:
2

Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
3
Product ID:
3
Date Established(yyyy-MM-dd):
2000-01-03
Cost of Product:
3
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
4
Product ID:
4
Date Established(yyyy-MM-dd):
2000-01-04
Cost of Product:
4

Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
5
Product ID:
5
Date Established(yyyy-MM-dd):
2000-01-05
Cost of Product:
5

Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
6
Product ID:
6
Date Established(yyyy-MM-dd):
2000-01-06
Cost of Product:
6
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
7
Product ID:
7
Date Established(yyyy-MM-dd):
2000-01-07
Cost of Product:
7

Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
8
Product ID:
8
Date Established(yyyy-MM-dd):
2000-01-08
Cost of Product:
8

Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
9
Product ID:
9
Date Established(yyyy-MM-dd):
2000-01-09
Cost of Product:
9
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
4
Account Number:
10
Product ID:
10
Date Established(yyyy-MM-dd):
2000-01-10
Cost of Product:
10
```

Account Table

	account_number	product_ID	date_established	cost_to_make
1	1	1	2000-01-01	1.00
2	2	2	2000-01-02	2.00
3	3	3	2000-01-03	3.00
4	4	4	2000-01-04	4.00
5	5	5	2000-01-05	5.00
6	6	6	2000-01-06	6.00
7	7	7	2000-01-07	7.00
8	8	8	2000-01-08	8.00
9	9	9	2000-01-09	9.00
10	10	10	2000-01-10	10.00

6.5. Screenshots showing the testing of query 5

Query 5 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
5
Complaint ID:
1
Customer Name:
a
Customer Address:
1
Product ID:
1
Date (yyyy-MM-dd):
2000-01-01
Description of Issue (200 char MAX):
Doesnt work
Treatment Expected (200 char MAX):
Make it work
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
5
Complaint ID:
2
Customer Name:
b
Customer Address:
2
Product ID:
2
Date (yyyy-MM-dd):
2000-01-02
Description of Issue (200 char MAX):
Still doesnt work
Treatment Expected (200 char MAX):
Please Make it work

Please input 1:20 and press enter or Enter 0 to see options again.
5
Complaint ID:
3
Customer Name:
c
Customer Address:
3
Product ID:
3
Date (yyyy-MM-dd):
2000-01-03
Description of Issue (200 char MAX):
Still doesnt
Treatment Expected (200 char MAX):
I quit
```

Complaint Table

ID	customer_name	customer_address	product_ID	date	description	treatment_expe...
1	A	1	1	2000-01-01	DOESNT WORK	MAKE IT WORK
2	B	2	2	2000-01-02	STILL DOESN...	PLEASE MAKE...
3	C	3	3	2000-01-03	STILL DOESNT	I QUIT

6.6. Screenshots showing the testing of query 6

Query 6 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
6
Accident Number:
1
Employee Name:
phil
Employee Address:
5
Product ID:
1
Date(yyyy-MM-dd):
1999-12-30
Expected Number of Work Days Lost:
3

Please input 1:20 and press enter or Enter 0 to see options again.
6
Accident Number:
2
Employee Name:
Mike
Employee Address:
2
Product ID:
7
Date(yyyy-MM-dd):
1999-08-30
Expected Number of Work Days Lost:
23
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
6
Accident Number:
3
Employee Name:
mitch
Employee Address:
4
Product ID:
6
Date(yyyy-MM-dd):
1999-10-23
Expected Number of Work Days Lost:
43
```

Accident Table

	accident_number	employee_name	employee_addr...	product_ID	date	num_work_days...
1	1	PHIL	5	1	1999-12-30	3
2	2	MIKE	2	7	1999-08-30	23
3	3	MITCH	4	6	1999-10-23	43

6.7. Screenshots showing the testing of query 7

Query 7 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
7
Product ID:
1
Date and Time Spent of Product 1:
2000-01-01 | 1

Please input 1:20 and press enter or Enter 0 to see options again.
7
Product ID:
2
Date and Time Spent of Product 2:
2000-01-02 | 1

Please input 1:20 and press enter or Enter 0 to see options again.
7
Product ID:
3
Date and Time Spent of Product 3:
2000-01-03 | 1
```

6.8. Screenshots showing the testing of query 8

Query 8 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
8
Worker Name:
phil
Worker Address:
5
Products produced by PHIL:
1
2
3
4
5
9

Please input 1:20 and press enter or Enter 0 to see options again.
8
Worker Name:
cassidy
Worker Address:
9
Products produced by CASSIDY:
7
8
```

```
Please input 1:20 and press enter or Enter 0 to see options again.
8
Worker Name:
mitch
Worker Address:
4
Products produced by MITCH:
6
10
```

6.9. Screenshots showing the testing of query 9

Query 9 Output

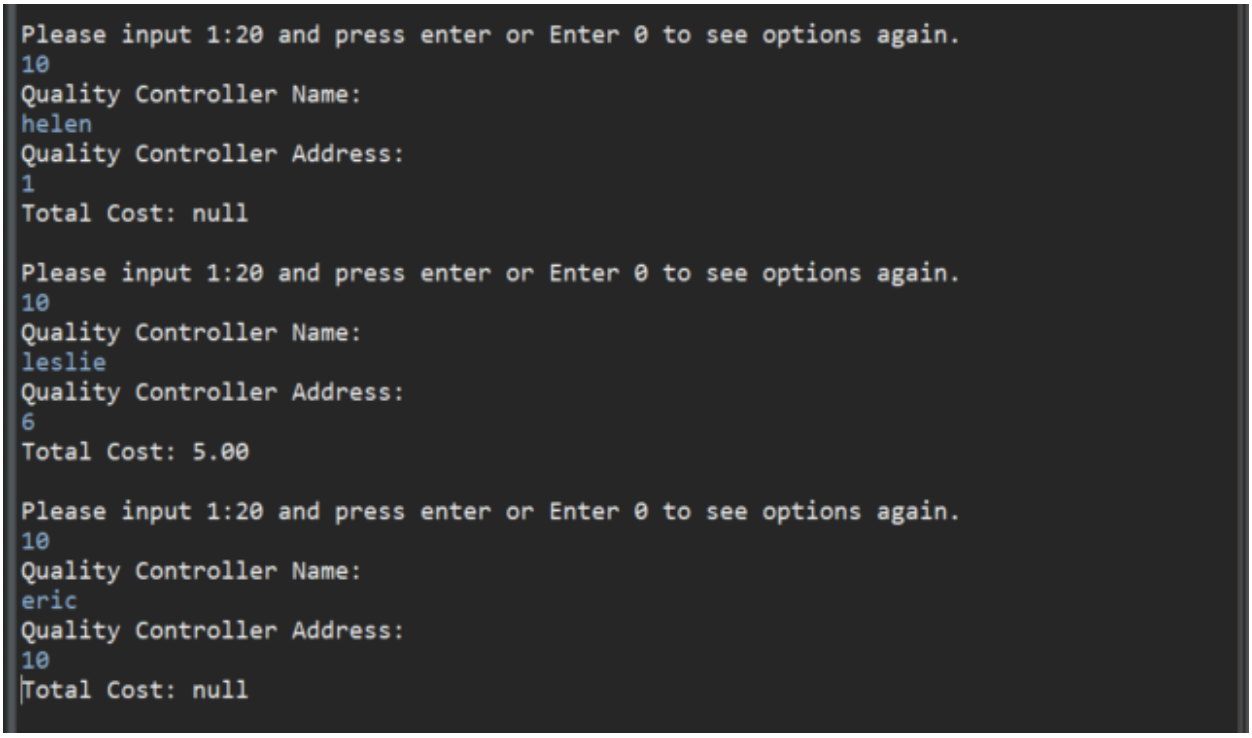
	ID	time_spent	worker_name	worker_address	quality_controll...	quality_controll...	technical_staff_...	technical_staff_a...	size	defect	certified
1	1	1	PHIL	5	CHASE	7	KYLE	8	SMALL	yes	yes
2	2	1	PHIL	5	CHASE	7			LARGE	yes	yes
3	3	1	PHIL	5	ERIC	10	KYLE	8	LARGE	yes	yes
4	4	1	PHIL	5	ERIC	10			MEDIUM	no	no
5	5	1	PHIL	5	LESLIE	6	MARRY	3	SMALL	no	yes
6	6	1	MITCH	4	LESLIE	6			MEDIUM	no	no
7	7	1	CASSIDY	9	HELEN	1	MIKE	2	SMALL	no	yes
8	8	1	CASSIDY	9	CHASE	7			LARGE	no	yes

With this information updated, we can access the correct information where product has been certified and has been shown to be defective.

```
Please input 1:20 and press enter or Enter 0 to see options again.
9
Quality Controller Name:
chase
Quality Controller Address:
7
Errors made by CHASE:
2

Please input 1:20 and press enter or Enter 0 to see options again.
9
Quality Controller Name:
eric
Quality Controller Address:
10
Errors made by ERIC:
1

Please input 1:20 and press enter or Enter 0 to see options again.
9
Quality Controller Name:
leslie
Quality Controller Address:
6
Errors made by LESLIE:
0
```

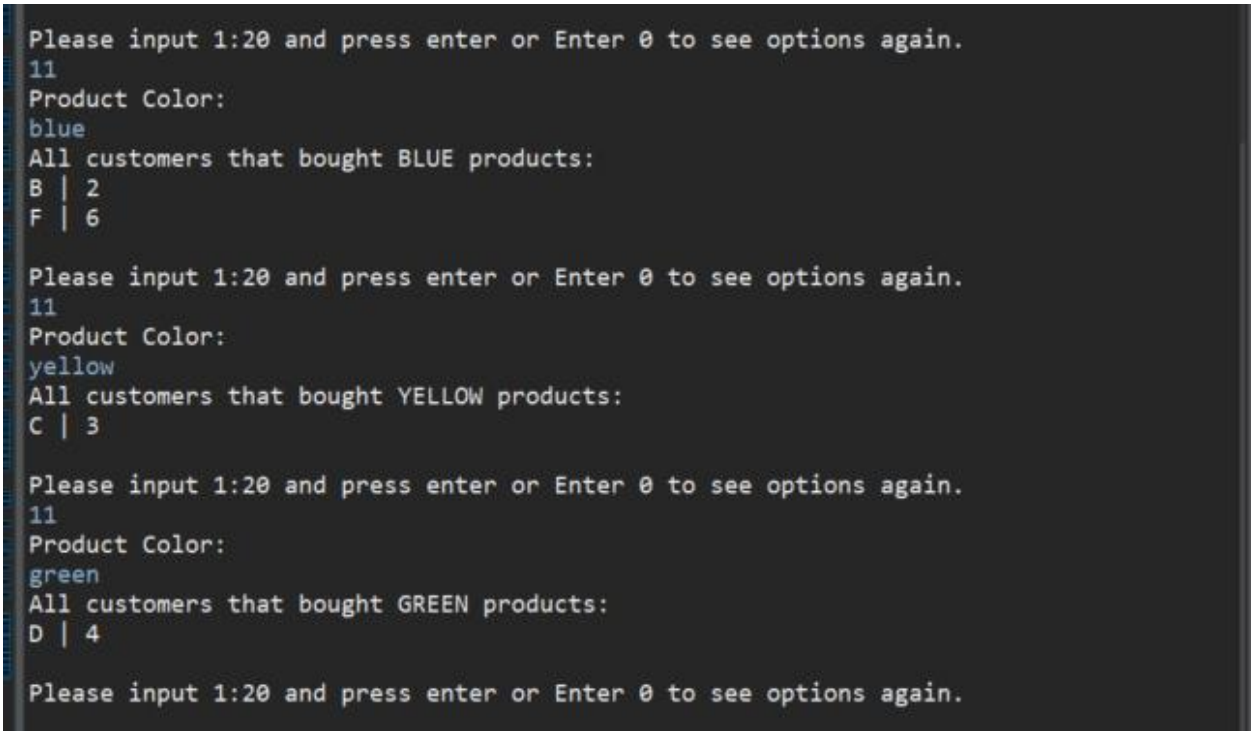
6.10. Screenshots showing the testing of query 10

Query 10 Output



6.11. Screenshots showing the testing of query 11

Query 11 Output



6.12. Screenshots showing the testing of query 12

Query 12 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
12
Total Work Days Lost: 3

Please input 1:20 and press enter or Enter 0 to see options again.
```

6.13. Screenshots showing the testing of query 13

Adding a new customer with the name of a worker will show output, query 2 and 3 are ran again.

Product Table

	ID	time_spent	worker_name	worker_address	quality_controll...	quality_controll...	technical_staff_...	technical_staff_a...	size	defect	certified
1	1	1	PHIL	5	CHASE	7	KYLE	8	SMALL	yes	yes
2	2	1	PHIL	5	CHASE	7			LARGE	yes	yes
3	3	1	PHIL	5	ERIC	10	KYLE	8	LARGE	yes	yes
4	4	1	PHIL	5	ERIC	10			MEDIUM	no	no
5	5	1	PHIL	5	LESLIE	6	MARRY	3	SMALL	no	yes
6	6	1	MITCH	4	LESLIE	6			MEDIUM	no	no
7	7	1	CASSIDY	9	HELEN	1	MIKE	2	SMALL	no	yes
8	8	1	CASSIDY	9	CHASE	7			LARGE	no	yes
9	9	1	PHIL	5	LESLIE	6			SMALL	no	yes
10	10	1	MITCH	4	ERIC	10			SMALL	no	no
11	100	12	PHIL	5	CHASE	7			MEDIUM	no	yes

Customer Table

	name	address
1	A	1
2	B	2
3	C	3
4	D	4
5	E	5
6	F	6
7	PHIL	5

Purchase Table

	name	address	product_ID
1	A	1	1
2	A	1	2
3	B	2	3
4	C	3	4
5	D	4	5
6	D	4	6
7	D	4	7
8	E	5	8
9	F	6	9
10	F	6	10
11	PHIL	5	100

Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
13
All customers that are also workers:
PHIL | 5
```

6.14. Screenshots showing the testing of query 14

Query 14 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
14
All customers that purchaced produces made, certified, or repaired by themselves:
PHIL | 5
```

6.15. Screenshots showing the testing of query 15

Query 15 Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
15
Year:
2000
Average Cost of products made in 2000: 5.500000
```

6.16. Screenshots showing the testing of query 16

Query 16 Output

	name	address	type_of_product...
1	ERIC	10	2
2	HELEN	1	2
3	LESLIE	6	3

	name	address	education	position
1	KYLE	8	PH.D	SOMETHING
2	MARRY	3	MS	NEW
3	MIKE	2	BS	LEAD

```
Please input 1:20 and press enter or Enter 0 to see options again.
16
Former Quality Controller Name:
chase
Former Quality Controller Address:
7
Former Quality Controller Education:
bs
Former Quality Controller Position:
Database Builder
Former Technical Staff Name:
Kyle
Former Technical Staff Address:
8
Type of Product Checked:
1

Please input 1:20 and press enter or Enter 0 to see options again.
```

	name	address	type_of_product...
1	ERIC	10	2
2	HELEN	1	2
3	KYLE	8	1
4	LESLIE	6	3

	name	address	education	position
1	CHASE	7	BS	DATABASE BU...
2	MARRY	3	MS	NEW
3	MIKE	2	BS	LEAD

6.17. Screenshots showing the testing of query 17

Query 17 Output

	accident_number	employee_name	employee_addr...	product_ID	date	num_work_days...
1	1	PHIL	5	1	1999-12-30	3
2	2	MIKE	2	7	1999-08-30	23
3	3	MITCH	4	6	1999-10-23	43

```
Please input 1:20 and press enter or Enter 0 to see options again.
17
Start Date(yyyy-MM-dd):
1999-06-30
End Date(yyyy-MM-dd):
1999-09-01

Please input 1:20 and press enter or Enter 0 to see options again.
```

	accident_number	employee_name	employee_addr...	product_ID	date	num_work_days...
1	1	PHIL	5	1	1999-12-30	3
2	3	MITCH	4	6	1999-10-23	43

6.18. Screenshots showing the testing of query 18 and 19

Import Output

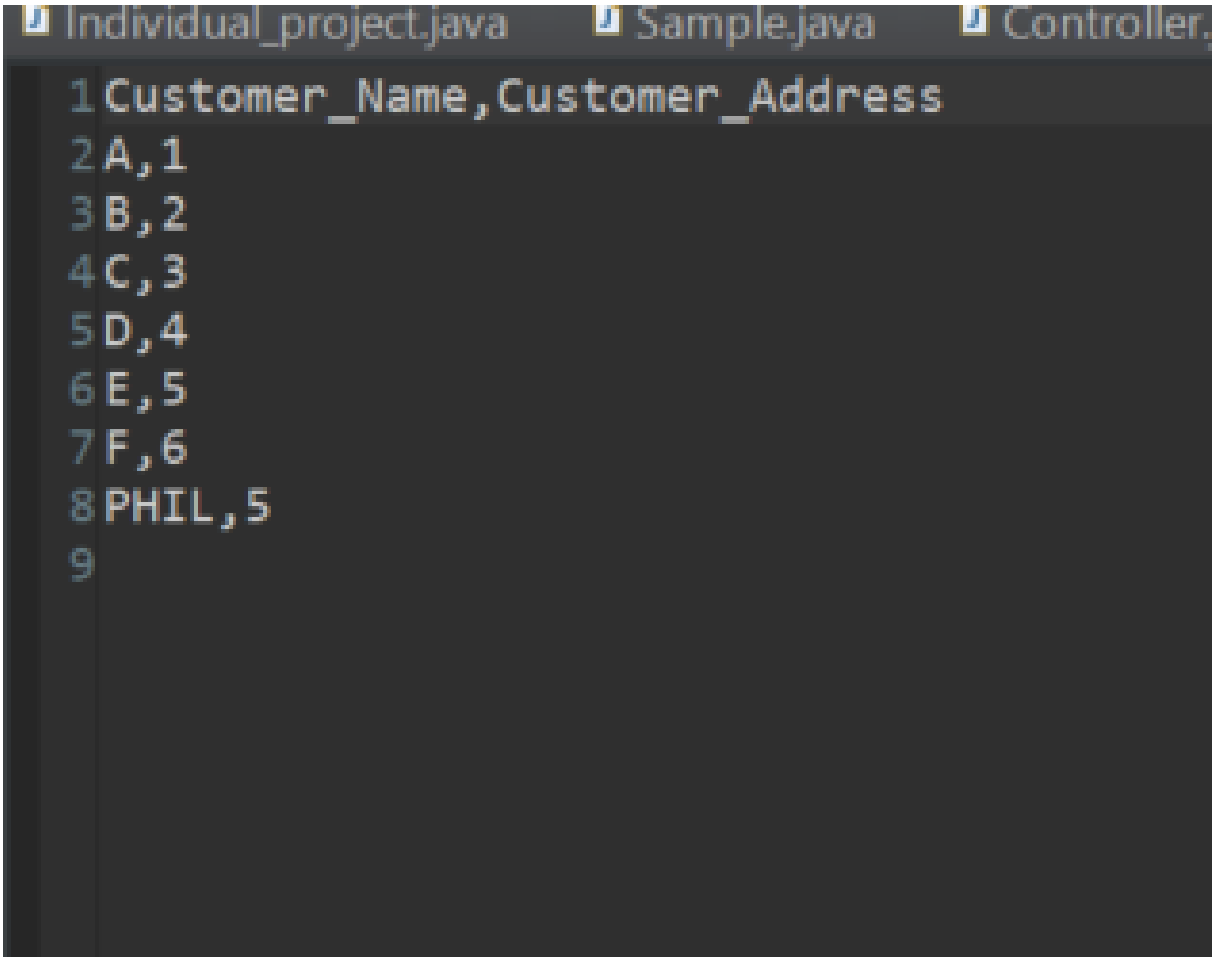
	name	address
1	A	1
2	B	2
3	C	3
4	D	4
5	E	5
6	F	6
7	PHIL	5

```
Please input 1:20 and press enter or Enter 0 to see options again.
18
Enter the name of Customer File as Import:
customers.txt

Please input 1:20 and press enter or Enter 0 to see options again.
```

	name	address
1	A	1
2	B	2
3	C	3
4	D	4
5	E	5
6	F	6
7	Frank	4
8	Hal	20
9	Leo	6
10	Pat	3
11	PHIL	5

Export Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
19
Enter Name of Customer File to Export:
customers.txt

Please input 1:20 and press enter or Enter 0 to see options again.
```



6.19. Screenshots showing the testing of query Errors

Query 1 Error Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
18
Enter the name of Customer File as Import:
customers.txt

Please input 1:20 and press enter or Enter 0 to see options again.
1
Employee Name:
grayson
Employee Address:
20
Employee Type:
(1) Quality Controller
(2) Technical Staff
(3) Worker
2
Education:
not bs, ms, or ph.d
Position:
error
Invalid education or type of product checked

Please input 1:20 and press enter or Enter 0 to see options again.
```

Query 2 Error Output

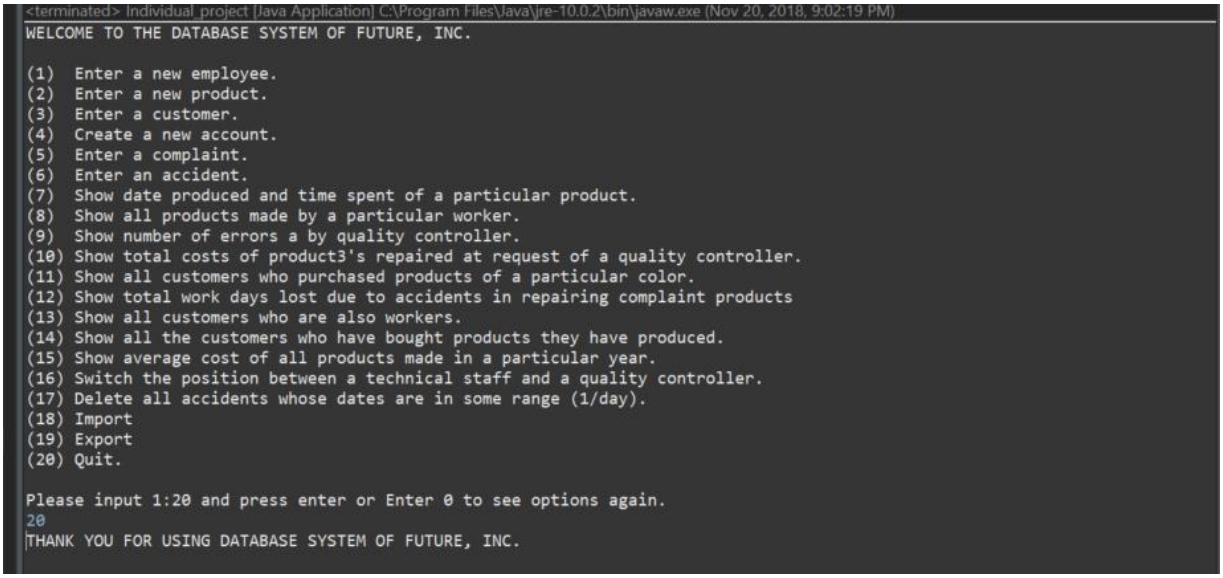
```
Please input 1:20 and press enter or Enter 0 to see options again.
2
Product ID:
1
Time Spent making the Product:
1
Worker Name:
chase
Worker Address:
7
Quality Controller Name:
kyle
Quality Controller Address:
1
Did the product need repaired?(yes/no)
no
Size of Product:
smal
Is the Product certified?(yes/no)
yes
Type of Product (1,2, or 3):
2
Color:
blue
com.microsoft.sqlserver.jdbc.SQLServerException: Violation of PRIMARY KEY constraint 'PK__product__
        at com.microsoft.sqlserver.jdbc.SQLServerException.makeFromDatabaseError(SQLServerExceptior
        at com.microsoft.sqlserver.jdbc.TDSTokenHandler.onEOF(tdsparser.java:258)
        at com.microsoft.sqlserver.jdbc.TDSParser.parse(tdsparser.java:104)
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.getNextResult(SQLServerStatement.java:16
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.doExecuteStatement(SQLServerStatement.ja
        at com.microsoft.sqlserver.jdbc.SQLServerStatement$StmtExecCmd.doExecute(SQLServerStatement
        at com.microsoft.sqlserver.jdbc.TDSCommand.execute(IOBuffer.java:7240)
        at com.microsoft.sqlserver.jdbc.SQLServerConnection.executeCommand(SQLServerConnection.java
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.executeCommand(SQLServerStatement.java:2
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.executeStatement(SQLServerStatement.java
```

Query 5 Error Output

```
Please input 1:20 and press enter or Enter 0 to see options again.
5
Complaint ID:
20
Customer Name:
bill
Customer Address:
oak
Product ID:
1
Date (yyyy-MM-dd):
2000-02-20
Description of Issue (200 char MAX):
error
Treatment Expected (200 char MAX):
error
com.microsoft.sqlserver.jdbc.SQLServerException: The INSERT statement conflicted with the FOREIGN K
        at com.microsoft.sqlserver.jdbc.SQLServerException.makeFromDatabaseError(SQLServerExceptior
        at com.microsoft.sqlserver.jdbc.TDSTokenHandler.onEOF(tdsparser.java:258)
        at com.microsoft.sqlserver.jdbc.TDSParser.parse(tdsparser.java:104)
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.getNextResult(SQLServerStatement.java:16
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.doExecuteStatement(SQLServerStatement.ja
        at com.microsoft.sqlserver.jdbc.SQLServerStatement$StmtExecCmd.doExecute(SQLServerStatement
        at com.microsoft.sqlserver.jdbc.TDSCommand.execute(IOBuffer.java:7240)
        at com.microsoft.sqlserver.jdbc.SQLServerConnection.executeCommand(SQLServerConnection.java
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.executeCommand(SQLServerStatement.java:2
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.executeStatement(SQLServerStatement.java
        at com.microsoft.sqlserver.jdbc.SQLServerStatement.execute(SQLServerStatement.java:739)
        at Individual_project.select5(Individual_project.java:471)
        at Individual_project.main(Individual_project.java:1121)

Please input 1:20 and press enter or Enter 0 to see options again.
```

6.20. Screenshots showing the testing of query 20

Output



This concludes the Individual Project for DBMS CS/DSA 4513.