

CSC 321 Database Project Written Report

Alice Li: Data Generation, Table Population, User Interface Insert, Write Report

Jiayi Zhou: ER Diagram, Data Generation, User Interface Report, Write Report

Problem Statement:

Our project is based on project option 2. The dealership, Friendly Cars, has a database system that tracks car and sales information. The existing database stores detail about cars, customers, salespeople, and car sales. However, the owner Jim Friendly wants to create a more efficient database system that can provide additional information. He wants the new one to generate specific reports and be able to add data to the database. The project needs a database to add data better and create reports. A user interface is needed for the project to let users directly use it without knowing the underlying implementation structure.

So we build a database and user interface. The car shop employee will be the primary user of the database and user interface, and they can connect to the database through MySQL or the user interface. The user interface is run with a Python command; then, employees can view and insert data if needed. We implemented the user interface with PyQt5 and MySQL connector. We choose PyQt5 because it offers a variety of features to choose from and can build large-scale GUI-based programs. If the users want to expand based on the current program, they can easily pick up from what we have and find abundant resources to learn to build programs.

Based on the current functions of the user interface, by running the Report.py file and clicking on the buttons in the pop-up user interface, users can view reports of employees, available cars in the system, customers who have purchased cars, customers who have not made a purchase, and sales reports.

Furthermore, in the user interface for inserting data, the user can add new data for cars, customers, and employees. The user can enter information about a car received from the manufacturer, including the car id, number of doors, weight capacity, color, list price, model, date of manufacture, date of delivery, etc. The user can also enter customer information, including the customer ID, their name, and whether they bought a car. The user can also enter employee information if a new employee comes, including the employee ID, their name, and their role.

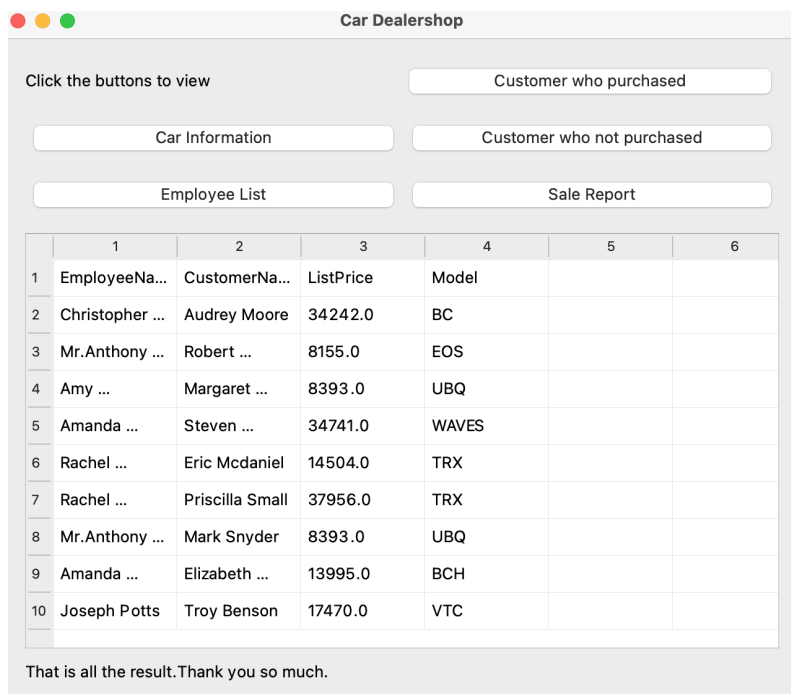
A potential next step is providing a view for customers to use the system to see. A login system to identify valid users and types of users would be another great function to add.

Description of user interfaces:

Our User Interface contains two files: report.py and insert.py, each representing either to get some reports from our database or to insert more data information into our database.

- Report User Interface:

This interface can be used with the terminal call. Users can call the Python file with python Report.py after installing the necessary libraries (pymysql, mysql-connector-python, PyQt5), and a window will pop up. Users can click to choose different options. Some options include generating sales reports and showing the list of customers who purchased. The result will be printed in the below window with headings in the first row.



	1	2	3	4	5	6
1	EmployeeNa...	CustomerNa...	ListPrice	Model		
2	Christopher ...	Audrey Moore	34242.0	BC		
3	Mr.Anthony ...	Robert ...	8155.0	EOS		
4	Amy ...	Margaret ...	8393.0	UBQ		
5	Amanda ...	Steven ...	34741.0	WAVES		
6	Rachel ...	Eric Mcdaniel	14504.0	TRX		
7	Rachel ...	Priscilla Small	37956.0	TRX		
8	Mr.Anthony ...	Mark Snyder	8393.0	UBQ		
9	Amanda ...	Elizabeth ...	13995.0	BCH		
10	Joseph Potts	Troy Benson	17470.0	VTC		

That is all the result.Thank you so much.

- Insert User Interface:

The Insert User Interface can be used to insert data, which will be directly inserted into the database. We can now run the file from the terminal using the python3 command, as shown in the graph below.

The user can first choose the kind of insert they want to add, either customer data, employee data, or car data.

Then, the interface would appear to determine what information could be inserted into the given table.

After finishing entering the data in the fields, we can click “Add Data” and see what the new data entered would ultimately be. After adding the data, the user interface show “successfully added data” or “fail to added”.

The image displays three screenshots of a PyQT application interface for adding data to a database. The first screenshot shows a terminal window with the command `python3 insert3.py` and a terminal output showing the last login and the command execution. The second screenshot shows the 'Add Data' dialog box with the 'Insert into table:' dropdown menu open, showing options: Customer, Employee, and Car. The third screenshot shows the 'Add Data' dialog box with the 'Employee' option selected in the dropdown menu. The dialog box contains several input fields for data entry, including 'Employee ID', 'Employee Name', and 'Employee Role'. The 'Add Data' button is visible at the bottom of the dialog box.

- Source code:

https://www.tutorialspoint.com/pyqt/pyqt_quick_guide.htm

<https://www.guru99.com/pyqt-tutorial.html>

<https://realpython.com/python-pyqt-database/>

<https://codeloop.org/connect-pyqt5-with-mysql-database/>

https://codeloop.org/pyqt5-tutorial-how-to-insert-data-in-mysql-database/?ref=moriorh.com&utm_source=moriorh.com
<https://codeloop.org/pyqt5-tutorial-retrieve-data-from-mysql-in-qtablewidget/>

ER Design for the database

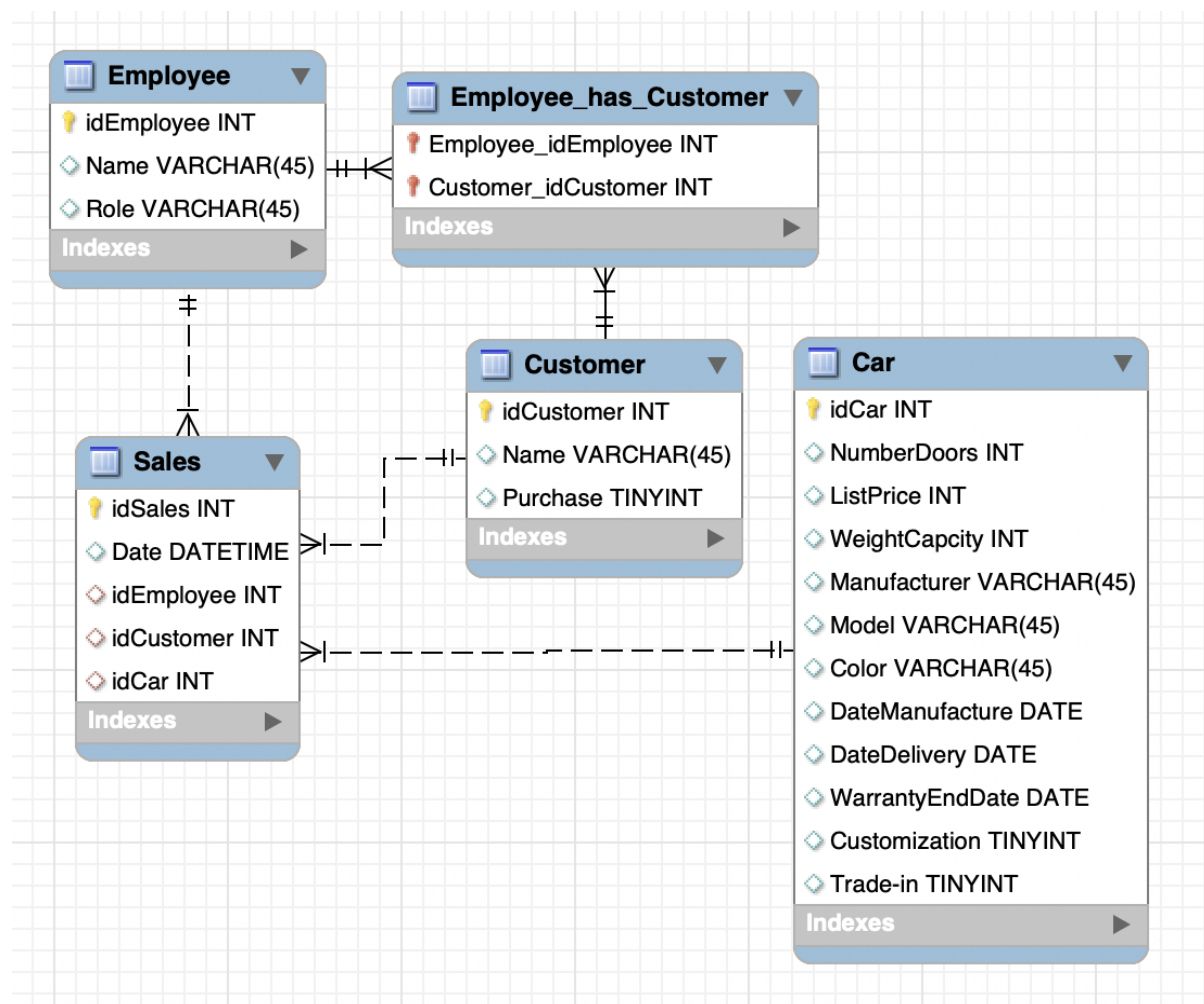
We want our ER diagram to be as straightforward as possible. The ER diagram demonstrates the main data entities and their relationships. We have tables for:

Employee (idEmployee, Name, Role)

Customer (idCustomer, Name, Purchase)

Car (idCar, NumberDoors, ListPrice, WeightCapacity, Manufacturer, Model, Color, DateManufacture, DateDelivery, WarrantyEndDate, Customization, Trade-in)

Sales (idSales, Date, idEmployee, idCustomer, idCar)



Database schema & creating tables

After creating the ER diagram, we used forward engineering to create the tables. Here are the scripts:

```
-- MySQL Workbench Forward Engineering
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-----
-- Schema 321ProjectCars
-----
-- Schema 321ProjectCars
-----
CREATE SCHEMA IF NOT EXISTS `321ProjectCars` DEFAULT CHARACTER SET utf8 ;
USE `321ProjectCars` ;

-----
-- Table `321ProjectCars`.`Employee`
-----
CREATE TABLE IF NOT EXISTS `321ProjectCars`.`Employee` (
  `idEmployee` INT NOT NULL,
  `Name` VARCHAR(45) NULL,
  `Role` VARCHAR(45) NULL,
  PRIMARY KEY (`idEmployee`))
ENGINE = InnoDB;

-----
-- Table `321ProjectCars`.`Customer`
-----
CREATE TABLE IF NOT EXISTS `321ProjectCars`.`Customer` (
  `idCustomer` INT NOT NULL,
  `Name` VARCHAR(45) NULL,
  `Purchase` TINYINT NULL,
  PRIMARY KEY (`idCustomer`))
ENGINE = InnoDB;

-----
-- Table `321ProjectCars`.`Car`
-----
CREATE TABLE IF NOT EXISTS `321ProjectCars`.`Car` (
  `idCar` INT NOT NULL,
  `NumberDoors` INT NULL,
  `ListPrice` DOUBLE NULL,
  `WeightCapacity` DOUBLE NULL,
  `Manufacturer` VARCHAR(45) NULL,
  `Model` VARCHAR(45) NULL,
  `Color` VARCHAR(45) NULL,
  `DateManufacture` DATETIME NULL,
  `DateDelivery` DATETIME NULL,
  `WarrantyEndDate` DATETIME NULL,
  `Customization` TINYINT NULL,
  `Trade-in` TINYINT NULL,
  PRIMARY KEY (`idCar`))
ENGINE = InnoDB;

-----
-- Table `321ProjectCars`.`Sales`
-----
CREATE TABLE IF NOT EXISTS `321ProjectCars`.`Sales` (
  `idSales` INT NOT NULL,
  `Date` DATETIME NULL,
  `idEmployee` INT NULL,
  `idCustomer` INT NULL,
  `idCar` INT NULL,
  PRIMARY KEY (`idSales`),
  INDEX `idEmployee_idx` (`idEmployee` ASC) VISIBLE,
  INDEX `idCustomer_idx` (`idCustomer` ASC) VISIBLE,
  INDEX `idCar_idx` (`idCar` ASC) VISIBLE,
  CONSTRAINT `idEmployee`
    FOREIGN KEY (`idEmployee`)
      REFERENCES `321ProjectCars`.`Employee` (`idEmployee`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `idCustomer`
    FOREIGN KEY (`idCustomer`)
      REFERENCES `321ProjectCars`.`Customer` (`idCustomer`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `idCar`
    FOREIGN KEY (`idCar`)
      REFERENCES `321ProjectCars`.`Car` (`idCar`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `321ProjectCars`.`Employee_has_Customer`
-----
CREATE TABLE IF NOT EXISTS `321ProjectCars`.`Employee_has_Customer` (
  `Employee_idEmployee` INT NOT NULL,
  `Customer_idCustomer` INT NOT NULL,
  PRIMARY KEY (`Employee_idEmployee`, `Customer_idCustomer`),
  INDEX `fk_Employee_has_Customer_Customer1_idx` (`Customer_idCustomer` ASC) VISIBLE,
  INDEX `fk_Employee_has_Customer_Employee1_idx` (`Employee_idEmployee` ASC) VISIBLE,
  CONSTRAINT `fk_Employee_has_Customer_Employee1`
    FOREIGN KEY (`Employee_idEmployee`)
      REFERENCES `321ProjectCars`.`Employee` (`idEmployee`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_Employee_has_Customer_Customer1`
    FOREIGN KEY (`Customer_idCustomer`)
      REFERENCES `321ProjectCars`.`Customer` (`idCustomer`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Normalization Review – are your tables in 3NF or BCNF? How do you know?

All tables are 3NF and BCNF. Customer is 3NF and BCNF, as idCustomer is the only primary key. As Name and Purchase are independent, the relationship satisfies the 3NF. As Customer is a 3NF and idCustomer is a candidate key, it satisfies the BCNF. Sales is 3NF and BCNF: The primary key of Sales is idSales. idCar, idCustomer, and idEmployee are foreign and primary keys of other tables. There is no transitive relationship. Thus sales is 3NF. As idSales is a candidate key, it satisfies the BCNF. Car is 3NF and BCNF: The primary key idCar. As other columns are independent and have no transitive relationship, thus Car is 3NF. As idCar is a candidate key, it satisfies the BCNF. The Employee is 3NF and BCNF: idEmployee is the primary and candidate keys. There are no other dependencies. Thus it is 3NF and BCNF.

How data was obtained & populated tables

*Note: As many names and IDs are generated randomly, the result shown here may differ from what we have in the database. It is just showing the commands and example results.

- Employee Table:

We randomly generate data for eight salespeople for the employee table and then add special data items for the office manager and owner.

```
def generate_employee_dataframe(int_cols, string_name_cols, rows):
    # Check that input is valid
    if not all(isinstance(i, int) and i > 0 for i in [int_cols, string_name_cols, rows]):
        raise ValueError("All inputs must be integers greater than 0.")

    # Create empty dataframe
    df = pd.DataFrame()

    # Create instance of Faker
    fake = Faker()

    # Add integer columns
    for i in range(int_cols):
        df['int_col_{}'.format(i)] = [random.randint(0, 100) for _ in range(rows)]

    # Add name string columns
    for i in range(string_name_cols):
        df['string_col_{}'.format(i)] = [fake.name() for _ in range(rows)]

    return df
```

```
employee = generate_employee_dataframe(1, 1, 8)
employee.columns = ["idEmployee", "Name"]
employee = employee.drop_duplicates(subset='idEmployee', keep="first")
employee
```

	idEmployee	Name
0	85	Kathryn Evans
1	73	Natalie Hoffman
2	3	Richard Molina
3	66	Daniel Gallegos
4	36	Scott Patton
5	80	Anthony Jefferson
6	16	Linda Jones

```

employee['Role'] = pd.Series(['salesperson' for x in range(len(employee.index))])
employee.loc[len(employee.index)] = [11, 'John Smith', 'office manager']
employee.loc[len(employee.index)] = [1, 'Jim Friendly', 'owner']
employee

```

	idEmployee	Name	Role
0	5	Regina Bailey	salesperson
1	50	Mike Vega	salesperson
2	56	Lisa Vargas	salesperson
3	52	Samantha Davidson	salesperson
4	29	Sherry Perez	salesperson
5	73	Teresa Kelley	salesperson
6	41	John Miller	salesperson
7	72	Kaitlyn Fisher	salesperson
8	11	John Smith	office manager
9	1	Jim Friendly	owner

- Customer table:

For the Customer table, we randomly generate the Customer names, IDs, and indications of whether or not they bought the car.

```

def generate_customer_dataframe(int_cols, string_name_cols, bool_cols, rows):
    # Check that input is valid
    if not all(isinstance(i, int) and i > 0 for i in [int_cols, string_name_cols, bool_cols, rows]):
        raise ValueError("All inputs must be integers greater than 0.")

    # Create empty dataframe
    df = pd.DataFrame()

    # Create instance of Faker
    fake = Faker()

    # Add integer columns
    for i in range(int_cols):
        df['int_col_{}'.format(i)] = [random.randint(0, 100) for _ in range(rows)]

    # Add name string columns
    for i in range(string_name_cols):
        df['string_col_{}'.format(i)] = [fake.name() for _ in range(rows)]

    # Add boolean columns
    for i in range(bool_cols):
        df['bool_col_{}'.format(i)] = [random.choice([1, 0]) for _ in range(rows)]

    return df

```

```

customer = generate_customer_dataframe(1, 1, 1, 20)
customer.columns = ["idCustomer", "Name", "Purchase"]
customer = customer.drop_duplicates(subset='idCustomer', keep="first")
customer

```

	idCustomer	Name	Purchase
0	100	John Smith	0
1	38	Cindy Green	0
2	85	Candace Gardner	1
3	53	Kevin Brewer	1

- Car table:

For the Car table, we randomly generate many int, float, string, date time, and boolean data points for idCar, NumberDoors, ListPrice, WeightCapacity, Manufacturer, Model, Color, DateManufacture, DateDelivery, WarrantyEndDate, Customization, and Trade-in, shown in the first graph below. And then, we make adjustments to make our data make more sense. For example, we limit the number of doors of the car instead of letting it be a random integer; we limit the car's color instead of letting it be an arbitrary string. The commands are shown in the second graph below.

```
def generate_car_dataframe(int_cols, float_cols, string_cols, datetime_cols, bool_cols, rows):
    # Check that input is valid
    if not all(isinstance(i, int) and i > 0 for i in [int_cols, float_cols, string_cols, datetime_cols, bool_cols, rows]):
        raise ValueError("All inputs must be integers greater than 0.")
    # Create empty dataframe
    df = pd.DataFrame()
    # Create instance of Faker
    fake = Faker()
    # Add integer columns
    for i in range(int_cols):
        df['int_col_{}'.format(i)] = [random.randint(0, 100) for _ in range(rows)]
    # Add float columns
    for i in range(float_cols):
        df['float_col_{}'.format(i)] = [random.uniform(0, 100) for _ in range(rows)]
    # Add string columns
    for i in range(string_cols):
        df['string_col_{}'.format(i)] = [fake.word() for _ in range(rows)]
    # Add datetime columns
    for i in range(datetime_cols):
        df['datetime_col_{}'.format(i)] = [fake.date_time() for _ in range(rows)]
    # Add boolean columns
    for i in range(bool_cols):
        df['bool_col_{}'.format(i)] = [random.choice([1, 0]) for _ in range(rows)]
    return df
```

```
car = generate_car_dataframe(2,2,3,3,2,30)
car.columns = ["idCar", "NumberDoors", "ListPrice", "WeightCapacity", "Manufacturer", "Model", "Color", "DateManufacture", "DateDelivery", "WarrantyEndDate", "Customization", "Trade-in"]
car = car.drop_duplicates(subset='idCar', keep="first")
car
```

	idCar	NumberDoors	ListPrice	WeightCapacity	Manufacturer	Model	Color	DateManufacture	DateDelivery	WarrantyEndDate	Customization	Trade-in
0	90	88	25.615026	41.963403	side	whole	world	1975-12-24 22:14:31	2008-04-10 00:04:36	1999-12-11 02:22:38		0
1	56	61	87.733448	41.110898	reveal	interesting	big	1977-04-13 03:56:50	1970-08-13 19:56:45	1971-05-12 13:26:05		0
2	17	46	44.449021	35.972753	nation	morning	tax	2017-04-12 10:39:23	1971-11-03 12:58:45	2001-08-19 05:41:55		0

```
fake = Faker()
car.columns = ["idCar", "NumberDoors", "ListPrice", "WeightCapacity", "Manufacturer", "Model", "Color", "DateManufacture", "DateDelivery", "WarrantyEndDate", "Customization", "Trade-in"]
car['idCar'] = pd.Series([random.randint(100, 1000) for x in range(len(car.index)+1)])
car['NumberDoors'] = pd.Series([random.randint(2, 6) for x in range(len(car.index)+1)])
car['ListPrice'] = pd.Series([random.randint(8000, 50000) for x in range(len(car.index)+1)])
car['WeightCapacity'] = pd.Series([random.randint(400, 2000) for x in range(len(car.index)+1)])
car['Model'] = pd.Series([fake.cryptocurrency_code() for x in range(len(car.index)+1)])
car['Color'] = pd.Series([fake.color_name() for x in range(len(car.index)+1)])
car['DateManufacture'] = pd.Series([fake.date_time_this_decade() for x in range(len(car.index)+1)])
car['DateDelivery'] = pd.Series([fake.date_time_this_year() for x in range(len(car.index)+1)])
car['WarrantyEndDate'] = pd.Series([fake.future_datetime() for x in range(len(car.index)+1)])
car
```

	idCar	NumberDoors	ListPrice	WeightCapacity	Manufacturer	Model	Color	DateManufacture	DateDelivery	WarrantyEndDate	Customization	Trade-in
0	788.0	5.0	26558.0	1384.0	side	VTC	LightSteelBlue	2022-08-06 04:53:48	2023-04-01 20:01:17	2023-05-13 10:35:17		0
1	776.0	4.0	43633.0	1903.0	reveal	AMP	Thistle	2020-10-07 15:56:10	2023-01-29 17:18:33	2023-05-09 04:33:55		0
2	768.0	6.0	46000.0	1875.0	nation	UBQ	RosyBrown	2022-04-11 08:41:26	2023-02-06 13:43:11	2023-05-20 00:22:34		0
3	735.0	6.0	42373.0	1759.0	explain	ADA	Wheat	2021-08-13 07:07:09	2023-04-21 08:23:08	2023-04-27 04:31:28		1
4	947.0	5.0	36327.0	1957.0	but	BCN	LightYellow	2020-05-07 22:40:24	2023-01-26 14:20:01	2023-05-25 12:25:06		1
5	339.0	3.0	45933.0	1759.0	look	ZCL	PapayaWhip	2022-04-28 03:18:11	2023-03-04 12:27:18	2023-04-27 12:56:34		0

- Sales table:

As for the sale table, it requires that the foreign keys match what we already have in our database; we generate our sale data through the SQL insert command.

```
INSERT INTO Sales
VALUES (111, Timestamp('2023-04-24 15:29:30'), 15, 1, 131),
(233, Timestamp('2023-04-25 16:30:30'), 33, 8, 938),
(373, Timestamp('2023-04-23 17:30:30'), 41, 15, 809),
(484, Timestamp('2023-04-22 18:29:30'), 44, 26, 729),
(555, Timestamp('2023-04-21 19:29:30'), 51, 59, 598),
(655, Timestamp('2023-04-22 14:29:30'), 51, 99, 471),
(745, Timestamp('2023-04-23 13:29:30'), 33, 31, 809),
(833, Timestamp('2023-04-24 12:29:30'), 44, 45, 753),
(939, Timestamp('2023-04-25 11:29:30'), 77, 66, 932)
```

- Employee_has_Customer:

As for the Employee_has_Customer table, it requires that the foreign keys match what we already have in our database. We generate our Employee_has_Customer data through the SQL insert command.

The screenshot shows a SQL IDE interface. At the top, there is a toolbar with various icons and a 'Limit to 1000 rows' dropdown. Below the toolbar, the SQL editor contains the following code:

```
1 • INSERT INTO Employee_has_Customer
2   VALUES (77, 66);
3
4 • select *
5   from Employee_has_Customer;
6
```

Below the SQL editor, the 'Result Grid' is displayed. It shows the results of the SQL query. The grid has two columns: 'Employee_idEmployee' and 'Customer_idCustomer'. The data is as follows:

Employee_idEmployee	Customer_idCustomer
15	1
33	8
41	15
44	26
33	31
44	45
51	59
77	66
51	99
NULL	NULL