**Chicago Harley-Davidson Project**

**DSC 451 Database Design for IS**

Yue You

**Introduction**

The database system is a structural data collection that organizations used to meet their business needs. Data is entered into the system and accessed on a routine basis by assigned users. It handles various types and large capacity of data efficiently. The design of the database can be critical to serving multiple use-purposes.

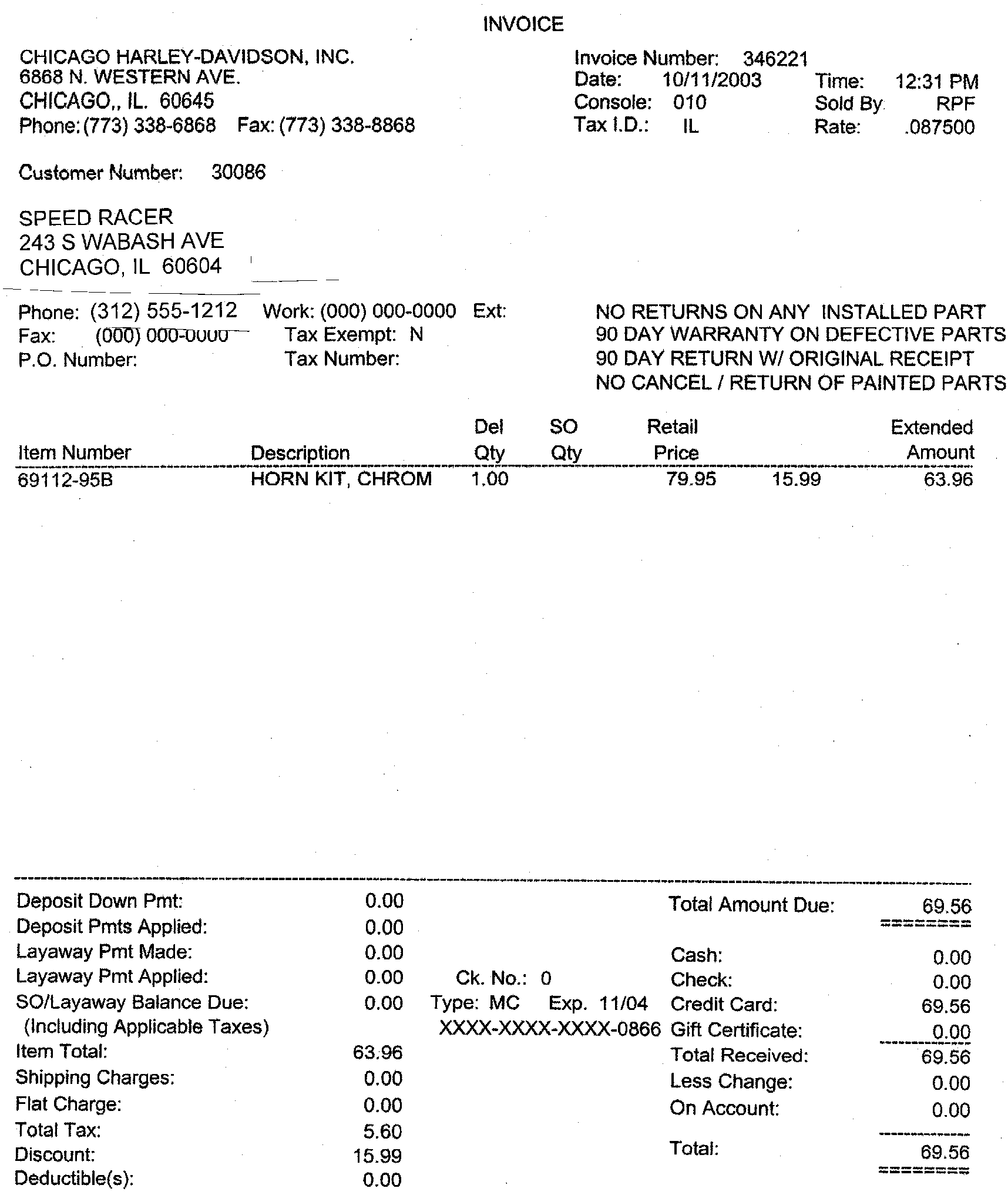
As a data analyst, it is necessary to understand the relational database design and be able to extract the target data from the database system. On the other hand, be able to understand the relational database will make the preliminary data cleaning and analysis much easier.

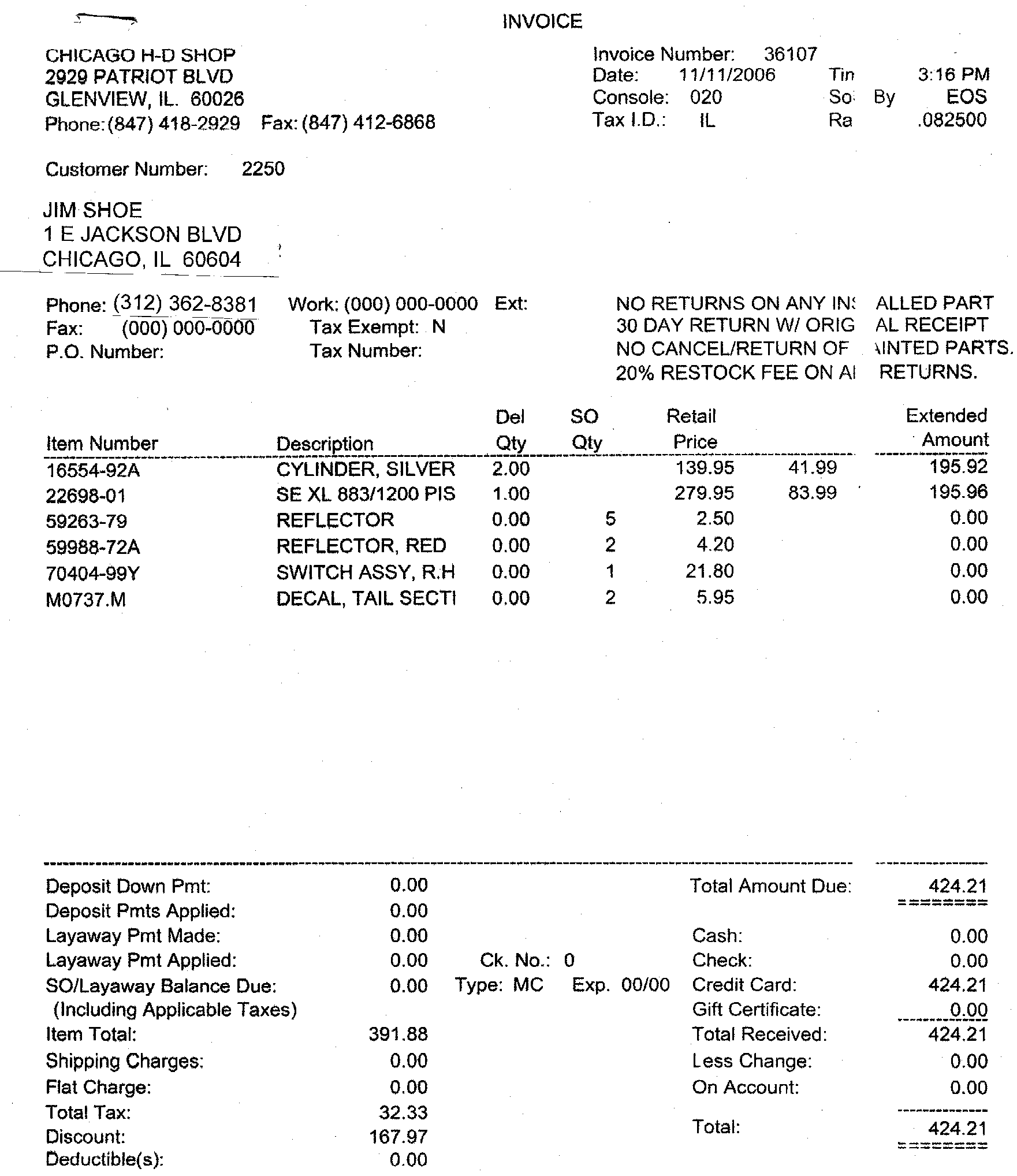
The study of this project to investigate the database relation based on three business transaction invoices from Chicago Harley-Davidson. Through understanding the data in and out on the invoices, three destinated goals need to be achieved in this study:

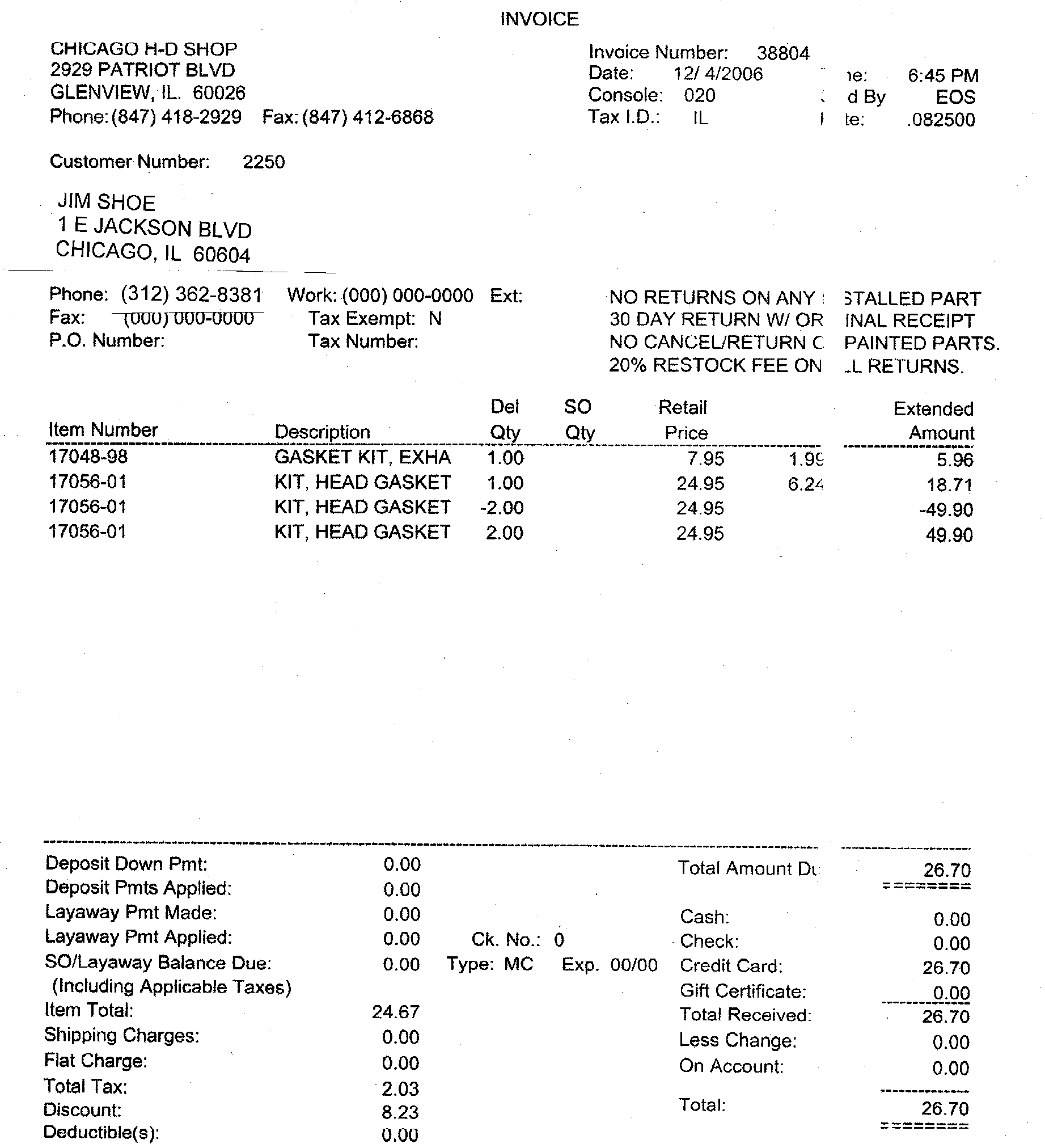
1. To create an Entity Relation Diagram, which represents the data relations on the invoices.
2. To develop the DDL and populate the data into the relational schema.
3. To develop SQL queries to execute demands.

**Project Source: Harley-Davidson Transaction Invoices**

The invoices below represent the three pieces of transactions that took place in Oct. 2003, Nov. 2006 and Dec. 2006.







Each page of the invoices includes various information, such as time, shop, item, item quality, customer info, and the transaction info. To connect the data, we first need to build logical entity-relationship among the all given data. The first step can be done by splitting the attributes and data from large data entry present on the invoices. This step is to help to clarify the information and to know how they possibly linked to each other. The second step is normalization. Normalization is needed to remove the partial dependencies and transitive dependencies so that there won’t be inconsistent data or anomalies when we update the data in the database. After that, the entity-relationship diagram can be created based on a 3rd normal form relations.

**Normalization**

**1st Normal Form**

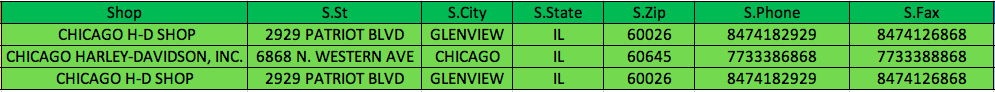
Based on the given information on the invoices, it initially got break down into the five entities. The tables below show the relevant attributes and transaction activities. However, tables are still in the original 1NF (first normal form), which can be found by the data redundancy show in these tables. In order to build entity-relationship and remove the redundancy from the database, we need to transform the first normal form into the second normal form.

***Invoice***

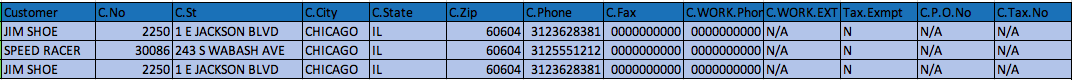
A screenshot of a cell phone

Description automatically generated

***Shop***



***Customer***



***Item (Product)***

***A screenshot of a cell phone

Description automatically generated***

***Transaction***

***A screenshot of a cell phone

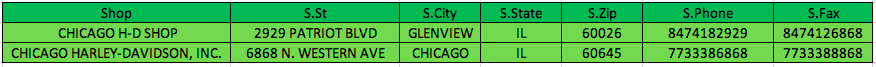
Description automatically generated***

**2nd Normal Form**

The normalization of 1NF relations to 2NF involves the removal of partial dependencies. If a partial dependency exists, we remove the partially dependent attributes from the relation by placing them in a new relation along with a copy of their determinant. This step of normalization is aimed to connect the 1-to-1 relations and the 1-to many relations as well.

To creating both entity relations, we need to define the primary keys of each entity table and to identify the high and low cardinalities of each table. High cardinality contains unique or uncommon data. On the other hand, low cardinality has more common or repeated data. The relations between tables are based on the high and low cardinality; they connect the entities' tables by clearly identifying the primary keys and foreign keys. The tables below are just examples of the 2nd normal form. The very first column of each table is the primary key of each entity table. There is an entity-relationship diagram created by Oracle SQL Developer with DDL that shows the relationship among entities. Since the invoices did not indicate any many-to-many relations, it won't be required to conduct further normalization to the 3rd normal form. However, the 3rd normal form can be done for this specific case.

***Shop***

******

***Customer***

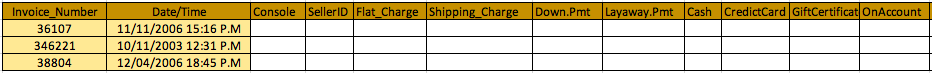
******

***Item (Product)***

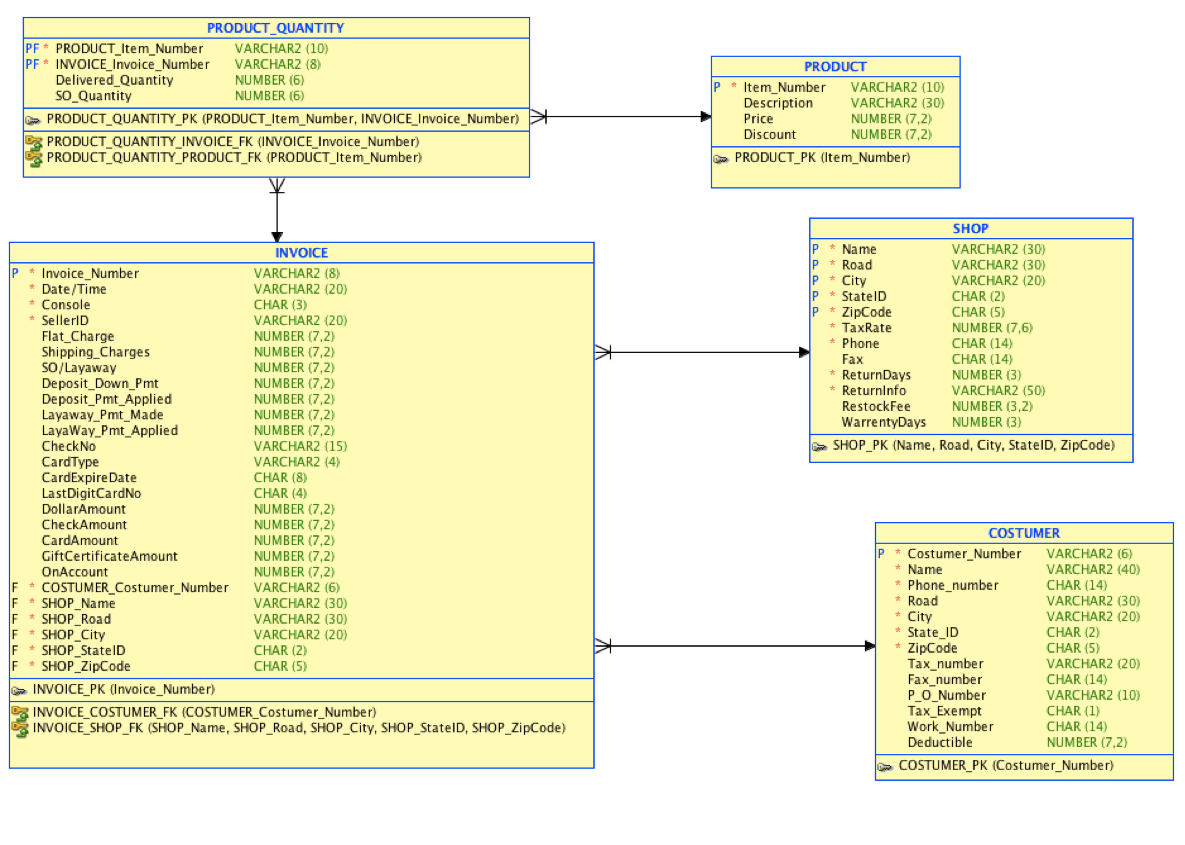
***A screenshot of a cell phone

Description automatically generated***

***Invoice***

******

**ER-Diagram**



1. DDL code:

CREATE TABLE product (

item\_number VARCHAR2(10) NOT NULL,

description VARCHAR2(30),

price NUMBER(7,2),

discount NUMBER(7,2)

);

ALTER TABLE product ADD CONSTRAINT product\_pk PRIMARY KEY ( item\_number );

CREATE TABLE shop (

name VARCHAR2(30) NOT NULL,

road VARCHAR2(30) NOT NULL,

city VARCHAR2(20) NOT NULL,

stateid CHAR(2) NOT NULL,

zipcode CHAR(5) NOT NULL,

taxrate NUMBER(7,6) NOT NULL,

phone CHAR(14) NOT NULL,

fax CHAR(14),

returndays NUMBER(3) NOT NULL,

returninfo VARCHAR2(50) NOT NULL,

restockfee NUMBER(3,2),

warrentydays NUMBER(3)

);

ALTER TABLE shop

ADD CONSTRAINT shop\_pk PRIMARY KEY ( name,

road,

city,

stateid,

zipcode );

CREATE TABLE costumer (

costumer\_number VARCHAR2(6) NOT NULL,

name VARCHAR2(40) NOT NULL,

phone\_number CHAR(14) NOT NULL,

road VARCHAR2(30) NOT NULL,

city VARCHAR2(20) NOT NULL,

state\_id CHAR(2) NOT NULL,

zipcode CHAR(5) NOT NULL,

tax\_number VARCHAR2(20),

fax\_number CHAR(14),

p\_o\_number VARCHAR2(10),

tax\_exempt CHAR(1),

work\_number CHAR(14),

deductible NUMBER(7,2)

);

CREATE TABLE invoice (

invoice\_number VARCHAR2(8) NOT NULL,

"Date/Time" VARCHAR2(20) NOT NULL,

console CHAR(3) NOT NULL,

sellerid VARCHAR2(20) NOT NULL,

flat\_charge NUMBER(7,2),

shipping\_charges NUMBER(7,2),

"SO/Layaway" NUMBER(7,2),

deposit\_down\_pmt NUMBER(7,2),

deposit\_pmt\_applied NUMBER(7,2),

layaway\_pmt\_made NUMBER(7,2),

layaway\_pmt\_applied NUMBER(7,2),

checkno VARCHAR2(15),

cardtype VARCHAR2(4),

cardexpiredate CHAR(8),

lastdigitcardno CHAR(4),

dollaramount NUMBER(7,2),

checkamount NUMBER(7,2),

cardamount NUMBER(7,2),

giftcertificateamount NUMBER(7,2),

onaccount NUMBER(7,2),

costumer\_costumer\_number VARCHAR2(6) NOT NULL,

shop\_name VARCHAR2(30) NOT NULL,

shop\_road VARCHAR2(30) NOT NULL,

shop\_city VARCHAR2(20) NOT NULL,

shop\_stateid CHAR(2) NOT NULL,

shop\_zipcode CHAR(5) NOT NULL

);

ALTER TABLE invoice ADD CONSTRAINT invoice\_pk PRIMARY KEY ( invoice\_number );

ALTER TABLE invoice

ADD CONSTRAINT invoice\_costumer\_fk FOREIGN KEY ( costumer\_costumer\_number )

REFERENCES costumer ( costumer\_number );

ALTER TABLE invoice

ADD CONSTRAINT invoice\_shop\_fk FOREIGN KEY ( shop\_name,

shop\_road,

shop\_city,

shop\_stateid,

shop\_zipcode )

REFERENCES shop ( name,

road,

city,

stateid,

zipcode );

CREATE TABLE product\_quantity (

product\_item\_number VARCHAR2(10) NOT NULL,

invoice\_invoice\_number VARCHAR2(8) NOT NULL,

delivered\_quantity NUMBER(6),

so\_quantity NUMBER(6)

);

ALTER TABLE product\_quantity ADD CONSTRAINT product\_quantity\_pk PRIMARY KEY ( product\_item\_number,

invoice\_invoice\_number );

ALTER TABLE product\_quantity

ADD CONSTRAINT product\_quantity\_invoice\_fk FOREIGN KEY ( invoice\_invoice\_number )

REFERENCES invoice ( invoice\_number );

ALTER TABLE product\_quantity

ADD CONSTRAINT product\_quantity\_product\_fk FOREIGN KEY ( product\_item\_number )

REFERENCES product ( item\_number );

**Testing**

The final step of this project is to test the validity of the logical database. There are two testings involved in the step. By executing the DDL code, we need to answer two questions:

1. How many parts did Jim Shoe purchase in November, 2006?

SELECT SUM (delivered\_quantity) +

SUM (SO\_quantity) as purchased\_quantity

FROM product\_quatity

WHERE invoice\_invoice\_number IN (

SELECT invoice\_number

FROM invoice

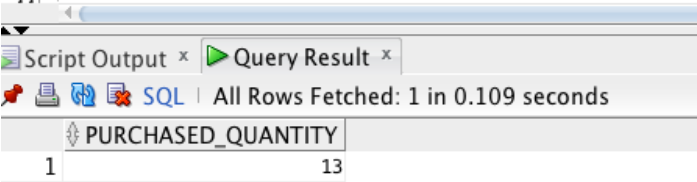
WHERE SUBSTR (“Date/Time, 4,8) = ‘NOV-2006’

AND customer\_customer\_number IN (

SELECT customer\_number

FROM customer

WHERE name = ‘Jim Shoe’))



1. What was the total amount of all purchase by Speed Racer?

CREATE VIEW tax AS

SELECT SUM ((price – discount) \* (taxrate)) AS tax

FROM Product, Product\_quantity, Invoice, Shop

WHERE PRODUCT.ITEM\_NUMBER = PRODUCT\_QUANTITY.PRODUCT\_ITEM\_NUMBER

AND PRODUCT\_QUANTITY.INVOICE\_INVOICE\_NUMBER = invoice.invoice\_number

AND invoice.shop\_name = shop.name

AND invoice\_invoice\_number IN (

SELECT invoice\_number

FROM invoice

WHERE customer\_customer\_number IN (

SELECT customer\_number

FROM customer

WHERE name = ‘Speed Racer’));

SELECT SUM ((price – discount) + (tax)) AS purchased\_amount

FROM product, tax, Product\_quantity, Invoice, Shop

WHERE PRODUCT.ITEM\_NUMBER = PRODUCT\_QUANTITY.PRODUCT\_ITEM\_NUMBER

AND PRODUCT\_QUANTITY.INVOICE\_INVOICE\_NUMBER = invoice.invoice\_number

AND invoice.shop\_name = shop.name

AND invoice\_invoice\_number IN (

SELECT invoice\_number

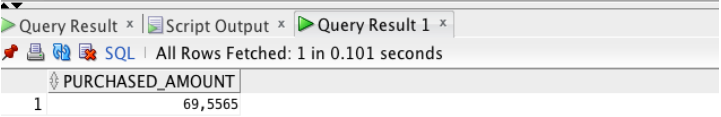
FROM invoice

WHERE customer\_customer\_number IN (

SELECT customer\_number

FROM customer

WHERE name = ‘Speed Racer’));



Based on these two testing, we can certainly verify the validity of the DDL and the logical relationship among the entities from the invoice data.

**Conclusion**

The purpose of this project is to learn and apply the database entity-relationship through a practical case study. This case study is creatively designed with three pieces of the invoice, which is very useful and helpful to understand how databases were designed to improve the efficiency of business activities and integrity and quality of data storage. As a data analyst, the design of the database helped us to understand the data correlations. On the other hand, a good design of the database will reduce the amount of preliminary data cleaning process.