# DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES2201800075 AMITHA NAYAK

The carSales database is a management system that uses database technology to construct, maintain and manipulate various kinds of data about the buying and selling of cars, trucks, bikes, ships, ..etc.

This database contains 8 tables.
(customers, employees, offices, orders, orderdetails, payments, products, productLine)

There are 2 triggers implemented on this database:

1) On the table orderdetails: This trigger updates the product inventory (i.e in particular the quantity in stock) after the customer purchases an item.
2) On the table payments: This trigger awards frequent heavy buyers by increasing their credit limit.

I have implemented 5 queries on this database (2 correlated-nested, 1 aggregate, 2 outer-joins).

This project aims at computerizing the manual process of a carSales storeroom.
This project is made for the company to keep a track of all the automobiles that are bought or sold.

The data stored on the database can be used for further data analysis that will help in creating a greater influx of profit for the company.

# Introduction

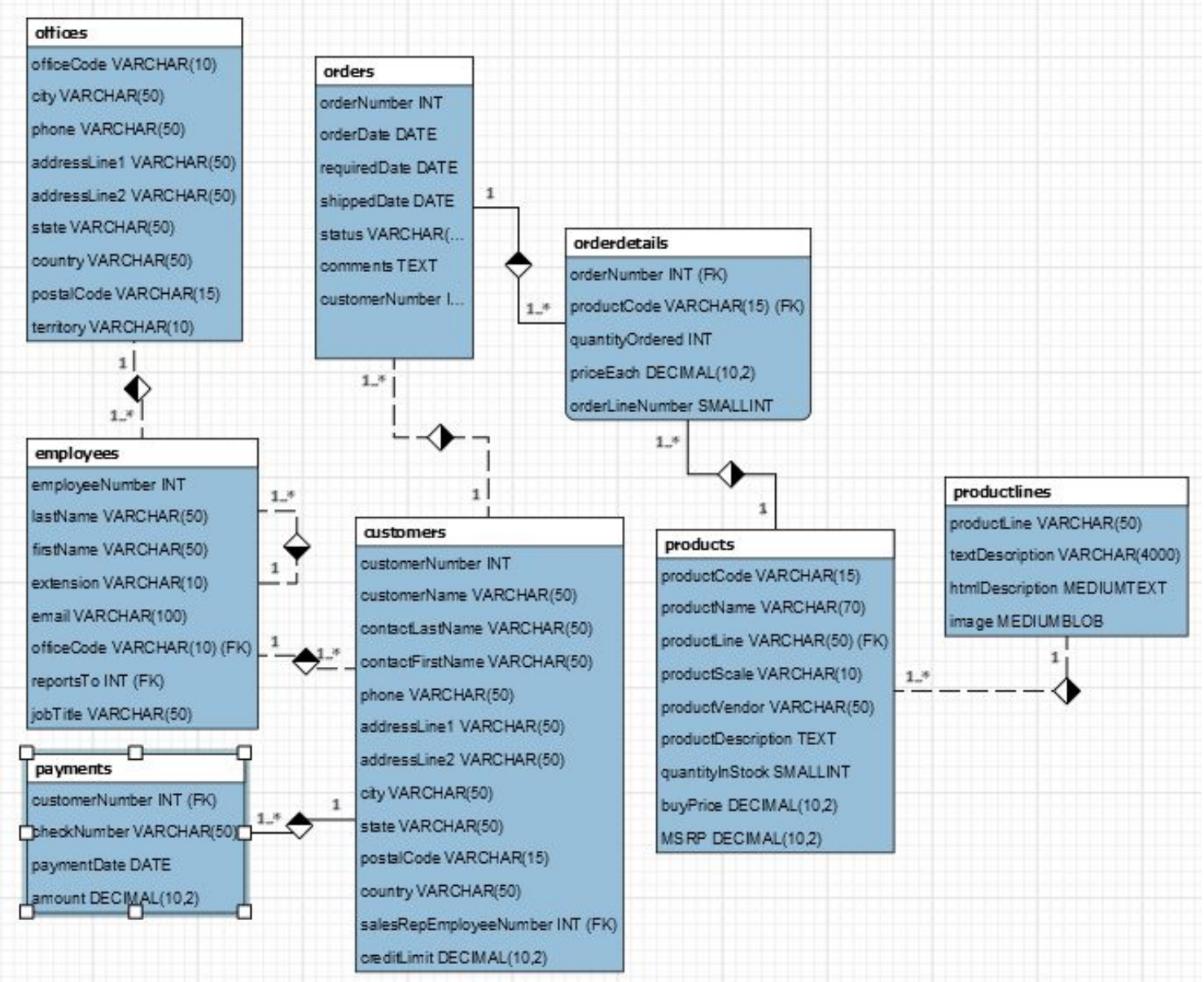
The Classic Models Inc. database has been developed as part of the Eclipse BIRT (Business Intelligence Reporting Tools) project. Its main goal is to be obvious and simple, yet able to support a wide range of interesting report examples. The database represents a fictitious company: Classic Models Inc. which buys collectable model cars, trains, trucks, buses, trains and ships directly from manufacturers and sells them to distributors across the globe.

The database consists of the following tables:

- **Customers**: stores customer's data.
- **Products**: stores a list of scale model cars.
- **ProductLines**: stores a list of product line categories.
- **Orders**: stores sales orders placed by customers.
- **OrderDetails**: stores sales order line items for each sales order.
- **Payments**: stores payments made by customers based on their accounts.
- **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
- **Offices**: stores sales office data

# Data Model

## ER MODEL:



SCHEMA:

```
mysql> desc offices;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| officeCode   | varchar(10) | NO   | PRI | NULL    |       |
| city         | varchar(50) | NO   |     | NULL    |       |
| phone        | varchar(50) | NO   | UNI | NULL    |       |
| addressLine1 | varchar(50) | NO   |     | NULL    |       |
| addressLine2 | varchar(50) | YES  |     | NULL    |       |
| state        | varchar(50) | YES  |     | NULL    |       |
| country      | varchar(50) | NO   |     | NULL    |       |
| postalCode   | varchar(15) | NO   |     | NULL    |       |
| territory    | varchar(10) | NO   |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
9 rows in set (0.07 sec)
```

```
mysql> desc orders;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| orderNumber    | int         | NO   | PRI | NULL    |       |
| orderDate      | date        | NO   |     | NULL    |       |
| requiredDate   | date        | NO   |     | NULL    |       |
| shippedDate    | date        | YES  |     | NULL    |       |
| status         | varchar(15) | NO   |     | NULL    |       |
| comments       | text        | YES  |     | NULL    |       |
| customerNumber | int         | NO   | MUL | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
7 rows in set (0.33 sec)
```

```
mysql> desc orderdetails;
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| orderNumber     | int           | NO   | PRI | NULL    |       |
| productCode     | varchar(15)   | NO   | PRI | NULL    |       |
| quantityOrdered | int           | NO   |     | NULL    |       |
| priceEach       | decimal(10,2) | NO   |     | NULL    |       |
| orderLineNumber | smallint      | NO   |     | NULL    |       |
+-----------------+---------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

```
mysql> desc employees;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| employeeNumber | int          | NO   | PRI | NULL    |       |
| lastName       | varchar(50)  | NO   |     | NULL    |       |
| firstName      | varchar(50)  | NO   |     | NULL    |       |
| extension      | varchar(10)  | NO   |     | NULL    |       |
| email          | varchar(100) | NO   |     | NULL    |       |
| officeCode     | varchar(10)  | NO   | MUL | NULL    |       |
| reportsTo      | int          | YES  | MUL | NULL    |       |
| jobTitle       | varchar(50)  | NO   |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
8 rows in set (0.12 sec)
```

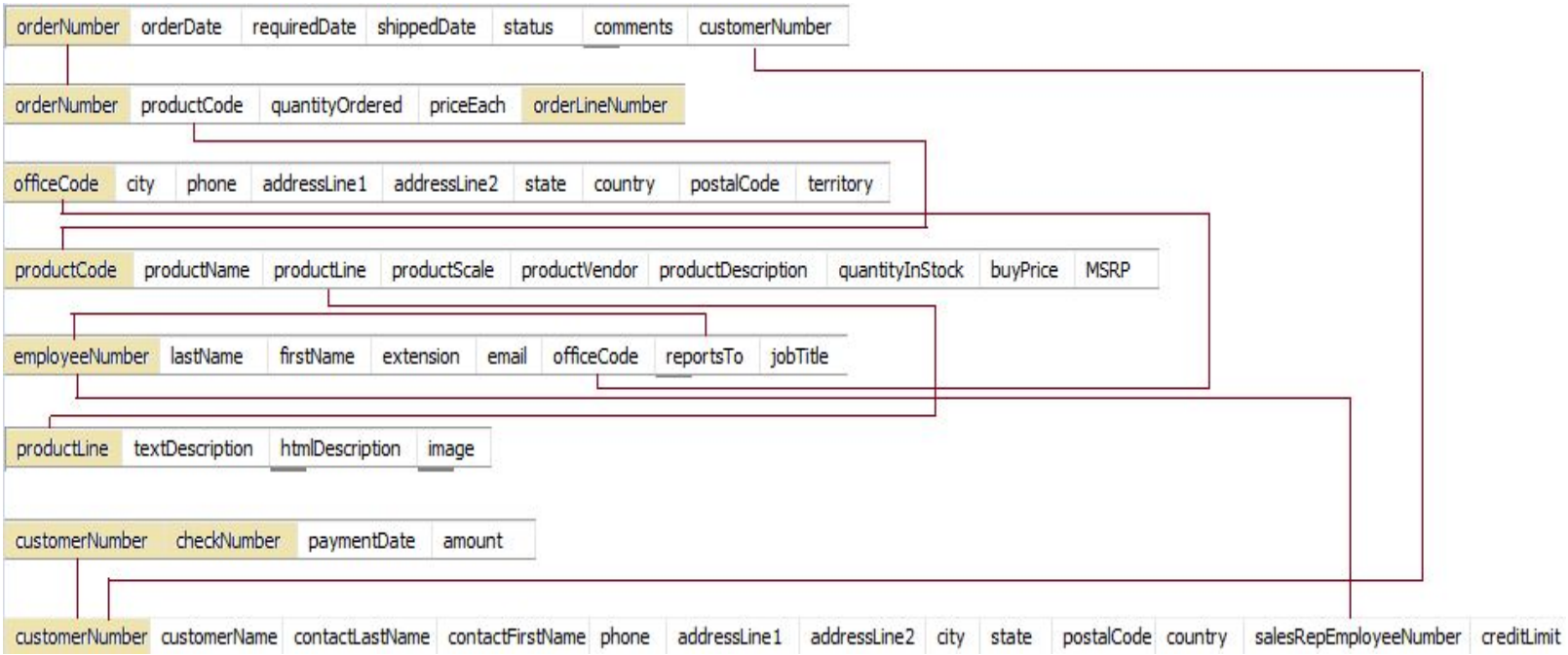
```
mysql> desc customers;
+-------------------------+---------------+------+-----+---------+-------+
| Field                   | Type          | Null | Key | Default | Extra |
+-------------------------+---------------+------+-----+---------+-------+
| customerNumber          | int           | NO   | PRI | NULL    |       |
| customerName            | varchar(50)   | NO   |     | NULL    |       |
| contactLastName         | varchar(50)   | NO   |     | NULL    |       |
| contactFirstName        | varchar(50)   | NO   |     | NULL    |       |
| phone                   | varchar(50)   | NO   |     | NULL    |       |
| addressLine1            | varchar(50)   | NO   |     | NULL    |       |
| addressLine2            | varchar(50)   | YES  |     | NULL    |       |
| city                    | varchar(50)   | NO   |     | NULL    |       |
| state                   | varchar(50)   | YES  |     | NULL    |       |
| postalCode              | varchar(15)   | YES  |     | NULL    |       |
| country                 | varchar(50)   | NO   |     | NULL    |       |
| salesRepEmployeeNumber  | int           | YES  | MUL | NULL    |       |
| creditLimit             | decimal(10,2) | YES  |     | NULL    |       |
+-------------------------+---------------+------+-----+---------+-------+
13 rows in set (2.15 sec)
```

```
mysql> desc products;
+--------------------+---------------+------+-----+---------+-------+
| Field              | Type          | Null | Key | Default | Extra |
+--------------------+---------------+------+-----+---------+-------+
| productCode        | varchar(15)   | NO   | PRI | NULL    |       |
| productName        | varchar(70)   | NO   | UNI | NULL    |       |
| productLine        | varchar(50)   | NO   | MUL | NULL    |       |
| productScale       | varchar(10)   | NO   |     | NULL    |       |
| productVendor      | varchar(50)   | NO   |     | NULL    |       |
| productDescription | text          | NO   |     | NULL    |       |
| quantityInStock    | smallint      | NO   |     | NULL    |       |
| buyPrice           | decimal(10,2) | NO   |     | NULL    |       |
| MSRP               | decimal(10,2) | NO   |     | NULL    |       |
+--------------------+---------------+------+-----+---------+-------+
9 rows in set (0.15 sec)
```

```
mysql> desc productLines;
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| productLine     | varchar(50)   | NO   | PRI | NULL    |       |
| textDescription | varchar(4000) | YES  |     | NULL    |       |
| htmlDescription | mediumtext    | YES  |     | NULL    |       |
| image           | mediumblob    | YES  |     | NULL    |       |
+-----------------+---------------+------+-----+---------+-------+
4 rows in set (0.06 sec)
```

```
mysql> desc payments;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| customerNumber | int          | NO   | PRI | NULL    |       |
| checkNumber    | varchar(50)  | NO   | PRI | NULL    |       |
| paymentDate    | date         | NO   |     | NULL    |       |
| amount         | decimal(10,2)| NO   |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
4 rows in set (0.12 sec)
```

## RELATIONAL SCHEMA:




# CHOICE OF KEYS:

## PRIMARY KEYS

customerNumber, productCode, employeeNumber, officeCode, productLine, orderNumber
(customerNumber, checkNumber), (orderNumber, orderLineNumber)

## CANDIDATE KEYS

checkNumber, productName, phone (in relation office)

## FOREIGN KEYS

(referred - referred by)
employeeNumber - reportTo (in relation employees) ,salesRepEmployeeNumber (in relation customers)
officeCode - officeCode (in relation employees)
productCode - productCode (in relation orderdetails)
productLine - productLine (in relation products)
customerNumber - customerNumber (in relation payments), customerNumber (in relation orders)
orderNumber - orderNumber (in relation orderdetails)

# DATA TYPES USED

## INT

In relation orders - orderNumber, customerNumber
In relation orderdetails - orderNumber, quantityOrdered
In relation employees - employeeNUmber, reportsTo
In relation customers - customerNumber, salesRepEmployeeNumber
In relation payments - customerNumber

## SMALLINT

In relation orderdetails - orderLineNumber
In relation products - quantityInStock

## DECIMAL

In relation orderdetails - priceEach
In relation customers - creditLimit
In relation products - buyPrice, MSRP
In relation payments - amount

## VARCHAR

In relation office - officeCode, city, phone, addressLine1, addressLine2, state, country, postalCode, territory
In relation orders - status
In relation orderdetails - productCode
In relation employees - lastName, firstName, extension, email, officeCode, jobTitle
In relation customers - customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, country
In relation products - productCode, productName, productLine, productScale, productVendor
In relation productLine - productLine, textDescription
In relation payments - checkNumber

## TEXT

In relation orders - comments
In relation products - productDescription

## DATE

In relation orders - orderDate, requiredDate, shippedDate
In relation payments - paymentDate

## MEDIUMTEXT

In relation productLine - htmlDescription

## MEDIUMBLOB

In relation productLine - image

# FD and Normalization

## FD:

In relation customers(customerNumber*,customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, country, salesRepEmployeeNumber, creditLimit)
customerNumber-> customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, salesRepEmployeeNumber, creditLimit
city-> state, country

In relation employees(employeeNumber*, lastName, firstName, extension, email, officeCode, reportsTo, jobTitle)
employeeNumber-> lastName, firstName, extension, email, officeCode, reportsTo, jobTitle

In relation orders(orderNumber*, orderDate, requiredDate, shippedDate, status, comments, customerNumber)
orderNumber-> orderDate, requiredDate, shippedDate, status, comments, customerNumber

In relation orderdetails(orderNumber*, productCode, quantityOrdered, priceEach, orderLineNumber*)
orderNumber, orderLineNumber-> productCode, quantityOrdered, priceEach
productCode-> priceEach

In relation products(productCode*, productName, productLine, productScale, productVendor, productDescription, quantityInStock, buyPrice, MSRP)
productCode-> productName, productLine, productScale, productVendor, productDescription, quantityInStock, buyPrice, MSRP
productName-> productCode, productLine, productScale, productVendor, productDescription, quantityInStock, buyPrice, MSRP

In relation productLines(productLine*, textDescription, htmlDescription, image)
productLine-> textDescription, htmlDescription, image

In relation payments(customerNumber*, checkNumber*, paymentDate, amount)

customerNumber, checkNumber-> paymentDate, amount
checkNumber->customerNumber,  paymentDate, amount

In relation offices(officeCode*, city, phone, addressLine1, addressLine2, state, country, postalCode, territory)
officeCode-> city, phone, addressLine1, addressLine2, state, country, postalCode, territory
phone->  officeCode, city, addressLine1, addressLine2, state, country, postalCode, territory
city-> country
postalCode,city,country->territory

# NORMAL FORMS OF EACH TABLE:

customers - 2NF
employees - 3NF
offices - 2NF
Products - 3NF
productLine - 3NF
Orders - 3NF
orderDetails - 3NF
Payments - 1NF

# NORMALISATION:

## 1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

There are no tables in my database which break 1NF rules.

However in the prior versions of table payments (customerNumber*, checkNumber, paymentDate, amount) the customer was allowed to fill in multiple values for checkNumber, as a customer could have multiple orders tied to them and hence multiple checks.

| customerNumber | checkNumber | paymentDate | amount |
|---|---|---|---|
| 103 | 'HQ336336', 'JM555205' | '2004-10-19', '2003-06-05' | 6066.78, 14571.44 |
| 112 | ‘BO864823’ | ‘2004-12-17’ | 14191.12 |

This would lead to a violation of 1NF.
To bring the table into 1NF form, we split each record in this manner.

| customerNumber | checkNumber | paymentDate | amount |
|---|---|---|---|
| 103 | 'HQ336336’ | '2004-10-19' | 6066.78 |
| 103 | 'JM555205' | '2003-06-05' | 14571.44 |
| 112 | ‘BO864823’ | ‘2004-12-17’ | 14191.12 |

Now as the primary key, i.e customerNumber is no more unique, we make both customerNumber and checkNumber as the primary key.

## 2NF (Second Normal Form) Rules

- Rule 1- Be in 1NF
- Rule 2- No partial dependency

In table orderdetails(orderNumber*, productCode, quantityOrdered, priceEach, orderLineNumber*)
Each order contains unique products (order line items) and is split as per the order line numbers into different records. Each order line item reflects the negotiated price per product (which is based on the corresponding product’s MSRP) as well as the quantity per product.
Hence both the attributes orderNumber and orderLineNumber are required to uniquely identify each record. It is in 2NF.

In table payments(customerNumber*, checkNumber*, paymentDate, amount)
Customers make payments (by check), however it is typical to have more than one order under a particular customer. In many such cases, a customer might have multiple checkNumbers tied to him under different orders.
In this table each record can be independently accessed by the checkNumber alone, as the checkNumber attribute values are unique. This shows partial dependency and hence violates 2NF.
To bring the table into 2NF form, we split the table into two.
**Original Table:**

| customerNumber | checkNumber | paymentDate | amount |
|---|---|---|---|
| 103 | 'HQ336336’ | '2004-10-19' | 6066.78 |
| 103 | 'JM555205' | '2003-06-05' | 14571.44 |
| 112 | ‘BO864823’ | ‘2004-12-17’ | 14191.12 |

**Table 1**

| checkNumber | ID | paymentDate | amount |
|---|---|---|---|
| 'HQ336336’ | 1 | '2004-10-19' | 6066.78 |
| 'JM555205' | 1 | '2003-06-05' | 14571.44 |
| ‘BO864823’ | 2 | ‘2004-12-17’ | 14191.12 |

**Table 2**

| ID | customerNumber |
|---|---|
| 1 | 103 |
| 2 | 112 |

Here the ID attribute acts as a foriegn key in case of Table 1 and a primary key in case of Table 2. ID helps us relate the customer details to the checkNumber associated with each order.

This also helps in data redundancy, as now the customerNumbers needn’t be repeated. Table2 associates a particular customer with one unique ID.

## 3NF (Third Normal Form) Rules

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies

In table customers(customerNumber*,customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, salesRepEmployeeNumber, creditLimit)
Customers feed in their address, however it is typical for customers to shift at times, this leads to a change in their address. However attributes such as state and country show functional dependency on the attribute city and hence need to be updated due to the changes in address. This is a case of transitional dependency and hence violates 3NF.
To bring the table into 3NF form, we split the table into two.

**Original Table**

| customerNumber | customerName | contactLastName | contactFirstName | phone | addressLine1 | addressLine2 | city | state | postalCode | country | salesRepEmployeeNumber | creditLimit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 103 | Atelier graphique | Schmitt | Carine | 40.32.2555 | 54, rue Royale | NULL | Nantes | NULL | 44000 | France | 1370 | 21000.00 |
| 112 | Signal Gift Stores | King | Jean | 7025551838 | 8489 Strong St. | NULL | Las Vegas | NV | 83030 | USA | 1166 | 71800.00 |
| 114 | Australian Collectors, Co. | Ferguson | Peter | 03 9520 4555 | 636 St Kilda Road | Level 3 | Melbourne | Victoria | 3004 | Australia | 1611 | 117300.00 |
| 119 | La Rochelle Gifts | Labrune | Janine | 40.67.8555 | 67, rue des Cinquante Otages | NULL | Nantes | NULL | 44000 | France | 1370 | 118200.00 |

**Table 1**

| customerNumber | customerName | contactLastName | contactFirstName | phone | addressLine1 | addressLine2 | city | postalCode | salesRepEmployeeNumber | creditLimit |
|---|---|---|---|---|---|---|---|---|---|---|
| 103 | Atelier graphique | Schmitt | Carine | 40.32.2555 | 54, rue Royale | NULL | Nantes | 44000 | 1370 | 21000.00 |
| 112 | Signal Gift Stores | King | Jean | 7025551838 | 8489 Strong St. | NULL | Las Vegas | 83030 | 1166 | 71800.00 |
| 114 | Australian Collectors, Co. | Ferguson | Peter | 03 9520 4555 | 636 St Kilda Road | Level 3 | Melbourne | 3004 | 1611 | 117300.00 |
| 119 | La Rochelle Gifts | Labrune | Janine | 40.67.8555 | 67, rue des Cinquante Otages | NULL | Nantes | 44000 | 1370 | 118200.00 |
| 121 | Baane Mini Imports | Bergulfsen | Jonas | 07-98 9555 | Erling Skakkes gate 78 | NULL | Stavern | 4110 | 1504 | 81700.00 |

**Table 2**

| city | state | country |
|---|---|---|
| Nantes | NULL | France |
| Las Vegas | NV | USA |
| Melbourne | Victoria | Australia |
| Stavern | NULL | Norway |

Here the city attribute acts as a foriegn key in case of Table 1 and a primary key in case of Table 2. This also helps in data redundancy, as now the attributes state and country needn't be repeated.

# DDL

```
CREATE TABLE `customers` (
  `customerNumber` int(11) NOT NULL,
  `customerName` varchar(50) NOT NULL,
  `contactLastName` varchar(50) NOT NULL,
  `contactFirstName` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL,
  `addressLine1` varchar(50) NOT NULL,
  `addressLine2` varchar(50) DEFAULT NULL,
  `city` varchar(50) NOT NULL,
  `state` varchar(50) DEFAULT NULL,
  `postalCode` varchar(15) DEFAULT NULL,
  `country` varchar(50) NOT NULL,
  `salesRepEmployeeNumber` int(11) DEFAULT NULL,
  `creditLimit` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`customerNumber`),
  KEY `salesRepEmployeeNumber` (`salesRepEmployeeNumber`),
  CONSTRAINT `customers_ibfk_1` FOREIGN KEY (`salesRepEmployeeNumber`)
REFERENCES `employees` (`employeeNumber`)
);

CREATE TABLE `employees` (
  `employeeNumber` int(11) NOT NULL,
  `lastName` varchar(50) NOT NULL,
  `firstName` varchar(50) NOT NULL,
  `extension` varchar(10) NOT NULL,
  `email` varchar(100) NOT NULL,
  `officeCode` varchar(10) NOT NULL,
  `reportsTo` int(11) DEFAULT NULL,
  `jobTitle` varchar(50) NOT NULL,
  PRIMARY KEY (`employeeNumber`),
  KEY `reportsTo` (`reportsTo`),
  KEY `officeCode` (`officeCode`),
  CONSTRAINT `employees_ibfk_1` FOREIGN KEY (`reportsTo`) REFERENCES
`employees` (`employeeNumber`),
  CONSTRAINT `employees_ibfk_2` FOREIGN KEY (`officeCode`) REFERENCES
`offices` (`officeCode`)
);
```

```
CREATE TABLE `offices` (
  `officeCode` varchar(10) NOT NULL,
  `city` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL, UNIQUE
  `addressLine1` varchar(50) NOT NULL,
  `addressLine2` varchar(50) DEFAULT NULL,
  `state` varchar(50) DEFAULT NULL,
  `country` varchar(50) NOT NULL,
  `postalCode` varchar(15) NOT NULL,
  `territory` varchar(10) NOT NULL,
  PRIMARY KEY (`officeCode`)
);

CREATE TABLE `orders` (
  `orderNumber` int(11) NOT NULL,
  `orderDate` date NOT NULL,
  `requiredDate` date NOT NULL,
  `shippedDate` date DEFAULT NULL,
  `status` varchar(15) NOT NULL,
  `comments` text,
  `customerNumber` int(11) NOT NULL,
  PRIMARY KEY (`orderNumber`),
  KEY `customerNumber` (`customerNumber`),
  CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`customerNumber`) REFERENCES
`customers` (`customerNumber`)
);

CREATE TABLE `orderdetails` (
  `orderNumber` int(11) NOT NULL,
  `productCode` varchar(15) NOT NULL,
  `quantityOrdered` int(11) NOT NULL,
  `priceEach` decimal(10,2) NOT NULL,
  `orderLineNumber` smallint(6) NOT NULL,
  PRIMARY KEY (`orderNumber`,`productCode`),
  KEY `productCode` (`productCode`),
  CONSTRAINT `orderdetails_ibfk_1` FOREIGN KEY (`orderNumber`)
REFERENCES `orders` (`orderNumber`),
  CONSTRAINT `orderdetails_ibfk_2` FOREIGN KEY (`productCode`)
REFERENCES `products` (`productCode`)
);


CREATE TABLE `payments` (
```

```
  `customerNumber` int(11) NOT NULL,
  `checkNumber` varchar(50) NOT NULL,
  `paymentDate` date NOT NULL,
  `amount` decimal(10,2) NOT NULL,
  PRIMARY KEY (`customerNumber`,`checkNumber`),
  CONSTRAINT `payments_ibfk_1` FOREIGN KEY (`customerNumber`)
REFERENCES `customers` (`customerNumber`)
);

CREATE TABLE `productlines` (
  `productLine` varchar(50) NOT NULL,
  `textDescription` varchar(4000) DEFAULT NULL,
  `htmlDescription` mediumtext,
  `image` mediumblob,
  PRIMARY KEY (`productLine`)
);

CREATE TABLE `products` (
  `productCode` varchar(15) NOT NULL,
  `productName` varchar(70) NOT NULL, UNIQUE
  `productLine` varchar(50) NOT NULL,
  `productScale` varchar(10) NOT NULL,
  `productVendor` varchar(50) NOT NULL,
  `productDescription` text NOT NULL,
  `quantityInStock` smallint(6) NOT NULL,
  `buyPrice` decimal(10,2) NOT NULL,
  `MSRP` decimal(10,2) NOT NULL,
  PRIMARY KEY (`productCode`),
  KEY `productLine` (`productLine`),
  CONSTRAINT `products_ibfk_1` FOREIGN KEY (`productLine`) REFERENCES
`productlines` (`productLine`)
);
```

## CHECK CONSTRAINTS

```
ALTER TABLE products
ADD CONSTRAINT products_profit_ensured CHECK(buyPrice <= MSRP);
ALTER TABLE products
ADD CONSTRAINT products_negative_stock CHECK(quantityInStock >=0);
ALTER TABLE payments
ADD CONSTRAINT amount_zero_or_negative CHECK(amount>0);
```

# Triggers

## Trigger 01:

Trigger Name: ‘orderdetails_BEFORE_INSERT’
Trigger applied on: orderdetails(orderNumber*, productCode, quantityOrdered, priceEach, orderLineNumber*)

```
CREATE DEFINER=`root`@`localhost` TRIGGER `orderdetails_BEFORE_INSERT`
BEFORE INSERT ON `orderdetails` FOR EACH ROW
BEGIN
      declare max_quantity int;
      declare msg varchar(128);
      SELECT quantityInStock into max_quantity FROM products WHERE
      productCode=new.productCode;
      IF max_quantity<new.quantityOrdered then
            set msg = concat('orderdetails_BEFORE_INSERT: Quantity
            Exceeds Available Stock', cast(new.orderNumber as char));
            signal sqlstate '45000' set message_text = msg;
      end IF;
      UPDATE products
      SET quantityInStock = quantityInStock - new.quantityOrdered
      WHERE productCode = new.productCode;
END;
```

This trigger updates the product inventory everytime the customer places an order. In particular, it updates the quantity in stock. If the quantity ordered by the customer exceeds the quantity in stock, this trigger raises an user-defined error.

## Trigger 02:

Trigger Name: ‘payments_AFTER_INSERT’
Trigger applied on: payments(customerNumber*, checkNumber*, paymentDate, amount)

```
CREATE DEFINER=`root`@`localhost` TRIGGER `payments_AFTER_INSERT`
BEFORE INSERT ON `payments` FOR EACH ROW BEGIN
DECLARE regular_factor int(3);
DECLARE amt decimal(10,2);
DECLARE msg varchar(50);
```

```
SELECT COUNT(customerNumber) INTO regular_factor FROM payments WHERE
customerNumber=new.customerNumber;
SELECT SUM(amount) INTO amt FROM payments WHERE
customerNumber=new.customerNumber;
IF regular_factor>3 and amt>=50000.00 THEN
/* is a priced customer */
IF new.amount > 10000 THEN
update customers
SET creditLimit = creditLimit + 5000.00
WHERE customerNumber=new.customerNumber;
END IF;
END IF;
END;
```

This trigger increases the credit Limit of a customer, if he/she has proven to be a valued customer and has issued another cheque for an amount > 10K.

# SQL Queries

## Correlated-nested query

### Query 01:

List the employees who report to those employees who report to Diane Murphy.

```
SELECT employeeNumber FROM employees WHERE reportsTo IN (SELECT
employeeNumber FROM employees WHERE reportsTo=(SELECT employeeNumber
FROM employees WHERE firstName='Diane' and lastName='Murphy'));
```

### Query 02:

Find the customer who spends the most.

```
SET @max:=(SELECT MAX(C.total_amt) FROM (SELECT customerNumber,
SUM(amount) AS total_amt FROM payments GROUP BY customerNumber) C);
SELECT C.customerNumber, C.amount FROM (SELECT customerNumber,
SUM(amount) as amount FROM payments GROUP BY customerNumber) C WHERE
amount = @max;
```

## Aggregate query

### Query 01:

Find out if each customer has an unique phone number

```
SET @dupli_num := (SELECT phone
FROM customers GROUP BY phone HAVING COUNT(phone) > 1);
SELECT customerNumber, contactLastName, contactFirstName, phone FROM
customers WHERE phone = @dupli_num;
```

# Outer-join query

## Query 01:

Find the total amount to be paid for an order by the customer.

```
SELECT C.customerNumber, O.orderNumber, SUM(OD.priceEach *
OD.quantityOrdered)
FROM customers C
LEFT OUTER JOIN orders O
      ON C.customerNumber = O.customerNumber
INNER JOIN orderdetails OD
      ON O.orderNumber = OD.orderNumber GROUP BY OD.orderNumber;
```

## Query 02:

What is the percentage value of each product in the inventory as a percentage of the quantity in stock for the product line to which it belongs?

```
SELECT P.productCode, P.quantityInStock * 100.00 / Q.total,
P.productLine
FROM products P
LEFT OUTER JOIN (SELECT SUM(quantityInStock) AS total, productLine
FROM products GROUP BY productLine) Q
      ON P.productLine = Q.productLine;
```

# Conclusion

This project aims at computerizing the manual process of a carSales storeroom.
This project is made for the company to keep a track of all the automobiles that were purchased or sold.

## Capabilities:

1. This database keeps tracks of the company’s purchases and sales.
2. It keeps tracks of the various suppliers and the prices at which they are selling the automobiles.
3. It keeps track of purchases made by the various globally distributed customers.
4. It keeps a record of the dates on which the order was received and shipped.
5. It keeps track of the product inventory and continuously updates it after every order is placed.

## Limitations:

1. This database contains no record of discounts, promos or advertising schemes that are usually provided by various companies on special occasions.

## Future enhancements:

1. The data stored on the database can be used for further data analysis that will help in creating a greater influx of profit for the company.
2. The data stored in this database can help the company keep track of the raising trends and invest accordingly.
3. The database contains information about a customer's previous buys and this can help in enhancing the company’s customer service.