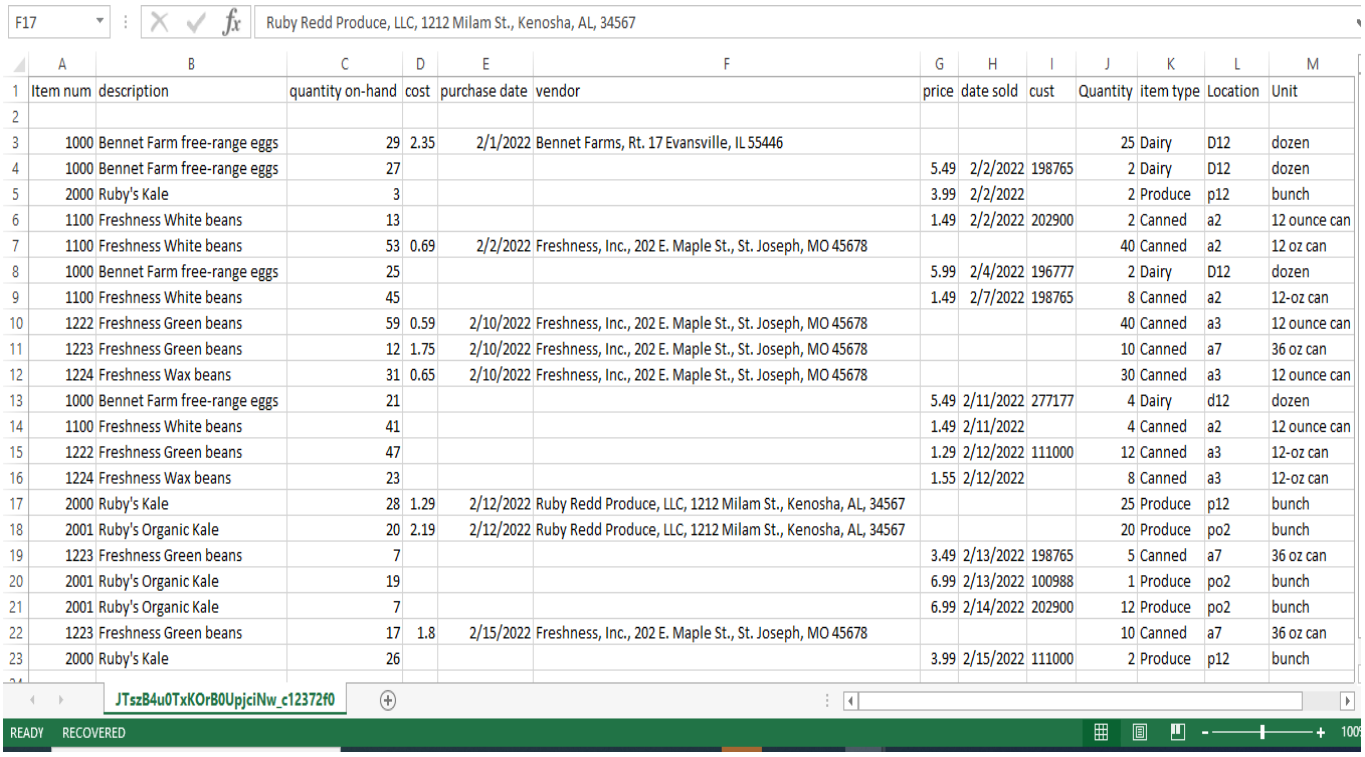
2023

DOCUMENTATION

EMMANUEL NGONGO

1. Data Analysis

When I was observing the spreadsheet, I noticed many rows had no values captured and there was redundancy within the data, in order to be able to reorganize the spreadsheet data into a database that supports an online grocery business as they expand their offerings, I proceeded to reorganize the data into relational tables using the modeling tool in MySQL workbench, but firstly I needed to find out entities that needed to be store and how will these entities be related in the database.



F17 | Ruby Redd Produce, LLC, 1212 Milam St., Kenosha, AL, 34567

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Item num	description	quantity on-hand	cost	purchase date	vendor	price	date sold	cust	Quantity	item type	Location	Unit
2													
3	1000	Bennet Farm free-range eggs	29	2.35	2/1/2022	Bennet Farms, Rt. 17 Evansville, IL 55446				25	Dairy	D12	dozen
4	1000	Bennet Farm free-range eggs	27				5.49	2/2/2022	198765	2	Dairy	D12	dozen
5	2000	Ruby's Kale	3				3.99	2/2/2022		2	Produce	p12	bunch
6	1100	Freshness White beans	13				1.49	2/2/2022	202900	2	Canned	a2	12 ounce can
7	1100	Freshness White beans	53	0.69	2/2/2022	Freshness, Inc., 202 E. Maple St., St. Joseph, MO 45678				40	Canned	a2	12 oz can
8	1000	Bennet Farm free-range eggs	25				5.99	2/4/2022	196777	2	Dairy	D12	dozen
9	1100	Freshness White beans	45				1.49	2/7/2022	198765	8	Canned	a2	12-oz can
10	1222	Freshness Green beans	59	0.59	2/10/2022	Freshness, Inc., 202 E. Maple St., St. Joseph, MO 45678				40	Canned	a3	12 ounce can
11	1223	Freshness Green beans	12	1.75	2/10/2022	Freshness, Inc., 202 E. Maple St., St. Joseph, MO 45678				10	Canned	a7	36 oz can
12	1224	Freshness Wax beans	31	0.65	2/10/2022	Freshness, Inc., 202 E. Maple St., St. Joseph, MO 45678				30	Canned	a3	12 ounce can
13	1000	Bennet Farm free-range eggs	21				5.49	2/11/2022	277177	4	Dairy	d12	dozen
14	1100	Freshness White beans	41				1.49	2/11/2022		4	Canned	a2	12 ounce can
15	1222	Freshness Green beans	47				1.29	2/12/2022	111000	12	Canned	a3	12-oz can
16	1224	Freshness Wax beans	23				1.55	2/12/2022		8	Canned	a3	12-oz can
17	2000	Ruby's Kale	28	1.29	2/12/2022	Ruby Redd Produce, LLC, 1212 Milam St., Kenosha, AL, 34567				25	Produce	p12	bunch
18	2001	Ruby's Organic Kale	20	2.19	2/12/2022	Ruby Redd Produce, LLC, 1212 Milam St., Kenosha, AL, 34567				20	Produce	po2	bunch
19	1223	Freshness Green beans	7				3.49	2/13/2022	198765	5	Canned	a7	36 oz can
20	2001	Ruby's Organic Kale	19				6.99	2/13/2022	100988	1	Produce	po2	bunch
21	2001	Ruby's Organic Kale	7				6.99	2/14/2022	202900	12	Produce	po2	bunch
22	1223	Freshness Green beans	17	1.8	2/15/2022	Freshness, Inc., 202 E. Maple St., St. Joseph, MO 45678				10	Canned	a7	36 oz can
23	2000	Ruby's Kale	26				3.99	2/15/2022	111000	2	Produce	p12	bunch

Figure 1.

From the image above, I was able to pick up **item, vendor, order, order_item, transfer, inventory, location, delivery, deliveryDetail, and customer** as entities, I added customer as an entity because in column **I** of the dataset, there is a need to store customers' information in the database which greatly influenced my choice on adding a customer as a table within the database. Afterwards, I needed to find out how these entities would be related within the database to create a scalable relational database which I managed to accomplish according to the following:

I. ITEM

Item is the main table or starting point for designing a database system, as I observed the spreadsheet we could notice some attributes that should be grouped for the item entity such as:

- ItemNum: This should be a unique number and the primary identifier (natural key) of the entity. I used an **INTEGER** type for this attribute.
- ItemDescription: A more detailed description of the item. I used **VARCHAR(2000)** datatype.
- ItemType: The item's type. I used **VARCHAR(100)** datatype.

II. LOCATION

This entity represents places where inventory is located. The online grocery store has many locations and each location has one or more inventory to facilitate the delivery of items to customers.

- LocationID: This attribute should be a unique ID number and the primary identifier (Surrogate primary key) of the entity. I used **INTEGER** datatype.
- LocationName: This attribute stores the name of the location. I used **VARCHAR(100)** datatype.
- LocationAddress: This attribute stores the full address, I used **VARCHAR(200)** datatype. The locationAddress could be broken into several attributes, for simplicity purposes I decided to leave it as it is.

III. VENDOR

Online grocery purchases items from vendors, a vendor table is needed to store information about vendors.

- VendorID: This attribute should be a unique ID and the primary identifier (surrogate primary key) of the entity. I used an **INTEGER** datatype.
- VendorName: The vendor's name. I used an **VARCHAR(100)** datatype
- VendorAddress: The vendor's address. I used **VARCHAR(200)** datatype. The vendor's address could be broken into several attributes but I kept it simple.
- Price: This attribute stores information about the price of of items. I used **DECIMAL(10,2)** datatype.

IV. ORDER & ORDER_ITEM

When online grocery purchase items from provider, they include information about the quantities that need to be delivered, the date sold. This information is store in the order and order_item entities:

ORDER

- OrderID: This attribute should be used as a unique ID number and surrogate primary key of the entity. I used an **INTEGER** datatype.
- Cost: This attribute stores information about the cost of procurement of items. I used **DECIMAL(10,2)** datatype.
- Date_sold: The date when items have been sold to the company. I used **DATE** datatype.
- Quantity: The amount of specific items ordered for the company. I used **INTEGER** datatype.
- VendorID: This attributes should be used as a **foreign key** to create a relationship between a vendor and orders, and assign orders to a vendor.

ORDER_ITEM

- Order_itemID: This attribute should be used as a unique ID number and surrogate primary key of the entity. I used an I**NTEGER** datatype.
- OrderID: This attribute should be used as a foreign key to create a relationship between the **ORDER** table and the **ORDER_ITEM** junction table.
- ItemNum: This attribute should be used as a foreign key to create a relationship between the **ITEM** table and the **ORDER_ITEM** junction table.
- ExpectedDate: The date when the item should arrive at the company's hand. I used **DATE** datatype.
- ActualDate: The date when the items were received by the company's hand. I used a **DATE** datatype.

V. INVENTORY AND TRANSFER

The inventory entity provides information on the quantity of available items. Each item may be stored in several inventories, and each inventory may contain many different items. This gave me a many-to-many relationship, to avoid multiple entries a junction table **TRANSFER** needed to be created to sustain the lowest level of normalization.

INVENTORY

- InventoryID: This attribute should be used as a unique ID number and the surrogate primary key. I used an **INTEGER** datatype.
- Quanty_on_hand: The quantity on hand of items. I used an **INTEGER** datatype.
- InvLocation: The location of items within the inventory. I used **VARCHAR(50)** datatype.
- Unit: The name of the unit where the item is categorized. I used an **VARCHAR(50)** datatype.
- LocationID: This attributes should be used as a **foreign key** to create a relationship between an inventory and physical locations.

TRANSFER

- TransferID: This attribute should be used as a unique ID number and the surrogate primary key. I used an **INTEGER** datatype.
- TransferQuantity: The amount of a specific item has been sent to the inventory. I used an **INTEGER** datatype.
- ReceivedDate: The date when the item arrived at the inventory. I used **DATE** datatype.
- ItemNum: This attribute should be used as a foreign key to create a relationship between the **ITEM** table and the **TRANSFER** junction table.
- InventoryID: This attribute should be used as a foreign key to create a relationship between the **INVENTORY** table and the **TRANSFER** junction table.

VI. CUSTOMER

Online grocery sells items to customers, so it is needed to store information about customers.

- CustomerID: This attribute should be used as a unique ID number and the surrogate primary key. I used an **INTEGER** datatype.
- CustNum: This attribute should be used as a unique number and the natural candidate key. I used an **INTEGER** datatype.
- CustFirstName: This attribute should store the customer's first name. I used a **VARCHAR(100)** datatype.
- CustLastName: This attribute should store the customer's last name. I used a **VARCHAR(100)** datatype.
- CustPhoneNum: This attribute should store the customer's phone number. I used an **INTEGER** datatype.
- CustAddress: The customer's full address. I used a **VARCHAR(200)** datatype.

VII. DELIVERY AND DELIVERYDETAIL

Once the online grocery sells items to customers, it may include different items from different inventory locations, depending on the item's availability and inventory's proximity based on the customer's address.

DELIVERY

- DeliveryID: This attribute should be used as a unique ID number and the surrogate primary key. I used an **INTEGER** datatype.
- CustomerID: This attribute should be used as a foreign key to create a relationship between the **CUSTOMER** table and the **DELIVERY** table.
- PurchaseDate: The date when a customer purchased items. I used a **DATE** datatype.

DELEVERYDETAIL

- DeliveryDetailID: This attribute should be used as a unique ID number and the surrogate primary key. I used an **INTEGER** datatype.
- DeliveryQuantity: The amount of items to be delivered to the customer. I used an **INTEGER** datatype.
- ExpectedDate: The date when the items should arrive at the customer's address. I used a **DATE** datatype.
- ActualDate: This date when the items were delivered. I used a **DATE** datatype.
- DeliveryID: This attribute should be used as a foreign key to create a relationship between the **DELIVERY** table and the **DELIVERYDETAIL** junction table.
- InventoryID: This attribute should be used as a foreign key to create a relationship between the **INVENTORY** table and the **DELIVERYDETAIL** junction table.

VIII. RELATIONSHIP BETWEEN ENTITIES

VENDOR AND ORDER

Each order is assigned to one vendor, not all vendors may have orders. A one to many (1:N) relationship needs to be established.

ORDER AND ORDERITEM

Each orderItem is assigned to one order, each order must have at least one or many orderItems. A one to many (1:N) relationship needs to be established.

ORDERITEM AND ITEM

Each orderItem must be associated with an item and each item may be included in one or many orders. A one-to-many relationship needs to be established.

CUSTOMER AND DELIVERY

Each delivery is assigned to a customer, each customer must have at least one order. A one to many (1:N) relationship needs to be established.

DELIVERY AND DELIVERYDETAIL

Each delivery must have at least one deliveryDetail, and each deliveryDetail belongs to one and only one delivery. This is a one to many relationship.

INVENTORY AND DELIVERYDETAIL

Each deliveryDetail is associated with an inventory, each inventory may have zero, one or many deliveryDetails. A one to many relationship needs to be established.

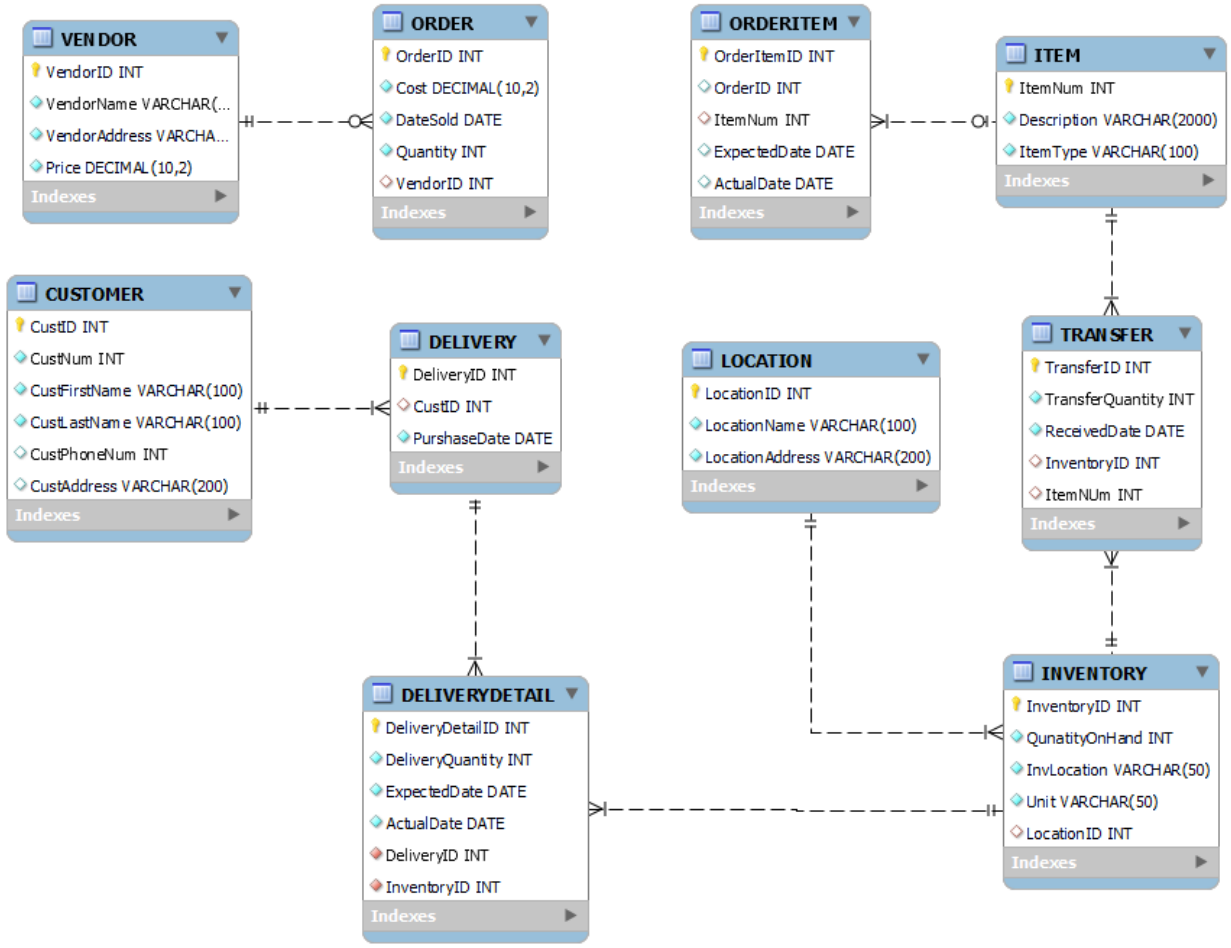
ITEM AND INVENTORY

Each item may be transferred to many inventory and each inventory may receive one or many items. A transfer table was created to resolve many to many relationship between ITEM and INVENTORY

LOCATION AND INVENTORY

Each location can have one or more inventories, an inventory must be assigned to one location. This is a one-to-many relationship.

IX. MODEL



P.S: This is my first time after a long time project based on databases so I am open to improvements and constructive critics.

X. SQL SCRIPT

-- 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_D
ATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITU
TION';


-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------


-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;

USE `mydb` ;


-- -----------------------------------------------------

-- Table `mydb`.`VENDOR`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`VENDOR` (

  `VendorID` INT NOT NULL AUTO_INCREMENT,

  `VendorName` VARCHAR(100) NOT NULL,

  `VendorAddress` VARCHAR(200) NOT NULL,

  `Price` DECIMAL(10,2) NOT NULL,

  PRIMARY KEY (`VendorID`))

ENGINE = InnoDB;



-- -----------------------------------------------------
```

```
-- Table `mydb`.`ORDER`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`ORDER` (
  `OrderID` INT NOT NULL AUTO_INCREMENT,
  `Cost` DECIMAL(10,2) NOT NULL,
  `DateSold` DATE NOT NULL,
  `Quantity` INT NOT NULL,
  `VendorID` INT NULL,
  PRIMARY KEY (`OrderID`),
  INDEX `OrderID` () VISIBLE,
  INDEX `VendorID_idx` (`VendorID` ASC) VISIBLE,
  CONSTRAINT `VendorID`
    FOREIGN KEY (`VendorID`)
    REFERENCES `mydb`.`VENDOR` (`VendorID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`ITEM`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`ITEM` (
  `ItemNum` INT NOT NULL AUTO_INCREMENT,
  `Description` VARCHAR(2000) NOT NULL,
  `ItemType` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`ItemNum`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`ORDERITEM`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`ORDERITEM` (
  `OrderItemID` INT NOT NULL AUTO_INCREMENT,
  `OrderID` INT NULL,
  `ItemNum` INT NULL,
  `ExpectedDate` DATE NULL,
  `ActualDate` DATE NULL,
  PRIMARY KEY (`OrderItemID`),
  INDEX `ItemNum_idx` (`ItemNum` ASC) VISIBLE,
  CONSTRAINT `ItemNum`
    FOREIGN KEY (`ItemNum`)
    REFERENCES `mydb`.`ITEM` (`ItemNum`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`LOCATION`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`LOCATION` (
  `LocationID` INT NOT NULL AUTO_INCREMENT,
  `LocationName` VARCHAR(100) NOT NULL,
  `LocationAddress` VARCHAR(200) NOT NULL,
```

```
  PRIMARY KEY (`LocationID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`INVENTORY`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`INVENTORY` (
  `InventoryID` INT NOT NULL AUTO_INCREMENT,
  `QunatityOnHand` INT NOT NULL,
  `InvLocation` VARCHAR(50) NOT NULL,
  `Unit` VARCHAR(50) NOT NULL,
  `LocationID` INT NULL,
  PRIMARY KEY (`InventoryID`),
  INDEX `LocationID_idx` (`LocationID` ASC) VISIBLE,
  CONSTRAINT `LocationID`
    FOREIGN KEY (`LocationID`)
    REFERENCES `mydb`.`LOCATION` (`LocationID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`TRANSFER`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`TRANSFER` (
  `TransferID` INT NOT NULL AUTO_INCREMENT,
```

```
  `TransferQuantity` INT NOT NULL,
  `ReceivedDate` DATE NOT NULL,
  `InventoryID` INT NULL,
  `ItemNUm` INT NULL,
  PRIMARY KEY (`TransferID`),
  INDEX `ItemNum_idx` (`ItemNUm` ASC) VISIBLE,
  INDEX `InventoryID_idx` (`InventoryID` ASC) VISIBLE,
  CONSTRAINT `ItemNum`
    FOREIGN KEY (`ItemNUm`)
    REFERENCES `mydb`.`ITEM` (`ItemNum`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `InventoryID`
    FOREIGN KEY (`InventoryID`)
    REFERENCES `mydb`.`INVENTORY` (`InventoryID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`CUSTOMER`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`CUSTOMER` (
  `CustID` INT NOT NULL AUTO_INCREMENT,
  `CustNum` INT NOT NULL,
  `CustFirstName` VARCHAR(100) NOT NULL,
  `CustLastName` VARCHAR(100) NOT NULL,
```

```
  `CustPhoneNum` INT NULL,
  `CustAddress` VARCHAR(200) NULL,
  PRIMARY KEY (`CustID`),
  UNIQUE INDEX `CustNum_UNIQUE` (`CustNum` ASC) VISIBLE,
  UNIQUE INDEX `CustPhoneNumb_UNIQUE` (`CustPhoneNum` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`DELIVERY`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`DELIVERY` (
  `DeliveryID` INT NOT NULL AUTO_INCREMENT,
  `CustID` INT NULL,
  `PurshaseDate` DATE NOT NULL,
  PRIMARY KEY (`DeliveryID`),
  INDEX `CustID_idx` (`CustID` ASC) VISIBLE,
  CONSTRAINT `CustID`
    FOREIGN KEY (`CustID`)
    REFERENCES `mydb`.`CUSTOMER` (`CustID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`DELIVERYDETAIL`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`DELIVERYDETAIL` (
  `DeliveryDetailID` INT NOT NULL AUTO_INCREMENT,
  `DeliveryQuantity` INT NOT NULL,
  `ExpectedDate` DATE NOT NULL,
  `ActualDate` DATE NOT NULL,
  `DeliveryID` INT NOT NULL,
  `InventoryID` INT NOT NULL,
  PRIMARY KEY (`DeliveryDetailID`),
  INDEX `DeliveryID_idx` (`DeliveryID` ASC) VISIBLE,
  INDEX `InventoryID_idx` (`InventoryID` ASC) VISIBLE,
  CONSTRAINT `DeliveryID`
    FOREIGN KEY (`DeliveryID`)
    REFERENCES `mydb`.`DELIVERY` (`DeliveryID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `InventoryID`
    FOREIGN KEY (`InventoryID`)
    REFERENCES `mydb`.`INVENTORY` (`InventoryID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```