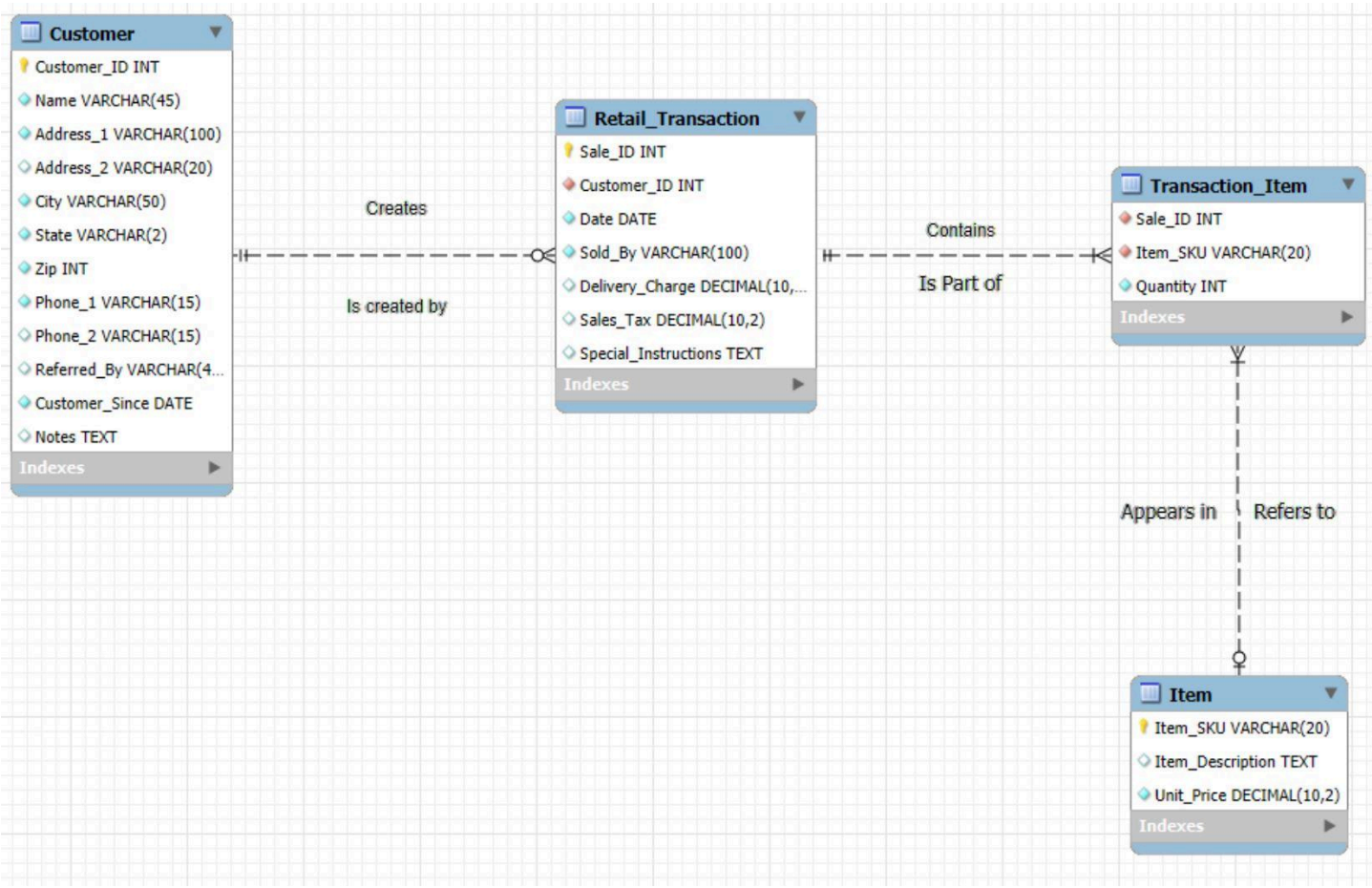
Ben Chen

Took me over 12 hours to complete this Lab because I had to redo my 3NF to make better sense. Then I had to build out the tables in phpMyAdmin. Then I had to setup mySQL Workbench environment by linking it to the CU Boulder server so I could access my progress.

Then, I worked on the data model + forward engineered the SQL code

Section 1: Data Model Screenshot



Self-note: Professor Knox said that Lab 2 is testing for Data Modeling relationship and not so much if individual columns were correct. So points docked in Lab 1 are independent of Lab 2 performance. Regardless, I tried to improve Lab 1 based on the Professor's feedback so that Lab 2 would go smoother.

I changed my 3NF table names to what they are because it was more intuitive than calling the tables "Customer", "Retail Sale", "Retail Sale Item", and "Item" (original naming convention and original tables shown below).

Unnormalized	1NF	2NF	3NF
CUSTOMER RECORD	Customers	Customers	Customers
Name	Customer ID (PK)	Customer ID (PK)	Customer ID (PK)
Address 1	Name	Name	Name
Address 2	Address 1	Address 1	Address 1
City, State, ZIP	Address 2	Address 2	Address 2
Mowing, Landscaping, Other	City	City	City
Phone 1	State	State	State
Phone 2	Zip	Zip	Zip
Referred By	Phone 1	Phone 1	Phone 1
Customer Since	Phone 2	Phone 2	Phone 2
Notes	Notes	Referred By	Referred By
	Mowing (Yes/No)	Customer Since	Customer Since
RETAIL SALES TICKET	Landscaping (Yes/No)	Notes	Notes
Date	Other (Yes/No)	Retail Sales	Retail Sales
Name	Referred By	Sale ID (PK)	Sale ID (PK)
Address	Customer Since	Name	Customer ID (FK -> Customers(Customer ID))
City, State, ZIP	Retail Sales	Address 1	Date
Phone	Sale ID (PK)	Address 2	Sold By
Sold By	Date	City	Delivery Charge
Quantity	Name	State	Sales Tax
Item SKU	Address 1	Zip	Special Instructions
Item Description	Address 2	Phone	Retail Sale Items (PK: Sale ID, Item SKU)
Unit Price	City	Date	Sale ID (FK -> Retail Sales(Sales ID))
Total Price	State	Sold By	Item SKU (FK -> Items(Item SKU))
Delivery Charge	Zip	Delivery Charge	Quantity
Sales Tax	Phone	Sales Tax	Items
Total Cost	Sold By	Total Cost	Item SKU (PK)
Special Instructions	Delivery Charge	Special Instructions	Item Description
	Sales Tax	Retail Sale Items (PK: Sale ID, Item SKU)	Unit Price
	Total Cost	Sale ID (FK -> Retail Sales(Sale ID))	
	Special Instructions	Item SKU (FK -> Items(Item SKU))	
	Retail Sale Items	Quantity	
	Sale ID (PK, FK -> Retail Sales(Sales ID))	Items	
	Item SKU (PK)	Item SKU (PK)	
	Item Description	Item Description	
	Quantity	Unit Price	
	Unit Price		
	Total Price		

Repeating Group

Sales tax varies

Section 2: DDL (SQL Code)

- Paste from workbench
- Must include CREATE statements for all tables in database (including definition of all data columns)

- PK and FK must be defined
- Include constraints including FK references (*can* remove “SET”, “Engine..”)

```
-- MySQL Script generated by MySQL Workbench
-- Wed Feb  5 04:44:36 2025
-- Model: New Model    Version: 1.0
-- 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 mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema bech1695
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema bech1695
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `bech1695` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `bech1695` ;

-- -----------------------------------------------------
-- Table `bech1695`.`Customer`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `bech1695`.`Customer` ;

CREATE TABLE IF NOT EXISTS `bech1695`.`Customer` (
  `Customer_ID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'Unique identifier
for customers\\r\\n\\r\\n',
  `Name` VARCHAR(45) NOT NULL COMMENT 'Allows full names up to 45 characters',
  `Address_1` VARCHAR(100) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci'
NOT NULL COMMENT 'First line of address',
  `Address_2` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci'
NULL DEFAULT NULL COMMENT 'Optional second line of address maybe for unit number or
apartment number',
```

```
  `City` VARCHAR(50) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL
COMMENT 'Customer city - accepts spaces between words as input',
  `State` VARCHAR(2) NOT NULL COMMENT 'Use 2-character state abbreviations (e.g.,
\"CA\")',
  `Zip` INT NOT NULL COMMENT 'Allows ZIP+4 (e.g., \"12345-6789\")',
  `Phone_1` VARCHAR(15) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT
NULL COMMENT 'Main phone number, largest international phone number is 15',
  `Phone_2` VARCHAR(15) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL
DEFAULT NULL COMMENT 'Second phone number',
  `Referred_By` VARCHAR(45) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci'
NULL DEFAULT NULL COMMENT 'Optional: who referred Customer',
  `Customer_Since` DATE NOT NULL COMMENT 'When did the customer start being a
customer? ',
  `Notes_About_Customer` TEXT CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci'
NULL DEFAULT NULL COMMENT 'Customer notes, note stay with customer ',
  PRIMARY KEY (`Customer_ID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci
COMMENT = 'Individual Customer';


-- -----------------------------------------------------
-- Table `bech1695`.`Item`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `bech1695`.`Item` ;

CREATE TABLE IF NOT EXISTS `bech1695`.`Item` (
  `Item_SKU` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT
NULL COMMENT 'Unique identifier for items',
  `Item_Description` TEXT CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL
DEFAULT NULL COMMENT 'Optional description of the item',
  `Unit_Price` DECIMAL(10,2) NOT NULL COMMENT 'Price per unit of the item for purchase',
  PRIMARY KEY (`Item_SKU`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci
COMMENT = 'Single Item';


-- -----------------------------------------------------
-- Table `bech1695`.`Retail_Transaction`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `bech1695`.`Retail_Transaction` ;
```

```
CREATE TABLE IF NOT EXISTS `bech1695`.`Retail_Transaction` (
  `Sale_ID` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'Unique identifier for each sale\\r\\n\\r\\nAutomatically increments for each new sale entry',
  `Customer_ID` INT UNSIGNED NOT NULL COMMENT 'References a specific customer from the Customer table\\r\\n\\r\\nLinks each sale to a customer who made the purchase',
  `Date_of_Transaction` DATE NOT NULL COMMENT 'Stores the date when the sale transaction occurred',
  `Sold_By` VARCHAR(100) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL COMMENT 'Represents the name or identifier of the employee who handled the sale\\r\\n\\r\\nUseful for tracking employee performance and accountability',
  `Delivery_Charge` DECIMAL(10,2) NULL DEFAULT NULL COMMENT 'Stores the delivery cost (if applicable) for the sale\\r\\n\\r\\nAllows NULL values for in-store purchases without delivery',
  `Sales_Tax` DECIMAL(10,2) NULL DEFAULT NULL COMMENT 'Stores the calculated sales tax for the transaction\\r\\n\\r\\nHelps maintain tax compliance and financial reporting',
  `Special_Instructions` TEXT CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NULL DEFAULT NULL COMMENT 'Captures any additional notes or special requests related to the sale',
  PRIMARY KEY (`Sale_ID`),
  INDEX `Customer ID` (`Customer_ID` ASC) VISIBLE,
  CONSTRAINT `FK_CustomerID`
    FOREIGN KEY (`Customer_ID`)
    REFERENCES `bech1695`.`Customer` (`Customer_ID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci
COMMENT = 'Single Sale Transaction Invoice';


-- -----------------------------------------------------
-- Table `bech1695`.`Transaction_Item`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `bech1695`.`Transaction_Item` ;

CREATE TABLE IF NOT EXISTS `bech1695`.`Transaction_Item` (
  `Sale_ID` INT UNSIGNED NOT NULL COMMENT 'References Retail Sale(Sale ID)',
  `Item_SKU` VARCHAR(20) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_0900_ai_ci' NOT NULL COMMENT 'References Item(Item SKU)',
  `Quantity` INT NOT NULL COMMENT 'The number of the item purchased',
  INDEX `Sale ID` (`Sale_ID` ASC) VISIBLE,
  INDEX `Item SKU` (`Item_SKU` ASC) VISIBLE,
```

```
  CONSTRAINT `FK_ItemSKU`
    FOREIGN KEY (`Item_SKU`)
    REFERENCES `bech1695`.`Item` (`Item_SKU`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `FK_SaleID`
    FOREIGN KEY (`Sale_ID`)
    REFERENCES `bech1695`.`Retail_Transaction` (`Sale_ID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci
COMMENT = 'Item Sold';


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

Section 3: Decisions & Explanations

- When deleting records, I set them to “CASCADE” because I wanted the records to be cleanly wiped when a customer is no longer a customer. This is just good data integrity.
    - This allows for data to be preserved consistently and make sure that child records are never left “dangling” or out of sync when parent records are altered, preventing orphan rows
    - On update, this allows for references to be accurate so that nothing breaks when something like a SKU changes is updated
- When a customer wants to update their data, I also want to CASCADE that so that the respective fields are updated accordingly.
- My data model is taking into consideration dashed/solid relationships
    - Dashed because there are foreign key relationships
    - Proper cardinality
        - Customer -> Retail_Transaction is 1 to many
            - One customer can have many Retail_Transaction but one transaction has exactly 1 customer
        - Retail_Transaction -> Transaction_Item is 1 to many
            - 1 Retail_Transaction can have many rows of Transaction_Item (meaning multiple items are bought), but each Transaction_Item links back to exactly 1 Retail_Transaction
        - Transaction_Item -> Item is many to 1

            - Each Trsnaction_Item references exactly 1 item, but not all Items are sold
            - The same SKU can be sold across multiple transactions
        - If we combine Retail_Transaction and Item, we can get a many-to-many link and this is not reflected in the data model because this relationship is resolved through Transaction_Item junction
        - Customer is related to Item, but there's no direct foreign key, so it does not show up in the data model.
    - Optionality
        - Customer may exist without making a transaction, but every transaction must have a customer, customers can be prospective and not necessarily buy anything (no circle on Customer, circle on Retail_Transaction)
        - Retail_Transaction must contain at least 1 Transaction_Item, and a Transaction_Item must belong to exactly 1 Retail_Transaction (no circle on both)
        - Transaction_Item must have an Item (no circle), but an Item does not need to necessarily be in a transaction (circle on Item)
    - Formatting (round edges because they are tables)
- One of the things that I made sure was that all the lengths of the columns and such made sense
    - VARCHAR is that length so that it gives people the freedom to write their first and last name (if last name is necessary)
    - Some columns are auto-incrementing so we can say "customer 1" vs. "customer 1001" and the number that is incrementing would be the customer ID we can refer to as a business (easy to search up, and intuitive)
    - Item_SKU with 20 alphanumeric digits is more than enough to represent billions of items uniquely - but the business won't ever need to exhaust this because it's a brick and mortar landscaping business. There *might* be a time where plants need to be sold in the future, and this database is supposed to be future proof for those types of scenarios where there might be nuances that make products more unique (like hybrid plant breeds that cannot be put singularly into one SKU).
    - I've let zip be 10 digits as integers so that they can have the freedom to add 5 digit zip codes additional digits for more defined location like a 9 digit zip
    - I've let phone number be 15 because international maximum phone number length is 15
- I've used Customer ID as a good candidate key to identify unique customers and I've let this be an unsigned auto incrementing integer based on the # of customers that exist, so we don't need to have excessively long ID numbers. I chose not to use Customer name because string-based keys actually take up more space/storage and slow down lookups compared to integers. Also there are a lot of "John Smith"s in the world.
    - "Unsigned" is so that these integers cannot be negative like "-1234" as an ID, same with item price and sale ID, etc.
- I've avoided VARCHAR for dates to avoid people's different interpretation of what a proper answer is like some might write, "Dec. 2025" others might want to write

"December 2025", and others might write "12/25/2025" (or even "Christmas" as a word) so I decided to keep it in "DATE" format allowing for faster queries

- My data model includes surrogate keys like "Customer" and "Retail_Transaction" and they use auto-incrementing primary keys
    - For both the Customer and Retail_Transaction tables, auto-incrementing IDs (Customer_ID and Sale_ID) serve as simple and unique identifiers
    - Customer_ID and Sale_ID provide a much smaller and more efficient storage mechanism that does not require dealing with the complexity of business-related fields
    - If a customer's name changes, or if you allow a customer to modify their email, the Customer_ID in the Retail_Transaction table stays the same
    - The primary key (Customer_ID) is unaffected by changes in other attributes (such as Name, Phone_1, Email), ensuring the integrity of foreign key relationships
- I've added detailed comments inside the data table itself so that we know exactly what each attribute means and stands for even if we change the column names