# DELITASTE PIZZA

Assignment 3

COMP1140

Jacob Boyce – c3264527
C3264526@uon.edu.au

# Contents

# Assignment 1 Revised

Below is a revision of assignment 1 based on the feedback received from the markers.

## Summary of Comments and Changes from Assignment 1

### Comments

- Stock-take Entity is irrelevant as the ingredient stock can be derived from the stock orders
- Although mode of order is valid it will cause some data redundancy and it suits the business rules better if it is simply a subclass of the relevant orders
- HumanAddress was a bit unconventional and unnecessary, as it is merely an attribute not an entity.
- Instore should be separated from employee as it is defined in the business rules as its own entity.
- Instore Pay should be separated from Payment Record. As they are paid by the number of hours worked in a shift. Number of deliveries is more relevant to the pay rather than the shift as they are paid on the number of deliveries.
- Many Attributes can have nulls

### Relevant Changes

- Removed ‘Stock Take’ Entity completely
- Removed ‘Mode of Order’ Entity and split it into the ‘Delivery’ and the ‘Pickup’ Entities. These entities are now inherited by the ‘Phone Order’ and the ‘Walk-In Order’ in accordance with the business rules
- Removed the ‘humanaddress’ entity and simply made it a composite attribute in all the relevant tables
- Made Instore an official sub entity of employee rather than having it be the employee
- Removed Driver Shift, Created the Driver Pay and Instore Pay Entities. Moved relevant information about the pay to said entities.
- Allowed nulls on several attributes

## Summary of Comments and Changes from Assignment 2

### Comments

- Payment entity is irrelevant as the payment is merely an attribute of the order. It does not have any independent defining characteristics causing it to be its own entity.
- Card payment is irrelevant for reasons above
- Payment Record is a salary and should not be mixed with the payment entity
- Employee structure is still not correct. Employees work shifts and are paid for said shifts. In a driver shift the number of deliveries is recorded and for an instore shift the number of hours worked in recorded. Payments are then handed out for x shifts. Drivers and Instore employees are paid differently for their respective shifts.
- Although the mapping was done correctly for the given EER the mapping was not correct to the correct EER and thus marks were deducted.
- Relationship table for inheritance should not be called ‘inheritance’ and rather 'generalisation’

### Relevant Changes

- Payment Record no longer inherits from payment. It is its own separate entity
- Payment has been merged into the orders entity
- Card Payment has been merged into the orders entity.
- Created the instore shift and the driver shift entities. These inherit from the 'shift' entity. Each payment record has many shifts, and each employee works many shifts.
- Changed relational mapping to the new EER.
- Changed 'inheritance' to 'generalisation' in the relationships table.

# Requirement Analysis

## Data Requirements

**Menu Item:**

Delitaste Pizza has several menu items offered to customers. Menu items have a unique code identifying them, size, name and selling price. Each menu item is composed of several ingredients and for each ingredient particular quantity of said ingredients for a menu item.

**Ingredient:**

Each Delitaste Pizza Menu Item is composed of ingredients. An ingredient has a unique code, name, description, type, suggested stock level and a suggested re-order level. An ingredient has a 'current stock level as of the last stocktake date'. To record the quantity of a given ingredient a stocktake is recorded once a week (i.e., all the actual stock of every ingredient in the physical store is recorded).

**Stock Order**

A stock-take occurs every week. In this event all the actual levels of ingredients in store and the date of the stocktake is recorded. A stocktake has a date, and many ingredients stock levels.

When stock levels are too low then new stock must be ordered, this is a stock order. Each stock order is kept and stored. This includes the order number, date of order, the date when the order was received, order status and total price. A stock order also contains for each of the ingredient ordered a quantity, price, description and ingredient code.

**Customer**

A customer is a person who uses Delitaste pizza. In the system a customer's phone number, name, address and whether they are or are not a hoax is recorded. A customer makes orders through Delitaste pizza.

**Order**

An order is made by a customer. An order is composed of menu items and for each menu item in an order there is a quantity. An order also has a total amount due, order status, price, staff id description, mode of reception and date. There are two different modes of making an order, one way is by phone and the other way is by walking in. There are 2 different modes of receiving an order, one being by pickup and the other being by delivery. An order also has a payment. If the payment type is via card, then the payment approval number is recorded.

**Mode of Order:**

This is how the order is taken.

Phone Order:

A phone order occurs when a customer calls Delitaste Pizza by phone to make an order. A phone order has a time when the call was made and terminated. (A phone order is a subclass of an order)

Walk-in order:

A walk-in order occurs when a customer walks into the physical store to make an order. A walk-in order records the time when the customer walked in. A walk-in order may only be received by pickup. (A walk-in order is a subclass of an order).

**Mode of Reception**

This is how the customer should receive the order.

Pickup order:

If an order is to be picked up by the customer in the store, then the pickup-time is recorded.
Delivery Order:

If an order is to be delivered to the customer at their desired address, then the delivery time, address and the driver to deliver the order is recorded.

**Payment:**

Each order has a payment. A payment is any kind of transaction that occurred in Delitaste Pizza where money is exchanged for menu items or for work. A payment consists of the amount due, and what type of payment was used (card, cash, visa, etc.) 6

Card Payment:

A card payment Is a payment which occurred through a debit or credit card. For each card payment that occurs the payment approval number is recorded. (This is a subclass of payment).

**Employee**

An employee is a person who provides work for Delitaste Pizza in exchange for money. An employee that is not a driver is a shop worker. Shop workers are paid for the hours worked. Each employee has a number (unique code), first name, last name, postal address, contact number, tax file number, payment rate, status and description. An employee also has a set of bank details.

The bank details include a bank code, bank name and account number.

Driver:

A driver is an employee at Delitaste Pizza who deliveries orders to customers. A driver delivers orders to customers. For each driver in addition to all the employee details kept their drivers license numbers are kept. A driver is paid for the number of deliveries they complete.

In-Store:

An instore employee is a person who works at Delitaste pizza and serves customer orders inside the shop. Each driver is paid on the number of hours they work in each shift.

**Shift**

Shifts are a pre-designated amount of time where an employee works for the store in exchange for money. Shifts are not regular and fixed beforehand, and employees are paid per shift. A shift has a start date, start time, end date, end time. A shift also has a payment record.

Driver Shift:

A driver shift is a shift that is completed by drivers. This records the number of deliveries done in each shift.

In Store Shift
An instore shift is completed by instore employees. This records the number of hours worked in the shop.

**Payment Record:**

A payment record is kept every time an employee is paid. This contains all the relevant details about payments to an employee's bank details. Employee payments include the gross payment, tax withheld, the total amount paid, payment date, payment start period, payment end date and the bank details of the employee. (This is a subclass of payment).

Driver Pay:

Drivers are paid on the number of orders they deliver in x shifts.

In-store pay:

In store employees are paid based on the amount of the number of hours they work in store on x shifts.

## Business Rules

- Drivers are paid for the number of deliveries they do, and shop workers are paid for the number of hours they complete.
- Shifts are not fixed or regular.
- Employees are paid per shift
- All employee payment records are kept.
- There are two distinct types of workers, shop workers and drivers.
- A stock-take is taken every week and is recorded in the system.
- All stock-orders are recorded in the system.
- Walk-in orders may not receive by delivery and only by pickup.
- Phone orders may be picked up or delivered.
- If a customer has not previously ordered at a store they must be recorded in the system and customers are uniquely identified by their phone number.
- All phone orders must be verified. If an order is not verified a customer is declared a hoax and the order status is changed to hold.

## Transactions:

**Data Manipulation Queries**

- Insert, Update and Delete and Employee
- Insert, update and delete a shift

- Insert and update or delete a payment record
- Insert update and delete a customer
- Insert update and delete an order
- Insert and update or delete a stock-take
- Insert update and delete a stock order
- Insert update and delete an ingredient
- Insert update and delete a menu item (Although this was not explicitly stated in the briefing if this could not be done then there would be no way to change the menu and thus if something went wrong, for example a menu item no longer complies with regulation then the business could not change this).
- Insert update or delete bank details
- Insert update or delete a payment
- Insert update or delete a payment record

**Queries**

- Search all information about an employee from name, contact number or id.
- Search all shifts of an employee from name, contact number or id.
- Search all shifts past a certain date (Although this transaction is not explicitly stated in the briefing if they could not do this the business would not have the ability to contact employees to work or have any kind of pre-planning for a shift).
- Search all payment records for an employee by name, contact number or id.
- Search the details of a customer by phone number
- Search a customer's order by their phone number and vice versa (Although this query is not explicitly stated in the briefing, if they could not do this, they would not this the verification process would be impossible, and it would be impossible to work out what order a custom has whether they walk in or where an order should be delivered.
- Search all orders by a given order status (Although this query is not explicitly stated in the briefing, if they could not do this then they would not know which orders to prepare, serve, deliver and to dial back for verification and thus the business could not fundamentally operate as according to the briefing).
- Search all information about an ingredient such as stock as of last stock-take date, suggested re-order levels, suggested stock levels, costs and re-order status (if it is being re-ordered and if so, how much is being reordered, the total cost of the re-order) from the description, name or unique code of an ingredient.
- Search all information about a menu item such a price, and ingredient composition from the name or unique code.

# Documentation of EER Model (Data Dictionary)

## Entity Types

| Entity Name | Description | Aliases | Occurrence |
|---|---|---|---|
| Menu Item | Any item that can be purchased by customers on the Delitaste Pizza Menu. Menu Items are identified by their code and casually by their name. | Pizza | In the store on the store menu. Items being cooked with ingredients. On the phone when customers make an order |
| Ingredient | Physical Ingredients which are used to make menu items. Ingredients can be identified by their code and casually by their name | | All ingredients are stored inside the store usually away from the customers view. |
| Stock Order | An event where ingredients (stock) is ordered and delivered to the business. The stock order is based on the suggested re-order quantities of the ingredients | | When stock is bought from external suppliers. |
| Order | An order is when an event where a customer purchases several items for money and then when the order is made out the customer can then receive their items. | | When the customer makes an order. |
| Customer | A person who purchases items at the shop. | | Any person who purchases item including people who walk in or order by phone. |
| Phone Order | An order which is taken by phone | Phone Order | When a customer orders something by phone |
| WalkInOrder | An order which is taken when a customer walks into the store | Walk In Order | When a customer enters the physical store and orders something. |
| Pickup | Represents an order that is being given to a customer by being picked up. That is the customer goes to the store to pick up the Pizza | Delivery Order, Pickup Order | When an order is delivered to a customer or picked up at the store by the customer |
| Delivery | When an order is delivered to the customer. A delivery has an address and a driver. Delivery receptions can only be made by phone orders. | Delivery | When an order is delivered to the customer |
| Employee | A person who works in the business providing useful work in exchange for money | Staff, Worker, Shop worker | Any person who works for Delitaste pizza (i.e provides work in exchange for money) |
| Driver | An employee who works for Delitaste Pizza that can drive so they can deliver orders to customer. Drivers earn money on | Delivery Person | Any employee who works for Delitaste Pizza that can drive. |

| | | | |
|---|---|---|---|
| | shifts based on the number of deliveries. | | |
| Instore | An employee who works for Delitaste Pizza who works instore and is paid on an hourly basis. | Instore Worker | Any employee who works for Delitaste Pizza who works in the shop. |
| Shift | A span of time across two date-times which an employee provides work. Shifts can be delegated to employees whenever they are needed | | When an employee turns up to work for a given amount of time. |
| DriverShift | A span of time across two dates in which a driver provides work. Each driver in each shift executes many deliveries. | Driver Worker Shift | When a driver is working for the shop and delivering orders |
| InStoreShift | A span of time across two dates in which a driver provides work. Each instore employee works in the shop and is paid for the amount of time they work | Shop Worker Shift | When an employee comes to the shop to work for a period of time. |
| PaymentRecord | A record that shows that an employee was paid. This includes other financial details about the payment such as the tax withheld and the bank details. | Payment Record, Employee Payment | When an employee is paid for the work that they do. |
| DriverPay | Drivers are paid on the number of deliveries they did in a shift. This is a payment record that includes this information | Driver Payment Record | When a driver is paid for the amount of deliveries, they did |
| InStorePay | InStore employees are paid on the number of hours they work. This is a payment record that includes this information | In Store Employee Payment Record | When an instore employee is paid for the number of hours they worked in a shift. |

## Relationships

| Entity Name | Multiplicity | Relationship | Multiplicity | Entity Name |
|---|---|---|---|---|
| StockOrder | 0..* | Orders | 0..* | Ingredients |
| MenuItem | 1..* | Contains | 1..* | Ingredients |
| Order | 0..* | Has | 1..* | MenuItem |
| Order | 0..* | Served by | 1..1 | Instore |
| Customer | 1..1 | Purchases | 0..* | Order |
| Order | 1..1 | Has | 1..1 | Payment |
| Delivery | 0..* | Delivered by | 1..1 | DeliveryShift |
| InStorePay | 1..1 | Paid for | 0..* | InstoreShift |
| DriverPay | 1..1 | Paid for | 0..* | DriverShift |
| Driver | 1..1 | Works | 0..* | Shift |
| Instore | 1..1 | Works | 0..* | InStoreShift |
| Driver | Mandatory, Or | Generalisation | Mandatory, Or | Employee |
| InStore | Mandatory, Or | Generalisation | Mandatory, Or | Employee |

| | | | | |
|---|---|---|---|---|
| DriverPay | Mandatory, Or | Generalisation | Mandatory, Or | Payment Record |
| InStorePay | Mandatory, Or | Generalisation | Mandatory, Or | Payment Record |
| WalkInOrder | Optional, Or | Generalisation | Mandatory, Or | Order |
| PhoneOrder | Optional, Or | Generalisation | Mandatory, Or | Order |
| Delivery | Mandatory, Or | Generalisation | Mandatory, Or | Phone Order |
| Pickup | Mandatory, Or | Generalisation | Mandatory, Or | Phone Order |
| PickUp | Mandatory, And | Generalisation | Mandatory, And | WalkInOrder |
| DeliveryShift | Mandatory, Or | Generalisation | Mandatory, Or | Shift |
| InstoreShift | Mandatory, Or | Generalisation | Mandatory, Or | Shift |

## Attributes

| Entity Name | Attributes | Description | Data Type & Length | Nulls | Multi-valued | Derived | Default |
|---|---|---|---|---|---|---|---|
| Ingredient | Ingredient Code | Unique code to identify an ingredient | Not enough information on the format of ingredient codes in the briefing! | No | No | No | None |
| | Name | Name of given ingredient | 30-character varchar | No | No | No | None |
| | Type | The type of ingredient | 30-character varchar | Yes | No | No | None |
| | Suggested Reorder Level | How much of an ingredient should be ordered in a stock order given it's below the suggested stock level | Positive integer | Yes | No | No | None |
| | Suggested Stock Level | How stock there should be of an ingredient after a stocktake | Positive Integer | Yes | No | No | None |
| | Quantity | How much of an ingredient there is in a store as of the last stock-take | Positive Integer | Yes | No | Yes, derived from the last stock order | |
| StockOrder | StockOrder Number | Unique number to identify a stock order | Not enough information on the format of a stock | No | No | No | None |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | order number to answer! | | | | |
| | Total cost | The total cost of a stock order | Integer | N/A | No | Yes, by summation of all of the prices of the ingredients in an order | None |
| | Date (Ordered) | The date when a stock order was made | Date (dd/mm/yyyy) | No | Yes | No | None |
| | Date (Received) | When an order is received | Date (dd/mm/yyyy) | Yes | Yes | No | None |
| | Status | The current status of a stock order i.e if it was purchased but not delivered, if it has been received etc | 2-character char (Status can be represented in a code) | No | No | No | PR (Purchased but not received) |
| MenuItem | Menu Item Code | A unique identifier for a menu item | Not enough information on the format of menu codes to answer! | No | No | No | None |
| | Name | The name of a menu item | 15-character varchar | No | No | No | None |
| | Size | The size of a menu item (L, M, S) etc. | 1-character char | Yes | No | No | None |
| | Selling Price | The current selling price of a menu item | Float point number, size 3 with 2 decimal places | Yes | No | No | None |
| Order | Order Number | A unique identifier for an order | Not enough information on the order number format to answer | No | No | No | None |
| | Description | A small description or other information | 150-character varchar | Yes | No | No | None |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | about an order | | | | | |
| | Status | The status of a given order, for example its being prepared, ready to serve, being verified etc. | 2-character char (Status can be represented in a code to save space). | No | No | No | PR (Purchased and being prepared) |
| | Date | The date when an order occurred | Date (dd/mm/yyyy) | No | No | No | None |
| | Total Price due | The total cost of a given order | Integer | Yes | No | Yes, derived from summing the cost of all menu items | None |
| | Payment Method | The way a payment was done | Varchar 15 character | No | No | Yes, Derived from the payment type of a payment | None |
| | Amount Due | The total amount of money spent in a payment | Floating point, size 5 with 2 decimal places | No | No | No | None |
| | Payment Type | A code to identify a type of payment | 1 character char | Yes | No | No | None |
| | Payment Approval Number | A unique code sent by a bank or financial institution to prove that a payment was received from a payment that occurs via card | Not enough information on the approval number format to answer | Yes | No | No | None |
| Walk-in Order | Walk-in Time | The time when a customer walked in the store to make an order | Time Format (This is usually a fraction of a day). | Yes | No | No | None |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Phone Order | Call Time | The time when a customer called the store | Time Format (as above) | Yes | No | No | None |
| | Termination Time | The time when a customer ended its call to the store | Time Format (As above) | Yes | No | No | None |
| | Call Duration | The amount of time a call lasted | Time format (as above) | Yes | No | Yes, derived from the difference in termination and call time | None |
| Customer | Customer ID | A unique code to identify a customer | Auto-incrementing integer | No | No | No | None |
| | Phone Number | The phone number of a customer | 15-character varchar, digits only | No | No | No | None |
| | Name (First) | The first name of a customer | 30-character varchar | Yes | Yes | No | None |
| | Name (Last) | The last name of a customer | 30-character varchar | Yes | Yes | No | None |
| | Is Hoax | Whether a customer is or is not declared a hoax (based on the verification procedure) | Boolean | Yes | No | No | None |
| | Address (Street) | A number which identifies which house an address is on a given street | 30 Character Varchar | Yes | Yes | No | None |
| | Address (City) | The city in which an address lies in | 30-character varchar | Yes | Yes | No | None |
| | Address (Postcode) | The postcode of an address | 4-character variable char | Yes | Yes | No | None |
| Employee | Employee Number | A unique number to | Not enough information on the format | No | No | No | None |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | identify an employee | of employee numbers to answer | | | | |
| | Name (First) | The first name of an employee | 30-character varchar | No | Yes | No | None |
| | Name (Last) | The last name of an employee | 30-character varchar | No | Yes | No | None |
| | Contact Number | The contact of the employee | 15-character varchar (digits only) | No | No | No | None |
| | Tax File Number | A unique number which the government uses to identify taxpayers | 9-character fixed char | No | No | No | None |
| | Description | A short description about an employee's role | 500-character variable char. | Yes | No | No | None |
| | BankDetails Account Number | The unique account number of a bank account. | 16-character varchar (digits only) | Yes | Yes | No | None |
| | BankDetails Bank Code | The Bank code that identifies a bank and its branch, also known as BSB | 6-character char, digits only | Yes | Yes | No | None |
| | BankDetails Bank name | The name of a bank | 50-character varchar | Yes | Yes | No | None |
| | Address (Street) | A number which identifies which house an address is on a given street | 30 Character Varchar | Yes | Yes | No | None |
| | Address (City) | The city in which an address lies in | 30-character varchar | Yes | Yes | No | None |
| | Address (Postcode) | The postcode of an address | 4-character variable char | Yes | Yes | No | None |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Shift | Shift ID | A unique ID identifying a shift | 8-character char | No | None | No | None |
| | Date-time(start) | The date in which the shift started | Date (dd/mm/yyyy) | Yes | Yes | No | None |
| | Date-time (end) | The date in which the shift ended | Date (dd/mm/yyyy) | Yes | Yes | No | None |
| Driver Shift | Number of Deliveries | The number of deliveries executed in a shift | Tiny Integer | Yes | No | Yes, from counting all the delivery relations | None |
| InStore Shift | Hours Worked | The number of hours worked in a shift | Tiny Integer | Yes | No | No | None |
| Payment Record | Total Payment | The gross amount on a payment without tax being removed. | 5-digit floating point number with 2 decimal places | No | No | No | None |
| | Payment Record ID | A unique number to identify an employee | Not enough information on the format of employee numbers to answer | No | No | No | None |
| | Date (Start) | The date when a payment begins | Date (dd/mm/yyyy) | No | No | No | None |
| | Date (End) | The date when a payment ends | Date (dd/mm/yyyy) | Yes | No | No | None |
| | Tax withheld | The amount of tax paid on a payment record | Floating point number 3 digits 2 decimal places | Yes | No | No | None |
| | Payment Period (Start) | The date from when a debt is incurred on the business | Date (dd/mm/yyyy) | Yes | No | No | None |
| | Payment Period (End) | The date due on a payment | Date (dd/mm/yyyy) | Yes | No | No | None |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Delivery | Delivery Time | The time when the delivery is to be received | Time | No | No | No | None |
| | Address (Street) | A number which identifies which house an address is on a given street | 30 Character Varchar | Yes | Yes | No | None |
| | Address (City) | The city in which an address lies in | 30-character varchar | Yes | Yes | No | None |
| | Address (Postcode) | The postcode of an address | 4-character variable char | Yes | Yes | No | None |
| Pickup | Pickup Time | The time when the order is to be picked up by the customer | Time | No | No | No | None |
| DriverPay | Number of deliveries Paid | The number of deliveries paid for in a record. This is what they are paid for | Tiny Integer | No | No | Yes, derived from the amount of deliveries in all of the shifts completed | None |
| InStorePay | Hours Worked Paid | The number of hours worked that were paid for. This is what they are paid for | Tiny Integer | No | No | | None |
| Driver Shift | Number Of Deliveries | The number of deliveries | Tiny Integer | No | No | Yes, derived from the amount of delivery relationships. | None |
| InStoreShift | Hours Worked | The amount of hours worked in a shift | Tiny Integer | Yes | No | No | None |

## EER Diagram

- Please find the **completed EER inside** of the **Assignments Folder**. The file is named ‘EER Diagram Revised.pdf’

## Relational Model

**StockOrder(**StockOrderNumber, DateOrdered, DateReceived, Status)
**Primary Key** StockOrderNumber

**Ingredient(**IngredientCode, Name, Description, Type, SuggestedReorderLevel, SuggestedStockLevel)
**Primary Key** IngredientCode

Note to marker – this is from the many to many relationship between the ingredients and the stockorder

**IngredientOrders(**StockOrderNumber, IngredientCode, Quantity, Price)

**Primary Key** StockOrderNumber, IngredientCode
**Foreign Key** StockOrderNumber **references** StockOrder(StockOrderNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** IngredientCode **references** Ingredient(IngredientCode) ON UPDATE CASCADE ON DELETE SET NULL

**MenuItem(**MenuItemCode, Name, Size, SellingPrice)
**Primary Key** MenuItemCode

**MenuItemComposition(**MenuItemCode, IngredientCode, Quantity)
**Primary Key** MenuItemCode, IngredientCode
**Foreign Key** MenuItemCode **References** MenuItem(MenuItemCode) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** IngredientCode **references** Ingredient(IngredientCode) ON UPDATE CASCADE ON DELETE CASCADE

**Order(**OrderNumber, Description, OrderStatus, Date, CustomerID,EmployeeNumber, Amount Due, PaymentMethod, PaymentApprovalNumber)
**Primary Key** OrderNumber
**Foreign Key** CustomerID **References** Customer(CustomerID) ON UPDATE CASCADE ON DELETE SET NULL
**Foreign Key** EmployeeNumber **References** Instore(EmployeeNumber) ON UPDATE CASCADE ON DELETE SET NULL

**WalkInOrder**(OrderNumber, WalkInTime, PickupTime)
**Primary Key** OrderNumber
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE NO ACTION ON DELETE CASCADE

**PhoneDeliveryOrder(**OrderNumber, CallTime, TerminationTime, DeliveryTime, AddressStreet, AddressCity, AddressPostcode, Driver)
**Primary Key** OrderNumber
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE NO ACTION ON DELETE CASCADE
**Foreign Key** Driver **references** DriverShift(ShiftID) ON UPDATE CASCADE ON DELETE SET NULL

**PhonePickupOrder(**OrderNumber, CallTime, TerminationTime, PickupTime)
**Primary Key** OrderNumber
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE NO ACTION ON DELETE CASCADE

**OrderItems(**OrderNumber, MenuItemCode, Quantity)
**Primary Key** OrderNumber, MenuItemCode
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** MenuItemCode **References** MenuItem(MenuItemCode) ON UPDATE CASCADE ON DELETE SET NULL

**Customer(**CustomerID, PhoneNumber, FirstName, LastName, isHoax, AddressStreet, AddressCity, AddressPostcode)
**Primary Key** CustomerID
**Alternate Key** Phone Number

**Instore(**EmployeeNumber, FirstName, LastName, ContactNumber, TaxFileNumber, Description, HourlyRate, AddressStreet, AddressCity, AddressPostcode, AccountNumber, BankCode, BankName**)**
**Primary Key** EmployeeNumber
**Alternate Key** TaxFileNumber

**Driver(**EmployeeNumber, FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, AddressPostcode, AccountNumber, BankCode, BankName, DriversLicenseNumber)
**Primary Key** EmployeeNumber
**Alternate Key** TaxFileNumber
**Alternate Key** DriversLicenseNumber

**DriverShift(**ShiftID, Date Start, Time Start, Time End, Date End, EmployeeNumber, PaymentID)
**Primary Key** ShiftID
**Foreign Key** EmployeeNumber **References** Driver(EmployeeNumber) ON UPDATE NO ACTION ON DELETE NO ACTION
**Foreign Key** PaymentID **References** DriverPay**(**PaymentID) ON UPDATE NO ACTION ON DELETE NO ACTION

**InstoreShift(**ShiftID, Date Start, Time Start, Time End, Date End, EmployeeNumber, PaymentID)
**Primary Key** ShiftID
**Foreign Key** EmployeeNumber **References** Instore(EmployeeNumber) ON UPDATE NO ACTION ON DELETE NO ACTION
**Foreign Key** PaymentID **References** InStorePay**(**PaymentID) ON UPDATE NO ACTION ON DELETE NO ACTION

**DriverPay**(PaymentID, EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, PaymentPeriodEnd, TotalPaid, NumberOfDeliveries)
**Primary Key** PaymentID

**InStorePay**(PaymentID, EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, PaymentPeriodEnd, TotalPaid, HoursWorked)
**Primary Key** PaymentID

# Normalisation Proccess

## First Normal Form

First check the schema is in first normal form (1NF). A relational schema is in 1NF if and only if

- There are only single valued attributes
- The attribute domain does not change
- Each Attribute has a unique name
- The order of data entry does not matter.

These rules apply to every entity in the database schema (by observation of the data dictionary and the relational model schema.

## Second Normal Form

Check the schema is in second normal form(2NF). A relational schema is in 2NF if and only if every non-prime attribute is fully functionally dependent on the primary key and the table is in 1NF. That is only the full primary key functionally determines all attributes in an entity and not a proper subset.

Check all functional dependencies for partial dependencies. The functional dependency set in the table below is where X -> Y where X is prime or a candidate key or a proper subset of the prime key and Y is non-prime and the dependency is not trivial.

| Entity Name | Functional Dependencies (prime to non-prime attributes only) | Reasoning (If applicable) | Entity in 2NF? |
|---|---|---|---|
| StockOrder | StockOrderNumber -> DateOrdered, DateReceived, Status | | Yes |
| Ingredient | IngredientCode -> name, description, type, suggested | | Yes |
| IngredientOrders | StockOrderNumber, IngredientCode -> Quantity, Price | Quantity and price are determined only by both keys because the quantity and price are for a particular ingredient on a given stock order | Yes |
| MenuItem | MenuItemCode -> name, size, selling price | | Yes |
| Menu Item Composition | MenuItemCode, IngredientCode -> Quantity | MenuItemCode and IngredientCode fully determine the quantity as the quantity is for a given ingredient on a particular menu item. | Yes |
| Order | OrderNumber -> description, orderstatus, date, customerid, employeenumber, PaymentMethod, TotalAmountDue, PaymentApprovalNumber | | Yes |
| WalkInOrder | OrderNumber -> walkintime, pickuptime | | Yes |
| PhoneDelivery Order | OrderNumber -> calltime, terminationtime, deliverytime, address street, address city, address postcode | | Yes |
| PhonePickup Order | OrderNumber -> CallTime, TerminationTime, PickupTime | | Yes |
| OrderItems | OrderNumber, MenuItemCode, -> Quantity | OrderNumber and menuitemcode fully determine quantity as the quantity is for a particular menu item for a given order | Yes |
| Customer | CustomerID -> Phone Number, First Name, Last Name, is Hoax, AddressStreet, AddressCity, AddressPostcode<br><br>PhoneNumber -> First Name, Last Name, is hoax, Address Street, Address City, Address Postcode | | Yes |

| | | | |
|---|---|---|---|
| | CustomerID, PhoneNumber -> First Name, Last Name, is hoax, Address Street, Address City, Address Postcode | | |
| InStore | EmployeeNumber -> FirstName, LastName, ContactNumber, TaxFileNumber, Description, HourlyRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name<br><br>TaxFileNumber -> FirstName, LastName, ContactNumber, Description, Hourly, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name | None of the attributes are functionally dependent on any proper subset of all 4 candidate keys (from business rules). All 3 candidate keys were identified as either prime or an alternate key. | Yes |
| Driver | EmployeeNumber -> FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name, DriversLicenseNumber<br><br>TaxFileNumber -> FirstName, LastName, ContactNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name, Drivers License Number<br><br>DriversLicenseNumber -> FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name | None of the attributes are functionally dependent on any proper subset of all 4 candidate keys (from business rules). All 4 candidate keys were identified as either prime or an alternate key. | Yes |
| DriverShift | ShiftID -> EmployeeNumber, Date Start, Date End, Time Start, Time End, PaymentID | | Yes |
| InStoreShift | ShiftID -> EmployeeNumber, Date Start, Date End, Time Start, Time End, PaymentID | | Yes |
| DriverPay | PaymentID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDatened, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Shift ID, NumberOfDeliveriesPaid | | Yes |
| InStorePay | PaymentID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDatened, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Shift ID, Hours WorkedPaid | | Yes |

As there are no partial dependencies present and the schema is in 1NF the schema is also in 2NF.

## Third Normal Form

Check the relational schema is in third normal form (3NF). A relational schema is in 3NF if and only if the schema is in 2NF and there are no transitive dependencies that is all non-prime attributes are solely and only dependent on the primary key and not on any non-prime attribute.

Check all the entities are in 3NF. The Functional dependencies F in the table is that where X -> Y where X is prime or non-prime, Y is non-prime, and the dependency is not trivial.

| Entity Name | Functional Dependencies F (prime to non-prime, and non-prime to non-prime) | Reasoning (If applicable) | Entity in BCNF? |
|---|---|---|---|
| StockOrder | StockOrderNumber -> DateOrdered, DateReceived, Status | Date Ordered, Date Received and Status are only determined by the given stock order. Status cannot be derived from the date. | Yes |
| Ingredient | IngredientCode -> name, description, type, suggested reorder level, suggested stock level | All attributes are only determined by the given ingredient. | Yes |
| IngredientOrders | StockOrderNumber, IngredientCode -> Quantity, Price | Quantity and price are determined only by both keys because the quantity and price are for a particular ingredient on a given stock order | Yes |
| MenuItem | MenuItemCode -> name, size, selling price | All attributes are only determined by the given menu item. Although size may affect the selling price no meaningful business rule is given to formally establish this. | Yes |
| Menu Item Composition | MenuItemCode, IngredientCode -> Quantity | MenuItemCode and IngredientCode fully determine the quantity as the quantity is for a given ingredient on a particular menu item. | Yes |
| Order | OrderNumber -> description, orderstatus, date, customerid, employeenumber, PaymentMethod, TotalAmountDue, PaymentApprovalNumber | Although a payment approval number is only recorded when the payment method is via card this is not a functional dependency as the value of the payment approval number cannot be determined from the payment method. | Yes |
| WalkInOrder | OrderNumber -> walkintime, pickuptime | All attributes are only dependent on the order number | Yes |

| | | | |
|---|---|---|---|
| PhoneDelivery Order | OrderNumber -> calltime, terminationtime, deliverytime, address street, address city, address postcode | Address is not determined by the customer(super) as the customer does not necessarily have to deliver at their home. | Yes |
| PhonePickup Order | OrderNumber -> CallTime, TerminationTime, PickupTime | All attributes are fully dependent on order number | Yes |
| OrderItems | OrderNumber, MenuItemCode, -> Quantity | OrderNumber and menuitemcode fully determine quantity as the quantity is for a particular menu item for a given order | Yes |
| Customer | CustomerID -> Phone Number, First Name, Last Name, is Hoax, AddressStreet, AddressCity, AddressPostcode<br><br>PhoneNumber -> First Name, Last Name, is hoax, Address Street, Address City, Address Postcode | Phone Number identifies all attributes in the entity however it was nominated as an alternate key. | Yes |
| InStore | EmployeeNumber -> FirstName, LastName, ConactNumber, TaxFileNumber, Description, HourlyRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name<br><br>TaxFileNumber -> FirstName, LastName, ContactNumber, Description, HourlyRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name | Contact Number and Tax File number also uniquely identify all fields, but they have been nominated as alternate keys. | Yes |
| Driver | EmployeeNumber -> FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name, DriversLicenseNumber<br><br>TaxFileNumber -> FirstName, LastName, ContactNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name, Drivers License Number<br><br>DriversLicenseNumber -> FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name | Contact Number, Tax File Number and Drivers License Number all uniquely identify all other fields however they have already been nominated as alternate keys. | Yes |
| DriverShift | ShiftID -> EmployeeNumber, Date Start, Date End, Time Start, Time End, PaymentID | Date start, time start and employee number cant determine the key as shifts are not fixed or scheduled in any kind of | Yes |

| | | | |
|---|---|---|---|
| | | way, that is there is no business rule that explicitly specifies there cant be two shifts at the same time. Thus all attributes are determined by the shiftID. | |
| InStoreShift | ShiftID -> EmployeeNumber, Date Start, Date End, Time Start, Time End, PaymentID | Date start, time start and employee number cant determine the key as shifts are not fixed or scheduled in any kind of way, that is there is no business rule that explicitly specifies there cant be two shifts at the same time. Thus all attributes are determined by the shiftID. | Yes |
| DriverPay | PaymentID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Shift ID, NumberOfDeliveries<br><br>ShiftID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDatened, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, NumberOfDeliveries | As each payment is for a shift, the shift must be unique for each payment and thus a payment is functionally dependent on a shift. However, this has been nominated as an alternate key. | Yes |
| InStorePay | PaymentID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDatened, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Shift ID, Hours Worked<br><br>ShiftID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Hours Worked | As each payment is for a shift, the shift must be unique for each payment and thus a payment is functionally dependent on a shift. However, this has been nominated as an alternate key. | Yes |

As shown the table is already in 3NF as all non-prime attributes are solely functionally dependent on the primary key.

## Boyce Codd Normal Form

Check that the relational schema is in Boyce Codd Normal Form (BCNF). A schema is in BCNF is and only if for all functional dependencies X -> Y

- The dependency is trivial
- X is a superkey in R

Check all the entities are in BCNF. The Functional dependencies F in the table is that where X -> Y where X -> Y is not trivial.

| Entity Name | Functional Dependencies F (prime to non-prime, and non-prime to non-prime) | Reasoning (If applicable) | Entity in BCNF? |
|---|---|---|---|
| StockOrder | StockOrderNumber -> DateOrdered, DateReceived, Status | Date Ordered, Date Received and Status are only determined by the given stock order. Status cannot be derived from the date. | Yes |
| Ingredient | IngredientCode -> name, description, type, suggested reorder level, suggested stock level | All attribtes are only determined by the given ingredient. | Yes |
| IngredientOrders | StockOrderNumber, IngredientCode -> Quantity, Price | Quantity and price are determined only by both keys because the quantity and price are for a particular ingredient on a given stock order | Yes |
| MenuItem | MenuItemCode -> name, size, selling price | All attributes are only determined by the given menu item. Although size may affect the selling price no meaningful business rule is given to formally establish this. | Yes |
| Menu Item Composition | MenuItemCode, IngredientCode -> Quantity | MenuItemCode and IngredientCode fully determine the quantity as the quantity is for a given ingredient on a particular menu item. | Yes |
| Order | OrderNumber -> description, orderstatus, date, customerid, employeenumber, PaymentMethod, TotalAmountDue, PaymentApprovalNumber | Although a payment approval number is only recorded when the payment method is via card this is not a functional dependency as the value of the payment approval number cannot be determined from the payment method. | Yes |
| WalkInOrder | OrderNumber -> walkintime, pickuptime | All attributes are only dependent on the order number | Yes |

| | | | |
|---|---|---|---|
| PhoneDelivery Order | OrderNumber -> calltime, terminationtime, deliverytime, address street, address city, address postcode | Address is not determined by the customer(super) as the customer does not necessarily have to deliver at their home. | Yes |
| PhonePickup Order | OrderNumber -> CallTime, TerminationTime, PickupTime | All attributes are fully dependent on order number | Yes |
| OrderItems | OrderNumber, MenuItemCode, -> Quantity | OrderNumber and menuitemcode fully determine quantity as the quantity is for a particular menu item for a given order | Yes |
| Customer | CustomerID -> Phone Number, First Name, Last Name, is Hoax, AddressStreet, AddressCity, AddressPostcode<br><br>PhoneNumber -> First Name, Last Name, is hoax, Address Street, Address City, Address Postcode | Phone Number identifies all attributes in the entity however it was nominated as an alternate key. | Yes |
| InStore | EmployeeNumber -> FirstName, LastName, ConactNumber, TaxFileNumber, Description, HourlyRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name<br><br>TaxFileNumber -> FirstName, LastName, ContactNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name | Tax File number also uniquely identify all fields, but they have been nominated as alternate keys. | Yes |
| Driver | EmployeeNumber -> FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name, DriversLicenseNumber<br><br>TaxFileNumber -> FirstName, LastName, ContactNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name, Drivers License Number<br><br>DriversLicenseNumber -> FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, Address Street, Address City, Address Postcode, Account Number, Bank Code, Bank Name | Tax File Number and Drivers License Number all uniquely identify all other fields however they have already been nominated as alternate keys. | Yes |
| DriverShift | ShiftID -> EmployeeNumber, Date Start, Date End, Time Start, Time End, PaymentID | Date start, time start and employee number cant determine the key | DriverShift |

| | | | |
|---|---|---|---|
| | | as shifts are not fixed or scheduled in any kind of way, that is there is no business rule that explicitly specifies there cant be two shifts at the same time. Thus all attributes are determined by the shiftID. | |
| InStoreShift | ShiftID -> EmployeeNumber, Date Start, Date End, Time Start, Time End, PaymentID | Date start, time start and employee number cant determine the key as shifts are not fixed or scheduled in any kind of way, that is there is no business rule that explicitly specifies there cant be two shifts at the same time. Thus all attributes are determined by the shiftID. | InStoreShift |
| DriverPay | PaymentID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Shift ID, NumberOfDeliveries<br><br>ShiftID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDatened, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, NumberOfDeliveries | As each payment is for a shift, the shift must be unique for each payment and thus a payment is functionally dependent on a shift. However, this has been nominated as an alternate key. | Yes |
| InStorePay | PaymentID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDatened, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Shift ID, Hours Worked<br><br>ShiftID -> EmployeeNumber, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, Payment Period End, Amount Due, Payment Type, Hours Worked | As each payment is for a shift, the shift must be unique for each payment and thus a payment is functionally dependent on a shift. However, this has been nominated as an alternate key. | Yes |

The relational schema is already in BCNF. As all entities meet the conditions.

## Normalisation Process.

The schema is already in BCNF. As specified in the Assignment 2 Q & A additional functional dependencies will be assumed for two tables.

Example 1, Normalising the Driver Table

**Driver(**EmployeeNumber, DriversLicenseNumber ,TaxFileNumber, FirstName, LastName, ContactNumber, Description, PaymentRate, AddressStreet, AddressCity, AddressPostcode, AccountNumber, BankCode, BankName)
**Primary Key** EmployeeNumber, DriversLicenseNumber

Assume the following functional dependencies

EmployeeNumber, DriversLicenseNumber, TaxFileNumber -> FirstName, LastName, ContactNumber, Description, PaymentRate, Address Street, Address City, AddressPostcode, Account Number, BankCode, BankName

TaxFileNumber -> AccountNumber, BankCode, BankName

Description -> PaymentRate

EmployeeNumber -> FirstName, LastName, ContactNumber

**Normalising**

Entity is in 1NF as there are only single valued attributes, the attribute domain is constant, the data entry order is irrelevant and each attribute is atomic.

Entity is not in 2NF as

- EmployeeNumber -> FirstName, LastName, ContactNumber
- EmployeeNumber is a candidate key

Perform Lossless Join Decomposition

Let R = EmployeeNumber, DriversLicenseNumber, TaxFileNumber, FirstName, LastName, ContactNumber, Description, paymentRate, AddressStreet, Address City, Address Postcode, Account Number, BankCode, Bank Name

Let X = Employee Number
Let Y = FirstName, LastName, ContactNumber

Thus R-Y = EmployeeNumber, DriversLicenseNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, AddressPostcode, AccountNumber, BankCode, BankName

XY = EmployeeNumber, FirstName, LastName, ContactNumber

This leaves with the two tables

**R-Y = Driver(**EmployeeNumber, DriversLicenseNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, Address Postcode, AccountNumber, BankCode, BankName)
**Primary Key** EmployeeNumber, DriversLicenseNumber, TaxFileNumber
**Foreign Key** EmployeeNumber **references** Employee(EmployeeNumber)

**XY = Employee**(EmployeeNumber, FirstName, LastName, ContactNumber)
**Primary Key** EmployeeNumber

Table is still not in 2NF as partial dependencies still exist

- TaxFileNumber -> AccountNumber, BankCode, BankName

Perform Lossless Join Decomposition

Let X = TaxFileNumber
Let Y = AccountNumber, BankCode, BankName
Let R = EmployeeNumber, DriversLicenseNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, Address Postcode, AccountNumber, BankCode, BankName

R-Y = EmployeeNumber, DriversLicenseNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, Address Postcode

XY= TaxFilenumber, AccountNumber, BankCode, BankName

This leaves with the three tables

**R-Y = Driver(**EmployeeNumber, DriversLicenseNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, Address Postcod)
**Primary Key** EmployeeNumber, DriversLicenseNumber, TaxFileNumber
**Foreign Key** EmployeeNumber **references** Employee(EmployeeNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** TaxFileNumber **references** AccountInfo(TaxFileNumber) ON UPDATE CASCADE ON DELETE CASCADE

**XY = AccountInfo(**TaxFileNumber, AccountNumber, BankCode, BankName)
**Primary Key** TaxFileNumber

**Employee**(EmployeeNumber, FirstName, LastName, ContactNumber)
**Primary Key** EmployeeNumber

All 3 entities are now in 2NF as all attributes are functionally dependent on only the primary key and not a proper subset of the primary key.

Entity is not in 3NF as transitive dependencies exist

Description -> PaymentRate

Perform Lossless Join Decomposition

Let X = Description
Let Y = PaymentRate
R = EmployeeNumber, DriversLicenseNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, Address Postcode, AccountNumber, BankCode, BankName

R-Y = EmployeeNumber, DriversLicenseNumber, TaxFileNumber, AddressStreet, AddressCity, Address Postcode
XY = Description, PaymentRate

This means there will be 4 tables

**R-Y = Driver(**EmployeeNumber, DriversLicenseNumber, TaxFileNumber, AddressStreet, AddressCity, Address Postcode, Description)
**Primary Key** EmployeeNumber, DriversLicenseNumber, TaxFileNumber
**Foreign Key** EmployeeNumber **references** Employee(EmployeeNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** TaxFileNumber **references** AccountInfo(TaxFileNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** Description **references** PaymentRates(Description) ON UPDATE CASCADE ON DELETE SET NULL

**AccountInfo(**TaxFileNumber, AccountNumber, BankCode, BankName)
**Primary Key** TaxFileNumber

**Employee**(EmployeeNumber, FirstName, LastName, ContactNumber)
**Primary Key** EmployeeNumber

**PaymentRates**(Description, PaymentRate)
**Primary Key** Description

The table is now fully in 3NF as there are no transitive dependencies across non-keyed attributes.

The Table is fully in BCNF as there are no transitive dependencies at all.

Example 2, Normalising the order table

**Order(**OrderNumber, Description, OrderStatus, Date, CustomerID,EmployeeNumber, PaymentID)

Assume the following functional dependencies exist

OrderNumber, CustomerID -> PaymentID, Date, OrderStatus, EmployeeNumber, CustomerID

OrderStatus -> CustomerID.

Entity is in 1NF as there are only single valued attributes, the attribute domain is constant, the data entry order is irrelevant and each attribute is atomic.

Entity is in 2NF as there are no partial dependencies in the functional dependency set F. That is a proper subset of the candidate key does not functionally depend on any non-prime attribute.

Entity is in 3NF. There are no transitive dependencies from the candidate keys to the non-prime attributes.

Entity is not in BCNF, a transitive dependency exists where OrderStatus is not a candidate key

- OrderStatus -> CustomerID

Perform lossless join decomposition.

Let X = OrderStatus
Let Y = CustomerID
Let R = OrderNumber, CustomerID, Description, orderStatus, Date, EmployeeNumber, PaymentID

R-Y = OrderNumber, Description, OrderStatus, Date, EmployeeNumber, PaymentID
XY = OrderStatus, CustomerID

R-Y = **Order**(OrderNumber, OrderStatus, Date, EmployeeNumber, PaymentID)
**Primary Key** OrderNumber
**Foreign Key** OrderStatus **References** statusCustomer(OrderStatus) ON UPDATE CASCADE ON DELETE NO ACTION.

XY = **statusCustomer(**OrderStatus, CustomerID)
**Primary Key** OrderStatus

## Final Relational Schema in BCNF

Note this is identical to the original relational schema as it was already in BCNF.

**StockOrder(**StockOrderNumber, DateOrdered, DateReceived, Status)
**Primary Key** StockOrderNumber

**Ingredient(**IngredientCode, Name, Description, Type, SuggestedReorderLevel, SuggestedStockLevel)
**Primary Key** IngredientCode

Note to marker – this is from the many to many relationship between the ingredients and the stockorder

**IngredientOrders(**StockOrderNumber, IngredientCode, Quantity, Price)
**Primary Key** StockOrderNumber, IngredientCode
**Foreign Key** StockOrderNumber **references** StockOrder(StockOrderNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** IngredientCode **references** Ingredient(IngredientCode) ON UPDATE CASCADE ON DELETE SET NULL

**MenuItem(**MenuItemCode, Name, Size, SellingPrice)
**Primary Key** MenuItemCode

**MenuItemComposition(**MenuItemCode, IngredientCode, Quantity)
**Primary Key** MenuItemCode, IngredientCode
**Foreign Key** MenuItemCode **References** MenuItem(MenuItemCode) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** IngredientCode **references** Ingredient(IngredientCode) ON UPDATE CASCADE ON DELETE CASCADE

**Order(**OrderNumber, Description, OrderStatus, Date, CustomerID,EmployeeNumber, Amount Due, PaymentMethod, PaymentApprovalNumber)
**Primary Key** OrderNumber
**Foreign Key** CustomerID **References** Customer(CustomerID) ON UPDATE CASCADE ON DELETE SET NULL
**Foreign Key** EmployeeNumber **References** Instore(EmployeeNumber) ON UPDATE CASCADE ON DELETE SET NULL

**WalkInOrder**(OrderNumber, WalkInTime, PickupTime)
**Primary Key** OrderNumber
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE CASCADE ON DELETE CASCADE

**PhoneDeliveryOrder(**OrderNumber, CallTime, TerminationTime, DeliveryTime, AddressStreet, AddressCity, AddressPostcode, Driver)
**Primary Key** OrderNumber
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** Driver **references** DriverShift(ShiftID) ON UPDATE CASCADE ON DELETE SET NULL

**PhonePickupOrder(**OrderNumber, CallTime, TerminationTime, PickupTime)
**Primary Key** OrderNumber
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE CASCADE ON DELETE CASCADE

**OrderItems(**OrderNumber, MenuItemCode, Quantity)
**Primary Key** OrderNumber, MenuItemCode
**Foreign Key** OrderNumber **References** Order(OrderNumber) ON UPDATE CASCADE ON DELETE CASCADE
**Foreign Key** MenuItemCode **References** MenuItem(MenuItemCode) ON UPDATE CASCADE ON DELETE SET CASCADE

**Customer(**CustomerID, PhoneNumber, FirstName, LastName, isHoax, AddressStreet, AddressCity, AddressPostcode)
**Primary Key** CustomerID
**Alternate Key** Phone Number

**Instore(**EmployeeNumber, FirstName, LastName, ContactNumber, TaxFileNumber, Description, HourlyRate, AddressStreet, AddressCity, AddressPostcode, AccountNumber, BankCode, BankName**)**
**Primary Key** EmployeeNumber
**Alternate Key** TaxFileNumber

**Driver(**EmployeeNumber, FirstName, LastName, ContactNumber, TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity, AddressPostcode, AccountNumber, BankCode, BankName, DriversLicenseNumber)
**Primary Key** EmployeeNumber
**Alternate Key** TaxFileNumber
**Alternate Key** DriversLicenseNumber

**DriverShift(**ShiftID, Date Start, Time Start, Time End, Date End, EmployeeNumber, PaymentID)
**Primary Key** ShiftID
**Foreign Key** EmployeeNumber **References** Driver(EmployeeNumber) ON UPDATE NO ACTION ON DELETE NO ACTION
**Foreign Key** PaymentID **References** DriverPay**(**PaymentID) ON UPDATE NO ACTION ON DELETE NO ACTION

**InstoreShift(**ShiftID, Date Start, Time Start, Time End, Date End, EmployeeNumber, PaymentID)
**Primary Key** ShiftID
**Foreign Key** EmployeeNumber **References** Instore(EmployeeNumber) ON UPDATE NO ACTION ON DELETE NO ACTION
**Foreign Key** PaymentID **References** InStorePay**(**PaymentID) ON UPDATE CASCADE ON DELETE CASCADE

**DriverPay**(PaymentID, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, PaymentPeriodEnd, TotalPaid, NumberOfDeliveries)
**Primary Key** PaymentID

**InStorePay**(PaymentID, TotalPayment, TaxWithheld, PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, PaymentPeriodEnd, TotalPaid, HoursWorked)
**Primary Key** PaymentID

## SQL Implementation

Note to marker – please check if something is a comment. There are a quite a few commented out things in the SQL Document. It would preferrable to view this in the SSMS as there is syntax highlighting.

```
DROP TABLE PhoneDeliveryOrder;
DROP TABLE PhonePickupOrder
DROP TABLE WalkInOrder;
DROP TABLE OrderItems;
DROP TABLE Orders;
DROP TABLE InStoreShift;
DROP TABLE DriverShift;
DROP TABLE DriverPay;
DROP TABLE InStorePay;
DROP TABLE Driver;
DROP TABLE InStore;
DROP TABLE Customer;
DROP TABLE MenuItemComposition;
DROP TABLE MenuItem;
DROP TABLE IngredientOrders;
DROP TABLE Ingredient;
DROP TABLE StockOrder;

--StockOrder(StockOrderNumber, DateOrdered, DateReceived, Status)
--Primary Key StockOrderNumber


CREATE TABLE StockOrder (
      StockOrderNumber       CHAR(10)    NOT NULL    PRIMARY KEY,
      DateOrdered                  DATE        NOT NULL,
      DateReceived                 DATE,
      OrderStatus                  CHAR(2)          NOT NULL    DEFAULT
      'PR', --PR means purchased but not received
      CONSTRAINT dteChk            CHECK (DateReceived > DateOrdered),
);

--Ingredient(IngredientCode, Name, Description, Type,
SuggestedReorderLevel, SuggestedStockLevel)
--Primary Key IngredientCode


CREATE TABLE Ingredient (
      IngredientCode               CHAR(10)    NOT NULL    PRIMARY KEY,
      IngredientName               VARCHAR(30) NOT NULL,
      IngredientType               VARCHAR(30),
      SuggestedReorderLevel  INTEGER,
      SuggestedStockLevel          INTEGER,
      CONSTRAINT sggstReorderPos   CHECK(SuggestedReorderLevel>0),
      CONSTRAINT sggstStckPos            CHECK(SuggestedStockLevel>0)
);

--IngredientOrders(StockOrderNumber, IngredientCode, Quantity, Price)
```

```
--Primary Key StockOrderNumber, IngredientCode
--Foreign Key StockOrderNumber references StockOrder(StockOrderNumber)
ON UPDATE CASCADE ON DELETE CASCADE
--Foreign Key IngredientCode

CREATE TABLE IngredientOrders (
      StockOrderNumber       CHAR(10)     NOT NULL,
      IngredientCode              CHAR(10)     NOT NULL,
      Quantity                    INTEGER           NOT NULL,
      Price                       DECIMAL(6,2) NOT NULL,
      PRIMARY KEY (StockOrderNumber, IngredientCode),
      CONSTRAINT fkStkOrd         FOREIGN KEY(StockOrderNumber)
      REFERENCES  StockOrder(StockOrderNumber)
            ON UPDATE CASCADE ON DELETE CASCADE,
      CONSTRAINT fkIng       FOREIGN KEY(IngredientCode)
      REFERENCES  Ingredient(IngredientCode)
            ON UPDATE CASCADE ON DELETE CASCADE,
      CONSTRAINT qntyPos               CHECK(Quantity>0),
      CONSTRAINT prcePos               CHECK(Price>=0)
);

--MenuItem(MenuItemCode, Name, Size, SellingPrice)
--Primary Key MenuItemCode

CREATE TABLE MenuItem (
      MenuItemCode                CHAR(10)    NOT NULL    PRIMARY KEY,
      MenuItemName                VARCHAR(35) NOT NULL,
      Size                        CHAR(1)          NOT NULL,
      SellingPrice                DECIMAL(5,2),
      Description                 VARCHAR(50),

      CONSTRAINT szechk           CHECK(Size IN ('S','M','L')),
      CONSTRAINT sellprcePos      CHECK(SellingPrice >= 0),
);

--MenuItemComposition(MenuItemCode, IngredientCode, Quantity)
--Primary Key MenuItemCode, IngredientCode
--Foreign Key MenuItemCode References MenuItem(MenuItemCode) ON UPDATE
CASCADE ON DELETE CASCADE
--Foreign Key IngredientCode references Ingredient(IngredientCode) ON
UPDATE CASCADE ON DELETE CASCADE


CREATE TABLE MenuItemComposition (
      MenuItemCode                CHAR(10)    NOT NULL,
      IngredientCode              CHAR(10)    NOT NULL,
      Quantity                    INTEGER          NOT NULL,

      PRIMARY KEY(MenuItemCode, IngredientCode),
      CONSTRAINT fkmnuitem   FOREIGN KEY(MenuItemCode)
      REFERENCES MenuItem(MenuItemCode)
            ON UPDATE CASCADE ON DELETE CASCADE,
      CONSTRAINT fkingcode   FOREIGN KEY(IngredientCode)
      REFERENCES Ingredient(IngredientCode)
            ON UPDATE CASCADE ON DELETE CASCADE,
      CONSTRAINT qntpos      CHECK(Quantity > 0)
);
```

```
--Customer(CustomerID, PhoneNumber, FirstName, LastName, isHoax,
AddressStreet, AddressCity, AddressPostcode)
--Primary Key CustomerID
--Alternate Key Phone Number


CREATE TABLE Customer (
      CustomerID                  CHAR(10)    NOT NULL    PRIMARY KEY,
      PhoneNumber                 VARCHAR(15) NOT NULL,
      FirstName                   VARCHAR(35) NOT NULL,
      LastName                    VARCHAR(35) NOT NULL,
      AddressStreet               VARCHAR(35) NOT NULL,
      AddressCity                 VARCHAR(30) NOT NULL,
      AddressPostcode             CHAR(4)           NOT NULL,
      isHoax                            VARCHAR(10) DEFAULT
      'unverified',
      CONSTRAINT ishx             CHECK(isHoax
IN('verified','unverified')),
      UNIQUE(PhoneNumber)
);

--Instore(EmployeeNumber, FirstName, LastName, ContactNumber,
TaxFileNumber, Description, HourlyRate, AddressStreet, AddressCity,
AddressPostcode, AccountNumber, BankCode, BankName)
--Primary Key EmployeeNumber
--Alternate Key TaxFileNumber

CREATE TABLE InStore (
      EmployeeNumber              CHAR(10)    NOT NULL    PRIMARY KEY,
      FirstName                   VARCHAR(35) NOT NULL,
      LastName                    VARCHAR(35) NOT NULL,
      HourlyRate                  DECIMAL(5,2),
      ContactNumber               VARCHAR(15) NOT NULL,
      TaxFileNumber               CHAR(9)           NOT NULL,
      AccountNumber               VARCHAR(16) NOT NULL,
      AddressStreet               VARCHAR(35) NOT NULL,
      AddressCity                 VARCHAR(30) NOT NULL,
      AddressPostcode             CHAR(4)           NOT NULL,
      BankCode                    CHAR(6)           NOT NULL,
      BankName                    VARCHAR(50) NOT NULL,
      Description                 VARCHAR(500),

      UNIQUE(TaxFileNumber),

);

--Driver(EmployeeNumber, FirstName, LastName, ContactNumber,
TaxFileNumber, Description, PaymentRate, AddressStreet, AddressCity,
AddressPostcode, AccountNumber, BankCode, BankName,
DriversLicenseNumber)
--Primary Key EmployeeNumber
--Alternate Key TaxFileNumber
--Alternate Key DriversLicenseNumber

CREATE TABLE Driver (
      EmployeeNumber              CHAR(10)    NOT NULL    PRIMARY KEY,
      FirstName                   VARCHAR(35) NOT NULL,
      LastName                    VARCHAR(35) NOT NULL,
```

```
      PaymentRate                   DECIMAL(5,2),
      ContactNumber                 VARCHAR(15) NOT NULL,
      TaxFileNumber                 CHAR(9)           NOT NULL,
      DriversLicenseNumber    VARCHAR(15) NOT NULL,
      AccountNumber                 VARCHAR(16) NOT NULL,
      AddressStreet                 VARCHAR(35) NOT NULL,
      AddressCity                   VARCHAR(30) NOT NULL,
      AddressPostcode               CHAR(4)           NOT NULL,
      BankCode                      CHAR(6)           NOT NULL,
      BankName                      VARCHAR(50) NOT NULL,
      Description                   VARCHAR(500),

      UNIQUE(TaxFileNumber),
      UNIQUE(DriversLicenseNumber),

);

--InStorePay(PaymentID, EmployeeNumber, TotalPayment, TaxWithheld,
PaymentDateStart, PaymentDateEnd, PaymentPeriodStart, PaymentPeriodEnd,
TotalPaid, HoursWorked)

CREATE TABLE InStorePay (
      PaymentID                     CHAR(10)    NOT NULL    PRIMARY KEY,
      TotalPayment                  FLOAT       NOT NULL,
      TaxWitheld                    FLOAT,
      PaymentDateStart        DATETIME,
      PaymentDateEnd                DATETIME,
      PaymentPeriodStart            DATETIME,
      PaymentPeriodEnd        DATETIME,
      HoursWorkedPaid               TINYINT           NOT NULL,

      CONSTRAINT pymtcorrin         CHECK(PaymentDateStart <
PaymentDateEnd),
      CONSTRAINT pymtprdcorrin      CHECK(PaymentPeriodStart <
PaymentPeriodEnd),
);

--DriverPay(PaymentID, TotalPayment, TaxWithheld, PaymentDateStart,
PaymentDateEnd, PaymentPeriodStart, PaymentPeriodEnd, TotalPaid,
NumberOfDeliveries)
--Primary Key PaymentID


CREATE TABLE DriverPay (
      PaymentID                     CHAR(10)    NOT NULL    PRIMARY KEY,
      TotalPayment                  FLOAT       NOT NULL,
      TaxWitheld                    FLOAT,
      PaymentDateStart        DATETIME,
      PaymentDateEnd                DATETIME,
      PaymentPeriodStart            DATETIME,
      PaymentPeriodEnd        DATETIME,
      DeliveriesPaid                TINYINT           NOT NULL,

      CONSTRAINT pymtcorrdr         CHECK(PaymentDateStart <
PaymentDateEnd),
      CONSTRAINT pymtprdcorrdr      CHECK(PaymentPeriodStart <
PaymentPeriodEnd),
);
```

```
--DriverShift(ShiftID, Date Start, Time Start, Time End, Date End,
EmployeeNumber, PaymentID)
--Primary Key ShiftID
--Foreign Key EmployeeNumber References Driver(EmployeeNumber) ON
UPDATE NO ACTION ON DELETE NO ACTION
--Foreign Key PaymentID References DriverPay(PaymentID) ON UPDATE NO
ACTION ON DELETE NO ACTION


CREATE TABLE DriverShift (
      ShiftID                          CHAR(10)    NOT NULL    PRIMARY
KEY,
      Start                      DATETIME,
      Finish                           DATETIME,
      EmployeeNumber             CHAR(10)    NOT NULL,
      PaymentID                  CHAR(10),

      CONSTRAINT startfinishcorrdr CHECK(Start <= Finish),
      CONSTRAINT employeeFKdr          FOREIGN KEY(EmployeeNumber)
REFERENCES Driver(EmployeeNumber)
            ON UPDATE NO ACTION ON DELETE NO ACTION,
      CONSTRAINT paymentFKdr       FOREIGN KEY(PaymentID)
      REFERENCES DriverPay(PaymentID)
            ON UPDATE NO ACTION ON DELETE NO ACTION,
);

--InstoreShift(ShiftID, Date Start, Time Start, Time End, Date End,
EmployeeNumber, PaymentID)
--Primary Key ShiftID
--Foreign Key EmployeeNumber References Instore(EmployeeNumber) ON
UPDATE NO ACTION ON DELETE NO ACTION
--Foreign Key PaymentID References InStorePay(PaymentID) ON UPDATE
CASCADE ON DELETE CASCADE
--DriverPay(PaymentID, TotalPayment, TaxWithheld, PaymentDateStart,
PaymentDateEnd, PaymentPeriodStart, PaymentPeriodEnd, TotalPaid,
NumberOfDeliveries)
--Primary Key PaymentID


CREATE TABLE InStoreShift (
      ShiftID                          CHAR(10)    NOT NULL    PRIMARY
KEY,
      Start                      DATETIME,
      Finish                           DATETIME,
      EmployeeNumber             CHAR(10)    NOT NULL,
      PaymentID                  CHAR(10),
      HoursWorked                TINYINT,

      CONSTRAINT startfinishcorrin CHECK(Start <= Finish),
      CONSTRAINT employeeFKin          FOREIGN KEY(EmployeeNumber)
REFERENCES InStore(EmployeeNumber)
            ON UPDATE NO ACTION ON DELETE NO ACTION,
      CONSTRAINT paymentFKin       FOREIGN KEY(PaymentID)
      REFERENCES DriverPay(PaymentID)
            ON UPDATE NO ACTION ON DELETE NO ACTION,
);
```

```
--Order(OrderNumber, Description, OrderStatus, Date,
CustomerID,EmployeeNumber, Amount Due, PaymentMethod,
PaymentApprovalNumber)
--Primary Key OrderNumber
--Foreign Key CustomerID References Customer(CustomerID) ON UPDATE
CASCADE ON DELETE SET NULL
--Foreign Key EmployeeNumber References Instore(EmployeeNumber) ON
UPDATE CASCADE ON DELETE SET NULL

CREATE TABLE Orders (
      OrderNumber                 CHAR(10)    NOT NULL    PRIMARY KEY,
      OrderTime                   DATETIME,
      AmountDue                   DECIMAL(7,2),
      PaymentMethod               VARCHAR(20) NOT NULL,
      PaymentApprovalNumber  VARCHAR(20) NOT NULL,
      OrderStatus                 CHAR(2)          DEFAULT 'PR', --
Purchased and being prepared
      CustomerID                  CHAR(10),
      EmployeeNumber              CHAR(10),

      CONSTRAINT fkcmID      FOREIGN KEY(CustomerID)      REFERENCES
Customer(CustomerID)
            ON UPDATE CASCADE ON DELETE SET NULL,
      CONSTRAINT emnmorFK          FOREIGN KEY(EmployeeNumber)
      REFERENCES InStore(EmployeeNumber)
            ON UPDATE CASCADE ON DELETE SET NULL,

      CHECK (AmountDue>0)
);

--OrderItems(OrderNumber, MenuItemCode, Quantity)
--Primary Key OrderNumber, MenuItemCode
--Foreign Key OrderNumber References Order(OrderNumber) ON UPDATE
CASCADE ON DELETE CASCADE
--Foreign Key MenuItemCode References MenuItem(MenuItemCode) ON UPDATE
CASCADE ON DELETE SET NULL


CREATE TABLE OrderItems(
      OrderNumber                 CHAR(10),
      MenuItemCode                CHAR(10),
      Quantity                    INTEGER,

      PRIMARY KEY(OrderNumber,MenuItemCode),
      CONSTRAINT fkOrdIdOrderItems FOREIGN KEY(OrderNumber) REFERENCES
Orders(OrderNumber)
            ON UPDATE CASCADE ON DELETE CASCADE,
      CONSTRAINT fkMenuItemCodeOrderItems     FOREIGN KEY(MenuItemCode)
REFERENCES MenuItem(MenuItemCode)
            ON UPDATE CASCADE ON DELETE CASCADE,
);

--WalkInOrder(OrderNumber, WalkInTime, PickupTime)
--Primary Key OrderNumber
--Foreign Key OrderNumber References Order(OrderNumber) ON UPDATE
CASCADE ON DELETE CASCADE
```

```
CREATE TABLE WalkInOrder (
      OrderNumber                  CHAR(10)    PRIMARY KEY,
      WalkInTime                   DATETIME2,
      PickupTime                   DATETIME2,

      CONSTRAINT fkOrderNumberWalkInOrder     FOREIGN KEY(OrderNumber)
REFERENCES Orders(OrderNumber)
            ON UPDATE CASCADE ON DELETE CASCADE
);

--PhonePickupOrder(OrderNumber, CallTime, TerminationTime, PickupTime)
--Primary Key OrderNumber
--Foreign Key OrderNumber References Order(OrderNumber) ON UPDATE
CASCADE ON DELETE CASCADE

CREATE TABLE PhonePickupOrder (
      OrderNumber                  CHAR(10)    PRIMARY KEY,
      CallTime                     DATETIME2,
      PickupTime                   DATETIME2,
      TerminationTime              DATETIME2,

      CONSTRAINT timecorrph        CHECK(TerminationTime > CallTime),
      CONSTRAINT fkOrderNumberPhoneOrderph    FOREIGN KEY(OrderNumber)
REFERENCES Orders(OrderNumber)
            ON UPDATE CASCADE ON DELETE CASCADE
);

--PhoneDeliveryOrder(OrderNumber, CallTime, TerminationTime,
DeliveryTime, AddressStreet, AddressCity, AddressPostcode, Driver)
--Primary Key OrderNumber
--Foreign Key OrderNumber References Order(OrderNumber) ON UPDATE
CASCADE ON DELETE CASCADE
--Foreign Key Driver references DriverShift(ShiftID) ON UPDATE CASCADE
ON DELETE SET NULL


CREATE TABLE PhoneDeliveryOrder (
      OrderNumber                  CHAR(10)    PRIMARY KEY,
      CallTime                     DATETIME2,
      TerminationTime              DATETIME2,
      DeliveryTime                 DATETIME2,
      ShiftID                            CHAR(10),
      AddressStreet                VARCHAR(35) NOT NULL,
      AddressCity                  VARCHAR(35) NOT NULL,
      AddressPostcode              CHAR(4)          NOT NULL,

      CONSTRAINT timecorrdl        CHECK(TerminationTime > CallTime),
      CONSTRAINT fkOrderNumberPhoneOrderdl    FOREIGN KEY(OrderNumber)
REFERENCES Orders(OrderNumber)
            ON UPDATE CASCADE ON DELETE CASCADE,
      CONSTRAINT fkdrvershft                        FOREIGN KEY(ShiftID)
       REFERENCES DriverShift(ShiftID)
      ON UPDATE CASCADE ON DELETE SET NULL,
);


--CREATE TABLE Ingredient (
```

```
--    IngredientCode              CHAR(10)    NOT NULL    PRIMARY KEY,
--    IngredientName              VARCHAR(30) NOT NULL,
--    IngredientType              VARCHAR(30),
--    SuggestedReorderLevel  INTEGER,
--    SuggeedStockLevel      INTEGER,
--    CONSTRAINT sggstReorderPos   CHECK(SuggestedReorderLevel>0),
--    CONSTRAINT sggstStckPos           CHECK(SuggestedStockLevel>0)
--);

--This will create NULL's as I did not specify Ingredient type

INSERT INTO
Ingredient(IngredientCode,IngredientName,SuggestedReorderLevel,Suggeste
dStockLevel) VALUES('I000000001','Dough',150,340);
INSERT INTO
Ingredient(IngredientCode,IngredientName,SuggestedReorderLevel,Suggeste
dStockLevel) VALUES('I000000002','Cheese',300,270);
INSERT INTO
Ingredient(IngredientCode,IngredientName,IngredientType,SuggestedReorde
rLevel,SuggestedStockLevel)
VALUES('I000000003','Toppings','Tomato',100,620);

--SELECT * FROM Ingredient;

--CREATE TABLE StockOrder (
--    StockOrderNumber       CHAR(10)    NOT NULL    PRIMARY KEY,
--    DateOrdered                 DATE        NOT NULL,
--    DateReceived                DATE,
--    OrderStatus                 CHAR(2)          NOT NULL    DEFAULT
      'PR', --PR means purchased but not received
--    CONSTRAINT dteChk           CHECK (DateReceived < DateOrdered),
--);

INSERT INTO StockOrder(StockOrderNumber, DateOrdered)
VALUES('S000000001','15-OCT-2021');
INSERT INTO StockOrder(StockOrderNumber, DateOrdered, DateReceived,
OrderStatus) VALUES('s000000002','15-JUN-2021','21-JUN-2021','RC');
INSERT INTO StockOrder(StockOrderNumber, DateOrdered, DateReceived,
OrderStatus) VALUES('s000000004','15-JUN-2020','21-JUN-2020','RC');
INSERT INTO StockOrder(StockOrderNumber, DateOrdered, OrderStatus)
VALUES('s000000003','1-AUG-2020','CL');
INSERT INTO StockOrder(StockOrderNumber, DateOrdered)
VALUES('S000000006','15-OCT-2020');
INSERT INTO StockOrder(StockOrderNumber, DateOrdered, DateReceived,
OrderStatus) VALUES('s000000005','15-JUN-2020','21-JUN-2020','RC');

--SELECT * FROM StockOrder;

--CREATE TABLE IngredientOrders (
--    StockOrderNumber       CHAR(10)     NOT NULL,
--    IngredientCode              CHAR(10)     NOT NULL,
--    Quantity                    INTEGER           NOT NULL,
--    Price                       DECIMAL(6,2) NOT NULL,
--    PRIMARY KEY (StockOrderNumber, IngredientCode),
--    CONSTRAINT fkStkOrd         FOREIGN KEY(StockOrderNumber)
      REFERENCES  StockOrder(StockOrderNumber)
--          ON UPDATE CASCADE ON DELETE CASCADE,
```

```
--    CONSTRAINT fkIng       FOREIGN KEY(IngredientCode)
      REFERENCES  Ingredient(IngredientCode)
--          ON UPDATE CASCADE ON DELETE CASCADE,
--    CONSTRAINT qntyPos                CHECK(Quantity>0),
--    CONSTRAINT prcePos                CHECK(Price>=0)
--);

INSERT INTO IngredientOrders VALUES('S000000001','I000000001',430,3);
INSERT INTO IngredientOrders
VALUES('S000000001','I000000002',230,0.34);
INSERT INTO IngredientOrders
VALUES('s000000003','I000000003',1000,0.03)
INSERT INTO IngredientOrders VALUES('S000000002','I000000001',430,3);
INSERT INTO IngredientOrders
VALUES('S000000002','I000000002',230,0.34);
INSERT INTO IngredientOrders
VALUES('s000000002','I000000003',1000,0.03);;

INSERT INTO IngredientOrders VALUES('S000000004','I000000001',430,3);
INSERT INTO IngredientOrders
VALUES('S000000004','I000000002',230,0.34);
INSERT INTO IngredientOrders
VALUES('s000000005','I000000003',1000,0.03)
INSERT INTO IngredientOrders VALUES('S000000006','I000000001',430,3);
INSERT INTO IngredientOrders
VALUES('S000000004','I000000003',230,0.34);
INSERT INTO IngredientOrders
VALUES('s000000006','I000000003',1000,0.03);;

--SELECT * FROM IngredientOrders;

--CREATE TABLE MenuItem (
--    MenuItemCode                CHAR(10)    NOT NULL    PRIMARY KEY,
--    MenuItemName                VARCHAR(35) NOT NULL,
--    Size                        CHAR(1)          NOT NULL,
--    SellingPrice                DECIMAL(5,2),
--    Description                 VARCHAR(50),

--    CONSTRAINT szechk           CHECK(Size IN ('S','M','L')),
--    CONSTRAINT sellprcePos      CHECK(SellingPrice >= 0),
--);

INSERT INTO MenuItem VALUES('MI00000001','Supreme','M',19.99,'Supreme
pizza is the best pizza');
INSERT INTO MenuItem VALUES('MI00000002','Hawaiin','L',12.99,'Pinapple
does belong on pizza!!!!');
INSERT INTO MenuItem(MenuItemCode,MenuItemName,Size) VALUES
('MI00000003','Meat Lovers', 'S');

--SELECT * FROM MenuItem;

--CREATE TABLE MenuItemComposition (
--    MenuItemCode                CHAR(10)    NOT NULL,
--    IngredientCode              CHAR(10)    NOT NULL,
--    Quantity                    INTEGER          NOT NULL,

--    PRIMARY KEY(MenuItemCode, IngredientCode),
```

```
--    CONSTRAINT fkmnuitem   FOREIGN KEY(MenuItemCode)
      REFERENCES MenuItem(MenuItemCode)
--          ON UPDATE CASCADE ON DELETE CASCADE,
--    CONSTRAINT fkingcode   FOREIGN KEY(IngredientCode)
      REFERENCES Ingredient(IngredientCode)
--          ON UPDATE CASCADE ON DELETE CASCADE,
--    CONSTRAINT qntpos      CHECK(Quantity > 0)
--);

INSERT INTO MenuItemComposition VALUES('MI00000001','I000000001',3);
INSERT INTO MenuItemComposition VALUES('MI00000001','I000000002',76);
INSERT INTO MenuItemComposition VALUES('MI00000001','I000000003',4);
INSERT INTO MenuItemComposition VALUES('MI00000002','I000000001',5);
INSERT INTO MenuItemComposition VALUES('MI00000002','I000000002',2);
INSERT INTO MenuItemComposition VALUES('MI00000003','I000000001',7);


--CREATE TABLE Customer (
--    CustomerID                  CHAR(10)    NOT NULL    PRIMARY KEY,
--    PhoneNumber                 VARCHAR(15) NOT NULL,
--    FirstName                   VARCHAR(35) NOT NULL,
--    LastName                    VARCHAR(35) NOT NULL,
--    AddressStreet               VARCHAR(35) NOT NULL,
--    AddressCity                 VARCHAR(30) NOT NULL,
--    AddressPostcode             CHAR(4)           NOT NULL,
--    isHoax                            VARCHAR(10) DEFAULT
      'unverified',
--    CONSTRAINT ishx             CHECK(isHoax
IN('verified','unverified')),
--    UNIQUE(PhoneNumber)
--);

INSERT INTO Customer VALUES('C000000001','04353982','John','Doe','11
Sesseme Street', 'Sesseme City','2234','verified');
INSERT INTO
Customer(CustomerID,PhoneNumber,FirstName,LastName,AddressCity,AddressS
treet,AddressPostcode)
      VALUES('C000000002','0490332245','Jane','Doe','Brisbane','11
Delitasteland','2345');
INSERT INTO Customer
VALUES('C000000003','0435398542','Johhny','Doe','11 Gamer Avenue',
'Abcland','2234','verified');

--SELECT * FROM Customer;

--CREATE TABLE InStore (
--    EmployeeNumber              CHAR(10)    NOT NULL    PRIMARY KEY,
--    FirstName                   VARCHAR(35) NOT NULL,
--    LastName                    VARCHAR(35) NOT NULL,
--    HourlyRate                  DECIMAL(5,2),
--    ContactNumber               VARCHAR(15) NOT NULL,
--    TaxFileNumber               CHAR(9)           NOT NULL,
--    AccountNumber               VARCHAR(16) NOT NULL,
--    AddressStreet               VARCHAR(35) NOT NULL,
--    AddressCity                 VARCHAR(30) NOT NULL,
--    AddressPostcode             CHAR(4)           NOT NULL,
--    BankCode                    CHAR(6)           NOT NULL,
--    BankName                    VARCHAR(50) NOT NULL,
```

```
--    Description                 VARCHAR(500),

--    UNIQUE(TaxFileNumber),

--);

INSERT INTO
InStore(EmployeeNumber,FirstName,LastName,ContactNumber,TaxFileNumber,A
ccountNumber,AddressStreet,AddressCity,AddressPostcode,BankCode,BankNam
e)
      VALUES('E000000001','Jane','Doe','04309535656','123456789','13245
46757','Good Street','Good City','2394','650000','Newcastle
Permanent');
INSERT INTO InStore
VALUES('E000000002','John','Doe',15.42,'043879473','987654321','3290823
5243','Bad Street','Bad City','8945','123456','Bannnnnkkk','Lorem
Ipsum');
INSERT INTO InStore
VALUES('E000000003','Johhny','Doe',15.70,'043895473','111111111','32823
5243','Bad Street','Bad City','8945','123456','Bannnnnkkk','Lorem Ipsum
2');

--SELECT * FROM InStore;

--CREATE TABLE Driver (
--    EmployeeNumber              CHAR(10)    NOT NULL    PRIMARY KEY,
--    FirstName                   VARCHAR(35) NOT NULL,
--    LastName                    VARCHAR(35) NOT NULL,
--    PaymentRate                 DECIMAL(5,2),
--    ContactNumber               VARCHAR(15) NOT NULL,
--    TaxFileNumber               CHAR(9)           NOT NULL,
--    DriversLicenseNumber   VARCHAR(15) NOT NULL,
--    AccountNumber               VARCHAR(16) NOT NULL,
--    AddressStreet               VARCHAR(35) NOT NULL,
--    AddressCity                 VARCHAR(30) NOT NULL,
--    AddressPostcode             CHAR(4)           NOT NULL,
--    BankCode                    CHAR(6)           NOT NULL,
--    BankName                    VARCHAR(50) NOT NULL,
--    Description                 VARCHAR(500),

--    UNIQUE(TaxFileNumber),
--    UNIQUE(DriversLicenseNumber),

--);

INSERT INTO
Driver(EmployeeNumber,FirstName,LastName,ContactNumber,TaxFileNumber,Dr
iversLicenseNumber,AccountNumber,AddressStreet,AddressCity,AddressPostc
ode,BankCode,BankName)
      VALUES('DR00000001','Jane','Doe','04309535656','123456789','34290
8534','1324546757','Good Street','Good City','2394','650000','Newcastle
Permanent');
INSERT INTO Driver
VALUES('DR00000002','John','Doe',3.42,'043879473','987654321','84375934
353','32908235243','Bad Street','Bad
City','8945','123456','Bannnnnkkk','Lorem Ipsum');
INSERT INTO Driver
VALUES('DR00000003','Johhny','Doe',6.70,'043895473','111111111','895486
```

```
940','328235243','Bad Street','Bad
City','8945','123456','Bannnnnkkk','Lorem Ipsum 2');

--SELECT * FROM Driver;

--CREATE TABLE InStorePay (
--    PaymentID                   CHAR(10)    NOT NULL    PRIMARY KEY,
--    TotalPayment                FLOAT       NOT NULL,
--    TaxWitheld                  FLOAT,
--    PaymentDateStart        DATETIME,
--    PaymentDateEnd              DATETIME,
--    PaymentPeriodStart          DATETIME,
--    PaymentPeriodEnd        DATETIME,
--    HoursWorkedPaid             TINYINT          NOT NULL,

--    CONSTRAINT pymtcorrin       CHECK(PaymentDateStart <
PaymentDateEnd),
--    CONSTRAINT pymtprdcorrin    CHECK(PaymentPeriodStart <
PaymentPeriodEnd),
--);

INSERT INTO InStorePay(PaymentID,TotalPayment,HoursWorkedPaid)
VALUES('INSTRP0001',501.34,'13');
INSERT INTO
InStorePay(PaymentID,TotalPayment,PaymentDateEnd,PaymentPeriodStart,Hou
rsWorkedPaid)
     VALUES('INSTRP0002',1000.34,'13-JUN-2021','14-JUN-2021',17);
INSERT INTO InStorePay VALUES('INSTRP0003',1302.23,80,'14-JAN-
2021','15-JAN-2021','12-JAN-2021','21-JAN-2021',34);
INSERT INTO InStorePay(PaymentID,TotalPayment,HoursWorkedPaid)
VALUES('INSTRP0004',501.34,'13');
INSERT INTO
InStorePay(PaymentID,TotalPayment,PaymentDateEnd,PaymentPeriodStart,Hou
rsWorkedPaid)
     VALUES('INSTRP0005',1000.34,'13-JUN-2020','14-JUN-2020',17);
INSERT INTO InStorePay VALUES('INSTRP0006',1302.23,80,'14-JAN-
2020','15-JAN-2020','12-JAN-2020','21-JAN-2020',34);

--SELECT * FROM InStorePay;

--CREATE TABLE DriverPay (
--    PaymentID                   CHAR(10)    NOT NULL    PRIMARY KEY,
--    TotalPayment                FLOAT       NOT NULL,
--    TaxWitheld                  FLOAT,
--    PaymentDateStart        DATETIME,
--    PaymentDateEnd              DATETIME,
--    PaymentPeriodStart          DATETIME,
--    PaymentPeriodEnd        DATETIME,
--    DeliveriesPaid              TINYINT          NOT NULL,

--    CONSTRAINT pymtcorrdr       CHECK(PaymentDateStart <
PaymentDateEnd),
--    CONSTRAINT pymtprdcorrdr    CHECK(PaymentPeriodStart <
PaymentPeriodEnd),
--);

INSERT INTO DriverPay(PaymentID,TotalPayment,DeliveriesPaid)
VALUES('INSTRP0001',501.34,'13');
```

```
INSERT INTO
DriverPay(PaymentID,TotalPayment,PaymentDateEnd,PaymentPeriodStart,Deli
veriesPaid)
      VALUES('INSTRP0002',1000.34,'13-JUN-2021','14-JUN-2021',17);
INSERT INTO DriverPay VALUES('INSTRP0003',1302.23,80,'14-JAN-2021','15-
JAN-2021','12-JAN-2021','21-JAN-2021',34);
INSERT INTO DriverPay(PaymentID,TotalPayment,DeliveriesPaid)
VALUES('INSTRP0004',501.34,'13');
INSERT INTO
DriverPay(PaymentID,TotalPayment,PaymentDateEnd,PaymentPeriodStart,Deli
veriesPaid)
      VALUES('INSTRP0005',1000.34,'13-JUN-2020','14-JUN-2020',17);
INSERT INTO DriverPay VALUES('INSTRP0006',1302.23,80,'14-JAN-2020','15-
JAN-2020','12-JAN-2020','21-JAN-2020',34);

--SELECT * FROM DriverPay;

--CREATE TABLE InStoreShift (
--    ShiftID                          CHAR(10)    NOT NULL    PRIMARY
KEY,
--    Start                       DATETIME,
--    Finish                           DATETIME,
--    EmployeeNumber              CHAR(10)    NOT NULL,
--    PaymentID                   CHAR(10),
--    HoursWorked                 TINYINT,

--    CONSTRAINT startfinishcorrin CHECK(Start < Finish),
--    CONSTRAINT employeeFKin           FOREIGN KEY(EmployeeNumber)
REFERENCES InStore(EmployeeNumber)
--          ON UPDATE NO ACTION ON DELETE NO ACTION,
--    CONSTRAINT paymentFKin      FOREIGN KEY(PaymentID)
      REFERENCES DriverPay(PaymentID)
--          ON UPDATE NO ACTION ON DELETE NO ACTION,
--);

INSERT INTO InStoreShift(ShiftID,EmployeeNumber)
      VALUES('SH00000001','E000000001');

INSERT INTO
InStoreShift(ShiftID,Start,EmployeeNumber,PaymentID,HoursWorked)
      VALUES('SH00000002','14-JAN-2021
00:00:00','E000000002','INSTRP0001',4);

INSERT INTO InStoreShift VALUES('SH00000003','14-AUG-2021
03:43:21','15-AUG-2021 00:23:31','E000000003','INSTRP0001',5);

INSERT INTO InStoreShift VALUES('SH00000004','14-JUN-2021
03:43:21','15-AUG-2021 00:23:31','E000000003','INSTRP0001',5);

INSERT INTO InStoreShift VALUES('SH00000005','14-FEB-2021
03:43:21','15-JUN-2021 00:23:31','E000000003','INSTRP0002',5);

INSERT INTO InStoreShift VALUES('SH00000006','14-FEB-2020
03:43:21','15-JUN-2020 00:23:31','E000000002','INSTRP0004',4);

INSERT INTO InStoreShift VALUES('SH00000007','14-OCT-2021
03:43:21','15-OCT-2021 00:23:31','E000000002','INSTRP0005',5);
```

```
INSERT INTO InStoreShift VALUES('SH00000008','14-NOV-2021
03:43:21','15-NOV-2021 00:23:31','E000000001','INSTRP0004',6);

INSERT INTO InStoreShift VALUES('SH00000009','14-DEC-2021
03:43:21','15-DEC-2021 00:23:31','E000000001','INSTRP0001',4);

INSERT INTO InStoreShift VALUES('SH00000010','14-DEC-2021
03:43:21','15-DEC-2021 00:23:31','E000000001','INSTRP0003',4);

--SELECT * FROM InStoreShift;

--CREATE TABLE DriverShift (
--    ShiftID                          CHAR(10)    NOT NULL    PRIMARY
KEY,
--    Start                       DATETIME,
--    Finish                           DATETIME,
--    EmployeeNumber              CHAR(10)    NOT NULL,
--    PaymentID                   CHAR(10),

--    CONSTRAINT startfinishcorrdr CHECK(Start < Finish),
--    CONSTRAINT employeeFKdr           FOREIGN KEY(EmployeeNumber)
REFERENCES Driver(EmployeeNumber)
--          ON UPDATE NO ACTION ON DELETE NO ACTION,
--    CONSTRAINT paymentFKdr      FOREIGN KEY(PaymentID)
      REFERENCES DriverPay(PaymentID)
--          ON UPDATE NO ACTION ON DELETE NO ACTION,
--);

INSERT INTO DriverShift VALUES('SH00000001','14-AUG-2021 03:43:21','15-
AUG-2021 00:23:31','DR00000003','INSTRP0004');

INSERT INTO DriverShift VALUES('SH00000002','14-JUN-2021 03:43:21','15-
AUG-2021 00:23:31','DR00000001','INSTRP0005');

INSERT INTO DriverShift VALUES('SH00000003','14-AUG-2021 03:43:21','15-
AUG-2021 00:23:31','DR00000003','INSTRP0001');

INSERT INTO DriverShift VALUES('SH00000004','14-JUN-2021 03:43:21','15-
AUG-2021 00:23:31','DR00000003','INSTRP0001');

INSERT INTO DriverShift VALUES('SH00000005','14-FEB-2021 03:43:21','15-
JUN-2021 00:23:31','DR00000003','INSTRP0002');

INSERT INTO DriverShift VALUES('SH00000006','18-OCT-2021 03:43:21','19-
OCT-2021 00:23:31','DR00000002','INSTRP0004');

INSERT INTO DriverShift VALUES('SH00000007','14-OCT-2021 03:43:21','15-
OCT-2021 00:23:31','DR00000002','INSTRP0005');

INSERT INTO DriverShift VALUES('SH00000008','14-NOV-2021 03:43:21','15-
NOV-2021 00:23:31','DR00000001','INSTRP0004');

INSERT INTO DriverShift VALUES('SH00000009','14-DEC-2021 03:43:21','15-
DEC-2021 00:23:31','DR00000001','INSTRP0001');

INSERT INTO DriverShift VALUES('SH00000010','14-DEC-2021 03:43:21','15-
DEC-2021 00:23:31','DR00000001','INSTRP0003');
```

```
--SELECT * FROM DriverShift;

--CREATE TABLE Orders (
--    OrderNumber                 CHAR(10)    NOT NULL    PRIMARY KEY,
--    OrderTime                   DATETIME,
--    AmountDue                   DECIMAL(7,2),
--    PaymentMethod               VARCHAR(20) NOT NULL,
--    PaymentApprovalNumber  VARCHAR(20) NOT NULL,
--    OrderStatus                 CHAR(2)          DEFAULT 'PR', --
Purchased and being prepared
--    CustomerID                  CHAR(10),
--    EmployeeNumber              CHAR(10),

--    CONSTRAINT fkcmID      FOREIGN KEY(CustomerID)      REFERENCES
Customer(CustomerID)
--          ON UPDATE CASCADE ON DELETE SET NULL,
--    CONSTRAINT emnmorFK          FOREIGN KEY(EmployeeNumber)
      REFERENCES InStore(EmployeeNumber)
--          ON UPDATE CASCADE ON DELETE SET NULL,

--    CHECK (AmountDue>0)
--);

INSERT INTO Orders
      VALUES('O000000001','12-OCT-
2021',45,'Card','P4e90384','AW','C000000001','E000000001');

INSERT INTO Orders
      VALUES('O000000002','12-DEC-
2021',45,'Card','P1223423','CC','C000000002','E000000002');

INSERT INTO Orders
      VALUES('O000000003','12-JAN-
2021',45,'Card','P1224353','CC','C000000003','E000000003');

INSERT INTO Orders
      VALUES('O000000004','13-JAN-
2021',45,'Card','P1224353','CC','C000000002','E000000003');

INSERT INTO Orders
      VALUES('O000000005','17-JAN-
2021',45,'Card','P3924353','DE','C000000002','E000000003');

INSERT INTO Orders
      VALUES('O000000006','18-JAN-
2021',45,'Card','P3924353','CP','C000000002','E000000003');

INSERT INTO Orders
      VALUES('O000000007','19-JAN-
2021',45,'Card','P3924353','CP','C000000001','E000000003');

INSERT INTO Orders
      VALUES('O000000008','20-JAN-
2021',45,'Card','P3924353','CP','C000000001','E000000001');

INSERT INTO Orders
      VALUES('O000000009','19-JAN-
2021',45,'Cash','P392454353','CP','C000000002','E000000002');
```

```
--SELECT * FROM Orders;

--CREATE TABLE OrderItems(
--    OrderNumber                 CHAR(10),
--    MenuItemCode                CHAR(10),
--    Quantity                    INTEGER,

--    PRIMARY KEY(OrderNumber,MenuItemCode),
--    CONSTRAINT fkOrdIdOrderItems FOREIGN KEY(OrderNumber) REFERENCES
Orders(OrderNumber)
--          ON UPDATE CASCADE ON DELETE CASCADE,
--    CONSTRAINT fkMenuItemCodeOrderItems     FOREIGN KEY(MenuItemCode)
REFERENCES MenuItem(MenuItemCode)
--          ON UPDATE CASCADE ON DELETE CASCADE,
--);

INSERT INTO OrderItems
      VALUES('O000000001','MI00000001',3);

INSERT INTO OrderItems
      VALUES('O000000002','MI00000002',4);

INSERT INTO OrderItems
      VALUES('O000000002','MI00000003',5);

INSERT INTO OrderItems
      VALUES('O000000003','MI00000001',6);

INSERT INTO OrderItems
      VALUES('O000000004','MI00000002',2);

INSERT INTO OrderItems
      VALUES('O000000005','MI00000003',1);

INSERT INTO OrderItems
      VALUES('O000000006','MI00000001',8);

INSERT INTO OrderItems
      VALUES('O000000007','MI00000002',9);

INSERT INTO OrderItems
      VALUES('O000000008','MI00000003',1);

INSERT INTO OrderItems
      VALUES('O000000009','MI00000001',4);

INSERT INTO OrderItems
      VALUES('O000000001','MI00000002',10);

INSERT INTO OrderItems
      VALUES('O000000002','MI00000001',3);

INSERT INTO OrderItems
      VALUES('O000000003','MI00000003',1);

--SELECT * FROM OrderItems;
```

```
--CREATE TABLE WalkInOrder (
--    OrderNumber                 CHAR(10)    PRIMARY KEY,
--    WalkInTime                  DATETIME2,
--    PickupTime                  DATETIME2,

--    CONSTRAINT fkOrderNumberWalkInOrder     FOREIGN KEY(OrderNumber)
REFERENCES Orders(OrderNumber)
--          ON UPDATE CASCADE ON DELETE CASCADE
--);

INSERT INTO WalkInOrder
     VALUES('O000000001','12-OCT-2021 00:00:00','12-OCT-2021
1:00:00');

INSERT INTO WalkInOrder
     VALUES('O000000002','13-OCT-2021 00:00:00','13-OCT-2021
1:00:00');

INSERT INTO WalkInOrder
     VALUES('O000000003','14-OCT-2021 00:00:00','14-OCT-2021
1:00:00');

--SELECT * FROM WalkInOrder;

--CREATE TABLE PhonePickupOrder (
--    OrderNumber                 CHAR(10)    PRIMARY KEY,
--    CallTime                    DATETIME2,
--    PickupTime                  DATETIME2,
--    TerminationTime             DATETIME2,

--    CONSTRAINT timecorrph       CHECK(TerminationTime > CallTime),
--    CONSTRAINT fkOrderNumberPhoneOrderph    FOREIGN KEY(OrderNumber)
REFERENCES Orders(OrderNumber)
--          ON UPDATE CASCADE ON DELETE CASCADE
--);

INSERT INTO PhonePickupOrder
     VALUES('O000000004','12-OCT-2021 00:00:00','12-OCT-2021
00:30:00','12-OCT-2021 1:00:00');

INSERT INTO PhonePickupOrder
     VALUES('O000000005','13-OCT-2021 00:00:00','13-OCT-2021
00:30:00','13-OCT-2021 1:00:00');

INSERT INTO PhonePickupOrder
     VALUES('O000000006','14-OCT-2021 00:00:00','14-OCT-2021
00:30:00','14-OCT-2021 1:00:00');

--SELECT * FROM PhonePickupOrder;

--CREATE TABLE PhoneDeliveryOrder (
--    OrderNumber                 CHAR(10)    PRIMARY KEY,
--    CallTime                    DATETIME2,
--    TerminationTime             DATETIME2,
--    DeliveryTime                DATETIME2,
--    ShiftID                          CHAR(10),
--    AddressStreet               VARCHAR(35) NOT NULL,
--    AddressCity                 VARCHAR(35) NOT NULL,
```

```
--    AddressPostcode             CHAR(4)          NOT NULL,

--    CONSTRAINT timecorrdl       CHECK(TerminationTime > CallTime),
--    CONSTRAINT fkOrderNumberPhoneOrderdl    FOREIGN KEY(OrderNumber)
REFERENCES Orders(OrderNumber)
--          ON UPDATE CASCADE ON DELETE CASCADE,
--    CONSTRAINT fkdrvershft                          FOREIGN KEY(ShiftID)
       REFERENCES DriverShift(ShiftID)
--    ON UPDATE CASCADE ON DELETE SET NULL,
--);

INSERT INTO PhoneDeliveryOrder
     VALUES('O000000007','12-OCT-2021 00:00:00','12-OCT-2021
00:30:00','12-OCT-2021 1:00:00','SH00000001','123 Gamer
Avenue','Gamerland','2222');

INSERT INTO PhoneDeliveryOrder
     VALUES('O000000008','13-OCT-2021 00:00:00','13-OCT-2021
00:30:00','13-OCT-2021 1:00:00','SH00000002','123 Im Bored
Street','Whyland','2223');

INSERT INTO PhoneDeliveryOrder
     VALUES('O000000009','14-OCT-2021 00:00:00','14-OCT-2021
00:30:00','14-OCT-2021 1:00:00','SH00000003','123 This took me many
','hours :(','2224');

--SELECT * FROM PhoneDeliveryOrder;

--For  an  in-office  staff  with  id  number  xxx,  print  his/her
--1stname, lname, and hourly payment rate.

SELECT InStore.FirstName, InStore.LastName, InStore.HourlyRate
FROM InStore
WHERE InStore.EmployeeNumber = 'E000000002';

--List all the ingredient details of a menu item named xxx.

SELECT
Ingredient.IngredientCode,Ingredient.IngredientName,Ingredient.Ingredie
ntType,Ingredient.SuggestedReorderLevel,Ingredient.SuggestedStockLevel
FROM Ingredient
INNER JOIN MenuItemComposition
ON Ingredient.IngredientCode=MenuItemComposition.IngredientCode
INNER JOIN MenuItem ON
MenuItem.MenuItemCode=MenuItemComposition.MenuItemCode
WHERE MenuItemName='Supreme';

--List  all  the  shift  details  of  a  delivery  staff  with  first
name
--xxx and last name ttt between date yyy and zzz

SELECT DriverShift.ShiftID, DriverShift.Start, DriverShift.Finish,
DriverShift.PaymentID
FROM DriverShift
INNER JOIN Driver
ON Driver.EmployeeNumber = DriverShift.EmployeeNumber
WHERE Driver.FirstName='Jane' AND Driver.LastName='Doe'
```

```
AND DriverShift.Start>'1-JAN-2020' AND DriverShift.Finish<'31-DEC-
2021';

--List all the order details of the orders that are made by
a
--walk-in customer with first name xxx and last name ttt between
--date yyy and zzz.

SELECT Customer.FirstName, Customer.LastName ,Orders.OrderNumber,
Orders.OrderStatus, Orders.OrderTime, Orders.OrderStatus,
Orders.AmountDue, Orders.PaymentApprovalNumber, Orders.PaymentMethod
FROM WalkInOrder
INNER JOIN Orders
ON Orders.OrderNumber=WalkInOrder.OrderNumber
INNER JOIN Customer
ON Customer.CustomerID=Orders.CustomerID
WHERE Customer.FirstName='Jane' AND Customer.LastName='Doe'
AND Orders.OrderTime > '1-JAN-2020' AND Orders.OrderTime<'31-DEC-2021';

--List all the order details of the orders that are taken by an
--in-office staff with first name xxx and last name ttt
between
--date yyy and zzz.

SELECT InStore.FirstName AS EmployeeFirstName, InStore.LastName AS
EmployeeLastName ,Orders.OrderNumber, Orders.OrderStatus,
Orders.OrderTime, Orders.OrderStatus, Orders.AmountDue,
Orders.PaymentApprovalNumber, Orders.PaymentMethod
FROM Orders
INNER JOIN InStore
ON InStore.EmployeeNumber=Orders.EmployeeNumber
WHERE InStore.FirstName='Jane' AND InStore.LastName='Doe'
AND Orders.OrderTime > '1-JAN-2020' AND Orders.OrderTime < '31-DEC-
2021';

--Print the salary paid to a delivery staff named xxx in the
--current month. Note the current month is the current month that
--is decided by the system

SELECT SUM(DriverPay.TotalPayment) AS 'Driver Pay Total'
FROM DriverShift
INNER JOIN DriverPay
ON DriverPay.PaymentID=DriverShift.PaymentID
INNER JOIN Driver
ON Driver.EmployeeNumber=DriverShift.EmployeeNumber
WHERE MONTH(DriverShift.Finish)=MONTH(GETDATE())
AND YEAR(DriverShift.Finish)=YEAR(GETDATE())
AND Driver.FirstName = 'John' AND Driver.LastName='Doe'

--List the name of the menu item that is mostly ordered in
--the current year. Note the current year is the current year that is
--decided by the system.

SELECT p.MenuItemCode,MenuItemName,p.orderamount FROM (
      SELECT y.MenuItemCode,SUM(y.Quantity) AS 'orderamount' FROM (
            SELECT
OrderItems.OrderNumber,OrderItems.MenuItemCode,OrderItems.Quantity
            FROM OrderItems -- get all the menu items ordered this year
```

```
            INNER JOIN Orders
            ON Orders.OrderNumber=OrderItems.OrderNumber
            WHERE YEAR(Orders.OrderTime) = YEAR(getDate())
      ) as y
      GROUP BY y.MenuItemCode
) AS p
INNER JOIN (
      SELECT MAX(h.orderamount) AS 'maxordamount' FROM (
            SELECT z.MenuItemCode,SUM(z.Quantity) AS 'orderamount' FROM
(
                  SELECT
OrderItems.OrderNumber,OrderItems.MenuItemCode,OrderItems.Quantity
                  FROM OrderItems -- get all the menu items ordered this
year
                  INNER JOIN Orders
                  ON Orders.OrderNumber=OrderItems.OrderNumber
                  WHERE YEAR(Orders.OrderTime) = YEAR(getDate())
            ) as z
            GROUP BY z.MenuItemCode
      ) AS h
) AS g
ON g.maxordamount=p.orderamount
INNER JOIN MenuItem
ON MenuItem.MenuItemCode=p.MenuItemCode;
```

```
DROP TABLE PhoneDeliveryOrder;

DROP TABLE PhonePickupOrder

DROP TABLE WalkInOrder;

DROP TABLE OrderItems;

DROP TABLE Orders;

DROP TABLE InStoreShift;

DROP TABLE DriverShift;

DROP TABLE DriverPay;

DROP TABLE InStorePay;

DROP TABLE Driver;

DROP TABLE InStore;

DROP TABLE Customer;

DROP TABLE MenuItemComposition;

DROP TABLE MenuItem;

DROP TABLE IngredientOrders;

DROP TABLE Ingredient;

DROP TABLE StockOrder;


--StockOrder(StockOrderNumber, DateOrdered, DateReceived, Status)

--Primary Key StockOrderNumber



CREATE TABLE StockOrder (

        StockOrderNumber        CHAR(10)        NOT NULL        PRIMARY KEY,

        DateOrdered                     DATE            NOT NULL,

        DateReceived            DATE,

        OrderStatus                     CHAR(2)                 NOT NULL        DEFAULT
        'PR',     --PR means purchased but not received

        CONSTRAINT dteChk               CHECK (DateReceived > DateOrdered),

);
```