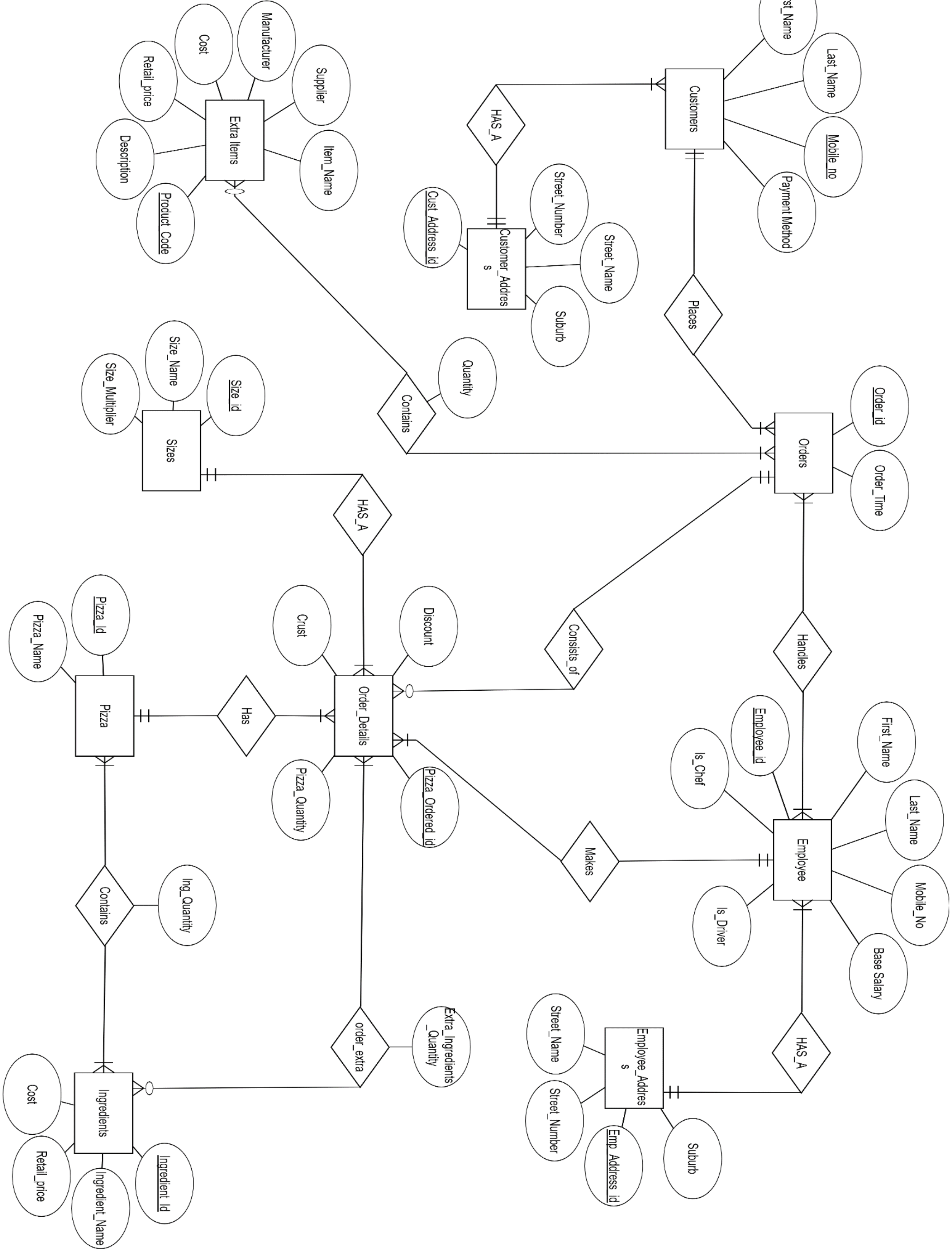
#TASK 1.a



Customers
First_Name
Last_Name
Mobile_no
Payment Method
HAS_A
Customer_Address
Cust_Address_id
Street_Number
Street_Name
Suburb
Places
Extra Items
Cost
Manufacturer
Supplier
Retail_price
Description
Item_Name
Product_Code
Contains
Quantity
Orders
Order_id
Order_Time
Sizes
Size_Name
Size_Multiplier
Size_id
HAS_A
Consists_of
Handles
Pizza
Pizza_Id
Pizza_Name
Has
Order_Details
Crust
Discount
Pizza_Quantity
Pizza_Ordered_id
Makes
Employee
Employee_id
Is_Chef
First_Name
Last_Name
Mobile_No
Is_Driver
Base Salary
Contains
Ing_Quantity
order_extra
Extra_Ingredients_Quantity
HAS_A
Employee_Address
Street_Name
Street_Number
Emp_Address_id
Suburb
Ingredients
Cost
Retail_price
Ingredient_Name
Ingredient_Id

#TASK 1.b

<u>Assumptions for Database:</u>

1. One employee will handle the cooking for a single type of pizza order and one employee will deliver the entire one order which may consist of many pizzas (They can be the same employee).
2. The discount is inputted manually by employee/customer into the system, hence allowing each customer to have a different discount rate, and this allows the businesses profits to be maximised rather than having a flat rate discount for different pizzas. This flat rate approach could potentially reduce profit margins by a large amount. The discount only applies to pre-prepared pizzas, and NOT the extra ingredients.
3. The profit function for the ingredients in this database is $Retail\_price - Cost$, which is used to justify ingredients profit by the number of sales of pizzas.
4. Every single ingredient in the ingredients table is used in at least one pizza. Therefore, for question 3d, it is assumed that ingredients not chosen by customers refers to extra ingredients which were not chosen by customers.
5. The company which this database is designed for, will not charge extra money for the crust type. They will however charge for different sizes of pizza. I have imposed this rule because in a realistic pizza shop, customers aren't usually charged extra for crust type, but rather the size of pizza they order.
6. In this pizza shop, pre-defined pizzas are available for sale, and customers can choose extra toppings if they would like. The price for a pizza is determined the ingredients which are in it.

#TASK 1.c

<u>Considerations for Database:</u>

I have decided to include an entity for customer and employee address separately (Customer_Address and Employee_Address) because this allows for easier differentiation between both a customer and an employee. This method also allows for an employee to be a customer, or vice versa. These address tables both have an artificial key associated with them, hence making it easier to create a foreign key within both the Customers and Employee tables. Furthermore, creating these two tables also helps to achieve third normal form (3NF). One customer address may be for many customers (e.g. two members in a family order item at two different times). Similarly, one employee address can be for many employees (e.g. two siblings living at the same house may work at the company). The orders entity connects to both the Customer entity and Employee entity. It has a many to many relationship with employee, hence requiring the formation of another entity within the middle. This table will take foreign keys from both Orders and Employee. The purpose for this table is to allocate an order to an employee to deliver as a driver, therefore making calculations for drivers' extra pay easier. Additionally, instead of creating IS_A attributes for the employee attribute, I decided to include Boolean attributes for cook and driver, hence making it easier to manipulate queries. No additional information was given about cook and driver, hence it seemed unnecessary to create an entire entity for it.

The Order entity had many to many relationships with both the Pizza and Extra_Items entities. Hence it was necessary to provide a table within the middle of these. Initially I had decided to only create the Order_Details entity with both Pizza and Extra_Items connected to it, however was faced with the issue of inputting null values into entries if a customer didn't want to order both a pizza and an extra. Hence, I created an additional table for Ordered_extras, which would prevent the entering of NULL entries to either table. The Order_Details entity is therefore entirely dedicated to the purchase of pizza related items. The Order_Details entity is connected to a Sizes entity. I had not initially considered to have a sizes table, but it made logical sense for the price for a pizza to be increased if a larger one was ordered. Hence the table contains a flat standard multiplying price for pizzas of different sizes. The Order_Details entity is also connected to the Pizza entity, containing pizza name information and a respective pizza_id, which in turn is also connected to the Ingredients entity with a many to many relationship. The table between these two entities, holds information regarding the ingredients for each different pizza, which supports the assumption made earlier. The Order_Details entity also contains information regarding the quantity of pizza a customer may want, as well as discount information and crust type. I decided to create an artificial key for the Order_Details table because it enables customers to add extra ingredients to specific pizzas instead adding them to all pizzas within the order. Additionally, the Order_Details table is connected to ingredients table, should a customer wish to add additional toppings to their pizza. I believe that this is a suitable solution for the problem at hand, as it allows for in-depth analysis of business issues such as revenue and profit from all areas/products via the use of SQL.

#TASK 2.a

3NF:

1. **Customer_address** (Customer_Address_id *INT(10)*, Street_No *INT(5)*, Street_Name *VARCHAR(40)*, Suburb *VARCHAR(40)*)
2. **Customers** (Mobile_No *CHAR(10)*, First_Name *VARCHAR(40)*, Last_Name *VARCHAR(40)*, Customer_Address_id *INT(10)*, Payment_Method *ENUM("Cash","Card","Cheque"))*
3. **Orders** (Order_id *INT(10)*, Order_time *TIMESTAMP*, Mobile_No *CHAR(10)*)
4. **Order_handled_by** (Order_id *INT(10)*, Employee_id *INT(10)*)
5. **Employee_address** (Employee_Address_id *INT(10)*, Street_No *INT(5)*, Street_Name *VARCHAR(40)*, Suburb *VARCHAR(40)*)
6. **Employee** (Employee_id *INT(10)*, Mobile_No *CHAR(10)*, First_Name *VARCHAR(40)*, Last_Name *VARCHAR(40)*, Employee_Address_id *INT(10)*, Is_Chef *BOOLEAN*, Is_Driver *BOOLEAN*, Base_Salary *FLOAT*)
7. **Order_details** (Pizza_Ordered_id *INT(10)*, Order_id *INT(10)*, Pizza_id *INT(10)*, Crust *ENUM("Thin","Regular","Thick")*, Size *INT(10)*, Pizza_Quantity *INT(2)*, Discount *INT(2)*, Employee_id *INT(10)*)
8. **Pizza** (Pizza_id *INT(10)*, Pizza_Name *VARCHAR(40)*)
9. **Sizes** (Size_Id *INT(10)*, Size_Type *ENUM("Small","Medium","Large","Family","Super")* , Size_Multiply_Price *FLOAT(5,2)*)
10. **Pizza_ingredients** (Pizza_id *INT(10)*, Ingredient_id *INT(2)*, Ingredients_Quantity *INT(2)*)
11. **Ingredients** (Ingredient_id *INT(2)*, Ingredient_Name *VARCHAR(20)*, Retail_Price *FLOAT*,Cost *FLOAT*)
12. **Order_extra_ingredients** (Pizza_Ordered_id *INT(10)*, Ingredient_id *INT(2)*, Extra_Ingredients_Quantity *INT(2)*)
13. **Extra_items** (Product_Code *INT(10)*, Item_Name *VARCHAR(20)*, Item_Description *VARCHAR(50)*, Manufacturer_Name *VARCHAR(30)*, Supplier *VARCHAR(30)*, Retail_price *FLOAT*, Cost *FLOAT*)
14. **Ordered_extras** (Order_id *INT(10)*, Product_Code *INT(10)*, Quantity, *INT(2)*)

Each many to many relationship within the ERD model, has a corresponding table created for it. The ERD model above, is in 3NF.

Decomposition process:

- Customer Entity
  - Initially, I had the customer entity with attributes for customer address. This however made the entity not in 2NF, therefore it was necessary to take out all the attributes relating to customer address and create a new table with those attributes and an artificial key for each record. Hence now, both the customer entity and the customer address entity are now both in 3NF because no partial dependencies exist.
- Order Entity
  - The order entity is already in 3NF, since all partial dependencies are eliminated.
- Employee Entity
  - The employee entity also faced the same issue as the customer entity with regards to address. The entity was not in 2NF, hence it was necessary to create a new table for employee address with the employee address attributes each with a unique artificial key for each record. Hence allowing both tables to now be in 3NF.
- Extras Entity

    - All the attributes for the extras entity functionally depend upon the primary key, hence the entity is in 3NF. It may be argued that a Manufacturer and Supplier table should be created, however since there is no additional information given relating to these two attributes, the entity is assumed to be in 3NF.
- Order details Entity
    - Initially, the Order details entity had a couple of attributes related to the size of a pizza. A single column for the size type and the associated multiplier price within another column. However, seeing that this table was not in 2NF due to this functional dependency, I decided to create a sizes table which contained an id for the size, with the size name and associated multiplier. Both the order details table and size table are now in 3NF.
- Pizza Entity
    - The pizza entity has two attributes and is already in 3NF since no partial dependencies exist.
- Ingredients Entity
    - The ingredients table is also in 3NF since no partial dependencies exist.

#TASK 2.b

Customer_Address Table

Name of Attribute	***Data Type***	***Description***
Customer_Address_id	INTEGER(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Street_No	INT(5)	Integer of length 5 for this attribute allows entering of high house numbers.
Street_Name	VARCHAR(40)	Different street names have different amount of characters, hence I decided to use VARCHAR of maximum length 40.
Suburb	VARCHAR(40)	Different suburbs have different amount of characters, hence I decided to use VARCHAR of maximum length 40.

Customer Table

Name of Attribute	***Data Type***	***Description***
Mobile_No	CHAR(10)	I used CHAR(10) because the maximum length of a mobile number is 10.
First_Name	VARCHAR(40)	Different first names have different amount of characters, hence I decided to use VARCHAR of maximum length 40.
Last_Name	VARCHAR(40)	Different last names have different amount of characters, hence I decided to use VARCHAR of maximum length 40.
Customer_Address_id	INT(10)	I decided to use integer as the datatype to match the datatype in parent table Customer_Address.
Payment_Method	ENUM("Cash","Card","Cheque")	Payment can only be by cash,card or cheque..

Orders

Name of Attribute	***Data Type***	***Description***
Order_id	INT(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Order_time	TIMESTAMP	I used timestamp as the data type because MYSQL automatically generated date and time data.
Mobile_No	CHAR(10)	Same data type as in parent table, Customer.

Employee_Address

Name of Attribute	***Data Type***	***Description***
Employee_Address_id	INTEGER(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Street_No	INT(5)	Integer of length 5 for this attribute allows entering of high house numbers.
Street_Name	VARCHAR(40)	Different street names have different amount of characters, hence I decided to use VARCHAR of maximum length 40.
Suburb	VARCHAR(40)	Different suburbs have different amount of characters, hence I decided to use VARCHAR of maximum length 40.

Employee

Name of Attribute	***Data Type***	***Description***
Employee_id	INT(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Mobile_No	CHAR(10)	I used CHAR(10) because the maximum length of a mobile number is 10.
First_Name	VARCHAR(40)	Different first names have different amount of characters, hence I decided to use VARCHAR of maximum length 40.
Last_Name	VARCHAR(40)	Different last names have different amount of characters, hence I decided to use VARCHAR of maximum length 40.
Employee_Address_id	INT(10)	I decided to use integer as the datatype to match the datatype in parent table Employee_Address.
Is_Chef	BOOLEAN	This attribute is boolean because formation of SQL queries is made easier. It is also easier to identify if employee is a chef or not.
Is_Driver	BOOLEAN	This attribute is boolean because formation of SQL queries is made easier. It is also easier to identify if employee is a driver or not.
Base_Salary	FLOAT	Certain employees may be able to have salaries which aren’t whole numbers. Therefore I have used FLOAT.

Order_Handled_By

Name of Attribute	***Data Type***	***Description***
Order_id	INT(10)	I decided to use integer as the datatype to match the datatype in parent table Orders.
Employee_id	INT(10)	I decided to use integer as the datatype to match the datatype in parent table Employee.

Pizza

Name of Attribute	***Data Type***	***Description***
Pizza_id	INT(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Pizza_Name	VARCHAR(40)	I decided to use varchar as the datatype because different pizzas can have different name lengths. Therefore this data type enables variability in length.

Extra_Items

Name of Attribute	***Data Type***	***Description***
Product_Code	INT(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Item_Name	VARCHAR(20)	I decided to use varchar as the datatype because different items can have different name lengths. Therefore this data type enables variability in length.
Item_Description	VARCHAR(50)	I decided to use varchar as the datatype because different items can have different description lengths. Therefore this data type enables variability in length.
Manufacturer_Name	VARCHAR(30)	I decided to use varchar as the datatype because different items can have different manufacturer name lengths. Therefore this data type enables variability in length.
Supplier	VARCHAR(30)	I decided to use varchar as the datatype because different items can have different supplier name lengths. Therefore this data type enables variability in length.
Retail_price	FLOAT	Certain items may have retail prices which aren’t whole numbers. Therefore, I have used FLOAT.
Cost	FLOAT	Certain items may have costs which aren’t whole numbers. Therefore, I have used FLOAT.

Ordered_Extras

Name of Attribute	***Data Type***	***Description***
Order_id	INT(10)	Parent table has same datatype
Product_Code	INT(10)	Parent table has same datatype
Quantity	INT(2)	Customer can't more than 99 of a single product. And quantity of something must be a whole number.

Ingredients

Name of Attribute	***Data Type***	***Description***
Ingredient_id	INT(2)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Ingredient_Name	VARCHAR(20)	Ingredients may have different name lengths
Retail_price	FLOAT	Certain ingredients may have retail prices which aren't whole numbers. Therefore, I have used FLOAT.
Cost	FLOAT	Certain ingredients may have costs which aren't whole numbers. Therefore, I have used FLOAT.

Sizes

Name of Attribute	***Data Type***	***Description***
Size_Id	INT(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Size_Type	ENUM("Small","Medium","Large","Family","Super")	Sizes can only be out of 5 options.
Size_Multiply_Price	FLOAT(5,2)	Can be of length 5, with 2 decimal places.

Order_Details

Name of Attribute	***Data Type***	***Description***
Pizza_Ordered_id	INT(10)	I decided to use integer as the datatype because it allows for auto incrementing and easier manipulation of data.
Order_id	INT(10)	Same as in parent table.
Pizza_id	INT(10)	Same as in parent table.
Crust	ENUM("Thin","Regular","Thick")	Crust can only be thin, regular or thick.
Size	INT(10)	Same as in parent table.
Pizza_Quantity	INT(2)	Number of pizzas selected can only be a whole number.
Discount	INT(2)	Customer can only have a whole number discount which is used for calculation in the queries.
Employee_id	INT(10)	Same as in parent table.

Order_Extra_Ingredients

Name of Attribute	***Data Type***	***Description***
Pizza_Ordered_id	INT(10)	Same as parent table
Ingredient_id	INT(2)	Same as parent table
Extra_Ingredients_Quantity	INT(2)	Customer can only choose whole number of extra ingredients.

Pizza_Ingredients

Name of Attribute	***Data Type***	***Description***
Pizza_id	INT(10)	Same as parent table
Ingredient_id	INT(2)	Same as parent table
Ingredients_Quantity	INT(2)	Customer can only choose whole number of extra ingredients.