**Object Description Document**

**Sequences**

1. Customer - customerID - auto-increment sequence for customerID starting from 1
2. Employee - employeeID - auto-increment sequence for employeeID starting from 1
3. Bill - billID auto-increment sequence for billID starting from 1
4. Orders - orderID - auto-increment sequence for orderID starting from 1

**Views**

1. customer\_view

* A view to show the customer ID, name, address, registered date, and customer type.
* Does not show other personal information like the phone#, fax#, email, date of birth and gender for privacy reasons.
* Access to this view would be given to those roles who need not all the personal details of the customer.

1. employee\_view

* A view to show the employee ID, name, email, and hire date. This prevents showing confidential information like salary and other personal information to others.

**Procedures**

1. insert\_customer

* Procedure to add new customers
* Accepts customer information like customer name, first and last name of the
* contact, address, email, whether the customer is a lead or not, number of employees in the company, registered date, comments, the type of customer it is, who referred them.
* It makes sure that the constraint checks and then inserts data into the table, else it raises an exception.

1. delivery\_schedule:

* Procedure to show the delivery schedule of a driver on a particular date
* Accepts the driver’s first and last name, and the date of interest as the parameters
* Checks if the parameters are valid and if they match with any data in the table. It will print the name of the route, customer name, order id, and order details in the output.
* Raises an exception if the input parameters are not correct.

1. create\_order:

* Procedure to place an order
* Takes the customer id, item ids and their quantities as input. If an item is not ordered its corresponding quantity would be marked as 0
* Checks if the customer id is valid
* Checks for the validity of the customer’s contract.
* Checks if the quantity on hand is enough to satisfy the entire order
* Creates order and enters data in orders and orderdetails table.
* Updates the quantity on hand attribute in item table by the subtracting the order quantity for each ordered item.
* Prints message on the success of the order.
* At each check if not valid than handled by proper exception.

**Triggers**

1. On updating the bill amount

* When the bill amount is updated due to a modification in the order or return of an order, it prints the old bill value, new bill value, and the difference in the values.

1. On returning an order

* When an order is returned (i.e. when the returnedby and returndate field in the orders table change/made not null), the bill amount for that particular order is updated and made negative to indicate that the full amount is returned back to the customer and the quantities are added back into to quantity on hand column of the item table.

**Job**

1. my\_job\_procedure

* A job to keep check the quantity of hand of each item in the item table so that if it goes below certain number, the company knows that it needs to order them from the supplier to avoid rejecting orders due to insufficient inventory.
* The threshold is 6000 quantities and if it goes below that, it is modified by adding 5000 units.

**Function**

1. Calculate late fee in bill

* Function to calculate the late fee in a bill
* Takes the billID as the input
* If the customer has not paid the bill before the due date, a late fee of 10% per month is calculated and stored in the bill which would be added to the total when the customer pays the bill eventually
* The late-fee is calculated as 10% of the bill amount per month. The number of days passed from the due date is calculated by using SYSDATE and then divided by 30 to calculate the number of months passed from the due date. This is further used for calculating the late-fee amount.

**Roles**

1. Accountant - select/create/update/delete only to BILL table
2. Delivery Manager - select/create/update/delete only to ROUTE table

**Package**

Package to hire and fire and employees. It has two procedures -

1. Hire employee - Takes the first name, last name, phone#, fax#, address, date of birth, role, gender, and email address as the input and inserts a row in the employee table
2. Fire employee - Takes the employeeID as the input and marks the end date as today’s date and nullifies the evaluation score

**Alternate Index**

1. Last name of employee - Since there would be a lot of data retrieval operations on the employee table, it would be beneficial to have an index on the “lname” column of the employee table to make retrieval faster.
2. Customer name - The customer name in the customer table is for the organization name. It can be used alternatively to the customerID and hence to make the information retrieval faster when joins are made, customer name has been made an alternate index.

**Instance of de-normalization**

The customer table has a field for customer type which is a value from a list of 5 allowed values such as Small business and Large corporation. In a normalized form, there would be a separate table for the customer type but that would unnecessarily increase the number of join operations and subsequently the time for a relatively small data. Hence, it has been left de-normalized to reduce time overhead.