Laundromat Management System

Group No. 145 Project ID: DBS_PR_07

Members:

Kshitij Garg	2020A7PS0120P
Chirag Maheshwari	2020A7PS0983P

System Requirement Specifications:

Back-End

1)MySQL: The project uses MySQL v8.0 for the database design and creation.
2) Node.js (1.0.0v) for making APIs and connecting MySQL (server-side scripting).
3)Express (4.17.3v) a back end web application framework for Node.js for building web applications and APIs.

Front-end

1) ReactJS: JavaScript library for building a single-page web app
2) Material UI: for making advanced react components
   Git: to collaborate smoothly throughout the project

Basic Assumptions About the System:

1) The laundromats deal with cloth bags and not with individual clothes. The delivery time/status of the wash cycle is believed to be unrelated to the type/quantity/individual characteristics of the clothes.
2) All bags take exactly 2 days for being washed (in a real scenario this would change on the basis of demand and the workforce available and would be handled by the individual laundromat managers).
3) Allocation of work to workers, bags to laundromat outlets, and bags to machines are done on the ground level by the outlet manager.
4) 'Deleting' a laundromat also means letting go of all employees working there, but the resources such as machines become free and are reallocated to the laundromat with LID 'L001'.

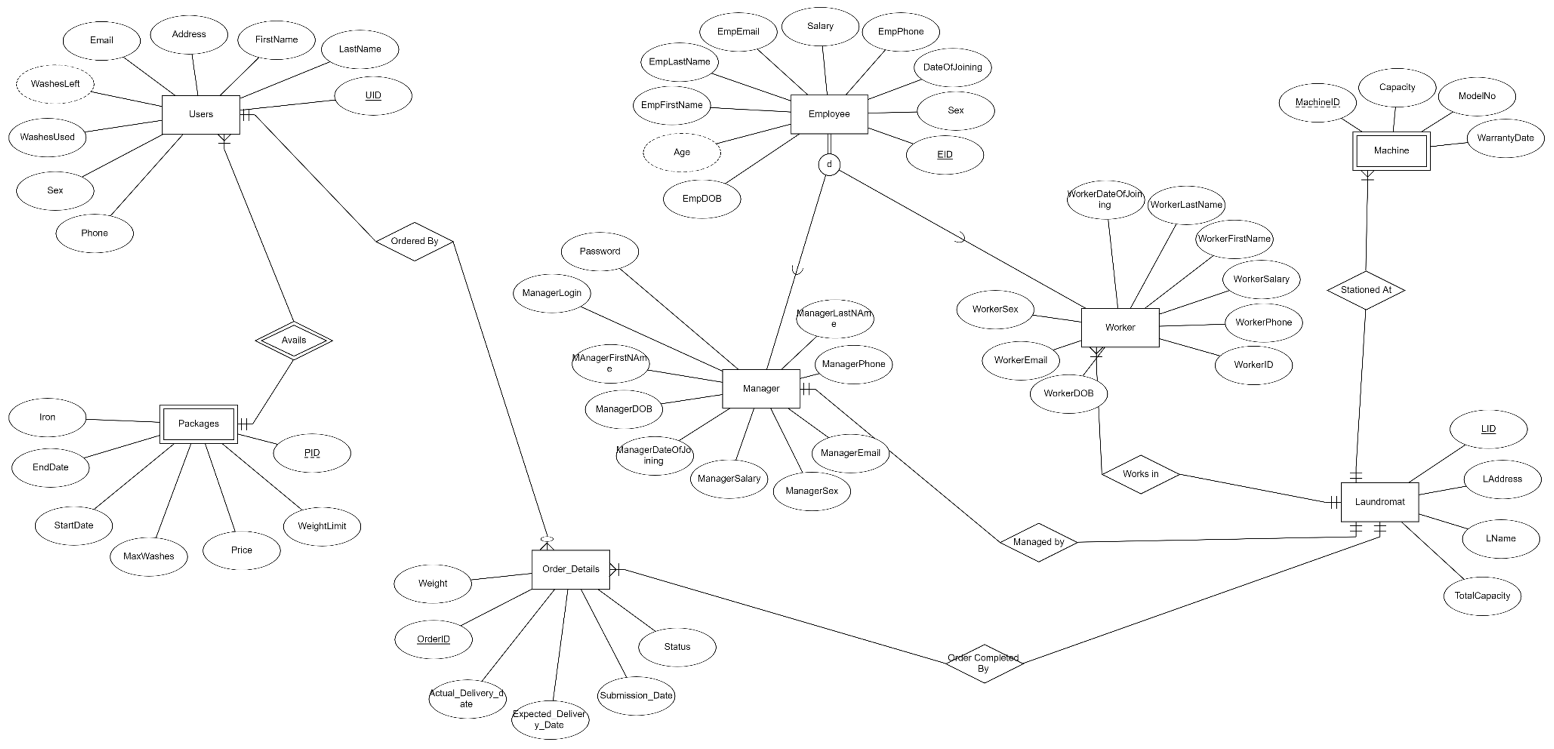
System Modelling:

Entity-Relationship Model:

The entity Relationship model was designed in two phases. The first phase developed a rough ER model which was further refined in the second phase by the removal of redundancy.

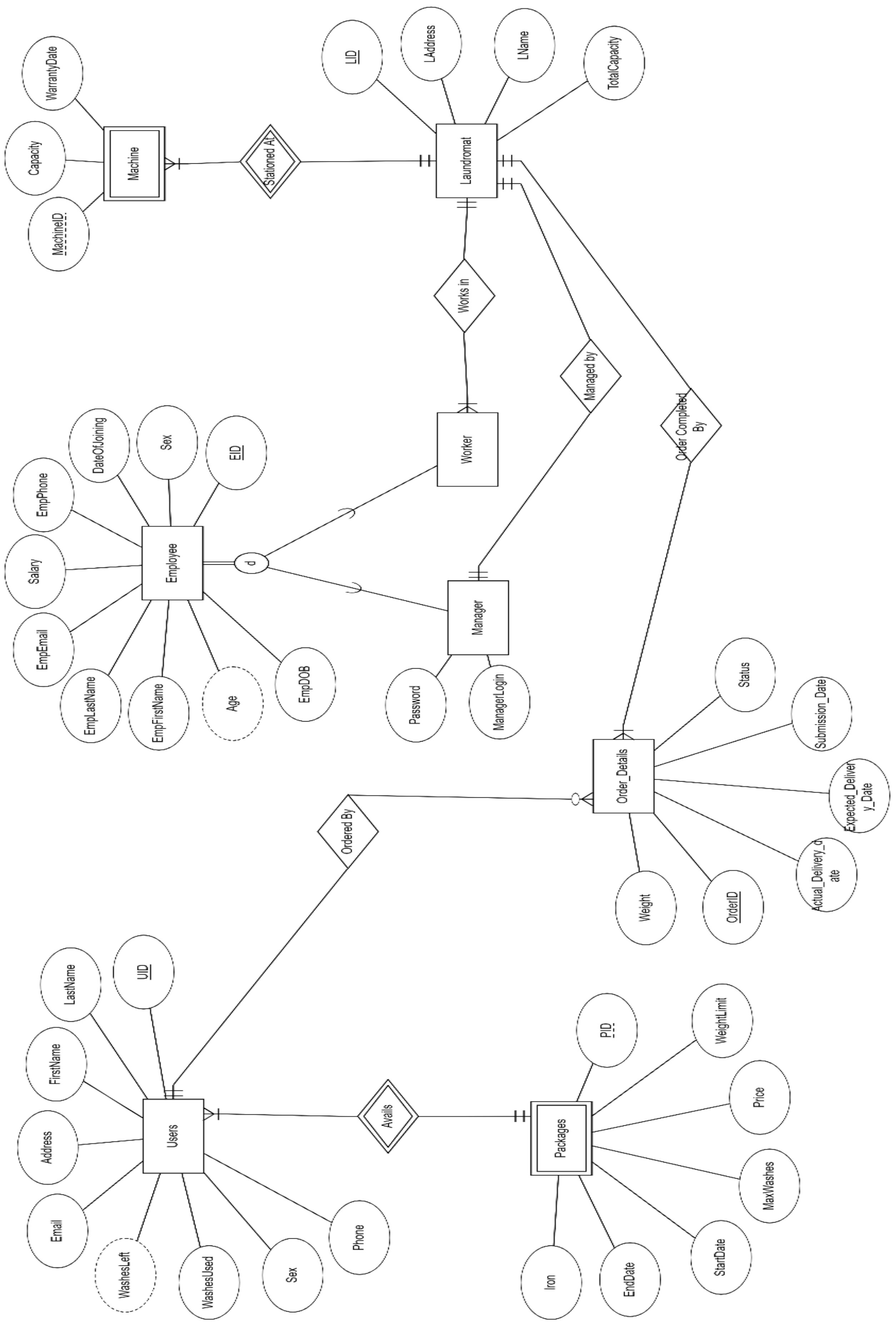
ER Diagram Phase 1

The diagram hosts a total of 8 entities of which 2 are weak and 6 are strong entities. The weak entities both have an identifying relationship with a strong entity denoted with the help of a double diamond. The attributes for different entities are marked within solid ovals, while the dotted ovals represent derived attributes. Unique attributes for strong entities are marked with a solid underline, while partial unique attributes are marked with a dotted underline in the case of weak entities.



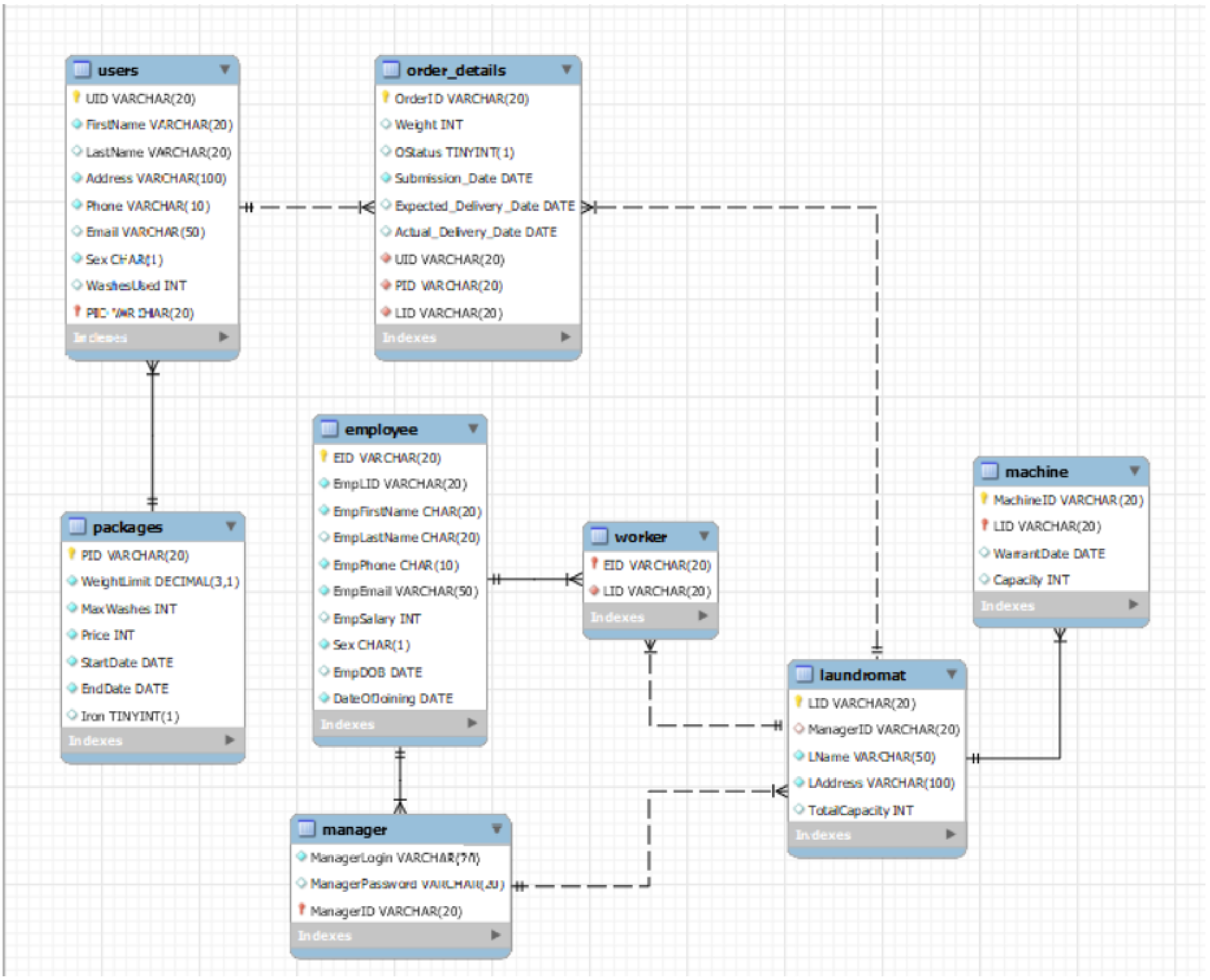
ER Diagram Phase 2

The Stationed At relationship was an identifying relationship for the machine entity, however, this was not depicted in the first diagram.
The sub-entity and the super-entity were defined incorrectly, and hence the attributes inside the ‘Worker’ and the ‘Manager’ entities are redundant. These redundant attributes were already present in the super-entity i.e. the employee entity. The redundant attributes were hence removed to yield the following final ER diagram:


WarrantyDate
Capacity
MachineID
Machine
Stationed At
Laundromat
LID
LAddress
LName
TotalCapacity
Works in
Managed by
Order Completed By
Worker
Manager
Employee
d
EmpPhone
DateOfJoining
Sex
EID
Salary
EmpEmail
EmpLastName
EmpFirstName
Age
EmpDOB
Password
ManagerLogin
Order_Details
Status
Submission_Date
Expected_Delivery_Date
Actual_Delivery_date
OrderID
Weight
Ordered By
Users
UID
LastName
FirstName
Address
Email
WashesLeft
WashesUsed
Sex
Phone
Avails
Packages
PID
WeightLimit
Price
MaxWashes
StartDate
EndDate
Iron

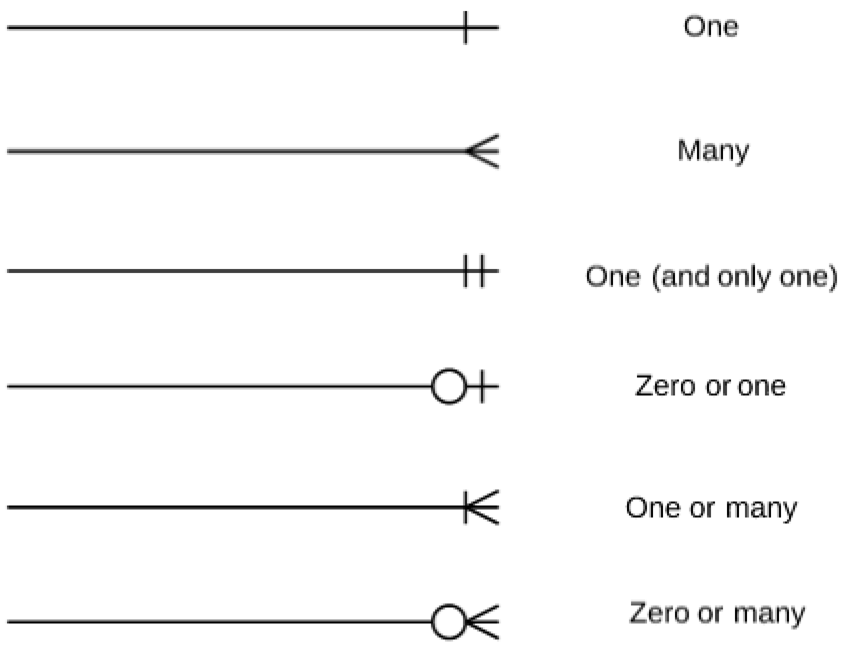
Schema Design:

The Schema was designed following the phase 2 ER diagram. The schema lays out the logical structure for the database and helps translate the data model into specific tables, columns, keys and interrelations.



The different tables were merged with the relationship tables from the ER diagram to reduce redundancy wherever possible, e.g. the identifying relationship ‘Avails’ is 1-N and has total participation from both the user and the packages entities and was hence merged into the users table. Similar steps for the reduction of redundancy were taken for the different relationships. The schema design follows the crow's foot notation.

Crow's foot notation:



Relational Model Stepwise:

Step1: Mapping of regular entity types
All the strong entity types are defined in relation where all simple attributes are taken into consideration and composite attributes' simple attributes are added to the relation (e.g. in Users composite attribute Name is added as FirstName and LastName).

Step 2: Mapping of Weak entity types
All weak entity types are defined in relation where all simple attributes are taken under consideration and composite attributes' simple attributes are added to the relation.

Step 3: Mapping of Binary 1:1 relation types
Since all 1:1 relations also followed complete participation, no separate table was used for them, they were merged with the entity tables to reduce redundancy. The primary key of one entity was included as the foreign key in the other entity and as the participation is complete, it is NOT NULL.

Step 4: Mapping of Binary 1:N relation types
Again as all 1:N relationships also had complete participation, it was possible to merge the relationship table within the entity table by including the primary key of the entity on the '1' side was added as the foreign key to the entity on the 'N' side.
e.g. 'Avails', 'Ordered by', 'Order Completed by', 'Works in' and 'Stationed at' relationships.

Step 5: Mapping of Binary M:N relation types
No such relationships exist in the ER Diagram.

Step 6: Mapping of multivalued attributes
No such relationships exist.

Step 7: Mapping of N-ary relationships
No such relationships exist.

Step 8: Mapping of Specialization
Employees entity is totally disjoint into two subclass types namely 'manager' and 'worker'. The two subclasses share most of their attributes which are therefore part of the superclass 'employee'. The subclass manager has 2 unique attributes namely ManagerLogin and ManagerPassword. The subclasses form 1:N binary relationships namely 'works at' and 'manages' with the 'laundromat' entity type. (the relationships have already been dealt with, in steps 3 and 4).

Data Normalization:

1NF :

The first normal form states that:

- Every column in the table must be unique
- Separate tables must be created for each set of related data
- Each table must be identified with a unique column or concatenated columns called the primary key
- No rows may be duplicated
- no columns may be duplicated
- no row/column intersections contain a null value
- no row/column intersections contain multivalued fields

Since all of the above conditions satisfy for all the generated relations, the database is in 1NF.

2NF :

A 1NF table is in 2NF form if and only if all of its non-prime attributes are functionally dependent on the whole of every candidate key. I.e there doesn't exist any partial dependency. Since the previous condition holds for all the generated relations, the database is already in 2NF.

3NF :

A relation already in 2NF is in 3NF if there is no transitive dependency for non-prime attributes. However since all attributes are functionally dependent on the primary key, another way of stating the previous condition is that no non-prime attribute should determine another non-prime attribute. In the relations such as users the attributes UID, Phone and Email are all candidate keys, however, UID has been chosen as the primary key. Therefore all the relations are already in 3NF.

BCNF :

3NF states that all data in a table must depend only on that table's primary key, and not on any other field in the table. A database table is in BCNF if and only if there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key. Since all functional dependencies for the generated relations are on candidate keys only, the database is also in BCNF.

There are no multivalued attributes in the database by design, and hence there is no scope for a multivalued functional dependency to even develop. Hence the presence of the database in higher normal forms is of no consequence here.

List Of Tables Required:

In the order in which they are made (order due to foreign key constraints)

a) **Packages Table**

This table holds the details about the different packages offered by the laundromat.

<u>PID</u> varchar(20)	WeightLimit INT	MaxWashes Int	Price Int	StartDate Date	EndDate Date	Iron Bool

b) **Employee Table**
This table holds the basic details about the employees working for the laundromat company.

EID Varchar (20)	EmpLID Varchar (20)	EmpFirst Name Varchar (20)	EmpLast Name Varchar (20)	EmpPhone Varchar (10)	EmpEmail Varchar (50)	EmpSalary Int	Sex Char (1)	Emp DOB Date	DateOfJoining Date

c) **Manager Table**
This table holds the details about the manager subclass of employee entity.

ManagerLogin Varchar(20)	ManagerPassword Varchar(20)	ManagerID Varchar(20)(Same as the EID)

d) **Laundromat Table**
This table holds the details about the different laundromat outlets.

LID Varchar(20)	ManagerID Varchar(20)	LName Varchar(20)	LAddress Varchar(100)	TotalCapacity Int

e) **Worker Table**
This table holds the EmployeeID and the LaundromatID for ‘worker’ employees for relating them to their respective laundromats.

EID Varchar(20)	LID Varchar(20)

f) **Machine Table**
This table holds the details about the different machines stationed at different laundromat outlets.

MachineID Varchar(20)	LID Varchar(20)	WarrantyDate Date	Capacity Int

g) **Users Table**
This table holds the basic details about the users and the packages they avail.

UID Varchar (20)	FirstName Varchar (20)	LastName Varchar (20)	Address Varchar (100)	Phone Varchar (10)	Email Varchar (50)	Sex Char(1)	WashesUsed Int	PID Varchar (20)

h) **Order_Details Table**
This table holds the details regarding the orders placed by the customers, ID of the user and the details of the laundromat fulfilling the order.

OrderID Varchar (20)	Weight Int	OStatus Bool	Submission_ Date Date	Expected_ Delivery_ Date Date	Actual_Delivery_ Date Date	UID Varchar (20)	PID Varchar (20)	LID Varchar (20)

Additional Components:
Procedures:

1) **add_order(IN orderID varchar(10), IN Weight int, IN UID varchar(10), IN PID varchar(10), IN LID varchar(10))**
The add_order procedure is responsible for adding new wash orders to the order_details table. It also updates the WashesUsed Attribute within the users table for the current user, after making sure that a series of conditions are met, including the availability of washes for the user, ensures that the given LID, PID and UID actually exist and that the weight of the bag given for washing is less than the permissible limit for the user's package.
The procedure throws a variety of errors depending on the violation. Moreover, the procedure automatically sets the Date_Of_Submission for the laundry bag as the current date and the Expected_Delivery_Date as two days after the current date.

2) **get_status(IN orderID varchar(10), IN UID varchar(10))**
The get_status procedure is responsible for getting the status of the laundry bag corresponding to the given 'OrderID' belonging to the user

with user ID ‘UID’. Again the procedure first ensures that a number of conditions are met.

Functions:

1) **is_overdue(orderID varchar(10)) RETURNS varchar(3)**
   The function takes in orderID as an attribute and returns whether or not the order is overdue. An order is said to be overdue if it hasn’t been delivered yet and the current date is > the expected date of delivery.

2) **count_overdue_days(orderID varchar(10)) RETURNS int**
   The function count_overdue_days takes orderID as a parameter and returns the number of days by which an order is overdue in the case that it is delayed. It returns the number of days by calling the datediff() function within itself and taking the difference between the currentDate() and the Expected_Delivery_Date.

Views:

1) **ManagerViewWorker**
   In this view the manager has access to all attributes of a worker from the employee table, including salary and DateOfJoining.
2) **ManagerViewManager**
   This view is more restrictive as compared to the ManagerViewWorker view. A manager cannot view another manager’s salary or DateOfJoining.

Concurrency and Consistency

The database achieves concurrency on a basic level by ensuring that all queries are atomic in nature, and hence act in a sense just like transactions, i.e are either executed fully or not at all. The queries are executed in a serialized manner hence removing the problem of inconsistency. All internal updates, like the increase of the no of washes used by a user, occur simultaneously and hence help in maintaining a consistent state for the complete database.
Internal Update/ Delete triggers like Cascading, Restricting and No Action were used to ensure the consistency of data as and where required. Hence Update and Delete Anomalies were taken care of using cases like

- Deleting an employee table entry also deleted the corresponding entry from the Manager/ Worker table.
- Deleting a Laundromat would also lead to the letting go of all employees that worked there (as was discussed in the assumptions about the

system), while the machines stationed there were reallocated to laundromat ‘L001’.
- Deleting a package that is currently being used by a User was forbidden by using ON DELETE RESTRICT, to ensure that packages were not dropped mid-term.

Future Improvements

- More complex transactions can be added in the future, complete with commits and rollbacks.
- Services like ‘Laundry on priority’ can be implemented.
- Streamlining of the laundry process, in terms of efficient allocation of washing machines, labour and other resources to minimize costs and the time for completing the tasks at hand.
- Use of Update/Delete/Insert Triggers can be made to ensure preconditions and reinforce consistency.
- More features like buy packages and get employee reports had been implemented, which could not be shown using the frontend at the time.