Advanced Databases - Hotel Management System

Oliver Favell, Marcel Krzyzanowski & Maryann Atakpa

A report submitted for the Advanced Databases module
for MSc Big Data Analytics at Birmingham City University

January 2021

Contents

List of Figures

Chapter 1

Introduction

1.1 Chapter List & Description

Chapter 2 Conceptual Design, outlines the domain rules, business requirements, assumptions, entities and relationships within the database system.

Chapter 3 Logical Design, showcases the Entity-Relationship Diagram (ERD) and normalisation stages.

Chapter 4 Database Implementation, showcases the Database Schema and how various SQL queries were used to create/populate tables with sample data.

Chapter 5 SQL Testing Queries and Optimisation, showcases a set of testing queries created by each individual in order to test system's technical capabilities as well as explore various optimisation techniques.

Chapter 6 Conclusion includes summary of any major outcomes and discoveries while undergoing the project as well as future work recommendations.

Chapter 2

Conceptual Design

2.1 Domain Description

The chosen project domain is a Hotel Management System; designed for storing employee and customer information, alongside orders, bookings, facility and department information.

Customer personal and contact information is kept on the system. Customers can make orders for food and facility access as well as book rooms and parking spaces. Each booking or order linked to a customer can be used to calculate their bills based on the total cost.

Employee information such as employee role, department, supervised employees, salary and contact information are logged on the system. This can be used to keep track of the employees within the hotel and each of the different departments, as well as track manager and subordinate information.

Information regarding facilities and their entry fees including parking and grounds facilities as well as opening and closing times are also stored within the system. Food is available for customers to order, with different meals, courses and prices for each entry on the menu. Room information including the size, price and en-suite option are also stored for booking and billing purposes.

2.2 Assumptions

Below are outlined the identified business rules and assumptions that the database system is build upon; this relates to how customers and employees can interact with the system and what information is required to be stored within the system.

- You must be a customer at the hotel in order to park
- A customer may not require parking.
- You must be a customer to access hotel facilities.
- Employees can be standard employees who have managers, or be managers of other employees.
- An employee can work in many departments.
- An employee can manage multiple people.
- An employee can manage multiple departments.
- An employee can have multiple phone numbers.
- An employee can have multiple email addresses.
- A department can have multiple managers.
- A customer can make multiple orders and bookings.
- A customer can book multiple rooms using the booking system.
- A customer can have multiple phone numbers.
- A customer can have multiple email addresses.
- A booking must always have a start and end time.
- A department has multiple facilities.
- A facility has only one department.

2.3 Entities and Attributes

This section details the list of identified business entities along side with their attributes to be stored within the database system. This list details primary keys and other special attributes where required.

2.3.1 Manager / Employee - Recursive

- Employee ID - Primary Key
- Name
- Role
- Salary
- Email - Multi-valued Attribute
- Phone - Multi-valued Attribute

2.3.2 Department

- Department Id - Primary Key
- Name
- Description
- Facilities - Multi-valued Attribute

2.3.3 Customer

- Customer Id - Primary Key
- Name
- Email - Multi-valued Attribute
- Phone - Multi-valued Attribute
- Total Bill

2.3.4 Room

- RoomId - Primary Key
- BookingId - Foreign Key
- Booking price
- Size

- Ensuite

2.3.5 Food Course

- Course Number - Primary Key
- Name
- Description
- Type
- Price

2.3.6 Facility

- FacilityId - Primary Key
- Name
- Description
- Schedule - Composite Attribute
  - Opening Time
  - Closing Time
- Entry cost

2.3.7 Order

- Order Number - Primary Key
- CustomerId - Foreign Key
- Total cost
- Description

2.3.8 Booking

- BookingId - Primary Key
- CustomerId - Foreign Key
- Duration - Composite Attribute
  - Start Time
  - End Time
- Total Cost

2.3.9 Parking

- Parking Id - Primary Key
- BookingId - Foreign Key
- Space Size
- Cost
- Duration

2.4 Relationships and Cardinality

The relationships between entities, cardinality and context of each relationship is summarised below to demonstrate how entities interact with related entities within the system.

- One employee supervises many employees (1:N)
- Many employees work in many departments (M:N)
- One department maintains many facilities (1:N)
- One customer can make many bookings (1:N)
- One customer can make many orders (1:N)
- One room can have many bookings (1:N)
- Many food courses can have many orders (M:N)
- Many orders can be placed for many facilities (M:N)
- One parking space can have many bookings (1:N)

Chapter 3

Logical Design

Below is the ERD diagram which represents different entities and their attributes. ERD demonstrates various relationships between the entities indicating how they will interact with each other within the database system. All primary keys and special entity types has been also specified as appropriate.

3.1 Entity Relationship Diagram

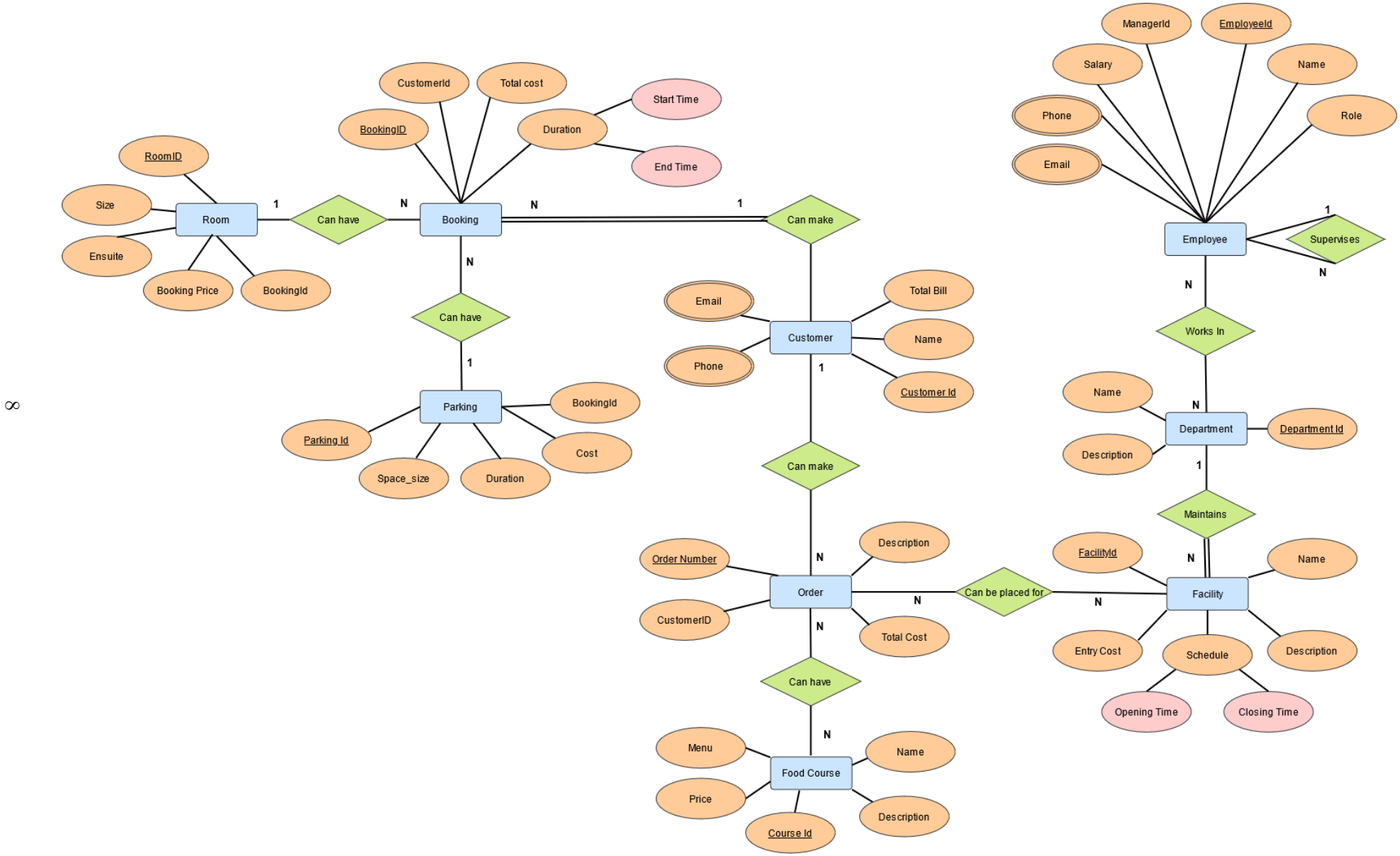

Figure 3.1: Entity Relationship Diagram

3.2 Database Normalisation

Normalisation was carried out to prevent duplicate information from causing data anomalies within the final database system; the ERD Figure 3.1 was transformed into a normalised database schema through the processes of 1st, 2nd and 3rd normal form.

3.2.1 1st Normal Form (1NF)

Composite attributes for Booking duration and Facility schedule were split into independent attributes within their respective entities.

Booking	
PK	Booking_ID int(10) NOT NULL
FK1	Customer_Id int(10) NOT NULL
	Total_cost VARCHAR(10) NOT NULL
	Duration_Start_Time VARCHAR(25) NOT NULl
	Duration_End_Time VARCHAR(25) NOT NULL

Figure 3.2: Booking Entity

Facility	
PK	Facility_ID int(10) NOT NULL
	Name char(50) NOT NULL
	Description char(100) NOT NULL
	Schedule_Opening_time VARCHAR(5) NOT NULL
	Schedule_closing_time VARCHAR(5) NOT NULL
	Entry_cost VARCHAR(10) NOT NULL

Figure 3.3: Facility Entity

The Phone and Email attributes for the Customer and Employee entities were separated into additional tables to allow multiple phone number and email values to be stored for the same customer and employee. The primary key of each entity is used as a foreign key in the new tables in order to connect an instance to related contact details.

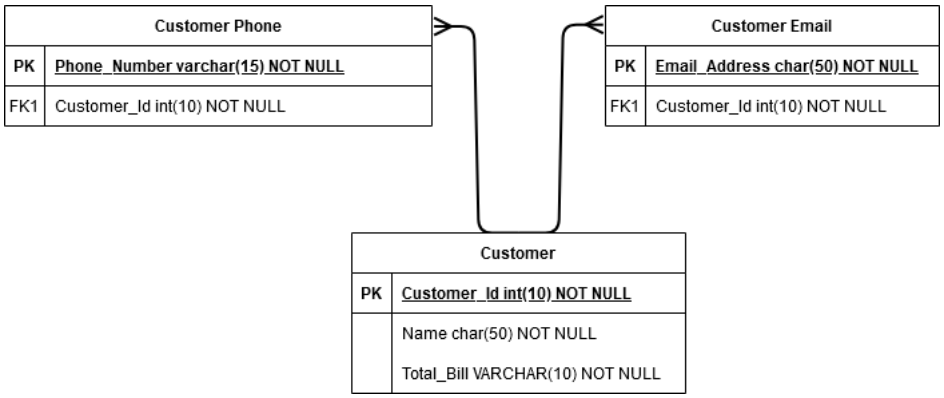

Figure 3.4: Customer Table with Additional Tables for Phone and Email

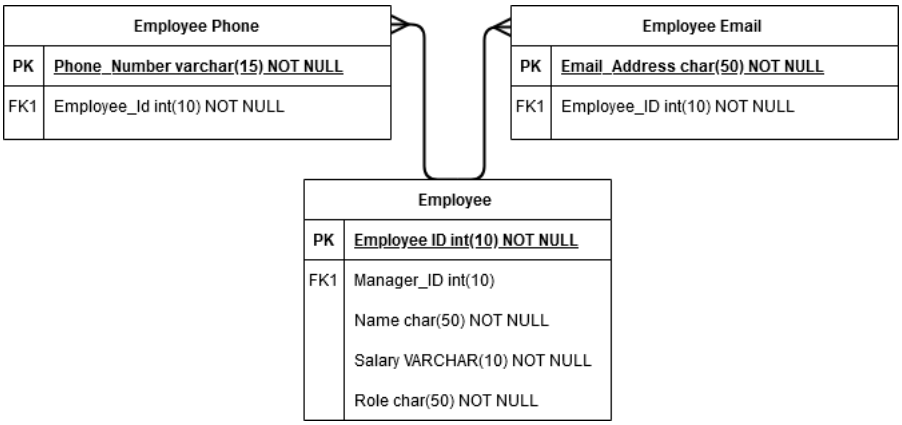

Figure 3.5: Employee Table with Additional Tables for Phone and Email

After these changes being completed, there are no attributes that do not have a single atomic value.

3.2.2 2nd Normal Form (2NF)

Each entity was checked to ensure that there were no non-prime attributes that did not depend on the primary key. All entities were found to fit this constraint and can therefore be considered to meet 2NF.

3.2.3 3rd Normal Form (3NF)

Link tables with foreign keys were created to link the many-to-many relationships together. Each entity only depends on its primary key and can use the corresponding link table to retrieve information about another linked entity.

As an employee can work in many departments, and a department can have many employees working within it, a link table is used to represent which departments an employee works in.

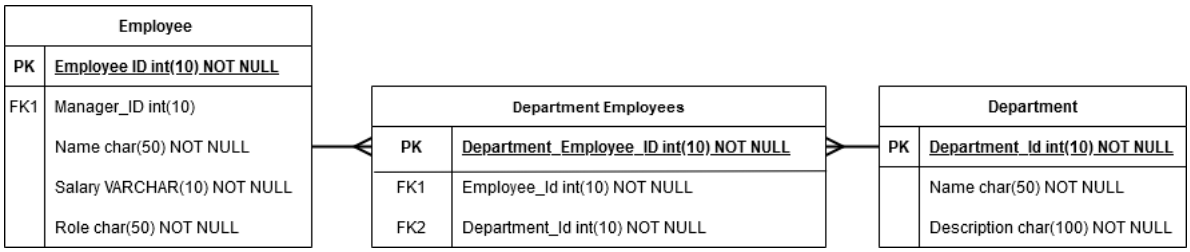

Figure 3.6: Link table between Employee and Department

A link table was created between Department and Facility as one department contains several facilities, and a facility always belongs to a department; a lookup table is therefore required to associate two instances of these entities together without violating the 2NF.

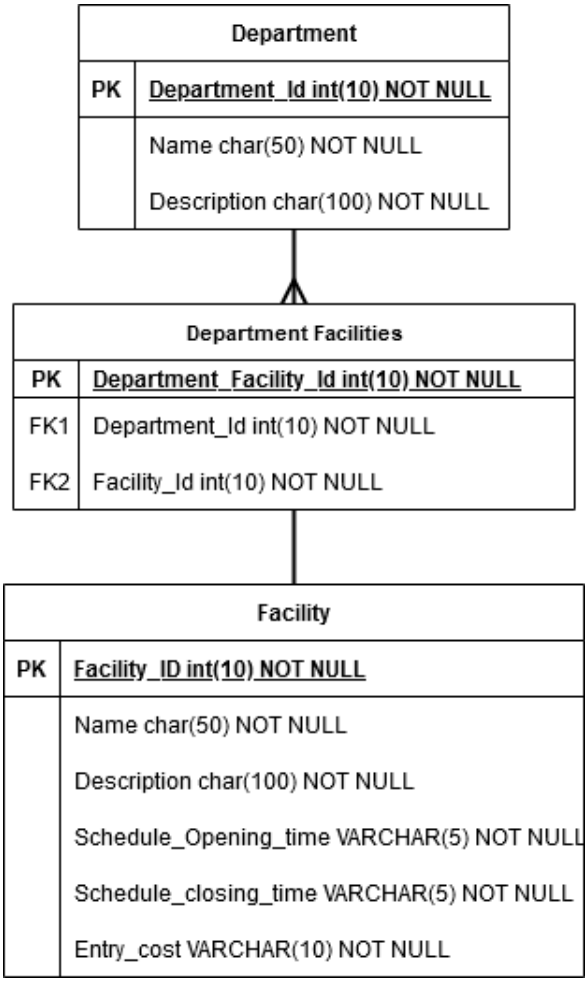

Figure 3.7: Link table between Department and Facility

Chapter 4

Database Implementation

4.1 Database Schema

The fully normalised database schema which the database system will be based upon can be found in Figure 4.1. This schema contains link tables, de-composed composite attribute tables and fully normalised entities.

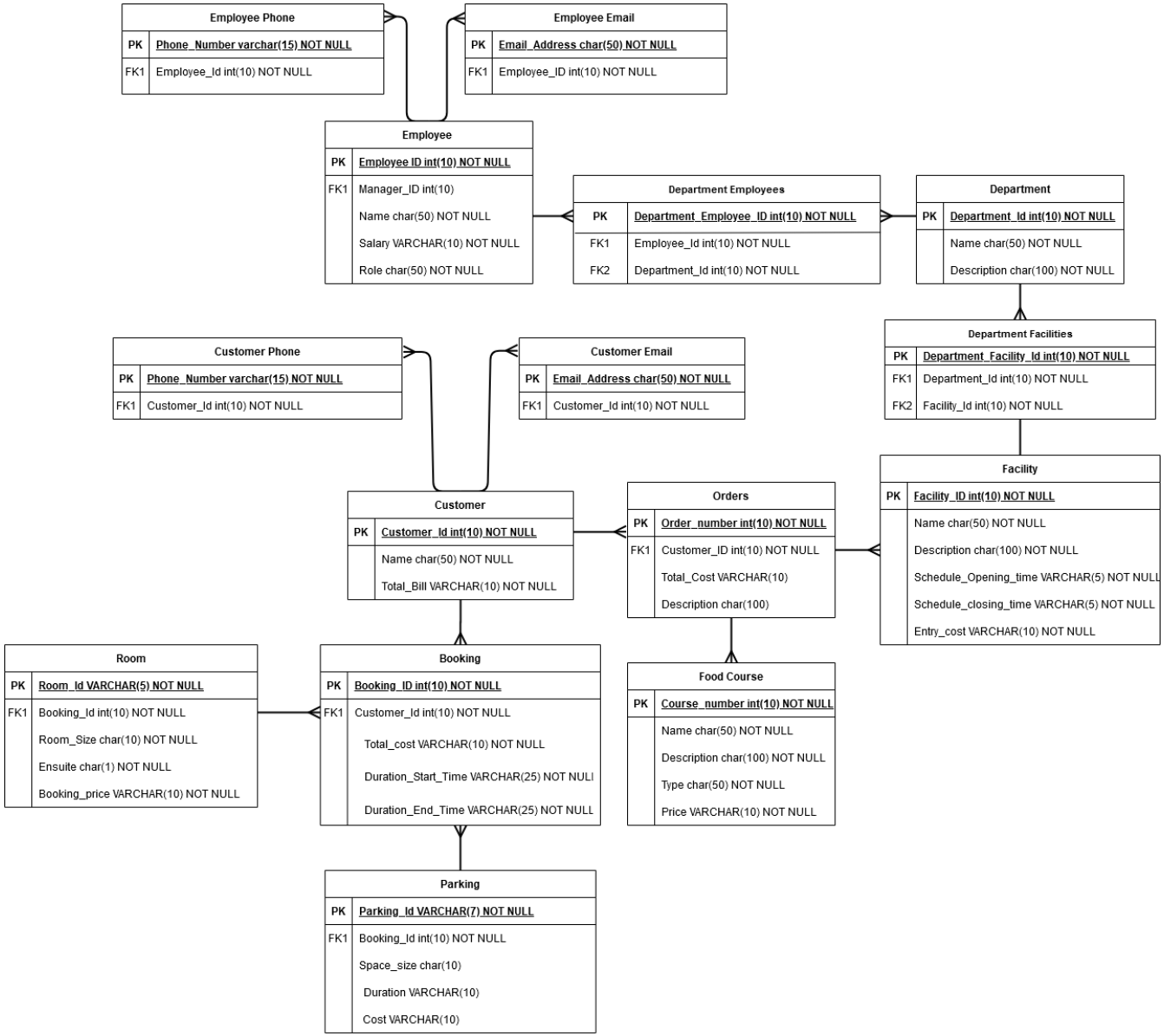

Figure 4.1: Normalised Database Table Schema

4.2 Database Creation Queries

Following section of the report will focus on the aspects regarding transforming the logical design into a working system via utilisation of various SQL queries. First sub-section will cover creation of the tables including constrains and attributes. Second part will focus on the queries which allowed to populate the tables with testing data. Last sub-section will cover testing of various constrains in order to test system's capabilities of identifying and capturing bad data.

4.2.1 Tables Creation

This section will showcase queries used for creating tables for the entities and their attributes.

Employee Table Creation

```
CREATE TABLE Employee (
Employee_ID int NOT NULL,
Manager_ID int NOT NULL UNIQUE,
Name char(50) NOT NULL,
Salary varchar(10) NOT NULL,
Role char(50) NOT NULL,

CONSTRAINT PK_Employee PRIMARY KEY (Employee_ID),
FOREIGN KEY (Manager_ID) REFERENCES Employee(Manager_ID))
```

Employee Phone Table Creation

```
CREATE TABLE Employee_Phone (
Phone_number varchar(15) NOT NULL PRIMARY KEY,
Employee_ID int NOT NULL,

FOREIGN KEY (Employee_ID) REFERENCES Employee(Employee_ID))
```

Employee Email Table Creation

```
CREATE TABLE Employee_Email (
Email_address char(50) NOT NULL PRIMARY KEY,
Employee_ID int NOT NULL,

FOREIGN KEY (Employee_ID) REFERENCES Employee(Employee_ID))
```

Department Employees Table Creation

```
CREATE TABLE Department_Employees (
Department_Employee_ID int NOT NULL PRIMARY KEY,
Employee_ID int NOT NULL,
Department_ID int NOT NULL,

FOREIGN KEY (Employee_ID) REFERENCES Employee(Employee_ID))
```

Department Table Creation

```
CREATE TABLE Department (
Department_ID int NOT NULL PRIMARY KEY,
Name char(50) NOT NULL,
Description char(100) NOT NULL)
```

Facility Table Creation

```
CREATE TABLE Facility (
Facility_ID int NOT NULL PRIMARY KEY,
Name char(50) NOT NULL,
Description char(100) NOT NULL,
Schedule_Opening_Time varchar(5) NOT NULL,
Schedule_Closing_Time varchar(5) NOT NULL,
Entry_Cost varchar(10) NOT NULL)
```

Department Facilities Table Creation

```
CREATE TABLE Department_Facilities (
Department_Facility_ID int NOT NULL PRIMARY KEY,
Department_ID int NOT NULL,
Facility_ID int NOT NULL,

FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID),
FOREIGN KEY (Facility_ID) REFERENCES Facility(Facility_ID))
```

Orders Table Creation

```
CREATE TABLE Orders (
Order_Number int NOT NULL PRIMARY KEY,
Customer_ID int NOT NULL,
Total_Cost varchar(10),
Description char(100),

FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID))
```

Customer Table Creation

```
CREATE TABLE Customer (
Customer_ID int NOT NULL PRIMARY KEY,
Name char(50) NOT NULL,
Total_Bill varchar(10) NOT NULL)
```

Customer Email Table Creation

```
CREATE TABLE Customer_Email (
Email_address char(50) NOT NULL PRIMARY KEY,
Customer_ID int NOT NULL,

FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID))
```

Customer Phone Table Creation

```
CREATE TABLE Customer_Phone (
Phone_number varchar(15) NOT NULL PRIMARY KEY,
Customer_ID int NOT NULL,

FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID))
```

Food Course Table Creation

```
CREATE TABLE Food_Course (
Course_Number int NOT NULL PRIMARY KEY,
Name char(50) NOT NULL,
Description char(100) NOT NULL,
Type char(100) NOT NULL,
Price varchar(10) NOT NULL)
```

Booking Table Creation

```
CREATE TABLE Booking (
Booking_ID int NOT NULL PRIMARY KEY,
Customer_ID int NOT NULL,
Total_Cost varchar(10) NOT NULL,
Duration_Start_Time varchar(25) NOT NULL,
Duration_End_Time varchar(25) NOT NULL,

FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID))
```

Parking Table Creation

```
CREATE TABLE Parking (
Parking_ID varchar(7) NOT NULL PRIMARY KEY,
Booking_ID int NOT NULL,
Space_Size char(10),
Duration varchar(10),
Cost varchar(10),

FOREIGN KEY (Booking_ID) REFERENCES Booking(Booking_ID))
```

Room Table Creation

```
CREATE TABLE Room (
Room_ID varchar(5) NOT NULL PRIMARY KEY,
Booking_ID int NOT NULL,
Room_Size char(10) NOT NULL,
Ensuite char(1) NOT NULL,
Booking_Price varchar(10) NOT NULL,

FOREIGN KEY (Booking_ID) REFERENCES Booking(Booking_ID))
```

4.2.2 Data Insertion

This section will showcase queries used for populating previously created tables with testing data.

Employee Table Population

```
INSERT INTO Employee (Employee_ID,Manager_ID,Name,Salary,Role)
VALUES ('201','1','John','20000','Hotel Manager');
INSERT INTO Employee (Employee_ID,Manager_ID,Name,Salary,Role)
VALUES ('202','2','Esther','15000','Sales Manager');
INSERT INTO Employee (Employee_ID,Manager_ID,Name,Salary,Role)
VALUES ('203','3','Pablo','10000','Head Chef');
INSERT INTO Employee (Employee_ID,Manager_ID,Name,Salary,Role)
VALUES ('204','202','Patricia','12500','Accountant');
INSERT INTO Employee (Employee_ID,Manager_ID,Name,Salary,Role)
VALUES ('205','201','Kate','10000','Front Desk Manager');
INSERT INTO Employee (Employee_ID,Manager_ID,Name,Salary,Role)
VALUES ('206','206','Thomas','8000','Food Deliveries Courier');
INSERT INTO Employee (Employee_ID,Manager_ID,Name,Salary,Role)
VALUES ('207','203','Jacob','12000','Sous Chef');
```

Department Table Population

```
1  INSERT INTO Department (Department_ID,Name,Description)
2  VALUES ('001','Front Office','Handles reservation, reception,
       registration, room assignment, bills');
3  INSERT INTO Department (Department_ID,Name,Description)
4  VALUES ('002','Human Resources','responsible for the recruiting,
       utilisation, training of employees');
5  INSERT INTO Department (Department_ID,Name,Description)
6  VALUES ('003','Sales and Marketing','responsible for increasing
       the sales of the hotels products and services');
7  INSERT INTO Department (Department_ID,Name,Description)
8  VALUES ('004','Catering','responsible for service of food and
       drinks to guests');
9  INSERT INTO Department (Department_ID,Name,Description)
10 VALUES ('005','Finance','responsible for all the financial
       transactions in the hotel');
11 INSERT INTO Department (Department_ID,Name,Description)
12 VALUES ('006','Deliveries','responsible for all deliveries to
       hotel premises');
```

Department Employees Table Population

```
INSERT INTO Department_Employees (Department_employee_ID,
    Employee_ID,Department_ID)
VALUES ('101', '201','002');
INSERT INTO Department_Employees (Department_employee_ID,
    Employee_ID,Department_ID)
VALUES ('102', '202','003');
INSERT INTO Department_Employees (Department_employee_ID,
    Employee_ID,Department_ID)
VALUES ('103','203','004');
INSERT INTO Department_Employees (Department_employee_ID,
    Employee_ID,Department_ID)
VALUES ('104', '204','005');
INSERT INTO Department_Employees (Department_employee_ID,
    Employee_ID,Department_ID)
VALUES ('105','205','001');
INSERT INTO Department_Employees (Department_employee_ID,
    Employee_ID,Department_ID)
VALUES ('106','206','006');
INSERT INTO Department_Employees (Department_employee_ID,
    Employee_ID,Department_ID)
VALUES ('107','207','004');
```

Employee Phone Table Population

```
INSERT INTO Employee_phone (Phone_Number,Employee_ID)
VALUES ('4409567823547','201');
INSERT INTO Employee_phone (Phone_Number,Employee_ID)
VALUES ('4475871864796','205');
INSERT INTO Employee_phone (Phone_Number,Employee_ID)
VALUES ('4407457612459','202');
INSERT INTO Employee_phone (Phone_Number,Employee_ID)
VALUES ('4407679853268','202');
INSERT INTO Employee_phone (Phone_Number,Employee_ID)
VALUES ('4475871309452','204');
```

Employee Email Table Population

```
INSERT INTO Employee_email (Email_address,Employee_ID)
VALUES ('johnero52@gmail.com','201');
INSERT INTO Employee_email (Email_address,Employee_ID)
VALUES ('Estherbanks@gmail.com','202');
INSERT INTO Employee_email (Email_address,Employee_ID)
VALUES ('triciathompson@yahoo.com','204');
INSERT INTO Employee_email (Email_address,Employee_ID)
VALUES ('johnHR@yahoo.com','201');
INSERT INTO Employee_email (Email_address,Employee_ID)
VALUES ('bookwithkate@yahoo.com','205');
```

Facility Table Population

```
INSERT INTO Facility (Facility_ID,Name,Description,
    Schedule_Opening_Time, Schedule_Closing_Time,Entry_Cost)
VALUES ('501','Parking','Car parking space for customers', '0:00'
    , '23:59','Free');
INSERT INTO Facility (Facility_ID,Name,Description,
    Schedule_Opening_Time, Schedule_Closing_Time,Entry_Cost)
VALUES ('502','Swimming Pool','Swimming as a leisure activity', '
    12:00', '22:00','£10');
INSERT INTO Facility (Facility_ID,Name,Description,
    Schedule_Opening_Time, Schedule_Closing_Time,Entry_Cost)
VALUES ('503','Conference Hall','Room is provided for events', '
    10:00', '20:00','£50');
INSERT INTO Facility (Facility_ID,Name,Description,
    Schedule_Opening_Time, Schedule_Closing_Time,Entry_Cost)
VALUES ('504','Gym','Workout 24/7', '0:00', '24:00','£30');
INSERT INTO Facility (Facility_ID,Name,Description,
    Schedule_Opening_Time, Schedule_Closing_Time,Entry_Cost)
VALUES ('505','Outdoor Catering Service','Provision of food
    outside the restaurant', '8:00', '21:00','£100');
```

Department Facilities Table Population

```
INSERT INTO Department_Facilities (Department_Facility_ID,
    Department_ID,Facility_ID)
VALUES ('301','001','501');
INSERT INTO Department_Facilities (Department_Facility_ID,
    Department_ID,Facility_ID)
VALUES ('302','001','502');
INSERT INTO Department_Facilities (Department_Facility_ID,
    Department_ID,Facility_ID)
VALUES ('303','001','503');
INSERT INTO Department_Facilities (Department_Facility_ID,
    Department_ID,Facility_ID)
VALUES ('304','001','504');
INSERT INTO Department_Facilities (Department_Facility_ID,
    Department_ID,Facility_ID)
VALUES ('305','004','505');
```

Customer Table Population

```
INSERT INTO Customer (Customer_ID,Name,Total_Bill)
VALUES ('111','Frank','£500');
INSERT INTO Customer (Customer_ID,Name,Total_Bill)
VALUES ('112','Betty','£530');
INSERT INTO Customer (Customer_ID,Name,Total_Bill)
VALUES ('113','Ronald','£400');
INSERT INTO Customer (Customer_ID,Name,Total_Bill)
VALUES ('114','Steve','£650');
INSERT INTO Customer (Customer_ID,Name,Total_Bill)
VALUES ('115','Veronica','£800');
```

Customer Phone Table Population

```
INSERT INTO Customer_Phone (Phone_number,Customer_ID)
VALUES ('4467438723095','111');
INSERT INTO Customer_Phone (Phone_number,Customer_ID)
VALUES ('4467438723765','111');
INSERT INTO Customer_Phone (Phone_number,Customer_ID)
VALUES ('4467498723785','113');
INSERT INTO Customer_Phone (Phone_number,Customer_ID)
VALUES ('4409749887051','115');
INSERT INTO Customer_Phone (Phone_number,Customer_ID)
VALUES ('4409747687432','115');
```

Customer Email Table Population

```
INSERT INTO Customer_Email (Email_address,Customer_ID)
VALUES ('bettyking@gmail.com','112');
INSERT INTO Customer_Email (Email_address,Customer_ID)
VALUES ('stevejobs@icloud.com','114');
INSERT INTO Customer_Email (Email_address,Customer_ID)
VALUES ('stevie5@gmail.com','114');
INSERT INTO Customer_Email (Email_address,Customer_ID)
VALUES ('ronnie@gmail.com','115');
INSERT INTO Customer_Email (Email_address,Customer_ID)
VALUES ('ronaldo4real@yahoo.com','113');
```

Booking Table Population

```
INSERT INTO Booking (Booking_ID,Customer_ID,Total_Cost,
    Duration_Start_Time,Duration_End_Time)
VALUES ('221','111','£300','8:00','17:30');
INSERT INTO Booking (Booking_ID,Customer_ID,Total_Cost,
    Duration_Start_Time,Duration_End_Time)
VALUES ('222','112','£250','9:00','17:00');
INSERT INTO Booking (Booking_ID,Customer_ID,Total_Cost,
    Duration_Start_Time,Duration_End_Time)
VALUES ('223','113','£220','9:00','18:30');
INSERT INTO Booking (Booking_ID,Customer_ID,Total_Cost,
    Duration_Start_Time,Duration_End_Time)
VALUES ('224','114','£500','8:00','20:00');
INSERT INTO Booking (Booking_ID,Customer_ID,Total_Cost,
    Duration_Start_Time,Duration_End_Time)
VALUES ('225','115','£700','8:00','21:30');
INSERT INTO Booking (Booking_ID,Customer_ID,Total_Cost,
    Duration_Start_Time,Duration_End_Time)
VALUES ('226','111','£450','10:00','18:30')
```

Parking Table Population

```
INSERT INTO Parking (Parking_ID, Booking_ID,Space_Size,Duration,
    Cost)
VALUES ('555','221','small','6 hours','Free');
INSERT INTO Parking (Parking_ID, Booking_ID,Space_Size,Duration,
    Cost)
VALUES ('444','225','large','72 hours','£50');
INSERT INTO Parking (Parking_ID, Booking_ID,Space_Size,Duration,
    Cost)
VALUES ('999','224','small','48 hours','£25');
INSERT INTO Parking (Parking_ID, Booking_ID,Space_Size,Duration,
    Cost)
VALUES ('888','222','small','24 hours','Free');
INSERT INTO Parking (Parking_ID, Booking_ID,Space_Size,Duration,
    Cost)
VALUES ('333','225','small','72 hours','£30');
```

Room Table Population

```
1  INSERT INTO Room (Room_ID, Booking_ID,Room_Size,Ensuite,
       Booking_Price)
2  VALUES ('321','221','Single','N','£80');
3  INSERT INTO Room (Room_ID, Booking_ID,Room_Size,Ensuite,
       Booking_Price)
4  VALUES ('322','222','Single','N','£80');
5  INSERT INTO Room (Room_ID, Booking_ID,Room_Size,Ensuite,
       Booking_Price)
6  VALUES ('323','223','Single','N','£70');
7  INSERT INTO Room (Room_ID, Booking_ID,Room_Size,Ensuite,
       Booking_Price)
8  VALUES ('324','224','Double','Y','£120');
9  INSERT INTO Room (Room_ID, Booking_ID,Room_Size,Ensuite,
       Booking_Price)
10 VALUES ('325','225','Double','Y','£150');
11 INSERT INTO Room (Room_ID, Booking_ID,Room_Size,Ensuite,
       Booking_Price)
12 VALUES ('326','226','Single','Y','£175');
```

Orders Table Population

```
1  INSERT INTO Orders (Order_Number,Customer_ID,Total_Cost,
       Description)
2  VALUES ('101','111','£100','Food order');
3  INSERT INTO Orders (Order_Number,Customer_ID,Total_Cost,
       Description)
4  VALUES ('102','112','£150','Food order');
5  INSERT INTO Orders (Order_Number,Customer_ID,Total_Cost,
       Description)
6  VALUES ('103','113','£80','Food order');
7  INSERT INTO Orders (Order_Number,Customer_ID,Total_Cost,
       Description)
8  VALUES ('104','114','£200','Gym and outdoor catering');
9  INSERT INTO Orders (Order_Number,Customer_ID,Total_Cost,
       Description)
10 VALUES ('105','115','£250','Gym and outdoor catering');
```

Food Course Table Population

```
1  INSERT INTO Food_Course (Course_Number,Name,Description,Type,
       Price)
2  VALUES ('001','Chinese Wok Rice','Stir fried rice with shrimps
       and egg','Main dish','£20');
3  INSERT INTO Food_Course (Course_Number,Name,Description,Type,
       Price)
4  VALUES ('002','Spaghetti Bolognese','Italian meat-based pasta','
       Main dish','£25');
5  INSERT INTO Food_Course (Course_Number,Name,Description,Type,
       Price)
6  VALUES ('003','Apple crumble','A sweet or savoury dish with ice
       cream or custard topped with apples','Dessert','£10');
7  INSERT INTO Food_Course (Course_Number,Name,Description,Type,
       Price)
8  VALUES ('004','Ham and cheese fillet','Ham and cheese bites','
       Appetiser','£15');
9  INSERT INTO Food_Course (Course_Number,Name,Description,Type,
       Price)
10 VALUES ('005','Soy braised chicken wings','sticky flavour wings
       with soy sauce','Appetiser','£15');
```

4.3 Constraints Testing

Following section will cover testing of the constraints within the database system. This will ensure the created system is capable of correctly capturing any data related errors and only allow good data to be inserted.

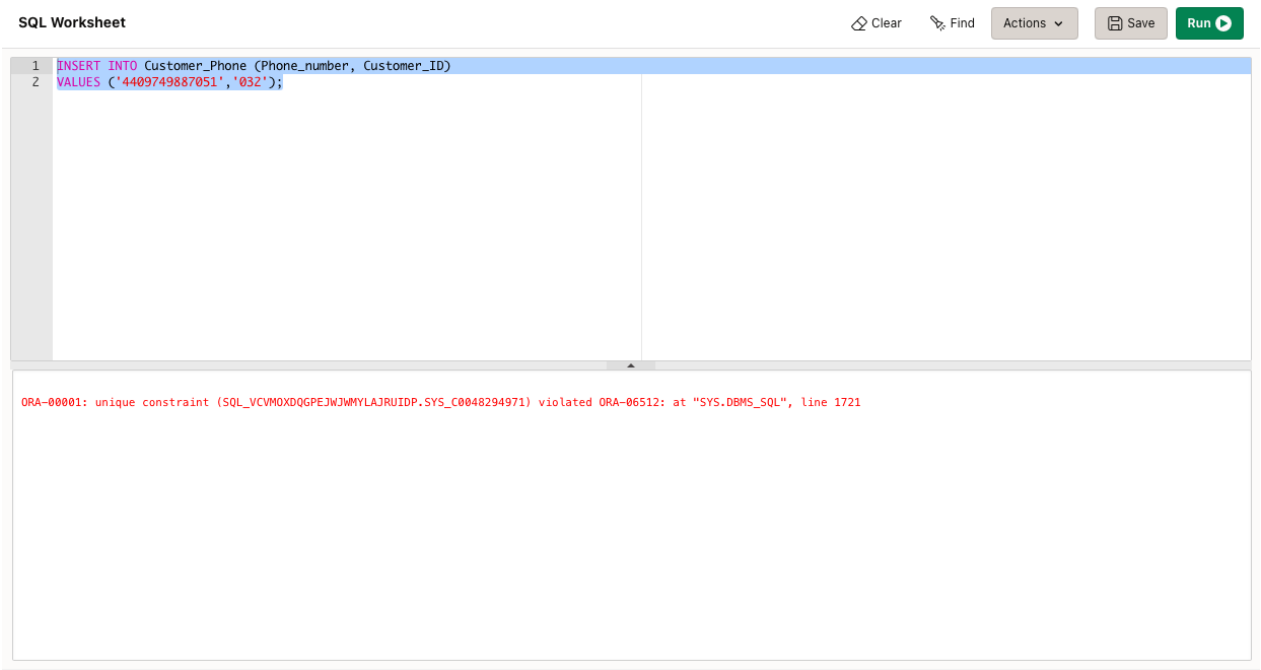

Figure 4.2: Unique Constraint Testing

Figure above shows results of the Unique constraint testing. As phone number was set as the Primary Key for this particular table, attempt of duplicating this value resulted in an error being displayed. The same phone number could not be included twice within the table as unique constraint ensured each phone number is an unique value.

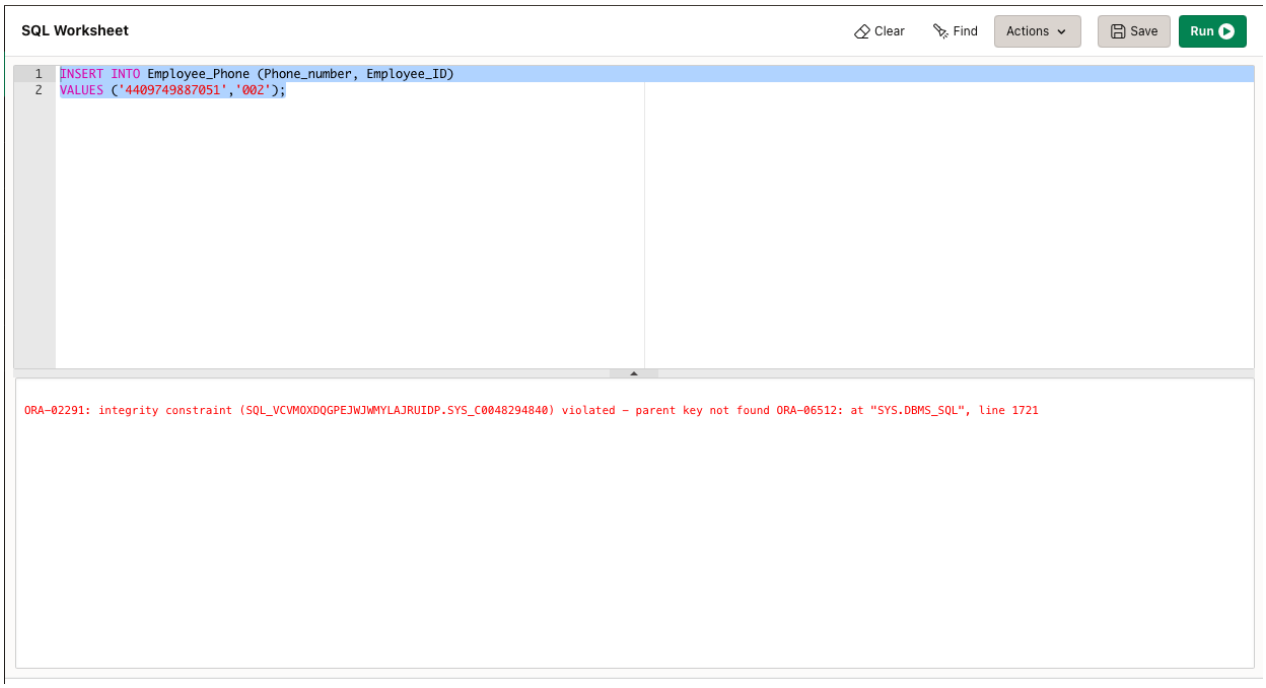

Figure 4.3: Integrity Constraint Testing

Figure above shows results of the Integrity constraint testing. This test was based on attempting to insert a value into the child table (with the foreign key), where the value does not exist in the corresponding parent table. This violated the integrity of the referential relationship and prompted Oracle to issue an error.

Chapter 5

SQL Testing Queries and Optimisation

This section will present variety of queries used to test technical capabilities of the created database system. It will also showcase possible query optimisation techniques used by businesses for extraction of valuable information from the system.

5.1 Marcel's Testing Queries (S17130681)

***Query*1**

Query below presents a join-on query aiming to display customer's name, ID and order description, where the total order cost is above £100. All names have been also ordered by descending.

```
1 SELECT Customer.Customer_ID,Customer.Name, Orders.Description
2 FROM Customer JOIN Orders
3 ON Customer.Customer_ID = Orders.Customer_ID
4 WHERE Total_Cost > '£100'
5 ORDER BY Name DESC;
```

CUSTOMER_ID	NAME	DESCRIPTION
115	Veronica	Gym and outdoor catering
114	Steve	Gym and outdoor catering
113	Ronald	Food order
112	Betty	Food order

Figure 5.1: Output of query 1

Query2

Query below presents a nested query aiming to display the name and ID of all customers, who's room size is double and also en-suite. All names have been additionally ordered by descending.

```
1  SELECT Customer_ID, Name
2  FROM CUSTOMER
3  WHERE Customer_ID IN (SELECT Booking.Customer_ID
4                FROM  Booking
5                WHERE Booking_ID IN
6                              (SELECT Room.Booking_ID
7                              FROM Room
8                              WHERE Room_Size = 'Double'
9                              AND Ensuite = 'Y'))
10 ORDER BY Name DESC;
```

CUSTOMER_ID	NAME
115	Veronica
114	Steve

Figure 5.2: Output of query 2

Query3

Query below presents a nested query aiming to display the name and ID of all customers who have booked a parking space and where the space size is small, cost free and duration of the stay is above 5 hours.

```
1  SELECT Customer_ID, Name
2  FROM CUSTOMER
3  WHERE Customer_ID IN (SELECT Booking.Customer_ID
4                FROM  Booking
5                WHERE Booking_ID IN
6                              (SELECT Parking.Booking_ID
7                              FROM Parking
8                              WHERE Space_Size = 'small'
9                              AND Cost = 'Free'
10                             AND Duration > '5 hours'));
```

CUSTOMER_ID	NAME
111	Frank

Figure 5.3: Output of query 3

Query4

Query below aims to display the name, salary, manager ID and role of all employees on the system who's salary is above £10,000 and their name starts with a letter 'P'.

```
1 SELECT  Manager_ID, Name, Salary, Role
2 FROM Employee
3 WHERE Salary > 10000
4 AND Name LIKE 'P%'
```

MANAGER_ID	NAME	SALARY	ROLE
3	Pablo	20000	Head Chef
4	Patricia	25000	Accountant

Figure 5.4: Output of query 4

Query5

Query below presents a sub-query aiming to display the name, role, salary and ID of all employees who work in Department 3. All names have been also sorted in an ascending order.

```
1 SELECT Employee_ID,Name, Role, Salary
2 FROM Employee
3 WHERE Employee_ID IN
4                 (SELECT Department_Employees.Employee_ID
5                 FROM Department_Employees
6                 WHERE Department_ID =  '3')
7 ORDER BY Name ASC;
```

EMPLOYEE_ID	NAME	ROLE	SALARY
202	Esther	Sales Manager	15000

Figure 5.5: Output of query 5

Query6

Query below presents an inner join of the Customer and Booking tables, aiming to extract the name, ID, total booking cost as well as the stay duration time of all customers who's total cost is under £700 and started their stay within the hotel after 7:00 and checked out after 17:00. Information have been also ordered by ascending total cost.

```
1 SELECT Customer.Customer_Id, Customer.Name, Booking.Total_Cost,
      Booking.Duration_Start_Time, Booking.Duration_End_Time
2 FROM CUSTOMER
3 INNER JOIN BOOKING ON Customer.Customer_ID = Booking.Customer_ID
4 WHERE Total_Cost < ' 700 '
5 AND Duration_Start_Time > '7:00'
6 AND Duration_End_Time > '17:00'
7 ORDER BY Total_Cost ASC;
```

CUSTOMER_ID	NAME	TOTAL_COST	DURATION_START_TIME	DURATION_END_TIME
113	Ronald	£220	9:00	18:30
111	Frank	£300	8:00	17:30
114	Steve	£500	8:00	20:00

Figure 5.6: Output of query 6

Query7

Query below presents a combination of an aggregate function with a nested query, which in this case is used to calculate an average salary of all employees who work in Department 4.

```
1 SELECT AVG (Salary) AS "Average Employee Salary in Pounds"
2 FROM EMPLOYEE
3 WHERE Employee_ID IN
4                     (SELECT Department_Employees.Employee_ID
5                      FROM Department_Employees
6                      WHERE Department_ID =  '4');
```

Average Employee Salary in Pounds
20000

Figure 5.7: Output of query 7

5.1.1 Marcel's SQL Query Optimisation (S17130681)

Table Indexing

Indexing optimisation method speeds up process of querying columns via creating pointers indicating where the data is stored within the system. Normal process of querying a specific table is based upon looking through each individual row until specific record is found. This can take long time especially when the database includes million of records and needed piece of information is located towards the end.

To address such querying issue, a Non-clustered Index will be created which will act as a sorted reference for a specific field within the main table, that holds pointers back to the original entries of the table. Non-clustered Index will be used to increase the speed of the queries on the table via creating more searchable columns.

Non-Clustered Indexes do not store the data themselves, instead they point to memory address instead. This approach makes this method slower than the Clustered Indexes but much faster than a non-indexed column.

As an example, if a business wanted to extract information about customers named Steve, it would take a long time for the system to search each of the rows. Such search could be especially difficult if the database does not sort customer names alphabetically as the desired information could be located anywhere within thousands of instances. See Figure 5.8

CUSTOMER_ID	NAME	TOTAL_BILL
111	Frank	£500
112	Betty	£530
113	Ronald	£400
114	Steve	£650
115	Veronica	£800

Figure 5.8: Customer Table Output

Non-Clustered Index Creation & Results

In order to create an index which will sort all customer names alphabetically, query below have been used. This created an index called "customer_name_asc", indicating that this index will store all names from the Customer table alphabetically in an ascending order.

```
1 CREATE INDEX Customer_name_asc ON Customer(Name ASC);
```

Screenshot below shows results of the query execution statistics before applying an index method to the Customer table. As a test, a simple SELECT query has been made to display all customers who's name is Steve. It shows that the elapsed time of this query was 3962 milliseconds with CPU usage of 4239. See Figure 5.9.

```
1 select * from Customer WHERE Name = 'Steve';
```

c9256ymhtw9h1	3962	3962	4239	select * from Customer WHERE Name = 'Steve'

Figure 5.9: Query execution before Indexing

Screenshot below shows results of the query execution statistics for the same query after applying an index method to the Customer table.

A significant improvement in the query speed as well as the CPU usage can be seen due to optimal search method also known as the Binary search. Binary search works by constantly cutting the data in half and checking if the required entry comes before or after the middle entry of the current portion of the data. Hence the elapsed time have decreased from 3962 to 3112 milliseconds, what can be denoted as 21% speed increase. CPU usage have also decreased from previous 4239 to 2779, which is an estimated 34% less usage. See Figure 5.10.

g6thqtnnhsqfq	3112	3112	2779	select * from Customer where Name = 'Steve'

Figure 5.10: Query execution after Indexing

Table Partitioning

As most organisations have a vast number of employees working within various departments, it might be difficult to search for a specific set of information. Looking for employees who earn specific salary could be one way of identifying the required instance. Although it can be challenging as exact salary amount is required for such search.

To speed up this process, a range partition can be created on an employee table which will subdivide the table into specific salary values allowing these objects to be managed and accessed at a finer level of granularity.

Partition Creation & Results

Query below have been used to create a new table Employee_part which will hold partitioned values. After creating the initial table, partition method have been specified by RANGE based off the Salary attribute. Next, 5 different partitions have been created within the new table with each being assigned to an unique salary range. As an example, P2 (partition 2) will only hold information about employees who's Salary is less than £2000.

```
1  CREATE TABLE Employee_part
2  (
3      EMPNO CHAR(3) NOT NULL PRIMARY KEY,
4      ENAME VARCHAR(20) NOT NULL,
5      SALARY CHAR(10) NOT NULL,
6      ROLE  VARCHAR(50) NOT NULL
7     )PARTITION BY RANGE (SALARY)
8
9                 (
10       PARTITION p1 VALUES LESS THAN (1000),
11       PARTITION p2 VALUES LESS THAN (2000),
12       PARTITION p3 VALUES LESS THAN (3000),
13       PARTITION p4 VALUES LESS THAN (4000),
14       PARTITION p5 VALUES LESS THAN (5000)
15       )ENABLE ROW MOVEMENT;
```

Query below have been used to confirm partition have been successful. Information regarding partition name, held values as well as the table can be seen. See Figure 5.11.

```
1  SELECT TABLE_NAME,PARTITION_NAME, PARTITION_POSITION, HIGH_VALUE
     FROM USER_TAB_PARTITIONS
```

TABLE_NAME	PARTITION_NAME	PARTITION_POSITION	HIGH_VALUE
EMPLOYEE_PART	P1	1	'1000'
EMPLOYEE_PART	P2	2	'2000'
EMPLOYEE_PART	P3	3	'3000'
EMPLOYEE_PART	P4	4	'4000'
EMPLOYEE_PART	P5	5	'5000'

Figure 5.11: Partition Details

To test functionality of the created range partition, a simple SELECT query have been used which displays all employees from the new table, who's Salary is below £2000. The query execution statistics shown the query elapsed time of 3542 milliseconds with CPU usage of 3778. See Figure 5.12.

```
1 SELECT * FROM Employee_part WHERE Salary < 2000;
```

5knvy98d0cdgb	3542	3542	3778	SELECT * FROM Employee_part WHERE Salary < 2000

Figure 5.12: Query execution before Partition

Now, the same query have been replicated but using the portioning approach. As only employees with Salary less than £2000 are required, partition 2 will be used to extract that data from the system. This time a significant increase can be seen in both the elapsed time as well as the CPU usage. Query time have increased from 3542 to 4593 milliseconds with CPU usage increase from 3778 to 4165. See Figure 5.13.

```
1 SELECT * FROM Employee_part PARTITION(P2);
```

9azw7dkzcjs1a	4593	4593	4165	SELECT * FROM Employee_part PARTITION(P2)

Figure 5.13: Query execution after Partition

As the two different queries have returned the same output, see Figure 5.14, partitioned table unperformed as query execution took estimated 29% longer and consumed 10% more CPU power. Partitioning not always will enhance performance, as this optimisation method is meant for use against vast volumes of data, more CPU usage will be naturally required in order to efficiently extract required information. As there are 5 different partitions within the new table, trying to extract data means system has to perform 5 times the amount of logical reads therefore increasing execution times.

EMPNO	ENAME	SALARY	ROLE
102	STEVE	1000	Sales Manager

Figure 5.14: Output of two different queries

5.2 Maryann's Queries (S20158627)

Query1

Query below displays inner joins from the customer's table and order table to get the total bill in ascending order.

```
1 SELECT Customer.Customer_ID, Name, Total_Bill ,Order_Number
2 FROM Customer
3 INNER JOIN Orders
4 ON Customer.Customer_id= Orders.Customer_id
5 ORDER BY Customer.Total_Bill asc;
```

CUSTOMER_ID	NAME	TOTAL_BILL	ORDER_NUMBER
113	Ronald	£400	103
111	Frank	£500	101
112	Betty	£530	102
114	Steve	£650	104
115	Veronica	£800	105

Figure 5.15: Output Of Inner Join Query

Query2

Query below displays aggregate functions with the use of Aliases to get the data overview of salaries of employees.

```
1 SELECT MIN(Salary) AS Lowest_Salary, MAX(Salary) AS
    Highest_Salary, AVG(Salary) AS Average_Salary
2 FROM Employee;
```

LOWEST_SALARY	HIGHEST_SALARY	AVERAGE_SALARY
10000	25000	16000

Figure 5.16: Output Of Aggregate Function Query

Query3

Query below displays name and employee ID of employees that have more than one email address.

```
1 SELECT b.Name, a.Employee_id, COUNT(*) AS NO_EMAIL
2 FROM Employee_email a
3 INNER JOIN Employee b ON a.Employee_id = b.Employee_id
4 GROUP BY a.Employee_id, b.Name
5 HAVING COUNT(*) > 1;
```

NAME	EMPLOYEE_ID	NO_EMAIL
John	201	2

Figure 5.17: Output Of 'Group By' Query

Query4

Query below displays a nested query that shows department name with the number of facilities that are open by 0:00.

```
1 SELECT Department.Department_ID, Department.Name, COUNT(*) AS
    NO_FACILITY
2 FROM Department_Facilities
3 INNER JOIN Department ON Department_Facilities.Department_ID =
    Department.Department_ID
4 WHERE Facility_ID IN (SELECT Facility_ID
5 FROM Facility
6 WHERE Schedule_Opening_Time = '0:00')
7 GROUP BY Department.Department_ID, Department.Name;
```

DEPARTMENT_ID	NAME	NO_FACILITY
1	Front Office	2

Figure 5.18: Output Of Nested Query

Query5

The Query below combines an employee's contact detail in one table

```
SELECT Employee_id, Phone_number AS CONTACT
FROM Employee_phone
WHERE Employee_id ='202'
UNION
SELECT Employee_id, Email_address
FROM Employee_email
WHERE Employee_id ='202'
ORDER BY Employee_id;
```

EMPLOYEE_ID	CONTACT
202	4407457612459
202	4407679853268
202	Estherbanks@gmail.com

Figure 5.19: Output Of Union Query

5.2.1 Maryann's Database Optimisation Techniques

5.2.2 Indexing

The Food Course table was indexed by the name column to speed up searches/queries in the Food Menu.

```
1 CREATE INDEX Food_Course_Name
2 ON Food_Course(Name);
```

The following displays a screenshot showing query result before adding the Food course table index to the indexing process. A selected query is made to view the Apple Crumble course, which shows time running as 4535 milliseconds with CPU use 3926.

```
1 SELECT *
2 FROM Food_Course
3 WHERE Name LIKE '%Apple crumble%';
```

COURSE_NUMBER	NAME	DESCRIPTION
3	Apple crumble	A sweet or savoury dish with icecream or custard topped with apples

Figure 5.20: Output Of Selecting Food course like Apple Crumble

SQL_TEXT	ELAPSED_TIME	AVG_ELAPSED	AVG_CPU
SELECT * FROM Food_Course WHERE Name LIKE '%Apple crumble%'	4535	4535	3926

Figure 5.21: Statistics Of Query Before Indexing

The following query was used to get the performance statistics of the Elapsed time, Avg Elapsed and AVG CPU

```
select * from
(
    select
        sql_id,
        elapsed_time,
        elapsed_time/executions avg_elapsed,
        cpu_time/executions avg_cpu,
        sql_text
    from v$sql
    order by avg_elapsed desc
);
```

After optimisation, there was a substantial change in the query speed as well as the Processor utilisation, thus the Elapsed time/Avg Elapsed was reduced from 4535 to 2919 milliseconds, which is calculated at 35.6 percent speed increase and the CPU use decreased from 3926 to 2588, estimated at 34 percent less consumption.

SQL_TEXT	ELAPSED_TIME	AVG_ELAPSED	AVG_CPU
SELECT * FROM Food_Course WHERE Name LIKE '%Apple crumble%'	2919	2919	2588

Figure 5.22: Statistics Of Query After Indexing

5.2.3 Clustering

A cluster is a schema object containing data from one or more tables that all have one or more similar columns. You can create tables in the cluster after building a cluster.

A cluster index must, however, be generated before any rows can be inserted into the clustered tables. The creation of additional indexes on the clustered tables does not impact the use of clusters; they can be created and dropped as normal.

Below displays query's from data that are clustered from two tables; Employee table and Department Employee Table which have a similar column Employee id.

```
CREATE CLUSTER emp_dept (employee_id int)
    SIZE 300
    TABLESPACE LIVESQL_USERS
    STORAGE (INITIAL 200K
        NEXT 300K
        MINEXTENTS 2
        PCTINCREASE 33);
```

```
CREATE TABLE Employees (
     Employee_id int NOT NULL PRIMARY KEY,
      Manager_ID int NOT NULL UNIQUE,
       Name char(50) NOT NULL,
       Salary varchar(10) NOT NULL,
        Role char(50) NOT NULL)
   CLUSTER emp_dept (Employee_id);
```

```
CREATE TABLE Dept (
    Department_employee_ID int NOT NULL UNIQUE,
     Employee_ID int NOT NULL UNIQUE,
      Department_ID int NOT NULL UNIQUE)
 CLUSTER emp_dept (Employee_id);
```

```
CREATE INDEX emp_dept_index
    ON CLUSTER emp_dept
    TABLESPACE LIVESQL_USERS
    STORAGE (INITIAL 50K
        NEXT 50K
        MINEXTENTS 2
        MAXEXTENTS 10
        PCTINCREASE 33);
```

```
SELECT * FROM USER_CLU_COLUMNS
```

CLUSTER_NAME	CLU_COLUMN_NAME	TABLE_NAME	TAB_COLUMN_NAME
EMP_DEPT	EMPLOYEE_ID	DEPT	EMPLOYEE_ID
EMP_DEPT	EMPLOYEE_ID	EMPLOYEES	EMPLOYEE_ID

Figure 5.23: Output Query Of the USER CLU Columns

This statement queries USER INDEXES to display the index attributes of the EMP DEPT cluster.

```
SELECT CLUSTER_NAME, TABLESPACE_NAME, CLUSTER_TYPE, PCT_INCREASE,
      MIN_EXTENTS, MAX_EXTENTS
FROM USER_CLUSTERS;
```

CLUSTER_NAME	TABLESPACE_NAME	CLUSTER_TYPE	PCT_INCREASE	MIN_EXTENTS	MAX_EXTENTS
EMP_DEPT	LIVESQL_USERS	INDEX	-	1	2147483645

Figure 5.24: Query For User Indexes to Display Emp Dept

A sample of the clustering technique was tested using an Inner Join Query.

```
SELECT Employee.Employee_ID, Name, Manager_id, Salary, Role
FROM Employee
INNER JOIN Department_Employees
ON Employee.Employee_id= Department_Employees.Employee_id;
```

ELAPSED_TIME	AVG_ELAPSED	AVG_CPU	SQL_TEXT
9350	9350	9331	SELECT Employee.Employee_ID, Name,Manager_id,Salary,Role FROM Employee INNER JOIN Department_Employees ON Employee.Employee_id= Department_Employees.Employee_id

Figure 5.25: Statistics Of Query Before Clustering

Clustering with SQL Server improves data redundancy which is an advantage to reduce downtime in a server.

After the clustering technique was implemented, it is observed that the AVG Elapsed and AVG CPU decreased by 47.6 Percent.

ELAPSED_TIME	AVG_ELAPSED	AVG_CPU	SQL_TEXT
9793	4896.5	4887	SELECT Employee.Employee_ID, Name,Manager_id,Salary,Role FROM Employee INNER JOIN Department_Employees ON Employee.Employee_id= Department_Employees.Employee_id

Figure 5.26: Statistics Of Query After Clustering

5.3 Oliver's Queries

5.3.1 Get employee information on employees working in a department with a specific manager.

```
SELECT Employee_ID, Name, Role, Salary
FROM Employee
WHERE Employee_ID IN
                (SELECT Department_Employees.Employee_ID
                FROM Department_Employees
                WHERE Department_ID =  '4')
                AND MANAGER_ID = '203'
ORDER BY Name ASC;
```

EMPLOYEE_ID	NAME	ROLE	SALARY
207	Jacob	Sous Chef	12000

Figure 5.27: Query output selects all employees who work in a department under a specified manager

5.3.2 Select employee and department information for employees who work under a specific manager

```
SELECT EMPLOYEE.EMPLOYEE_ID, EMPLOYEE.NAME, EMPLOYEE.SALARY,
    EMPLOYEE.ROLE, DEPARTMENT_EMPLOYEES.DEPARTMENT_ID
FROM EMPLOYEE
INNER JOIN DEPARTMENT_EMPLOYEES
ON Employee.Employee_Id = DEPARTMENT_EMPLOYEES.Employee_Id
WHERE EMPLOYEE.MANAGER_ID = '201'
ORDER BY EMPLOYEE.Name ASC
```

EMPLOYEE_ID	NAME	SALARY	ROLE
205	Kate	10000	Front Desk Manager

Figure 5.28: Query output selects all employees under specified manager

5.3.3 Get information about room bookings for a customer

```
SELECT Customer.Customer_Id, Booking.Booking_Id, Customer.Name,
    Room.Room_Id, Room.Ensuite, Room.Room_Size, Room.Booking_Price
    , Booking.Duration_Start_Time, Booking.Duration_End_Time
FROM CUSTOMER
INNER JOIN BOOKING
ON Customer.Customer_Id = Booking.Customer_Id
INNER JOIN ROOM
ON Booking.Booking_Id = Room.Booking_Id
WHERE Customer.Customer_Id = '111'
ORDER BY Booking.Booking_Id ASC
```

CUSTOMER_ID	BOOKING_ID	NAME	ROOM_ID	ENSUITE	ROOM_SIZE	BOOKING_PRICE	DURATION_START_TIME	DURATION_END_TIME
111	221	Frank	321	N	Single	£80	8:00	17:30
111	226	Frank	326	Y	Single	£175	10:00	18:30

Figure 5.29: Customer room details output

5.3.4 Parking information for spaces of a certain size, ordered by cost

```
SELECT Parking_Id, Space_Size, Duration, Cost
FROM PARKING
WHERE Space_Size = 'small'
ORDER BY Cost ASC
```

PARKING_ID	SPACE_SIZE	DURATION	COST
555	small	6 hours	Free
888	small	24 hours	Free
999	small	48 hours	£25
333	small	72 hours	£30

Figure 5.30: Parking query output

5.3.5 Minimum and Maximum price of food per course type

```
1 SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price
2 FROM FOOD_COURSE
3 WHERE Type = 'Main dish'
```

MIN_PRICE	MAX_PRICE
£20	£25

Figure 5.31: Food query output

5.3.6 Booking Indexing

The Bookings list was indexed in descending order of Booking_Id to speed up queries looking for more recent bookings made by customers.

Customer 111 has made another booking after staying with the hotel initially before; as bookings are indexed in ascending order from oldest to newest, this would mean it would take longer to retrieve the most recent booking for this customer.

BOOKING_ID	CUSTOMER_ID	TOTAL_COST	DURATION_START_TIME	DURATION_END_TIME
221	111	£300	8:00	17:30
222	112	£250	9:00	17:00
223	113	£220	9:00	18:30
224	114	£500	8:00	20:00
225	115	£700	8:00	21:30
226	111	£450	10:00	18:30

Figure 5.32: Output from selecting all bookings

The following query gives us performance statistics of the elapsed time, average elapsed time and average CPU usage, see Figure 5.33.

```
1 SELECT * FROM BOOKING WHERE BOOKING_ID = '226'
```

dvbwkbgskx038	2955	2955	2918	SELECT * FROM BOOKING WHERE BOOKING_ID = '226'

Figure 5.33: Query statistics - Pre-indexing

To increase the search speed of queries looking for recent bookings, the Booking_Id index can be inverted to be in descending order (newest to oldest record) with the following code.

```
1 CREATE INDEX Booking_Id_Desc ON BOOKING(BOOKING_ID Desc)
```

After optimisation the previous query runs faster, reducing the execution time by almost 1/3rd and CPU load by almost 20% see Figure 5.34.

dvbwkbgskx038	2087	2087	2383	SELECT * FROM BOOKING WHERE BOOKING_ID = '226'

Figure 5.34: Query statistics - Post-indexing

5.3.7 Customer Booking Clustering

The customer and booking tables are frequently joined together based on the customer_id attribute, to search for bookings that a customer has made. A clustered index can be created alongside making the customer and booking tables cluster tables, so that the clustered index stores queried information for faster retrieval on subsequent queries; this does however create additional tables, which use additional computation power and storage space.

The cluster index and tables were created using the following code:

```
CREATE CLUSTER CUST_BOOKINGS (customer_id int)
SIZE 200
TABLESPACE LIVESQL_USERS
STORAGE (INITIAL 100K
    NEXT 200K
    MINEXTENTS 2
    PCTINCREASE 25)

CREATE TABLE Customer (
     Customer_ID int NOT NULL PRIMARY KEY,
     Name char(50) NOT NULL,
     Total_Bill varchar(10) NOT NULL)
CLUSTER CUST_BOOKINGS (Customer_ID)

CREATE TABLE Booking (
     Booking_ID int NOT NULL PRIMARY KEY,
     Customer_ID int NOT NULL,
     Total_Cost varchar(10) NOT NULL,
     Duration_Start_Time varchar(25) NOT NULL,
     Duration_End_Time varchar(25) NOT NULL,
     FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID))
CLUSTER CUST_BOOKINGS (Customer_ID)

CREATE INDEX CUST_BOOKINGS_INDEX
     ON CLUSTER CUST_BOOKINGS
     TABLESPACE LIVESQL_USERS
     STORAGE (INITIAL 50K
       NEXT 50K
       MINEXTENTS 2
       MAXEXTENTS 5
       PCTINCREASE 25)
```

Creation of the cluster could then be checked by searching the clusters created by the user, returning the result shown in Figure 5.35.

```
1 SELECT CLUSTER_NAME, TABLESPACE_NAME, CLUSTER_TYPE, PCT_INCREASE,
      MIN_EXTENTS, MAX_EXTENTS FROM USER_CLUSTERS
```

CLUSTER_NAME	TABLESPACE_NAME	CLUSTER_TYPE	PCT_INCREASE	MIN_EXTENTS	MAX_EXTENTS
CUST_BOOKINGS	LIVESQL_USERS	INDEX	-	1	2147483645

Figure 5.35: Checking created clusters within the users clusters

Running the following sample query that utilises an inner join is initially slower and more CPU intensive on the clustered index (see Figure 5.37), but subsequent retrievals of indexed records will be faster once information has been stored in the indexed cluster.

```
1 SELECT Customer.Customer_Id, Customer.Name, Booking.Booking_Id,
      Booking.Duration_Start_Time, Booking.Duration_End_Time
2 FROM CUSTOMER
3 INNER JOIN BOOKING ON Booking.Customer_Id = Customer.Customer_Id
```

SQL_ID	ELAPSED_TIME	AVG_ELAPSED	AVG_CPU	
5mbhuxaphnah7	13794	13794	12263	SELECT Customer.Customer_Id, Customer.Name, Booking.Booking_Id, Booking.Duration_Start_Time

Figure 5.36: Query statistics without clustered index

SQL_ID	ELAPSED_TIME	AVG_ELAPSED	AVG_CPU	
5mbhuxaphnah7	19884	19884	20027	SELECT Customer.Customer_Id, Customer.Name, Booking.Booking_Id, Booking.Duration_Start_Time

Figure 5.37: Query statistics with clustered index implemented

Chapter 6

Conclusion

6.1 Conclusions

The initial idea was successfully fulfilled and a fully functional database system was created. All designs have been closely followed to ensure this system can be re-created as accurately as possible. Functional testing of aspects such as data extraction or query optimisation have been successfully carried out. No major issues were encountered while undergoing design and development stages. The only changes taken into consideration were related to unavailable data types in Oracle Live, which was used due to external circumstances facilitating remote working.

Such changes were implemented to the Ensuite attribute within the Room entity, as the Boolean data type was not valid within Oracle Live, it was changed to VARCHAR(1). Data types for attributes related to time such as the Duration_Start_Time within the Booking entity also had to be adjusted; as a distinct time data type is not supported, only datetime, VARCHAR(10) was used instead for manual time insertion. The rest of the development cycle went according to plan and on track to meet the final deadline.

6.2 Future Work

Further work could be done to incorporate customer and employee financial details, to further augment the salary system and handling of customer fees within the system. The customer and employee tables could also be expanded to contain more details as necessary, such as employee working hours, or the size of a customer's party staying within the hotel. Additional information about parked vehicles and room availability could also be stored within a booking to be used to notify staff of when parking spaces and rooms become free, allowing these to be cleared for use by other customers.

Bibliography

Al-Masree, H.K., 2015. Extracting entity relationship diagram (erd) from relational database schema. International Journal of Database Theory and Application, 8(3), pp.15–26.

Kanellakis, P.C., 1990. Elements of relational database theory. Formal models and semantics, Elsevier, pp.1073–1156.

Kent, W., 1983. A simple guide to five normal forms in relational database theory. Communications of the ACM, 26(2), pp.120–125.

Roy-Hubara, N., Rokach, L., Shapira, B. and Shoval, P., 2017. Modeling graph database schema. IT Professional, 19(6), pp.34–43.

Teorey, T.J., Yang, D. and Fry, J.P., 1986. A logical design methodology for relational databases using the extended entity-relationship model. ACM Computing Surveys (CSUR), 18(2), pp.197–222.