DATABASE PRINCIPLES COURSEWORK REPORT

Do not write your name on your work unless your lecturer has explicitly told you to do so.

Student ID number	Title of degree studying	Level/Year
2200902	**BSC (HONS) DATA SCIENCE AND ANALYTICS (TOP UP) - PT UOP BSCDSA 11**	1
2200918	**BSC (HONS) DATA SCIENCE AND ANALYTICS (TOP UP) - PT UOP BSCDSA 11**	1
2200923	**BSC (HONS) DATA SCIENCE AND ANALYTICS (TOP UP) - PT UOP BSCDSA 11**	1

Short unit name:	M32363 DP				Due date: 20 February 2022	Deadline: 20 February	
Full unit name:	DATABASE PRINCIPLES						
Unit lecturer name:	**Dr Liu Kaiping**				Group: **10**		
Additional items e.g. CD/disk/USB:	Yes		No	✓	**Details:**		

All additional items should be clearly labelled with ID number and unit name and securely attached to your work.

Candidates are reminded that the following are defined as Assessment Offences and will be dealt with in accordance with the University's Code of Student Discipline:

a) Any attempt to complete an assessment by means considered to be unfair;
b) Plagiarism, which the University defines as the incorporation by a student in work for assessment of material which is not their own, in the sense that all or a substantial part of the work has been copied without any adequate attempt at attribution or has been incorporated as if it were the student's own work when in fact it is wholly or substantially the work of another person or persons.

Please note: Group **coursework** will be filed under the first Student ID number at the top of the list. Ensure you know all

Group member's ID numbers.

2200902
2200918
2200923

NB: **Coursework not collected** will be disposed of six months **after** the hand-in date.

FOR OFFICIAL USE ONLY

Administration Office	Academic Staff Member
Date received/Office stamp	Provisional mark % / Comments

Database Principles

Title: Design Documentation - Travel Booking System for Chan Brothers Travel Corporation

Student ids: 2200902, 2200918, 2200923

University of Portsmouth

Course Number: M32363 Database Principles (DP)

INDEX

Table of Contents

1. Introduction

1.1 Background

Chan Brothers Travel Corporation is a global tourism enterprise specializing in providing comprehensive tour packages, encompassing flight bookings, accommodations, and even optional car rental services. In light of the company's growth, they are in pursuit of a **centralized database solution** that allows seamless web access for their staff. The database is expected to accommodate diverse employee roles while effectively handling customer bookings, flight arrangements, hotel reservations, and payment tracking.

1.2 Purpose:

This document outlines the design, development, and implementation of a centralized relational database for Chan Brothers Travel Pte Ltd. The database will be designed using an Enhanced Entity Relationship Diagram (EERD), along with a Data Dictionary to specify constraints and descriptions. Additionally, it includes the physical implementation details and sample data for testing. Furthermore, we present five sample queries to demonstrate the database's usability and functionality.

1.3 Business Requirements

- Central database to facilitate booking by employees from various office locations
- Develop algorithms for **generating package options** based on customer preferences.
- Provide customers with **package details and options**.
- Create a **streamlined booking process** that employees can follow.
- Define user **roles and permissions** for employee access to the system.
- Design and implement a **secure database structure** for storing booking and package data.
- Develop **reporting mechanisms** to track booking records, payment and installment details and system usage.
- Ensure the system can **handle concurrent booking requests** without performance issues.

1.4 Input Requirements

Based on the provided input criteria, employers can generate a list of suitable holiday packages that align with the customer's preferences and requirements. This process ensures that the offered packages are personalized and cater to the customer's needs, enhancing the overall customer experience.

Database design must consider all the below aspects to conform with the business requirements

- **Location:** The desired travel destination or location.
- **Duration:** The intended duration of the holiday (e.g., number of days).
- **Budget Range:** The customer's budget constraints for the package.
- **Travel Dates:** Preferred travel dates or date range.
- **Accommodation Type:** references for hotel features, facilities, and star ratings.
- **Number of Travelers: Number** of adults and children traveling together.
- **Additional Features**: Any specific amenities or features the customer is interested in (e.g., all-inclusive, beachfront, family-friendly, etc.).
- **Contact Information:** Customer's contact details to communicate the package options and details.

1.5 Document Outline

The remainder of the document is structured as follows:

- **Section 2: Database Design:** Logical design of the database, including an Enhanced Entity-Relationship Diagram (EERD) with explanations of entities, relationships, constraints and a comprehensive Data Dictionary with attribute details.
- **Section 3: Database Implementation:** This section creates tables, applies constraints, and inserts sample data for testing the database's functionality.
- **Section 4: Query Design:** Showcases five sample queries that demonstrate the utility and functionality of the database system.
- **Section 5: Security, Optimization, Professional, Legal and Ethical Issues:** Address crucial aspects related to security, optimization, legal compliance, and ethical considerations for the designed database system.

- **Section 6: Conclusion and Future Work:** Summarizes the project and discusses potential improvements and future extensions of the database system.

2. Database Design

2.1 Enhanced Entity Relationship Diagram (EERD):

Based on the business requirements, we can begin by identifying the entities, attributes, relationships, and cardinalities.

2.1.1 Entities & Attributes:

- **Customer** (PK: cust_id, cust_fname, cust_lname, cust_email, cust_phone, cust_street_address, cust_city, cust_postcode, cust_dob)
- **Employee** (PK: emp_id, emp_username, emp_firstname, emp_lastname, emp_role, emp_branch, emp_contact, emp_active_login)
- **Payment** (PK: pay_id, FK: book_id, pay_amt_paid, pay_install_no, pay_status)
- **Payment_Installment** (PK: pay_install_id, FK: pay_id, pay_install_amt, pay_install_date, pay_install_status)
- b(PK: book_id, FK: cust_id, pack_id, emp_id, book_depart_date, book_adult_No, book_child_no,book_discount, book_total_amt, book_install)
- **Package** (PK: pack_id, FK: hotel_id, FK:flight_id, pack_name, pack_pricePP, pack_duration, pack_location)
- **Hotel_Amenity** (PK/FK: amenity_id, hotel_id)
- **Hotel** (PK: hotel_id,hotel_rating, hotel_name, hotel_location)
- **Amenity** (PK: amenity_id, amenity_description)
- **Hotel_Facility** (PK/FK: fac_id, FK: hotel_id)
- **Facility** (PK: fac_id, fac_description)
- **Flight**(PK: flight_id, out_flight_no , out_destination, out_location out_depart_date, out_arrival_date, ret_flight_no, ret_destination, ret_location, ret_depart_date, ret_arrival_date)

2.1.2 Relationships (Using Crow's Foot Notation)

- Customer to Booking (One-to-Many)
- Booking to Package (Many-to-One)
- Booking to Payment (One-to-One or Many)
- Package to Flight (Many-to-One)
- Payment to Payment_Installment (One-to-Many Optional)
- Package to Hotel (One or Many-to-One)
- Hotel to Hotel_Amenity(One or Many to One)
- Hotel_Amenity to Amenity(One to One or Many)
- Hotel to Hotel_Facility(One or Many to One)
- Hotel_Facility to Facility(One to One or Many)
- Booking to Employee (Many to One)

2.1.3 Assumptions / considerations for relationship:

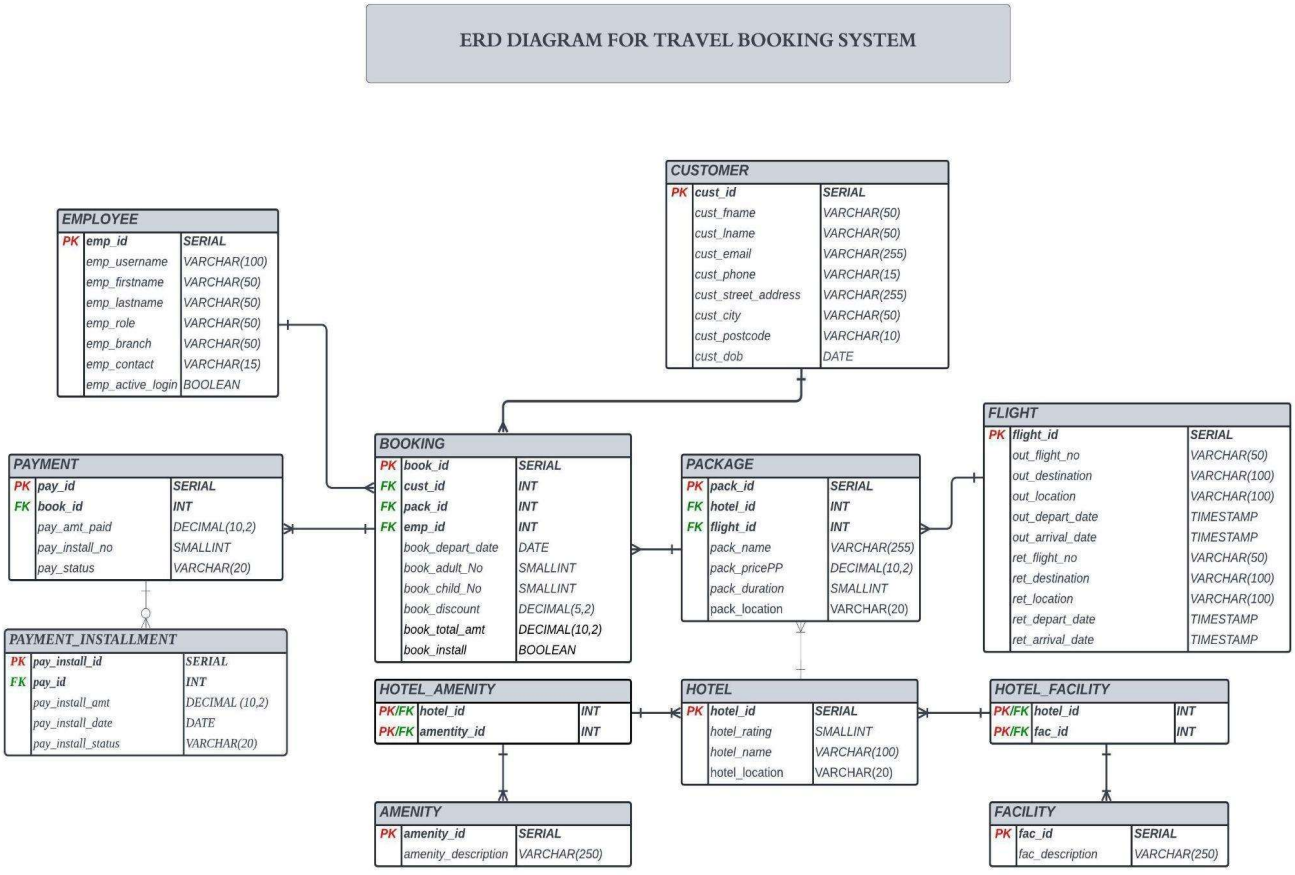
- 1 hotel (e.g Swiss) can be part of multiple packages
- Customer name, address and dob is optional
- Discount is implemented as optional. Can be 0 – 50%
- Each employee has only one role assigned and **only employees can perform the booking**
- **One Booking, Multiple Payments**: Each booking can have one or multiple payments associated with it. For example, a customer may choose to pay for a holiday package in multiple installments or through different payment methods.
- **Payment Installments:** As mentioned in the use case, the business allows customers to pay for a holiday in one or more installments. Each installment is a separate payment record, and the relationship between Payment and Installment (if implemented as a separate table) would also be one-to-many.
- **Data Integrity:** A one-to-many relationship ensures that each payment is associated with a specific booking, and there are no orphans or duplicate payment records. This ensures data integrity and helps avoid inconsistencies in the database.

2.1.4 Normalization:

- Hotel and Amenity have a many-to-many relationship, as a hotel can have multiple amenities, and an amenity can be present in multiple hotels. To resolve this, we create an intermediary table (junction table) to represent this relationship. (**Hotel_Amenity)**
- Hotel and Facility have a many-to-many relationship, as a hotel can have multiple facilities, and a facility can be present in multiple hotels. To resolve this, we create an intermediary table (junction table) to represent this relationship. (**Hotel_Facility)**

For the provided use case, the current design appears to be appropriate, and the relationships are well-defined. However, if we encounter specific performance issues or identify data redundancy during implementation or testing, we may consider reevaluating the design and potential normalization opportunities.

Regular performance testing, query optimization, and proper indexing are some of the strategies that can help mitigate performance issues in a well-designed database.



2.2 Data Dictionary:

TABLE 1: **EMPLOYEE**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
emp_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each employee
emp_username			VARCHAR(100)	UNIQUE, NOT NULL		Username for login purposes
emp_firstname			VARCHAR(50)	NOT NULL		
emp_lastname			VARCHAR(50)			
emp_role			VARCHAR(50)	NOT NULL, CHECK (emp_role IN ('manager_role', 'employee_role', 'accountant_role'))		Role of the employee (“Manager",”Employee”,Accountant")
emp_branch			VARCHAR(50)			Branch or office where the employee works
emp_contact			VARCHAR(15)	UNIQUE, NOT NULL		Contact information for the employee
emp_active_login			BOOLEAN	DEFAULT FALSE		Whether the employee is an active user

TABLE 2: **PAYMENT**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
pay_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each payment
book_id	FK	Y	INT	FOREIGN KEY	BOOKING(book_id)	Foreign key referencing the booking for which the payment was made
pay_amt_paid			DECIMAL(10,2)	CHECK (pay_amt_paid >= 0)		Amount paid for the booking
pay_install_no			SMALLINT	CHECK (pay_install_no IN (1, 3)		Number of installments for the payment (max 3 installments)
pay_status		Y	VARCHAR(20)	DEFAULT ‘Unpaid’ CHECK (pay_status IN ('Fully Paid', 'Partially Paid', ‘Unpaid’))		Default set to Unpaid Payment status ("Fully Paid", “Partially Paid”, “Unpaid”)

TABLE 3: **PAYMENT_INSTALLMENT**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
pay_install_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each payment installment
pay_id	FK	Y	INT	FOREIGN KEY	PAYMENT(pay_id)	Foreign key referencing the payment to which the installment belongs
pay_install_amt			DECIMAL(10,2)	CHECK (pay_install_amt >= 0)		Amount paid for the installment
pay_install_date			DATE			Date of the payment installment
pay_install_status			VARCHAR(20)	CHECK (pay_install_status IN ('Fully Paid', 'Partially Paid', 'Unpaid'))		Status of the payment installment

TABLE 4: **CUSTOMER**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
cust_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each customer
cust_fname			VARCHAR(50)	NOT NULL		First name of the customer
cust_lname			VARCHAR(50)	NOT NULL		Last name of the customer
cust_email			VARCHAR(255)	UNIQUE, NOT NULL		Email address of the customer
cust_phone			VARCHAR(15)	UNIQUE, NOT NULL		Phone number of the customer
cust_street_address			VARCHAR(255)			Street address of the customer
cust_city			VARCHAR(20)	NOT NULL		City in which the customer reside
cust_postcode			VARCHAR(10)			Pin code of the customer
cust_dob			DATE			Date of birth of the customer

TABLE 5:**BOOKING**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
book_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each booking
cust_id	FK		INT	FOREIGN KEY	CUSTOMER(cust_id)	Foreign key referencing the customer who made the booking
pack_id	FK		INT	FOREIGN KEY	PACKAGE(pack_id)	Foreign key referencing the package booked
emp_id	FK		INT	FOREIGN KEY	EMPLOYEE(emp_id)	Foreign key referencing the employee who managed the booking
book_ depart_date			DATE			Departure date of the booking

book_adult_no			SMALLINT	CHECK (book_adult_no > 0)		Number of adult travelers in the booking
book_child_no			SMALLINT	CHECK (book_child_no >= 0)		Number of child travelers in the booking
book_discount			DECIMAL (5,2)	CHECK (book_discount BETWEEN 0 AND 50)		The discount for booking
book_total_amt			DECIMAL (10,2)			Total amount to be paid based on package selected, number of adults and number of children
book_install			BOOLEAN	DEFAULT FALSE		If False (F), payment done in full; if True (T), payment done in installments. Defaults to false.

TABLE 6: **PACKAGE**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
pack_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each package
hotel_id	FK		INT	FOREIGN KEY	HOTEL(hotel_id)	Foreign key referencing the hotel included in the package
flight_id	FK		INT	FOREIGN KEY	FLIGHT(flight_id)	Foreign key referencing the flight included in the package
pack_name		Y	VARCHAR(255)	NOT NULL		Name of the package
pack_pricePP			DECIMAL(10,2)	CHECK (pack_pricePP > 0)		Price per person for the package
pack_duration			SMALLINT	CHECK (pack_duration > 0)		Duration of the package in days
pack_location			VARCHAR(20)			Location of the package

TABLE 7: **HOTEL_AMENITY**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
hotel_id	CK	Y	INT	FOREIGN KEY	HOTEL(hotel_id)	Part of the composite primary key. Unique identifier for a hotel.
amenity_id	CK	Y	INT	FOREIGN KEY	AMENITY(amenity_id)	Part of the composite primary key, foreign key referencing the amenity associated with the hotel amenity

TABLE 8: **HOTEL**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
hotel_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each hotel
hotel_rating			SMALLINT	CHECK (hotel_rating BETWEEN 3 AND 5)		Rating for hotel (3 to 5 stars)
hotel_name		Y	VARCHAR(100)	NOT NULL		Name of the hotel
hotel_location		Y	VARCHAR(100)			Location of the hotel

TABLE 9: **AMENITY**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
amenity_id	PK		SERIAL	PRIMARY KEY, UNIQUE		Unique identifier for each amenity
amenity_description			TEXT	NOT NULL		Description of the amenity

TABLE 10: **HOTEL_FACILITY**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
hotel_id	CK	Y	INT	FOREIGN KEY	HOTEL(hotel_id)	Part of the composite primary key. Unique identifier for each hotel.
fac_id	CK	Y	INT	FOREIGN KEY	FACILITY (fac_id)	Part of the composite primary key, foreign key referencing the facility associated with the hotel facility

TABLE 11**: FACILITY**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
fac_id	PK		SERIAL	PRIMARY KEY		Unique identifier for each facility
fac_description			TEXT	NOT NULL		Description of the facility

TABLE 12: **FLIGHT**

Attribute_Name	KEY	INDEX	Data Type & Size	Domains & Constraints	FK Reference	Description
flight_id	PK		SERIAL	PRIMARY KEY		Primary key, unique identifier for each flight.
out_flight_no		Y	VARCHAR(50)	NOT NULL		Flight number for the outbound flight.

out_destination		Y	VARCHAR(100)			Destination of the outbound flight.
out_location			VARCHAR(100)			Departure location of the outbound flight.
out_depart_date		Y	TIMESTAMP			Date and time of departure for the outbound flight.
out_arrival_date		Y	TIMESTAMP	CHECK (out_arrival_date > out_depart_date)		Date and time of arrival for the outbound flight.
ret_flight_no			VARCHAR(50)	NOT NULL		Flight number for the return flight.
ret_destination			VARCHAR(100)			Destination of the return flight.
ret_location			VARCHAR(100)			Departure location of the return flight.
ret_depart_date			TIMESTAMP			Date and time of departure for the return flight.
ret_arrival_date			TIMESTAMP	CHECK (ret_arrival_date > ret_depart_date)		Date and time of arrival for the return flight.

3. Database Implementation:

We can now use the above data dictionary to write SQL code to create the necessary tables and establish the relationships between them.

3.1 Physical Implementation:

SQL Code:

<Please refer to the CODE.sql file uploaded seperately>

3.2 Trigger Summary:

Below triggers were created together with creation of a database to cater to multiple scenarios. Below is the summary of the same.

Trigger/Function Name	Description	Attached To	Trigger Type
update_package_attributes	Placeholder trigger function for future updates	Hotel and Flight	AFTER INSERT/UPDATE
update_total_amount	Calculates booking total amount based on package	Booking	BEFORE INSERT/UPDATE

update_total_amount_trigger	Invokes update_total_amount	Booking	BEFORE INSERT/UPDATE
update_payment_trigger	Updates Payment table based on booking updates	Booking	AFTER INSERT/UPDATE
update_installment_trigger_function	Updates Payment_Installment table based on payment status change	Payment_Installment	AFTER UPDATE
insert_payment_trigger	Update Payment table with details of payment for the booking made	Payment	AFTER INSERT TO BOOKING

3.3 Sample Data for Testing:

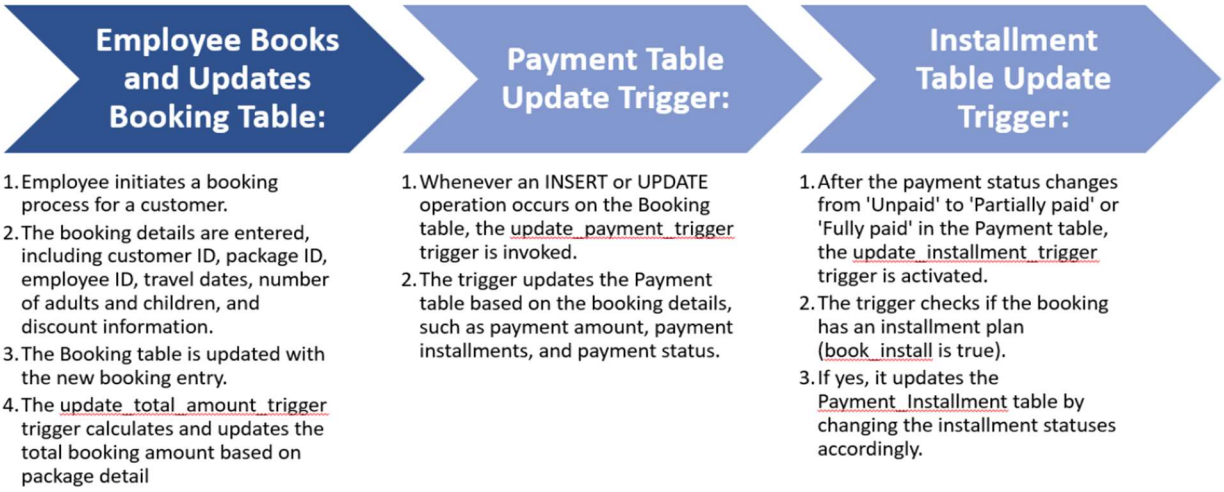
<<Also included in the attached CODE.sql file>

4. Query & View Design:

As part of evaluating the database's usability, we have designed and created views, queries and inserts that are considered useful. The queries are based on a Transaction Analysis conducted earlier to identify key transaction types.

By creating views, we can encapsulate the logic of complex queries and make them easily accessible for different analytical purposes. Below are the views based on which queries will be created subsequently

Based on the below booking process, appropriate views and queries has been designed



Below table summaries the basic business requirements vs queries/inserts/views created for the same:

Business Requirement	Description	Query /Insert/View
Precise Package Information	Display comprehensive package details for informed decisions.	View: PackageDetailsWithFlightAndHotel
Package Selection Assistance	Help employees generate packages based on customer criteria and offer suitable choices.	View: PackageSelectionCriteria
Enable Employee Booking	Allow employees to make bookings on behalf of customers.	Insert: Booking
Enter Customer Requirements	Provide an interface for employees to input customer requirements for package selection.	Insert: PackageSelectionCriteria
External Payment Process	After the booking and payment entry are created, the external payment process managed by the third-party application handles the actual payment from the customer.	Update: Payment and/or Payment_Installment
Viewing Installment Details	Provides a comprehensive list of pending installments for each customer. Accessible by Accountant_role only	View: Installment_Details_With_Customers
Accurate Booking Records	Maintain accurate records of customer bookings and details.	View: BookingDetailsByCustom
Data Analytics and Reporting	Retrieves the top 5 revenue-generating packages based on bookings.	Query: Top 5 Revenue-Generating Packages
	Identifies the most popular package based on bookings.	Query: Most Popular Package
	Provides a performance report for employees, highlighting successful bookings.	Query: Employee Performance Report

Codes for the below views are included within the CODE.sql file too

4.1.1 View: TotalRevenuePerPackage: This view calculates the total revenue for each package based on the sum of payments made for bookings associated with that package.

4.1.2: View: MostPopularPackage: This view determines the most popular package based on the number of bookings made for each package.

4.1.3: View: BookingDetailsByCustomer: This view retrieves all booking details for a specific customer based on their customer ID.

4.1.4: View: PackageDetailsWithFlightAndHotel: This view retrieves package details along with the associated flight and hotel information.

4.1.5: View: EmployeePerformance: This view tracks the performance of staff based on the number of successful bookings made by each employee.

4.1.6 View: Installment_Details_Per_Customers: Provides a comprehensive list of pending installments for each customer. Accessible by Accountant_role only

4.2 Queries & Inserts:

Once the .sql file has been executed and all tables, indexes, triggers and views are created, below queries can now be executed.

- **4.2.1 Precise Package Information:** Use the view: PackageDetailsWithFlightAndHotel

```
SELECT
    c.cust_city AS from_location,
    p.pack_id AS package_id,
    p.pack_name AS package_name,
    pd.package_description AS package_description,
    pd.flight_no AS onward_flight_no,
    pd.flight_destination AS destination,
    pd.departure_date AS onward_depart_date,
    pd.arrival_date AS destination_arrival_date
FROM
```

```
    CUSTOMER c
JOIN
    BOOKING b ON c.cust_id = b.cust_id
JOIN
    PACKAGE p ON b.pack_id = p.pack_id
JOIN
    PackageDetailsWithFlightAndHotel pd ON p.pack_name = pd.package_description;
```

Output:

```
travel_booking=# SELECT
    c.cust_city AS from_location,
    p.pack_id AS package_id,
    p.pack_name AS package_name,
    pd.package_description AS package_description,
    pd.flight_no AS onward_flight_no,
    pd.flight_destination AS destination,
    pd.departure_date AS onward_depart_date,
    pd.arrival_date AS destination_arrival_date
FROM
    CUSTOMER c
JOIN
    BOOKING b ON c.cust_id = b.cust_id
JOIN
    PACKAGE p ON b.pack_id = p.pack_id
JOIN
    PackageDetailsWithFlightAndHotel pd ON p.pack_name = pd.package_description;
 from_location | package_id |              package_name               |            package_description            | onward_flight_no | destination | onward_depart_date  | destination_arrival_date
---------------+------------+-----------------------------------------+-------------------------------------------+------------------+-------------+---------------------+-------------------------
 Los Angeles   |          1 | 7 Days to London at Cityscape Suites    | 7 Days to London at Cityscape Suites      | BA222            | London      | 2023-09-06 00:00:00 | 2023-09-07 00:00:00
 New York      |          2 | 7 Days to Sydney at Coastal Comfort Resort | 7 Days to Sydney at Coastal Comfort Resort | QF555          | Sydney      | 2023-09-10 00:00:00 | 2023-09-11 00:00:00
 Chicago       |          3 | 7 Days to Singapore at Urban Oasis Suites | 7 Days to Singapore at Urban Oasis Suites | SQ111            | Singapore   | 2023-09-15 00:00:00 | 2023-09-16 00:00:00
 Miami         |          4 | 7 Days to Tokyo at Skyline Lodge        | 7 Days to Tokyo at Skyline Lodge          | AA333            | Tokyo       | 2023-08-23 00:00:00 | 2023-08-24 00:00:00
 San Francisco |          5 | 7 Days to Paris at Elegant Chateau      | 7 Days to Paris at Elegant Chateau        | AF555            | Paris       | 2023-08-11 00:00:00 | 2023-08-12 00:00:00
 Los Angeles   |          6 | 7 Days to Dubai at Golden Sands Resort  | 7 Days to Dubai at Golden Sands Resort    | EK111            | Dubai       | 2023-07-10 00:00:00 | 2023-07-11 00:00:00
 New York      |          7 | 7 Days to Los Angeles at Palm Paradise Hotel | 7 Days to Los Angeles at Palm Paradise Hotel | DL222      | Los Angeles | 2023-09-07 00:00:00 | 2023-09-08 00:00:00
(7 rows)
```

- **4.2.2 Package Selection Assistance - View: PackageDetailsWithFlightAndHotel**

```
SELECT package_description, flight_no, flight_destination,
       departure_date, arrival_date, hotel_name, hotel_location
FROM PackageDetailsWithFlightAndHotel
JOIN package p ON PackageDetailsWithFlightAndHotel.package_description =
p.pack_name
WHERE p.pack_duration >= 5
  AND p.pack_duration <= 8
  AND p.pack_pricePP <= 1000;
```

Output:

```
travel_booking=# SELECT package_description, flight_no, flight_destination,
      departure_date, arrival_date, hotel_name, hotel_location
FROM PackageDetailsWithFlightAndHotel
JOIN package p ON PackageDetailsWithFlightAndHotel.package_description = p.pack_name
WHERE p.pack_duration >= 5
  AND p.pack_duration <= 8
  AND p.pack_pricePP <= 1000;
           package_description            | flight_no | flight_destination |   departure_date    |    arrival_date     |      hotel_name      | hotel_location
------------------------------------------+-----------+--------------------+---------------------+---------------------+----------------------+---------------
 7 Days to London at Cityscape Suites     | BA222     | London             | 2023-09-06 00:00:00 | 2023-09-07 00:00:00 | Cityscape Suites     | London
 7 Days to Sydney at Coastal Comfort Resort | QF555   | Sydney             | 2023-09-10 00:00:00 | 2023-09-11 00:00:00 | Coastal Comfort Resort | Sydney
 7 Days to Singapore at Urban Oasis Suites | SQ111   | Singapore          | 2023-09-15 00:00:00 | 2023-09-16 00:00:00 | Urban Oasis Suites   | Singapore
 7 Days to Tokyo at Skyline Lodge         | AA333     | Tokyo              | 2023-08-23 00:00:00 | 2023-08-24 00:00:00 | Skyline Lodge        | Tokyo
 7 Days to Paris at Elegant Chateau       | AF555     | Paris              | 2023-08-11 00:00:00 | 2023-08-12 00:00:00 | Elegant Chateau      | Paris
 7 Days to Dubai at Golden Sands Resort   | EK111     | Dubai              | 2023-07-10 00:00:00 | 2023-07-11 00:00:00 | Golden Sands Resort  | Dubai
 7 Days to Los Angeles at Palm Paradise Hotel | DL222 | Los Angeles        | 2023-09-07 00:00:00 | 2023-09-08 00:00:00 | Palm Paradise Hotel  | Los Angeles
(7 rows)
```

- **4.2.3 Enable Employee Booking - Insert: Booking**

```
INSERT INTO booking (cust_id, pack_id, book_depart_date, book_adult_no,
book_child_no, book_discount)
VALUES (3, 2, '2023-10-10', 2, 2, 0.10);
```

Output:

```
travel_booking=# INSERT INTO booking (cust_id, pack_id, book_depart_date, book_adult_no, book_child_no, book_disc
ount)
VALUES (3, 2, '2023-10-10', 2, 2, 0.10);
INSERT 0 1
travel_booking=# SELECT * FROM booking;
travel_booking=# SELECT * FROM booking;
 book_id | cust_id | pack_id | emp_id | book_depart_date | book_adult_no | book_child_no | book_discount | book_total_amt | book_install
---------+---------+---------+--------+------------------+---------------+---------------+---------------+----------------+-------------
       1 |       1 |       1 |      1 | 2023-09-15       |             2 |             1 |          0.10 |        2319.30 | f
       2 |       2 |       2 |      2 | 2023-10-10       |             1 |             0 |          0.20 |         590.40 | f
       3 |       3 |       3 |      3 | 2023-09-20       |             2 |             0 |          0.00 |        1270.00 | f
       4 |       4 |       4 |      4 | 2023-11-05       |             1 |             1 |          0.15 |        1232.50 | f
       5 |       5 |       5 |      5 | 2023-10-15       |             2 |             2 |          0.05 |        2223.00 | f
       6 |       6 |       6 |      6 | 2023-09-25       |             1 |             0 |          0.00 |         892.00 | f
       7 |       7 |       7 |      7 | 2023-12-01       |             2 |             1 |          0.10 |        2581.20 | f
       8 |       3 |       2 |        | 2023-10-10       |             2 |             2 |          0.10 |        2656.80 | f
(8 rows)
```

- **4.2.4 Accurate Booking Records - View: BookingDetailsByCustomer**

```
SELECT * FROM BookingDetailsByCustomer;
```

Output:

```
travel_booking=# SELECT * FROM BookingDetailsByCustomer;
 book_id |             package_description             | departure_date | number_of_adults | number_of_children
---------+---------------------------------------------+----------------+------------------+--------------------
       1 | 7 Days to Paris at Riverside Inn            | 2023-09-15     |                2 |                  1
       2 | 7 Days to London at Cityscape Suites        | 2023-10-10     |                1 |                  0
       3 | 7 Days to Sydney at Coastal Comfort Resort  | 2023-09-20     |                2 |                  0
       4 | 7 Days to Dubai at Golden Sands Resort      | 2023-11-05     |                1 |                  1
       5 | 7 Days to Singapore at Marina Bay Sands     | 2023-10-15     |                2 |                  2
       6 | 7 Days to Los Angeles at Palm Paradise Hotel | 2023-09-25    |                1 |                  0
       7 | 7 Days to Tokyo at Urban View Suites        | 2023-12-01     |                2 |                  1
       8 | 7 Days to London at Cityscape Suites        | 2023-10-10     |                2 |                  2
(8 rows)
```

- **4.2.5 Queries for reporting:**

Top 5 Revenue-Generating Packages:

```
SELECT package_description, total_revenue
FROM TotalRevenuePerPackage
ORDER BY total_revenue DESC
LIMIT 5;
```

Output:

```
travel_booking=# SELECT package_description, total_revenue
travel_booking-# FROM TotalRevenuePerPackage
travel_booking-# ORDER BY total_revenue DESC
travel_booking-# LIMIT 5;
             package_description              | total_revenue
----------------------------------------------+---------------
 7 Days to London at Cityscape Suites         |       1719.90
 7 Days to Singapore at Urban Oasis Suites    |       1608.00
 7 Days to Los Angeles at Palm Paradise Hotel |       1401.30
 7 Days to Sydney at Coastal Comfort Resort   |        715.20
 7 Days to Tokyo at Skyline Lodge             |        385.33
(5 rows)
```

Most Popular Package:

```
SELECT most_popular_package AS package_name, total_bookings AS booking_count

FROM MostPopularPackage;
```

Output:

```
travel_booking=# SELECT most_popular_package AS package_name, total_bookings AS booking_count
FROM MostPopularPackage;
            package_name             | booking_count
-------------------------------------+---------------
 7 Days to London at Cityscape Suites |             2
(1 row)
```

Employee Performance Report:

```
SELECT emp_id, emp_username, total_successful_bookings

FROM EmployeePerformance;
```

Output:

```
travel_booking=# SELECT emp_id, emp_username, total_successful_bookings
FROM EmployeePerformance;
 emp_id | emp_username | total_successful_bookings
--------+--------------+---------------------------
      1 | posuere      |                         1
      2 | convallis    |                         1
      3 | odio         |                         1
      4 | felis,       |                         1
      5 | gravida      |                         1
      6 | eleifend     |                         1
      7 | iaculis      |                         1
(7 rows)
```

By creating views for the initial queries, we simplify the process of accessing valuable information from the database and make it easier to use these queries in various analytical scenarios

5. Security, Optimization, Professional, Legal and Ethical Issues:

5.1 Security

Role-based access control: Create roles for employees (e.g., office employees, accountants, managers), and grant appropriate privileges to access specific tables or views. Only authorized personnel should access payment-related information.

Below roles have been created and only system admin has access to assign these for now.

NOTE:

- Roles including creation of views and changing the structure of the table, creation of table has been restricted for all but sysadmin
- This can later be delegated to a centralized DB management team if required later.

<<Code in .sql file>>

Role	Privileges	Description
employee_role	- SELECT on CUSTOMER - SELECT, INSERT, and UPDATE on BOOKING - SELECT, INSERT, and UPDATE on PAYMENT - SELECT, INSERT on	Is assigned to employees from different locations to perform bookings

	PAYMENT_INSTALLMENT - SELECT on PACKAGE, HOTEL, HOTEL_AMENITY, AMENITY, HOTEL_FACILITY, FACILITY, FLIGHT	
accountant_role	- SELECT, INSERT, and UPDATE on PAYMENT - SELECT, INSERT on PAYMENT_INSTALLMENT	Accountants will have access to payments and installment table
manager_role	- SELECT, INSERT, and UPDATE on HOTEL_AMENITY, AMENITY, HOTEL_FACILITY, FACILITY - SELECT, INSERT, UPDATE, and DELETE on HOTEL	Manager will have access to hotels and amenities.
sys_admin_role	-CREATE on SCHEMA, TABLE, VIEWS -ALTER on TABLE	System Administrators have access to the database system.

Permissions for each view:

View Name	Permissions for employee_role	Permissions for accountant_role	Permissions for manager_role
TotalRevenuePerPackage		SELECT	SELECT
MostPopularPackage	SELECT	SELECT	SELECT
BookingDetailsByCustomer	SELECT	SELECT	SELECT
PackageDetailsWithFlightAndHotel	SELECT	SELECT	SELECT
EmployeePerformance		SELECT	SELECT
Installment_Details_With_Customers		SELECT	

New employees will be assigned their designated roles, which will limit their access to the database. By implementing row-level security, it ensures that only the employee who owns the row has access to the data, other employees will not be able to see the other employee's data. (Optional, since other employees might need to access the data during the employee's absence)

- Row-level security: Implement row-level security to ensure that employees can only access data relevant to their responsibilities.

```
--Enable RLS
ALTER TABLE employee ENABLE ROW LEVEL SECURITY;
--Create policy
CREATE POLICY employee_access_policy
ON employee
USING (emp_id = current_setting('myvars.current_user_id')::integer)
WITH CHECK (emp_id = current_setting('myvars.current_user_id')::integer);
--Create a temporary table to store user ID
CREATE TEMPORARY TABLE current_app_user(username text);
--Insert current user ID into table
INSERT INTO current_app_user(username) VALUES (current_user())
--Create trigger that fires after each INSERT, DELETE, OR UPDATE OPERATION
CREATE TRIGGER user_id_trigger AFTER INSERT OR DELETE OR UPDATE ON employee
FOR EACH ROW
BEGIN
--get user id
SELECT username INTO current_user_id FROM current_app_user;
--Set the 'myvars.current_user_id' into the current User ID
SET myvars.current_user_id = current_user_id;
END;
--Enable trigger
ALTER TRIGGER user_id_trigger ENABLE;
--Whenever an INSERT, DELETE, UPDATE operation done on employee table
--Current user ID will be automatically added to myvars.current_user_id
```

- Views with limited information: Create `VIEWS` for specific roles that display only the necessary information while hiding sensitive data like payment details.
- After assigning privileges to the roles, `ALTER USER` command is used to implement username into their role, this will limit their access to the database. (This will be done by the administrators of the database.)

Implementing security methods such as permissions, privileges, and roles on the views can help ensure data security and control access to sensitive information. Appropriate roles were assigned with only specific views.

Code: updated in Code.sql file

Encryption: One way to use encrypted passwords is to implement one-time passwords. This will randomize the password required and make it hard to figure out the original password.

Using One-time Password:

```
CREATE FUNCTION verify_login_and_password(username TEXT, password TEXT) RETURNS
BOOLEAN AS $$
DECLARE
    otp TEXT;
    salt TEXT;
BEGIN
    -- Generate a random salt.
    salt := gen_salt('bf');
    -- Generate a one-time password using the salt.
    otp := pgcrypto(password, salt, 'md5');
    -- Check if the username and password match the one-time password.
    IF username = otp AND password = otp THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
--The OTP will be sent to an external program/application to be displayed for the
employee.
```

`gen_salt` will generate a random salt, then `pgcrypto` will generate a one-time password(otp) using the salt. Lastly, it will check if the username and password matches the one-time password.

- The OTP can be sent to the employee's email address.
- The OTP can be generated with an authenticator app.
- The OTP can be displayed on a physical token.

Chan Brothers Pte Ltd. should pick the method based on its own requirements.

Audit Trails:

Chan Brothers Pte Ltd. should implement their own database audit tracker. Making use of triggers and extensions can allow easier logging for the administrators to track database activities.

Triggers:

PostgreSQL offers auditing capabilities through the Audit Extension (pgaudit). This allows Chan Brothers Pte Ltd. to capture various events like logins, logouts, queries and more.

Firstly Install **pgaudit extension** if not done already:

```
--Installing the extension
CREATE EXTENSION IF NOT EXISTS pgaudit;
```

Configuration to capture desired events:

```
--Configuring the extension
ALTER SYSTEM SET pgaudit.log='ddl, role';
--reload config
SELECT pg_reload_conf();
```

Database Triggers can be set to capture specific events and perform actions accordingly.

Create a new logging table to store Login attempts:

```
--Configuring the extension
ALTER SYSTEM SET pgaudit.log='ddl, role';
--reload config
SELECT pg_reload_conf();
```

Create a **trigger** on **`pg_stat_activity`** to view to call the trigger function when a login attempt occurs:

```
CREATE TRIGGER on_login_attempt
AFTER INSERT OR UPDATE ON pg_stat_activity
FOR EACH ROW
WHEN (NEW.state = 'active')
EXECUTE FUNCTION log_login_attempt_with_ip();
```

Upon execution, another function "**log_login_attempt_with_ip()**" will be called:

```
CREATE OR REPLACE FUNCTION log_login_attempt_with_ip() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO login_log (emp_username, login_time, login_success, ip_address)
    VALUES (NEW.emp_username, NOW(), NEW.state = 'active', NEW.client_addr);
    RETURN NULL;
END;
```

This function will log the username, login time, login status and ip address of each login attempt.

The admins can follow up by:

1. Investigate the login attempts. With the log enabled, they can verify the ip address, times attempted, time and usernames that were used.

2. Block the ip address from connecting. This will stop the attacker from trying to attempt using the same ip address

3. Forcibly reset the account's passwords. The attacker will not be able to access the account as the password will no longer work.

4. Monitor the login log for further attempts. Track if the attackers is still trying to gain access to the database.

5. Further steps that can be done.

a) Enforce strong passwords, disallowing default password like *password123.*
b) Enable a Two-factor Authentication for employees Keeping the security software up to date, with security patches. Setup firewalls to block unauthorized traffic into the system

5.2 Optimization

Transactional Analysis:

A Transactional Analysis Matrix is a tool used to analyze and manage database transactions. It helps identify the necessary CRUD operations for each table or entity in the database.

Common types of Transactions in Chan Brothers' Use Case:

1. Booking a Package
2. Processing Payment
3. Tracking Payment Installments
4. Checking Payment Status
5. Searching Packages
6. Retrieving Customer Information
7. Managing Employee Information
8. Viewing Hotel Details
9. Updating Package Details
10. Modifying Flight Information

Below is the IRUD matrix for each of the transactions against each table.

The table focuses on the important tables that play a key role in the main transactions of the Chan Brothers' database.

Transaction Analysis

Table -->	Customer				Installment				Payment				Package				Employee				Hotel				Amenity				Facility				Flight				Booking				Cost of each
Common Transactions	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	Transaction
Booking a Package	1	1	1		1	1	1		1	1	1																										1	1	1		12
Processing Payment		1			1	1	1		1	1	1							1																			1	1	1		11
Tracking Payment Installments	1					1				1				1				1								1								1								7
Checking Payment Status		1				1				1								1																								4
Searching Packages														1				1				1				1				1				1								6
Retrieving Customer Info	1	1																																								2
Managing Employee Info																	1	1	1	1																						4
Viewing Hotel Details														1								1				1				1												4
Updating Package Details													1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1					24
Modifying Flight Info																																	1	1	1	1					4
Cost per transaction	3	4	1	0	2	4	2	0	2	4	2	0	1	4	1	1	0	6	2	0	0	3	1	1	0	4	1	0	0	3	1	1	2	4	2	0	2	2	2	0	78
Cost of each table	8				8				8				7				8				5				5				5				8				6				

Based on the above analysis, below are the top 3 transactions:

I. **Booking a Package:** This transaction involves inserting data into the Booking table, which includes foreign key references to the Customer and Package tables. It also requires reading data from the Package and Hotel tables to check package availability and amenities.

II. **Processing Payment:** This transaction involves inserting and updating data into Payment, Installment and Booking tables. Furthermore, it reads all three tables mentioned which will take a lot of performance as the database scales up.

III. **Updating Package Details:** This transaction involves updating data in the Package table, which includes foreign key references to the Hotel and Flight tables.

Additionally, below is also vital given that the employees would frequently need to execute this query to cater to customer's queries:

IV. **Searching Packages:** This transaction requires reading data from the Hotel, Hotel_Amenity, Amenity, flight and associated tables to provide information about the whole package.

To ensure these transactions perform well, **we can employ various database optimization techniques as listed below:**

Indexes

Creating appropriate indexes on the frequently queried columns and views can significantly improve the performance of read operations.

Based on the frequently used query/view: "**PackageDetailsWithFlightAndHotel**", the following columns are frequently involved in filtering and joining:

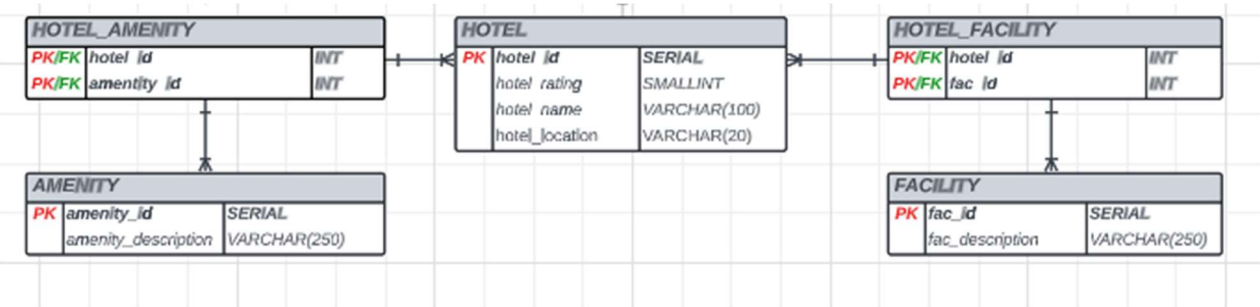
- p.pack_name
- f.out_flight_no
- f.out_destination
- f.out_depart_date
- f.out_arrival_date
- h.hotel_name
- h.hotel_location

Considering this, you can create indexes on the columns that are most likely to be used in WHERE clauses, JOIN conditions, and ORDER BY clauses. B**elow indexes were created as part of the code to optimize the performance:**

Index Name	Table Name	Description
idx_package_pack_name	package	Index on column pack_name
idx_flight_out_flight_no	flight	Index on column out_flight_no
idx_flight_out_destination	flight	Index on column out_destination
idx_flight_out_depart_date	flight	Index on column out_depart_date
idx_flight_out_arrival_date	flight	Index on column out_arrival_date
idx_hotel_hotel_name	hotel	Index on column hotel_name
idx_hotel_hotel_location	hotel	Index on column hotel_location
idx_pay_book_id	payment	Index on column book_id
idx_pay_status	payment	Index on column pay_status
idx_payinstallment_payid	payment_installment	Index on column pay_id

Normalization

Ensuring that the database is properly normalized can help reduce redundancy and improve data integrity. This involves breaking down the data into separate tables and linking them using foreign keys. For example, normalizing the **`hotel_amenity`** and **`hotel_facility`** tables can improve data storage efficiency and simplify queries.



Caching: Implementing caching mechanisms can enhance the performance of read-heavy transactions like viewing hotel amenities. In PostgreSQL, it employs TWO main types of caching:

1. Shared Buffer Cache
    a. Often referred simply as 'cache', is a portion of the system's memory (RAM) dedicated to caching database pages; which contains data and indexes.
    b. When a query or modification is executed, PSQL checks the cache first if the required data is in the memory.
    c. If it is, it is a 'hit' and will be retrieved from there rather than reading from disk.
    d. PSQL uses clock-sweep algorithm; It keeps frequently accessed data in memory, while removing less frequent used data. *(Egor Rogov, WAL in PostgreSQL: 1. Buffer Cache)*

2. Operating System File System Cache
    a. In addition to Shared Buffer Cache, PSQL also benefit from file system cache provided by the operating system (Linux).
    b. When reading/writing to the database, the OS may cache some of this data in memory.
    c. Future reads of the same data will be from the OS cache. *(refer to Reference 12.)*

```
travel_booking=# EXPLAIN ANALYZE SELECT
    c.cust_city AS from_location,
    p.pack_id AS package_id,
    p.pack_name AS package_name,
    pd.package_description AS package_description,
    pd.flight_no AS onward_flight_no,
    pd.flight_destination AS destination,
    pd.departure_date AS onward_depart_date,
    pd.arrival_date AS destination_arrival_date
FROM
    CUSTOMER c
JOIN
    BOOKING b ON c.cust_id = b.cust_id
JOIN
    PACKAGE p ON b.pack_id = p.pack_id
JOIN
    PackageDetailsWithFlightAndHotel pd ON p.pack_name = pd.package_description;
                                                                     QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=46.33..78.76 rows=1000 width=1183) (actual time=0.225..0.231 rows=7 loops=1)
   Hash Cond: (b.cust_id = c.cust_id)
   ->  Hash Join  (cost=35.20..64.95 rows=1000 width=1069) (actual time=0.167..0.171 rows=7 loops=1)
         Hash Cond: (b.pack_id = p.pack_id)
         ->  Seq Scan on booking b  (cost=0.00..20.00 rows=1000 width=8) (actual time=0.004..0.005 rows=7 loops=1)
         ->  Hash  (cost=33.70..33.70 rows=120 width=1065) (actual time=0.149..0.150 rows=7 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Hash Join  (cost=20.20..33.70 rows=120 width=1065) (actual time=0.140..0.147 rows=7 loops=1)
                     Hash Cond: (p_1.hotel_id = h.hotel_id)
                     ->  Hash Join  (cost=16.95..30.13 rows=120 width=1069) (actual time=0.089..0.095 rows=7 loops=1)
                           Hash Cond: (p_1.flight_id = f.flight_id)
                           ->  Hash Join  (cost=12.70..25.55 rows=120 width=1044) (actual time=0.027..0.031 rows=7 loops=1)
                                 Hash Cond: ((p.pack_name)::text = (p_1.pack_name)::text)
                                 ->  Seq Scan on package p  (cost=0.00..11.20 rows=120 width=520) (actual time=0.003..0.004 rows=7 loops=1)
                                 ->  Hash  (cost=11.20..11.20 rows=120 width=524) (actual time=0.013..0.014 rows=7 loops=1)
                                       Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                       ->  Seq Scan on package p_1  (cost=0.00..11.20 rows=120 width=524) (actual time=0.002..0.003 rows=7 loops=1)
                           ->  Hash  (cost=3.00..3.00 rows=100 width=33) (actual time=0.045..0.045 rows=100 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 15kB
                                 ->  Seq Scan on flight f  (cost=0.00..3.00 rows=100 width=33) (actual time=0.005..0.020 rows=100 loops=1)
                     ->  Hash  (cost=2.00..2.00 rows=100 width=4) (actual time=0.034..0.034 rows=100 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 12kB
                           ->  Seq Scan on hotel h  (cost=0.00..2.00 rows=100 width=4) (actual time=0.006..0.015 rows=100 loops=1)
   ->  Hash  (cost=10.50..10.50 rows=50 width=122) (actual time=0.043..0.043 rows=34 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 10kB
         ->  Seq Scan on customer c  (cost=0.00..10.50 rows=50 width=122) (actual time=0.016..0.021 rows=34 loops=1)
 Planning Time: 0.456 ms
 Execution Time: 0.331 ms
(28 rows)
```

```
                                                                     QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=19.60..46.38 rows=1000 width=1183) (actual time=0.212..0.220 rows=7 loops=1)
   Hash Cond: (b.cust_id = c.cust_id)
   ->  Hash Join  (cost=8.48..32.58 rows=1000 width=1069) (actual time=0.143..0.148 rows=7 loops=1)
         Hash Cond: (b.pack_id = p.pack_id)
         ->  Seq Scan on booking b  (cost=0.00..20.00 rows=1000 width=8) (actual time=0.007..0.008 rows=7 loops=1)
         ->  Hash  (cost=8.39..8.39 rows=7 width=1065) (actual time=0.120..0.122 rows=7 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Hash Join  (cost=4.85..8.39 rows=7 width=1065) (actual time=0.093..0.116 rows=7 loops=1)
                     Hash Cond: ((p_1.pack_name)::text = (p.pack_name)::text)
                     ->  Hash Join  (cost=3.69..7.13 rows=7 width=545) (actual time=0.063..0.083 rows=7 loops=1)
                           Hash Cond: (f.flight_id = p_1.flight_id)
                           ->  Seq Scan on flight f  (cost=0.00..3.00 rows=100 width=33) (actual time=0.004..0.012 rows=100 loops=1)
                           ->  Hash  (cost=3.60..3.60 rows=7 width=520) (actual time=0.045..0.046 rows=7 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                 ->  Hash Join  (cost=1.16..3.60 rows=7 width=520) (actual time=0.028..0.041 rows=7 loops=1)
                                       Hash Cond: (h.hotel_id = p_1.hotel_id)
                                       ->  Seq Scan on hotel h  (cost=0.00..2.00 rows=100 width=4) (actual time=0.005..0.011 rows=100 loops=1)
                                       ->  Hash  (cost=1.07..1.07 rows=7 width=524) (actual time=0.009..0.010 rows=7 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                             ->  Seq Scan on package p_1  (cost=0.00..1.07 rows=7 width=524) (actual time=0.002..0.004 rows=7 loops=1)
                     ->  Hash  (cost=1.07..1.07 rows=7 width=520) (actual time=0.018..0.019 rows=7 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  Seq Scan on package p  (cost=0.00..1.07 rows=7 width=520) (actual time=0.005..0.007 rows=7 loops=1)
   ->  Hash  (cost=10.50..10.50 rows=50 width=122) (actual time=0.056..0.056 rows=34 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 10kB
         ->  Seq Scan on customer c  (cost=0.00..10.50 rows=50 width=122) (actual time=0.004..0.042 rows=34 loops=1)
 Planning Time: 1.208 ms
 Execution Time: 0.294 ms
(28 rows)
```

Comparing the 2 runtime, both are using the same operation plans but are executed differently. The 1st time is done in 0.331ms while the 2nd operation ran at 0.296ms. This will improve efficiency since it does not need to read from the disk, allowing better resource management.

Partitioning: If the size of the Booking or Payment table grows significantly, partitioning the data based on a key such as <u>date</u> or <u>location</u> can improve query performance. Another way is to partition a table by <u>range</u>.

This technique allows the database to focus on specific partitions when executing queries, reducing the search space.

This technique can be implemented in the future as the database grows, allowing for scaling.

Query Optimization: Use `EXPLAIN ANALYZE` to see how the database plans to execute the query. It is easier to identify ways to improve the performance of the query.

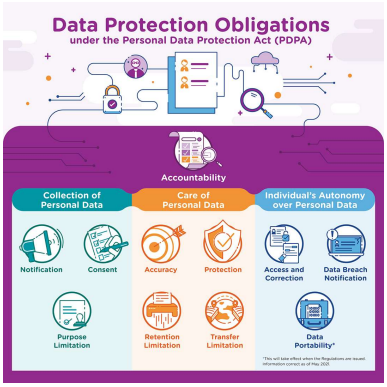
```
                                                       QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=1.42..6.64 rows=1 width=572) (actual time=0.053..0.076 rows=7 loops=1)
   ->  Nested Loop  (cost=1.28..4.73 rows=1 width=549) (actual time=0.047..0.061 rows=7 loops=1)
         ->  Hash Join  (cost=1.14..2.24 rows=1 width=524) (actual time=0.032..0.037 rows=7 loops=1)
               Hash Cond: ((p_1.pack_name)::text = (p.pack_name)::text)
               ->  Seq Scan on package p_1  (cost=0.00..1.07 rows=7 width=524) (actual time=0.010..0.011 rows=7 loops=1)
               ->  Hash  (cost=1.12..1.12 rows=1 width=516) (actual time=0.014..0.014 rows=7 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on package p  (cost=0.00..1.12 rows=1 width=516) (actual time=0.006..0.009 rows=7 loops=1)
                           Filter: ((pack_duration >= 5) AND (pack_duration <= 8) AND (pack_pricepp <= '1000'::numeric))
         ->  Index Scan using flight_pkey on flight f  (cost=0.14..2.45 rows=1 width=33) (actual time=0.003..0.003 rows=1 loops=7)
               Index Cond: (flight_id = p_1.flight_id)
   ->  Index Scan using hotel_pkey on hotel h  (cost=0.14..1.87 rows=1 width=31) (actual time=0.001..0.001 rows=1 loops=7)
         Index Cond: (hotel_id = p_1.hotel_id)
 Planning Time: 0.333 ms
 Execution Time: 0.119 ms
(15 rows)
```

By using the transaction analysis, Chan Brothers Pte Ltd. will be able to prepare the required system and its resources to run the database efficiently and smoothly. Techniques like indexing, partitioning and Query optimizing can help the database perform better without the need to change any physical hardware. PostgreSQL has multiple caching functions that will greatly help users during execution operations; the administrators can also specify how much resources can be allocated into certain operations.

Above information has been referenced in references section

5.3 Professional, Legal and Ethical Issues:

Chan Brothers Travel Pte Ltd. in Singapore has to follow the rules of the Personal Data Protection Act (PDPA). This law makes sure people's private details are kept safe. This section talks about how the company should handle people's data in a professional, legal, and right way.



Professional:

- **Data Protection Officer (DPO):** The company must have a designated individual or team, the DPO, to oversee PDPA compliance. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p. 9)*
- **Data Protection Policies**: Draft and execute comprehensive data protection policies. Ensure that all employees are trained and regularly updated about these policies. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.15)*
- **Secure Data Storage:** Adopt robust security measures like encryption, secure servers, and regular backups to shield personal data. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.13)*
- **Limit Data Access:** Implement role-based access controls to ensure only authorized personnel access the necessary data. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.13)*
- **Regular Compliance Audits**: Periodically evaluate the company's adherence to data protection practices. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.31)*

Legal:

- **Consent for Data Collection:** Clearly inform individuals about data collection intentions, ensuring that consent is voluntarily given, specific, informed, and unambiguous. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.21)*

- **Retention and Disposal:** Define a data retention period based on operational necessities and legal criteria. Securely dispose of obsolete data. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.20)*
- **Overseas Data Transfer:** When transmitting personal data beyond Singapore, comply with the protective measures outlined by the PDPA. *(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.20)*
- **Data Breach Management**: Establish a comprehensive strategy to address data breaches, mandating prompt notifications to the Personal Data Protection Commission and the affected individuals.*(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.28)*
- **Vendor Management:** Ensure third-party associates handling personal data adhere to data protection standards, reinforcing this through contractual clauses.*(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.20, 26)*

Ethical:

- **Trust and Reputation:** Beyond mere compliance, foster trust and uphold the company's reputation. This commitment should be foundational to the company's ethos.*(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.6, 21)*
- **Transparent Communication:** Regularly inform individuals about data collection, use, and disclosure through clear data protection notices and privacy policies.*(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.21)*
- **Responding to Data Access Requests:** Efficiently manage requests from individuals looking to access, modify, or retract their consent for their personal data.*(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.17)*
- **Stay Informed**: Proactively monitor changes to Singapore's data protection laws, ensuring policies and procedures remain current and ethical.*(Personal Data Protection Council, GUIDE TO DEVELOPING A DATA PROTECTION MANAGEMENT PROGRAMME 2023, p.31)*

6.Conclusion & Future Enhancements:

In conclusion, this assignment outlines the design, development, and implementation of a centralized database for Chan Brothers Travel Pte Ltd. The Enhanced Entity Relationship Diagram (EERD) provides a clear representation of the data and its relationships, while the Data Dictionary defines the constraints and descriptions for each entity. The physical implementation showcases the SQL code for creating the tables and relationships, and the sample data enables testing of the database's functionality. Furthermore, the included queries demonstrate the usability of the database in catering to the business's needs.

Future Enhancements:

- **Enhanced Role-Based Access Control:** Define custom roles for specific user groups, e.g., "Sales Team" or "Accounting Team," to ensure appropriate access rights.
- **Automated Data Backup and Recovery:** Set up automated data backups to ensure data integrity and provide an efficient way to restore data in case of failures.
- **Automated Notifications:** Set up automated notifications to alert users about booking confirmations, payment due dates, and other important updates.
- **External Data Integration:** Implement a mechanism to automatically refresh flight data from external sources (like airlines) to ensure accurate and up-to-date flight information.
- **Scalability Planning:** Plan for future growth by designing the system architecture to handle increasing data volumes and user traffic.

7. References:

1. PostgreSQL Official Documentation: - Website: https://www.postgresql.org/docs/

2. W3Schools SQL Tutorial: - Website: https://www.w3schools.com/sql/

3. MySQL Official Documentation: - Website: https://dev.mysql.com/doc/

4. Microsoft SQL Server Documentation: - Website: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

5. Oracle Database SQL Language Reference: - Website: https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/index.html

6. Database Systems: The Complete Book

7. PDPA Overview:Website:https://www.pdpc.gov.sg/Overview-of-PDPA/The-Legislation/Personal-Data-Protection-Act

8. PDPA: Guide to developing a Data Management Programme

- Website: https://www.pdpc.gov.sg/help-and-resources/2019/07/guide-to-developing-a-data-protection-management-programme
- https://www.pdpc.gov.sg/-/media/Files/PDPC/PDF-Files/Other-Guides/DPMP/Guide-to-Developing-a-Data-Protection-Management-Programme-(Aug-2023).pdf
- PDPC. (n.d.). *PDPA Protection Obligation*. Retrieved from https://www.pdpc.gov.sg/-/media/Images/PDPC/PAW2021/IMDA-PAW_Infographic-Post_FA-v2.jpg?h=100%25&w=100%25

9. Github: PGaudit extension Website: https://github.com/pgaudit/pgaudit

10. *Database er diagram: Lucidchart*. Database ER diagram | Lucidchart. (n.d.).

https://lucid.app/publicSegments/view/89b86a56-24e6-483b-a6e0-116b89c43bba/image.jpeg

References for Optimization:

PostgreSQL Wiki, "Performance Optimization."
https://wiki.postgresql.org/wiki/Performance_Optimization

Craig Kerstiens, "PostgreSQL Indexing - How Many Indexes Should You Create?"
https://www.citusdata.com/blog/2018/03/07/choosing-postgres-index-types

Compose Articles, "Making the Most of PostgreSQL Connection Pooling."
https://www.compose.com/articles/making-the-most-of-postgresql-connection-pooling

Ken Rugg, "When Should You Partition Your PostgreSQL Database?"
https://info.crunchydata.com/blog/when-should-you-partition-your-postgresql-database

Compose Articles, "Speeding up PostgreSQL with Partial Indexes."
https://www.compose.com/articles/speeding-up-postgresql-with-partial-indexes/

Jignesh Shah, "Top 10 PostgreSQL Performance Tuning Tips."

https://www.cybertec-postgresql.com/en/top-10-postgresql-performance-tuning-tips