Travel Planner Database Design & Architecture

> ***Version:*** *2.0 (Simplified)*
> ***Database:*** *Oracle 21c (Free Edition)*
> ***Last Updated:*** *2025*

Table of Contents

---

1. Overview

1.1 Business Domain

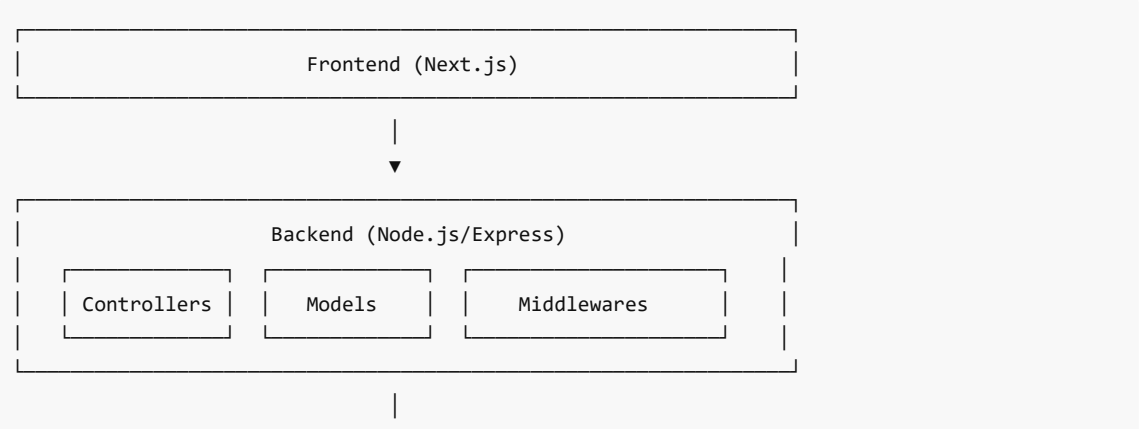
The Travel Planner system is an **Online Transaction Processing (OLTP)** application for a travel agency. It allows customers to:

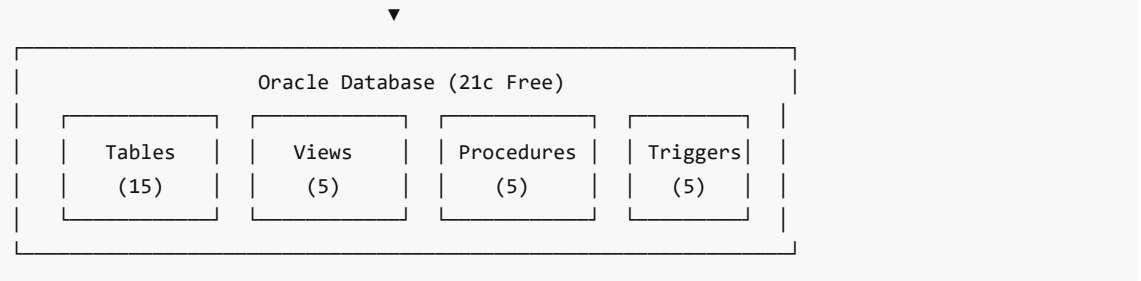
- Browse pre-made travel packages (catalogs)
- Create custom bookings with individual services
- Book hotels, transport, and food plans
- Process payments for bookings

Administrators can:

- Manage travel packages and services
- View and manage all bookings
- Track payments and revenue
- Access customer analytics

1.2 System Architecture





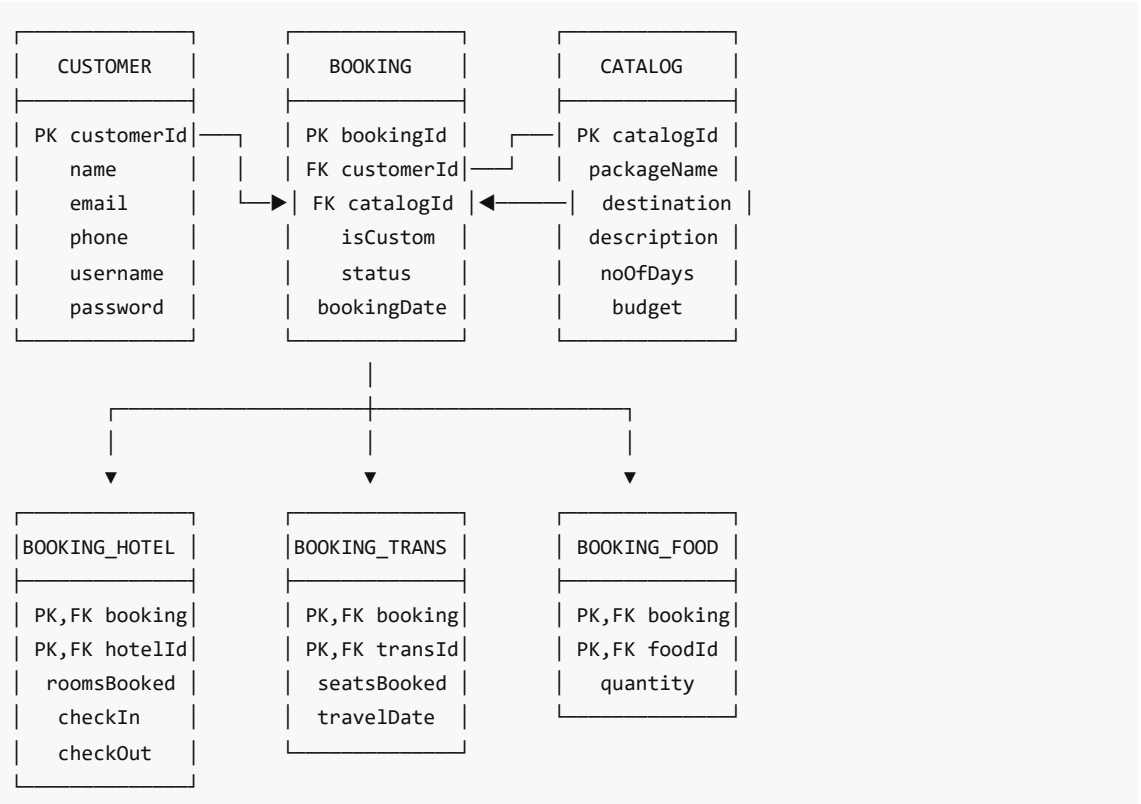
1.3 Database Statistics

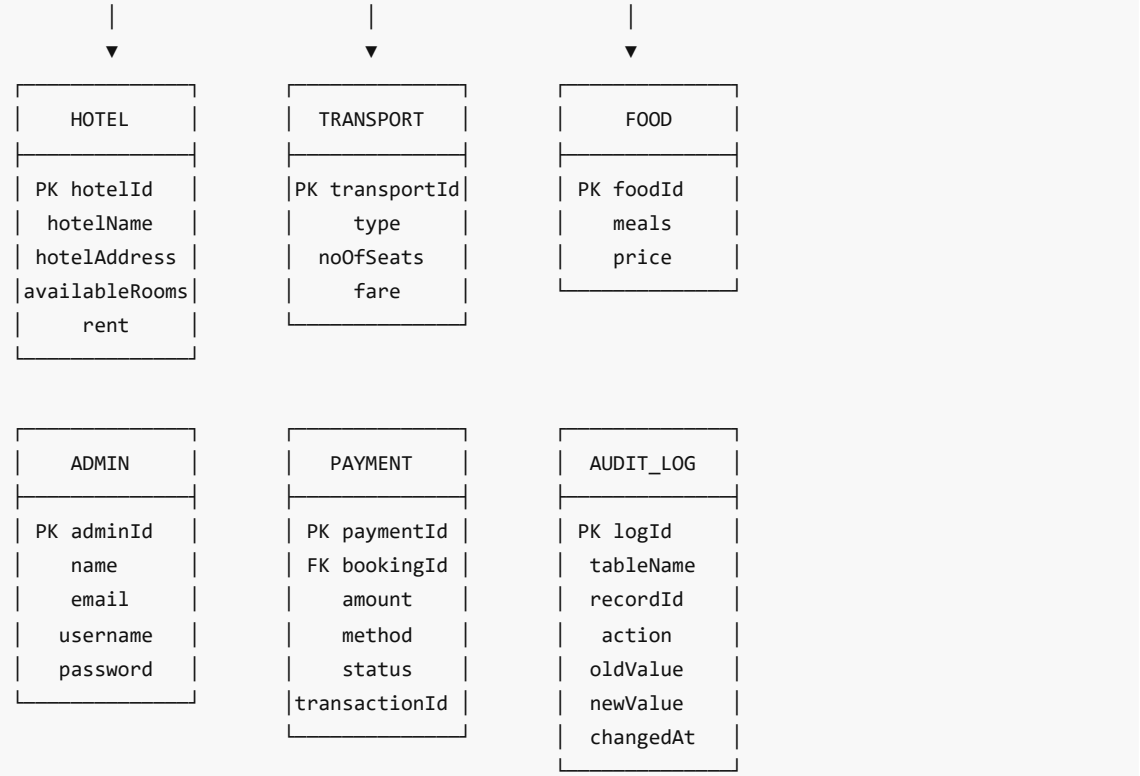
Object Type	Count
Tables	15
Views	5
Stored Procedures	5
Triggers	5
Foreign Keys	14
Check Constraints	6

---

2. Entity-Relationship Model

2.1 ER Diagram (Crow's Foot Notation)





2.2 Entity Descriptions

Entity	Description	Cardinality
Customer	End users who make bookings	1 Customer → Many Bookings
Admin	System administrators	Independent entity
Catalog	Pre-made travel packages	1 Catalog → Many Bookings
Hotel	Hotel accommodations	Many Hotels ↔ Many Catalogs/Bookings
Transport	Transportation options	Many Transports ↔ Many Catalogs/Bookings
Food	Meal plans	Many Foods ↔ Many Catalogs/Bookings
Booking	Customer reservations	1 Booking → 1 Payment
Payment	Financial transactions	1 Payment → 1 Booking
Audit_Log	System change tracking	Independent audit trail

2.3 Relationship Summary

Relationship	Type	Description
Customer → Booking	1:N	A customer can have many bookings
Catalog → Booking	1:N	A catalog can be booked multiple times
Booking → Payment	1:1	Each booking has one payment

Catalog ↔ Hotel	M:N	Via Catalog_Hotel junction table
Catalog ↔ Transport	M:N	Via Catalog_Transport junction table
Catalog ↔ Food	M:N	Via Catalog_Food junction table
Booking ↔ Hotel	M:N	Via Booking_Hotel junction table
Booking ↔ Transport	M:N	Via Booking_Transport junction table
Booking ↔ Food	M:N	Via Booking_Food junction table

---

3. Database Schema

3.1 Schema Diagram

```
travel_planner (schema)
│
├── Core Entities (8 tables)
│   ├── Customer
│   ├── Admin
│   ├── Catalog
│   ├── Hotel
│   ├── Transport
│   ├── Food
│   ├── Booking
│   └── Payment
│
├── Junction Tables (6 tables)
│   ├── Catalog_Hotel
│   ├── Catalog_Transport
│   ├── Catalog_Food
│   ├── Booking_Hotel
│   ├── Booking_Transport
│   └── Booking_Food
│
├── Audit Tables (1 table)
│   └── Audit_Log
│
├── Views (5)
│   ├── VW_CATALOG_FULL_DETAILS
│   ├── VW_BOOKING_DETAILS
│   ├── VW_CUSTOMER_BOOKING_SUMMARY
│   ├── VW_REVENUE_REPORT
│   └── VW_HOTEL_AVAILABILITY
│
├── Stored Procedures (5)
│   ├── SP_CREATE_BOOKING_FROM_CATALOG
│   ├── SP_CALCULATE_BOOKING_TOTAL
│   ├── SP_PROCESS_PAYMENT
│   ├── SP_COMPLETE_PAYMENT
```

```
│   └── SP_CANCEL_BOOKING
│
└── Triggers (5)
    ├── TRG_BOOKING_AUDIT
    ├── TRG_AUTO_CONFIRM_ON_PAYMENT
    ├── TRG_PREVENT_DOUBLE_BOOKING
    ├── TRG_CUSTOMER_EMAIL_LOWERCASE
    └── TRG_CASCADE_BOOKING_CANCEL
```

---

4. Tables

4.1 Customer

Stores customer/user information for the travel platform.

Column	Data Type	Constraints	Description
customerId	NUMBER	PK, IDENTITY	Auto-generated unique ID
name	VARCHAR2(100)	NOT NULL	Customer full name
email	VARCHAR2(100)	UNIQUE, NOT NULL	Email address (stored lowercase)
phone	VARCHAR2(20)	-	Phone number
username	VARCHAR2(50)	UNIQUE, NOT NULL	Login username
password	VARCHAR2(255)	NOT NULL	Bcrypt hashed password

4.2 Admin

Stores administrator accounts.

Column	Data Type	Constraints	Description
adminId	NUMBER	PK, IDENTITY	Auto-generated unique ID
name	VARCHAR2(100)	NOT NULL	Admin full name
email	VARCHAR2(100)	UNIQUE, NOT NULL	Email address
username	VARCHAR2(50)	UNIQUE, NOT NULL	Login username
password	VARCHAR2(255)	NOT NULL	Bcrypt hashed password

4.3 Catalog

Pre-made travel packages offered by the agency.

Column	Data Type	Constraints	Description
catalogId	NUMBER	PK, IDENTITY	Auto-generated unique ID
packageName	VARCHAR2(200)	NOT NULL	Package display name

destination	VARCHAR2(200)	NOT NULL	Travel destination
description	VARCHAR2(1000)	-	Package description
noOfDays	NUMBER	NOT NULL	Trip duration in days
budget	NUMBER(10,2)	NOT NULL	Advertised budget/price
departure	DATE	NOT NULL	Departure date
arrival	DATE	NOT NULL	Return date

4.4 Hotel

Hotel accommodation options.

Column	Data Type	Constraints	Description
hotelId	NUMBER	PK, IDENTITY	Auto-generated unique ID
hotelName	VARCHAR2(200)	NOT NULL	Hotel name
hotelAddress	VARCHAR2(500)	NOT NULL	Full address
availableRooms	NUMBER	NOT NULL	Total available rooms
rent	NUMBER(10,2)	NOT NULL	Price per room per night

4.5 Transport

Transportation options (bus, van, flight, train).

Column	Data Type	Constraints	Description
transportId	NUMBER	PK, IDENTITY	Auto-generated unique ID
type	VARCHAR2(50)	NOT NULL, CHECK	Type: bus/van/flight/train
noOfSeats	NUMBER	NOT NULL	Total seats available
fare	NUMBER(10,2)	NOT NULL	Price per seat

Check Constraint: `type IN ('bus', 'van', 'flight', 'train')`

4.6 Food

Meal plan options.

Column	Data Type	Constraints	Description
foodId	NUMBER	PK, IDENTITY	Auto-generated unique ID
meals	VARCHAR2(500)	NOT NULL	Meal plan description
price	NUMBER(10,2)	NOT NULL	Price per day

4.7 Booking

Customer reservations (from catalog or custom).

Column	Data Type	Constraints	Description
bookingId	VARCHAR2(36)	PK	UUID generated by application
customerId	NUMBER	FK, NOT NULL	Reference to Customer
catalogId	NUMBER	FK, NULL	Reference to Catalog (NULL for custom)
isCustom	NUMBER(1)	DEFAULT 0, CHECK	0=package, 1=custom booking
bookingDescription	VARCHAR2(1000)	-	Notes/special requests
bookingDate	DATE	DEFAULT SYSDATE	When booking was made
status	VARCHAR2(20)	DEFAULT 'pending', CHECK	Booking status

Check Constraints:

- `isCustom IN (0, 1)`
- `status IN ('pending', 'confirmed', 'cancelled')`

Foreign Keys:

- `customerId` → Customer(customerId) ON DELETE CASCADE
- `catalogId` → Catalog(catalogId) ON DELETE SET NULL

4.8 Payment

Financial transactions for bookings.

Column	Data Type	Constraints	Description
paymentId	VARCHAR2(36)	PK	UUID generated by application
bookingId	VARCHAR2(36)	FK, NOT NULL	Reference to Booking
amount	NUMBER(10,2)	NOT NULL	Payment amount
paymentDate	DATE	DEFAULT SYSDATE	When payment was made
method	VARCHAR2(50)	NOT NULL, CHECK	Payment method
status	VARCHAR2(20)	DEFAULT 'pending', CHECK	Payment status
transactionId	VARCHAR2(100)	-	External transaction reference

Check Constraints:

- `method IN ('credit_card', 'debit_card', 'paypal', 'bank_transfer')`
- `status IN ('pending', 'completed', 'failed', 'refunded')`

4.9 Junction Tables

Catalog_Hotel

Links catalogs to included hotels.

Column	Data Type	Constraints
catalogId	NUMBER	PK, FK
hotelId	NUMBER	PK, FK
roomsIncluded	NUMBER	DEFAULT 1

Catalog_Transport

Links catalogs to included transport.

Column	Data Type	Constraints
catalogId	NUMBER	PK, FK
transportId	NUMBER	PK, FK
seatsIncluded	NUMBER	DEFAULT 1

Catalog_Food

Links catalogs to included food plans.

Column	Data Type	Constraints
catalogId	NUMBER	PK, FK
foodId	NUMBER	PK, FK

Booking_Hotel

Links bookings to booked hotels with dates.

Column	Data Type	Constraints
bookingId	VARCHAR2(36)	PK, FK
hotelId	NUMBER	PK, FK
roomsBooked	NUMBER	DEFAULT 1
checkIn	DATE	NOT NULL
checkOut	DATE	NOT NULL

Booking_Transport

Links bookings to booked transport with travel date.

Column	Data Type	Constraints
bookingId	VARCHAR2(36)	PK, FK
transportId	NUMBER	PK, FK
seatsBooked	NUMBER	DEFAULT 1
travelDate	DATE	NOT NULL

Booking_Food

Links bookings to food plans with quantity.

Column	Data Type	Constraints
bookingId	VARCHAR2(36)	PK, FK
foodId	NUMBER	PK, FK
quantity	NUMBER	DEFAULT 1

4.10 Audit_Log

System audit trail for tracking booking changes.

Column	Data Type	Constraints	Description
logId	NUMBER	PK, IDENTITY	Auto-generated ID
tableName	VARCHAR2(50)	NOT NULL	Table that changed
recordId	VARCHAR2(36)	NOT NULL	ID of changed record
action	VARCHAR2(20)	NOT NULL	INSERT/UPDATE/DELETE
oldValue	VARCHAR2(4000)	-	Previous values
newValue	VARCHAR2(4000)	-	New values
changedBy	VARCHAR2(100)	-	User who made change
changedAt	TIMESTAMP	DEFAULT SYSTIMESTAMP	When change occurred

---

5. Views

5.1 VW_CATALOG_FULL_DETAILS

Purpose: Shows catalog packages with all included services and calculated total price.

Usage: Display full package details on frontend, admin reports.

Key Columns:

Column	Description

catalogId, packageName, destination	Basic catalog info
noOfDays, budget, departure, arrival	Trip details
totalHotels	Count of included hotels
totalHotelCost	Sum of hotel costs
totalTransports	Count of included transports
totalTransportCost	Sum of transport costs
totalFoodPlans	Count of included food plans
totalFoodCost	Sum of food costs
calculatedTotalCost	Grand total of all services

Example Query:

```
SELECT packageName, destination, noOfDays, calculatedTotalCost
FROM travel_planner.VW_CATALOG_FULL_DETAILS
WHERE destination LIKE '%Karachi%';
```

5.2 VW_BOOKING_DETAILS

Purpose: Comprehensive booking view with customer info, services, and totals.

Usage: Booking detail pages, admin booking management.

Key Columns:

Column	Description
bookingId, bookingDate, status	Booking info
customerId, customerName, customerEmail	Customer info
catalogId, packageName, destination	Package info (if from catalog)
bookingType	'Custom Booking' or 'Package Booking'
totalHotelCost	Calculated hotel costs
totalTransportCost	Calculated transport costs
totalFoodCost	Calculated food costs
grandTotal	Sum of all costs
paymentStatus, paidAmount	Payment info

Example Query:

```sql
SELECT bookingId, customerName, grandTotal, status, paymentStatus
FROM travel_planner.VW_BOOKING_DETAILS
WHERE status = 'pending';
```

5.3 VW_CUSTOMER_BOOKING_SUMMARY

Purpose: Customer statistics including bookings, spending, and loyalty tier.

Usage: Customer profile page, loyalty programs, admin reports.

Key Columns:

Column	Description
customerId, name, email, phone, username	Customer info
totalBookings	Total number of bookings
confirmedBookings	Count of confirmed bookings
cancelledBookings	Count of cancelled bookings
totalSpent	Total amount spent (completed payments)
customerTier	Bronze/Silver/Gold/Platinum

Tier Calculation:

Tier	Spending Threshold
Platinum	≥ PKR 200,000
Gold	≥ PKR 100,000
Silver	≥ PKR 50,000
Bronze	Default

Example Query:

```sql
SELECT name, totalBookings, totalSpent, customerTier
FROM travel_planner.VW_CUSTOMER_BOOKING_SUMMARY
WHERE customerTier IN ('Gold', 'Platinum');
```

5.4 VW_REVENUE_REPORT

Purpose: Monthly revenue analytics for admin dashboard.

Usage: Admin reports, business analytics.

Key Columns:

Column	Description

paymentMonth	Month in YYYY-MM format
totalTransactions	Count of payment records
totalBookings	Count of bookings
uniqueCustomers	Distinct customer count
completedRevenue	Revenue from completed payments
pendingRevenue	Revenue from pending payments
refundedAmount	Total refunded amount
packageRevenue	Revenue from catalog bookings
customBookingRevenue	Revenue from custom bookings

Example Query:

```
SELECT paymentMonth, completedRevenue, totalBookings
FROM travel_planner.VW_REVENUE_REPORT
WHERE paymentMonth >= '2025-01'
ORDER BY paymentMonth DESC;
```

5.5 VW_HOTEL_AVAILABILITY

Purpose: Real-time hotel room availability.

Usage: Booking forms, availability checks.

Key Columns:

Column	Description
hotelId, hotelName, hotelAddress	Hotel info
totalRooms	Total rooms in hotel
pricePerNight	Room rate
currentlyBookedRooms	Rooms with active bookings
availableRoomsNow	Rooms available for booking

Example Query:

```
SELECT hotelName, totalRooms, availableRoomsNow, pricePerNight
FROM travel_planner.VW_HOTEL_AVAILABILITY
WHERE availableRoomsNow > 0;
```

---

6. Stored Procedures

6.1 SP_CREATE_BOOKING_FROM_CATALOG

Purpose: Creates a complete booking from a catalog package. Copies all hotels, transport, and food from the catalog to the booking in a single atomic transaction.

Why Use This? Instead of multiple INSERT statements in the application, this procedure handles everything in one database call, ensuring data consistency.

Parameters:

Name	Type	Direction	Description
p_customer_id	NUMBER	IN	Customer making the booking
p_catalog_id	NUMBER	IN	Catalog package to book
p_description	VARCHAR2	IN	Optional booking description
p_check_in	DATE	IN	Hotel check-in date
p_check_out	DATE	IN	Hotel check-out date
p_travel_date	DATE	IN	Transport travel date
p_booking_id	VARCHAR2	OUT	Generated booking ID
p_total_cost	NUMBER	OUT	Calculated total cost

What It Does:

1. Validates customer exists
2. Validates catalog exists
3. Validates dates (check-out after check-in, not in past)
4. Creates booking record
5. Copies hotels from catalog → booking
6. Copies transport from catalog → booking
7. Copies food from catalog → booking
8. Calculates total cost
9. Returns booking ID and total

Error Codes:

Code	Description
-20001	Customer not found
-20002	Catalog not found
-20003	Check-out must be after check-in
-20004	Check-in date cannot be in past

Example Usage:

```
DECLARE
    v_booking_id VARCHAR2(36);
```

```
    v_total NUMBER;
BEGIN
    travel_planner.SP_CREATE_BOOKING_FROM_CATALOG(
        p_customer_id => 1,
        p_catalog_id  => 1,
        p_description => 'Family vacation',
        p_check_in    => DATE '2025-12-01',
        p_check_out   => DATE '2025-12-05',
        p_travel_date => DATE '2025-12-01',
        p_booking_id  => v_booking_id,
        p_total_cost  => v_total
    );
    DBMS_OUTPUT.PUT_LINE('Booking: ' || v_booking_id || ', Total: ' || v_total);
END;
```

6.2 SP_CALCULATE_BOOKING_TOTAL

Purpose: Calculates the total cost breakdown of a booking.

Why Use This? Centralizes the cost calculation logic in the database. The same formula is used consistently everywhere.

Parameters:

Name	Type	Direction	Description
p_booking_id	VARCHAR2	IN	Booking to calculate
p_hotel_cost	NUMBER	OUT	Total hotel cost
p_transport_cost	NUMBER	OUT	Total transport cost
p_food_cost	NUMBER	OUT	Total food cost
p_total_cost	NUMBER	OUT	Grand total

Calculation Logic:

- **Hotel cost** = rent × rooms × nights (checkOut - checkIn)
- **Transport cost** = fare × seats
- **Food cost** = price × quantity

Example Usage:

```
DECLARE
    v_hotel NUMBER;
    v_transport NUMBER;
    v_food NUMBER;
    v_total NUMBER;
BEGIN
    travel_planner.SP_CALCULATE_BOOKING_TOTAL(
        'abc-123-def',
        v_hotel, v_transport, v_food, v_total
    );
```

```
    DBMS_OUTPUT.PUT_LINE('Hotel: ' || v_hotel);
    DBMS_OUTPUT.PUT_LINE('Transport: ' || v_transport);
    DBMS_OUTPUT.PUT_LINE('Food: ' || v_food);
    DBMS_OUTPUT.PUT_LINE('TOTAL: ' || v_total);
END;
```

6.3 SP_PROCESS_PAYMENT

Purpose: Creates a payment record for a booking with full validation.

Why Use This? Ensures all payment validation rules are enforced consistently and calculates the correct amount automatically.

Parameters:

Name	Type	Direction	Description
p_booking_id	VARCHAR2	IN	Booking to pay for
p_method	VARCHAR2	IN	Payment method
p_transaction_id	VARCHAR2	IN	External transaction ID (optional)
p_payment_id	VARCHAR2	OUT	Generated payment ID
p_amount	NUMBER	OUT	Payment amount

Valid Payment Methods:

- `credit_card`
- `debit_card`
- `paypal`
- `bank_transfer`

What It Does:

1. Validates booking exists
2. Checks booking is not cancelled
3. Checks no existing payment
4. Validates payment method
5. Calculates amount using SP_CALCULATE_BOOKING_TOTAL
6. Creates payment record

Error Codes:

Code	Description
-20005	Booking not found
-20006	Cannot pay for cancelled booking
-20007	Payment already exists
-20008	Invalid payment method

-20009	No items to pay for

6.4 SP_COMPLETE_PAYMENT

Purpose: Marks a payment as completed and confirms the booking.

Why Use This? Updates both payment and booking status atomically, ensuring they stay in sync.

Parameters:

Name	Type	Direction	Description
p_payment_id	VARCHAR2	IN	Payment to complete
p_transaction_id	VARCHAR2	IN	External transaction ID (optional)

What It Does:

1. Validates payment exists
2. Checks payment is in 'pending' status
3. Updates payment status to 'completed'
4. Updates booking status to 'confirmed'

Error Codes:

Code	Description
-20010	Payment not found
-20011	Payment is not pending

6.5 SP_CANCEL_BOOKING

Purpose: Cancels a booking and handles refund if payment was made.

Why Use This? Handles the complex cancellation logic including refund processing in one procedure.

Parameters:

Name	Type	Direction	Description
p_booking_id	VARCHAR2	IN	Booking to cancel
p_reason	VARCHAR2	IN	Cancellation reason (optional)
p_refund_issued	NUMBER	OUT	1 if refund issued, 0 otherwise

What It Does:

1. Validates booking exists
2. Checks booking is not already cancelled
3. If completed payment exists → marks as 'refunded', sets refund_issued = 1
4. If pending payment exists → marks as 'failed'
5. Updates booking status to 'cancelled'

6. Appends cancellation reason to description

Error Codes:

Code	Description
-20005	Booking not found
-20012	Booking already cancelled

Example Usage:

```
DECLARE
    v_refund NUMBER;
BEGIN
    travel_planner.SP_CANCEL_BOOKING(
        p_booking_id    => 'abc-123-def',
        p_reason        => 'Customer requested cancellation',
        p_refund_issued => v_refund
    );
    IF v_refund = 1 THEN
        DBMS_OUTPUT.PUT_LINE('Refund was issued');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No refund needed');
    END IF;
END;
```

---

7. Triggers

7.1 TRG_BOOKING_AUDIT

Purpose: Logs all changes to the Booking table for audit trail.

Fires: AFTER INSERT, UPDATE, or DELETE on Booking table.

Why Use This? Maintains a complete history of all booking changes for compliance and debugging.

What It Does:

- On INSERT: Logs new booking details (ID, customer, status)
- On UPDATE: Logs old and new status values
- On DELETE: Logs deleted booking details

Example Audit Entry:

```
tableName: BOOKING
recordId: abc-123-def
action: UPDATE
oldValue: Status: pending
newValue: Status: confirmed
changedBy: TRAVEL_PLANNER
changedAt: 2025-01-15 10:30:00
```

7.2 TRG_AUTO_CONFIRM_ON_PAYMENT

Purpose: Automatically confirms a booking when payment is completed.

Fires: AFTER UPDATE on Payment table, when status changes to 'completed'.

Why Use This? Automates the business rule that a completed payment should confirm the booking. No manual intervention needed.

What It Does:

- When payment status changes from anything → 'completed'
- Automatically updates the associated booking status to 'confirmed'
- Only affects bookings that are currently 'pending'

Business Logic:

```
Payment.status = 'completed' → Booking.status = 'confirmed'
```

7.3 TRG_PREVENT_DOUBLE_BOOKING

Purpose: Prevents booking hotel rooms when not enough are available.

Fires: BEFORE INSERT on Booking_Hotel table.

Why Use This? Enforces inventory management at the database level. Even if the application has a bug, the database will prevent overbooking.

What It Does:

1. Counts rooms already booked for overlapping dates
2. Adds the requested rooms to that count
3. If total exceeds available rooms → raises error

Error Code: -20030

Error Message Example:

```
Not enough rooms at Pearl Continental for dates 2025-12-01 to 2025-12-05.
Available: 2, Requested: 3
```

7.4 TRG_CUSTOMER_EMAIL_LOWERCASE

Purpose: Ensures customer email is always stored in lowercase.

Fires: BEFORE INSERT or UPDATE on Customer table (email column).

Why Use This? Normalizes email data for consistent searching and prevents duplicates like "John@Email.com" and "john@email.com".

What It Does:

```
:NEW.email := LOWER(:NEW.email);
```

Example:

- Input: "John.Doe@Gmail.COM"
- Stored: "john.doe@gmail.com"

7.5 TRG_CASCADE_BOOKING_CANCEL

Purpose: When a booking is cancelled, automatically update payment status.

Fires: AFTER UPDATE on Booking table, when status changes to 'cancelled'.

Why Use This? Ensures data consistency. When a booking is cancelled, payments should reflect that.

What It Does:

- Pending payments → status becomes 'failed'
- Completed payments → status becomes 'refunded'

Business Logic:

```
Booking.status = 'cancelled'
    → Payment.status = 'pending' becomes 'failed'
    → Payment.status = 'completed' becomes 'refunded'
```

---

8. Normalization

8.1 First Normal Form (1NF)

All tables satisfy 1NF:

- ✅ All columns contain atomic (indivisible) values
- ✅ Each column contains values of a single type
- ✅ Each row is unique (identified by primary key)
- ✅ No repeating groups

8.2 Second Normal Form (2NF)

All tables satisfy 2NF:

- ✅ All tables are in 1NF
- ✅ All non-key attributes depend on the entire primary key
- ✅ Junction tables have composite primary keys with no partial dependencies

Example: In `Booking_Hotel`, both `bookingId` and `hotelId` are required to determine `roomsBooked`, `checkIn`, `checkOut`.

8.3 Third Normal Form (3NF)

All tables satisfy 3NF:

- ✅ All tables are in 2NF
- ✅ No transitive dependencies exist
- ✅ Non-key attributes depend only on the primary key

Design Decisions:

- Customer information is not stored in Booking (only FK reference)
- Hotel prices are stored in Hotel table, not duplicated in booking

- Catalog budget is separate from calculated service costs

8.4 Normalization Proof

Table	Non-Key Attributes	Depend On	3NF?
Customer	name, email, phone, username, password	customerId	✅
Booking	customerId, catalogId, isCustom, description, date, status	bookingId	✅
Booking_Hotel	roomsBooked, checkIn, checkOut	(bookingId, hotelId)	✅
Payment	bookingId, amount, date, method, status, transactionId	paymentId	✅

---

9. Business Rules

9.1 Booking Rules

Rule	Implementation
Booking requires valid customer	FK constraint + SP validation
Check-out must be after check-in	SP_CREATE_BOOKING_FROM_CATALOG validation
Dates cannot be in the past	SP validation
Cancelled bookings trigger payment updates	TRG_CASCADE_BOOKING_CANCEL
All booking changes are logged	TRG_BOOKING_AUDIT

9.2 Payment Rules

Rule	Implementation
One payment per booking	SP_PROCESS_PAYMENT validation
Cannot pay for cancelled booking	SP_PROCESS_PAYMENT validation
Completed payment confirms booking	TRG_AUTO_CONFIRM_ON_PAYMENT
Amount calculated from booking items	SP_CALCULATE_BOOKING_TOTAL

9.3 Inventory Rules

Rule	Implementation
Cannot overbook hotel rooms	TRG_PREVENT_DOUBLE_BOOKING
Room availability is real-time	VW_HOTEL_AVAILABILITY

9.4 Data Integrity Rules

Rule	Implementation
Emails stored in lowercase	TRG_CUSTOMER_EMAIL_LOWERCASE

Booking changes are audited	TRG_BOOKING_AUDIT
Cascade deletes for bookings	FK ON DELETE CASCADE
Catalog deletion preserves bookings	FK ON DELETE SET NULL

Appendix A: Error Code Reference

Code	Location	Description
-20001	SP_CREATE_BOOKING_FROM_CATALOG	Customer not found
-20002	SP_CREATE_BOOKING_FROM_CATALOG	Catalog not found
-20003	SP_CREATE_BOOKING_FROM_CATALOG	Check-out must be after check-in
-20004	SP_CREATE_BOOKING_FROM_CATALOG	Check-in date in past
-20005	SP_CALCULATE_BOOKING_TOTAL, SP_PROCESS_PAYMENT, SP_CANCEL_BOOKING	Booking not found
-20006	SP_PROCESS_PAYMENT	Cannot pay for cancelled booking
-20007	SP_PROCESS_PAYMENT	Payment already exists
-20008	SP_PROCESS_PAYMENT	Invalid payment method
-20009	SP_PROCESS_PAYMENT	No items to pay for
-20010	SP_COMPLETE_PAYMENT	Payment not found
-20011	SP_COMPLETE_PAYMENT	Payment not pending
-20012	SP_CANCEL_BOOKING	Already cancelled
-20030	TRG_PREVENT_DOUBLE_BOOKING	Insufficient rooms

Appendix B: Quick Reference

Views

View	One-Line Description
VW_CATALOG_FULL_DETAILS	Catalog with calculated costs
VW_BOOKING_DETAILS	Full booking info with totals
VW_CUSTOMER_BOOKING_SUMMARY	Customer stats and tier
VW_REVENUE_REPORT	Monthly revenue analytics
VW_HOTEL_AVAILABILITY	Real-time room availability

Stored Procedures

Procedure	One-Line Description
SP_CREATE_BOOKING_FROM_CATALOG	Create booking from package
SP_CALCULATE_BOOKING_TOTAL	Calculate cost breakdown
SP_PROCESS_PAYMENT	Create payment for booking
SP_COMPLETE_PAYMENT	Complete payment & confirm booking
SP_CANCEL_BOOKING	Cancel booking & handle refund

Triggers

Trigger	One-Line Description
TRG_BOOKING_AUDIT	Log all booking changes
TRG_AUTO_CONFIRM_ON_PAYMENT	Auto-confirm on payment completion
TRG_PREVENT_DOUBLE_BOOKING	Prevent hotel overbooking
TRG_CUSTOMER_EMAIL_LOWERCASE	Normalize email to lowercase
TRG_CASCADE_BOOKING_CANCEL	Update payments on cancel

End of Documentation