Introduction

This assignment involves designing and implementing a car rental database system using Oracle Database Server. The system efficiently manages core operations such as customer registrations, vehicle inventory, reservations, rentals, and payments while ensuring data integrity and security. Structured tables with appropriate constraints, sequences for ID generation, and partitioning for large datasets optimize performance and maintainability. Business logic is enforced through triggers for automated calculations (e.g., rental charges, late fees) and audit trails to track critical changes.

Security is a key focus, with role-based access control defining permissions for rental agents, managers, and accountants. Advanced features like Virtual Private Database (VPD) restrict data access by branch, while Transparent Data Encryption (TDE) safeguards sensitive information. The system also supports reporting through optimized queries and materialized views, ensuring smooth business analysis. Concurrency techniques (optimistic/pessimistic locking) maintain data consistency in multi-user environments, making this a robust solution for car rental management.

1. Background to Our Project

A. The Technology (Database)

The project utilizes a relational database management system (RDBMS) to create a structured and efficient database for the car rental service. The chosen technology allows for the organization of data into tables, enabling relationships between different entities such as customers, vehicles, reservations, and payments. The SQL language is employed for data manipulation and retrieval, ensuring that the system can handle complex queries and transactions. Features such as triggers, views, and stored procedures enhance the functionality and performance of the database, making it a powerful tool for managing the car rental operations.

B. The Organization

The organization is a car rental service that provides a diverse fleet of vehicles for customers, ranging from economy cars to luxury vehicles. The company aims to offer exceptional customer service and a seamless rental experience. With multiple branches in various locations, the organization seeks to optimize its operations through a centralized database system that can manage customer data, vehicle inventory, reservations, and financial transactions effectively.

C. The Purpose of the Project

The primary purpose of this project is to develop a comprehensive database system that supports the car rental service's operational needs. The system will facilitate efficient management of customer information, vehicle availability, reservation processes, and payment transactions. By implementing this database, the organization aims to improve operational efficiency, enhance customer satisfaction, and provide valuable insights through data analytics.

2. Problem Statement

The Reason for Developing This Database Project is the car rental industry faces several challenges, including managing customer data, tracking vehicle availability, and processing reservations and payments efficiently. Without a centralized database, these processes can become cumbersome, leading to errors, delays, and decreased customer satisfaction. The need for a robust database system arises from the necessity to streamline operations, reduce manual errors, and provide a better overall experience for customers. This project aims to address these challenges by creating a structured database that enhances data management and operational efficiency.

3. Objectives of the Project

The general objective of the project is to create a comprehensive database system for the car rental service that enhances operational efficiency and customer experience. Specific objectives include:

- Designing a structured database schema with tables for customers, vehicles, reservations, rentals, and payments.
- Implementing data integrity constraints to ensure accurate and reliable data.
- Developing automated processes using triggers for calculating rental charges and auditing changes.

- Creating user-friendly views for easy access to critical data and reporting.
- Ensuring data security and compliance with industry standards.

4. Methodology

A. Type of Data Model Used

The project employs a relational data model, which organizes data into tables (relations) that can be linked through foreign keys. This model allows for efficient data retrieval and manipulation, ensuring that relationships between different entities (e.g., customers and their reservations) are maintained. The use of primary keys ensures that each record is unique, while foreign keys establish connections between related tables.

B. The Type of Tools Are Used (HW, SW):

The following tools and technologies are utilized in this project:

- **Hardware**: A server or cloud-based infrastructure to host the database, ensuring high availability and performance.
- **Software**:
  - **Database Management System (DBMS)**: We Used a RDBMS such as Oracle to create and manage the database.
  - **SQL**: Structured Query Language for data definition, manipulation, and retrieval.
  - **Development Tools**: Integrated Development Environments (IDEs) such as Oracle SQL Developer for database design and management.
  - **Backup and Recovery Tools**: RMAN or Data Pump for data backup and recovery processes.
  - **Security Tools**: Encryption and auditing features provided by the DBMS to protect sensitive data.

This structured approach ensures that the car rental database system is efficient, secure, and capable of meeting the operational needs of the organization.

Chapter Two: Database Designing

1. Business Process

The business process for the car rental service involves several key activities that are essential for smooth operations. These processes include:

- **Customer Management:** Collecting and maintaining customer information, including personal details, contact information, and membership status.
- **Vehicle Management**: Tracking the availability, status, and details of the vehicle fleet, including make, model, year, type, and daily rental rates.
- **Reservation Process:** Allowing customers to reserve vehicles for specific dates, managing the status of reservations (e.g., pending, confirmed, cancelled), and ensuring that vehicles are available for the requested time.
- **Rental Transactions**: Managing the actual rental process, including recording pickup and return times, calculating total charges, and applying late fees if necessary.
- **Payment Processing:** Handling payments for rentals, including different payment methods and recording payment details.
- **Reporting and Analytics**: Generating reports on rentals, revenue, customer behavior, and vehicle utilization to support decision-making and improve business operations.

2. Building Blocks of Database

The basic building blocks of the database for the car rental system include:

- **Entities**: These are the main objects or concepts represented in the database. In this case, the primary entities are:
  - **Customer**: Represents individuals renting vehicles.
  - **Vehicle**: Represents the cars available for rent.
  - **Reservation**: Represents the booking of a vehicle by a customer.
  - **Rental**: Represents the actual transaction of renting a vehicle.
  - **Payment**: Represents the financial transactions related to rentals.
  - **Employee**: Represents staff members managing the rental process.

- **Attributes**: These are the properties or characteristics of each entity. For example:
  - **Customer**: cust_id, name, license_num, phone, email, membership_status, reg_date.
  - **Vehicle**: vehicle_id, make, model, year, type, daily_rate, status, location, last_service.
  - **Reservation**: resv_id, cust_id, vehicle_id, pickup_date, return_date, status, created_at.
  - **Rental**: rental_id, resv_id, emp_id, actual_pickup, actual_return, total_charge, late_fee.
  - **Payment**: payment_id, rental_id, amount, payment_date, method.

- **Relationships**: These define how entities are related to one another. For example:
  - A **Customer** can have multiple **Reservations**.
  - A **Reservation** is associated with one **Vehicle**.
  - A **Rental** is linked to a **Reservation** and an **Employee**.
  - A **Payment** is associated with a **Rental**.

3. Draw ER/Class Diagram of Your Database

An Entity-Relationship (ER) diagram visually represents the entities, their attributes, and the relationships between them. Below is a textual representation of how the ER diagram would look:

5. Map ER Diagram into Relational Data Model

Mapping the ER diagram into a relational data model involves translating the entities, their attributes, and the relationships defined in the ER diagram into tables, columns, and constraints in a relational database. Below is a detailed mapping for the car rental database:

To clarify, the mapping process typically includes the following steps:

1. **Identify Entities**: Each entity in the ER diagram corresponds to a table in the relational model.
2. **Define Attributes**: Each attribute of the entity becomes a column in the table.
3. **Establish Relationships**: Relationships between entities are represented through foreign keys in the tables.
4. **Define Constraints**: Primary keys, foreign keys, unique constraints, and check constraints are established to maintain data integrity.

Relational Data Model Mapping

1. **Customer Table**
   - Table Name: customer
     - Attributes:
     - **cust_id** (NUMBER, PRIMARY KEY): Unique identifier for each customer.
     - **name** (VARCHAR2(100), NOT NULL): Name of the customer.
     - **license_num** (VARCHAR2(20), UNIQUE, NOT NULL): Unique driver's license number for the customer.
     - **phone** (VARCHAR2(15)): Contact phone number of the customer.
     - **email** (VARCHAR2(100)): Email address of the customer.
     - **membership_status** (VARCHAR2(20), CHECK): Membership status (e.g., REGULAR, GOLD, PLATINUM).
     - **reg_date** (DATE, DEFAULT SYSDATE): Registration date of the customer.

2. **Vehicle Table**
   - Table Name: vehicle
     - Attributes:
       - **vehicle_id** (NUMBER, PRIMARY KEY): Unique identifier for each vehicle.
       - **make** (VARCHAR2(50), NOT NULL): Manufacturer of the vehicle.
       - **model** (VARCHAR2(50), NOT NULL): Model of the vehicle.
       - **year** (NUMBER(4), NOT NULL): Year of manufacture.
       - **type** (VARCHAR2(30), CHECK): Type of vehicle (e.g., ECONOMY, SUV).
       - **daily_rate** (NUMBER(8,2), NOT NULL): Daily rental rate for the vehicle.
       - **status** (VARCHAR2(20), CHECK): Current status of the vehicle (e.g., AVAILABLE, RENTED).
       - **location** (VARCHAR2(50), NOT NULL): Location where the vehicle is available.
       - **last_service** (DATE): Date of the last service performed on the vehicle.

3. Reservation Table
   - Table Name: reservation
     - Attributes:
       - **resv_id** (NUMBER, PRIMARY KEY): Unique identifier for each reservation.
       - **cust_id** (NUMBER, FOREIGN KEY): References customer(cust_id), linking the reservation to a customer.

- **vehicle_id** (NUMBER, FOREIGN KEY): References vehicle(vehicle_id), linking the reservation to a vehicle.
- **pickup_date** (DATE, NOT NULL): Date and time when the vehicle is to be picked up.
- **return_date** (DATE, NOT NULL): Date and time when the vehicle is to be returned.
- **status** (VARCHAR2(20), CHECK): Status of the reservation (e.g., PENDING, CONFIRMED).
- **created_at** (TIMESTAMP, DEFAULT SYSTIMESTAMP): Timestamp when the reservation was created.

4. **Rental Table**
   - Table Name: rental
     - Attributes:
     - **rental_id** (NUMBER, PRIMARY KEY): Unique identifier for each rental transaction.
     - **resv_id** (NUMBER, UNIQUE, FOREIGN KEY): References reservation(resv_id), linking the rental to a reservation.
     - **emp_id** (NUMBER, FOREIGN KEY): References employee(emp_id), linking the rental to an employee who processed it.
     - **actual_pickup** (TIMESTAMP, NOT NULL): Actual date and time when the vehicle was picked up.
     - **actual_return** (TIMESTAMP): Actual date and time when the vehicle was returned.
     - **total_charge** (NUMBER(10,2)): Total charge for the rental.
     - **late_fee** (NUMBER(8,2), DEFAULT 0): Late fee applied if the vehicle is returned after the due date.

5. **Payment Table**
   - Table Name: payment
     - Attributes:
       - **payment_id** (NUMBER, PRIMARY KEY): Unique identifier for each payment transaction.
       - **rental_id** (NUMBER, FOREIGN KEY): References rental(rental_id), linking the payment to a rental.
       - **amount** (NUMBER(10,2), NOT NULL): Amount paid for the rental.
       - **payment_date** (TIMESTAMP, DEFAULT SYSTIMESTAMP): Date and time when the payment was made.
       - **method** (VARCHAR2(20), CHECK): Payment method used (e.g., CASH, CREDIT).

6. Employee Table
   - Table Name: employee
     - Attributes:
       - **emp_id** (NUMBER, PRIMARY KEY): Unique identifier for each employee.
       - **name** (VARCHAR2(100), NOT NULL): Name of the employee.
       - **position** (VARCHAR2(50), NOT NULL): position of the employee.
       - **branch** (VARCHAR2(50), NOT NULL): Branch location where the employee works.
       - **login_id** (VARCHAR2(30), UNIQUE, NOT NULL): Unique login identifier for the employee.

Summary of the Relational Data Model

The relational data model for the car rental database consists of the following tables, each representing a key entity in the business process:

1. **Customer Table**: Stores customer information, including personal details and membership status.
2. **Vehicle Table**: Contains details about the vehicles available for rent, including their status and rental rates.
3. **Reservation Table:** Manages reservations made by customers, linking them to specific vehicles and tracking their status.
4. **Rental Table**: Records the actual rental transactions, including pickup and return times, total charges, and late fees.
5. **Payment Table**: Handles payment transactions related to rentals, including the amount paid and payment method.
6. **Employee Table**: Maintains information about employees who manage the rental process.

Relationships

- **Customer to Reservation**: One-to-Many (A customer can have multiple reservations).
- **Vehicle to Reservation**: One-to-Many (A vehicle can be reserved multiple times).
- **Reservation to Rental**: One-to-One (Each reservation corresponds to one rental).
- **Rental to Payment**: One-to-One (Each rental can have one associated payment).
- **Employee to Rental**: One-to-Many (An employee can process multiple rentals).

Constraints

- **Primary Keys (PK)**: Ensure that each record in a table is unique.
- **Foreign Keys (FK)**: Establish relationships between tables, ensuring referential integrity.
- **Unique Constraints**: Ensure that certain fields, like license_num and login_id, are unique across the database.
- **Check Constraints**: Validate that certain fields contain acceptable values (e.g., membership status, vehicle type, and payment method).

Conclusion

The mapping of the ER diagram into a relational data model provides a clear structure for the car rental database. This design ensures that all necessary data is captured, relationships are maintained, and data integrity is enforced. The relational model allows for efficient data retrieval and manipulation, supporting the operational needs of the car rental service. This structured approach will facilitate better management of customer interactions, vehicle availability, reservations, and financial transactions, ultimately enhancing the overall efficiency of the business.

Chapter Three: Database Development

1. Using Oracle Database Server to Develop Our Database System

A. DDL (Data Definition Language)

DDL is used to define and manage all database objects. Here are the SQL statements for creating the tables in the car rental database:

```
CREATE TABLE customer (
    cust_id NUMBER PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    license_num VARCHAR2(20) UNIQUE NOT NULL,
    phone VARCHAR2(15),
    email VARCHAR2(100),
    membership_status VARCHAR2(20) CHECK (membership_status IN
('REGULAR', 'GOLD', 'PLATINUM')),
    reg_date DATE DEFAULT SYSDATE
);

CREATE TABLE vehicle (
    vehicle_id NUMBER PRIMARY KEY,
    make VARCHAR2(50) NOT NULL,
    model VARCHAR2(50) NOT NULL,
    year NUMBER(4) NOT NULL,
    type VARCHAR2(30) CHECK (type IN ('ECONOMY', 'COMPACT',
'MID_SIZE', 'FULL_SIZE', 'SUV', 'LUXURY')),
    daily_rate NUMBER(8,2) NOT NULL,
    status VARCHAR2(20) CHECK (status IN ('AVAILABLE', 'RENTED',
'MAINTENANCE', 'RETIRED')),
    location VARCHAR2(50) NOT NULL,
    last_service DATE
);

CREATE TABLE reservation (
    resv_id NUMBER PRIMARY KEY,
    cust_id NUMBER REFERENCES customer(cust_id),
    vehicle_id NUMBER REFERENCES vehicle(vehicle_id),
    pickup_date DATE NOT NULL,
    return_date DATE NOT NULL,
    status VARCHAR2(20) CHECK (status IN ('PENDING', 'CONFIRMED',
'CANCELLED', 'COMPLETED')),
    created_at TIMESTAMP DEFAULT SYSTIMESTAMP,
    CONSTRAINT chk_dates CHECK (return_date > pickup_date)
);

CREATE TABLE employee (
    emp_id NUMBER PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    position VARCHAR2(50) NOT NULL,
    branch VARCHAR2(50) NOT NULL,
    login_id VARCHAR2(30) UNIQUE NOT NULL
);
```

```
CREATE TABLE rental (
    rental_id NUMBER PRIMARY KEY,
    resv_id NUMBER UNIQUE REFERENCES reservation(resv_id),
    emp_id NUMBER REFERENCES employee(emp_id),
    actual_pickup TIMESTAMP NOT NULL,
    actual_return TIMESTAMP,
    total_charge NUMBER(10,2),
    late_fee NUMBER(8,2) DEFAULT 0,
    CONSTRAINT chk_actual_dates CHECK (actual_return IS NULL OR
actual_return > actual_pickup)
)
PARTITION BY RANGE (actual_pickup) (
    PARTITION rentals_2023 VALUES LESS THAN (TO_DATE('2024-01-01',
'YYYY-MM-DD')),
    PARTITION rentals_2024 VALUES LESS THAN (TO_DATE('2025-01-01',
'YYYY-MM-DD')),
    PARTITION rentals_future VALUES LESS THAN (MAXVALUE)
);

CREATE TABLE payment (
    payment_id NUMBER PRIMARY KEY,
    rental_id NUMBER REFERENCES rental(rental_id),
    amount NUMBER(10,2) NOT NULL,
    payment_date TIMESTAMP DEFAULT SYSTIMESTAMP,
    method VARCHAR2(20) CHECK (method IN ('CASH', 'CREDIT', 'DEBIT',
'ONLINE'))
);
```

B. DML (Data Manipulation Language)

DML is used to manipulate data within the database. Here are examples of DML statements for inserting, updating, and deleting records:

```
-- Insert a new customer
INSERT INTO customer VALUES (seq_cust_id.NEXTVAL, 'John Smith',
'DL123456', '555-0101', 'john.smith@email.com', 'GOLD', SYSDATE);

-- Update a customer's phone number
UPDATE customer SET phone = '555-0102' WHERE cust_id = 1000;

-- Delete a customer
DELETE FROM customer WHERE cust_id = 1000;
```

C. DCL (Data Control Language)

DCL is used to control access to data within the database. Here are examples of granting and revoking permissions:

```
-- Create Roles
CREATE ROLE rental_agent;
CREATE ROLE manager;
CREATE ROLE accountant;

-- Grant Privileges
GRANT SELECT, INSERT, UPDATE ON customer TO rental_agent;
GRANT SELECT, INSERT, UPDATE ON reservation TO rental_agent;
GRANT SELECT, INSERT ON rental TO rental_agent;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES TO manager;
GRANT EXECUTE ON ALL PROCEDURES TO manager;

GRANT SELECT ON rental TO accountant;
GRANT SELECT, INSERT, UPDATE ON payment TO accountant;

-- Create Users
CREATE USER agent1 IDENTIFIED BY agentpass1;
CREATE USER mgr1 IDENTIFIED BY mgrpass1;

-- Assign Roles
GRANT rental_agent TO agent1;
GRANT manager TO mgr1;
```

C. TCL (Transaction Control Language)

TCL is used to manage transactions in the database. Here are examples of TCL commands:

```
-- Transaction Example
BEGIN
    -- Start transaction
    SAVEPOINT start_transaction;

    -- Update vehicle status
    UPDATE vehicle SET status = 'RENTED' WHERE vehicle_id = 5000;

    -- Create rental record
    INSERT INTO rental VALUES (seq_rental_id.NEXTVAL, 10000, 101,
        SYSTIMESTAMP, NULL, NULL, 0);

    -- If everything succeeds
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK TO start_transaction;
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

E. Triggers

Triggers are special procedures that automatically execute in response to certain events on a table. Here's an example of a trigger that calculates rental charges:

```
  -- Get daily rate
    SELECT daily_rate INTO v_daily_rate
    FROM vehicle v
    JOIN reservation r ON v.vehicle_id = r.vehicle_id
    WHERE r.resv_id = :NEW.resv_id;

    -- Calculate days rented
    v_days := CEIL(:NEW.actual_return - :NEW.actual_pickup);

    -- Calculate late days if any
    SELECT CEIL(:NEW.actual_return - r.return_date) INTO v_late_days
    FROM reservation r
    WHERE r.resv_id = :NEW.resv_id;

    -- Set charges
    :NEW.total_charge := v_daily_rate * v_days;

    -- Apply late fee if applicable
    IF v_late_days > 0 THEN
        :NEW.late_fee := v_late_days * (v_daily_rate * 0.2); -- 20%
of daily rate as late fee
    END IF;
END;
```

F. View

A view is a virtual table based on the result set of a SQL query. Here are examples of creating views for available vehicles and rental details:

```
-- View for available vehicles
CREATE OR REPLACE VIEW vw_available_vehicles AS
SELECT vehicle_id, make, model, year, type, daily_rate, location
FROM vehicle WHERE status = 'AVAILABLE'
ORDER BY make, model;

-- View for rental details
CREATE OR REPLACE VIEW vw_rental_details AS
SELECT r.rental_id, c.name AS customer_name,
       v.make || ' ' || v.model AS vehicle,
       r.actual_pickup, r.actual_return,
       r.total_charge, r.late_fee,
       p.amount AS payment_received, p.payment_date
FROM rental r
JOIN reservation res ON r.resv_id = res.resv_id
JOIN customer c ON res.cust_id = c.cust_id
JOIN vehicle v ON res.vehicle_id = v.vehicle_id
LEFT JOIN payment p ON r.rental_id = p.rental_id;
```

G. Join

Joins are used to combine rows from two or more tables based on a related column. Here are examples of different types of joins:

```
-- Inner Join: Current rentals with customer and vehicle info
SELECT r.rental_id, c.name, v.make, v.model,
       r.actual_pickup, r.actual_return
FROM rental r
INNER JOIN reservation res ON r.resv_id = res.resv_id
INNER JOIN customer c ON res.cust_id = c.cust_id
INNER JOIN vehicle v ON res.vehicle_id = v.vehicle_id
WHERE r.actual_return IS NULL;

-- Left Outer Join: All vehicles with their current rental status
SELECT v.vehicle_id, v.make, v.model, v.status,
       r.rental_id, r.actual_pickup, r.actual_return
FROM vehicle v
LEFT JOIN reservation res ON v.vehicle_id = res.vehicle_id
    AND res.status = 'CONFIRMED'
    AND SYSDATE BETWEEN res.pickup_date AND res.return_date
LEFT JOIN rental r ON res.resv_id = r.resv_id;

-- Self Join: Find customers with multiple rentals
SELECT c1.cust_id, c1.name, COUNT(r.rental_id) AS rental_count
FROM customer c1
JOIN reservation res ON c1.cust_id = res.cust_id
JOIN rental r ON res.resv_id = r.resv_id
GROUP BY c1.cust_id, c1.name
HAVING COUNT(r.rental_id) > 1;
```

H. OQL (Object Query Language)

In Oracle, OQL can be represented through SQL queries that utilize object types. Here's an example of querying using object attributes:

```
-- Create object types
CREATE TYPE address_type AS OBJECT (
    street VARCHAR2(100),
    city VARCHAR2(50),
    state VARCHAR2(30),
    zip VARCHAR2(10)
);

CREATE TYPE phone_type AS OBJECT (
    home_phone VARCHAR2(15),
    work_phone VARCHAR2(15),
    mobile_phone VARCHAR2(15)
);
```

```
-- Modify customer table to use objects
ALTER TABLE customer ADD (
    customer_address address_type,
    customer_phones phone_type
);

-- Query using object attributes
SELECT c.cust_id, c.name,
       c.customer_address.street AS street,
       c.customer_address.city || ', ' || c.customer_address.state AS
location
FROM customer c
WHERE c.customer_address.state = 'CA';

-- Update object attributes
UPDATE customer
SET customer_address = address_type('123 Main St', 'Los Angeles',
'CA', '90001')
WHERE cust_id = 1000;
```

I. Aggregate Functions

Aggregate functions perform calculations on a set of values and return a single value. Here are examples of using aggregate functions in the context of the car rental database:

```
-- Monthly revenue by vehicle type
SELECT
    EXTRACT(YEAR FROM r.actual_pickup) AS year,
    EXTRACT(MONTH FROM r.actual_pickup) AS month,
    v.type AS vehicle_type,
    COUNT(*) AS rentals,
    SUM(r.total_charge + r.late_fee) AS total_revenue,
    AVG(r.total_charge) AS avg_rental_amount
FROM rental r
JOIN reservation res ON r.resv_id = res.resv_id
JOIN vehicle v ON res.vehicle_id = v.vehicle_id
GROUP BY EXTRACT(YEAR FROM r.actual_pickup),
         EXTRACT(MONTH FROM r.actual_pickup),
         v.type
ORDER BY year, month;

-- Customer loyalty analysis
SELECT
    c.membership_status,
    COUNT(DISTINCT c.cust_id) AS customer_count,
    COUNT(r.rental_id) AS total_rentals,
    SUM(r.total_charge) AS total_spent,
    AVG(r.total_charge) AS avg_spent_per_rental
FROM customer c
LEFT JOIN reservation res ON c.cust_id = res.cust_id
LEFT JOIN rental r ON res.resv_id = r.resv_id
GROUP BY c.membership_status
ORDER BY total_spent DESC;
```

```
-- Vehicle utilization
SELECT
    v.make,
    v.model,
    COUNT(r.rental_id) AS rental_count,
    SUM(r.actual_return - r.actual_pickup) AS total_days_rented,
    ROUND(SUM(r.actual_return - r.actual_pickup) / 365 * 100, 2) AS
utilization_percentage
FROM vehicle v
LEFT JOIN reservation res ON v.vehicle_id = res.vehicle_id
LEFT JOIN rental r ON res.resv_id = r.resv_id
WHERE r.actual_return IS NOT NULL
GROUP BY v.make, v.model
ORDER BY utilization_percentage DESC;
```

J. Backup and Recovery

Backup and recovery are critical for data protection. In Oracle, you can use RMAN (Recovery Manager) for backup and recovery. Here's a simple command to back up the database:

```
-- RMAN Backup Script
RUN {
    ALLOCATE CHANNEL ch1 DEVICE TYPE DISK FORMAT
'/backups/car_rental_%U.bkp';
    BACKUP DATABASE PLUS ARCHIVELOG;
    BACKUP CURRENT CONTROLFILE;
    RELEASE CHANNEL ch1;
}

-- Data Pump Export
expdp car_rental/securepass123 DIRECTORY=DATA_PUMP_DIR
DUMPFILE=car_rental_export.dmp LOGFILE=export.log SCHEMAS=car_rental

-- Flashback Query Example
-- Recover accidentally deleted customer
INSERT INTO customer
SELECT * FROM customer AS OF TIMESTAMP TO_TIMESTAMP('2023-12-01
14:00:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE cust_id = 1005;

-- Flashback Table
FLASHBACK TABLE rental TO TIMESTAMP TO_TIMESTAMP('2023-12-01
14:00:00', 'YYYY-MM-DD HH24:MI:SS');
```

K. Database Security and Authorization

Database security involves protecting the database from unauthorized access. This can be achieved through user roles and privileges. Here's an example of creating a role and granting it privileges:

```
-- Transparent Data Encryption (TDE)
ALTER TABLE customer MODIFY (license_num ENCRYPT);
ALTER TABLE payment MODIFY (amount ENCRYPT, method ENCRYPT);
```

```
-- Audit sensitive operations
AUDIT SELECT, INSERT, UPDATE, DELETE ON customer BY ACCESS;
AUDIT SELECT, INSERT, UPDATE, DELETE ON payment BY ACCESS;

-- Fine-Grained Auditing
BEGIN
    DBMS_FGA.ADD_POLICY(
        object_schema => 'car_rental',
        object_name => 'customer',
        policy_name => 'audit_sensitive_customer_access',
        audit_condition => 'membership_status = ''PLATINUM''',
        audit_column => 'email, phone',
        handler_schema => NULL,
        handler_module => NULL,
        enable => TRUE
    );
END;

-- Password policies
CREATE PROFILE rental_profile LIMIT
    FAILED_LOGIN_ATTEMPTS 5
    PASSWORD_LIFE_TIME 90
    PASSWORD_REUSE_TIME 365
    PASSWORD_REUSE_MAX 5
    PASSWORD_LOCK_TIME 1;

ALTER USER car_rental PROFILE rental_profile;
```

l. Write a Query That Is Used for Report Purpose (Optimal Query)

This query provides a comprehensive overview of rental activity over the past year, broken down by month and branch, including metrics such as rental count, total revenue, and revenue per day. Here's an example of an optimal query to generate a report of all active rentals along with customer and vehicle details:

```
-- Executive Dashboard Query
WITH monthly_data AS (
    SELECT
        TRUNC(r.actual_pickup, 'MM') AS month,
        e.branch,
        v.type,
        COUNT(*) AS rental_count,
        SUM(r.total_charge + r.late_fee) AS revenue,
        SUM(r.actual_return - r.actual_pickup) AS rental_days
    FROM rental r
    JOIN reservation res ON r.resv_id = res.resv_id
    JOIN vehicle v ON res.vehicle_id = v.vehicle_id
    JOIN employee e ON r.emp_id = e.emp_id
    WHERE r.actual_return IS NOT NULL
    GROUP BY TRUNC(r.actual_pickup, 'MM'), e.branch, v.type
),
branch_totals AS (
    SELECT
```

```
   month,
        branch,
        SUM(rental_count) AS total_rentals,
        SUM(revenue) AS total_revenue,
        SUM(rental_days) AS total_days
    FROM monthly_data
    GROUP BY month, branch
)
SELECT
    TO_CHAR(d.month, 'YYYY-MM') AS month,
    d.branch,
    d.type,
    d.rental_count,
    d.revenue,
  ROUND(d.revenue / NULLIF(d.rental_days, 0), 2) AS revenue_per_day,
  ROUND(d.revenue * 100.0 / NULLIF(t.total_revenue, 0), 2) AS
percent_of_branch_revenue
  FROM monthly_data d
  JOIN branch_totals t ON d.month = t.month AND d.branch = t.branch
  WHERE d.month >= ADD_MONTHS(TRUNC(SYSDATE, 'MM'), -12)
  ORDER BY d.month, d.branch, d.revenue DESC;
```

M. Concurrency Techniques

Concurrency control is essential for managing simultaneous operations without conflicts. In Oracle, you can use various techniques to handle concurrency, including optimistic and pessimistic locking. Here are examples of both:

1. **Optimistic Concurrency Control**: This approach assumes that multiple transactions can complete without affecting each other. It checks for conflicts before committing.

```
CREATE OR REPLACE PROCEDURE update_customer_info(
    p_cust_id IN NUMBER,
    p_name IN VARCHAR2,
    p_phone IN VARCHAR2,
    p_email IN VARCHAR2,
    p_original_reg_date IN DATE
) AS
    v_updated_count NUMBER;
BEGIN
    UPDATE customer
    SET name = p_name,
        phone = p_phone,
        email = p_email
    WHERE cust_id = p_cust_id
    AND reg_date = p_original_reg_date;

    v_updated_count := SQL%ROWCOUNT;

    IF v_updated_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Customer record was modified
by another user. Please refresh and try again.');
```

```
  END IF;

    COMMIT;
END;
```

2. **Pessimistic Locking:** This approach locks the data when it is read, preventing other transactions from modifying it until the lock is released.

```
DECLARE
    CURSOR c_rentals IS
    SELECT * FROM rental
    WHERE actual_return IS NULL
    FOR UPDATE NOWAIT;
BEGIN
    FOR r_rec IN c_rentals LOOP
        -- Process each rental record
        UPDATE rental SET late_fee = late_fee + 10
        WHERE CURRENT OF c_rentals;
    END LOOP;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

3. **Serializable Transaction**: This ensures that transactions are executed in a way that they appear to be executed one after the other, preventing any concurrent modifications.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN
    -- Check vehicle availability
    SELECT COUNT(*) INTO v_count
    FROM vehicle
    WHERE vehicle_id = p_vehicle_id
    AND status = 'AVAILABLE'
    FOR UPDATE;

    IF v_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Vehicle not available');
    END IF;

    -- Create reservation
    INSERT INTO reservation VALUES (...);

    -- Update vehicle status
    UPDATE vehicle SET status = 'RESERVED'
    WHERE vehicle_id = p_vehicle_id;
```

```
   COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        -- Handle error
END;
```

4. **Versioning with ORA_ROWSCN**: This allows tracking changes to rows and can be used to implement optimistic concurrency control.

```
SELECT cust_id, name,
       ORA_ROWSCN AS row_version
FROM customer
WHERE cust_id = 1005;
```

Conclusion

In Chapter Three, we explored the comprehensive development of a car rental database system using Oracle Database Server. We began by defining the database structure through Data Definition Language (DDL), creating essential tables for customers, vehicles, reservations, rentals, payments, and employees, while ensuring data integrity with appropriate constraints and relationships. We then delved into Data Manipulation Language (DML) for managing data, alongside Data Control Language (DCL) to establish user permissions and access control.

The chapter also highlighted the importance of transaction management using Transaction Control Language (TCL), the automation of processes through triggers, and the simplification of complex queries with views. We examined the use of joins to combine data from multiple tables, the application of aggregate functions for insightful reporting, and the implementation of backup and recovery strategies to safeguard data. Additionally, we discussed security measures, including encryption and auditing, to protect sensitive information. Overall, this chapter provided a solid foundation for building a robust, secure, and efficient database system that meets the operational needs of a car rental business while ensuring data integrity and compliance with regulations.