IS 420: Database Application Development
Fall 2023

Group Project
Mobile Parking Payment System

Overview

You will be assigned into groups of four to five people in this project. Please read the whole document carefully before starting your project.

Your assignment is to design Mobile Parking Payment system (like ParkMobile https://parkmobile.io/). You will design the database, insert some sample data, and implement a set of required features in the backend. Each feature will be implemented as one or more Oracle PL/SQL procedures/functions. You do **NOT** need to write a graphic user interface. You can test your features using SQL or PL/SQL scripts.

Assumptions:

You can make the following assumptions in this project.

1. The system will store information about customer, including customer ID, name, address, zipcode, state, email, phone number, payment card number.

2. The system store information about parking zones. Each zone has a zone ID, address, zipcode, capacity (number of parking spots), number of spots currently available, hourly rate, maximal length of parking, the effective period of the parking rate (outside the period parking will be free).

The effective period can be represented using 4 columns: start day of the week (1 means Sunday, 2 means Monday,..., 6 means Saturday), start time (e.g., 7 am), end day of the week, end time (e.g., 7 pm).

E.g., a parking zone may have an hourly rate of $2 per hour from Monday to Saturday from 7 am to 7 pm, and with a maximal parking length of 3 hours (someone has to move parked car to another zone if that person wants to park longer than 3 hours).

This can be represented as hourly_rate = 2.00, start_day = 2 (start on Monday), start_time = interval '0 07:00:00.00' day to second (start on 7 am, which can be represented as an interval counting from 12 am), end_day = 7 (end on Saturday), end_time = interval '0 19:00:00.00' day to second (ends 7 pm each day), max_len = interval '3' hour.

3. Each customer can register severals vehicles. Each vehicle has a vid, license plate number (as a string), state, maker, model, year, and color.

4. The system stores information about parking sessions. Each parking session has a session ID, associated customer ID, associated vehicle ID, zone ID, start time of the session, estimated end time of the session, and total charge of the parking session.

5. The system will store payment transactions about a parking session. Each payment transaction has an associated payment ID, a session ID, a payment time, payment amount, and number of hours the payment covers.

Usually when a customer starts a parking session, the customer will enter an initial length of parking session, and pay for the initial length. The payment table will store a row for this initial payment transaction, which includes a fixed fee and hourly rate of the zone multiplied by the initial length if it falls within the effective period of the parking zone.

Subsequently the customer can extend the parking session multiple times, each time by a certain number of hours as long as the total number of hours does not exceed the maximal parking length and the time is still within the effective period of the parking zone. For each extension, the customer needs to pay for the extension and the payment table will have an additional row to store the payment transaction. The payment amount for an extension will be just hourly rate multiplied by extended hours.

6. The system stores a message table, which has a message ID, the associated customer ID, message time, and message body.

Features: There are five individual features and five group features. Each member needs to implement one individual feature. So if your group has X members your group will implement any X individual features. Each group also needs to implement X (X=size of group) group features. For example, if you group has five members, your group will do all features. If you group has four members, your group will implement any four individual features plus any four group features.

Individual features will be graded individually (you group member's individual feature will have no impact on your grade), but group features will be graded group-wise. So each group should work together to make sure the group features are done correctly.

Feature 6 and 7 are more difficult than others.

Member 1:

Feature 1: create a new customer. Input includes customer name, phone number, address, state, zip, email, and credit card number.

This feature checks whether any customer with the same phone number exists. If so, it prints a message 'the user already exists' and updates address, state, zip, email and credit card number. Otherwise it generates a new customer ID (using sequence) and insert a row into customer table with the new customer ID, phone, name, address, state, zip, email and credit card number. Please also print out the new customer ID.

Member 2:

Feature 2: add a vehicle. Input includes license plate number, state of the license, customer ID, maker, model, year and color of the vehicle. The procedure does ALL of the following:

1) it first checks whether the customer ID exists. If not print a message 'Invalid customer ID'.

2) it then checks whether there is a vehicle with the same plate number and state as the input, if so prints a message 'vehicle already exists' and stops.

3) if the vehicle does not exist, insert a row into the vehicle table with a newly generated vehicle ID (using sequence), the input license plate number, state, customer ID, maker, model, year, and color. Print out the new vehicle ID.

Member 3:

Feature 3: search for a parking zone. Input is a zipcode, current time, and intended length of parking.

1) Find all parking zones that satisfy condition 1 and 2 but **not condition 3**: 1) located at the given zipcode, 2) with at least one available spot, 3) the session start time is in effective period of the zone and the requested parking length exceeds the zone's maximal parking length (if condition 3 is true then the parking session will exceed maximal parking length so it is not allowed).
2) For each zone found in step 1, print out zone ID, address, number of available spots, hourly rate, maximal parking length, and effective period (including start day of the week, end day of the week, start time and end time). If there is no matching zone, print out no available zones.

Hint: to check whether the current time is within the effective period of a zone, you can first extract day of the week from the current time by

```
to_char(current_time,'D')
```

where current_time is the input time and has timestamp data type.

Next extract the time of day from the current time, which can be done by

```
current_time - trunc(current_time)
```

where trunc function will return 12 am of the same day as current_time, and the subtraction computes the time of day (i.e., a day to second interval from current time to 12 am same day). E.g., if current_time is 9 am on a given date, the above expression returns interval '9' hour.

So current time is within the effective period if

```
to_char(current_time,'D') between start_day and end_day
and current_time - trunc(current_time) between start_time and end_time
```

Condition 3 can be represented as the above condition and max_len >= input_length
Not satisfying condition 3 can be represented as NOT (condition 3)

Member 4:
Feature 4: list all parking sessions in a period for a customer. Input is customer ID, a start date and an end date. It does the following:

1) This feature first checks whether the customer ID is valid (whether there is a customer with that ID). If not it prints out an error message 'No such customer').
2) If the customer exists, this feature prints out all parking sessions of the customer within the start and end date, and print out each parking session's session id, start and end time, zone id, vehicle ID, and total charge.
3) Finally print out a total charge of all these parking sessions.

Member 5:

Feature 5: list all vehicles with an active session at a parking zone. The input is a parking zone ID and a current time. The feature does the following steps:
1) it first checks whether the zone ID is valid (there is a row in the parking_zone table with the input zone ID). If not, print an error message 'Incorrect zone ID' and stop.
2) it then lists all vehicles with an active parking session in this zone. An active session means that the input current time is between the session's start time and end time. Please print out vehicle Id, customer ID, plate number, state, maker, model, and color.

Group features:

Feature 6: Start a session. Input is customer ID, vehicle ID, zone ID, session start time, and number of hours to park (you can use integer). The feature does the following:

1) it checks whether customer ID, vehicle ID, and zone ID are valid (meaning there is a row in customer with the matching customer ID, a row in vehicle table with the matching vehicle ID, and a row in the parking_zone table with the matching zone ID). If any of them is not valid, print out an error message and stop.

2) it then checks whether condition a) is satisfied and condition b) is NOT satisfied:
a) the zone has at least one available spot,
b) the session start time is in effective period of the zone (refer to feature 3's hint), and the requested parking length exceeds the zone's maximal parking length.
If condition a) is not satisfied, print a message saying that the parking session is not possible due to no available spot and stop. If condition b) is satisfied, print a message saying that the parking length exceeds maximal length and stop.

Hint: you may need to convert an integer number of hours to an interval in checking parking length. You can use function
NUMTODSINTERVAL(v_hours, 'hour')
where v_hours is the number of hours in integer format. The function returns a day to second interval type.

3) if condition a) is satisfied and condition b) is not satisfied, check whether the start time is within effective period. If so, compute a total charge as a fee of $1.00 plus hourly rate * number of hours of intended parking length.
Otherwise (start time not in effective period so parking is free), print a message 'parking is free now!' and stop the procedure.

4) if total charge is greater than zero, insert a row into parking_session table with a newly created session ID, input customer ID, vehicle ID, zone ID, start time, end time as start time + number of hours computed in step 3, total as what is computed in step 3. Insert a message into the message table with the customer id as input customer id and time as session start time and body as 'a new parking session with ID X created' where X is the new session ID. Please also reduce available spots for the parking zone by one.

5) if total charge is greater than zero, insert a row into payment table as payment for the initial session, with a newly generated payment ID, session ID as the session ID created in step 4, time as session start time, amount as total charge computed in step 3, and the input number of hours to park.

Feature 7: extend a session. Input: a parking_session ID, a current_time (timestamp type), the number of hours to extend.
The procedure does the following steps:
1) it checks whether there is a parking_session with the input ID. If not, print a message 'Invalid session ID' and stop.
2) it then checks whether the extended parking session length will exceed the maximal length of the parking zone. If so, print a message: 'Cannot extend the session because maximal length reached' and stop.

Hint: to compute extended length, you can use

```
end_time - start_time +   NUMTODSINTERVAL(v_hours, 'hour')
```

where end_time and start_time are end time and start time of the session, respectively, and v_hours is input number of hours to extend.

3) it then checks whether current_time is before session's end_time. If not print a message 'You can only extend a session before it expires.' and stop.

4) if current time is before the session's end time, update the row in the parking session matching the input session ID as follows:
a) update end_time to
end_time +   NUMTODSINTERVAL(v_hours, 'hour')
b) update total charge by adding hourly_rate * input number of hours to the current total charge.

5) insert a message into message table with the parking session's associated customer ID, the current time, and body as 'Parking session X extended to Y' where X is session ID, Y is new end time.

6) insert a row into payment table with a newly generated payment ID, the input session ID, current_time, amount as hourly_rate * number of extended hours, and the number of extended hours.

Feature 8: stop a session. Input includes session ID and a current time.
The procedure does the following:

1) it first checks whether the session ID is valid (there is a parking session with that ID). If not, print a message 'Invalid session ID' and stop.
2) it then checks whether the current time is after the end time of the session. If so, insert into message table a row with a new message ID, the customer ID for the customer associated with the session, current time, and message body saying 'Session X expired. You may get a ticket.' where X is the session ID.
3) if current time is no later than the end time of the session, update the parking session row's end time to the current time and insert into message table a row with a new message ID, the associated customer ID, current time, and message body
'Session X ends at Y' where X is session ID, Y is time.

Feature 9: create a reminder for all sessions about to expire in 15 minutes. Input includes a current time. It does the following:

1) The procedure goes over every session whose end time is between current time minus 15 minutes to current time, and insert a message to the message table with a newly generated message ID, the customer ID associated with the session, message time as current time, and message body as

'Session X will expire in Y minutes, please extend it if necessary.'.

Here X is session ID. Y is the number of minutes between the current time and end time of the session.

2) The procedure also prints a message saying 'Message generated for session X' where X is session ID.

Hint: the number of minutes between the current time and end time of the session can be computed as

```
 EXTRACT(HOUR FROM x) * 60 + EXTRACT(MINUTE FROM x)
```

where x=end time of session - current time.

Feature 10: Print out statistics. Input is a start date and end date.

This procedure prints out the following statistics:

1) total number of customers

2) total number of vehicles registered by customers

3) total number of parking zones

For steps 4) to 7), a parking session is within the period from the start date to end date if and only if both its start time and end time of the session are both between the input start date and end date.

4) compute for the period from start date to end date, the total number of parking sessions, total revenue (sum of charges in each parking session) in the period.

5) compute average charges per session, average parking length for sessions in the period from state date to end date.

Hint: parking length in number of hours can be computed as extract (hour from end_time-start_time) + extract(minute from (end_time-start_time))/60 where end_time is end time of a session and start_time is start time of a session.

6) for the period from start date to end date, for each parking zone, print out parking zone ID, capacity, number of parking sessions associated with the zone in that period, total revue for the zone (sum of charges of sessions in the zone) in that period.

7) compute occupancy rate for each zone in the input period and print out this rate along with zone ID. Occupancy rate is computed as

sum of parking hours of all sessions in the zone divided by (capacity * 24 * number of days between start date and end date)

Parking hours of a session =

extract (hour from end_time-start_time) + extract(minute from (end_time-start_time))/60

where end_time is end time of a session and start_time is start time of a session.

number of days between start and end date = end_date-start_date

Deliverables:

1. 10%. Due 9/26. Project Management Schedule.

    a. Include team members and a timeline showing each phase of your project with its activities and time duration, for the entire effort.

    b. It is expected that every member should participate in all phases of the project.

    c. Please specify which feature is assigned to which member (for group features it is still possible to assign a lead for each feature).

    d. Activities should include system design, populating tables, writing code, testing code, running example queries, writing documents, preparing for presentation, etc. Smaller milestones shall be set for deliverable 3 and 4.

    e. This deliverable will be graded based on whether items a) to d) are included and whether the schedule is reasonable (e.g., enough time such as 2-3 weeks are left for testing and integration).

2. 25%. Due 10/17. Design Document which includes the following:

a. ER diagram of the database. You don't have to follow exact notations of ER diagram, but need to show tables, columns, primary keys, and foreign key links.

b. SQL statements to create database tables and to insert some sample data (at least 3 rows per table). Please include drop table and drop sequence statements before create table, create sequence and insert.

c. Specification for each required feature. The specification should include a description of input parameters and output (usually printing a message), and a few test cases (normally there should be one normal case and a few special cases). You don't need to implement any of these procedures at this point.

3. 35%. Due 12/6. Presentation of database design and demonstration of all individual features plus two group features. You can finish the remaining group features by D4 deadline (12/19). The project demo will be online (through Webex). You can sign up for the time of presentation/demo through a google form. To demo each feature, you need to prepare a couple of test cases, usually one normal case and the rest as special cases. For each test case you need to be able to explain why your answer is correct (this can be typically done by showing some tables or screen output).

4. 30%. Due 12/19. Please upload final code through blackboard. The code should include:

a. Drop table and sequence statements to drop tables if they exist (remember to use cascade constraints).

b. Create table statements and create sequence statements

c. Insert statements

d. Create procedure statements (with code for the procedures) for all features. Each feature can be implemented as one PL/SQL procedure (in the procedure you may call other procedures or functions). Please include some comments in your code explaining the major steps. You should use create or replace to avoid procedure name conflict.

e. Test script to show that all your features work correctly. The script shall include some examples to test different cases. E.g., for feature 1, one example for new customer added (phone is not in database) and one example for existing customer. Please include:

i. PL/SQL script to call the appropriate PL/SQL procedure for this feature. E.g., exec procedure-name(parameter values)

ii. Explanation of what should be the correct output. The output could be updated tables (you can have some select statement to show the updated tables), some print out, etc.

iii. Make sure you have tested your examples from beginning to end. Remember that database tables may have been changed in the process. So you may need to start with a clean database (i.e., right after you execute all the drop table, create table, and insert statements).

Grading Guidelines

What I look for while grading software code (deliverable 4):

1. Existence of code and whether all code can be compiled without any error.
2. Comments: Both descriptive and inline for every procedure/function
3. Software quality
    a. Whether it is correct (giving correct results).
    b. Whether it is complete and clear.
    c. Efficiency of code. You shall not use too many SQL statements, and you shall put as much work as possible in SQL. For example, if you can do a join, do not use two select statements and then do a join in your program.
    d. Whether it has considered all special cases such as whether a user has already registered in Feature 1.

Regarding the presentation of your project: Each student must participate in the project demonstration by presenting to the entire class some slides. You will be graded on:

1. Timeliness of presentation
2. Presentation Style
3. Demo (running the code)

For the demo, you will be graded on the following items:

1. Existence of tables and data. You need to have at least 3 rows in each table.
2. The correctness of features. This can be shown by checking whether the screen output is correct and the database has been updated correctly.

Each member of the team shall contribute more or less equally. It is unfair for a few members to do most of the work while others do less. You will be asked to evaluate your teammate's effort at the end of the project. The instructor will adjust the grade based on the evaluation. Normally if most of your teammates agree that you do not contribute at all or contribute too little (e.g., your group has 4 members and you contribute only 5%), you may lose up to 80% of your project grade. If your teammates agree that you contribute much more than anyone else (e.g., your group has 4 members and you contribute 40%), you may gain up to 20% of your project grade (but not exceeding 100% of project grade). A peer evaluation will be conducted at the end of the semester to determine the contribution of each team member.

Tips:

1. Work as a team. Each member can do individual features by yourself but should work on other parts of the project including group features as a team. This means do NOT miss group meeting, do not miss internal deadlines, and help each other for tasks that belong to the whole group. You should also divide up group tasks fairly and according to everyone's strength.
2. Start early. Do not wait until last month to start coding. Do not wait until one week before the demo to start putting things together. Past experiences show that more than 50% of time shall be devoted to testing and putting things together.
3. Learn how to debug SQL and PL/SQL code. Most of time the error is from the SQL part of your code. So you can test SQL part separately (e.g., by copy &

paste the SQL statement in a cursor and replace PL/SQL variables/parameters with values). You can insert screen output statements to check intermediate results. Oracle also returns error messages and error code. You can google the error messages and error code to find possible causes. You may also use Oracle SQL Developer which allows you to insert break points during debugging.

4. It is highly recommended to use SQL Developer rather than the web interface for the project.
5. Use homework, in class exercises, and programs in slides as templates of your PL/SQL program. For example, if you need to write a cursor, find a cursor example and use it as a starting point.
6. Make sure special cases are handled.
7. At demo time, different data in the database may lead to different results. So usually you will start with a standard database (with a fixed set of tables and rows), and keep track of the sequence of the demo (e.g., a course can only be scheduled if it has been added first).