THE UNIVERSITY OF SYDNEY

School of Information Technologies
A/Prof Uwe Röhm

INFO2120: Database Systems **1.Sem./2014**

DB Project Assignment Part 3: Database Programming

Group Assignment (15%)
(INFO2120) **02.05.2014**

Introduction and Objectives

This assignment is about programming a car sharing booking application based on our database. The objectives are to gain practical experience in database and transaction programming.

This is a group assignment for **teams of up-to 3 members** that is **due on Tuesday of Week 12 (27 May) at 4 pm**. Your solution will be marked for correctness and completeness of the database application with regard to the grading scheme as described on the last page.
Please note: The individual mark awarded for each assignment is conditional on *each individual team member* being able to explain details of your database application code to your tutor or the subject coordinator if asked. Late submissions will attract a 20% penalty per day late.

Please also keep an eye on *the discussion forum and announcements* in Piazza. You will find there also links to on-line documentation and hints on tools and languages needed for this assignment.

Academic Honesty

IMPORTANT: Policy relating to Academic Dishonesty and Plagiarism.
All teams must declare that the work is original and not plagiarised from the work of others. In assessing a piece of submitted work, the School of IT may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program. A copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.

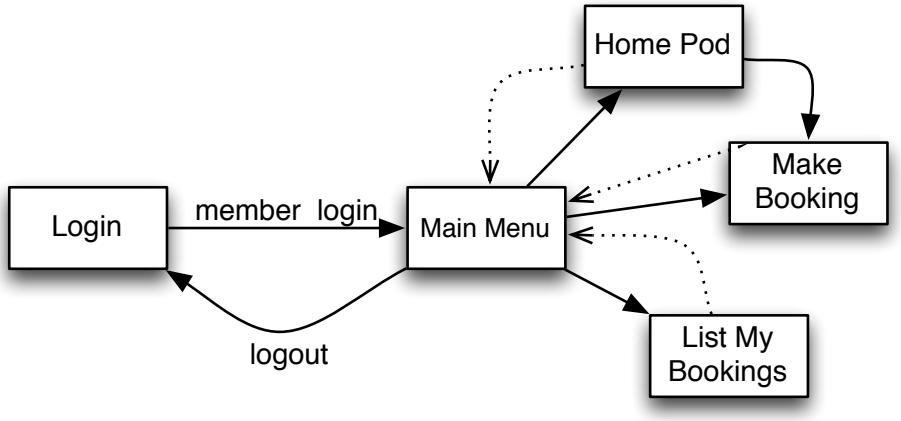
Programming an Car-Share Booking System

In this assignment your task is to implement the functions required to support the database interactions of an car-share booking system. We recommend using **PHP/PDO as programming language** for your client code. We will provide a complete user interface written in PHP, for which you need to write the appropriate database functions using the PDO API introduced in Week 8. If you wish to write an alternative client interface (for instance in another language) you may do so, but no support can be provided, and your work will only be assessed on the quality of the database interface code. We also will provide a **reference schema for PostgreSQL, as well as some example data**.

Note: Please check the provided SQL for any schema additions as compared to assignment 2.

Design Brief for the Car-Share Booking System (INFO2120)

Core Booking Functionality: As core functionality, implement the following six screens using the provided database schema:



a) At the LOGIN screen, car-sharing members can log in with their username and password. Your code should verify those values against the data stored in your database. When a valid user/password combination was entered, members shall be directed to the MAIN MENU screen. If username or password are incorrect, then give an appropriate error message.

b) The MAIN MENU screen should

- show the members full name, the name of his home pod (if any), and also any members booking statistics.
- Let the user then select from any of the available sub-screens

c) The HOME POD screen should show the details of the cars at the current user's home pod and whether each car it is available at the current time (i.e. it is not currently booked). Note that there could be more than one car at this car pod. For members without a home pod, prompt the user to enter a location name.

d) The MAKE BOOKING screen shall allow a logged-in user to book an available car. To do so , allow the user to enter a car (entering the car's name would be Ok) and to specify the start and end date and time of the booking (full hours only).

The following *transaction* should be an important focus of your submission:
In a single database transaction, your system should check whether the car is available for the whole time period, and if yes, create a new booking and update the user statistics in case of a successful booking (how many bookings a user has made). If a car is not available for the whole time period, give an appropriate error message.

If the booking was successful, generate a booking confirmation page that summarises the booking for the user again: Which car at what car pod (including car and location names and address), for what times and duration and what the estimated costs will be by just looking at the booking duration.

e) The LIST MY BOOKINGS screen should list all bookings of the current user in reverse chronological order (with regard to the start time of a booking and showing the latest booking first).

Distinction-Level Functionality Option 1: Car Reviews and Ratings
Add some functionality that allows users to rate cars. This should include:

- a facility to see the ratings and reviews for a certain car (and who wrote the reviews(s));
- a screen to add a review to a car, including a review text and a numerical rating (1 to 5);
- a correct **SQL transaction** that stores the reviews in the corresponding tables of our database – including who entered the review and also updating any corresponding member statistics.

Distinction-Level Functionality Option 2: FRAT Analysis
Add a 'Member Analysis' page that gives a report about all members with the following information:

Frequency How frequent a user books cars in average (since his/her first booking) on a **Scale of 1 to 5**

Recency How recent a user has booked a car (referring to the start date of a booking) **Scale: 1 to 5**

Amount How much money a user has spend in average per month (from first time booking) **Scale: 1 to 5**

Type What type of car-share user it is, with two possible values:
'weekend' : mainly books cars on weekends (Saturday or Sunday)
'weekday': mainly books cars during the week (Monday to Friday)

The scales (1 to 5) of the first three dimensions are quintile values: If you order the members' values for the corresponding dimension from top to bottom, members in the top 20% should be rated with a value of 5, in the next 20% it should be value 4, and so on until members in the bottom 20% would get a scale value of 1.
The report shall list each member with the name and the four 'FRAT' values in descending order of the FRAT values (so frequency 5 first, then recency 5 etc). Highlight in the current user's entry.

Distinction-Level Functionality Option 3: Invoicing
Program a database function `create_invoice(account, year, month)` that creates the monthly invoice for a certain member account for a given month and year. This function should read the database and match the trip log entries against the bookings of the account members in the given month and year and calculate from those trips the overall booking duration and used distance to be charged.

Distinction-Level Functionality Option 4: Database Abstraction Layer and Security
The web-based application in its current form has unrestricted access to your database. If a hacker were able to access or change the code they could do a lot of damage. Improve the security of your system by connecting as a public user (username and password '`info2120public`'), and granting minimal privileges to this user. You should avoid granting any direct access to tables, and instead provide a secure layer of views and stored procedures that only allow specific operations.

Distinction-Level Functionality Option 5: Physical Database Optimisations
Suggest and create indexes which make your most frequent queries and transactions faster. Also create one 'materialised' view using triggers that is used in your application rather than a dynamic query (e.g . the car availabilities or the FRAT analysis of Option 2). Explain your choices and also include a discussion of the effect of your decisions on updates in the attached discussion file.

Distinction-Level Functionality Option 6: Own Extension
The final option is to implement the required data entry and listing screens for your own model extension from the previous two assignments. This only applies if you did an extension so far. Note that a pure extension of the database schema is not enough; you need to have corresponding user-level functionality implemented too, including appropriate SQL transactions and queries.

Submission Details

The main assessment will take place in your lab of Week 12 where your team should give **a short demo** (10-15 mins) of your solution to your tutor. Each team member should present a selected part of the functionality. The tutors will also do a brief code review of one selected function of your solution.

Please submit your solution **in the 'Assignment' section of our eLearning site** on the **Tuesday of Week 12** as a *zip* or *tar* archive. Your submission should include

- a filled-in (or scanned) **assignment cover sheet** (Word file) that the work is original and not plagiarised from the work of others. .
- any **client-side source code** (if you use our skeleton code, clearly indicate your own contributions)
- an **SQL script to create any server-side stored procedures, functions, triggers, indexes or grant statements** for PostgreSQL which your created as plain text file with ending .sql
- a **Discussion.txt** file containing a (brief) discussion of any implementation problems and how your team resolved them, as well as any further comments on your solution.

Assessment Criteria

Your solution will be marked on the quality of your team's demo, and for correctness of the database access and completeness of the functionality, as well as on general code quality (code review). You can find the detailed marking rubric on the assignment submission page in eLearning.

Grade Descriptors

High Distinction 85 – 100%	The submitted code demonstrates an outstanding understanding of database application development and the associated security issues including a clear separation of the data access layer from the application logic and the presentation layer. It is a complete solution of the whole scenario including **at least two of the additional functionalities** with excellent quality of all deliverables, especially with regard to the transaction semantics or the correct usage of SQL queries. The application correctly protects against SQL injection attacks, gives only user-level error messages.
Distinction 75 – 84%	Thorough understanding of database application development and security: The submitted application is a complete solution of the core car sharing functionality and also implements **at least one of the additional functionalities**; All update processes are coded (correctly) as SQL transactions, and at least one of these transactions is implemented as a stored procedure; The application correctly protects against SQL injection attacks and in case of a database error, appropriate user-level error messages are given.
Credit 65 – 74%	Good understanding of database application development and security: The submitted application adequately implements the core car-share booking functionality and makes correctly use of the database queries; All update processes are coded (correctly) as SQL transactions, and at least one of these transactions is implemented as a stored procedure; The application tries to protects against SQL injection attacks and in case of a database error, appropriate user-level error messages are given; average quality of deliverables.
Pass 50 – 64%	The submitted application implements the core car booking functionality (cf. description on page 2), and makes correctly use of database queries; for example any possible filter condition is expressed in SQL queries rather than just scanning a database table and searching for the right entry outside the database. The submission also codes at least the 'Make Booking' functionality as a correct SQL transaction.
Fail below 50%	Falls short of the basic requirements for a Pass.