THE UNIVERSITY OF SYDNEY

School of Information Technologies
A/Prof Uwe Roehm

INFO2120: Database Systems I **Semester 1, 2015**

Assignment 3 Brief: Database Application Programming

1 Introduction

This assignment is about the programming of database interaction code within an application, building on the carpark-sharing database scenario introduced in Assignments 1 and 2. The objectives are to gain practical experience interacting with a relational database using an Application Programming Interface (API) and transaction programming, and to understand the importance and application of basic security measures. There are also opportunities to use more advanced database application programming techniques, such as stored procedures, triggers, indexes and access control privileges. We also included an optional extension regarding a suitable interface design.

This is a group assignment for teams of about 3 members, and it is assumed that you will continue in your Assignment 2 group. You should inform your tutor as soon as possible if you wish to change groups.

Please also keep an eye on the discussion forum and further announcements in Piazza.

2 Submission Details

The final version should be submitted via eLearning by midnight Monday Week 12. There will also be a submission demo by the whole group in the labs of Week 12.

2.1 Submission Items

Please submit your solution in the 'Assignment' section of the unit e-learning site by the deadline, including the following items:

Client Source Code For most groups this will be a modified version of the `include/database.php`, but if more substantial changes have been made you should provide a zip file containing each of the files you have changed, along with a short `Changelog.txt` file clearly summarising your group's contributions to each file;

Database Schema DDL If you have done any extensions that modify the database you should include all such additions (`ALTER TABLE` statements, views, server-side stored procedures, functions, triggers, indexes or grant statements for PostgreSQL which your created as plain text file with `.sql` file suffix). *You should ensure that this file runs on a clean version of the original schema on the PostgreSQL 9.4 database without errors.*

2.2 Plagiarism

By uploading your submission to eLearning your group implicitly agrees to abide by the University policies regarding academic honesty, and in particular that all the work is original and not plagiarised from the work of others. If you believe that part of your submission is not the work of your group members you must bring this to the attention of your tutor or lecturer immediately. See the policy slides released in Week 1 for further details.

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.

2.3 Late submissions

Late submissions will be penalised 20% per day late.

3 Marking

This assignment is worth 15% of your final grade for INFO2120. Your group's final submission will be marked according to the attached rubric both offline, and during a demo in the labs of Week 12.

3.1 Rubric

Your submissions will be marked according to the following rubric (maximum score: 15 pts).

	Novice (0 pts)	**Competent (1-2 pts)**	**Proficient (3 pts)**
Core functionality	Core functionality not satisfactory implemented	Good attempt at most functionality but some significant omissions or mistakes	All functionality implemented with no or only a few minor mistakes.
SQL	Less than competent use of SQL	Sound use of basic SQL statements throughout.	Excellent use of SQL throughout client, including appropriate use of complex SQL (e.g., GROUP BY, OUTER JOIN).
Transactions	Less than competent model of the given scenario	Demonstrated understanding and implementation of transactions for at least one major function.	All functions implemented with excellent understanding of transactions, with evidence of handling of failed commits and consideration of appropriate isolation levels.
Security and Stored Procedures	No protection against SQL injection or use of stored procedures	Good attempt of a (non-trivial) stored procedure and/or protection against SQL injection.	Comprehensive protection against SQL injection and excellent use of stored procedures for all non-trivial queries.
Extension(s)	No extensions beyond the core functionality	One extension was attempted, but either with mistakes or not very substantial.	At least one substantial extension to the given scenario was included, described and implemented correctly.

3.2 Feedback

Your group's final submission will receive feedback, explaining your final mark with respect to the marking rubric. In the tutorials before the submission, there will be the possibility to gain initial feedback from your tutor during the development process.

3.3 Group member participation

If members of your group do not contribute sufficiently you should alert your tutor as soon as possible. The tutor has the discretion to scale the group's mark for each member as follows, based on the outcome of the group's demo in Week 12:

Level of contribution	Proportion of final grade received
No participation.	0%
Full understanding of the submitted work.	50%
Minor contributor to the group's submission.	75%
Major contributor to the group's submission.	100%

4 Design Brief: Programming a Carpark-Sharing Client Application

In this assignment your task is to implement the functions required to support the database interactions of an online carpark-sharing system, hosted on the School's PostgreSQL server. You will be provided with a *reference schema* for PostgreSQL, as well as some *example data*. We will also provide a *complete user interface written in PHP*, for which you need to write the appropriate database interaction functions using the PHP/PDO API introduced in Week 8. In writing these functions you should consider the following issues, which will be taken into account during marking:

SQL Your code should make best use of the database to correctly retrieve and update data. In particular, you should avoid writing client-side code for operations, such as joins, that could be better done within the database.

Transaction Handling You should assume that multiple clients will be running concurrently on the same database, so your functions should make suitable use of transactions. You should consider where to commit or roll back these transactions, and what to do if a transaction fails. D/HD students should also select appropriate isolation levels for their transactions.

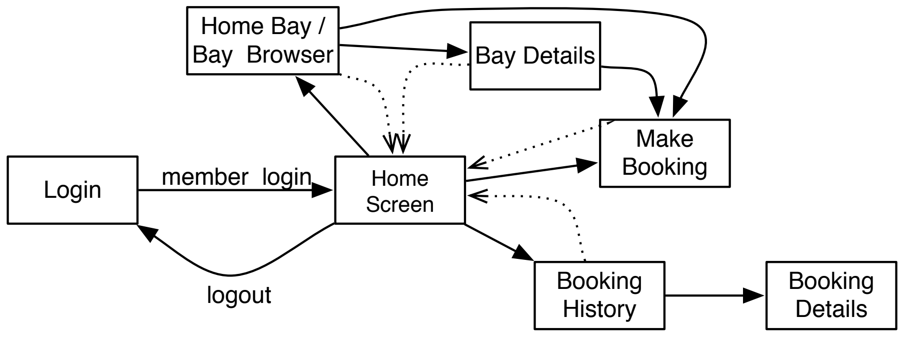
Security Multi-tier architectures increase the scope for nefarious users to gain unintended access to query or modify your database. You should take steps to limit this by preventing SQL Injection attacks, and limiting the privileges available to the client to specific operations on tables, views and stored procedures.

Stored Procedures Network traffic can be reduced (and cross-client portability increased) by wrapping complex database operations into stored procedures that are run within the database rather than in the client. You should make use of these where appropriate.

Your submission should support the functionality detailed below.

4.1 Core Functionality

The application features a set of screens, described as follows.



4.1.1 Login

At the login screen, members can log in with their email address (or optionally their nickname) and password. Your interface should verify those values against the data stored in your database. When a valid user/password combination is entered, members shall be directed to the member home screen.

4.1.2 Home Screen

In the member's home screen the user should be greeted with their full name, and see the following details:

- A list of the member's cars (including name, registration number, make and model)
- Number of bookings made by the user (from the statistical information stored for each member).

4.1.3 Booking Creation

A user should be able to use this page to make a booking of a car for a bay for a specific period. In making the booking the application must:

i) Check availability (basic availability plus no clashes with other bookings)

ii) Check car fits the bay space

iii) Create a new booking entry

iv) Estimate the cost of the booking according to the member's plan

If successful, details are shown in the Booking Details screen.

4.1.4 Booking Details

For a given booking, specified by its booking ID number, this screen should display:

- Car details (name, registration)
- Bay details (location of bay)
- Booked period
- Time and date of when the booking was made
- Booking cost (estimated if not yet invoiced)

4.1.5 Booking History

This screen should list all the member's bookings. Each item in the list should include:

- the bay location
- the corresponding car's name
- the reserved date

The bookings should be in reverse chronological order (most recent first) according to the reserved date. A user can choose to see further details of a booking in the Booking Details screen.

4.1.6 Bay Browser

This screen allows the user to search for a bay according to the bay's address. All bays matching the given address string – even only partially – should be listed. The matching bays should be listed with the following attributes:

- full address
- site details

4.1.7 Bay Details

This page should give all the details of a particular bay (by default the member's preferred bay), including:

- full address
- site details
- bay dimensions
- general week-day and weekend availability

Also, the details should include a list of which hours the bay is available for the current day, taking account of any existing bookings.

4.2 Extensions

Proficiency in core skills and application of more advanced skills can be demonstrated through implementation of extensions to the core functionality. Students wishing to attain a D/HD mark for this assignment should implement at least one extension that demonstrates research and application of skills or techniques beyond those covered in the core brief, such as indexes, triggers, views, or recursive/analytical SQL. You should consult your tutor for guidance on what would be an appropriate extension for your group, and what the criteria will be for marking them. Below are a few suggestions for possible extensions . Depending upon the scope you may wish to do one large extension or a couple of smaller ones.

4.2.1 Option 1: Physical Optimisations and Reservation Materialised View

A common task handled by the database is checking whether a bay is available for a given period. To reduce the burden of this operation , an reservation table can be written to record which hours are reserved for any booked bays. Availability checks can then perform a query on this table. Support this by:

i) Adding this reservation table
ii) Write a query to populate this table for the existing bookings
iii) Write triggers for the Booking table to make corresponding updates to the reservation table.
iv) Suggest and create indexes which make your reservation queries and transactions faster.

4.2.2 Option 2: Invoicing

Each month a new invoice is generated by the company for each member to calculate the costs owed and due to the member. This needs to include the plan fees, booking costs for any cars owned by the member and parking revenues for any bays. Write a stored procedure to populate the invoice data for a specific member for a given month, and support this with functionality for a member to view a list of all their invoices and the specific details for a particular invoice.

4.2.3 Option 3: Data-User-Interaction Analysis and Design

Analyse the user interface of the given skeleton code with regard to

- its usability on a mobile device such as a smartphone,
- finding an available parking bay close to the current position,
- an efficient way to extend a currently active booking.

Which parts of the user interface would you want to change to support these usability criterions? Design an alternative interface that supports those functions better, and explain how the database access part is affected by your changes (such as which kinds of queries would either needed to be changed or added). You submission for this extension should include:

- a textual discussion of your usability analysis with regard to above's criteria,
- a wireframe of your planned revised site layout,
- a mockup of your new interface design, and
- a discussion of which parts of the database-related code would be affected by your design.

Option 4: FRAT Analysis

Add a ’Member Analysis’ page that gives a report about all members with the following information:

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

Recency How recent a user has booked a car park (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 carpark-share user it is, with two possible values:
’weekend’ : mainly books car parks on weekends (Saturday or Sunday)
’weekday’: mainly books car parks 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.