THE UNIVERSITY OF SYDNEY

School of Information Technologies
Dr. Uwe Röhm

INFO2120: Database Systems **1.Sem./2013**

DB Project Assignment Part 3: Database Programming

Group Assignment (15%)
(INFO2120) **08.05.2013**

Introduction and Objectives

This assignment is about the programming an online treasure hunt game 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 Friday of Week 12 (31 May) at 6pm**. 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.

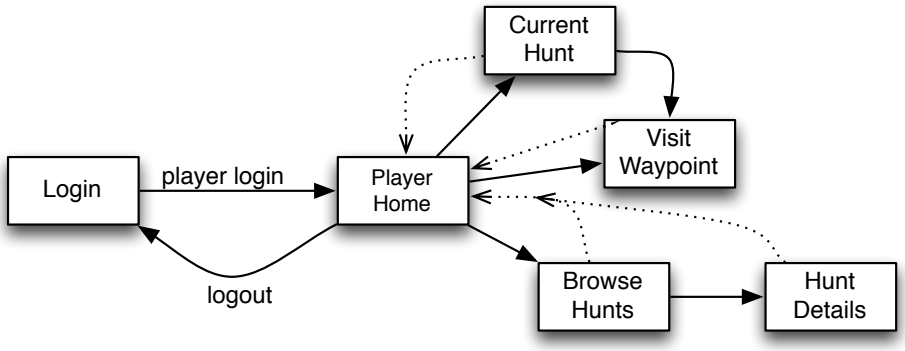
Programming an Online Treasure Hunt Game

In this assignment your task is to implement the functions required to support the database interactions of an online treasure hunt 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 Online Treasure Hunt System (INFO2120)

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



a) At the LOGIN screen, players 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, players shall be directed to the PLAYERS DETAILS screen. If username or password are incorrect, then give an appropriate error message.

b) The PLAYER HOME screen should

- show player details such as name, address, and the name of the current team (if any), and also any player statistics.
- Also include here a list of the achieved badges.
- Let the user then select from any of the available sub-screens

c) The CURRENT HUNT screen should show the details of the player's latest hunt (if any), such as the progress so far (duration and number of correct visits) and information about the next waypoint, or when the hunt is finished, the team's duration, score and final rank.

d) The VISIT WAYPOINT screen shall allow a logged-in player to attempt to visit a waypoint. To do so, allow the player to enter a verification code (it's ok to ignore the GPS position here).

The following transaction should be an important focus of your submission:
In a single database transaction, your system should check whether the entered code matches the expected code of the next waypoint for your team in its current hunt. If yes, this should be recorded as a valid Visit and the team's current waypoint and score should be updated. The team should then receive the clue to the next waypoint. If the entered code does not match the expected code (the team is not at the correct waypoint), your code still should store a Visit attempt, but marked as incorrect, and give appropriate feedback to the player. In both cases, the visit should be saved with the current timestamp.

If this was the last visit of the current hunt – and it was correct –, the system should as part of the same transaction update the team's hunt statistics, such as the overall duration, score and rank. Then show a congratulation message to the player.

e) The BROWSE HUNTS screen should list all available (not under construction) hunts in alphabetical order. From each entry , one should be able to get to a HUNT DETAILS page.

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

- a facility to see the ratings and reviews for a certain hunt (and who wrote the reviews(s));
- a screen to add a review to a specific hunt, including a review text and a numerical rating;
- 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 Player Analysis
Add a 'Player Analysis' page that gives a report about all players with the following information:

Frequency How frequent a player participates in hunts (since his/her first hunt) on a **Scale of 1 to 5**

Recency How recent a player has participated in a hunt (referring to the start date of a hunt) **Scale: 1 to 5**

Amount How much points the player's teams have achieved in average per month **Scale: 1 to 5**

Type What type of player a user it is, with two possible values:
'weekend' : mainly plays hunts on weekends (Saturday or Sunday)
'weekday': mainly plays hunts during the week (Monday to Friday)

The scales (1 to 5) of the first three dimensions are quintile values: If you order the player's values for the corresponding dimension from top to bottom, players 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 players 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 player's entry.

Distinction-Level Functionality Option 3: Database Abstraction Layer and Security

1. Firstly , implement all database functionalities (including queries) as separate stored procedures, that are invoked from your PHP front-end.
2. In addition, your application should use a generic 'info212_public' user to connect to the database, who you granted only minimal access to tables and stored procedures as needed.

Distinction-Level Functionality Option 4: Hunt Visualisation
The next option is to implement a visualisation of the current hunt, such as where teams currently are in relation to each other and the overall hunt length. This could be in abstract form of some graphic, or plotted spatially on a map. The exact format is up to you, as long as the data is queried dynamically from the database.

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 (for example about 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 13 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 **Friday 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 treasure hunt 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 treasure hunt 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 treasure hunt 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 the 'Visit Waypoint' functionality as a correct SQL transaction.
Fail below 50%	Falls short of the basic requirements for a Pass.