Project 2 PLpgSQL

Please note that we have different deadline for each question. Please make sure you have submitted each question by its own deadline:

Section 1: ~~Fri 5~~ Sun 7 May, 5:00 pm

Section 2: Fri 19 May, 5:00 pm

Section 1

1. Aims

This project aims to give you practice in

- reading and understanding a moderately large relational schema (MyMyUNSW)
- implementing SQL queries and views to satisfy requests for information
- implementing PLpgSQL functions to aid in satisfying requests for information

The goal is to build some useful data access operations on the MyMyUNSW database. A theme of this project is "dirty data". As I was building the database, using a collection of reports from UNSW's information systems and the database for the academic proposal system (MAPPS), I discovered that there were some inconsistencies in parts of the data (e.g. duplicate entries in the table for UNSW buildings, or students who were mentioned in the student data, but had no enrolment records, and, worse, enrolment records with marks and grades for students who did not exist in the student data). I removed most of these problems as I discovered them, but no doubt missed some. Some of the exercises below aim to uncover such anomalies; please explore the database and let me know if you find other anomalies.

2. How to do this project:

- read this specification carefully and completely
- familiarize yourself with the database **schema** (description, SQL schema, summary)
- make a private directory for this project, and put a copy of the **proj2.sql** template there
- you **must** use the create statements in **proj2.sql** when defining your solutions
- look at the expected outputs in the expected_qX tables loaded as part of the **check.sql** file
- solve each of the problems below, and put your completed solutions into **proj2.sql**
- check that your solution is correct by verifying against the example outputs and by using the check_qX() functions
- test that your **proj2.sql** file will load *without error* into a database containing just the original MyMyUNSW data
- double-check that your **proj2.sql** file loads in a *single pass* into a database containing just the original MyMyUNSW data

3. Introduction

All Universities require a significant information infrastructure in order to manage their affairs. This typically involves a large commercial DBMS installation. UNSW's student information system sits behind the MyUNSW web site. MyUNSW provides an interface to a PeopleSoft enterprise management system with an underlying Oracle database. This back-end system (Peoplesoft/Oracle) is often called NSS.

UNSW has spent a considerable amount of money ($80M+) on the MyUNSW/NSS system, and it handles much of the educational administration plausibly well. Most people gripe about the quality of the MyUNSW interface, but the system does allow you to carry out most basic enrolment tasks online.

Despite its successes, however, MyUNSW/NSS still has a number of deficiencies, including:

- no waiting lists for course or class enrolment
- no representation for degree program structures
- poor integration with the UNSW Online Handbook

The first point is inconvenient, since it means that enrolment into a full course or class becomes a sequence of trial-and-error attempts, hoping that somebody has dropped out just before you attempt to enrol and that no-one else has grabbed the available spot.

The second point prevents MyUNSW/NSS from being used for three important operations that would be extremely helpful to students in managing their enrolment:

- finding out how far they have progressed through their degree program, and what remains to be completed
- checking what are their enrolment options for next semester (e.g. get a list of suggested courses)
- determining when they have completed all of the requirements of their degree program and are eligible to graduate

NSS contains data about student, courses, classes, pre-requisites, quotas, etc. but does not contain any representation of UNSW's degree program structures. Without such information in the NSS database, it is not possible to do any of the above three. So, in 2007 the COMP9311 class devised a data model that could represent program requirements and rules for UNSW degrees. This was built on top of an existing schema that represented all of the core NSS data (students, staff, courses, classes, etc.). The enhanced data model was named the MyMyUNSW schema.

The MyMyUNSW database includes information that encompasses the functionality of NSS, the UNSW Online Handbook, and the CATS (room allocation) database. The MyMyUNSW data model, schema and database are described in a separate document.

4. Setting Up

To install the MyMyUNSW database under your Grieg server, simply run the following two commands:

```
$ createdb proj2
$ psql proj2 -f /home/cs9311/web/17s1/proj/proj2/mymyunsw.dump
```

If everything proceeds correctly, the load output should look something like:

```
SET
... a whole bunch of these
SET
CREATE EXTENSION
COMMENT
SET
CREATE DOMAIN
CREATE DOMAIN
CREATE DOMAIN
CREATE DOMAIN
CREATE TYPE
CREATE DOMAIN
... a whole bunch of these
CREATE DOMAIN
CREATE FUNCTION
CREATE FUNCTION
SET
SET
CREATE TABLE
... a whole bunch of these
CREATE TABLE
ALTER TABLE
ALTER TABLE
... a whole bunch of these
ALTER TABLE
```

Please carefully investigate any occurrence of ERROR during the load. The database loading should take less than 60 seconds on Grieg, assuming that Grieg is not under heavy load. (If you leave your project until the last minute, loading the database on Grieg will be considerably slower, thus delaying your work even more. The solution: at least load the database Right Now, even if you don't start using it for a while.) (Note that the mymyunsw.dump file is 50MB in size; copying it under your home directory or your /srvr directory is not a good idea).

If you have other large databases under your PostgreSQL server on Grieg or you have large files under your /srvr/YOU/ directory, it is possible that you will exhaust your Grieg disk quota. In particular, you will not be able to store two copies of the MyMyUNSW database under your Grieg server. The solution: remove any existing databases before loading your MyMyUNSW database.

If you are running PostgreSQL at home, the file proj2.tar.gz contains copies of the files: mymyunsw.dump, proj2.sql to get you started. You can grab the check.sql separately, once it becomes available. If you copy proj2.tar.gz to your home computer, extract it, and perform commands analogous to the above, you should have a copy of the MyMyUNSW database that you can use at home to do this project.

A useful thing to do initially is to get a feeling for what data is actually there. This may help you understand the schema better, and will make the descriptions of the exercises easier to understand. Look at the schema. Ask some queries. Do it now.

Examples ...

```
$ psql proj2
... PostgreSQL welcome stuff ...
proj2=# \d
... look at the schema ...
proj2=# select * from Students;
... look at the Students table ...
proj2=# select p.unswid,p.name from People p join Students s on (p.id=s.id);
... look at the names and UNSW ids of all students ...
proj2=# select p.unswid,p.name,s.phone from People p join Staff s on (p.id=s.id);
... look at the names, staff ids, and phone #s of all staff ...
proj2=# select count(*) from Course_Enrolments;
... how many course enrolment records ...
proj2=# select * from dbpop();
... how many records in all tables ...
proj2=# ... etc. etc. etc.
proj2=# \q
```

You will find that some tables (e.g. Books, Requirements, etc.) are currently unpopulated; their contents are not needed for this project. You will also find that there are a number of views and functions defined in the database (e.g. dbpop() and transcript() from above), which may or may not be useful in this project.

Summary on Getting Started

To set up your database for this project, run the following commands in the order supplied:

```
$ createdb  proj2
$ psql  proj2  -f  /home/cs9311/web/17s1/proj/proj2/mymyunsw.dump
$ psql  proj2
... run some checks to make sure the database is ok
$ mkdir  Project2Directory
... make a working directory for Project 2
$ cp  /home/cs9311/web/17s1/proj/proj2/proj2.sql  Project2Directory
```

The only error messages produced by these commands should be those noted above. If you omit any of the steps, then things will not work as planned.

Notes

Read these before you start on the exercises:

- the marks reflect the relative difficulty/length of each question
- use the supplied **proj2.sql** template file for your work

- you may define as many additional functions and views as you need, provided that (a) the definitions in proj2.sql are preserved, (b) you follow the requirements in each question on what you are allowed to define
- make sure that your queries would work on any instance of the MyMyUNSW schema; don't customize them to work just on this database; we may test them on a different database instance
- do not assume that any query will return just a single result; even if it phrased as "most" or "biggest", there may be two or more equally "big" instances in the database
- you are not allowed to use ***limit*** in answering any of the queries in this project
- when queries ask for people's names, use the `People.name` field; it's there precisely to produce displayable names
- when queries ask for student ID, use the `People.unswid` field; the `People.id` field is an internal numeric key and of no interest to anyone outside the database
- unless specifically mentioned in the exercise, the order of tuples in the result does not matter; it can always be adjusted using `order by`. In fact, our check.sql will order your results automatically for comparison.
- the precise formatting of fields within a result tuple **does** matter; e.g. if you convert a number to a string using to_char it may no longer match a numeric field containing the same value, even though the two fields may look similar
- develop queries in stages; make sure that any sub-queries or sub-joins that you're using actually work correctly before using them in the query for the final view/function

Each question is presented with a brief description of what's required. If you want the full details of the expected output, take a look at the expected_qX tables supplied in the checking script.

5. Tasks

Note that the mymyunsw.dump used in project 2 is different from that used in project 1, please confirm that you load the correct database when you start your work.

Q1(5 marks)

You may use any combination of views, SQL functions and PLpgSQL functions in this question. However, you must define at least a PLpgSQL function called `Q1`.

The way UNSW computes Equivalent full-time student load (EFTSL) of each subject depends on uoc of this subject. The value of EFTSL equals the uoc of this subject divided by 48. However, we found that not all subject records are correct with this rule.

Please write a PLpgSQL function `Q1(pattern text, uoc_threshold integer)` that outputs two numbers: (1) among all the subjects with incorrect EFTSL, the number of subjects whose code matches `pattern`; (2) among all the subjects with incorrect EFTSL, the number of subjects whose code matches `pattern` and uoc is greater than the given threshold `uoc_threshold`.

You should use the following type definition and function header:

```
create type IncorrectRecord as (pattern_number integer, uoc_number
integer);
create or replace function Q1(pattern text, uoc_threshold integer)
returns IncorrectRecord...
```

Note: You do not need to consider tuples where uoc is null or eftsload is null

Sample results (details can be found in check.sql):
proj2=#select * from q1('ECO%', 6);

pattern_number	uoc_number
79	5

Q2 (6 marks)

You may use any combination of views, SQL functions and PLpgSQL functions in this question. However, you must define at least a PLpgSQL function called `Q2`.

Please write a PLpgSQL function `Q2(stu_unswid integer)` that takes as parameter a student's `unswid` and returns all transcript records of the given student. Each transcript tuple should contain the following information: cid, term, code, name, uoc, mark, grade, rank and totalEnrols.

You should use the following type definition for the transcript tuples:

```
create type TranscriptRecord as (
     cid integer,     -- course ID
     term char(4),    -- semester code (e.g. 98s1)
     code char(8),    -- UNSW-style course code (e.g. COMP1021)
     name text, -- corresponding subject name of the course
     uoc integer,     -- units of credit the student obtained from
                         this course (full uoc grant to students
                         who have passed the course, 0 otherwise)
     mark integer,    -- numeric mark achieved
     grade char(2),   -- grade code (e.g. FL, UF, PS, CR, DN, HD)
     rank integer,    -- the rank of the student in this course
     totalEnrols integer,  -- the total number of students
                              enrolled in this course with non-null
                              mark
);
```

Note: Here is all the grades that indicate a pass: 'SY', 'RS', 'PT', 'PC', 'PS', 'CR', 'DN', 'HD', 'A', 'B', 'C', 'D', 'E'. (For your interest, please refer to this for information regarding which grades indicate a pass.) You don't have to worry about invalid `unswid` of students. We are not going to assess you on this. In case that mark is null, you should set rank as null. Also, a student whose mark is null should not be considered when you are ranking other students.

You should use the following and function header:

```
create or replace function Q2(stu_unswid integer) returns setof
TranscriptRecord...
```

Sample results (details can be found in check.sql):
proj2=#select * from q2(2220747);

cid	term	code	Name	uoc	mark	grade	rank	totalEnrols
6295	03s1	HIST5233	Mod China: History & HIstoring	8	89	HD	1	1

14104	05s1	ARTS5024	Research Writing	6	72	CR	3	4
…	…	…	…	…	…	…	…	…

Q3 (7 marks)

You may use any combination of views, SQL functions and PLpgSQL functions in this question. However, you must define at least a PLpgSQL function called `Q3`.

Given the id of an organizational unit, please write a PLpgSQL function `Q3 (org_id integer, num_sub integer, num_times integer)` to help the UNSW administrative officers to find out all the staff members satisfying the following: (1) he/she has delivered more than `num_sub` distinct subjects in the given organization over time; (2) he/she has delivered more than `num_times` times for at least one subject offered by the given organization. Each tuple should include:

- His/Her unswid
- His/Her name
- His/Her teaching records

Teaching records of a staff is a concatenation of several records. Each record is about a subject he/she has delivered more than `num_times` times, and is offered by the given organization. Each record should include the code of the subject, the times that he/she has delivered this subject, the name of the organization (you should look at `Subjects.offeredby`).

Note:
(1). `Course Tutor` is not included.
(2). A given organization may have lots of sub-organizations, and you need to include sub-organizations. For example, the faculty of engineering has 10 schools, such as biomedical engineering and CSE.
(3). Please note that, for `teaching_records`, there is a space after each ‘,’. All text fields are verified with exact text matching.
(4). Teaching records of a staff should be ordered in increasing order of `Subjects.id`. Records should be concatenated by ‘\n’, so that each record will be displayed in a separate line.

You should use the following type definition and function header:

```
create type TeachingRecord as as (unswid integer, staff_name text,
teaching_records text);
create or replace function Q3(org_id integer, num_sub integer,
num_times integer) returns setof TeachingRecord...
```

Sample results (details can be found in check.sql):
proj2=#select * from q3(52, 20, 8);

unswid	staff_name	teaching_records
8254273	Chris Sorrell	MATS1464, 10, Materials Science & Engineering, School of+ MATS6605, 9, Materials Science & Engineering, School of +
9282965	Chris Winder	SESC9810, 9, Risk & Safety Science, School of + SESC9820, 9, Risk & Safety Science, School of +
3053938	Susan Hagon	GENS4001, 9, Physics, School of + PHYS5012, 9, Physics, School of +

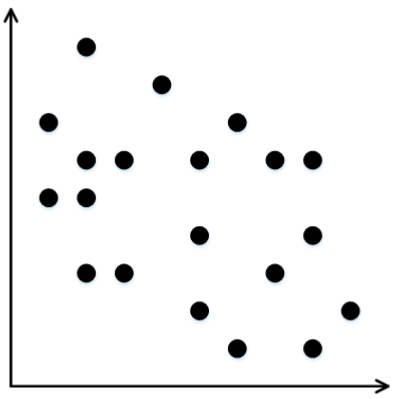
Note that PostgreSQL uses a + character to indicate an end-of-line in its output (as well as printing

'\n').

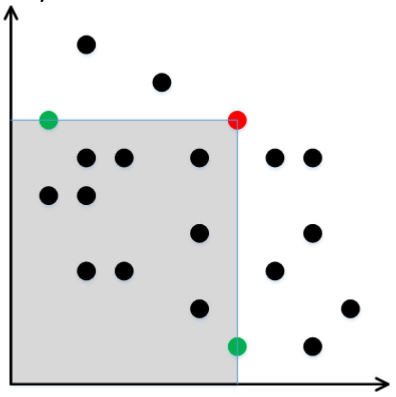
Section 2

1. Introduction

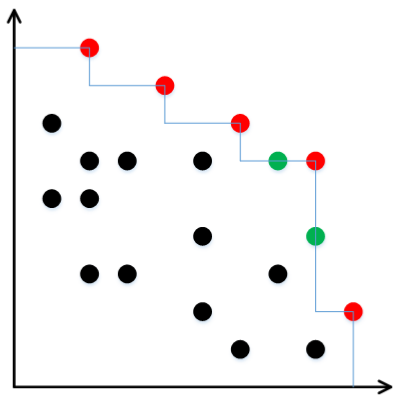
A spatial database is a database that is optimized to store and query data that represents objects defined in a geometric space. Most spatial databases allow representing simple geometric objects such as points, lines and polygons. In this project, we consider only points in 2 dimensional Euclidean spaces like this:



Let a point P be denoted as $(P.x, P.y)$. Assume higher values are preferred in both dimensions. A point P_1 is said to dominant another point P_2, if we have $(P_1.x \geq P_2.x \; and \; P_1.y > P_2.y)$ or $(P_1.x > P_2.x \; and \; P_1.y \geq P_2.y)$. Alternatively, we can also say that P_2 is dominated by P_1. For example, in the following image, the red point dominants all the points in the grey area, including the two green points on the boundary.



Now, we can introduce you the Skyline Operator. The Skyline Operator is used in a query and performs a filtering of results from a dataset so that it keeps only those points that are not worse than any other. That is, it selects all the points from a dataset where none of them are dominated by any points. These points are referred to as the Skyline of a dataset. For example, in the following image, the red points belong to the Skyline. But be careful with the two green points, they do not belong to the Skyline.



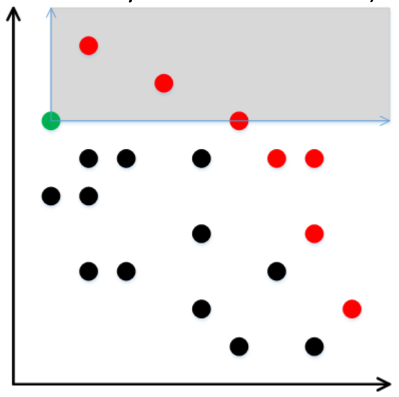
The Skyline Operator finds many applications. For example, let a point represent a computer, and x axis represents the quality of CPU while y axis represents the size of RAM. Without considering price, customers would always prefer computers with better CPU and larger RAM. Now, assume a customer evaluates a computer using this formula:

$$W.x * P.x + W.y * P.y$$

Where $W.x + W.y = 1$, and reflects the relative importance of $P.x$ and $P.y$ to a customer. Customers may weight $P.x$ and $P.y$ differently. For example, you may think they are equally important and have $W.x = 0.5$ and $W.y = 0.5$, or you may think that CPU has three times the importance of RAM and have $W.x = 0.75$ and $W.y = 0.25$. However, no matter how you weight the two parameters, your top one choice always belongs to the Skyline, but you may end up with different ones in the Skyline.

However, we are not always interested in the top one only. For example, a customer may found that all the top three choices are good enough in terms of their CPU and RAM, and realize that the third best comes with his/her favorite color, and end up purchasing the third best other than the best. In this case, we need to extend the concept of Skyline:

We define k-Skyband as the set of all the points such that each one of them is dominated by less than k other points. Let k=3, all the red points in the following image belong to 3-Skyband. The green point does not belong to 3-Skyband because there are three points dominating it (see the three red points in the shaded area or on the boundary of the shaded area).



2. Tasks

You are given datasets as 2 dimensional points. And your first task is to implement functions that perform the same as Skyline Operator. That is to find out Skyline from a given dataset.

A naïve way to do this is simply loop through all the points, and check for each point that if there is any points dominant this point. This process requires a join of the given dataset with itself.

Q4 (1 mark)

Implement a function, **skyline_naive(dataset text)**, that takes dataset (name of the given dataset) as input, create a view **$dataset$_skyline_naive(x, y)** that contains all the points that belong to the Skyline, and outputs the number of points belong to the Skyline.

Note:

(1). You should replace **$dataset$** with the name of the dataset. For example, invoking **skyline_naive('small')** should create a view **small_skyline_naive(x, y)**.

(2). You may want to use the dynamic SQL, the execute clause, in PL/pgSQL.

The naïve algorithm above has complexity $O(n^2)$, where n is the number of points in the dataset. There are algorithms achieve complexity better than that. For example, the following algorithm works in 2 dimensional space and has complexity $O(n \log n)$.

Sort the dataset by y axis in descending order, break ties with x axis in descending order. It can be proved that the first point in the sorted list always belongs to the Skyline. For the rest of the points in the sorted list, it belongs to the Skyline when it comes strictly to the right of the last found Skyline point, otherwise, it does not belong to the Skyline.

Next, we are going to implement the faster algorithm.

Q5 (2 marks)

Implement a function, **skyline(dataset text)**, that takes dataset (name of the given dataset) as input, create a view **$dataset$_skyline(x, y)** that contains all the points that belong to the Skyline, and outputs the number of points belong to the Skyline.

Note:

(1). You should replace **$dataset$** with the name of the dataset. For example, invoking **skyline('small')** should create a view **small_skyline(x, y)**.

(2). You may want to use the dynamic SQL, the execute clause, in PL/pgSQL.

(3). You need to pay attention to points that share the same value on either x axis or y axis (green points in previous examples).

Your second task is to implement functions that find out points belong to k-Skyband from a given dataset. Note that k is a variable given via parameters of your functions.

A naïve way to do this is simply loop through all the points, and check for each point that if there are less than k other points dominant this point. This process requires a join of the given dataset with itself.

Q6 (2 mark)

Implement a function, **skyband_naive(dataset text, k integer)**, that takes dataset (name of the given dataset) and k as input, create a view **$dataset$_skyband_naive(x, y)** that contains all the points that belong to the Skyband, and outputs the number of points belong to the Skyband.

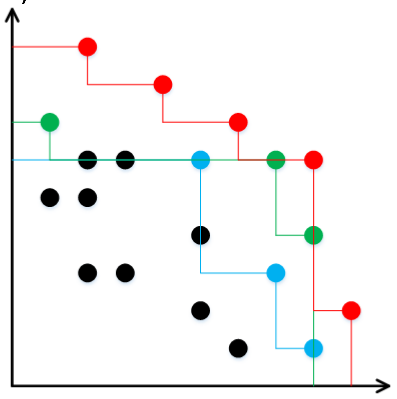
Note:

(1). You should replace **$dataset$** with the name of the dataset. For example, invoking **skyband_naive('small', 3)** should create a view **small_skyband_naive(x, y)**.

(2). You may want to use the dynamic SQL, the execute clause, in PL/pgSQL.

The naïve algorithm above has complexity $O(n^2)$. There are algorithms achieve complexity better than that. For example, the following algorithm works in 2 dimensional space and has complexity $O(n \log n + kn + m^2)$.

Sort the dataset by y axis in descending order, break ties with x axis in descending order. It can be proved that the first point in the sorted list always belong to the Skyline. We call this Skyline the 1st Skyline. Now, we remove points belong to the 1st Skyline, and calculate Skyline for the rest of the points, and we call this Skyline the 2nd Skyline. The following image illustrates 1st Skyline (red), 2nd Skyline (green) and 3rd Skyline (blue).



We continue this process, and calculate the k-th Skyline. Combine these k Skylines, we get a set of points. Let m be the number of points in this set. It can be proved that this set of points is a super set of the set of points belong to k-Skyband. And it can be proved that the k-Skyband of this set of points is the same as the k-Skyband of the original dataset. Thus, we can calculate this set of points in time $O(n \log n + kn)$, and then calculate k-Skyband in time $O(m^2)$ based on the smaller set of points.

Next, we are going to implement the faster algorithm.

Q7 (2 marks)

Implement a function, **skyband(dataset text, k integer)**, that takes dataset (name of the given dataset) and k as input, create a view **$dataset$_skyband(x, y)** that contains all the points that belong to the Skyband, and outputs the number of points belong to the Skyband.

Note:

(1). You should replace **$dataset$** with the name of the dataset. For example, invoking **skyband('small')** should create a view **small_skyband(x, y)**.

(2). You may want to use the dynamic SQL, the execute clause, in PL/pgSQL.

(3). You need to pay attention to points that share the same value on either x axis or y axis (green points in previous examples).

Specifications: we provide several files to help you finish this question:

- spatial.sql. a template file for your work.
- load_spatial.sh. The script helps create a database "spatial", a table "small" and a table "large", and insert the points specified in "small.txt" into the table "small" and the points specified in "large.txt" into the table "large". Note that we will test several datasets other than "small" and "large", so you are required to take a parameter "dataset" in all functions.
- small.txt, a small spatial dataset
- large.txt, a large spatial dataset
- check_spatial.sql. This is the test file. While implementing all functions, you can use "check_spatial.sql" to check whether the results are correct. Simply load the "check_spatial.sql" into the "spatial" database. If you have completed all the lab practices, you should know how to do this. Then, you can use "select * from check_all();" to check if your functions work properly on the dataset "small" and "large". The testing outputs would be as follows when everything is correct:

functions	dataset	result

skyline_naive	small	correct
skyline_naive	large	correct
skyline	small	correct
skyline	large	correct
skyband_naive	small	correct
skyband_naive	large	correct
skyband	small	correct
skyband	large	correct

Section 3

1. Submission

You can submit this project by doing the following:

- Ensure that you are in the directory containing the files to be submitted, includes proj2.sql and spatial.sql.
- For section 1, type “give cs9311 proj2s1 proj2.sql” to submit.
- For section 2, type “give cs9311 proj2s2 spatial.sql” to submit.
- If you submit your project more than once, the last submission will replace the previous one
- **To prove successful submission, please take a screenshot as assignment submission manual shows and keep it by yourself.**
- If you have any problems in submissions, please email to swan398@cse.unsw.edu.au or xwang@cse.unsw.edu.au. You can also ask questions about this project in our project online Q&A group, or come to our designated consultation session for projects (3:30pm-4:30pm Monday, K17-403), we will answer your questions as soon as possible.

Before you submit your solution, you should check that it will load correctly. If we need to manually fix problems with your file in order to test it, you will be fined via half of the mark penalty for each problem.

2. Late Submission Penalty

10% reduction for the 1st day, then 30% reduction.