University of Waterloo: Final Examination **Winter Term 2016**

Student Name: ______________________________

Student Signature: ______________________________

Student Identification Number: ______________________________

E&CE 356	Database Systems
Section: 001	Instructor: Paul Ward
Friday, April 22nd 2016	9:00 AM to 11:30 AM
Duration of Exam:	2.5 hours
Number of Exam Pages (including cover sheet):	10 pages / 6 questions
Exam Type:	Closed Book

General Notes

1. No electronic devices including no calculators.
2. Read the entire questions *carefully* before answering.
3. State any assumptions clearly and whenever needed.
4. No questions permitted. When in doubt, make an assumption, write it down.
5. Some SQL reminders are listed on Page 10.
6. *aquila non captat muscas*

Question	**Maximum**	**Score**				**Total**
1	12	(a)	(b)	(c)		
2	18	(a)	(b)	(c)		
3	25	(a)	(b)	(c)	(d)	
4	15	(a)	(b)	(c)	(d)	
5	25	(a)		(b)		
6	25	(a)		(b)		
Total	120					

1. [12 marks] SQL Comprehension: Consider the following tables:

Student		
ID	Name	Average
123	Bart	93
456	Lisa	78
789	Maggie	72
101	Homer	82

Takes		
StudentID	CourseID	Grade
123	ECE356	78
123	ECE358	85
456	ECE356	75
789	ECE358	68

Courses		
ID	Name	Cap
ECE356	Database Systems	84
ECE358	Computer Networks	82

The table "Student" gives basic information about students, including their average over all courses taken (Average). The "Courses" table lists basic information about courses, including the class size limit (Cap). The "Takes" table lists courses students have taken this term, togther with their grade in the course.

(a) What is the following query computing (in one sentence) and what is its output for this dataset:

```
select count(distinct StudentID) from Takes;
```

How many students are taking one or more courses?

count(distinct StudentID)
3

(b) What is the following query computing (in one sentence) and what is its output for this dataset:

```
select Name,avg(Grade) from Student inner join Takes on (StudentID=ID) group by StudentID
                                                        order by avg(Grade) desc;
```

What are the names and average grades of students, ordered from highest grade to lowest grade?

Name	avg(Grade)
Bart	81.5
Lisa	75
Maggie	68

(c) What is the following query computing (in one sentence) and what is its output for this dataset:

```
select Name,CourseID,Grade from Student inner join Takes as T on (ID=StudentID) where not exists
                                               (select * from Takes where Grade > T.Grade);
```

What is/are the names of the student(s) who received the highest grade over all courses

Name	CourseID	Grade
Bart	ECE358	85

2. [18 marks] Query Creation: For this question you will use the same relations as in the previous question, though you should assume the tables contain substantially more data than was present in the previous question.

(a) Using SQL, determine the average grade for each course; your SQL should list the courses by name as well as ID, and should sort them from highest average grade to lowest average grade.

```
select Name, Average from Courses inner join
  (select CourseID,avg(grade) as Average from Takes group by CourseID order by Average desc) as B
    on(CourseID=ID);
```

(b) Using SQL, determine which students have a higher average grade on the courses they took this term than their current average over all courses.

```
select Name, Average as OverallAverage, A as CurrentAverage from Student inner join
  (select StudentID,avg(Grade) as A from Takes group by StudentID) as B on (StudentID=ID);
```

Note that OverallAverage and CurrentAverage are not necessary as part of the query output.

(c) Using SQL, determine which courses have an enrollment that is at least 90% of their enrollment cap (*i.e.* the number of students taking the course is $\geq 0.9*$Cap). You should list such courses by name as well as ID, and sort the list from the higest percentage of Cap enrolled.

```
select Name, ID, Enrollment, Cap from Courses inner join
  (select CourseID,count(distinct StudentID) as Enrollment from Takes group by CourseID) as B
    on (CourseID=ID) where Enrollment > 0.9*Cap;
```

Note that Enrollment and Cap are not required as part of the query output.

3. [25 marks] Indexing, Hashing, and Query Optimization: For this question you will use the same relations as in the previous question. **(a)** What are the primary and foreign keys for the three relations, Student, Takes, and Courses?

Student: Primary Key: ID; no foreign keys
Courses: Primary Key: ID; no foreign keys
Takes: Primary Key StudentID,CourseID; StudentID is Foreign Key to Student(ID); CourseID is Foreign Key to Course(ID)

(b) We wish to determine the enrollment cap for a course given the name of the course; the query is "select Cap from Courses where name = "...";" Assuming only the default indexes created by specifying primary keys are present, what is the evaluation plan going to be for this query? Assuming that the Courses table is stored on disk in contiguous blocks, though not in sequential order, what is the approximate execution time of this query in terms of the size of the table (number of rows, r, and/or number of blocks, b), the time to find a random block on disk, T_s and the time to transfer a block from disk, T_t?

There is no index on Name, so the evaluation plan will be to scan the entire table.

Execution time is $T_s + b * T_t$

(c) If you could add any indexes to this schema, what index or indexes would you add to make this query execute faster, if any? Specify if you would use a hash index or a B+-tree. Using the same parameters as above (r, b, T_s, T_t), what would be the approximate execution time with your proposed index?

Add an index to the "Name" attribute of the Courses relation. A hash index is better because it is constant time, where a B+-tree is O(height of the tree).

Execution plan using the hash index would be to use the index to find the relevant block containing the course (and there is likely only one record since course names are probably unique), and then load that block. Time to execute will be $T_s + T_t$.

If a B+-tree index is used, if the index is on disk it will take $h(T_s + T_t) + (T_s + T_t)$; if the index is in memory, then it will be $T_s + T_t$.

(d) For the query shown in question 1(c), what would be a possible execution plan and its approximate execution time?

The query from 1(c) looks like a join with a correlated subquery, but if we ignore the projection (*i.e.*, the select) in the outer query, what is really going on is a simple select from Takes with a correlated subquery. The result of that is joined with Student to project the Name (together with the courseID and Grade from Takes). We know this from question 1(c). Therefore, we effectively have a select over the Takes, with a test, and the results of that are joined with Student. Therefore, read the blocks of Takes, one block at a time, and compare them to the complete set of blocks from Takes. If we can store all of Takes in memory, then this requires the time to read Takes into memory ($T_s + b_{takes}T_t$) and then iterate over that; this can then be joined with Student using an index lookup. If the index is a hash index or in memory, that will be constant additional time.

4. [15 marks] Transactions, Concurrency Control, and Recovery:

Consider the following transactions, where the table on the left shows the specific transactions, while the table on the right shows just the read and write operations required by each transaction.

T_1	T_2	T_3
Begin	Begin	Begin
x = x + 1	y = z * 2	x = z * y
z = y * x	w = x	Commit
Commit	Commit	

which then are

T_1	T_2	T_3
Begin$_1$	Begin$_2$	Begin$_3$
$R_1(x)$	$R_2(z)$	$R_3(z)$
$W_1(x)$	$W_2(y)$	$R_3(y)$
$R_1(y)$	$R_2(x)$	$W_3(x)$
$W_1(z)$	$W_2(w)$	Commit$_3$
Commit$_1$	Commit$_2$	

Consider the schedule:

$Start(T_1), R_1(x), Start(T_2), R_2(x), W_1(x), R_2(z), W_2(y), W_2(w), Commit(T_2),$
$Start(T_3), R_3(y), R_1(y), W_1(z), Commit(T_1), R_3(z), W_3(x), Commit(T_3)$

(a) Assuming that every operation in the schedule is written to the recovery log on disk immediately prior to its execution, and assuming that the initial values are $x = 6; y = 2; z = 7$, what is the contents of the recovery log after this schedule is executed?

```
1:                       10:
2:                       11:
3:                       12:
4:                       13:
5:                       14:
6:                       15:
7:                       16:
8:                       17:
9:                       18:
```

(b) The system crashes immediately after executing "$W_3(x)$." What is added to the recovery log as a result of executing the recovery algorithm as described in class and in the lecture notes?

(c) Is the schedule, as shown, a recoverable schedule? If so, why; if not, explain what the problem is.

(d) Is the schedule, as shown, cascadeless? Explain.

5. [25 marks] Database Design: You are required to design a database schema for a multi-vendor loyalty-card program (*e.g.*, for AirMiles). Customers will be offered a loyalty card that contains a unique ID. For each purchase where the customer uses the loyalty card, the customer will earn 1 FrequentPurchaseDollar for every $100 that the customer spent. To encourage membership in the program, customers can use their earned FrequentPurchaseDollars to purchase products from any of the vendors. Vendors who wish to be part of the program will have their check-out terminals modified so as to include a scan of the loyalty-card ID. The card ID and the associated purchase information (location, time, day, items purchased (identified by some vendor-specific product ID), including quantity and price) are sent to the loyalty-card database.

(a) Create an entity-relationship model for this loyalty-card program. Your diagram may follow any of the notational conventions described in the lecture notes, in class, or the course textbook, but should be consistent in its usage. Include all relevant entities, attributes and relationships, cardinality constraints, and participation constraints. Indicate primary keys of entities by underlining them.

(b) Translate your ER model into a relational model. You do not need to give specific **create table** commands. However, you should identify the following constraints: primary keys, foreign keys, any attributes which must be unique in a relation, and any attributes which may be NULL. You should also identify any plausible functional dependencies between attributes and ensure that your relational design is normalized or you have justified where you have left it unnormalized.

6. [25 marks] Data Analysis: Consider the credit-card data classification-tree example we covered in class. In that dataset we had four attributes: age, sex, income, and credit rating. We showed how to compute the impurity of the unsplit set and then of a set split for over sex (M/F) and then over income.

(a) When computing impurity over a split on income, we needed to decide on a set of possible splits. What we chose ($< 25k, < 50k, < 75k, \geq 75k$) was not necessarily the ideal choice; we could have chosen both different values for the split points (*e.g.*, $< 30k, < 60k, < 90k, \geq 90k$) or a different number of split points (*e.g.*, $< 40k, < 80k, \geq 80k$). There are two broad ways we might have addressed this issue: either by considering different alternatives, computing their impurity implications, and selecting the preferred method, or by using some clustering techniques to try to pick preferred split points. Describe in detail (with reference to specific algorithms discussed in class) how to address this issue.

(b) An alternative to building a classification tree over the credit-card data would be to try to determine if there are assocation rules that can be extracted from it that are of the form “some set of characteristics” $\rightarrow$ “credit rating” Describe how you would modify the *a priori* algorithm to achive this. Identify limitations in this approach, especially with respect to the classification-tree for this particular problem.

Some SQL Reminders

DDL:

```
create table
drop table [if exists <table>]
alter table <table> add <attribute> <domain>
alter table <table> drop <attribute>
create view <view> as <query expression>

Domain Types: char(n), varchar(n), int, smallint, numeric(p,d)
              real, double precision, float(n), date, time

Integrity     not null, default <V>, primary key (<a1, ..., an>),
Constraints:  unique(<a1, ..., an>),
              foreign key (<am, ..., an>) references (<am', ..., an'>)

Referential   on update <...>, on delete <...>
Actions:      cascade, set null, set default, no action

Check Constraints: check (<predicate>)
```

DML:

```
insert into <table> values (...)
update <table> set <attribute = ...> where <predicate>
delete from <table> where <predicate>
select <a1,...,an> from <t1,...,tn> where <predicate>

Options on selection attributes: distinct, as

Options on selection tables: natural join, inner join, outer join
                             <join type> on <predicate> using <attributes>

Set Operations: union, interset, except (use 'all' to retain duplicates)

Aggregate Functions: avg, min, max, sum, count

Grouping: group by, having, order by

Options on predicate: =, !=, <, >, <=, >=, in, not in, exists, not exists,
                      is null, is not null, between, like <string pattern>
```

Other Random Reminders

Gini Impurity(S) = $1 - \Sigma_i p_i^2$

Entropy Impurity(S) = $-\Sigma_i p_i log_2(p_i)$