Assignment #1

CS 348 - Winter 2025

Due : 11:59 p.m., Fri, Jan 31, 2025

Appeal Deadline: One week after being returned

Submission Instruction

This assignment consists of 2 parts. Part I will be submitted through Crowdmark. See the website for more detailed instructions. In particular, do not forget to submit one file per question to make the lives of TAs easier. Part 2 is a programming question and requires submission of files through Marmoset.
For Part 1: You don not have to type the questions if you are using Latex. Just specifying the question number is good enough. You do not need to keep the same font style. Here is a draft/empty latex template you may download and use: `https://www.overleaf.com/read/pxmgghkqksvg#39ea01`. Handwritten solutions and scanned to pdf are also acceptable.

Part I

This part consists of 3 questions covering topics on general dbms concepts, relational model, relational algebra, and SQL.
*Important Note on SQL Questions: Unless otherwise specified in a particular question, you can only use the SQL clauses that appear in the *lectures* (including slides). For example, do not use clauses like COALESCE or IFNULL. Our goal is *not* to make the lives of students who are experienced in SQL harder. Limiting to what we covered in lectures forces you to stay closer to the core relational algebraic way of programming, which is a learning goal of this assignment. Do not worry that using a clause that has been used infrequently in the lectures might lead to TAs taking off marks unfairly and do not double check with the teaching staff whether you can use a clause or not. The rule is simple: if a clause was used in the lectures (including slides), you can. Otherwise you can't.*

Question 1.

[24 marks in total]

(a) (6 marks) Explain each of the following terms using at most four sentences.

1. (**2 marks**) Data model

   Possible solution : A high-level/logical formal notation that describes three aspects of a database: (a) structure of the data; (b) constraints; and (c) operations on the data.

2. (**2 marks**) Data-definition language (DDL)

   Possible solution : A DDL is a language for specifying the database schema, e.g., the attributes of relations and the types of these attributes.

3. (**2 marks**) Candidate key

   Possible solution : A minimal set of fields that uniquely identifies a tuple is called a *candidate key* for the relation.

(b) (2 marks) Give an example of how concurrent requests can cause data inconsistency (example should differ from but can be similar to the book example used in Lecture 1).

Possible solution : (2 points for the example)

- Any example where 2 concurrent requests read the state of data and modify it incorrectly. For example, 2 requests read that one ticket is left, both buy a ticket by writing ticket = ticket-1 back, leading the tickets left value to be -1.

(c) (2 marks) Explain what is a foreign key and what is a foreign key constraint in a relational database.

Possible solution : A foreign key is one or more columns in one table whose value(s) correspond to the primary key of another table. Foreign key constraints enforce referential integrity, which essentially says that if column value A refers to column value B, then column value B must exist.

(d) (2 marks) Answer the following question as true or false and provide a brief justification for your answer: Every relational algebra expression that contains set difference is non-monotone? (Note relational algebrea expression is non-monotone as long as it is non-monotone on one of its base relations.) Ans: "**T**" or "**F**"

Possible solution : False A counter example is R - (R-S). As we showed in the class, this is equivalent to set intersection of R and S, which is monotone on both R and S.

(e) (4 marks) Give an example of when one would use a foreign key constraint and not a tuple-based CHECK. Give an example of when one would use a tuple-based CHECK and not a foreign key constraint. Think of something a user can instruct the DBMS to do using a foreign key constraint but not a tuple-based check and vice versa.

Possible solution : (2 points for each example)
(1) You can cascade foreign key constraint violations by deleting tuples of the referencing table or setting foreign key constraints. This form of "triggered" actions is integrated directly into foreign key constraints.

(2) Foreign key constraints require that the referenced column of the referenced table P is the primary key of P. Whereas, tuple-based CHECKs are more general and one can e.g., implement a referential integrity constraint where a column C of table T references another table P's column A, where A is not the primary key. However, the CHECK assertion will not be checked again when P is modified in a way that may violate the referential integrity.

(f) (4 marks) Consider a database that consists of 3 tables `Customer(cID, cname)`, `Product(pID, pname)`, `Order(oID, cID, pID)`. The ID columns are integer and name columns are string. The `cID` and `pID` columns of `Order` store, respectively, the `cID` of the customer $c \in$ `Customer`, who made the order and the `pID` of the product $p \in$ `Product`, that c ordered. Assume the ID column of each table is the primary key. Consider the question: Who are the pairs of co-purchaser customers, where two customers c_i and c_j are co-purchasers if $c_i \neq c_j$ and they have both ordered at least one common product p. We are looking for an output table with two columns (e.g., $CID1$ and $CID2$). Output each $CID1$ and $CID2$ once as ($CID1$, $CID2$) or ($CID2$ and $CID1$) but not both. Write this query in relational algebra.

Possible solution :

$$\Pi_{C1.cID,C2.cID}(\\ \sigma_{C1.cID!=C2.cID,c1.cID<c2.cID}(\\ \rho_{C1}(Customer) \bowtie_{C1.cID=O.cID} \rho_{O1}(Order) \bowtie_{O1.pID=P.pID} \bowtie\\ Product \bowtie_{P.pID=O2.pID} \rho_{O2}(Order) \bowtie_{O2.cID=C2.cID} \rho_{C2}(Customer)))$$

(g) (4 marks) Let the division operator between two relations $R(A_1, ..., A_{k_1}, B_1, ..., B_{k_2})$ and $S(B_1, ..., B_{k_2})$ be defined as follows. Let $T(A_1, ..., A_{k_1}) = R \div S$ be the result of the division operator. T contains exactly the tuples t (over $A_1, ..., A_{k_1}$) such that for every tuple $s \in S$, $t \bullet s \in R$, where $\bullet$ concatenates t and s. Express the division operator as an expression that consists of core and/or derived relational algebra operators from Lectures 2 and early 3.
Possible solution : Let $A = A_1, ..., A_{k_1}$.
$T = R \div S = \Pi_A(R) - \Pi_A((\Pi_A(R) \times S) - R)$.
Let us consider the expression $(\Pi_A(R) \times S)$ and call this H. H finds all possible tuples if we took the Cartesian product of *every A projection of each tuple $t \in R$ with S*. This is a set that contains R. Let t_p be the projection of a tuple $t \in R$ onto A. If t_p satisfies the condition that it "matches" with every s in S, then if we take the set difference of H and R and project onto A, t_p will not appear in the result. Otherwise t_p will appear in the result. So t_p will not appear in the expression: $G = \Pi_{A_1,...,A_{k-1}}((\Pi_{A_1,...,A_{k-1}}(R) \times S) - R)$ if and only if it satisfies the division condition. Therefore, if we finallly take the set difference of $\Pi_A(R) - G$, then t will appear in the result.

Question 2.

[10 marks in total] Consider the following database schema:

`Product` (<u>`maker`, `model`, `type`</u>)
`PC` (<u>`model`</u>, `speed`, `ram`, `hd`, `price`)
`Laptop` (<u>`model`</u>, `speed`, `ram`, `hd`, `screen`, `price`)
`Printer` (<u>`model`</u>, `color`, `type`, `price`)

The `Product` relation gives the manufacturer (or maker), model number, and type(PC, laptop, or printer) of various products. The `PC` relation gives for each model number that is a PC the speed (of the processor, in gigahertz), the amount of RAM (in megabytes), the size of the hard disk (in gigabytes), and the price. The `Laptop` relation is similar, except that the screen size (in inches) is also included. The `Printer` relation records for each printer model whether the printer produces color output (true, if so), the process type (laser or ink-jet, typically), and the price. The primary keys designated for these four relations are underlined in the schema above. In addition, suppose that there are a set of application-level constraints that are defined on the database. We omit these from the schema above but assume that they hold (in fact think of how to express these as relational algebra expressions as an exercise). The constraints are the following. The models for all the pc products in the `Product` relation must be contained in the `PC` relation. Similarly, the models for all the laptop products in the `Product` must be contained in the `Laptop` relation, and the models for all the printer products in the `Product` must be contained in the `Printer` relation. The following is a sample instance:

`Product`

maker	model	type
A	001	*pc*
B	003	*printer*
D	001	*laptop*
D	001	*printer*

`PC`

model	speed	ram	hd	price
001	2.66	1024	250	2114
002	2.10	512	250	995
003	1.40	512	80	478
004	2.80	1024	250	649

`Laptop`

model	speed	ram	hd	screen	price
001	2.00	2048	240	20.1	3673
007	1.73	1024	80	17.0	949
003	1.80	512	60	15.4	549
006	2.00	512	60	13.3	1150

`Printer`

model	color	type	price
001	*true*	*inkjet*	99
003	*false*	*laser*	209
004	*true*	*laser*	391
006	*true*	*dry*	129

Based on the given schema and instance, for each of the following questions, please **circle** (or **state**) "**T**" (for "true") or "**F**" (for "false") and **explain your choice** using no more than 4 sentences.

(a) (2 marks) {`maker`, `type`} is a candidate key for the `Product` relation. Ans: "**T**" or "**F**"

Hint: The database on page 3 is just one possible instance. Your decision should not simply base on this instance.

Possible solution : False.
{`maker`, `model`, `type`} is a key to `Product` and it is minimal. None of its proper subset can be a key.

(b) (2 marks) {`model`, `speed`} is a superkey for the `Laptop` relation? Ans: "**T**" or "**F**"

Possible solution : True.
{`model`} is the primary key for the `Laptop` relation. Any superset of a candidate key is a superkey.

(c) (2 marks) We cannot insert a new row with value ("A, 001, pc") to the `Product` table instance shown on the previous page?

Ans: "**T**" or "**F**"

Possible solution : True.
{`maker`, `model`, `type`} is designated as a primary key in the `Product` relation. This key value already exists in the instance, and hence this new row will not be accepted.

(d) (2 marks) If the `PC` table only has 4 models listed in the instance on the previous page (001, 002, 003, 004), we can still insert a new row with value ("A, 005, pc") to the Product table. Ans: "**T**" or "**F**"

Possible solution : False.
This is because the models for all the pc products in the `Product` must be contained in the `PC` relation (an integrity constraint at database level).

(e) (2 marks) The `Printer` relation has 8 superkeys. Ans: "**T**" or "**F**"

Possible solution : True.
The only candidate key is `model` in `Printer`. Adding any of the remaining 3 attributes to `model` will form new superkeys.

Question 3.

[22 marks in total] Using the database schema and sample instance from Question 2, which shows the `Product`, `PC`, `Laptop`, `Printer` relations, write the following queries in relational algebra.

Notes: (i) Only use the relational algebra operators used in lectures (including lecture slides), shich include selection, projection, Cartesian product, union, difference, renaming, theta join, natural join, intersection. (ii) The answers can be expressed either as a tree of relational algebra operators or as single lines of algebraic expressions.

(a) (3 marks) Which manufacturers make laptops that have a speed of at least 2.3;

Possible solution : $\pi_{maker}(\sigma_{type='laptop'}(Product) \bowtie \sigma_{speed \geq 2.3}(Laptop))$

(b) (4 marks) Find the model number, type, and price of all products (of any type) made by manufacturer D;

Possible solution : $\pi_{model,type,price}(\sigma_{maker='D'}($
$((\sigma_{type=`pc'}(Product) \bowtie \pi_{model,price}(PC))\cup$
$(\sigma_{type=`laptop'}(Product) \bowtie \pi_{model,price}(Laptop)))\cup$
$(\sigma_{type=`printer'}(Product) \bowtie \pi_{model,price}(Printer)))$

(c) (4 marks) Find those pairs of PC models that have both the same speed and RAM (A pair should be listed only once; e.g. list(i,j) but not (j,i);

Possible solution : $\pi_{o1,o2}($
$\rho_{PC1(model \rightarrow o1, speed \rightarrow s1, ram \rightarrow r1)}(PC)$
$\bowtie_{s1=s2, r1=r2, o1>o2}$
$\rho_{PC2(model \rightarrow o2, speed \rightarrow s2, ram \rightarrow r2)}(PC))$

(d) (5 marks) Find the manufacturers that make the cheapest printers.

Possible solution :

(i) $\rho_{L1(model1 \rightarrow o1, price \rightarrow p1, \ldots)}(Printer)$

(ii) $\rho_{L2(model1 \rightarrow o2, price \rightarrow p2, \ldots)}(Printer)$

(iii) $\pi_{maker}(\sigma_{type='printer'}(Product) \bowtie_{model=o1} (\pi_{o1}(L1) - \pi_{o1}(L1 \bowtie_{p1>p2} L2))))$

(e) (6 marks) Find printer models that are sold by *exactly* two manufacturers.

Possible solution :

(i) $\rho_{P1(maker \rightarrow m1, model \rightarrow o1)}(\pi_{maker,model}(\sigma_{type='printer'}(Product)))$

(ii) $\rho_{P2(maker \rightarrow m2, model \rightarrow o2)}(\pi_{maker,model}(\sigma_{type='printer'}(Product)))$

(iii) $\rho_{P3(maker \rightarrow m3, model \rightarrow o3)}(\pi_{maker,model}(\sigma_{type='printer'}(Product)))$

(iv) $\rho_{PP}((P1 \bowtie_{(m1 \neq m2) \wedge (o1=o2)} P2))$

(v) $\pi_{o1}(PP) - \pi_{o1}(PP \bowtie_{(o1=o3) \wedge (m3 \neq m1) \wedge (m3 \neq m2)} P3)$

Part II.

[21 marks]

Students will work with DuckDB, which is an embedded SQL system that is state-of-the-art in its query speed. You can install and run DuckDB on your local machines easily, which is what we encourage you to do. We will also provide instructions to use a school machine though this should be less convenient. The address of the school machines is linux.student.cs.uwaterloo.ca - you can remotely connect to these machines via ssh. DuckDB is already installed on these machines and you can access it via the `duckdb` command. If you are connecting from off campus you will need to have already setup ssh keys on the server. You can find detailed information about these servers here: `https://uwaterloo.ca/computer-science-computing-facility/teaching-hosts`. It is important to note that we cannot guarantee the uptime of the school's machines so we encourage you to install DuckDB on your own machine.

Get Started

You should first install DuckDB on your local machine (if you are using the school's machine you can skip this step). Installing DuckDB is simple. The installation instructions can be found here: `https://duckdb.org/docs/installation/`. You can use DuckDB through several interfaces, such as its command line interface (CLI), or Python or C++ bindings. You can use any interface as you work on the assignment. In the end, for each part of this question, you will need to provide us a SQL query in a .sql file. So regardless of which interface you pick to work on, your submission will be a set of .sql files. Below, we will describe how to use DuckDB's CLI.

Install DuckDB CLI: To get started on your own machines, you need to install the CLI. Choose your specific operating system for the correct installation command for the CLI and then run the provided command to install (e.g., `brew install duckdb` if you are on a MacOS).

Once DuckDB is installed on your system you should be able to access the `duckdb` command and the associated DuckDB CLI. Some overview information about the CLI can be found here: `https://duckdb.org/docs/api/cli/overview`. DuckDB runs in two separate modes:

- In-memory mode: If on your system's CLI, you just type `$duckdb`, DuckDB CLI starts in in-memory mode. That is, any tables you create will be temporary and be lost when you close the CLI (with the `$.quit` command in DuckDB's CLI.)
- Persistent mode: You can also point the DuckDB CLI to a file on a local disk, in which case DuckDB will persist your database in that file. For example, you can type `$duckdb yourDbFile` and created some tables and stored tuples in them. Then DuckDB will store your records in the `yourDbFile`. Later on if you close your DuckDB CLI and open it again by pointing to `yourDbFile`, you will find the latest state of your database.

You can open and start issuing queries within the DuckDB CLI. You can also run SQL scripts (.sql files) using the `duckdb` command from the system's shell. This might be handy to check that the SQL query you put into your submission .sql files work as expected. Suppose we want to run the SQL query in the file `question1.sql`. To do this we just need to direct it to DuckDB by running the following: `$duckdb yourDbFile < question1.sql`.

Run Test Database

We have provided some starter files for this assignment. These are in the `A1/StarterDatabase.zip` file on Learn. Download and unzip this file and you can find the files under the `Database/testdb/` folder. You can use these files to get started. You should review the schema in `createTables.sql` The starter files include some sample data to create a test database. To setup this test database simply run the following:
`$duckdb testDb < createTables.sql` and
`$duckdb testDb < populateTables.sql`.
We have also added a shell script for quickly running both of the above commands on Linux. Once you have created the tables and populated them with data you can start writing your queries and testing them on the sample data.
NOTE: This data is arbitrary. It is NOT guaranteed that the sample database is sufficient data to test all requirements of the queries. It is up to you to test your queries by creating different instances of the database.

Queries

In this assignment, you are required to submit 6 SQL files. You can find starter files in `Database/testdb/queries/`: `question1.sql, question2.sql, ..., question6.sql`.

Each file contains an incomplete SQL statement. Complete these SQL statements based on the problems below.

IMPORTANT:
Unless otherwise specified each file should contain only a single query.
You should NOT create views or new tables to answer any of the SQL questions.
You should NOT update the database except in question 6 where you are specifically asked to delete tuples.

Submission

You need to submit your assignment via `https://marmoset.student.cs.uwaterloo.ca/`.

Marmoset will automatically test and mark your code. **IMPORTANT: Your submission must be a single zipped folder which includes only the 6 SQL files `question1.sql, question2.sql, ..., question6.sql`**. This means that when you unzip the folder you should see the 6 SQL files. The files must be named exactly `question1.sql, question2.sql, ..., question6.sql`.

Each question has only 1 public test and many private tests. You will only see the results of the public test (whether or not you passed or failed the public test). **IMPORTANT: the public tests are extremely simple - they are intended only for you to check that your submission went through correctly and that you do not have any obvious problems. Passing a public test does not guarantee that you pass the private tests.**
To receive full marks your SQL must pass all of the tests. Therefore, it is important that you test your SQL yourself.

To pass a test the tuples returned by your query must match the expected output. The number of columns returned by your query must match what the question asks for however, the column names and column order

do not matter so feel free to use aliases if you like. The row/tuple order however matters if the question asks for a specific order (otherwise it does not matter).
You can re-submit until the deadline and your best submission will be used for the final grade. Do not be alarmed if marmoset is slow, marmoset can often take a rather long time to run and grade your submission.

All of the SQL files that you submit must work in DuckDB and they must NOT produce any errors/warning. Queries that produce errors/warnings will automatically fail the test cases.

question1.sql

[4 marks in total] Print the average age of students enrolled in classes taught by faculty members that have taught at least 3 courses.
Possible solution :

```
with profs(fid) as (
    select fid
    from class
    group by fid
    having count(name) >= 3
),
taughtStudents(snum) as (
    select distinct snum
    from profs, enrolled, class
    where enrolled.cname = class.name and class.fid = profs.fid
)
select SUM(age) / count(taughtStudents.snum) f
rom taughtStudents, student
where taughtStudents.snum = student.snum;
```

question2.sql

[3 marks in total] Print the list of names of faculty members that do not teach any classes. Sort the list of names in ascending order. Note that it is possible for two faculty members with different fid's to have the same fname. For example, if two different faculty members share the same name 'Aisha Miller' and neither of them teach any classes, then 'Aisha Miller' should appear twice in the output. Other than that case, the name of the same person should not appear more than once in the output.
Possible solution :

```
SELECT fname
FROM faculty
WHERE fid IN (
        SELECT fid
        FROM faculty
        EXCEPT
        SELECT fid
        FROM class);
```

question3.sql

[3 marks in total] Print the list of names of students who take the class 'Computer Networks', and the class 'Algorithms'. The output list should be sorted by the student name in ascending order. Note that it is possible for two students with different snum's to have the same sname. For example, if two different students share the same name 'Lisa Walker', then 'Lisa Walker' should appear twice in the output. Other than that case, the name of the same student should not appear more than once in the output.

Possible solution :

```
SELECT sname
FROM student
WHERE snum IN (SELECT e1.snum
               FROM enrolled e1, enrolled e2
               WHERE e1.snum=e2.snum AND
                       e1.cname = 'Computer Networks' AND
                       e2.cname = 'Algorithms')
ORDER BY sname ASC;
```

question4.sql

[4 marks in total] Print the list of class names from the class relation, together with the name of the corresponding lecturer and class sizes. The output list should be sorted according to the class size in descending order. If there is a class in class relation with no one enrolled, then that class should appear in the output as well, with a count of 0.

Possible solution :

```
SELECT cs.name, f.fname, cs.size
FROM faculty f,
     (SELECT c.name AS name, c.fid AS fid, COUNT(e.snum) AS size
      FROM class c
           LEFT OUTER JOIN enrolled e
           ON c.name  = e.cname
           GROUP BY c.name, c.fid) AS cs
WHERE f.fid = cs.fid
ORDER BY cs.size DESC;


(not correct)
SELECT c.name, f.fname, cs.csize
FROM class c, faculty f,
        (SELECT cname AS cname, COUNT(e.snum) AS csize
        FROM enrolled
        GROUP BY cname) AS cs
WHERE c.name = cs.cname AND c.fid = f.fid
ORDER BY cs.csize DESC;
```

question5.sql

[4 marks in total] Print the number of students who are enrolled in exactly one class of size <4. (The students considered in the output can take classes of size ≥ 4, but cannot take two or more than two classes of size <4).

Possible solution :

```
SELECT COUNT(*) FROM
        (SELECT e2.snum
        FROM enrolled e2
        WHERE e2.cname IN
                (SELECT e.cname
                        FROM enrolled e
                        GROUP BY e.cname
                        HAVING COUNT(e.snum) <4)
        GROUP BY e2.snum
        HAVING COUNT(e2.cname)=1);
```

question6.sql

[3 marks in total] Delete classes that are enrolled by fewer than 3 students. (You may use more than 1 query for this part.)

Possible solution :

```
DELETE FROM enrolled
WHERE cname IN (
        SELECT cname
        FROM enrolled
        GROUP BY cname
        HAVING COUNT(snum)<3);

DELETE FROM class
WHERE name IN (
        SELECT name FROM class
        EXCEPT
        SELECT cname FROM enrolled);
```