CSC 411 DBMS Design

Spring 2021

Final Examination (Take-Home)

Release Time: April 13 12:30 PM
Due Time: April 24 11:59 PM
Total points: 100

NAME Jackie Diep

STUDENT ID# W10076331

1. Intermedia SQL (Chapter 4) [20 points]

1.1 [7 points] Rewrite the query

select *

from *section* **natural join** *classroom*

without using a **natural join** but instead using an inner join with a using condition.

SELECT *

FROM section INNER JOIN classroom

USING (column_name);

1.2 [13 points] For the following Employee database, write a query to find the ID of each employee with no manager. Note that an employee may simply have no manager listed or may have a *null* manager. Write your query using an outer join and then write it again using no outer join at all.

---

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

SELECT ID

FROM employee NATURAL LEFT OUTER JOIN manages

WHERE manager_id IS NULL;

2. JDBC (Chapter 5) [12 points]

Consider the following Java code which uses the JDBC API. Assume that the userid, password, machine name, etc. are all okay. Describe in concise English what the Java program does. (That is, produce an English sentence like "It finds the manager of the toy department," not a line-by-line description of what each Java statement does.)

static void testFunc(String r)

```
{
    try
    {
        Class.forName("oraclejdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@db.yale.edu:2000:univdb",user,passwd);
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.ExecuteQuery(r);
        ResultSetMetaData rsmd = rs.getMetaData();
        int count = rsmd.getColumnCount();
        System.out.println("<tr>");
        for(int i=1;i<=count;i++){
            System.out.println("<td>"+rsmd.getColumnName(i)+"</td>");
        }
        System.out.println("</tr>");
        while(rs.next(){
            System.out.println("<tr>");
            for(int i=1;i<=count;i++){
                System.out.println("<td>"+rs.getString(i)+"</td>");
            }
            System.out.println("</tr>");
        }
        stmt.close();
        conn.close();
    }
    catch(SQLException sqle)
    {
        System.out.println("SQLException : " + sqle);
    }
}
```

The code prints out the column names and results of the data set returned by "String r" in a readable html table.

3. Relation Schema Optimization (Chapter 7) [24 points]

3.1 [6 points] Although the BCNF algorithm ensures that the resulting decomposition is lossless, it is possible to have a schema and a decomposition that was not generated by the algorithm, that is in BCNF, and is not lossless. Give an example of such a schema and its decomposition.

employee(ID, name, salary)

With dependencies
ID → name
ID → salary

Decomposed into
→ emp_1(ID, name)
→ emp_2(name, salary)

The original relation is in BCNF as there are no dependencies other than where the form X→Y does not have X as a superkey of the relation.

As there are only two attributes in each decomposed schema, they are also both in BCNF, but in the case that two employees with the same name were employed, a natural join would cause incorrect results and is a lossy decomposition.

3.2 [4 points] Show that every schema consisting of exactly two attributes must be in BCNF regardless of the given set *F* of functional dependencies.

To be in BCNF, every functional dependency in the relation is in the form of a → b, when a is a super key of the relation.

There are four cases of possible dependency in a two attribute schema.

1) **a → b, then it is in the proper form, and a is a super key of the relation because both b and a can be derived from it.**
2) **b → a, in this case it is in the proper form, and b is a super key of the relation because both a and b can be derived from it.**
3) **In the case that there is no dependency between the two, then there is no relation to violate the BCNF dependency rule**
4) **If both a → b and b → a, then they are both in the proper form, and they are both super keys of the relation.**

As no possible dependencies violate the BCNF rule, all two attribute relations must be in BCNF.

3.1 [14 points] Consider the following set F of functional dependencies on the relation schema (A, B, C, D, E, G):

$A \rightarrow BCD$
$BC \rightarrow DE$
$B \rightarrow D$
$D \rightarrow A$

Give a BCNF decomposition of the given schema using the original set F of functional dependencies.

First → {(A, B, C, D), (A, E, G)}

Finally → {(A, B, C, D), (A, E), (A, G)}

4. Big Data (Chapter 10) [28 points]

4.1 [4 points] Give four ways in which information in web logs pertaining to the web pages visited by a user can be used by the web site.

Deciding what types of information (such as news or social media posts) to recommend to users to keep them interested based on previously viewed pages or products.

Deciding how the navigation of a web site should be structured to easily lead to the most likely pages of information they are looking for based on what pages are typically viewed in sequence by users.

Selling information to manufacturers or vendors on what items to produce or stock more of depending on increased trends of user preferences and page views.

Deciding what advertisement offers to accept from companies or display to users based on click-through and conversion rates on the site as key metrics.

4.2 [4 points] One of the characteristics of Big Data is the variety of data. Explain why this characteristic has resulted in the need for languages other than SQL for processing Big Data.

Big Data requires the ability to scale to large volumes and velocities of data while also having parallel storage and data processing in its database. Combining these things with an SQL supporting database and other necessary features like transactions can be very difficult, which creates a need for other types of languages.

More specifically, there are three types of information that Big Data ends up using

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

As SQL is only able to not able to fully handle Semi-structured information and almost not at all with Unstructured information, NoSQL languages became a necessity.

4.3 [20 points] The map-reduce framework is quite useful for creating inverted indices on a set of documents. An inverted index stores for each word a list of all document IDs that it appears in (offsets in the documents are also normally stored, but we shall ignore them in this question).

For example, if the input document IDs and contents are as follows:

1: data clean
2: data base
3: clean base

then the inverted lists would

data: 1, 2
clean: 1, 3
base: 2, 3

Give pseudocode for `Map` and reduce functions to create inverted indices on a given set of files (each file is a document). Assume the document ID is available using a function `context.getDocumentID()`, and the `Map` function is invoked once per line of the document. The output inverted list for each word should be a list of document IDs separated by commas. The document IDs are normally sorted, but for the purpose of this question you do not need to bother to sort them.

```
Map (String docID, String record) {
        String key = docID;
        for each word in record:
                // emission will be word: docID
                emit(word + ":", key);
}

Reduce (String word, List docID_list) {
        String key = word;
        // "empty" just for rudimentary formatting purposes
        String foundIn = "empty";
        for each docID in docID_list:
                // if structure just for formatting purposes (comma not added before first
                        element)
                if foundIn = "empty":
                        foundIn = str(docID);
                else:
                        foundIn = foundIn + ", " + str(docID);

        // output will be      word: docID, docID, docID
        output (key + ":", foundIn);
}
```

5. Transactions (Chapter 17) [16 points]

5.1 [6 points] During its execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which a transaction may pass. Explain why each state transition may occur.

The possible states:

1. **Active: initial state, while the transaction is executing it remains in this state**
2. **Partially committed: state of the transaction after the final statement has been executed**
3. **Failed: when normal execution can no longer proceed**
4. **Aborted: state of the transaction after rollback and the database is restored to its prior state**
5. **Committed: state of the transaction after successful completion**

- **Also, a transaction is considered terminated after it has been aborted or committed**

The three possible sequences of states:

1. **Active → Partially committed → Committed**

In this case, the statement starts executing, the final statement is executed, the changes are committed, and the database has a new consistent state.

2. **Active → Partially committed → Failed → Aborted**

In this case, the statement starts executing, the final statement is executed, because of a hardware or logistical error the statement enters the failed state, the transaction is aborted, and the database is returned to its prior consistent state.

3. **Active → Failed → Aborted**

In this case, the statement starts executing, a hardware or logistical error occurs before the final statement is executed and the statement enters the failed state, the transaction is aborted, and the database is returned to its prior consistent state.

5.2 [10 points] Consider a database with objects X and Y and assume that there are two transactions T_1 and T_2. Transactions T_1 reads objects X and Y and then writes object X. Transactions T_2 reads objects X and Y and then writes objects X and Y.

T_1: R(X) → R(Y) → W(X) → commit

T_2: R(X) → R(Y) → W(X) → W(Y) → commit

(a) [5 points] Give an example schedule with actions of transactions T_1 and T_2 on objects X and Y that results in a write-read conflict.

Write-read = reading uncommitted data

T_1	T_2
R(X)	
R(Y)	
W(X)	
	R(X)
	R(Y)
commit	
	W(X)
	W(Y)
	commit

In this schedule, T_2 reads X after T_1 writes to X. The commit has not taken place yet, so this is a write-read conflict.

(b) [5 points] Give an example schedule with actions of transactions T_1 and T_2 on objects X and Y that results in a write-write conflict.

Write-write = overwriting data from another transaction before commit

T_1	T_2
R(X)	
R(Y)	
W(X)	
	R(X)
	R(Y)
	W(X)
commit	
	W(Y)
	commit

In this schedule, T_2 writes to X after T_1 writes to X before commit has happened. T_2 is overwriting the write from T_1 causing a write-write conflict.