HW5A-Written

May 15, 2025

W4111_2025_002_1: Introduction to Databases:Homework 5A

1 Overview

1.1 Scope

The material in scope for this homework is: - The content of lectures: - All material from lectures 1 to the lecture on 25-April. - This includes any material in the slides, even if not explicitly presented in lecture. - Any information provided or discussed in lectures, even if not in slides. - The slides associated with the recommended textbook for - All material from the textbook slides that were in scope for previous homework assignments. - Chapter 16: 16.1 - 18.28, 16.39-16.53 - Chapter 17 - Chapter 18: 18.1 - 18.17. 18.21 - 18.26, 18.33 - 18.38

1.2 Submission Instructions

Note to DFF: Create necessary links.

- Due date: TBD, 11:59 PM EDT on GradeScope.
- You submit on GradeScope. We will create a GradeScope submission for the homework.
- Your submission is a PDF of this notebook. You must tag the submission with locations in the PDF for each question. You must solve problems you experience producing a PDF including images. Please do not wait until the last minute.

There is a DFF TODO on Ed Discussions that we will use to resolve questions and issues with respect to homework 4A.

1.3 Brevity

Brevity

Students sometimes just write a lot of words hoping to get something right. We will deduct points if your answer is too long.

2 Initialization

2.1 Python Environment

[2]:
```
import copy
```

[3]:
```
import json
```

[4]:
```
import pandas
```

[5]:
```
import numpy
```

[6]:
```
import json
```

[ ]:
```
import csv
```

2.2 MySQL

[7]:
```
# You should have installed the packages for previous homework assignments
#
import pymysql
import sqlalchemy
from sqlalchemy import create_engine
```

[8]:
```
# You have installed and configured ipython-sql for previous assignments.
# https://pypi.org/project/ipython-sql/
#
%load_ext sql
```

[9]:
```
# This is a hack to fix a version problem/incompatibility  with some of the␣
 ↪packages and magics.
#
%config SqlMagic.style = '_DEPRECATED_DEFAULT'
```

[10]:
```
# Make sure that you set these values to the correct values for your␣
 ↪installation and
# configuration of MySQL
#
db_user = "root"
db_password = "rootpass"
```

[11]:
```
# Create the URL for connecting to the database.
# Do not worry about the local_infile=1, I did that for wizard reasons that you␣
 ↪should not have to use.
#
db_url = f"mysql+pymysql://{db_user}:{db_password}@localhost?local_infile=1"
```

```
[12]: # Initialize ipython-sql
      #
      %sql $db_url
```

```
[13]: # Your answer will be different based on the databases that you have created on␣
       ↪your local MySQL instance.
      #
      %sql select * from db_book.student limit 10;
```

```
 * mysql+pymysql://root:***@localhost?local_infile=1
10 rows affected.
```

```
[13]: [('00128', 'Zhang', 'Comp. Sci.', Decimal('102')),
       ('12345', 'Shankar', 'Comp. Sci.', Decimal('32')),
       ('19991', 'Brandt', 'History', Decimal('80')),
       ('23121', 'Chavez', 'Finance', Decimal('110')),
       ('44553', 'Peltier', 'Physics', Decimal('56')),
       ('45678', 'Levy', 'Physics', Decimal('46')),
       ('54321', 'Williams', 'Comp. Sci.', Decimal('54')),
       ('55739', 'Sanchez', 'Music', Decimal('38')),
       ('70557', 'Snow', 'Physics', Decimal('0')),
       ('76543', 'Brown', 'Comp. Sci.', Decimal('58'))]
```

```
[14]: default_engine = create_engine(db_url)
```

```
[15]: result_df = pandas.read_sql(
          "select * from db_book.student limit 10", con=default_engine
      )
      result_df
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[15], line 1
----> 1 result_df = pandas.read_sql(
      2     "select * from db_book.student limit 10", con=default_engine
      3 )
      4 result_df

File ~/anaconda3/lib/python3.11/site-packages/pandas/io/sql.py:590, in␣
 ↪read_sql(sql, con, index_col, coerce_float, params, parse_dates, columns,␣
 ↪chunksize)
    581     return pandas_sql.read_table(
    582         sql,
    583         index_col=index_col,
   (…)
    587         chunksize=chunksize,
    588     )
    589 else:
```

```
--> 590     return pandas_sql.read_query(
    591         sql,
    592         index_col=index_col,
    593         params=params,
    594         coerce_float=coerce_float,
    595         parse_dates=parse_dates,
    596         chunksize=chunksize,
    597     )

File ~/anaconda3/lib/python3.11/site-packages/pandas/io/sql.py:1560, in␣
 ↪SQLDatabase.read_query(self, sql, index_col, coerce_float, parse_dates,␣
 ↪params, chunksize, dtype)
   1512 """
   1513 Read SQL query into a DataFrame.
   1514
   (…)
   1556
   1557 """
   1558 args = _convert_params(sql, params)
-> 1560 result = self.execute(*args)
   1561 columns = result.keys()
   1563 if chunksize is not None:

File ~/anaconda3/lib/python3.11/site-packages/pandas/io/sql.py:1405, in␣
 ↪SQLDatabase.execute(self, *args, **kwargs)
   1403 def execute(self, *args, **kwargs):
   1404     """Simple passthrough to SQLAlchemy connectable"""
-> 1405     return self.connectable.execution_options().execute(*args, **kwargs

AttributeError: 'OptionEngine' object has no attribute 'execute'
```

2.3 MongoDB

[17]:
```
!pip install pymongo
import pymongo
```

```
Collecting pymongo
  Downloading pymongo-4.13.0-cp311-cp311-macosx_11_0_arm64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Downloading pymongo-4.13.0-cp311-cp311-macosx_11_0_arm64.whl (857 kB)
                         857.2/857.2 kB
15.4 MB/s eta 0:00:00
Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.13.0
```

[18]:
```python
mongodb_user_id = "adm2216"
mongodb_password = "passpass"
mongodb_url = f"mongodb+srv://{mongodb_user_id}:{mongodb_password}@cmongodb+srv:
  ↪//adm2216:Ss8DnxZkDPnQfOTe@hw4.n8z9n4b.mongodb.net/"
```

[ ]:
```python
# You can follow the tutorial.
#
from pymongo import MongoClient

# Uncomment the following if you are using Compass
#
import certifi
mongo_client = pymongo.MongoClient(mongodb_url, tlsCAFile=certifi.where())

# Uncomment if you are using local MongoDB
#
# client = MongoClient(mongodb_url)

try:
    # start example code here

    # end example code here

    client.admin.command("ping")
    print("Connected successfully")

    # other application code

    # client.close()

except Exception as e:
    raise Exception(
        "The following error occurred: ", e)
```

3 Written Questions

3.1 Query Processing and Statistics

Question

Query Processing Flow

The preceding figure is an overview of the process and steps in query processing. Give three examples of statistics that the query processor/optimizer might use in optimization. Briefly explain how the query processor/optimizer would use the statistics.

Answer These are the three phases of query processing: 1. Parsing and translation 2. Optimization 3. Evaluation In the optimization phase, the optimizer uses statistics about the data to decide among alternative execution plans. These statistics help estimate costs (I/O, CPU, memory) and choose the most efficient path to execute the query. The optimizer can use any of the following when considering how to optimize: 1. Cardinality - This is important since it helps to estimate the cost of scanning a table. Likewise, it affects decisions like whether to use an index or a full table scan. 2. Selectivity - This relates to the number of rows that satisfy a particular query/"predicate". High selectivity (i.e., few rows returned) → better for using indexes. 3. Index information - the optimizer also considers whether an index exists on a column and if it's a clustered index. If a filter condition involves an indexed column, the optimizer may prefer an index scan over a full scan.

3.2 Equivalent Queries

Question

Consider the following relational algebra expression –

```
 ID, name, dept_name, course_id (  dept_name='Comp. Sci.'(instructor   teaches))
```

Give an equivalent query that shows two optimizations that the query optimizer might apply. Explain the reasons.

Answer ID, name, dept_name, course_id ((ID, name, dept_name (dept_name = 'Comp. Sci.' instructor)) (ID, course_id teaches))

Here we are using both selection pushdown to apply filter before join, which reduces the number of rows entering the join. Similarly, we are doing a projection pushdown, which discards unneeded columns early, reducing memory and I/O costs.

3.3 Query Optimization

Question

Query Expression Tree

The preceding figure shows an example of transforming an expression tree as part of optimization. What additional step is necessary to produce an evaluation plan?

Answer From 16.4 "Choice of evaluation plans" To produce a complete evaluation plan from the transformed expression tree, we need to complete the additional step of choosing specific algorithms for each operator to convert the logical expression tree into a physical evaluation plan. After logical optimization (e.g. applying algebraic transformations), the query is still expressed in high-level operators (like , ,), but they have no clear implementation method. This choice of algorithms — along with the order and access paths — defines the evaluation plan. This includes thinking about how to access (indexes, sequential vs. index scan, etc.), join (nested-loop, sort-merge, etc.).

3.4 Index Selectivity

Question

The follow is the DDL for `db_book.instructor` —

```
create table if not exists db_book.instructor
(
    ID        varchar(5)    not null
        primary key,
    name      varchar(20)   not null,
    dept_name varchar(20)   null,
    salary    decimal(8, 2) null,
    constraint instructor_ibfk_1
        foreign key (dept_name) references db_book.department (dept_name)
            on delete set null,
    check (`salary` > 29000)
);

create index dept_name
    on db_book.instructor (dept_name);
```

Define and briefly the concept of *most selective index.* Why does index selectivity matter?

Write and execute an SQL statement in the code cell below that demonstrates the index selectivity for all indexes on the table, and explain how it shows the index selectivity.

Answer:

Simply put, a most selective index is the index on a column (or combination of columns) where the predicate filters out the largest fraction of rows — i.e., it returns the fewest rows relative to the total number in the table. The higher the selectivity, the more efficient the index becomes in narrowing down the data (selectivity of .01 is better than .1, as it means the index returns fewer rows).

Place and execute the SQL below.

[24]:
```
%%sql
SELECT COUNT(*) AS total_rows FROM instructor;

-- Low-selectivity query (many rows match)
EXPLAIN SELECT * FROM instructor WHERE dept_name = 'Physics';
```

```
 * mysql+pymysql://root:***@localhost?local_infile=1
1 rows affected.
2 rows affected.
1 rows affected.
```

[24]:
```
[('10101', 'Srinivasan', 'Comp. Sci.', Decimal('65000.00'))]
```

[29]:
```
%%sql
-- Low-selectivity query (many rows match)
SELECT count(*) FROM instructor WHERE dept_name = 'Comp. Sci.';
```

```
 * mysql+pymysql://root:***@localhost?local_infile=1
1 rows affected.
```

[29]:
```
[(3,)]
```

[33]:
```
%%sql
-- High-selectivity query (only one row matches)
SELECT count(*) FROM instructor WHERE ID = '10101';
```

```
 * mysql+pymysql://root:***@localhost?local_infile=1
1 rows affected.
```

[33]:
```
[(1,)]
```

Comp. Sci. selects 3 professors out of 12 total, giving it a selectivity of .25, whereas the ID selects just one professor, giving it a selectivity of .0833. This means ID has the higher selectivity.

3.5 Commutative Operations

Question

The following is an equivalence rule: `E1      E2              E2    E1`.

Explain under what conditions on the data the rule would be useful and why.

Answer This expresses the "commutativity" of the join operator. In other words, the order of the operands does not affect the final result of the join, assuming it's an equi-join and the join condition is symmetric. This rule is useful when doing query optimization, specifically in reordering joins to minimize query cost. In relation to a more concrete case, if E1 where much larger than E2, then it would be more efficient to treat E2 as the outer relation in a nested-loop or hash join (p. 752-4).

3.6 Query Cost

Question

Assume the following for a relation: 1. The "cost" of accessing a block is t_b. 2. There is a primary key. 3. There is a B+ Tree with degree 10.

Using the approach used in class and O() notation, what is your estimate of the worst case number of block accesses for an equality select on a primary key for a relation with N tuples?

Answer Assumptions: - Table has N tuples - The primary key is indexed with a B+ tree, which has a fan-out of 10, meaning each internal node has up to 10 children - Each block access costs t_b, which we'll count/multiply where needed - Operation being performed is an equality selection (i.e., SELECT * FROM R WHERE ID = 10101;)

In estimating the worst-case number of block accesses to find a single tuple by its primary key, using the B+ Tree index, we need to think about the height of the tree in the worst case, which is $h = O(\log_{10}(N))$. h is the number of index node accesses needed to reach the leaf.

If clustered $O(\log_{10}(N))$ block accesses and if not clustered $O(\log_{10}(N)) + 1$ for additional access to the actual data block.

3.7 Multiple Conditions

Question

Consider the `db_book,instructor` table above. Consider the two following queries: 1. `SELECT * FROM db_book.instructor WHERE dept_name='Comp. Sci.' AND name='Smith'` 2. `SELECT * FROM db_book.instructor WHERE dept_name='Comp. Sci.' OR name='Smith'`

Does the index on `dept_name` help with (1), (2), both or none? Why?

Answer

In the first query, one side (dept_name) is indexed while the other side is not (name). In the case of selection criteria combined with AND the index still partially helps because the optimizer can use the index on dept_name to locate all the instructors in "Comp. Sci." and then it can apply the additional filter of name = "Smith". In the second query, even though it has the same setup, the OR makes it unusable to simply rely on the index to condense the "filtering scan" as it was done in query 1.

3.8 Hash Index

Question

Consider the query `SELECT name, salary from db_book.instructor`.

What modification to the query might cause the query optimizer to build and use a hash index for processing? Briefly explain how the index would help.

Answer

A hash index is most useful when performing: • Equality comparisons (=), especially on a key or low-cardinality attribute • Joins based on equality (=) • Lookups where you care about existence or exact matches (e.g., WHERE id = '12345')

So adding a filter would be sufficient for prompting the optimizer to use a hash index or hash-based access method. `SELECT name, salary  FROM db_book.instructor  WHERE ID = '10101';`

3.9 Serializability

Question

Briefly explain the concepts of *serializable* and *conflict serializable.*

Answer

A schedule, which is an interleaving of operations from multiple transactions, is serializable if its outcome is equivalent to some serial execution of those transactions — that is, as if the transactions were run one after another with no interleaving.

A schedule is conflict serializable if it is conflict-equivalent to some serial schedule. In other words, if we can reorder non-conflicting operations to produce a serial schedule, the schedule is conflict serializable. This is typically easier to detect algorithmically like in a precedence graph (p. 816).

As a side note: All conflict-serializable schedules are serializable, but not all serializable schedules are conflict-serializable.

```
[ ]:
```

3.10 Buffer Management, Transactions and Recovery

Question

Buffer pool management policies may be: - Force or No Force - Steal or No Steal

Which combination has the best performance/is most desirable? What technique does a DBMS use to ensure ACID properties?

Answer - Force means any modified/dirty pages are written to disk at commit time - No force means that modified pages may remain in memory at commit time - Steal means that the buffer manager can write dirty pages to disk before the transaction commits. - No steal means that dirty pages cannot be written to disk until the transaction completes.

Most desirable for performance = No Force + Steal - No force improves performance by avoiding unnecessary disk writes at commit time, and steal improves flexibility because pages can be flushed when memory pressures demands it (in order to avoid buffer overflow). There are major downsides of this combo though, as it requires a robust recovery mechanism since No force will need a REDO after a crash and steal might need an UNDO.

DBMS uses write ahead logging and the ARIES recovery protocol (pg. 941). This actually supports STEAL + NO FORCE because it uses a log-based recovery algorithm to handle an analysis phase, redo phase, and undo phase.

3.11 Scalability

Question

Briefly explain the concepts of *scale up* and *scale out.*

What are some pros and cons of each technique?

What are some properties and queries in relational databases that make achieving scale-out difficult for SQL?

Answer

Scaling up (also known as vertical scaling) relates to adding more resources on a single machine (i.e., increasing CPU, expanding memory, improving throughput). Some relational DBMS (like MySQL) respond well to this having a "singular powerful node." Scaling out (also known as horizontal scaling) means adding more machines to distribute workloads. This second approach requires changes to the underlying operations because you need to account for how to distrubute/shard the information between multiple machines.

3.12 Locking

Question

What is a deadlock? Give three algorithms/techniques that a DBMS might apply to resolve deadlocks?

Answer

A deadlock occurs in a database when two or more transactions are waiting for each other to release locks, and none of them can proceed. In other words, each transaction holds a resource the other needs, leading to a circular wait that prevents progress. Techniques 1. Timeouts (i.e., Deadlock Detection via Timeout) - Each transaction is allowed to wait only for a fixed amount of time, and if the time is exceeded, the system assumes a deadlock and rolls back the transaction. This is one of the simplest implementations. 2. Wait-for Graph and Deadlock Detection - Nodes = transactions; Edges = "T1 is waiting for T2"; The system periodically checks for cycles in the graph, and if a cycle is found → deadlock detected → rollback one transaction in the cycle. 3. Wait-Die (Deadlock Prevention via Timestamps) - The strongly named "Wait-Die" means the older transaction waits; younger one dies (is rolled back).

3.13 Isolation

Question

With respect to isolation levels/locking, what is a *phantom read?* Explain how a phantom read can occur?

Answer

A phantom read is a phenomenon in transaction isolation where a transaction reads a set of rows that satisfy a condition, but later reads the same condition and finds new rows ("phantoms") that were inserted (or deleted) by another transaction.

It typically happens in this scenario where Transaction T1 runs: SELECT * FROM employees WHERE department = 'Sales'; (reads 10 rows); then transaction T2 inserts a new employee into the 'Sales' department INSERT INTO employees VALUES (..., 'Sales'); COMMIT; Transaction T1 re-runs the same SELECT, but now it reads 11 rows — the "phantom" row appears.

3.14 Information Integration

Question

Briefly explain the following concepts: 1. Extract-Transform-Load 2. Extract-Load-Transform 3. Data Warehouse 4. Data Lake

Answer 1. ETL is a traditional process for preparing data for analysis in a data warehouse, where we first extract, or pull data from multiple sources (databases, APIs, etc.), then transform or clean/normalize/aggregate, and apply "business rules" before loading. Load moves the cleaned, transformed data into the target system, typically a data warehouse. 2. ELT flips the last two steps and is usually associated with data lakes. In this regard, load moves raw data into a data lake or scalable cloud storage, and transform cleans and processes the data after it's loaded (this often happens using SQL or pipelines inside the data platform). 3. A data warehouse is a centralized, structured repository designed for: Querying and reporting, OLAP (Online Analytical Processing), and supports predefined schemas, high-performance aggregations, etc. 4. A data lake is essentially a centralized storage system that holds raw data in its original format, i.e., Structured (SQL tables), Semi-structured (JSON, XML), and unstructured (videos, images, text).

3.15 Big Data

Question

Briefly explain the concepts of: 1. RDDs 2. DAG 3. Algebraic Operations

in technology like Spark.

Answer 1. RDDs are the “fundamental data structure of Apache Spark” and are immutable, lazy-evaluated, and fault-tolerant collections of data, distributed across multiple nodes. They support two kinds of operations: Transformations “(e.g., map, filter) → create a new RDD from an existing one” and actions (e.g., collect, count) → trigger computation and return results to the driver. 2. DAG (Directed Acyclic Graph) is a concept from graph theory and ensures no loops. When an action (like collect()) is called, the DAG is submitted to the DAGScheduler which breaks it into stages and tasks. DAGs allow Spark to optimize execution, group narrow transformations into stages, and recover from failures by rerunning only the necessary stages (pg. 13, “How does SPARK…”). 3. Though not labeled explicitly as “algebraic” in the slides, Spark transformations like reduceByKey, aggregate, and fold are algebraic operations in that they are associative and commutative, which enables partial aggregation on partitions, and are “Divide-and-conquer” transformations (e.g., reduceByKey).

4 Practical Questions

4.1 MongoDB

Question

This question uses the `characters` collection that you loaded for HW3.

Write a query that uses the `characters` collection to produce the result below, which is a table of Arya Stark and murder victims.

Answer

[ ]:
```
# Place your aggregation pipeline here.
#
aggregation_pipeline = [
    {
        "$project": {
            "characterName": "$killedBy",    # attacker
            "killed": "$characterName"       # victim
        }
    }
]
```

[ ]:
```
result = mongo_client['S25_GoT']['characters'].aggregate(aggregation_pipeline)
result = pandas.DataFrame(list(result))
result
```

For reference, my result looks like the following

Mongo Query Result

4.2 Neo4j

Question

Use Neo4j Aura and the Movie dataset for this question.

Write and execute a Cypher query in the Neo4j Aura query console the produces a graph that shows the Movies Tom Hanks directed and the Persons that acted in them. Copy and paste your query in the text cell below. Replace the image in the subsequent cell with a screen capture of successful execution of your query.

Answer

MATCH (h:Person {name: "Tom Hanks"})-[:DIRECTED]->(m:Movie)<-[:ACTED_IN]-(a:Person) RETURN h, m, a

Replace this image with successful execution.

Neo4j Result

```
[ ]:
```