CSCI 4125/5125: Data Models and Database Systems

Phase 2: SQL & Transaction Processing

Dr. James Wagner

Fall 2025

Due: Wednesday, October 22nd @ 11:59pm (e.c. if submitted early. No late submissions.)
Reading: Silberschatz Chapters 3, 4, & 17

Submission Guidelines:

1. This assignment is worth 60 points for all students.
   - +12 extra credit points if submitted by Thursday, October 16th @ 11:59pm.
   - +6 extra credit points if submitted by Sunday, October 19th @ 11:59pm.
2. All answers in the form of images or screenshots must be readable.
3. It is your responsibility to make sure all files are readable and submitted on time.

Submission Files:

- **[lastname]_Phase2.pdf** – a main PDF file with your name at the top.
- **Phase2_Task1.sql** – a .sql file containing your SQL DDL.
- **Phase2_Task2.java** – a Java source file for your data loading code. Name as a comment at the top.
- **Phase2_Task3.sql** – a .sql file containing your retrieval queries.
- **Phase2_Task4.java** – a Java source file your query processing code. Name as a comment at the top.

Submission Checklist:

Task	Pts	Submission Items
SQL DDL	10	A single .sql file.
Data Loading	10	A screenshot in your main PDF & a Java source file.
SQL Queries	30	A single .sql file.
Query Processing	10	A screenshot in your main PDF & a Java source file.
Transactions	10	Solution in your main PDF.

Table 1: Submission Checklist

Recommended Completion of Tasks. You have ∼5 weeks to complete this project phase. I highly recommend allocating several hours/week to this rather than starting on this the day before it is due. Table 2 is a recommended timeline to complete tasks.

Task	Finish	Estimated Time
SQL DDL	Thu, 9/25	1 - 2 hrs
Data Loading	Thu, 9/25	2 - 3 hrs
SQL Queries	Thu, 10/9	2 - 4 hrs
Query Processing	Thu, 10/9	2 - 4 hrs
Transactions	Thu, 10/16	1 - 2 hrs

Table 2: Recommended completion timeline of tasks.

1 SQL DDL

Write a SQL DDL script to create the tables for the project. Use the schema in Figure 1 and the following guidelines:

- At the top of your script include a `DROP` table command for each of your tables. Note that you must pay attention to referential integrity when considering the order to drop tables.
- Column names should match the attributes in the relational schema. This will allow for consistency in our queries later.
- Columns must use reasonable domains based on the data in the included .txt files.
- All primary keys must be declared.
- All foreign keys must be declared.
- Run your SQL script and debug any errors.
- **Submit:** a single .sql file

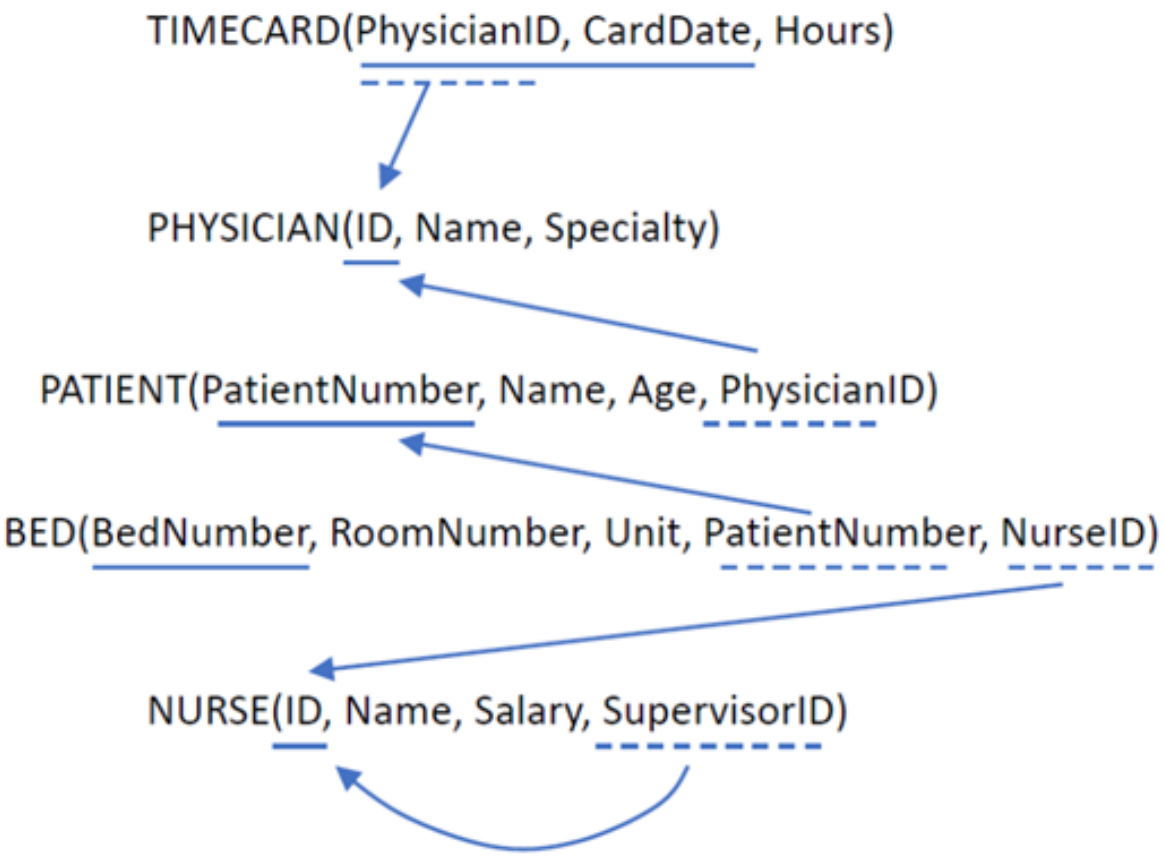

Figure 1: Relatioanl Schema

2 Data Loading

Writing SQL insert statements is tedious if you have a few dozen records and unrealistic if you have thousands or millions of records to populate. Also, picture a user interface (e.g., a website) you may develop for a typical, non-CS end user. You would embed SQL statements in the code and automatically generate SQL based on user input; you don't want the typical user writing SQL! You will now modify your Java code from Phase 1 to automatically generate SQL `INSERT` statements. Your Java program must include the following functionality:

1. It must accept a command line argument, which is the table name that the record will be inserted into. Note: in this project, table names will match the names of the text files I provided to you. For example, if I provided the *animal.txt* file, it would contain data for the *"animal"* table. When you run `$java Phase2_Task2 animal`, your code should read the file *animal.txt* and output to *animal.sql*. Hint: string placeholders (e.g., "%s.txt" and "%s.sql") simplifies this.

2. Datatypes should be correctly formatted in your `INSERT` statements. Do not hardcode the positions of values in files. I gave you input text files with varying schemas. Your Java program should work for any schema without making changes.

   (a) Numbers and floats don't use single quotes.

   (b) Strings use single quotes.

   (c) Also, consider NULL values now. NULL values do not use single quotes. As an example, if an input line for table Example is:
   `hello, world, 4125, NULL, 5125`
   my output line should be:
   `INSERT INTO Example VALUES('hello', 'world', 4125, NULL, 5125);`
   Hint: Again, string placeholders can really simplify this for you. I recommend using a placeholder for your table name and the values inside the parenthesis (e.g., `INSERT INTO %s VALUES (%s);`). Then you can simply place your table name and formatted values into the placeholder values.

Use the included text files with your DDL to test your `INSERT` statements. Below is a checklist of items:

- Accept a `tablename` as a command line argument.
- Read the input file as [`tablename`].txt.
- Create an output file as [`tablename`].sql.
- Build a SQL `INSERT` statement template
- For each line in file, determine the datatype for each value and format accordingly. Strings use single quotes. Integers, floats, and NULLs do not use single quotes.
- Complete the SQL `INSERT` statement. Add the `tablename` and the correctly formatted values into your SQL `INSERT` template.
- Write the `INSERT` statement to your output file.
- Test your output!

Using your Java program, generate `INSERT` statements for the six .txt files included with this document. You should generate a total of six SQL scripts containing `INSERT` statements. Use the following guidelines to submit your work:

- Data types must be properly formatted, e.g., strings must use single quotes, dates must use the correct format.
- Some strings might contain a single quote. Make sure the single quote appears in the value. This can be easily addressed with the replace() method in your Java code.
- Each script should include a commit (i.e., "`commit;`") at the end of the file. You can modify your Java program to simply write that before you close the outfile.
- Name each file [table name].sql.
- Run each script in SQL Developer. Remember that you can run SQL scripts using @[path]\[file].sql or you can copy-paste the statements into SQL Developer.
- You will need to run the `INSERT`'s in the proper order that does not violate referential integrity.

- Read the output that is generated. If there are any errors, it is up to you to fix them. Errors to watch out for include, improper syntax, violating domain constraints, violating primary key constraints, and violating referential integrity.

Submit: Your Java source file and a screenshot like Figure 2 in your main PDF. To get this screenshot, run the following query. You may need to fix some characters that do not properly copy-paste, such as quotations. Your screenshot must include: your connection name, query, and output (like Figure 2).

```
SELECT 'Bed:'|| COUNT(*) AS Cnt FROM Bed
UNION
SELECT 'Nurse:'|| COUNT(*) AS Cnt FROM Nurse
UNION
SELECT 'Patient:'|| COUNT(*) AS Cnt FROM Patient
UNION
SELECT 'Physician:'|| COUNT(*) AS Cnt FROM Physician
UNION
SELECT 'Timecard:'|| COUNT(*) AS Cnt FROM Timecard;
```

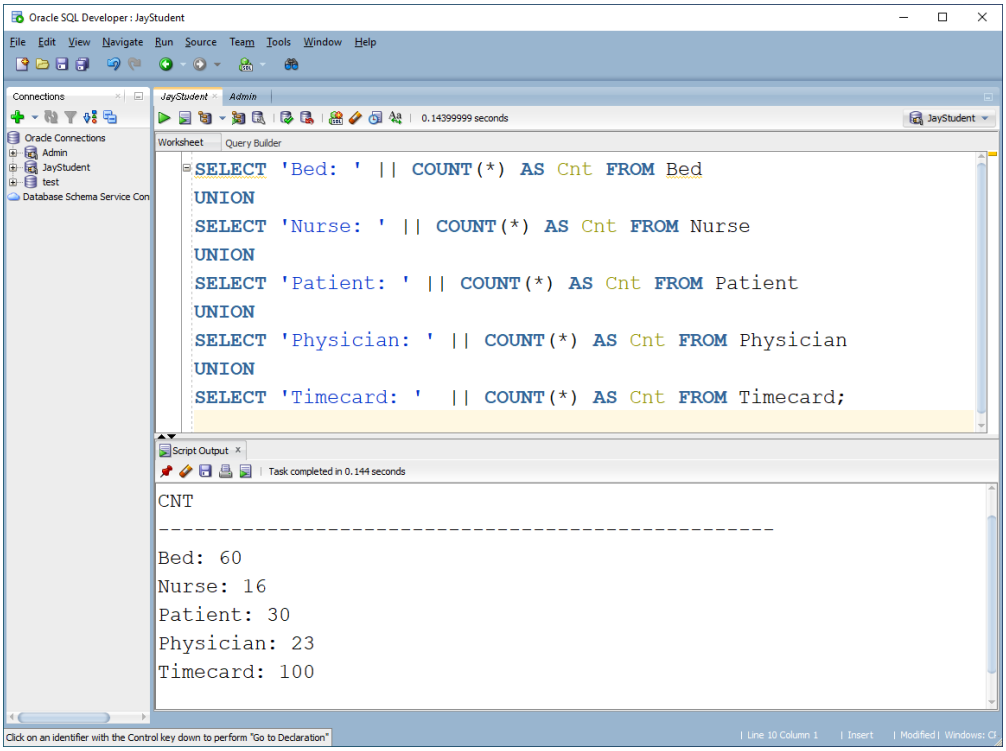

Figure 2: Verifying the data loaded.

3 Retrieval Queries

Write a SQL query for each of the following problems. Unless values are specified in the problem, do not hardcode values in queries. In other words, your query should still answer the problem if the data changes.

Submit: your answers in a single .sql. You must use a comment to label each answer.

-- Queries 1 - 8 are worth 1 point each.
--1. Find all the "surgery" (for the specialty) physicians and sort the query output by names (any direction).
--2. Find all physicians with medicine in their specialty name (hint: remember the LIKE operator and case sensitivity).
--3. Find all nurses who do not have a supervisor.
--4. Find the names of all nurses with a salary between $70,000 and $80,000 (inclusive).
--5. Find the minimum and maximum salaries amongst all nurses. Use only one query that returns a single row (ex. 50000, 100000).
--6. Find the average salary for all nurses.
--7. Find the name of the nurse that has the highest salary. Do not hardcode any salaries or other values – you must use SQL without assuming you know the current database snapshot.
--8. Find the nurses with a salary less than the average overall salary for all nurses + 20% (i.e., less than 1.2 * average salary). Do not hardcode any salaries or other values – you must use SQL without assuming you know the current database snapshot.

-- Queries 9 - 19 are worth 2 points each.
--9. Retrieve the names of all nurses who monitor at least one bed. Make sure to remove duplicates.
--10. For each physician ID, list the total number of hours worked.
--11. Retrieve the names of all nurses who do not monitor any beds.
--12. Find the names of all nurses who are directly supervised by Chris Summa. Note: You must use the string 'Chris Summa' and not hardcode the nurse ID.
--13. Retrieve the names of all physicians who specialize in dermatology and have work more than 22 hours.
--14. For each physician specialty, list the specialty, the number of physicians that have that specialty, and the total number of hours worked by those physicians.
--15. Retrieve the names of all nurses whose supervisor's supervisor has N01 for their ID.
--16. For each physician specialty, find the total number of patients whose physician has that specialty.
--17. Find the patient who is assigned to a bed that is monitored by the nurse with the lowest salary.
--18. Retrieve the average age of patients assigned to a bed.
--19. This problem requires (no points if you don't) you to use the regular expression function that we discussed in class. Find all beds that have a room number that end in an odd number (e.g., return 101, but not 102).

4 Query Processing

One of the benefits of SQL is program-data independence supported by the relational model (or the DBMS). However, to fully understand the power of SQL, it is helpful to know how the equivalent could be implemented in the flat file model. In this task, you will write Java methods, not SQL, that are the flat file model solution for Queries #9 and #10 in Task 3. Your method should include the following:

- Read the relevant .csv/.txt file(s). You can hardcode the file name(s).
- Process the input to compute the query result. When you process the input, you will now have to hardcode the column positions.
- Print the results to the console. As a sanity check, your results should be the same as in Task 3.
- None of your code will use SQL – Java only.
- Hint: We have used some useful methods in the project, e.g., trim(), split(), etc.
- Hint: It may be useful to store results in a data structure, such as a HashMap, that you can later print after you after done all of the processing.
- Hint: One way to process a JOIN or a subquery is a double for-loop.

Submit: A single Java file containing a method that solves only Queries #9 and #10 in Task 3, i.e., a total of two methods besides main() and a screenshot of the output printed to the console in your main PDF.

5 Transaction Processing

Consider the three transactions T1, T2, and T3, and the schedules S1 and S2 given below. Draw the serializability (precedence) graphs for S1 and S2, and state whether each schedule is serializable or not. If a schedule is serializable, provide the equivalent serial schedule(s). Include your work and your answers in your main PDF. You can draw your graphs using a program of your choice or pen and paper.

Transactions:

T1: R1(X), R1(Z), W1(X)

T2: R2(Z), R2(Y), W2(Z), W2(Y)

T3: R3(X), R3(Y), W3(Y)

Schedules:

S1: R1(X), R2(Z), R1(Z), R3(X), R3(Y), W1(X), W3(Y), R2(Y), W2(Z), W2(Y)

S2: R1(X), R2(Z), R3(X), R1(Z), R2(Y), R3(Y), W1(X), W2(Z), W3(Y), W2(Y)