CSCI 4125/5125: Data Models and Database Systems

Phase 3: JDBC, PL/SQL, Normalization

Dr. James Wagner

Fall 2025

Due: Monday, December 1st @ 11:59pm (e.c. if submitted early. No late submissions.)
Reading: Silberschatz Chapters 5 & 7

Submission Guidelines:

1. This assignment is worth 60 points for all students.
   - +12 extra credit points if submitted by Tuesday, November 25th @ 11:59pm.
   - +6 extra credit points if submitted by Friday, November 28th @ 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]_Phase3.pdf** – a main PDF file with your name at the top.
- **Phase3_Task1.java** – a Java source file containing your code.
- **Phase3_Task2.sql** – a .sql file containg your procedure.
- **Phase3_Task3.sql** – a .sql file containing your three triggers.

Submission Checklist:

Task	Pts	Submission Items
JDBC	15	A Java file and a screenshot in your main PDF.
PL/SQL: Procedure	15	A .sql file for your procedure.
PL/SQL: Trigger	15	A .sql file for your 3 triggers.
Normalization	15	Answer the seven questions in your main PDF.

Table 1: Submission Checklist

Recommend 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
JDBC	Tue, 11/4	1 - 2 hrs
PL/SQL: Procedure	Tue, 11/11	1 - 2 hrs
PL/SQL: Trigger	Tue, 11/18	2 - 3 hrs
Normalization	Tue, 11/25	1 - 2 hrs

Table 2: Recommended completion timeline of tasks.

1 JDBC

Introduction. Your task is to write a Java application that generates a supervisor report for a given nurse ID. This report will also compute the derived attribute, beds monitored, we saw in Phase 1.

Functionality. Your application must include the following:

- Accept one argument: nurse ID.
- When you use the argument in a SQL query, you are required to use prepared statements. Points will be deducted if you use string concatenation to put these values into a SQL query.
- If a nurse does not exist for the given ID, print "No nurse found for the given ID!" and return.
- If a nurse is not a supervisor for the given ID, print the nurse's name and "This nurse is not a supervisor." and then return.
- Provide a report header that includes: the nurse's name, the number of nurse's they directly supervise, and the total salary of the nurse's they directly supervise.
- List all of the nurses directly supervised. Include the nurse ID, name, and the total number of beds that they are monitoring.
- You are welcome to add additional information if you'd like to.

Testing. Figure 1 contains some example values passed to my application along with the report they return. Your formatting does not need to match mine exactly, but it must contain all of the relevant information listed above.

Submit. Submit a Java source code file and a screenshot that demonstrates your running program. Figure 1 provides an example screenshot.

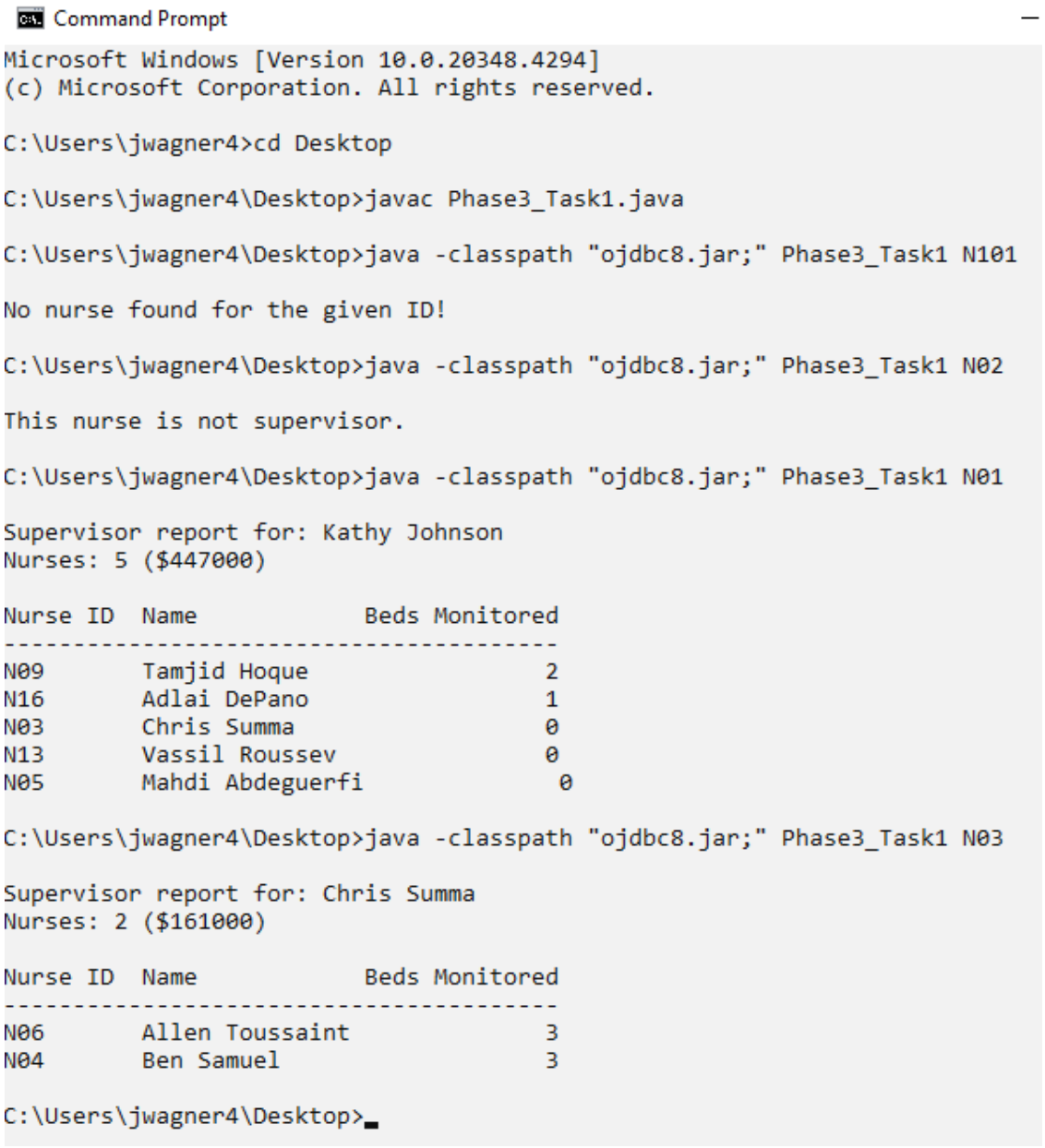

Figure 1: Supervisor reports created using JDBC.

2 PL/SQL: Procedures

Introduction. Your task is to write a PL/SQL procedure that generates a supervisor report for a given nurse ID. This report will also compute the derived attribute, beds monitored, we saw in Phase 1. You will use DBMS_OUTPUT.PUT_LINE to print your report text. Keep in mind that it may be easier to start by building an anonymous PL/SQL block that you can later store as a procedure.

Functionality. Your procedure must include the following:

- Accept one argument: nurse ID.
- If a nurse does not exist for the given ID, print "No nurse found for the given ID!" and return.
- If a nurse is not a supervisor for the given ID, print the nurse's name and "This nurse is not a supervisor." and then return.
- Provide a report header that includes: the nurse's name, the number of nurse's they directly supervise, and the total salary of the nurse's they directly supervise.
- List all of the nurses directly supervised. Include the nurse ID, name, and the total number of beds that they are monitoring.
- You are welcome to add additional information if you'd like to.

Testing. Figure 2 contains some example procedure calls along with the invoice they return. Your formatting does not need to match mine exactly, but it must contain all of the relevant information listed above.

Hint: You will need to use a cursor, but not more than one. If you find yourself using several cursors, 1) look at how you could improve your cursor and 2) ask if you could solve the problem with a `SELECT INTO`.

Hint: You should be able to write this procedure in ∼35 - 45 lines of code. If your solution uses more than 70 lines, I encourage you to ask questions in class or office hours.

Hint: If you want to format your output similar to mine, PL/SQL supports the LPAD() and RPAD() functions you have probably seen in other programming languages. Here is an example of this that you can run:

```
SELECT LPAD('$' || TO_CHAR(95000), 10) AS "Salary" FROM Dual;
```

Submit. Submit a seperate .sql with the code for your procedure.

```
call HospitalReport('N101');

    No nurse found for the given ID!


call HospitalReport('N02');

    Supervisor report for: James Wagner
    This nurse is not supervisor.


call HospitalReport('N01');

    Supervisor report for: Kathy Johnson
    Nurses supervised: 5 ($447000)

    Nurse ID  Name            Beds Monitored
    ----------------------------------------
    N09       Tamjid Hoque                 2
    N16       Adlai DePano                 1
    N03       Chris Summa                  0
    N13       Vassil Roussev               0
    N05       Mahdi Abdeguerf              0


call HospitalReport('N03');

    Supervisor report for: Chris Summa
    Nurses supervised: 2 ($161000)

    Nurse ID  Name            Beds Monitored
    ----------------------------------------
    N06       Allen Toussaint              3
    N04       Ben Samuel                   3
```

Figure 2: Supervisor reports created using a PL/SQL procedure.

3 PL/SQL: Triggers

Introduction. Your task is to write three triggers that will maintain the number of beds a nurse is monitoring. While you are not required to use DBMS_OUTPUT.PUT_LINE, I highly recommend using it for testing and debugging your code. Note, this is an example of a computed column that we saw in Chapter 6: E-R Modeling. First, add a column to the Nurse table to store this value using the following command:

```
ALTER TABLE Nurse ADD BedsMonitored NUMBER DEFAULT 0;
```

If you make a mistake along the way, you can reset this column with a simple update:

```
UPDATE Nurse SET BedsMonitored = 0;
```

Trigger 1 Functionality. Write a trigger that will fire when you `INSERT` a row into the Bed table. This trigger will check the value of BedsMonitored for the corresponding Nurse's ID.

- If Nurse.BedsMonitored is less than 2, then a nurse can monitor more beds.
- If Nurse.BedsMonitored is equal to 2, then the nurse is too busy so your trigger will cancel the INSERT and display a custom error message stating that the Nurse is too busy.
- Also, be sure to handle the inserts for bed where the NurseID is NULL. One way I suggest doing this is too to handle the NO_DATA_FOUND EXCEPTION (i.e., WHEN NO_DATA_FOUND THEN).

Testing. You can test your trigger with the following:

- Delete all records from Bed: `DELETE FROM Bed;`
- Run your .sql script containing your `INSERT` statements for the Bed table.
- The Bed table should have 57 records since the inserts for B25, B26, B55 were rejected.
- The insert for Bed B25 should produce an error message that looks Figure 3.

```
1  INSERT INTO bed VALUES('B25', 205, 'Surgery', 'P27', 'N04')
2  Error report -
3  SQL Error: ORA-20201: Nurse is too busy!
4  ORA-06512: at "JAYSTUDENT.ADDBED", line 15
5  ORA-04088: error during execution of trigger 'JAYSTUDENT.ADDBED'
```

Figure 3: Nurse is too busy error for Bed B25.

Trigger 2 Functionality. Write a trigger that will fire when a user attempts to `DELETE` one or more rows from BED. This trigger will update the Nurse.BedsMonitored for the corresponding Nurse's ID by decreasing the value by one each time a row is removed from BED.

Testing. You can test your trigger with the following:

- Choose a Bed to delete. Ex: `DELETE FROM Bed WHERE BedNumber = 'B12';`
- `SELECT * FROM Nurse` should return the output in Figure 4 if you delete B12:

ID	NAME	SALARY	SUP	BEDSMONITORED
N01	Kathy Johnson	120000		2
N13	Vassil Roussev	92000	N01	0
N03	Chris Summa	88000	N01	0
N05	Mahdi Abdeguerfi	90000	N01	0
N09	Tamjid Hoque	88000	N01	1
N02	James Wagner	75000	N05	2
N04	Ben Samuel	79000	N03	2
N06	Allen Toussaint	82000	N03	2
N07	Ted Holmberg	81000	N05	0
N08	David Pace	80000	N09	2
N10	Dr. John	80000	N09	0
N11	Joe Sylve	78000	N05	2
N12	Redwan Newaz	82000	N13	0
N14	Shreya Banerjee	83000	N13	1
N15	Yasin Nur	77000	N13	2
N16	Adlai DePano	89000	N01	1

Figure 4: Tamjid Hoque's BedsMonitored was updated after deleting Bed B12.

Trigger 3 Functionality. Write a trigger that fires when a nurse ID for a bed is changed (i.e., `UPDATE`).

- This trigger will update the Nurse.BedsMonitored for both the previous Nurse and the new Nurse using their respective ID's.
- If the Nurse.BedsMonitored value for the new Nurse in the update is equal to 3, then the nurse is too busy so your trigger will cancel the UPDATE and display an error message stating that the Nurse is too busy (similar to the first trigger).

Testing. You can test your trigger with the following:

- Update a bed's NurseID (which is currently NULL) to a nurse that does not monitor too many beds. Here's one query you can use, and the corresponding Nurse records in Figure 5:
  `UPDATE Bed SET NurseID = 'N03'WHERE BedNumber = 'B01';`
- Update a bed's NurseID for a nurse that is already monitoring the max number of beds allowed. Here's one query you can use and the corresponding error message in Figure 6:
  `UPDATE Bed SET NurseID = 'N04'WHERE BedNumber = 'B03';`
- Update a bed's NurseID that switches the nurse's monitoring the bed. Here's one query you can use, and the corresponding content from the Nurse table in Figure 7.
  `UPDATE Bed SET NurseID = 'N14'WHERE BedNumber = 'B21';`

```
ID  NAME                      SALARY SUP BEDSMONITORED
--- ------------------------- ------ --- -------------
N01 Kathy Johnson             120000                 2
N13 Vassil Roussev             92000 N01             0
N03 Chris Summa                88000 N01             1
N05 Mahdi Abdeguerfi           90000 N01             0
N09 Tamjid Hoque               88000 N01             1
N02 James Wagner               75000 N05             2
N04 Ben Samuel                 79000 N03             2
N06 Allen Toussaint            82000 N03             2
N07 Ted Holmberg               81000 N05             0
N08 David Pace                 80000 N09             2
N10 Dr. John                   80000 N09             0
N11 Joe Sylve                  78000 N05             2
N12 Redwan Newaz               82000 N13             0
N14 Shreya Banerjee            83000 N13             1
N15 Yasin Nur                  77000 N13             2
N16 Adlai DePano               89000 N01             1
```

Figure 5: Chris Summa's BedsMonitored was updated after assigning him to Bed B01.

```
UPDATE Bed SET NurseID = 'N04' WHERE BedNumber = 'B03'
Error report -
SQL Error: ORA-20201: Nurse is too busy!
ORA-06512: at "JAYSTUDENT.UPDATEBED", line 15
ORA-04088: error during execution of trigger 'JAYSTUDENT.UPDATEBED'
```

Figure 6: Trying to assign nusre N04 to bed B03 returns an error message.

```
ID  NAME                      SALARY SUP BEDSMONITORED
--- ------------------------- ------ --- -------------
N01 Kathy Johnson             120000                 2
N13 Vassil Roussev             92000 N01             0
N03 Chris Summa                88000 N01             1
N05 Mahdi Abdeguerfi           90000 N01             0
N09 Tamjid Hoque               88000 N01             1
N02 James Wagner               75000 N05             2
N04 Ben Samuel                 79000 N03             2
N06 Allen Toussaint            82000 N03             2
N07 Ted Holmberg               81000 N05             0
N08 David Pace                 80000 N09             2
N10 Dr. John                   80000 N09             0
N11 Joe Sylve                  78000 N05             1
N12 Redwan Newaz               82000 N13             0
N14 Shreya Banerjee            83000 N13             2
N15 Yasin Nur                  77000 N13             2
N16 Adlai DePano               89000 N01             1
```

Figure 7: Bed B21 is reassigned from Joe Sylve to Shreya Banerjee.

Hint: You should be able to write all three triggers in ∼70 lines of code in total. The three triggers will also use a lot of the same code.

Submit. Submit a single .sql with the code for your three triggers.

4 Normalization

4.1 2NF & 3NF: Physician Relation

Introduction. Since you originally designed the hospital database, fields have been added to the original Physician relation (Figure 8). You start to notice some data inconsistencies (due to update anomalies caused by data redundancy). Your job is to evaluate and fix these problems.

Submit. In your main PDF file, include your answers to all 7 questions.

Original Physician Relation
PHYSICIAN(ID, Name, Specialty)

New Physician Relation
PHYSICIAN(ID, Name, Specialty, CellPhone, HireDate, ParkingSpot, UnitFloor, UnitPhone, UnitName, OfficeNumber, OfficeSize, OfficePhone)

F = {
ID, UnitName → ParkingSpot

ID → Name, Specialty, CellPhone, HireDate, OfficeNumber

OfficeNumber → OfficeSize, OfficePhone

UnitName → UnitFloor, UnitPhone
}

Figure 8: Product relation with new attributes.

1. What is the primary key of the NEWRELATION in Figure 8? *Hint:* It's not ID.

2. Remove any partial key dependencies to create a set of linked relational schemas in Second Normal Form. Primary keys require a solid underline. Foreign keys require a dotted underline and an arrow to the attribute(s) they reference.

3. Remove any transitive dependencies to create a set of linked relational schemas in Third Normal Form. Primary keys require a solid underline. Foreign keys require a dotted underline and an arrow to the attribute(s) they reference.

4.2 1NF, 2NF, & 3NF: Customer Relation

Introduction. Someone in your organization created the new Patient relation in Figure 9 to store (multiple) physician IDs for patient.

Original Patient Relation
PATIENT(PatientNumber, Name, Age, PhysicianID)

New Patient Relation
PATIENT(PatientNumber, Name, Age, PhysicianID1, PhysicianID2, PhysicianID3)

Figure 9: New customer relation with payment methods attribute.

4. To allow multiple physician IDs for each patient, how would you alter the relation so that it is in proper 1NF? Show the proper 1NF schema and describe why you chose this. Hint: think back to the pizza toppings example.

5. Did your new relation in 1NF introduce any redundancies? If so, what functional dependencies do you need to decompose on? Hint: it may help to create some example values.

6. Using the functional dependencies you identified, show the new relation(s) in proper 3NF. Primary keys require a solid underline. Foreign keys require a dotted underline and an arrow to the attribute(s) they reference.

4.3 Final Project Schema

7. Using your answers from Sections 4.1 and 4.2, draw the updated schema for the entire hospital in proper 3NF.