SE424: Distributed Systems **Recitation 2**
Semester 1 5785 **11 Nov 2024**
Lecturer: Michael J. May **Kinneret College**

Multi-Tiered Architectures

1 Three Tiered Example

This recitation creates a simple example of a three tiered division of labor. A student information system contains two servers with different information. The first server contains student test registration information - who the students are and which courses for which they have taken tests. The second server contains just booklet numbers, information about whether the booklet was graded, and a grade for the booklet (if there is one).

The schemas for the databases are:

1. Server 1 (Booklets) contains the following database tables:
   - Students (tz:integer, sname:varchar(20), address:varchar(50))
   - Courses (cid:varchar(20), cname:varchar(80), year:integer, semester:integer)
   - Testbooklets (tz:integer, cid:varchar(20), bookno:integer)
2. Server 2 (Grades) contains the following database table:
   - Grades (bookno:integer, grade:integer, checked:integer)

1.1 Components and Deployment

The parts of the system and their deployment are shown in Figure 1.

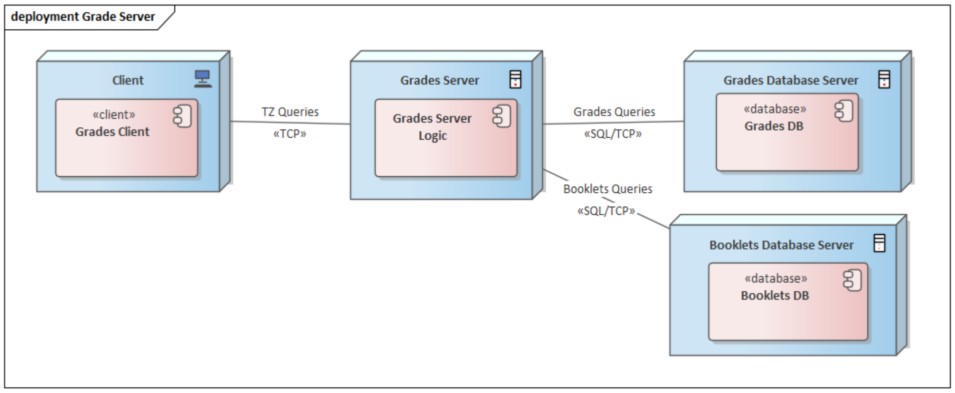


Figure 1: Deployment Diagram for Grades Three Tier Server and Client

1.2 Database Connections and Queries

There are two options you can choose from for the database engine for this recitation.

1. If you have a local installation of MS SQL or MySQL on your computer, you can use it as the database engine. In that case, make two separate databases, one called GradesDB and one called BookletsDB.

2. You can use the SQLite database engine. In that case, you'll need to use two database files, one called gradesdb and one called bookletsdb. You can create your own database files or use the grades.db and booklets.db provided on Moodle.

Depending on which of the database engines you'll choose, you need to connect to the database using a database connection string. The database connection strings for the two databases are different and will be given in class. As a rule, database connection strings to local and remote database engines contain some sort of authentication. Database connections to SQLite databases do not need authentication. The basic steps for connecting to a database using a connection string, sending a query, and retrieving data via a cursor are sketched in Figure 2.

The code we will develop will create an application which will show students information about all of their test booklets, including grades (if there are any). The user inputs a student's id number (tz) and receives a listing of all of the student's test booklets along with an indication for each whether it was graded. If it was, the grade is displayed.

1.3 Using SQLite

You will need to import the Sqlite JDBC driver from Maven to use SQLite (org.xerial:sqlite-jdbc:jar:3.47.0.0). You can read more about the project at Maven Central's web page about the driver (`https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc/3.47.0.0`). Once you have the driver included in the dependencies, you'll need to create an instance of it using the following line of code in the server:

```
Class.forName("org.sqlite.JDBC");
```

Once you have the JDBC driver available, you can use it for queries. To help get you started, the following are two useful queries for this task (note the use of ? for parameters). For MS SQL, the syntax is a bit different.

In Server 1:

```
SELECT S.tz as TZ, S.sname as 'Student Name', C.cname as 'Course Name', C.year
as Year, C.semester as Semester, T.bookno as 'Booklet Number'
FROM TestBooklets T, Students S, Courses C
WHERE S.tz = T.tz AND C.cid = T.cid
AND T.tz = ?
ORDER BY bookno;
```

In Server 2:

```
SELECT bookno as 'Booklet Number', checked as Checked?, grade as Grade
FROM Grades
WHERE bookno = ?;
```

1.4 Useful Classes

The following classes will be useful for you when you want to connect to the database:

- java.sql.Connection (`https://docs.oracle.com/en/java/javase/19/docs/api/java.sql/java/sql/Connection.html`)
- java.sql.DriverManager (`https://docs.oracle.com/en/java/javase/19/docs/api/java.sql/java/sql/DriverManager.html`)
- java.sql.PreparedStatement (`https://docs.oracle.com/en/java/javase/19/docs/api/java.sql/java/sql/PreparedStatement.html`)

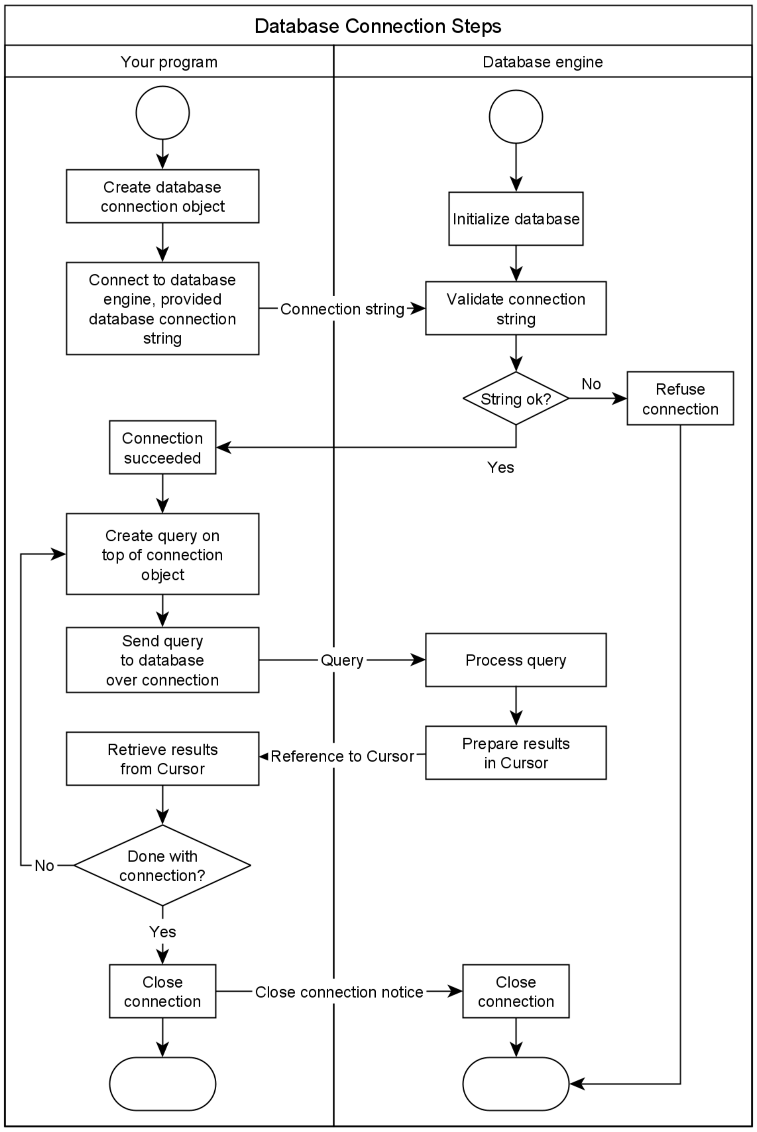

Database Connection Steps
Your program
Database engine
Create database connection object
Connect to database engine, provided database connection string
Connection string
Initialize database
Validate connection string
String ok?
No
Refuse connection
Yes
Connection succeeded
Create query on top of connection object
Send query to database over connection
Query
Process query
Prepare results in Cursor
Reference to Cursor
Retrieve results from Cursor
Done with connection?
No
Yes
Close connection
Close connection notice
Close connection

Figure 2: General steps for connecting to and using a database connection

- java.sql.ResultSet (`https://docs.oracle.com/en/java/javase/19/docs/api/java.sql/java/sql/ResultSet.html`)

You can read about them on the JavaDoc webpages at the URLs above.

1.5 What to do: Processing Server

We'll begin with the processing server part of the work. The server needs to have the following functionality:

- Let the user select which IP address to listen on (or to listen on all IP addresses).
- It must receive the configuration information (port, database connection strings) for its operation from a non-hard coded place (parameters or external file).
- It must be able to handle multiple client sessions at a time (multithreading).
- It must let the user start and stop listening without quitting (interrupting threads).

The server will follow the following interaction protocol:

1. The client connects
2. The client provides a TZ to receive grades for
3. The server responds with a string array representation of the data (it performs the two queries on the databases and joins them)
4. The client can repeat steps 2-3 above as needed.
5. The client disconnects

The server is read only - there is no way to update data on the databases.

The server's output should be as shown in Figure 3.

1.6 What to do: Client

To make things go a bit quicker, I will give you ready code for the client. The client receives the IP address and port of the server from a configuration file and then lets the user connect to the server. It has the following functionality:

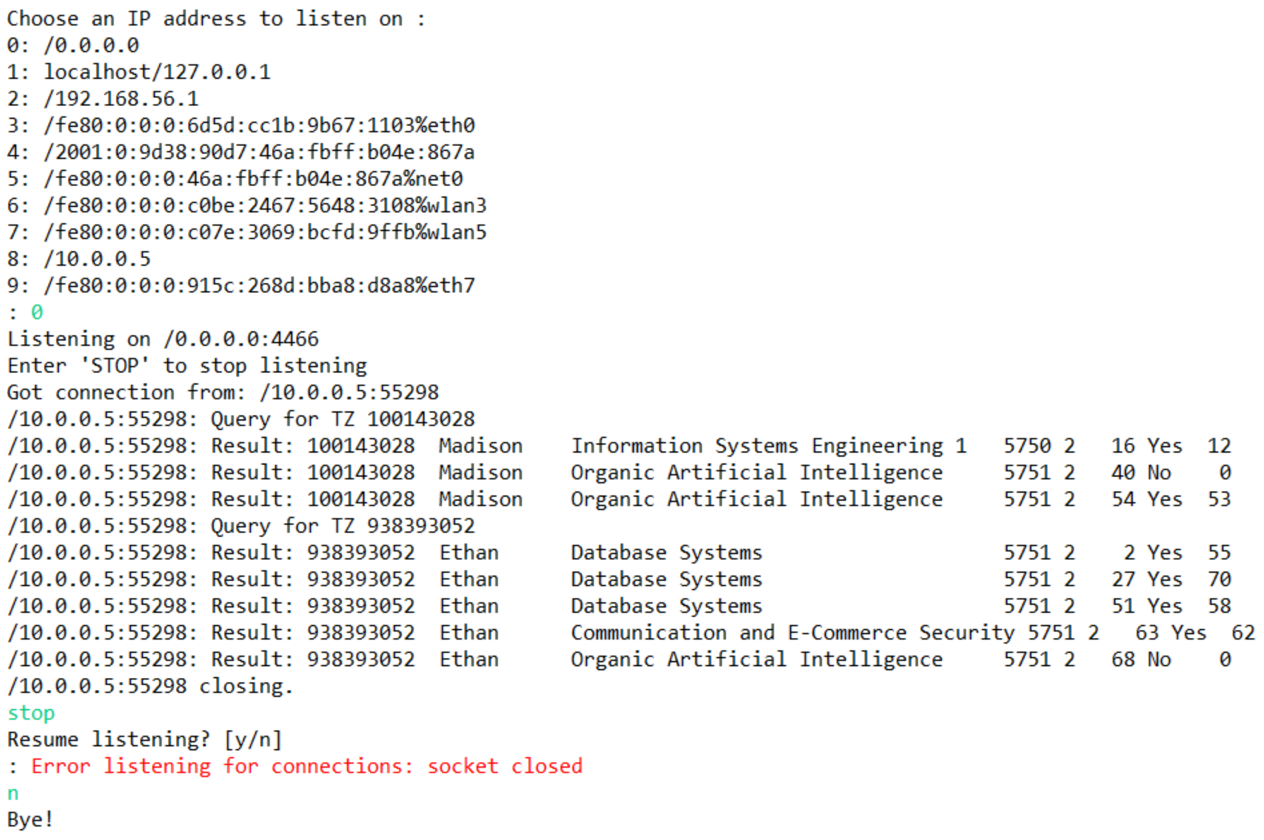
- Let the user connect or disconnect from the server without needing to restart the program
- Send a TZ to the server and show its results.
- Print the output from the server on the screen.

The client's output should be as shown in Figure 4.

2 Next Steps

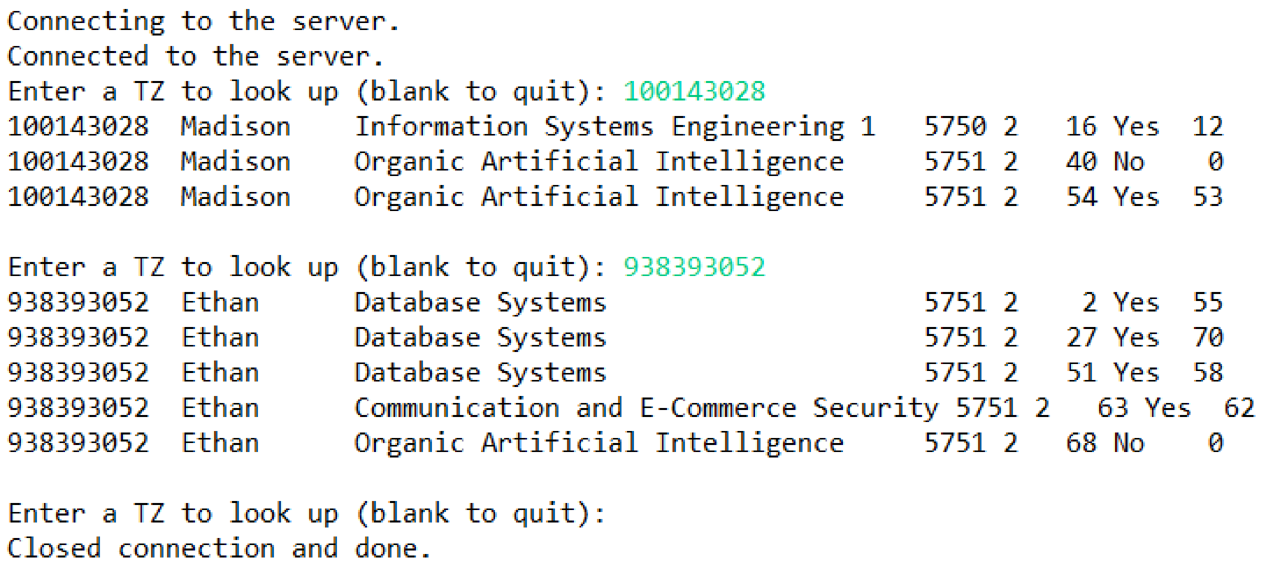
If you finish the application as given, add the following functionality:

1. Enable searching by ID number or student name. This requires including an option at the client program to enter the student name or ID.
2. Retrieve a list of student IDs from the server automatically and use them to populate a list of potential student IDs to query.



```
Choose an IP address to listen on :
0: /0.0.0.0
1: localhost/127.0.0.1
2: /192.168.56.1
3: /fe80:0:0:0:6d5d:cc1b:9b67:1103%eth0
4: /2001:0:9d38:90d7:46a:fbff:b04e:867a
5: /fe80:0:0:0:46a:fbff:b04e:867a%net0
6: /fe80:0:0:0:c0be:2467:5648:3108%wlan3
7: /fe80:0:0:0:c07e:3069:bcfd:9ffb%wlan5
8: /10.0.0.5
9: /fe80:0:0:0:915c:268d:bba8:d8a8%eth7
: 0
Listening on /0.0.0.0:4466
Enter 'STOP' to stop listening
Got connection from: /10.0.0.5:55298
/10.0.0.5:55298: Query for TZ 100143028
/10.0.0.5:55298: Result: 100143028  Madison    Information Systems Engineering 1   5750 2   16 Yes  12
/10.0.0.5:55298: Result: 100143028  Madison    Organic Artificial Intelligence     5751 2   40 No    0
/10.0.0.5:55298: Result: 100143028  Madison    Organic Artificial Intelligence     5751 2   54 Yes  53
/10.0.0.5:55298: Query for TZ 938393052
/10.0.0.5:55298: Result: 938393052  Ethan      Database Systems                    5751 2    2 Yes  55
/10.0.0.5:55298: Result: 938393052  Ethan      Database Systems                    5751 2   27 Yes  70
/10.0.0.5:55298: Result: 938393052  Ethan      Database Systems                    5751 2   51 Yes  58
/10.0.0.5:55298: Result: 938393052  Ethan      Communication and E-Commerce Security 5751 2   63 Yes  62
/10.0.0.5:55298: Result: 938393052  Ethan      Organic Artificial Intelligence     5751 2   68 No    0
/10.0.0.5:55298 closing.
stop
Resume listening? [y/n]
: Error listening for connections: socket closed
n
Bye!
```

Figure 3: Sample Server Output Trace



```
Connecting to the server.
Connected to the server.
Enter a TZ to look up (blank to quit): 100143028
100143028  Madison    Information Systems Engineering 1   5750 2   16 Yes  12
100143028  Madison    Organic Artificial Intelligence     5751 2   40 No    0
100143028  Madison    Organic Artificial Intelligence     5751 2   54 Yes  53

Enter a TZ to look up (blank to quit): 938393052
938393052  Ethan      Database Systems                    5751 2    2 Yes  55
938393052  Ethan      Database Systems                    5751 2   27 Yes  70
938393052  Ethan      Database Systems                    5751 2   51 Yes  58
938393052  Ethan      Communication and E-Commerce Security 5751 2   63 Yes  62
938393052  Ethan      Organic Artificial Intelligence     5751 2   68 No    0

Enter a TZ to look up (blank to quit):
Closed connection and done.
```

Figure 4: Sample Client Output Trace

3 Appendix: Students table contents

To help with testing, here are the student IDs of the students in the sample database I gave you.

tz	sname	address
100143028	Madison	Los Angeles
235129602	Anthony	Los Angeles
246574437	Michael	Chicago
260249572	Isabella	Phoenix
339495799	Sophia	Houston
446994639	Jacob	New York
523105229	Ava	New York
546140696	Abigail	Los Angeles
581901327	Mia	Phoenix
624656748	Daniel	New York
626986137	Noah	New York
697897081	Jayden	Los Angeles
710884273	Chloe	Houston
752855348	Alexander	Houston
832855660	William	Philadelphia
841448578	Olivia	Philadelphia
885557442	Emily	Philadelphia
938393052	Ethan	Los Angeles
959133623	Joshua	New York