## General

- Each group books a half-hour slot. We'll aim for a demo of 20 minutes; the remainder of the time will be a Q&A session.
    - Your project TA will give you a spreadsheet or poll to allow you to sign up for a demo time.  Scheduling is a very hard problem so please try your best to be accommodating.
- All group members are present and on time for the demo.
    - Penalties may be given to group members who are late or do not show up
- All group members should participate in the presentation.
- If the demo starts late, no extra time will be given.
    - In cases where the demo starts late because of an issue on the part of the teaching team, the group will be given the full 20 minutes to complete the demo.
- The group should be ready to go when the demo starts. That is, you should not use the first part of the demo to boot up your laptop and search for the files you need.
    - Be sure to also have SQL Plus ready so your TA can use it to double check the result of a given action in your application.
- The group is responsible for the demo flow and can choose the order to present the queries in. See the “Demo Checklist” section to ensure you will present all the required queries.
- Groups will not receive their demo grade right away. Your TAs will need time to look over the deliverables submitted on Canvas.

## Tentative Demo Workflow

1. The group will be asked to demonstrate that the code used in the demo has not been changed since the milestone 4 deadline.
    a. This can be done through Git (e.g., re-pulling from the repo or running something like `git log/git diff`) or recompiling the code. Your TA will let you know which method they prefer.
2. The TA will ask the group if anything listed in their formal specs (list of deliverables) does NOT work. This will save a bunch of demo time, and it will help the TA with marking the Canvas-submitted deliverables.
3. The group will be asked to re-create and repopulate the tables with their .sql file.
    a. It is also possible that the TA will ask the group to change the database location to another spot (e.g., the TA's own account).
4. The group will start demonstrating their queries in whatever order they choose.
    a. During the demonstration, the TA may ask to change input values (e.g., something like "Instead of inserting this value, please insert this other value instead"). The TA may also use something like SQL*Plus or equivalent to verify the result of the action.
5. In the Q&A session, the TA will ask the group technical questions. For example, "If I wanted to change "this" to "that", how would I do so and where in the code would that change have to be implemented?"

## Demo Checklist

This is a **tentative** marking scheme (we may adjust it), but it gives you an idea of approximately what we're after.

| Checklist | Points |
|---|---|
| **Overall Preparedness –** The team is ready to go at the beginning of the demo. Preparation files for the TA have been sent by the deadline.<br><br>The queries make sense in the context of the project and are not shoehorned in order to satisfy a particular rubric item. | 2 |
| **General Project Understanding -** Students are able to answer questions about their project. Individual deductions may apply if a student cannot answer questions about the project (the TA will take care of ensuring that the questions are directed appropriately). | 2 |
| **GUI**: The GUI doesn't need to be fancy, but a basic GUI is necessary. If the project does not have a GUI (e.g., it is run through command line or a command-type interface), a penalty of 40% on the total value of milestone 4 will be applied.<br><br>Do **not** provide text boxes for a user to type in a SQL query. Doing so will cause a penalty of 20% on the total value of milestone 4. | 1 |
| **Same Deliverables as What was Handed In? –** The group shows the TA that the deliverables haven't changed from what they handed in (e.g., timestamps). The TA may ask the group to run some Git commands to prove no changes have been made since the deadline. | 0.5 |
| **Insert Operation –** Provide an interface for the user to specify some input for the insert operation. You can choose which table the user should insert to but you cannot hardcode any values.<br><br>For example, you may choose to have a GUI that allows a fictional user to enter in details for a new employee. The table to insert to has been predetermined (the employee table) but the various details that are to be inserted should be dependent on the user of the GUI. | 0.5 |

| | |
|---|---|
| **Delete Operation –** Implement a cascade-on-delete situation. Provide an interface for the user to specify some input for the deletion operation. You can choose which table to delete from but you cannot specify which tuple to delete.<br><br>For example, you can allow your fictional user to delete an employee (so you predetermine the delete operation is for the Employee table) but you cannot predetermine which employee is to be deleted. | 0.5 |
| **Update Operation –** Provide an interface for the user to specify some input for the update operation. You can choose which table to update but the user should be able to specify which attribute(s) in that table to update the value(s) for.<br><br>For example, you can allow your fictional user to update information for an employee (so you predetermine the update operation is for the Employee table) but you cannot predetermine which employee is to be updated and what the new information is. | 1 |
| **Selection –** Create one query of this category and provide an interface for the user to specify the values of the selection conditions to be returned. Example:<br>`SELECT …`<br>`FROM   …`<br>`WHERE  Field1 = :Var1 AND Field2 > :Var2`<br><br>The user must have the ability to choose which table and which attributes to select on as well as the values of the selection conditions. I.e., you cannot predetermine which table the selection is for. | 1 |
| **Projection –** Create one query of this category, with 3-5 attributes in the projection condition, but not SELECT *. The user should be able to select t any number of attributes to project on.<br><br>Provide an interface for the user to execute this query. | 1 |
| **Join –** Create one query in this category, which joins at least 2 tables and performs a meaningful query, and provide an interface for the user to execute this query. The user must provide at least one value to qualify in the WHERE clause (e.g. join the Customer and the Transaction table to find the names and phone numbers of all customers who have purchased a specific item). | 1 |

| | |
|---|---|
| **Aggregation with Group By –** Create one query that requires the use of distinct aggregation (`min`, `max`, `average`, or `count` are all fine), and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query.<br><br>You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples). | 1 |
| **Aggregation with Having –** Create one meaningful query that requires the use of a HAVING clause, and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query.<br><br>You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples). | 1 |
| **Nested Aggregation with Group By –** Create one query that finds some aggregated value *for each group* (e.g., use a nested subquery, such as finding the average number of items purchased per customer, subject to some constraint) and provide an interface (i.e., HTML button, etc.) for the user to execute this query.<br><br>Some examples for the Sailors table are below. Note the difference between this query and the above Aggregation Query. You must use separate distinct queries for this criterion and the Aggregation Query (i.e., do not double dip).<br><br>You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples). | 1.5 |

| | |
|---|---|
| **Division–** Create one query of this category and provide an interface (i.e., HTML button, etc.) for the user to execute this query (e.g., find all the customers who bought all the items).<br><br>You can hardcode this query (i.e., you do not have to allow for user input) but you must display the tuples in the result within the GUI. The resulting tuples should be presented in a clear manner (e.g., if you use a table to display the results, there should be column headings that give context to the displayed tuples). | 1 |
| **Total** | /15 |

Lastly, here are some examples of nested aggregations:

```
SELECT    S.rating
FROM      Sailors S
WHERE     AVG(S.age) <= ALL ( SELECT    AVG(s2.age)
                              FROM      Sailors s2
                              GROUP BY  rating );


SELECT    S.rating, AVG (S.age) as avgage
FROM      Sailors S
GROUP BY  S.rating
HAVING    1 < ( SELECT  COUNT(*)
                FROM    SAILORS S2
                WHERE   S.rating = S2.rating );
```