- There is a transaction for voting, rollback is useful for testing
- 3 different workbench sessions to test concurrent transactions, there is an example of concurrent transactions on the class website

Phase A:

- Executive summary that includes a brief introduction to the project, high level description of the requirements, and a succinct discussion of your accomplishments (as well as any significant limitations).

Phase E

- Prototype, Development, and Testing. This consists of the following:
  • Develop (an) appropriate applications and/or user interface(s), using Python Jupyter Notebooks, which satisfy all functional user requirements.
  • SQL scripts for creating indices for the database application. Justify the reasons for creating any such indices.
  Your notebooks should have appropriate cells acquiring input from the user and providing output to the user (as needed).

Phase F

- Make a user's guide for the database application. The user guide could be integrated within your Jupyter notebook.

Complete phases E, F, and A, and submit your final report and all applicable materials for ALL the project phases. For this milestone you should submit a single archive/zip file that includes the following files:

1. **report.pdf** :: a PDF file with the contents of the written report of your project (see the Project Description about the content requirements of your report).
2. your (possibly updated) **dropAll.sql, createAll.sql, loadAll.sql** script files with SQL statements from Milestone 2.
3. **queryAll.sql** :: a SQL script file with sample SQL select statements, one for each of the queries B.1-B.10 (on page 5-6) required for the project. For each query that needs a value to to be provided input by the user, use MySQL scripting variables and clearly indicate that via appropriate documentation/comments in the script file, e.g.

   ```
   -- @x and @y are scripting variables
   SET @x = 123;
   SET @y = 'abc'
   SELECT @x, @y;
   ```

   Develop a Jupyter App with a section for each of the following:
   A. Activities: 1. a clerk creating a new ballot
   2. a folk registering to vote at a center
   3. a clerk modifying the availability period of a current ballot
   4. a voter casting a ballot while confirming a valid voting registration o Use a SQL transaction to ensure ACID properties (serializability) in the face of other concurrent operations or failures.
   5. a staff removing a folk (and all their associated information)
   B. Queries/Reports:
   I. 1. List the name, city, and email (any single email suffices) of all folks.

II. 2. List the city, state, and the number of residents of each city in Wonderland (skip cities with no residents) in decreasing order of number of residents.
III. 3. List each center together with its number of currently registered folks (include states with no inhabited places) in increasing alphabetical order of their zipcode.
IV. 4. Find the distinct identifiers and names of folks registered at a given voting center within a given time period.
V. 5. Find for a given month, the number of unique registrations at any voting center which is within 5 miles from the center of Megapolis, except for voting centers in a given (exclusion) list of voting centers.
VI. 6. Find the most popular voting center(s) (in terms of total number of registrations) in a given time period among those in a given city.
VII. 7. Find the unique folk that have valid registrations with every voting center on a given state.
VIII. 8. Find folks that registered at a voting center that is farther away than the voting center closest to their residence (break ties alphabetically by center's acronym).
IX. 9. Write a SQL function that returns the acronym of the voting center closest to the residence of a given folk among those whose operating period(s) contain a given date (return NULL if no such center exists; break ties at alphabetically by acronym).
X. 10. For a given ballot, construct a cross-tabulation of voting centers (acronym) as rows, unique ballot answers (options) as columns, and cells indicating number of cast votes at the row's center with the column's option.
XI. Populate your database with sufficient data to demonstrate successfully execution of the 15 operations/queries/reports above. For any of these operations/queries/reports, you may assume that the default start and end date of a time period is the current date of the system when the operation is invoked, e.g. NOW().

4. **transaction.sql** :: for operation A.4 (on page 5), which requires transactional support due to concurrency and failures, specify the most appropriate isolation level for it. Justify your choice. Provide a SQL script file with appropriate SQL transaction code for it. For each parameter whose value needs to be provided input by the user, use appropriate MySQL scripting variables and clearly indicate that via appropriate documentation/comments in the script file.
5. **README** :: a text file with instructions for running your application (eg. commands with relevant command-line arguments, parameters that depend on the mysql server and database that need to changed, etc.
6. one of more file(s) with your **IPython Jupyter notebooks** for your application.
7. any additional source code files you may developed for your application.

Triggers to be added to createAll.sql

- No overlapping operating times for a single VotingCenter in Center_times
- Validate if a person's date to register to vote in VotingRegistry falls within the operating periods of a VotingCenter
- Validate if a person's date to cast a vote via Ballot falls within the operating periods of the VotingCenter
- "Changes to a center's operating period(s) should be allowed only if it would not invalidate the (previously) valid registration of an already cast vote." This will be implemented later with triggers.
- "A ballot may be modified only if there are not any cast votes for the ballot. Ballots with no registrations may be deleted."
- "A folk can cast at most one vote for any single ballot. Attempts to vote more than once or without a currently valid registration are rejected."

For each trigger, use DELIMITER // notation to make sure that each trigger definition is not stopped prematurely by semi-colons. For violating triggers, we use SIGNAL SQLSTATE '45000' and a custom message to print a custom error message to the console, for debugging.

Now that the triggers are defined, adjust the loadAll.sql script to resolve any conflicts that violate triggers. Change start and end times for voting centers, fix FKRCs that reference center_times, then validate the rest of the CastedVotes such that they are available in both center_times ranges and Ballots ranges.

For query 5, "distances between places are computed using the standard Pythagorean Theorem," therefore use the euclidean distance formula of $\textbf{distance}^2 = (x_1-x_2)^2 + (y_1-y_2)^2$ POW() computes the square, SUBSTRING_INDEX extracts X and Y from the table. Use the same formula for query 8 and query 9 for finding distance.

For query 10, a cross tabulation is a table that summarizes the frequencies of occurrences at the intersection of variables, which thus shows how many votes for a single ballots are casted at every votingcenter that offered

For operation A.4, the most appropriate isolation level is serializable, there are 4 levels: Read Uncommitted, Read Committed, Repeatable Read, and Serializable. Serializable ensures that no dirty reads occur, and atomicity is maintained, which is essential for transactions.

Phase A

The wonderland database depicts a DBMS for folks and staff to manage and cast votes with ballots to voting centers. The project attempts to maintain ACID properties and parallel operations. The project requires a detailed entity-relationship model, a reduction to a relational model, and implementation of the model in SQL for all tables and relationships, and a jupyter notebook client application using python. The database must have triggers that validate certain constraints in regards to votes in the database being within certain time ranges for selected voting centers. The project must also have 10 detailed queries and reports that fulfill the

requirements of returned data/results. There must be a working transaction and transactional support, and the project must be able to clean itself after use. Hardware requirements are approximately 1 megabyte of storage for the scripts and client, installation of SQL, MySQL workbench, and jupyter notebook, low-end CPU and disk drive, etc. Accomplishments of this project include: relational database with triggers to validate constraints, relationships with foreign key reference constraints that define relationships and validate data, data insertions that follow the FKRCs and triggers (rules of the database), sufficient amount of data in the insert script such that the advanced queries are able to execute and return results, a client that connects to the database and inserts/modifies data, a cleanup script that drops the database's tables and data. Limitations include the complexity of the distance formula for some of the queries that require distance calculation from coordinates data for certain tables, performance and design of certain queries/triggers may not be optimized or studied long enough to make better, the user interface is not user-friendly, and errors or issues are not completely handled in the client and database scripts.