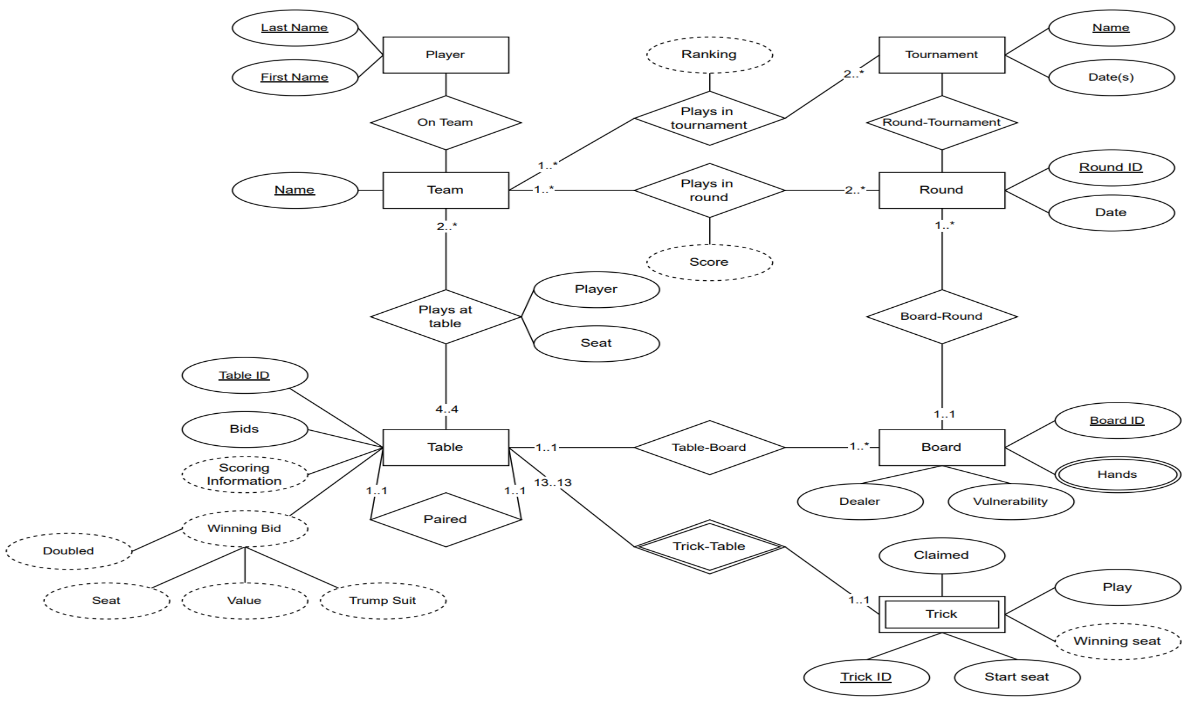
Alex Carney
Evan Sawyer
Corey Reinholdtsen
Behrooz Mansouri
COS 457 - Database Systems

Bridge Application Documentation

This document presents a comprehensive overview of our Bridge Application, an application created from a collection of many records of bridge tournaments worldwide. Traditionally, such data is recorded in .lin files, which isn't readable by humans. By parsing these files, importing them to a MySQL database, and designing a UI in Python, our team was able to present clean and organized data that provides significant context to bridge tournaments for anyone from the casual observer to the dedicated competitor.

__Phase One:__ Database Design

During this phase, our team began designing the ER diagram. We identified early on that consistent and communal communication amongst the team was necessary, and Discord was used for meetings where Corey lead the team in designing our early database graphs:



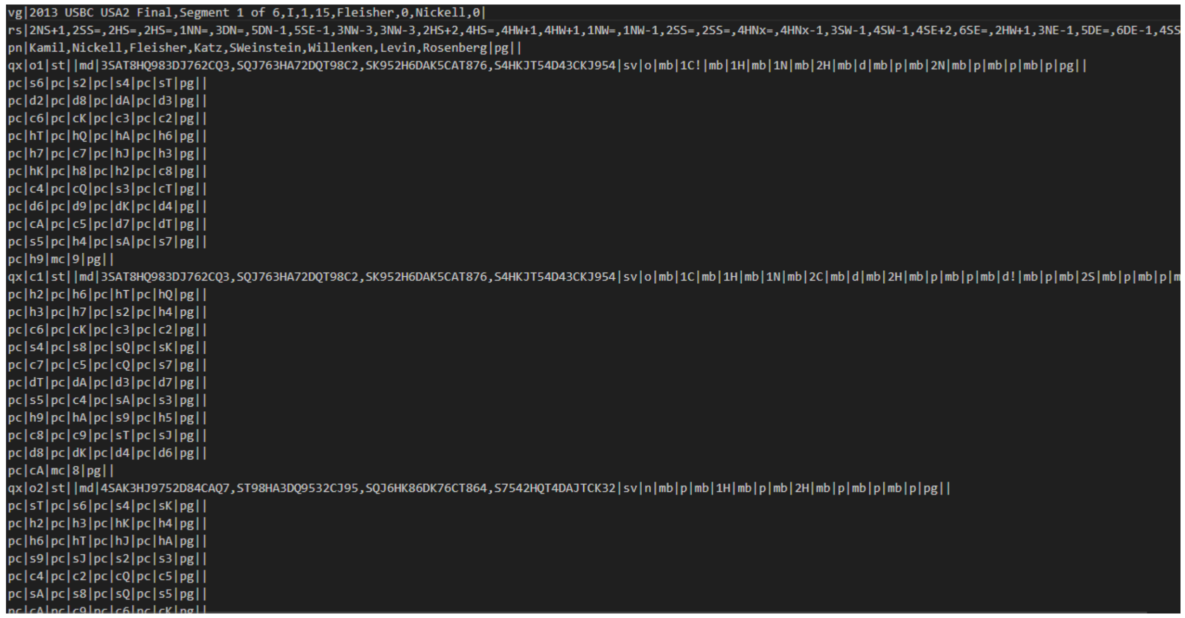
Phase Two

This phase is where we began using Git. This phase generally involved three components:

1. **Parsing data from .lin files to CSV**

Completed by Corey, this task involved parsing the .lin files into CSV files using Python.

For example, this is what a .lin file looks like:



```
vg|2013 USBC USA2 Final,Segment 1 of 6,I,1,15,Fleisher,0,Nickell,0|
rs|2NS+1,2SS=,2HS=,2HS=,1NN=,3DN=,5DN-1,5SE-1,3NW-3,3NW-3,2HS+2,4HS=,4HW+1,4HW+1,1NW=,1NW-1,2SS=,2SS=,4HNx=,4HNx-1,3SW-1,4SW-1,4SE+2,6SE=,2HW+1,3NE-1,5DE=,6DE-1,4SS
pn|Kamil,Nickell,Fleisher,Katz,SWeinstein,Willenken,Levin,Rosenberg|pg||
qx|o1|st||md|3SAT8HQ983DJ762CQ3,SQJ763HA72DQT98C2,SK952H6DAK5CAT876,S4HKJT54D43CKJ954|sv|o|mb|1C!|mb|1H|mb|1N|mb|2H|mb|d|mb|p|mb|2N|mb|p|mb|p|mb|p|pg||
pc|s6|pc|s2|pc|s4|pc|sT|pg||
pc|d2|pc|d8|pc|dA|pc|d3|pg||
pc|c6|pc|cK|pc|c3|pc|c2|pg||
pc|hT|pc|hQ|pc|hA|pc|h6|pg||
pc|h7|pc|c7|pc|hJ|pc|h3|pg||
pc|hK|pc|h8|pc|h2|pc|c8|pg||
pc|c4|pc|cQ|pc|s3|pc|cT|pg||
pc|d6|pc|d9|pc|dK|pc|d4|pg||
pc|cA|pc|c5|pc|d7|pc|dT|pg||
pc|s5|pc|h4|pc|sA|pc|s7|pg||
pc|h9|mc|9|pg||
qx|c1|st||md|3SAT8HQ983DJ762CQ3,SQJ763HA72DQT98C2,SK952H6DAK5CAT876,S4HKJT54D43CKJ954|sv|o|mb|1C|mb|1H|mb|1N|mb|2C|mb|d|mb|2H|mb|p|mb|p|mb|d!|mb|p|mb|2S|mb|p|mb|p|m
pc|h2|pc|h6|pc|hT|pc|hQ|pg||
pc|h3|pc|h7|pc|s2|pc|h4|pg||
pc|c6|pc|cK|pc|c3|pc|c2|pg||
pc|s4|pc|s8|pc|sQ|pc|sK|pg||
pc|c7|pc|c5|pc|cQ|pc|s7|pg||
pc|dT|pc|dA|pc|d3|pc|d7|pg||
pc|s5|pc|c4|pc|sA|pc|s3|pg||
pc|h9|pc|hA|pc|s9|pc|h5|pg||
pc|c8|pc|c9|pc|sT|pc|sJ|pg||
pc|d8|pc|dK|pc|d4|pc|d6|pg||
pc|cA|mc|8|pg||
qx|o2|st||md|4SAK3HJ9752D84CAQ7,ST98HA3DQ9532CJ95,SQJ6HK86DK76CT864,S7542HQT4DAJTCK32|sv|n|mb|p|mb|1H|mb|p|mb|2H|mb|p|mb|p|mb|p|pg||
pc|sT|pc|s6|pc|s4|pc|sK|pg||
pc|h2|pc|h3|pc|hK|pc|h4|pg||
pc|h6|pc|hT|pc|hJ|pc|hA|pg||
pc|s9|pc|sJ|pc|s2|pc|s3|pg||
pc|c4|pc|c2|pc|cQ|pc|c5|pg||
pc|sA|pc|s8|pc|sQ|pc|s5|pg||
pc|cA|pc|c9|pc|c6|pc|cK|pg||
```

As you can see, this format is not human readable. Using the parser.py script in the data/code directory, Corey was able to derive necessary data and reject poorly formatted files.

2. **Creating the database**

Led by Evan, a MySQL 8.0 server was created using MySQL Workbench 8.0 CE. The following tables were constructed:

Table Name	Primary Key	Foreign Key	Context
Round	RoundID	TeamOneName, TeamTwoName	In a bridge tournament, a round refers to a set of boards played between two teams.
Board	BoardID	RoundID	A board is a specific set of 4 hands that is played.
TableEntity[1]	TableID	PairedTableID,	This is a session of play involving 4 players and

[1] Name was chosen due to “table” being a reserved word in SQL.

		BoardID	matched with a paired table.
Trick	Weak Entity	TableID	A trick is a set of four cards, one played by each player in turn, starting with the person who won the last trick or the declarer. The player with the best card value wins a trick.
Hand	Weak Entity	BoardID	A hand refers to the set of 13 cards given to each player at the beginning of a game.
PlaysTable	Look up	TableName, PlayerName	Stores players at table and what at position.
Player	Name, TeamName[2]	TeamName	Tracks players
Team	Name	N/A	Tracks teams

3. **Developing queries**

Led by Evan, the following queries were developed for this phase:

- BoardSearch → searches for board by ID
- TricksForTable → search for all tricks at given table
- TournamentName → returns tables given a tournament
- BoardsInTournament → searches for boards in a given tournament
- DealerSearch → searches for boards with a given dealer
- EndingBidSearch → searches for tables by winning bid
- GetSeat → searches for a seat for a given player on a given team
- HCPSearchInRange → searches for hands with varying numbers of high cards.[3]
- PlayerSearchByTeam → searches for players in a given team
- PlayerSearch → searches for a player by name and team
- RawScoreSearch → searches for a table with varying scores
- RoundSearch → searches for rounds in a given tournament
- StartingBidSearch → searches for the first bid for a table
- SuitSearch → searches for hands with a specific suit combination
- TableInTournament → searches for tables in a given tournament
- TotalTricksByPlayer → searches for the number of tricks for a given player
- VulnerabilitySearch → searches for boards containing a vulnerability[4]

Phase Three

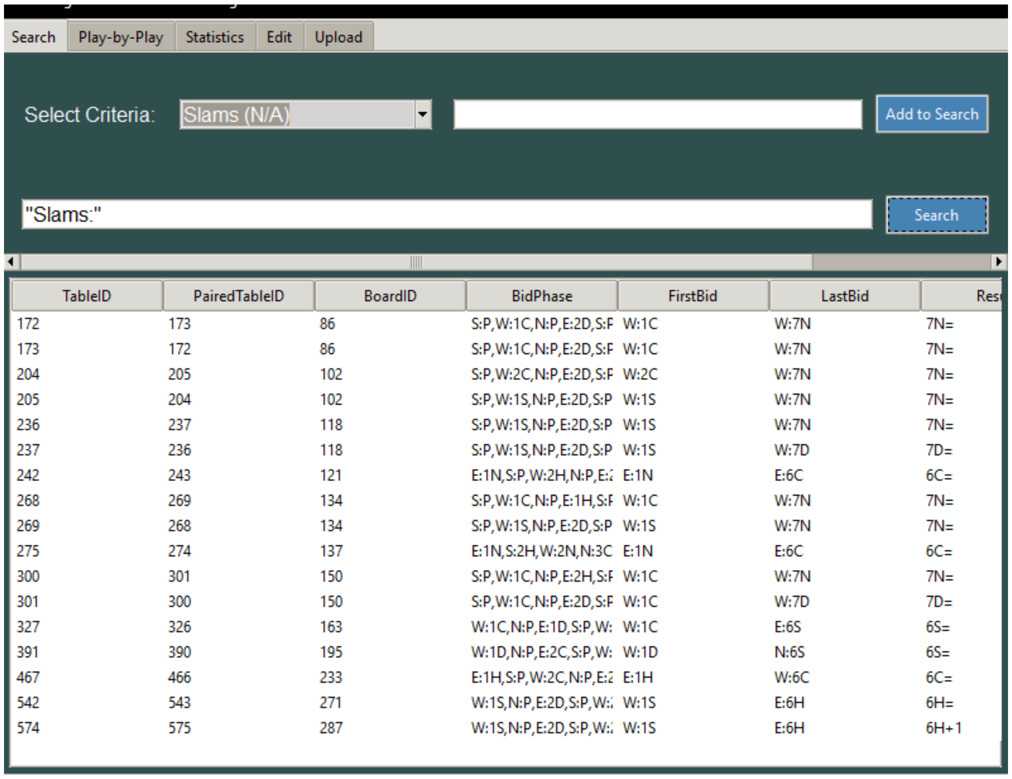
During our final phase, we continued developing our queries while also developing a UI and a database connection using Python. Due to the complexity of this phase, a much more significant

[2] This is a composite primary key.
[3] High cards are 10, J, Q, K, A
[4] In bridge, a vulnerability is a situation where a team receives higher penalties for failing to make their contract, but also higher rewards for achieving certain contracts.

time was spent on Discord, where we engaged in team discussions on things such as high-level design patterns, query debugging and merge conflicts. Overall this phase involved the following:

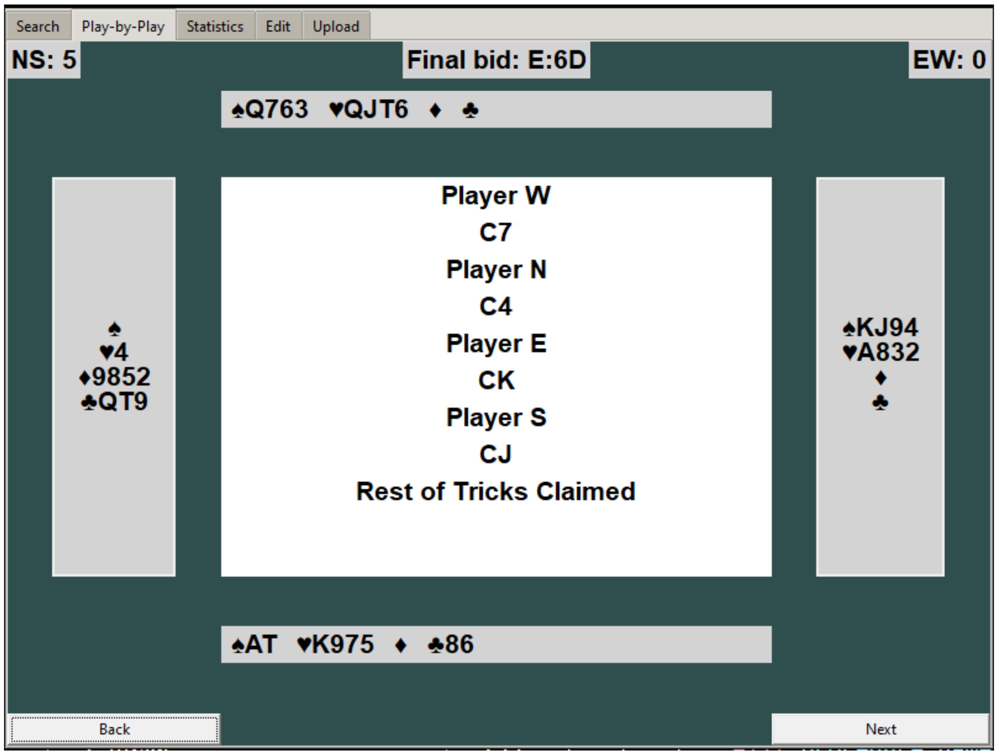
1. **Creating UI**

Led by Evan, our UI was constructed using Python's Tkinter. Users are able to run specific searches, with the results returned in a table. For example, this is what a search for slams returns:



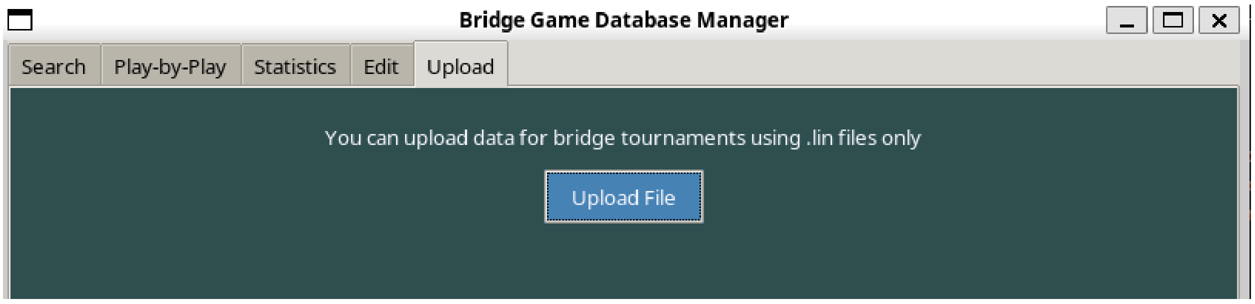
TableID	PairedTableID	BoardID	BidPhase	FirstBid	LastBid	Res
172	173	86	S:P,W:1C,N:P,E:2D,S:P	W:1C	W:7N	7N=
173	172	86	S:P,W:1C,N:P,E:2D,S:P	W:1C	W:7N	7N=
204	205	102	S:P,W:2C,N:P,E:2D,S:P	W:2C	W:7N	7N=
205	204	102	S:P,W:1S,N:P,E:2D,S:P	W:1S	W:7N	7N=
236	237	118	S:P,W:1S,N:P,E:2D,S:P	W:1S	W:7N	7N=
237	236	118	S:P,W:1S,N:P,E:2D,S:P	W:1S	W:7D	7D=
242	243	121	E:1N,S:P,W:2H,N:P,E:2	E:1N	E:6C	6C=
268	269	134	S:P,W:1C,N:P,E:1H,S:P	W:1C	W:7N	7N=
269	268	134	S:P,W:1S,N:P,E:2D,S:P	W:1S	W:7N	7N=
275	274	137	E:1N,S:2H,W:2N,N:3C	E:1N	E:6C	6C=
300	301	150	S:P,W:1C,N:P,E:2H,S:P	W:1C	W:7N	7N=
301	300	150	S:P,W:1C,N:P,E:2D,S:P	W:1C	W:7D	7D=
327	326	163	W:1C,N:P,E:1D,S:P,W:	W:1C	E:6S	6S=
391	390	195	W:1D,N:P,E:2C,S:P,W:	W:1D	N:6S	6S=
467	466	233	E:1H,S:P,W:2C,N:P,E:2	E:1H	W:6C	6C=
542	543	271	W:1S,N:P,E:2D,S:P,W:	W:1S	E:6H	6H=
574	575	287	W:1S,N:P,E:2D,S:P,W:	W:1S	E:6H	6H+1

If the user clicks on a specific entry, they can then navigate over to other tabs and see additional info. For example, if a user clicks on "Play-by-Play", they can view the details for each hand:



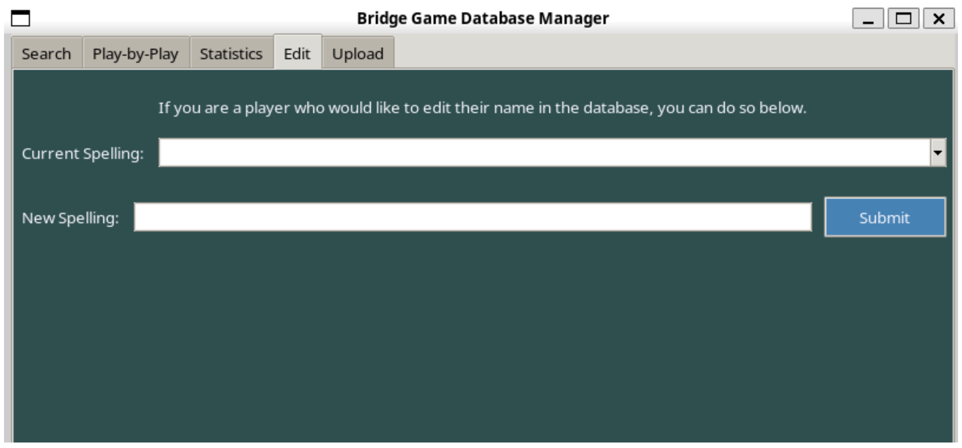
Player W
C7
Player N
C4
Player E
CK
Player S
CJ
Rest of Tricks Claimed

Led by Alex, two additional tabs were added.. “Upload”, which allows users to put their own .lin files in the database:



You can upload data for bridge tournaments using .lin files only

And “Edit”, which allows users to edit a player’s name in the database:



If you are a player who would like to edit their name in the database, you can do so below.

Current Spelling:

New Spelling:

2. Creating DB Connector

Led by Alex, we created a python class that connects the interface to the database. Using Python's MySQL Connector, the class uses an OOP design with the following criteria:

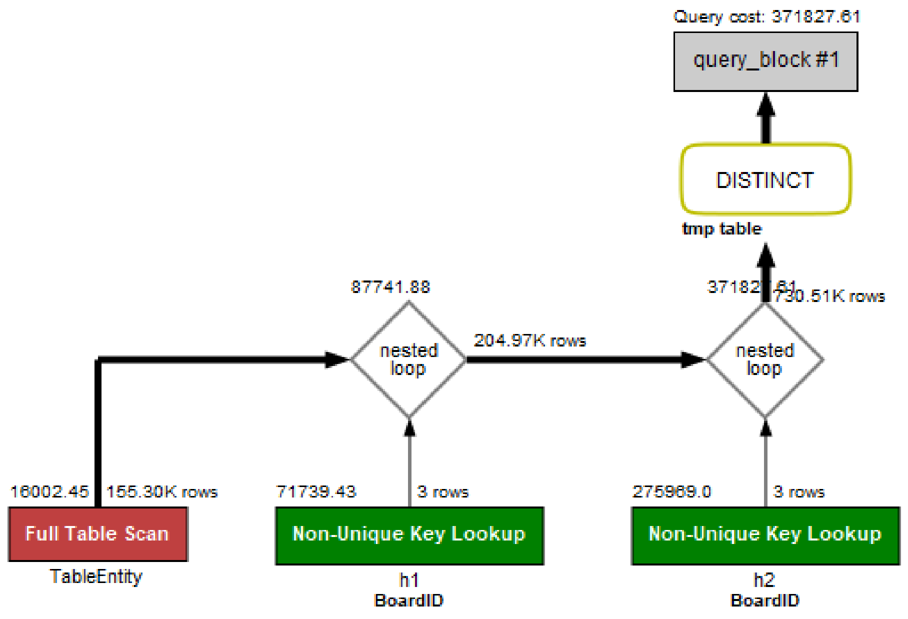
→ Constructor must initialize a connection that is disconnected when the object is destroyed
→ Queries must have the ability to turn auto-commit on/off
→ Errors with long messages customized for readability
→ Queries able to accept 0, 1 or many parameters.
→ Query results returned as a dictionary for easy communication with UI

3. Query Evaluation Plan Analysis

The query we chose to analyze the evaluation plan for is the following:

```
WITH Team_HCP(BoardID, Val, Position) AS
(SELECT h1.BoardID, h1.HighCardPoints + h2.HighCardPoints, h1.Position
FROM BridgeDB.Hands AS h1 JOIN BridgeDB.Hands AS h2 ON h1.BoardID = h2.BoardID
WHERE h1.Position != h2.Position AND h1.Position % 2 = h2.Position % 2
AND h1.HighCardPoints + h2.HighCardPoints <= 25)
SELECT DISTINCT TableID, PairedTableID, BoardID, BidPhase, FirstBid, LastBid,
Result, RawScore
FROM Team_HCP
NATURAL JOIN (SELECT * FROM BridgeDB.TableEntity WHERE TableEntity.RawScore >=
400) AS t1
WHERE ((Team_HCP.Position % 2 = 0 AND SUBSTRING(t1.LastBid, 1, 1) IN ('E',
'W'))
OR (Team_HCP.Position % 2 = 1 AND SUBSTRING(t1.LastBid, 1, 1) IN ('N', 'S')))
```

This query produced the following evaluation plan (Visualized using the MySQL Workbench visual explain feature):



In Tabular form, the query evaluation plan is as follows:

id	select_type	table	type	possible_ keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	TableEntity	ALL	BoardID	NULL	NULL	NULL	155297	33.33	Using where; Using temporary
1	SIMPLE	h1	ref	BoardID	BoardID	4	BridgeDB.Table Entity.BoardID	3	100.00	Using index condition
1	SIMPLE	h2	ref	BoardID	BoardID	4	BridgeDB.Table Entity.BoardID	3	90.00	Using index condition; Using where; Distinct

This evaluation plan first performs a search on TableEntity using a where clause. This is where most of the cost for this evaluation plan comes from, as both where clauses on TableEntity (not including the joins) are on non-indexed columns (i.e. LastBid and RawScore). The reason TableEntity is scanned first is because there are around half as many entries in TableEntity as Hands, meaning that there will be less memory overhead for temporary tables. The result from the full scan is then joined with h1 on BoardID, which is not very intensive, as BoardID is indexed, which means its lookup time is greatly reduced. The result of this join is then joined with h2, where another where clause is applied (because both tables used in the subquery have been joined, and MySQL can therefore calculate h1.HighCardPoints + h2.HighCardPoints). After this final join, the resulting table is sorted on TableID and duplicates are removed.

What we Learned/Contributions:

Alex

Contributions:

- Worked with team on ER diagrams
- Assisted with query debugging and conversion to stored procedures
- Wrote DB communicator
- Worked on UI (Edit, Upload, and drop down menu)
- This write up

What I Learned:

This project helped me understand the overall process of database design, especially the implementation part. Prior to this course, I only had experience with high-level database connections that were set up for me by a DBA or person with more experience. I now feel confident to construct large database systems alone, which can be useful at small companies where employees need to take on multiple roles. I also learned a bit more involving Git than previous, as I had only done simple commands previously. I believe I have gotten much faster at sorting merge conflicts and coordinating between branches. Finally, and perhaps most importantly, I became much more confident in advanced SQL expressions while debugging queries during my development of the DB Communicator.

Evan

Contributions:

- Assisted in design of ER diagram and initial database design
- Git repository setup and instructions to team on best practices
- Implemented first database by creating create_table.sql
- Created crawler.py, a python for collecting the .lin files
- Imported CSVs into database and resolved conflicts regarding data constraints
- Created queries for part 2 in collaboration with Corey
- Designed and implemented python interface.
- Managed remote server and database used for testing

What I learned:

Before taking this course, I had only introductory experience with databases from my use of Microsoft Access and writing basic queries for work. This project has taught me many skills that I believe will shape my career. I have developed a strong foundation of SQL, database design principles, implementation of middleware, and common data practices just to name a few. I am glad we picked such an exciting project using real world data.

Corey

Contributions:

- Collaborated with team members on ER diagram design
- Created ER diagram legend/glossary
- Created linparser.py and parser_classes.py to convert source files to CSV/SQL insert
- Helped resolve data insertion/inconsistency issues on initial insertion
- Created README file in data directory (data generation/insertion instructions)
- Collaborated with team members on Part 2 documentation
- Tested and rewrote several queries for Part 2 turn-in.
- Converted several queries into procedures
- Wrote both triggers for part 3
- Wrote method for inserting new .lin file directly into database (transaction implemented in Python)
- Wrote stored procedure for updating a player name (with transaction/rollback)
- Wrote query evaluation analysis

What I Learned:

Throughout this project, I learned the details of each step of database design and development, from ER diagram design (and what to avoid), to data acquisition and the difficulties of cleaning dirty data (~1/3 of our input files had to be thrown out due to inconsistency), to the details of writing acceptably efficient queries in SQL. Before taking this course, I had no experience in any form of SQL, nor any experience coding in Python, nor any experience with Git. Needless to say, the first few weeks had a steep learning curve. After this course, I feel more confident collaborating remotely on a team, and feel confident enough to construct and maintain complex databases.