Chess Database
INFOH417 Database System Architecture 2023/24

Maxime Schoemans and Mahmoud SAKR

maxime.schoemans@ulb.be, mahmoud.sakr@ulb.be

In this project your team will create a PostgreSQL extension for storing and retrieving chess games. There exist multiple notations for encoding complete chess games as well as for encoding the board status at a certain move in the game. In this project you shall base your solution on the following notations:

- Portable Game Notation (PGN), used to record full games. https://en.wikipedia.org/wiki/Portable_Game_Notation
  You will only use a subset of this notation. Nevertheless it may help if you learn it to help understanding.
- Standard Algebraic Notation (SAN), only the moves part of the PGN. https://en.wikipedia.org/wiki/Algebraic_notation_(chess)
- Forsyth–Edwards Notation (FEN), used to store board states. https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation
- The website with lots of PGN files: https://www.pgnmentor.com/files.html

Your extension will implement data types, functions and indexes as follows:
Data types: chessgame, chessboard. The input/output for your types shall use SAN and FEN notation for chess-game and chess-board respectively.
Functions:

- getBoard(chessgame, integer) -> chessboard: Return the board state at a given half-move (A full move is counted only when both players have played). The integer parameter indicates the count of half moves since the beginning of the game. A 0 value of this parameter means the initial board state, i.e.,(rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1).

- getFirstMoves(chessgame, integer) -> chessgame: Returns the chessgame truncated to its first N half-moves. This function may also be called getOpening(...). Again the integer parameter is zero based.
- hasOpening(chessgame, chessgame) -> bool: Returns true if the first chess game starts with the exact same set of moves as the second chess game. The second parameter should not be a full game, but should only contain the opening moves that we want to check for, which can be of any length, i.e., m half-moves.
- hasBoard(chessgame, chessboard, integer) -> bool: Returns true if the chessgame contains the given board state in its first N half-moves. Only compare the state of the pieces and not compare the move count, castling right, en passant pieces, ...

Indices:

- Implement an index to support the hasOpening predicate. The index should then be defined on the chessgame type. One idea for a suitable index would be the B-tree, since a chessgame can be seen as a string, with a total order property. But you are free to extend other postgres index structures.
- Implement an index to support the hasBoard predicate. For this you may consider extending the GIN index for the chessgame type. The reason is that a chessgame can also be seen as a sequence of chessboard states. GIN would allow for indexing the individual states per chessgame. But again you are free to implement other index structures that you think could fit.

For chess operations, such as manipulating the notation, and performing moves, etc, we propose that you should use the c chess library: https://codeberg.org/drummyfish/smallchesslib. It is a header file library, meaning that you only need to copy the header file in your project folder, and in the make command. The only required include file is smallchesslib.h, but the other files can be interesting to see examples on how to use it.

A link on the UV page of the course will be created for submitting your extension. Per group only one submission must be made. You need to submit one zip file including:

1. The source code of your extension. Normally your extension will need to be written in C. But if you manage to do it with some of the other accepted languages for postgres extensions, you may do so. But please note that we may not be able to support you if you have problems/questions about using languages other than C.
2. An SQL file for testing your extension, along with the query plans and short snippets of the results that you obtained during your tests.
3. A presentation that explains your work, focusing on the technical details and even samples of the important code. We will publish in UV slots for booking your project evaluation, close to the end of the semester. Then you shall use this presentation for explaining your work.

Evaluation

The evaluation jury consists of the two course instructors. The grading will consider the following factors:

- Implementing the two data types (10 points). Favor an internal representation which optimizes the storage space, without jeopardizing the query time.
- Implementing the functions and predicates (10 points)
- Implementing the two indexes (20 points). Favor implementations that minimize the false positives of the index query. Optimally the predicates will be answered by an index-only scan, i.e., without needing to actually execute the predicate on the tuples that the index returns. Also favor implementations that make the index access transparent to user queries. In other words, it shouldn't be the case that the user needs to change her query in order to make use of the index.

Even if you have problems in 'make' and 'create extension', for some or all of the points above you still can collect points based on your theoretical

design and based on your code. A penalty of 25% will be deducted for lacking the actual running.

ANNEX

A few example chess games in SAN:

- Fool's mate, fastest checkmate in chess:
  1. f3 e5 2. g4 Qh4#
- Game of the century (https://en.wikipedia.org/wiki/The_Game_of_the_Century_(chess)):
  1. Nf3 Nf6 2. c4 g6 3. Nc3 Bg7 4. d4 O-O 5. Bf4 d5 6. Qb3 dxc4 7. Qxc4 c6 8. e4 Nbd7 9. Rd1 Nb6 10. Qc5 Bg4 11. Bg5 Na4 12. Qa3 Nxc3 13. bxc3 Nxe4 14. Bxe7 Qb6 15. Bc4 Nxc3 16. Bc5 Rfe8+ 17. Kf1 Be6 18. Bxb6 Bxc4+ 19. Kg1 Ne2+ 20. Kf1 Nxd4+ 21. Kg1 Ne2+ 22. Kf1 Nc3+ 23. Kg1 axb6 24. Qb4 Ra4 25. Qxb6 Nxd1 26. h3 Rxa2 27. Kh2 Nxf2 28. Re1 Rxe1 29. Qd8+ Bf8 30. Nxe1 Bd5 31. Nf3 Ne4 32. Qb8 b5 33. h4 h5 34. Ne5 Kg7 35. Kg1 Bc5+ 36. Kf1 Ng3+ 37. Ke1 Bb4+ 38. Kd1 Bb3+ 39. Kc1 Ne2+ 40. Kb1 Nc3+ 41. Kc1 Rc2#
- Oldest recorded chess game from 1475 (https://www.chess.com/forum/view/general/the-oldest-chess-game):
  1. e4 d5 2. exd5 Qxd5 3. Nc3 Qd8 4. Bc4 Nf6 5. Nf3 Bg4 6. h3 Bxf3 7. Qxf3 e6 8. Qxb7 Nbd7 9. Nb5 Rc8 10. Nxa7 Nb6 11. Nxc8 Nxc8 12. d4 Nd6 13. Bb5+ Nxb5 14. Qxb5+ Nd7 15. d5 exd5 16. Be3 Bd6 17. Rd1 Qf6 18. Rxd5 Qg6 19. Bf4 Bxf4 20. Qxd7+ Kf8 21. Qd8#

And some example board states in FEN:

- Starting board state:
  rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
- Draw by 50-move rule in a pawn endgame:
  8/5k2/3p4/1p1Pp2p/pP2Pp1P/P4P1K/8/8 b - - 99 50
- A random FEN from a Magnus Carlsen game:
  r4r1k/ppp1qpb1/7p/4pRp1/1PB1P3/1QPR3P/P4P2/6K1 w - - 4 26

Example queries to help illustrate some of the functions of the extension:

How many games had the given board state at any time during the game:

```
SELECT count(*)
FROM games
WHERE hasboard(game,
   'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1', 10);
```

How many games started with the given sequence of moves:

```
SELECT count(*)
FROM games
WHERE hasopening(game, '1. e4 e5 2. Nf3 Nf6 3. d3');
```

Which games have the same 10 first half-moves as any of the games stored in table favoriteGames:

```
SELECT g.game
FROM games g, favoriteGames f
WHERE hasopening(g.game, getFirstMoves(f.game, 10));
```

Documentation related to extneding PostgreSQL. It is an important webpage, which links to most of the info needed for the implementation of this extension:
https://www.postgresql.org/docs/current/extend.html