INFOH417 Database System Architecture

PROJECT: POSTGRESQL EXTENSION FOR CHESS GAMES

Prof. Mahmoud SAKR and Mr. Maxime Schoemans

GROUP MEMBERS

- Min Zhang 000586970
- Yutao Chen 000585954
- Ziyong Zhang 000585736
- Jintao Ma 000586557

Slides at reveal.js format

Code at Github

PROJECT OVERVIEW

PROJECT REQUIREMENTS

- Create a PostgreSQL extension to store and retrieve chess games. Including:
- 2 data types.
- 4 functions.
- 2 index types(Btree & GIN).

STRATEGY

Follow PostgreSQL Extension development documentation and utilize SmallChessLib when possible.

PROJECT COMPLETION SUMMARY

Data types (100%)

chessgame Finished
chessboard Finished

Functions (100%)

getBoard Finished
getFirstMoves Finished
hasOpening Finished
hasBoard Finished

B-tree index (100%)

Support the hasOpening predicate Finished

GIN index (99%)

Support the hasBoard predicate In progress

DATA TYPES

DATA TYPES

chessboard

We use `SCL_board` as the data type of chessboard as the `SCL_boardFromFEN` and `SCL_boardToFEN` are implemented in smallchesslib.h

CHESSBOARD DATA TYPE

```
-- Create chessboard datatype
CREATE TYPE chessboard (
internallength = 1024,
input          = chessboard_in,
output         = chessboard_out
);
CREATE OR REPLACE FUNCTION chessboard_in(cstring)
RETURNS chessboard
AS 'MODULE_PATHNAME','chessboard_in'
LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;

 CREATE OR REPLACE FUNCTION chessboard_out(chessboard)
RETURNS cstring
AS 'MODULE_PATHNAME','chessboard_out'
LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE:
```

CHESSBOARD_IN

```
1  PG_FUNCTION_INFO_V1(chessboard_in);
2    Datum chessboard_in(PG_FUNCTION_ARGS) {
3      char *fen_str = PG_GETARG_CSTRING(0);
4      // Allocate memory for SCL_Board
5      SCL_Board *result_board = palloc0(sizeof(SCL_Board));
6
7      // Transfer FEN to SCL_Board
8      if (!SCL_boardFromFEN(*result_board, fen_str))
9      {
10       // If conversion fails, throw an error
11       ereport(ERROR,
12           (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
13             errmsg("invalid input syntax for FEN: \"%s\"", fen_str)));
14     }
15     // Return the chessboard
```

CHESSBOARD_OUT

```
1  PG_FUNCTION_INFO_V1(chessboard_out);
2    Datum chessboard_out(PG_FUNCTION_ARGS) {
3      SCL_Board *board = (SCL_Board *) PG_GETARG_POINTER(0);
4
5      char fen_str[SCL_FEN_MAX_LENGTH];
6
7      // Convert the board state to a FEN string using the SCL_boardToFEN fu
8      if (!SCL_boardToFEN(*board, fen_str))
9      {
10       // If conversion fails, throw an error
11       ereport(ERROR,
12           (errcode(ERRCODE_INTERNAL_ERROR),
13             errmsg("failed to convert SCL_Board to FEN")));
14     }
15     // Return the fen str
```

DATA TYPES

chessgame

CHESSGAME DATA TYPE

While `SCL_Game` is available, we've opted for `SCL_Record` due to:

- `SCL_Game` is more complex and not necessary for our needs.
- Lack of a direct conversion function from PGN to `SCL_Game`.
- `SCL_Record` efficiently stores game halfmoves, allowing restoration to any game state.
- Our choice aligns with our indexing strategy, making `SCL_Record` ideal for our chessgame data type.

CHESSGAME DATA TYPE

```
1  -- Create chessgame datatype
2  CREATE TYPE chessgame (
3    internallength = 1024,
4    input          = chessgame_in,
5    output         = chessgame_out
6  );
7  CREATE OR REPLACE FUNCTION chessgame_in(cstring)
8  RETURNS chessgame
9  AS 'MODULE_PATHNAME','chessgame_in'
10 LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
11
12  CREATE OR REPLACE FUNCTION chessgame_out(chessgame)
13 RETURNS cstring
14 AS 'MODULE_PATHNAME','chessgame_out'
15 LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

CHESSGAME_IN

```
1  PG_FUNCTION_INFO_V1(chessboard_in);
2                  PG_FUNCTION_INFO_V1(chessgame_in);
3                  Datum chessgame_in(PG_FUNCTION_ARGS) {
4                    // Retrieve the input SAN string
5                    char *san_str = PG_GETARG_CSTRING(0);
6
7                    text *record = cstring_to_text(san_str); //highlight
8
9                    PG_RETURN_TEXT_P(record);
10                 }
11
```

CHESSGAME_OUT

```
1  PG_FUNCTION_INFO_V1(chessboard_out);
2                PG_FUNCTION_INFO_V1(chessgame_out);
3                Datum chessgame_out(PG_FUNCTION_ARGS) {
4
5                  text *record_txt = PG_GETARG_TEXT_P(0);
6
7                  char *san_str = text_to_cstring(record_txt); //highlight
8
9                  PG_RETURN_CSTRING(san_str);
10
11               }
```

FUNCTIONS IMPLEMENTATION

Our approach to implementing the 4 required functions involves:

- Writing the logic in C language.
- Invoking these C functions within PostgreSQL.

GET BOARD FUNCTION

1. Invoke `SCL_recordFromPGN` to retrieve `SCL_Record` from PGN.
2. Use `SCL_Record` with `SCL_recordApply` to get board state after N halfMoves.
3. Convert board state to FEN with `SCL_boardToFEN`.

GETBOARD FUNCTION

```
1 char* getBoard_internal(text* san_text, int halfMoves) {
2   // getBoard_internal
3 }
4
5 PG_FUNCTION_INFO_V1(getBoard);
6 Datum getBoard(PG_FUNCTION_ARGS) {
7   // getBoard
8 }
```

GETBOARD FUNCTION

```
1  //  Return the board state at a given half-move
2  char* getBoard_internal(text* san_text, int halfMoves){
3
4    char *san_str = text_to_cstring(san_text);
5
6    SCL_Record record;
7
8    SCL_Board *board;
9
10   char *fenstr;
11
12   SCL_recordFromPGN(record,san_str); //highlight
13
14   board = palloc0(sizeof(SCL_Board));
15
```

GETBOARD FUNCTION

```
5
6   SCL_Record record;
7
8   SCL_Board *board;
9
10  char *fenstr;
11
12  SCL_recordFromPGN(record,san_str); //highlight
13
14  board = palloc0(sizeof(SCL_Board));
15
16  SCL_recordApply(record, *board, halfMoves); //highlight
17
18  fenstr = palloc(70 * sizeof(char));//allocation of size 70 char(FEN nota
19
```

GETBOARD FUNCTION

```
 9
10   char *fenstr;
11
12   SCL_recordFromPGN(record,san_str); //highlight
13
14   board = palloc0(sizeof(SCL_Board));
15
16   SCL_recordApply(record, *board, halfMoves); //highlight
17
18   fenstr = palloc(70 * sizeof(char));//allocation of size 70 char(FEN nota
19
20   strcpy(fenstr,"");
21
22   SCL_boardToFEN(*board, fenstr);//highlight
23
```

GETBOARD FUNCTION

```
15
16    SCL_recordApply(record, *board, halfMoves); //highlight
17
18    fenstr = palloc(70 * sizeof(char));//allocation of size 70 char(FEN nota
19
20    strcpy(fenstr,"");
21
22    SCL_boardToFEN(*board, fenstr);//highlight
23
24    return fenstr;
25  }
26
27  //Function to return the board state at a given halfMoves
28  PG_FUNCTION_INFO_V1(getBoard);
29  Datum getBoard(PG_FUNCTION_ARGS) {
```

GETBOARD FUNCTION

```
31   text *san_text = PG_GETARG_TEXT_PP(0);
32
33   uint16_t halfMoves = PG_GETARG_UINT16(1);
34
35   char *result = NULL;
36
37   // if input halfMoves greater than the recoardlength, throw ERROR
38   if(errorNumCheck(san_text,halfMoves) == true) //highlight
39   {
40     result = getBoard_internal(san_text,halfMoves); //highlight
41   }
42
43   PG_RETURN_CSTRING(result);
44 }
```

GETFIRSTMOVES FUNCTION

Due to the absence of `SCL_recordToPGN` in smallchesslib:

- A string parsing mechanism is integrated.
- It returns a string of the first N halfMoves.

GET_FIRST_MOVES FUNCTION

```
1 char* get_first_moves_internal(const char* chessgame, int halfMoves) {
2   // get_first_moves_internal
3 }
4
5 PG_FUNCTION_INFO_V1(getFirstMoves);
6 Datum getFirstMoves(PG_FUNCTION_ARGS) {
7   // getBoard
8 }
```

GET_FIRST_MOVES FUNCTION

```c
char* get_first_moves_internal(const char *chessgame, int halfMoves) {
  int N_all = halfMoves + halfMoves/ 2 + halfMoves % 2;
  // Variables to store the result
  char* result = NULL;
  int resultLength = 0;
  // Counter for half moves
  int halfMovesCounter = 0;
  // Pointer to the current position in the PGN
  const char* currentPos = chessgame;

  while (*currentPos != '\0' && halfMovesCounter < N_all) {
    // Skip spaces and move to the next character
    while (*currentPos == ' ') {
      currentPos++;
    }
```

GET_FIRST_MOVES FUNCTION

```c
char* get_first_moves_internal(const char *chessgame, int halfMoves) {
  int N_all = halfMoves + halfMoves/ 2 + halfMoves % 2;
  // Variables to store the result
  char* result = NULL;
  int resultLength = 0;
  // Counter for half moves
  int halfMovesCounter = 0;
  // Pointer to the current position in the PGN
  const char* currentPos = chessgame;

  while (*currentPos != '\0' && halfMovesCounter < N_all) {
    // Skip spaces and move to the next character
    while (*currentPos == ' ') {
      currentPos++;
    }
```

GET_FIRST_MOVES FUNCTION

```
 4   char* result = NULL;
 5   int resultLength = 0;
 6   // Counter for half moves
 7   int halfMovesCounter = 0;
 8   // Pointer to the current position in the PGN
 9   const char* currentPos = chessgame;
10
11   while (*currentPos != '\0' && halfMovesCounter < N_all) {
12     // Skip spaces and move to the next character
13     while (*currentPos == ' ') {
14       currentPos++;
15     }
16
17     // Check for the end of the PGN
18     if (*currentPos == '\0') {
```

GET_FIRST_MOVES FUNCTION

```
19        break;
20      }
21
22      // Check for the start of a new move
23      if (isalnum(*currentPos)) {
24        // Increment the halfMoves counter
25        halfMovesCounter++;
26
27        // Skip characters until the next space or the end of the PGN
28        while (isalnum(*currentPos) || *currentPos == '-') {
29          result = realloc(result, resultLength + 1);
30          if (result == NULL) {
31            fprintf(stderr, "Memory allocation error\n");
32            exit(EXIT_FAILURE);
```

GET_FIRST_MOVES FUNCTION

```
31            fprintf(stderr, "Memory allocation error\n");
32            exit(EXIT_FAILURE);
33          }
34          result[resultLength++] = *currentPos;
35          currentPos++;
36        }
37        if(halfMovesCounter%3 == 1){
38        // Add a dot after the move number
39        result = realloc(result, resultLength + 1);
40        if (result == NULL) {
41          fprintf(stderr, "Memory allocation error\n");
42          exit(EXIT_FAILURE);
43        }
44        result[resultLength++] = '.';
45        // result[resultLength++] = ' ';
```

GET_FIRST_MOVES FUNCTION

```
37        if(halfMovesCounter%3 == 1){
38        // Add a dot after the move number
39        result = realloc(result, resultLength + 1);
40        if (result == NULL) {
41          fprintf(stderr, "Memory allocation error\n");
42          exit(EXIT_FAILURE);
43        }
44        result[resultLength++] = '.';
45        // result[resultLength++] = ' ';
46        }
47
48        // Add a space between moves if not the last move
49        if (halfMovesCounter < N_all) {
50          result = realloc(result, resultLength + 1);
51          if (result == NULL) {
```

HASOPENING FUNCTION

1. Return false if record2 length is greater than record1.
2. Trim record1 to match record2 length, then compare.

HASOPENING FUNCTION

```
1 bool hasOpening_internal(text *san_text1, text *san_text2) {
2   // hasOpening_internal
3 }
4
5 PG_FUNCTION_INFO_V1(hasOpening);
6 Datum hasOpening(PG_FUNCTION_ARGS)  {
7   // hasOpening
8 }
```

HASOPENING FUNCTION

```
1  //Returns true if the first chess game starts with the exact same set of m
2  bool hasOpening_internal(text *san_text1, text *san_text2)
3
4  {
5
6    char *san_str1 = text_to_cstring(san_text1);
7
8    char *san_str2 = text_to_cstring(san_text2);
9    //Assume that the input is in the correct san format
10   if(strlen(san_str1)<strlen(san_str2)) //highlight
11   {
12     return false;
13   }
14   return strncmp(san_str2, san_str1, strlen(san_str2))==0; //highlight
15 }
```

HASOPENING FUNCTION

```
 5
 6   char *san_str1 = text_to_cstring(san_text1);
 7
 8   char *san_str2 = text_to_cstring(san_text2);
 9   //Assume that the input is in the correct san format
10   if(strlen(san_str1)<strlen(san_str2)) //highlight
11   {
12     return false;
13   }
14   return strncmp(san_str2, san_str1, strlen(san_str2))==0; //highlight
15 }
16 //Returns true if the first chess game starts with the exact same set of m
17 PG_FUNCTION_INFO_V1(hasOpening);
18 Datum
19 hasOpening(PG_FUNCTION_ARGS) {
```

HASOPENING FUNCTION

```
14   return strncmp(san_str2, san_str1, strlen(san_str2))==0; //highlight
15 }
16 //Returns true if the first chess game starts with the exact same set of m
17 PG_FUNCTION_INFO_V1(hasOpening);
18 Datum
19 hasOpening(PG_FUNCTION_ARGS) {
20
21   text *record1 = PG_GETARG_TEXT_PP(0);
22
23   text *record2 = PG_GETARG_TEXT_PP(1);
24
25   bool result = hasOpening_internal(record1,record2); //highlight
26
27   PG_RETURN_BOOL(result);
28 }
```

HASBOARD FUNCTION

1. Iterate through halfMoves with `getBoard_internal`.
2. Convert each board state to FEN and compare with targetBoard.
3. Return true if a matching state is found.

HASBOARD FUNCTION

```
bool hasBoard_internal(text *san_text, SCL_Board *targetBoard, int halfMove
  // hasBoard_internal
}

PG_FUNCTION_INFO_V1(hasBoard);
Datum
hasBoard(PG_FUNCTION_ARGS) {
  // hasBoard
}
```

HASBOARD FUNCTION

```
1  //Returns true if the chessgame contains the given board state in its firs
2  bool hasBoard_internal(text *san_text, SCL_Board *targetBoard, int halfMov
3    char *result = NULL;
4
5    char *fenstr = (char*)malloc(70 * sizeof(char));//allocation of size 70
6
7    strcpy(fenstr,"");
8
9    // Iterate through the first N half-moves
10   for (int i = 1; i <= halfMoves; ++i) {
11
12     result = getBoard_internal(san_text, i); //highlight
13
14     SCL_boardToFEN(*targetBoard, fenstr);   //highlight
15
```

HASBOARD FUNCTION

```
6
7   strcpy(fenstr,"");
8
9   // Iterate through the first N half-moves
10  for (int i = 1; i <= halfMoves; ++i) {
11
12    result = getBoard_internal(san_text, i); //highlight
13
14    SCL_boardToFEN(*targetBoard, fenstr);   //highlight
15
16    // Check if the current board matches the target board
17    if (strcmp(result, fenstr) == 0) {      //highlight
18      return true;  // Found a match
19    }
20  }
```

HASBOARD FUNCTION

```
10    for (int i = 1; i <= halfMoves; ++i) {
11
12      result = getBoard_internal(san_text, i); //highlight
13
14      SCL_boardToFEN(*targetBoard, fenstr);   //highlight
15
16      // Check if the current board matches the target board
17      if (strcmp(result, fenstr) == 0) {      //highlight
18        return true;  // Found a match
19      }
20    }
21
22    return false;  // Board not found in the first N half-moves
23  }
24
```

INDEX IMPLEMENTATION

B-TREE INDEX

1. The B-tree is a self-balancing tree data structure that maintains sorted data.
2. This B-tree index is created on the chessgame type, where each node of the tree stores a range of chessgame strings.

C IMPLEMENTATION FOR B-TREE INDEX

Purpose: Compare chess games for B-tree indexing.

- **Function:** `chessgame_cmp`
- **Input:** Text strings in SAN format.
- **Output:** Integer from string comparison.
- **Method:** Leverage `strcmp` for lexicographic order.

Companion comparison functions (like `chessgame_eq`, `chessgame_lt`, etc.) use this for various boolean checks.

SQL EXPLANATION FOR OPERATOR DEFINITIONS

Purpose: Define SQL operators for chessgame indexing.

- **Operators:** =, <>, <, <=, >, >=
- **Function:** Correspond to comparison functions.
- **Output:** Boolean indicating comparison result.

The `chessgame_btree` operator class utilizes `chessgame_cmp` for B-tree index ordering, enhancing search and query capabilities.

COMPARISON FUNCTIONS

```
PG_FUNCTION_INFO_V1(chessgame_cmp);
Datum chessgame_cmp(PG_FUNCTION_ARGS) {
  // chessgame_cmp
}

PG_FUNCTION_INFO_V1(chessgame_eq);
Datum chessgame_eq(PG_FUNCTION_ARGS) {
  // chessgame_eq
}

PG_FUNCTION_INFO_V1(chessgame_ne);
Datum chessgame_ne(PG_FUNCTION_ARGS) {
  // chessgame_ne
}
```

COMPARISON FUNCTIONS

```
1  PG_FUNCTION_INFO_V1(chessgame_cmp);
2  Datum chessgame_cmp(PG_FUNCTION_ARGS) {
3    // Extract the two input SAN strings
4    text *san_text_1 = PG_GETARG_TEXT_PP(0);
5    text *san_text_2 = PG_GETARG_TEXT_PP(1);
6    // Convert the text to C strings
7    char *san_str_1 = text_to_cstring(san_text_1);
8    char *san_str_2 = text_to_cstring(san_text_2);
9    // Compare the two strings using strcmp
10   int result = strcmp(san_str_1, san_str_2);
11   // Free the allocated memory
12   pfree(san_str_1);
13   pfree(san_str_2);
14   // Return the comparison result
15   PG_RETURN_INT32(result);
```

COMPARISON FUNCTIONS

```
3   // Extract the two input SAN strings
4   text *san_text_1 = PG_GETARG_TEXT_PP(0);
5   text *san_text_2 = PG_GETARG_TEXT_PP(1);
6   // Convert the text to C strings
7   char *san_str_1 = text_to_cstring(san_text_1);
8   char *san_str_2 = text_to_cstring(san_text_2);
9   // Compare the two strings using strcmp
10  int result = strcmp(san_str_1, san_str_2);
11  // Free the allocated memory
12  pfree(san_str_1);
13  pfree(san_str_2);
14  // Return the comparison result
15  PG_RETURN_INT32(result);
16 }
17
```

COMPARISON FUNCTIONS

```
15    PG_RETURN_INT32(result);
16  }
17
18  PG_FUNCTION_INFO_V1(chessgame_eq);
19  Datum chessgame_eq(PG_FUNCTION_ARGS) {
20    // Call chessgame_cmp and check for equality
21    int32 result = DatumGetInt32(DirectFunctionCall2(chessgame_cmp, PG_GETAR
22
23    PG_RETURN_BOOL(result == 0);
24  }
25
26  PG_FUNCTION_INFO_V1(chessgame_ne);
27  Datum chessgame_ne(PG_FUNCTION_ARGS) {
28    // Call chessgame_cmp and check for inequality
29    int32 result = DatumGetInt32(DirectFunctionCall2(chessgame_cmp, PG_GETAR
```

COMPARISON FUNCTIONS

```
23   PG_RETURN_BOOL(result == 0);
24 }
25
26 PG_FUNCTION_INFO_V1(chessgame_ne);
27 Datum chessgame_ne(PG_FUNCTION_ARGS) {
28   // Call chessgame_cmp and check for inequality
29   int32 result = DatumGetInt32(DirectFunctionCall2(chessgame_cmp, PG_GETARG
30
31   PG_RETURN_BOOL(result != 0);
32 }
33
34 PG_FUNCTION_INFO_V1(chessgame_lt);    // Call chessgame_cmp and check for
35 PG_FUNCTION_INFO_V1(chessgame_le);    // Call chessgame_cmp and check for
36 PG_FUNCTION_INFO_V1(chessgame_gt);    // Call chessgame_cmp and check for
37 PG_FUNCTION_INFO_V1(chessgame_ge);    // Call chessgame_cmp and check for
```

FUNCTION DEFINITION IN POSTGRESQL

```
1   CREATE FUNCTION chessgame_cmp(chessgame, chessgame)
2   RETURNS integer
3   AS 'MODULE_PATHNAME', 'chessgame_cmp'
4   LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
5
6 CREATE FUNCTION chessgame_eq(chessgame, chessgame)
7   RETURNS boolean
8   AS 'MODULE_PATHNAME', 'chessgame_eq'
9   LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
10
11 CREATE FUNCTION chessgame_ne(chessgame, chessgame)
12  RETURNS boolean
13  AS 'MODULE_PATHNAME', 'chessgame_ne'
14  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

OPERATOR DEFINITION

```
CREATE OPERATOR = (
LEFTARG = chessgame, RIGHTARG = chessgame,
PROCEDURE = chessgame_eq,
COMMUTATOR = =,
NEGATOR = <>
);
-- Not equal
CREATE OPERATOR <> (
LEFTARG = chessgame, RIGHTARG = chessgame,
PROCEDURE = chessgame_ne,
COMMUTATOR = <>,
NEGATOR = =
);
```

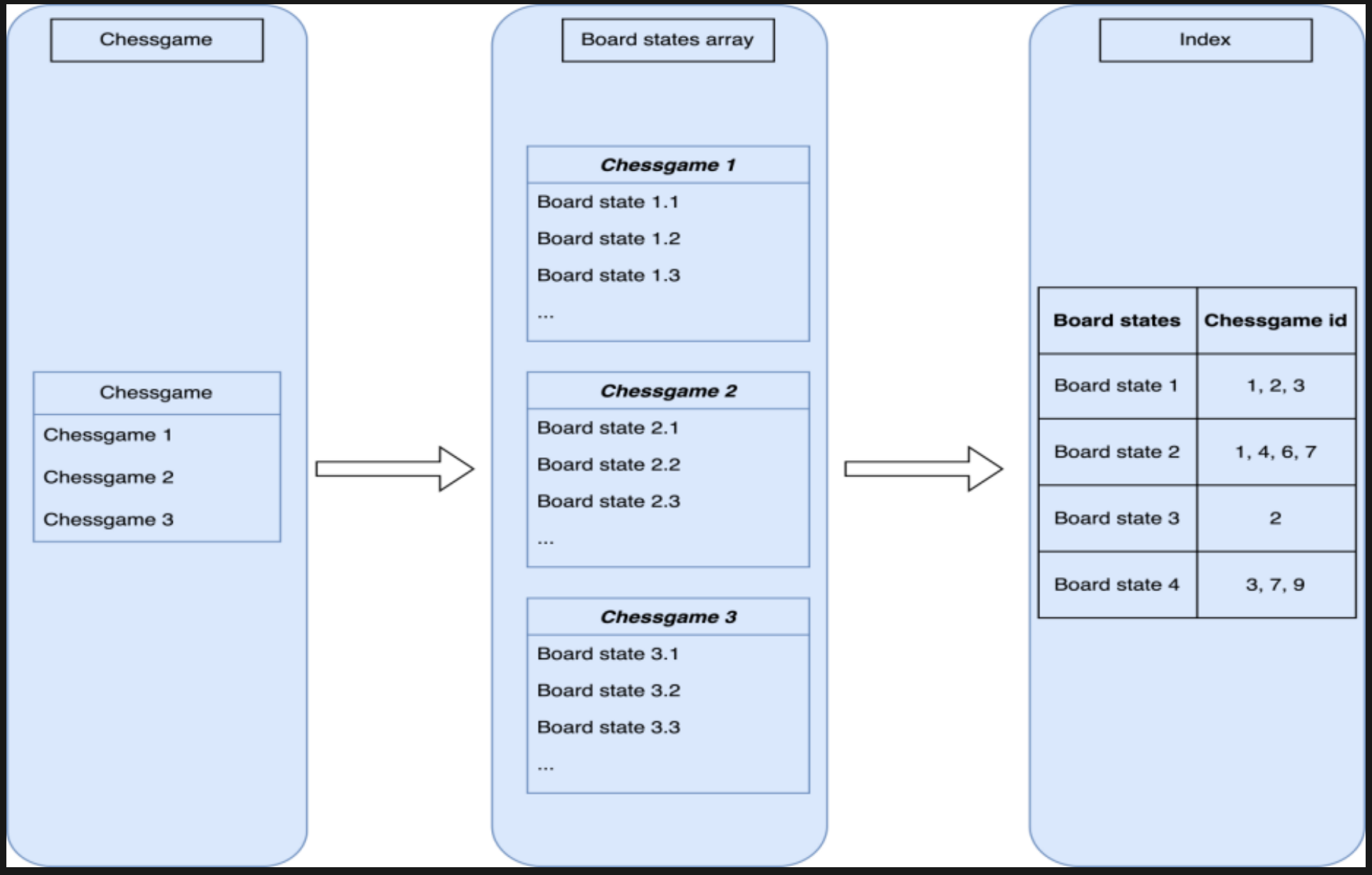
OPERATOR DEFINITION

```
CREATE OPERATOR = (
LEFTARG = chessgame, RIGHTARG = chessgame,
PROCEDURE = chessgame_eq,
COMMUTATOR = =,
NEGATOR = <>
);
-- Not equal
CREATE OPERATOR <> (
LEFTARG = chessgame, RIGHTARG = chessgame,
PROCEDURE = chessgame_ne,
COMMUTATOR = <>,
NEGATOR = =
);
```

B TREE OPERATOR CLASS

```
1 CREATE OPERATOR CLASS chessgame_btree
2 DEFAULT FOR TYPE chessgame USING btree AS
3 OPERATOR 1 < (chessgame, chessgame),
4 OPERATOR 2 <= (chessgame, chessgame),
5 OPERATOR 3 = (chessgame, chessgame),
6 OPERATOR 4 >= (chessgame, chessgame),
7 OPERATOR 5 > (chessgame, chessgame),
8 FUNCTION 1 chessgame_cmp(chessgame, chessgame);
```

GIN INDEX


Chessgame
Chessgame
Chessgame 1
Chessgame 2
Chessgame 3
Board states array
Chessgame 1
Board state 1.1
Board state 1.2
Board state 1.3
...
Chessgame 2
Board state 2.1
Board state 2.2
Board state 2.3
...
Chessgame 3
Board state 3.1
Board state 3.2
Board state 3.3
...
Index
Board states
Chessgame id
Board state 1
1, 2, 3
Board state 2
1, 4, 6, 7
Board state 3
2
Board state 4
3, 7, 9

GIN OPERATOR CLASS

```
1 CREATE OPERATOR CLASS chessgame_gin
2 DEFAULT FOR TYPE chessgame USING gin AS
3 OPERATOR 5 @>,
4 FUNCTION 1 compare(internal, internal),
5 FUNCTION 2 extractValue(internal, internal, internal),
6 FUNCTION 3 extractQuery(internal, internal, internal, internal, internal, i
7 FUNCTION 4 consistent(internal, internal, internal, internal, internal, int
8 STORAGE text;
```

CREATE INDEX

EXTRACTVALUE FUNCTION

```
1 Datum *extractValue(Datum itemValue,
2 int32 *nkeys, bool **nullFlags)
```

1. Purpose: the goal of the extractValue function is to take a chessgame as input (itemValue), extract board states array from it, and then compute a list of unique boardstate values along with their corresponding chess game IDs. The result will be used to populate the nkeys and nullFlags parameters.

EXTRACTVALUE PSEUDOCODE

```
Datum *extractValue(Datum itemValue,
int32 *nkeys, bool **nullFlags) {
  chessgame *chessGame = (chessgame *) DatumGetPointer(itemValue);
   for each game in chessGameList {
     chessboard currentBoardState = extractBoardState(game.boardgame);
     if (contains(boardStateMap, currentBoardState)) {
        int index = getIndex(boardStateMap, currentBoardState);
       addGameId(gameIdList[index], game.id);
     } else {
       addEntry(boardStateMap, currentBoardState);
       addGameId(gameIdList, [game.id]);  // Create a new list with the cu
     }
   }
   *nkeys = size(boardStateMap);
   Datum *resultArray = createResultArray(boardStateMap, gameIdList);
```

EXTRACTVALUE PSEUDOCODE

```
Datum *extractValue(Datum itemValue,
int32 *nkeys, bool **nullFlags) {
  chessgame *chessGame = (chessgame *) DatumGetPointer(itemValue);
   for each game in chessGameList {
     chessboard currentBoardState = extractBoardState(game.boardgame);
     if (contains(boardStateMap, currentBoardState)) {
        int index = getIndex(boardStateMap, currentBoardState);
       addGameId(gameIdList[index], game.id);
     } else {
       addEntry(boardStateMap, currentBoardState);
       addGameId(gameIdList, [game.id]);  // Create a new list with the cu
     }
   }
   *nkeys = size(boardStateMap);
   Datum *resultArray = createResultArray(boardStateMap, gameIdList);
```

QUERY USING INDEX

EXTRACTQUERY FUNCTION

```
1 Datum *extractQuery(Datum query, int32 *nkeys, StrategyNumber n,
2 bool **pmatch, Pointer **extra_data, bool **nullFlags, int32 *searchMode)
```

1. Purpose: the extractQuery function takes a query as input, which represents the target board state. The goal is to search for this target board state in the map obtained from the extractValue function and retrieve the corresponding chess game IDs.

EXTRACTQUERY PSEUDOCODE

```
Datum *extractQuery(Datum query, int32 *nkeys, StrategyNumber n,
bool **pmatch, Pointer **extra_data, bool **nullFlags, int32 *searchMode) {
  boardstate targetBoardState = (chessboard) DatumGetPointer(query);
  // Initialize data structures to store matching results
  List matchingGameIds;  // List to store chess game IDs that match the ta
  Map boardStateMap = getBoardStateMapFromResult(valueResult);
  List gameIdList = getGameIdListFromResult(valueResult);
  // Check if the target board state is in the map
  if (contains(boardStateMap, targetBoardState)) {  // Retrieve the index
     int index = getIndex(boardStateMap, targetBoardState);
    add index into matchingGameIds ;  // Add the chess game ID to the matc
  }     // Convert the matchingGameIds list to an array for the output
  return matchingGameIds ;
}
```

EXTRACTQUERY PSEUDOCODE

```
Datum *extractQuery(Datum query, int32 *nkeys, StrategyNumber n,
bool **pmatch, Pointer **extra_data, bool **nullFlags, int32 *searchMode)
  boardstate targetBoardState = (chessboard) DatumGetPointer(query);
  // Initialize data structures to store matching results
  List matchingGameIds;  // List to store chess game IDs that match the ta
  Map boardStateMap = getBoardStateMapFromResult(valueResult);
  List gameIdList = getGameIdListFromResult(valueResult);
  // Check if the target board state is in the map
  if (contains(boardStateMap, targetBoardState)) {  // Retrieve the index
     int index = getIndex(boardStateMap, targetBoardState);
    add index into matchingGameIds ;  // Add the chess game ID to the matc
  }     // Convert the matchingGameIds list to an array for the output
  return matchingGameIds ;
}
```

QUERY USING INDEX

CONSISTENT FUNCTION

`bool consistent` (bool check[], int32 nkeys, List matchingGameIds, Datum queryKeys[])

1. Purpose: the consistent function initializes the check array with false values. The element of the check array is updated as true if the indexed item contains the corresponding query key, i.e., if (check[i] == true) the i-th key is present in matchingGameIds, which is obtained from the extractQuery function.

EVALUATION

TESTING METHODOLOGY OVERVIEW

- Three datasets:
  - `t(Adams)`
  - `t2(TrompowskyOther)`
  - `fen(practice data)`
- Test cases design:
  - `getboard` - Retrieve and compare game state with practice data at a specific half-move.
  - `getfirstmoves` - Show the first N half-moves.
  - `hasopening` - Verify if two chess games share the same opening moves.
  - `hasboard` - Check for a given board state within the first N half-moves.
  - `B-tree` - Implement with and without B-tree indexing on different dataset scales.

TESTING METHODOLOGY OVERVIEW

- Resources:
  - Use Apronus PGN Viewer to record and verify SAN and FEN from practiced games.

EVALUATION OF GETBOARD

```
postgres=# SELECT
    getBoard(t.Moves, 6)::text AS generated_fen,
    f.fen AS fen_data,
    (CASE
        WHEN getBoard(t.Moves, 6)::text = f.fen THEN true
        ELSE false
    END) AS match
FROM
    t,
    fen_data f
WHERE
    t.Moves = '1. e3 d5 2. g4 Bxg4 3. e4 Bxd1 *' AND
    f.id = 6;
                      generated_fen                       |                          fen_data                          | match
----------------------------------------------------------+------------------------------------------------------------+------
-
 rn1qkbnr/ppp1pppp/8/3p4/4P3/8/PPPP1P1P/RNBbKBNR w KQkq - 0 4 | rn1qkbnr/ppp1pppp/8/3p4/4P3/8/PPPP1P1P/RNBbKBNR w KQkq - 0 4 | t
(1 row)
```

1. Create fen table
2. Import fen data
3. Query if the generated_fen euqal the true fen in our chess website and return two string and T/F

EVALUATION OF GETFIRSTMOVES

```
postgres=# SELECT CAST(getFirstMoves(t.Moves, 3) AS TEXT) AS first_moves, COUNT(*) AS move_count
FROM t
WHERE Black = 'Adams, Michael' AND Result='0-1'
GROUP BY first_moves
ORDER BY move_count DESC
LIMIT 1;
   first_moves    | move_count
------------------+------------
 1. e4 e5 2. Nf3 |         46
(1 row)
```

1. Query the most frequent first three half-moves in games when Adams was playing as Black and won the game

EVALUATION OF HASOPENING

```
postgres=# SELECT count(*)
FROM t
WHERE hasopening(Moves, '1. e4 e6');
 count
-------
   209
(1 row)
```

1. Query the number of games starting with '1. e4 e6'

EVALUATION OF HASOPENING

```
postgres=# SELECT id
FROM t
WHERE hasopening(Moves,'1. e4 c6 2. d4 d5') AND Result='0-1';
  id
------
   43
   49
   61
   65
   86
   98
   99
  107
  124
  150
  175
  479
  586
  613
  685
  746
  760
  831
  949
  990
 1046
 1155
 1317
 1319
 2969
 3191
 3300
 3341
 3435
(29 rows)
```

1. Query the id of games starting with '1. e4 e6' and Black won the game

EVALUATION OF HASBOARD

```
postgres=# SELECT count(*)
FROM t
WHERE hasboard(Moves,
'rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1', 10);
 count
-------
  2298
(1 row)

postgres=# SELECT count(*)
FROM t
WHERE hasboard(Moves,
'rnbqkbnr/pppppppp/8/8/8/5N2/PPPPPPPP/RNBQKB1R b KQkq - 1 1', 10);
 count
-------
   200
(1 row)

postgres=# SELECT count(*)
FROM t
WHERE hasboard(Moves,
'rnbqkb1r/pppppppp/5n2/8/8/5N2/PPPPPPPP/RNBQKB1R w KQkq - 2 2', 10);
 count
-------
   149
(1 row)
```

1. The number of chessgames containing board state 'rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1' in the first ten moves.
2. The number of chessgames containing board state 'rnbqkbnr/pppppppp/8/8/8/5N2/PPPPPPPP/RNBQKB1R b KQkq - 1 1' in the first ten moves.
3. The number of chessgames containing board state'rnbqkb1r/pppppppp/5n2/8/8/5N2/PPPPPPPP/RNBQKB1R w KQkq - 2 2' in the first ten moves.

BTREE FOR ADAMS (3K+)

```
postgres=# DROP INDEX my_chessgame_index;
ERROR:  index "my_chessgame_index" does not exist
postgres=# CREATE INDEX my_chessgame_index ON t USING btree (Moves chessgame_btree);
CREATE INDEX
postgres=# SET enable_seqscan = OFF;
SET
postgres=# EXPLAIN (ANALYZE, BUFFERS, TIMING ON) SELECT hasOpening('1. e3 Nh6', Moves) FROM t;
                                                              QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------
----
 Index Only Scan using my_chessgame_index on t  (cost=0.79..16894.15 rows=20664 width=1) (actual time=0.064..36.707 rows=20664 loops
=1)
   Heap Fetches: 5
   Buffers: shared hit=2 read=3445
 Planning:
   Buffers: shared hit=43 read=1 dirtied=1
 Planning Time: 0.235 ms
 Execution Time: 39.102 ms
(7 rows)

postgres=# SET enable_seqscan = ON;
SET
postgres=# EXPLAIN (ANALYZE, BUFFERS, TIMING ON) SELECT hasOpening('1. e3 Nh6', Moves) FROM t;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on t  (cost=0.00..3207.30 rows=20664 width=1) (actual time=0.008..8.455 rows=20664 loops=1)
   Buffers: shared hit=2949
 Planning Time: 0.044 ms
 Execution Time: 9.073 ms
(4 rows)
```

1. Create a table with more than 3000 rows
2. Test with index
3. Test without index

BTREE FOR TROMPOWSKYOTHER (17K+)

```
postgres=# CREATE INDEX my_chessgame_index ON t2 USING btree (Moves chessgame_btree);
CREATE INDEX
postgres=# SET enable_seqscan = OFF;
SET
postgres=# EXPLAIN (ANALYZE, BUFFERS, TIMING ON) SELECT hasOpening('1. e3 Nh6', Moves) FROM t2;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on t2  (cost=4.45..15.21 rows=60 width=1) (actual time=0.002..0.003 rows=0 loops=1)
   Buffers: shared hit=1
   ->  Bitmap Index Scan on my_chessgame_index  (cost=0.00..4.44 rows=60 width=0) (actual time=0.001..0.001 rows=0 loops=1)
         Buffers: shared hit=1
 Planning:
   Buffers: shared hit=17 read=1
 Planning Time: 0.151 ms
 Execution Time: 0.016 ms
(8 rows)

postgres=# SET enable_seqscan = ON;
SET
postgres=# EXPLAIN (ANALYZE, BUFFERS, TIMING ON) SELECT hasOpening('1. e3 Nh6', Moves) FROM t2;
                                            QUERY PLAN
---------------------------------------------------------------------------------------------------
 Seq Scan on t2  (cost=0.00..10.75 rows=60 width=1) (actual time=0.003..0.003 rows=0 loops=1)
 Planning:
   Buffers: shared hit=3
 Planning Time: 0.173 ms
 Execution Time: 0.072 ms
(5 rows)
```

1. Create table TROMPOWSKYOTHER
2. Test with index
3. Test without index

APPENDIX

EVALUATION OF GETBOARD

```
postgres=# DELETE FROM t WHERE Moves = '1. e3 d5 2. g4 Bxg4 3. e4 Bxd1 *';
DELETE 2
postgres=# DROP TABLE IF EXISTS fen_data;
DROP TABLE
postgres=# INSERT INTO t(Moves) VALUES('1. e3 d5 2. g4 Bxg4 3. e4 Bxd1 *');
INSERT 0 1
postgres=# CREATE TABLE fen_data (
    id SERIAL PRIMARY KEY,
    fen TEXT
);
CREATE TABLE
postgres=# \COPY fen_data(fen) FROM '/home/jintao/Pictures/Screenshots/Test_Case/getBoard/verify/total.fen' WITH (FORMAT text);
COPY 6
postgres=# \pset pager off
Pager usage is off.
```

1. Create fen table
2. Import fen data

EVALUATION OF GETBOARD

```
postgres=# SELECT getBoard(t.Moves,3) AS chess_board FROM t WHERE id=1;
                        chess_board
------------------------------------------------------------
 rnbqkbnr/pppp1ppp/4p3/8/3PP3/8/PPP2PPP/RNBQKBNR b KQkq d3 0 2
(1 row)

postgres=# SELECT CAST(getBoard(t.Moves, 3) AS text) AS board, COUNT(*) AS count
FROM t
WHERE id IN (1, 2)
GROUP BY CAST(getBoard(t.Moves, 3) AS text);
                            board                             | count
--------------------------------------------------------------+-------
 rnbqkbnr/pppp1ppp/4p3/8/3PP3/8/PPP2PPP/RNBQKBNR b KQkq d3 0 2 |     2
(1 row)

postgres=# SELECT CAST(getBoard(t.Moves, 3) AS TEXT) AS chess_board, COUNT(*) AS count
FROM t
WHERE White = 'Adams, Michael'
GROUP BY CAST(getBoard(t.Moves, 3) AS TEXT)
ORDER BY count DESC
LIMIT 1;
                        chess_board                          | count
-------------------------------------------------------------+-------
 rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2 |   218
(1 row)

postgres=# SELECT CAST(getBoard(t.Moves, 3) AS TEXT) AS chess_board, COUNT(*) AS count
FROM t
WHERE Black = 'Adams, Michael' AND Result='0-1'
GROUP BY CAST(getBoard(t.Moves, 3) AS TEXT)
ORDER BY count DESC
LIMIT 1;
                        chess_board                          | count
-------------------------------------------------------------+-------
 rnbqkbnr/pppp1ppp/8/4p3/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2 |    46
(1 row)
```

EVALUATION OF GETFIRSTMOVES

```
postgres=# SELECT CAST(getFirstMoves(t.Moves, 7) AS TEXT) AS first_moves, COUNT(*) AS move_count
FROM t
WHERE White = 'Adams, Michael'
GROUP BY first_moves
ORDER BY move_count DESC
LIMIT 1;
            first_moves              | move_count
-------------------------------------+------------
 1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 |        104
(1 row)

postgres=# SELECT CAST(getFirstMoves(t.Moves, 5) AS TEXT) AS first_moves, COUNT(*) AS move_count
FROM t
WHERE Black = 'Adams, Michael' AND Result='0-1'
GROUP BY first_moves
ORDER BY move_count DESC
LIMIT 1;
        first_moves         | move_count
----------------------------+------------
 1. e4 e5 2. Nf3 Nc6 3. Bb5 |         35
(1 row)
```

BTREE FOR ADAMS (3K+)

```
postgres=# DROP TABLE IF EXISTS t;
DROP TABLE
postgres=# CREATE TABLE t (
    id SERIAL PRIMARY KEY,
    Event TEXT,
    Site TEXT,
    Date TEXT,
    Round INT,
    White TEXT,
    Black TEXT,
    Result TEXT,
    Moves chessgame
);
CREATE TABLE
postgres=# \COPY t(Event, Site, Date, Round, White, Black, Result, Moves) FROM '/home/jintao/Downloads/test/Adams4.csv' DELIMITER ',
' CSV HEADER;
COPY 3485
```

1. create a bigger table named t2 just for B tree test

BTREE FOR TROMPOWSKYOTHER (17K+)

```
postgres=# DROP TABLE IF EXISTS t2;
DROP TABLE
postgres=# CREATE TABLE t2 (
    id SERIAL PRIMARY KEY,
    Event TEXT,
    Site TEXT,
    Date TEXT,
    Round INT,
    White TEXT,
    Black TEXT,
    Result TEXT,
    Moves chessgame
);
CREATE TABLE
postgres=# \COPY t(Event, Site, Date, Round, White, Black, Result, Moves) FROM '/home/jintao/Downloads/test/TrompowskyOther4.csv' DE
LIMITER ',' CSV HEADER;
COPY 17178
```

1. Create table TROMPOWSKYOTHER

REFERENCES

Documentation and tools used in the project:

- PostgreSQL Indexes Documentation: www.postgresql.org/docs/current/xindex.html
- Data Sources:
  - `t(Adams)`
  - `t2(TrompowskyOther)`
  - `fen(practice data)`
- Apronus PGN Viewer for SAN and FEN Records:
  - www.apronus.com/chess/pgnviewer/