Chess extension PostgreSQL

Younes El Mokhtari

Alexandre Bienfait

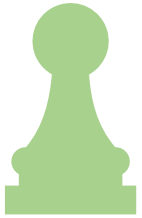
Brenno Ferreira

Philippe Mutkowski

Structure

ChessGame Type

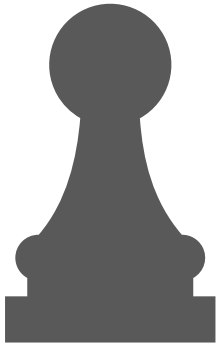
Chessboard type



Functions and predicates

Indexes: B-tree and GIN

ChessGame Type



chess.c

```c
typedef struct {
  int32 length;
  char pgn[FLEXIBLE_ARRAY_MEMBER];
} chessgame_t;
```

```c
static chessgame_t *
chessgame_make(const char *pgn)
{
chessgame_t *chessgame = (chessgame_t *)
palloc(VARHDRSZ + strlen(pgn) + 1);
SET_VARSIZE(chessgame, VARHDRSZ + strlen(pgn) + 1);
memcpy(chessgame->pgn, pgn, strlen(pgn) + 1);
return chessgame;
}
```

chess--1.0.sql

```
CREATE TYPE chessgame (
internallength = variable,
input = chessgame_in,
output = chessgame_out,
receive = chessgame_recv,
send = chessgame_send
);
```

```
Test 1: Size of data types
Test 1.1: Size of each game - SELECT game, pg_size_pretty(pg_column_size(game)::bigint) AS game_size FROM games;
                                 game                                  | game_size
-----------------------------------------------------------------------+-----------
 1.e4 e6 2.d4 d5 3.Nd2 Nf6 4.e5 Nfd7 5.f4 c5 6.c3 Nc6 7.Ndf3 cxd4      | 70 bytes
 1.e4 e6 2.d4 d5 3.Nd2 c5 4.exd5 Qxd5 5.Ngf3 cxd4 6.Bc4 Qd6 7.O-O Nf6  | 74 bytes
 1.e4 c6 2.c4 d5 3.exd5 cxd5 4.cxd5 Nf6 5.Nc3 g6 6.Bc4 Bg7 7.Nf3 O-O   | 73 bytes
 1.e4 c6 2.c4 d5 3.exd5 cxd5 4.cxd5 Nf6 5.Nc3 Nxd5 6.d4 Nc6 7.Nf3 e6   | 73 bytes
 1.e4 c5 2.Nf3 d6 3.d4 cxd4 4.Nxd4 Nf6 5.Nc3 a6 6.Bg5 e6 7.f4 Be7      | 70 bytes
(5 rows)

Time: 0.184 ms
```

ChessBoard Type

chess.c

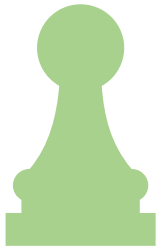
```c
typedef struct {
  int32 length;
  char fen[FLEXIBLE_ARRAY_MEMBER];
} chessboard_t;
```

```c
static chessboard_t *
chessboard_make(const char *fen)
{
chessboard_t *chessboard = (chessboard_t *)
palloc(VARHDRSZ + strlen(fen) + 1);
SET_VARSIZE(chessboard, VARHDRSZ + strlen(fen) + 1);
memcpy(chessboard->fen, fen, strlen(fen) + 1);
return chessboard;
}
```

chess--1.0.sql

```
CREATE TYPE chessboard (
internallength = variable,
input = chessboard_in,
output = chessboard_out,
receive = chessboard_recv,
send = chessboard_send
);
```

Functions and predicates



Definition of the two sql functions

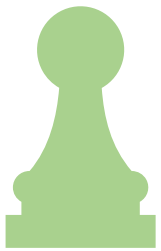
chess--1.0.sql

```
CREATE FUNCTION getFirstMoves(chessgame, integer)
  RETURNS chessgame
  AS 'MODULE_PATHNAME', 'getFirstMoves'
  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

```
CREATE FUNCTION getBoard(chessgame, integer)
  RETURNS chessboard
  AS 'MODULE_PATHNAME', 'getBoard'
  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

Functions and predicates

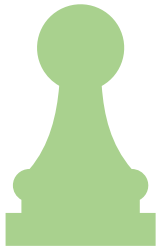
chess.c




```c
PG_FUNCTION_INFO_V1(getFirstMoves);
Datum
getFirstMoves(PG_FUNCTION_ARGS)
{
  chessgame_t *chessgame = PG_GETARG_CHESSGAME_P(0);
  uint16_t number_half_moves = PG_GETARG_INT16(1);
  chessgame_t *truncated_chessgame = truncate_chessgame(chessgame, number_half_moves);
  PG_FREE_IF_COPY(chessgame, 0);
  PG_RETURN_CHESSGAME_P(truncated_chessgame);
}
```

```c
chessgame_t *
truncate_chessgame(chessgame_t *chessgame, uint16_t number_half_moves)
{
  char delimeter = ' ';
  uint16_t i = 0;
  uint16_t counter = 0;
  char *truncated_pgn = (char *) malloc(sizeof(char) * MAX_PGN_LENGTH);
  while (chessgame->pgn[i] != '\0' && counter < number_half_moves) {
    if(chessgame->pgn[i] == delimeter) {
      counter += 1;
    }
    truncated_pgn[i] = chessgame->pgn[i];
    i += 1;
  }
  truncated_pgn[i] = '\0';
  chessgame_t *truncated_chessgame = PGN_to_chessgame(truncated_pgn);
  free(truncated_pgn);
  return truncated_chessgame;
}
```

Functions and predicates



chess.c

```c
PG_FUNCTION_INFO_V1(getBoard);
Datum
getBoard(PG_FUNCTION_ARGS)
{
  chessgame_t *chessgame = PG_GETARG_CHESSGAME_P(0);
  uint16_t number_half_moves = PG_GETARG_INT16(1);
  chessboard_t *chessboard = chessgame_to_chessboard(chessgame, number_half_moves);
  PG_FREE_IF_COPY(chessgame, 0);
  PG_RETURN_CHESSBOARD_P(chessboard);
}
```

```c
static chessboard_t *
chessgame_to_chessboard(chessgame_t *chessgame, uint16_t number_half_moves)
{
  SCL_Record record;
  SCL_Board board;
  char *fen = (char *) malloc(sizeof(char) * SCL_FEN_MAX_LENGTH);
  SCL_recordFromPGN(record, chessgame->pgn);
  SCL_recordApply(record, board, number_half_moves);
  SCL_boardToFEN(board, fen);
  chessboard_t *chessboard = FEN_to_chessboard(fen);
  free(fen);
  return chessboard;
}
```

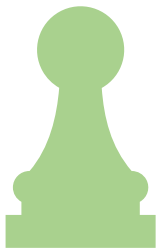
Functions and predicates

Two versions of the hasOpening method.

- Both use the Index Only Scan
- Both work the same way but use Filter instead of Index Condition

hasOpening

hasOpening



chess--1.0.sql

```
CREATE FUNCTION hasOpening(chessgame, chessgame)
  RETURNS boolean
  AS 'MODULE_PATHNAME', 'hasOpening'
  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

chess.c

```
PG_FUNCTION_INFO_V1(hasOpening);
Datum
hasOpening(PG_FUNCTION_ARGS)
{
  chessgame_t *chessgame_1 = PG_GETARG_CHESSGAME_P(0);
  chessgame_t *chessgame_2 = PG_GETARG_CHESSGAME_P(1);
  bool hasOpening = compare_moves(chessgame_1, chessgame_2);
  PG_FREE_IF_COPY(chessgame_1, 0);
  PG_FREE_IF_COPY(chessgame_2, 1);
  PG_RETURN_BOOL(hasOpening);
}
```

Use strstr()

hasOpening2

chess--1.0.sql

```
CREATE FUNCTION hasOpening2(a chessgame, b chessgame)
  RETURNS boolean
  AS $$
    SELECT a LIKE b;
  $$ IMMUTABLE LANGUAGE sql;
```

Functions and predicates

Two versions of the hasBoard method

- The C one is complete, but doesn't use the index
- The SQL one uses the index but falls short of the requirements

hasBoard

hasBoard

chess--1.0.sql

```
CREATE FUNCTION hasBoard(chessgame, chessboard, integer)
  RETURNS boolean
  AS 'MODULE_PATHNAME', 'hasBoard'
  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

chess.c

```
PG_FUNCTION_INFO_V1(hasBoard);
Datum
hasBoard(PG_FUNCTION_ARGS)
{
  chessgame_t *chessgame = PG_GETARG_CHESSGAME_P(0);
  chessboard_t *chessboard = PG_GETARG_CHESSBoard_P(1);
  Uint16_t number_half_moves = PG_GETARG_INT16(2);
  bool hasBoard = chessgame_contains_chessboard(chessgame,
chessboard, number_half_moves);
  PG_FREE_IF_COPY(chessgame, 0);
  PG_FREE_IF_COPY(chessboard, 1);
  PG_RETURN_BOOL(hasBoard);
}
```


hasBoard2

chess--1.0.sql

```
CREATE FUNCTION hasBoard2(a chessgame, b chessboard,
            c integer)
  RETURNS boolean
  AS $$
    SELECT chessgame_to_chessboards(a) @> ARRAY[b];
$$ IMMUTABLE LANGUAGE sql;
```

Indexes B-tree

Operators

chess--1.0.sql

```
CREATE OPERATOR = (
  LEFTARG = chessgame, RIGHTARG = chessgame,
  PROCEDURE = chess_opening_eq,
  COMMUTATOR = =, NEGATOR = <>
);

CREATE OPERATOR < (
  LEFTARG = chessgame, RIGHTARG = chessgame,
  PROCEDURE = chess_opening_lt,
  COMMUTATOR = >, NEGATOR = >=
);

CREATE OPERATOR <= (
  LEFTARG = chessgame, RIGHTARG = chessgame,
  PROCEDURE = chess_opening_le,
  COMMUTATOR = >=, NEGATOR = <
);
```

```
CREATE OPERATOR >= (
  LEFTARG = chessgame, RIGHTARG = chessgame,
  PROCEDURE = chess_opening_ge,
  COMMUTATOR = <=, NEGATOR = <
);

CREATE OPERATOR > (
  LEFTARG = chessgame, RIGHTARG = chessgame,
  PROCEDURE = chess_opening_gt,
  COMMUTATOR = <, NEGATOR = <=
);
```

Additional custom operator for hasOpening that is not in the B-tree operators:

```
CREATE OPERATOR ~~ (
  LEFTARG = chessgame, RIGHTARG = chessgame,
  PROCEDURE = chess_opening_like,
  COMMUTATOR = ~~, NEGATOR = !~
);
```

Indexes B-tree

Operators : Implementation

Each operator has been implemented the same way. Here is the one we add for the predicate function *hasOppening().*


```
CREATE OR REPLACE FUNCTION
chess_opening_like(chessgame, chessgame)
  RETURNS boolean
  AS 'MODULE_PATHNAME'
  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

```
PG_FUNCTION_INFO_V1(chess_opening_like);
Datum
chess_opening_like(PG_FUNCTION_ARGS)
{
        chessgame_t *c = PG_GETARG_CHESSGAME_P(0);
        chessgame_t *d = PG_GETARG_CHESSGAME_P(1);

        bool result = chess_opening_cmp_internal(chessgame_truncated_internal(c, chessgame_to_number_internal(d)), d) == 0;
        PG_FREE_IF_COPY(c, 0);
        PG_FREE_IF_COPY(d, 1);
        PG_RETURN_BOOL(result);
}
```

Indexes B-tree

Operators : Implementation

Each operator has been implemented the same way. Here is the one we add for the predicate function *hasOppening().*


```
PG_FUNCTION_INFO_V1(chess_opening_like);
Datum
chess_opening_like(PG_FUNCTION_ARGS)
{
        chessgame_t *c = PG_GETARG_CHESSGAME_P(0);
        chessgame_t *d = PG_GETARG_CHESSGAME_P(1);

        bool result = chess_opening_cmp_internal(chessgame_truncated_internal(c, chessgame_to_number_internal(d)), d) == 0;
        PG_FREE_IF_COPY(c, 0);
        PG_FREE_IF_COPY(d, 1);
        PG_RETURN_BOOL(result);
}
```

```
static int
chess_opening_cmp_internal(chessgame_t *a, chessgame_t *b)
{
int cmp_result = strcmp(opening(a), opening(b));
if (cmp_result < 0)
{
return -1;
}
if (cmp_result > 0)
{
return 1;
}
return 0;
}
```

Indexes B-tree

Query plan with hasOpening predicate without index

```
Test 6.1: without btree index
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=257.59..257.60 rows=1 width=8) (actual time=0.643..0.644 rows=1 loops=1)
   ->  Seq Scan on games  (cost=0.00..254.74 rows=1140 width=0) (actual time=0.005..0.639 rows=8 loops=1)
         Filter: hasopening(game, '1.e4 c6 2.c4 d5 3.exd5 cxd5 '::chessgame)
         Rows Removed by Filter: 3411
 Planning Time: 0.078 ms
 Execution Time: 0.656 ms
(6 rows)

Time: 1.103 ms
```

Indexes B-tree

Query plan with hasOpening and hasOpening2 predicate with B-Tree index

```
                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=1099.21..1099.22 rows=1 width=8) (actual time=1.697..1.698 rows=1 loops=1)
   ->  Index Only Scan using games_game_btree_idx on games  (cost=0.53..1096.36 rows=1140 width=0) (actual time=0.969..1.690 rows=8 loops=1)
         Filter: hasopening(game, '1.e4 c6 2.c4 d5 3.exd5 cxd5 '::chessgame)
         Rows Removed by Filter: 3411
         Heap Fetches: 0
 Planning Time: 0.095 ms
 Execution Time: 1.714 ms
(7 rows)

Time: 2.170 ms
```

```
Test 6.3: with btree index and hasOpening2() function
                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=1100.64..1100.65 rows=1 width=8) (actual time=67.531..67.532 rows=1 loops=1)
   ->  Index Only Scan using games_game_btree_idx on games  (cost=0.53..1096.36 rows=1710 width=0) (actual time=38.566..67.520 rows=8 loops=1)
         Filter: (game ~~ '1.e4 c6 2.c4 d5 3.exd5 cxd5 '::chessgame)
         Rows Removed by Filter: 3411
         Heap Fetches: 0
 Planning Time: 0.100 ms
 Execution Time: 67.575 ms
(7 rows)

Time: 68.061 ms
```

Indexes B-tree

Query plan with operators and B-Tree index

This query has the same logic as hasOpening() but it is by hand

```
chess=# EXPLAIN ANALYZE SELECT * from games where game > '1.e4 e6 ' and game < '1.e4 e7 ';
                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Index Only Scan using games_game_btree_idx on games  (cost=0.13..4.15 rows=1 width=72) (actual time=0.013..0.014 rows=2 loops=1)
   Index Cond: ((game > '1.e4 e6 '::chessgame) AND (game < '1.e4 e7 '::chessgame))
   Heap Fetches: 0
 Planning Time: 0.046 ms
 Execution Time: 0.025 ms
(5 rows)
```

```
chess=# EXPLAIN ANALYZE SELECT * from games where game ~~ '1e4 e6';
                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Index Only Scan using games_game_btree_idx on games  (cost=0.13..8.22 rows=2 width=72) (actual time=0.049..0.050 rows=0 loops=1)
   Filter: (game ~~ '1e4 e6'::chessgame)
   Rows Removed by Filter: 5
   Heap Fetches: 0
 Planning Time: 0.051 ms
 Execution Time: 0.076 ms
(6 rows)
```

Indexes GIN

Operators

chess--1.0.sql

```
CREATE OPERATOR && (
  LEFTARG = _chessboard, RIGHTARG = _chessboard,
  PROCEDURE = _chessboard_overlap,
  COMMUTATOR = &&, NEGATOR = <>
);

CREATE OPERATOR @> (
  LEFTARG = _chessboard, RIGHTARG = _chessboard,
  PROCEDURE = chessboard_contains,
  COMMUTATOR = <@, NEGATOR = <>
);
```

```
CREATE OPERATOR <@ (
  LEFTARG = _chessboard, RIGHTARG = _chessboard,
  PROCEDURE = _chessboard_contained,
  COMMUTATOR =  @>, NEGATOR = <>
);

CREATE OPERATOR = (
  LEFTARG = _chessboard, RIGHTARG = _chessboard,
  PROCEDURE =  _chessboard_eq,
  COMMUTATOR = =, NEGATOR = <>
);
```

Indexes GIN

Operators

chess--1.0.sql

```
CREATE OR REPLACE FUNCTION
 _chessboard_contains(_chessboard, _chessboard)
 RETURNS boolean
 AS 'MODULE_PATHNAME'
 LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

chess_gin.c

```
PG_FUNCTION_INFO_V1(_chessboard_contains);
Datum
_chessboard_contains(PG_FUNCTION_ARGS)
{
    ArrayType *a = PG_GETARG_ARRAYTYPE_P(0);
    ArrayType *b = PG_GETARG_ARRAYTYPE_P(1);
    PG_RETURN_BOOL(_chessboard_contains_internal(a, b));
}
```

Operators : Implementation

Each operator has been implemented the same way. Let's take a closer look on the *contains* operator

Indexes GIN

chess_gin.c

```c
static bool
_chessboard_contains_internal(ArrayType *a, ArrayType *b)
{
    chessboard_t **_chessboard_1;
    chessboard_t **_chessboard_2;
    bool *elementNulls1;
    bool *elementNulls2;
    int numElements1;
    int numElements2;
    deconstruct_array(a, TypenameGetTypid("chessboard"), -1, false, 'i', (Datum **)&_chessboard_1, &elementNulls1, &numElements1);
    deconstruct_array(b, TypenameGetTypid("chessboard"), -1, false, 'i', (Datum **)&_chessboard_2, &elementNulls2, &numElements2);
    for (int i = 0; i < ARRNELEMS(b); ++i) {
        bool found = false;
        for (int j = 0; j < ARRNELEMS(a); ++j) {
            if (chessboard_cmp_internal(_chessboard_1[j], _chessboard_2[i]) == 0) {
                found = true;
                break;
            }
        }
        if (!found) {
            pfree(_chessboard_1);
            pfree(_chessboard_2);
            return false;
        }
    }
    pfree(_chessboard_1);
    pfree(_chessboard_2);
    return true;
}
```

Indexes GIN

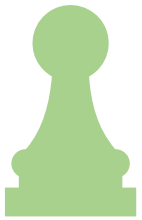
Query plan with hasBoard and hasBoard2 predicates and Gin index

```
chess=# EXPLAIN ANALYZE SELECT count(*) FROM games WHERE hasboard(game, 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1', 10);
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=10000000257.59..10000000257.60 rows=1 width=8) (actual time=811.927..811.929 rows=1 loops=1)
   ->  Seq Scan on games  (cost=10000000000.00..10000000254.74 rows=1140 width=0) (actual time=71.743..811.663 rows=3419 loops=1)
         Filter: hasboard(game, 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1'::chessboard, 10)
 Planning Time: 0.054 ms
 JIT:
   Functions: 4
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 0.348 ms, Inlining 49.091 ms, Optimization 12.616 ms, Emission 9.870 ms, Total 71.925 ms
 Execution Time: 833.004 ms
(9 rows)
```

```
chess=# EXPLAIN ANALYZE SELECT count(*) FROM games WHERE hasboard2(game, 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1', 10);
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=272.24..272.25 rows=1 width=8) (actual time=1.096..1.097 rows=1 loops=1)
   ->  Bitmap Heap Scan on games  (cost=30.32..267.97 rows=1710 width=0) (actual time=0.533..0.878 rows=3419 loops=1)
         Recheck Cond: (chessgame_to_chessboards(game) @> '{"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"}'::chessboard[])
         Heap Blocks: exact=212
         ->  Bitmap Index Scan on games_game_gin_idx  (cost=0.00..29.89 rows=1710 width=0) (actual time=0.484..0.484 rows=3419 loops=1)
               Index Cond: (chessgame_to_chessboards(game) @> '{"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"}'::chessboard[])
 Planning Time: 0.546 ms
 Execution Time: 1.224 ms
(8 rows)
```

Conclusion

- Storage of chessgames
- Datatypes are optimized
- Query execution time is reduced by use of indexes on operators
- Operators on datatypes are able to compare datatypes



- Predicates uses IndexOnlyScan but with a filter and not a index condition