

A close-up photograph of a chessboard with a black king and a white pawn in the foreground, and other pieces blurred in the background. The text 'Chess extension PostgreSQL' is overlaid in the center.

# Chess extension PostgreSQL

Younes El Mokhtari

Alexandre Bienfait

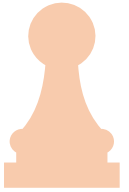
Brenno Ferreira

Philippe Mutkowski

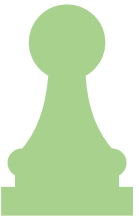
# Structure



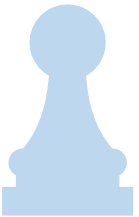
Chessgame type



Chessboard type



Functions and predicates



Indexes: B-tree and GIN

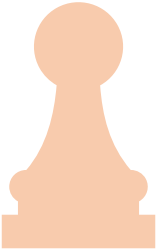
# ChessBoard Type



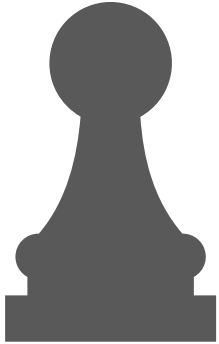
chess.c

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

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



# ChessGame Type



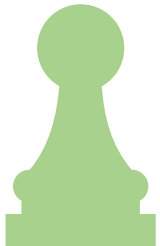
chess.c

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

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



# Functions and predicates



chess.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);
}
```

```
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);
}
```

# Functions and predicates

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);
}
```

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

Two versions of the hasOpening method.

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

# Functions and predicates

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
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);
}
```

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

Two versions of the hasBoard method

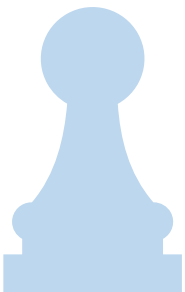
- The C one is complete, but doesn't use the index
- The SQL one uses the index but falls short of the requirements

# 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:

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

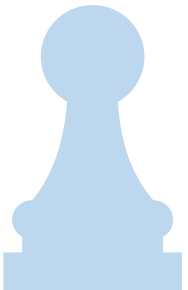


# Indexes

B-tree

## Operators : Implementation

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



```
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 *hasOpening()*.

```
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;
```

```
    }
```

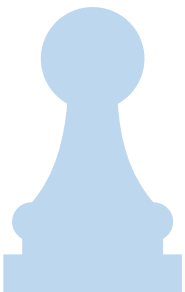
```
    return 0;
```

```
}
```

# Indexes

B-tree

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

# Indexes GIN

chess--1.0.sql

## Operators

```
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

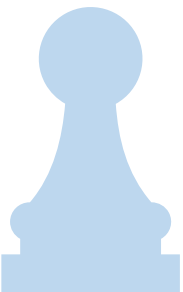
chess--1.0.sql

```
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));
}
```

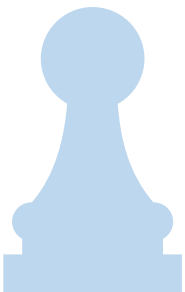


## Operators : Implementation

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

# Indexes GIN

chess\_gin.c



```
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, TypenameGetTypeId("chessboard"), -1, false, 'i', (Datum **)&_chessboard_1, &elementNulls1, &numElements1);
    deconstruct_array(b, TypenameGetTypeId("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;
}
```

# Conclusion

