

INFOH417 Database System Architecture

Chess Extension for PostgreSQL

By Cuellar Sanchez Edgardo, Dubois Alexandre,
Mema Klevis, and Hieu Nguyen Minh

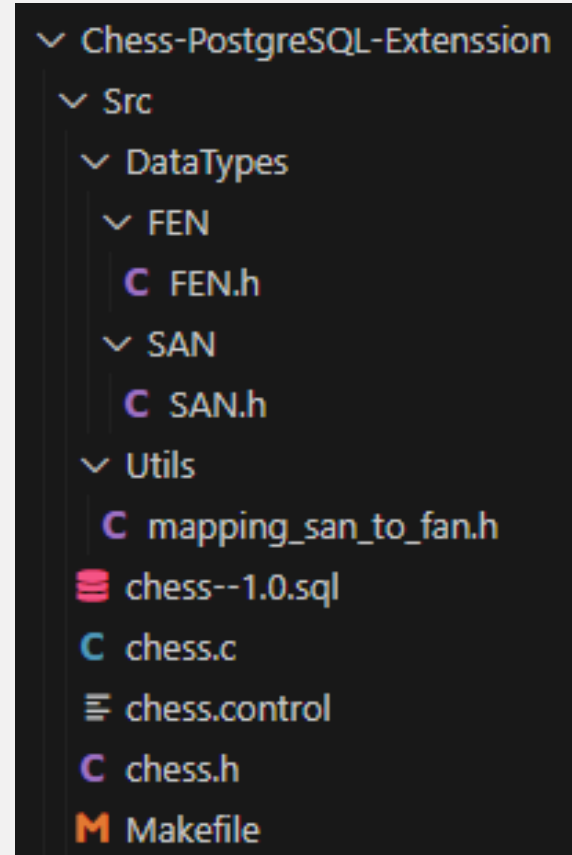
Outline



- Project Structure
- Data types (SAN, FAN)
- Functions (getBoard, getFirstMoves, hasOpening, hasBoard)
- Indexes (Btree for hasOpening, GIN for hasBoard)

Project Structure

- Project Structure
 - Data types: `FEN.h`, `SAN.h`
 - `mapping_san_to_fen.h`: A header containing a utility function `san_to_fan` written with Python embedded code
 - `chess-1.0.sql`, `chess.h`, `chess.c`, `chess.control`: files for defining the Postgres extension named `chess`, version 1.0
 - `makefile`: for installing Postgres extensions



Data Types

Standard Algebraic Notation (SAN)

1. Define the SAN type with struct

```
typedef struct {  
    char data[MAX_PGN_LENGTH]; // Array to store the SAN string.  
} SAN;
```

2. Define two functions `parseStr_ToPGN` and `parsePGN_ToStr` to convert a string (written in SAN format) to SAN data type and vice versa.

```
void parseStr_ToPGN(const char *pgn, SAN *game) {  
    game->data[MAX_PGN_LENGTH - 1] = '\0'; // Ensure null-termination.  
    strncpy(game->data, pgn, MAX_PGN_LENGTH - 1); // Copy the string.  
}
```

```
void parsePGN_ToStr(SAN *game, char **result){  
    *result = (char *)malloc(strlen(game->data) + 1); // Allocate memory for the string.  
    strcpy(*result, game->data); // Copy the SAN data.  
}
```

3. Define a function `truncate_san` to get N half moves of a SAN (to be used in `getFirstMoves` function)

Idea:

- Loop over the input SAN
- Append the SAN token to result
- Count the half moves
- Stop when reaching the desired n half moves.

```
SAN *truncate_san(SAN *inputGame, int nHalfMoves) {  
    SAN *resultGame = (SAN *) malloc(sizeof(SAN)); // Allocate memory for the result SAN structure.  
    int halfMoveCount = 0; // Counter for the number of half-moves processed.  
    char *token, *rest, *inputCopy; // Pointers for string tokenization.  
    bool isMoveNumber; // Flag to check if the token is a move number.  
  
    // Create a copy of the input SAN string to tokenize.  
    inputCopy = malloc(strlen(inputGame->data) + 1);  
    strcpy(inputCopy, inputGame->data); // Copy the SAN string.  
    rest = inputCopy; // Set the rest pointer for strtok_r usage.  
  
    resultGame->data[0] = '\0'; // Initialize the result SAN string.  
  
    // Tokenize the SAN string and process each token.  
    while ((token = strtok_r(rest, " ", &rest)) && halfMoveCount < nHalfMoves) {  
        rest = skipCommentsAndAnnotations(rest); // Skip any comments and annotations.  
        isMoveNumber = (strchr(token, '.') != NULL); // Check if the token is a move number.  
        strcat(resultGame->data, token); // Append the token to the result SAN string.  
        strcat(resultGame->data, " "); // Append a space after the token.  
  
        // Increment the half-move count if the token is not a move number.  
        if (!isMoveNumber)  
            halfMoveCount++;  
    }  
  
    // Free the memory allocated for the input copy.  
    pfree(inputCopy);  
  
    // Check if the desired number of half-moves was reached.  
    if (halfMoveCount < nHalfMoves){  
        pfree(resultGame); // Free the result structure if not.  
        return NULL; // Return NULL to indicate insufficient half-moves.  
    }  
  
    return resultGame; // Return the truncated SAN structure.  
}
```

Data Types

Forsyth–Edwards Notation (FEN)

1. Define the FEN type with struct

```
typedef struct
{
    char positions[MAX_FEN_LENGTH];
    char turn;
    char castling[5];
    char en_passant[3];
    int halfmove_clock;
    int fullmove_number;
} FEN;
```

2. Define two functions `parseStr_ToFEN` and `parseFEN_ToStr` to convert a string (written in FEN format) to FEN data type and vice versa.

```
void parseStr_ToFEN(const char *fenStr, FEN *result){
    if (sscanf(fenStr, "%s %c %s %s %d %d",
               result->positions,
               &result->turn,
               result->castling,
               result->en_passant,
               &result->halfmove_clock,
               &result->fullmove_number) != 6)
    {
        // Raise an error if the FEN string doesn't match the expected format.
        ereport(ERROR,
                (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
                 errmsg("failed to parse FEN string: %s", fenStr)));
    }
}
```

```
char* parseFEN_ToStr(const FEN *cb){
    static char result[MAX_FEN_LENGTH];

    // Format the FEN structure into a string using snprintf for safe string handling.
    snprintf(result, MAX_FEN_LENGTH, "%s %c %s %s %d %d",
             cb->positions,
             cb->turn,
             cb->castling,
             cb->en_passant,
             cb->halfmove_clock,
             cb->fullmove_number);

    return result;
}
```

3. Define a function `isValidFEN` to check if the input string has the FEN format

```
static bool isValidFEN(const char *fen)
{
    int ret;
    /* Regular expression pattern for FEN validation */
    static const char *fen_pattern = "^[KQRBNPkqrbnp1-8]+(/[KQRBNPkqrbnp1-8]+){7} [wb] (-|([KQkq]+) (-|[a-h36]+) \\d+ \\d+)$";

    /* Compile the regular expression pattern */
    regex_t re;
    if (regcomp(&re, fen_pattern, REG_EXTENDED | REG_NOSUB) != 0)
        return false;

    /* Execute the regular expression and check if it matches the FEN string */
    ret = regexec(&re, fen, 0, NULL, 0);

    /* Free the regex resources */
    regfree(&re);

    /* Temporary fix, always returns true for now */
    ret = 0; // TODO: Address the temporary fix.

    /* Return true if the FEN string matches the pattern */
    return (ret == 0);
}
```

Functions

Utility function `san_to_fen`

Input: (SAN) gamegame

Output: (FEN) chessboard

Idea:

- Use a Python library named `chess` (installed with `pip install chess`) to convert SAN to FEN.
- Declare `#include <python3.11/Python.h>` to use Python embedded code in C code.
- Write the Python embedded code with `PyRun_SimpleString`, in which we define a function `get_fen_from_san`.
- In the main C code, get the Python function using `PyObject_GetAttrString`.
- Get the result using `PyObject_CallObject`, then convert to C string using `PyUnicode_AsUTF8`

```
const char* san_to_fen(SAN *gameTruncated)
{
    // Initialize variables for Python interaction.
    PyObject *pModule, *pFunc, *pArgs, *pValue;
    const char *result = NULL;

    // Initialize the Python interpreter.
    Py_Initialize();

    // Define the Python function for SAN to FEN conversion.
    PyRun_SimpleString(
        "import chess\n"
        "import chess.pgn\n"
        "import io\n"
        "def get_fen_from_san(san):\n"
        "    if not san.strip():\n"
        "        return 'rnbqkbnr/pppppppp/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1' # Default FEN for empty input\n"
        "    game = chess.pgn.read_game(io.StringIO(san))\n"
        "    board = game.board()\n"
        "    for move in game.mainline_moves():\n"
        "        board.push(move)\n"
        "    return board.fen()\n"
    );

    // Add the defined Python function to the '__main__' module.
    pModule = PyImport_AddModule("__main__");

    // Check if the Python function is loaded and callable.
    if (pModule != NULL) {
        pFunc = PyObject_GetAttrString(pModule, "get_fen_from_san");
        if (pFunc && PyCallable_Check(pFunc)) {
            // Prepare the arguments for the Python function call.
            pArgs = PyTuple_New(1);
            PyTuple_SetItem(pArgs, 0, PyUnicode_FromString(gameTruncated->data));

            // Call the Python function and retrieve the result.
            pValue = PyObject_CallObject(pFunc, pArgs);
            Py_DECREF(pArgs);

            if (pValue != NULL) {
                // Convert the Python string result to C string.
                result = PyUnicode_AsUTF8(pValue);
                Py_DECREF(pValue);
            } else {
                // Handle Python call failure.
                Py_DECREF(pFunc);
                PyErr_Print();
                ereport(ERROR, (errmsg("Call to 'get_fen_from_san' failed")));
            }
        } else {
            // Handle errors related to function loading or callability.
            if (PyErr_Occurred())
                PyErr_Print();
            ereport(ERROR, (errmsg("Cannot find function 'get_fen_from_san'")));
        }
        Py_XDECREF(pFunc);
    } else {
        // Handle errors in loading the '__main__' module.
        PyErr_Print();
        ereport(ERROR, (errmsg("Failed to load '__main__' module")));
    }

    // Finalize the Python interpreter.
    Py_Finalize();

    return result;
}
```

Functions

getBoard(**get_board_state**)

Input:

- (SAN) chessgame
- (int) N: number of half moves

Output:

- (FEN) board state after N half moves

Idea:

- Use the function `truncate_san` (defined in `SAN.h`) to get the truncated SAN after N half moves
- Convert to FEN type using the function `san_to_fan` (defined in `mapping_san_to_fan.h`)

```
Datum get_board_state(PG_FUNCTION_ARGS) {
    FEN *fen;
    SAN *gameTruncated, *game;

    int half_moves;
    const char *fenConversionStrResult;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("get_board_state: One of the arguments is null")));

    game = (SAN *) PG_GETARG_POINTER(0);
    half_moves = PG_GETARG_INT32(1);

    if (half_moves < 0)
        ereport(ERROR, (errmsg("get_board_state: Non-positive number of half moves")));

    gameTruncated = truncate_san(game, half_moves);

    if (gameTruncated == NULL)
        ereport(ERROR, (errmsg("Game is incomplete or shorter than the requested number of half-moves")));

    fenConversionStrResult = san_to_fan(gameTruncated);

    if (fenConversionStrResult == NULL) {
        ereport(ERROR, (errmsg("No FEN result returned from mapping san to fen")));
    }

    fen = (FEN *) palloc(sizeof(FEN));

    parseStr_ToFEN(fenConversionStrResult, fen);

    PG_RETURN_POINTER(fen);
}
```

Functions

getFirstMoves (get_firstMoves)

Input:

- (SAN) chessgame
- (int) N: number of half moves

Output:

- (SAN) chessgame after N half moves

Idea:

- Similar to the function `get_board_state`, but do not need to convert the result to FEN.
- Return the output right after using the function `truncate_san`.

```
Datum get_FirstMoves(PG_FUNCTION_ARGS)
{
    SAN *result, *inputGame;
    int nHalfMoves;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("get_FirstMoves: One of the arguments is null")));

    inputGame = (SAN *) PG_GETARG_POINTER(0);
    nHalfMoves = PG_GETARG_INT32(1);

    if (nHalfMoves < 0)
        ereport(ERROR, (errmsg("get_FirstMoves: Non-positive number of half moves")));

    result = truncate_san(inputGame, nHalfMoves);

    if (result == NULL)
        ereport(ERROR, (errmsg("Game is incomplete or shorter than the requested number of half-moves")));

    PG_RETURN_POINTER(result);
}
```


Functions

hasOpening(**has_opening**)

Input:

- (SAN) chessgame 1
- (SAN) chessgame 2

Output:

- (boolean) **True** if chessgame 1 starts with the same set of moves as chessgame 2.

Idea:

- Use the C built-in function `strncmp` to compare chessgame 1 with chessgame 2 in terms of beginning moves, where the compared string length is the length of chessgame 2.
- Get **True** if chessgame 2 is included in the beginning moves of chessgame 1.

```
Datum has_opening(PG_FUNCTION_ARGS)
{
    int opening_length;
    SAN *game1, *game2;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("has_opening: One of the arguments is null")));

    game1 = (SAN*)PG_GETARG_POINTER(0);
    game2 = (SAN*)PG_GETARG_POINTER(1);

    opening_length = strlen(game2->data);

    if (opening_length > 0 && strncmp(game1->data, game2->data, opening_length) == 0)
        PG_RETURN_BOOL(true);

    PG_RETURN_BOOL(false);
}
```

Functions

hasBoard (has_Board)

Input:

- (SAN) chessgame
- (FEN) chessboard
- (int) N half moves

Output:

- (boolean) **True** if chessgame contains chessboard in its first N half-moves.

Idea:

- Truncate the chessgame after N half moves with the function `truncate_san`.
- Convert to FEN with the function `san_to_fen`
- Use the C built-in function `strcmp` to compare the converted FEN with input chessboard
- Get **True** if they are the same

```
Datum has_Board(PG_FUNCTION_ARGS){
    FEN *current_board, *input_board;
    SAN *gameTruncated, *input_game;

    int input_half_moves;
    bool positions_match;
    const char *fenConversionStrResult;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1) || PG_ARGISNULL(2))
        ereport(ERROR, (errmsg("has_Board: One of the arguments is null")));

    input_game = (SAN*) PG_GETARG_POINTER(0);
    input_board = (FEN*) PG_GETARG_POINTER(1);
    input_half_moves = PG_GETARG_INT32(2);

    if (input_half_moves < 0)
        ereport(ERROR, (errmsg("hasBoard: Non-positive number of half moves")));

    gameTruncated = truncate_san(input_game, input_half_moves);

    if (gameTruncated == NULL)
        ereport(ERROR, (errmsg("Game is incomplete or shorter than the requested number of half-moves")));

    fenConversionStrResult = san_to_fan(gameTruncated);

    if (fenConversionStrResult == NULL) {
        ereport(ERROR, (errmsg("No FEN result returned from mapping san to fen")));
    }

    current_board = (FEN *)palloc(sizeof(FEN));

    parseStr_ToFEN(fenConversionStrResult, current_board);

    positions_match = strcmp(input_board->positions, current_board->positions) == 0;

    PG_RETURN_BOOL(positions_match);
}
```

Indexes

B-Tree Index

6 operations: less than (`san_lt`), equal (`san_eq`), greater than (`san_gt`), comparator (`san_cmp`), like (`san_like`), not like (`san_not_like`)

Beforehand, define a utility function `san_compare` to compare two SAN strings

Input: (SAN) `a`, (SAN) `b`

Output: (int) `-1`, `0`, or `1`

Idea: Simply use the C built-in function `strncmp` to compare two SAN strings, where `MAX_PGN_LENGTH` is a pre-defined variable.

```
static int san_compare(SAN *a, SAN *b)
{
    int cmp_result;

    cmp_result = strncmp(a->data, b->data, MAX_PGN_LENGTH);

    if (cmp_result < 0)
        return -1;
    else if (cmp_result > 0)
        return 1;
    else
        return 0;
}
```

This function will be used in B-tree operations.

Indexes

B-Tree Index

`san_lt`

Input: (SAN) a, (SAN), b

Output: (boolean) **True** if SAN string a is less than SAN string b

Get the result by checking if the returned value of `san_compare(a, b)` is less than 0

```
Datum san_lt(PG_FUNCTION_ARGS)
{
    SAN *a, *b;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("san_lt: One of the arguments is null")));

    a = (SAN *) PG_GETARG_POINTER(0);
    b = (SAN *) PG_GETARG_POINTER(1);

    PG_RETURN_BOOL(san_compare(a, b) < 0);
}
```

`san_eq`

The same as `san_lt`, except that the checking condition is equal to 0

```
Datum san_eq(PG_FUNCTION_ARGS)
{
    SAN *a, *b;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        PG_RETURN_BOOL(false);

    a = (SAN *) PG_GETARG_POINTER(0);
    b = (SAN *) PG_GETARG_POINTER(1);

    PG_RETURN_BOOL(san_compare(a, b) == 0);
}
```

`san_gt`

The same as `san_lt`, except that the checking condition is greater than 0

```
Datum san_gt(PG_FUNCTION_ARGS)
{
    SAN *a, *b;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("san_gt: One of the arguments is null")));

    a = (SAN *) PG_GETARG_POINTER(0);
    b = (SAN *) PG_GETARG_POINTER(1);

    PG_RETURN_BOOL(san_compare(a, b) > 0);
}
```

Indexes

B-Tree Index

san_cmp

Get the result directly from the returned value of the function

san_compare(a, b)

```
Datum san_cmp(PG_FUNCTION_ARGS)
{
    SAN *a, *b;
    int cmp_result;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("san_cmp: One of the arguments is null")));

    a = (SAN *) PG_GETARG_POINTER(0);
    b = (SAN *) PG_GETARG_POINTER(1);

    cmp_result = san_compare(a, b);

    PG_RETURN_INT32(cmp_result);
}
```

san_like

Use Postgres built-in function DirectFunctionCall2 (defined in fmgr.h) to call the function textlike, taking

```
Datum san_like(PG_FUNCTION_ARGS)
{
    SAN *san;
    text *pattern, *san_text;

    bool result;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("san_like: One of the arguments is null")));

    san = (SAN *) PG_GETARG_POINTER(0);
    pattern = PG_GETARG_TEXT_PP(1);
    san_text =cstring_to_text(san->data);

    result = DatumGetBool(DirectFunctionCall2(textlike,
                                                PointerGetDatum(san_text),
                                                PointerGetDatum(pattern)));

    pfree(san_text);

    PG_FREE_IF_COPY(san, 0);
    PG_FREE_IF_COPY(pattern, 1);

    PG_RETURN_BOOL(result);
}
```

san_not_like

The same as san_like, but return negation.

```
Datum san_not_like(PG_FUNCTION_ARGS)
{
    SAN *san;
    text *pattern, *san_text;

    bool like_result;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("san_not_like: One of the arguments is null")));

    san = (SAN *) PG_GETARG_POINTER(0);
    pattern = PG_GETARG_TEXT_PP(1);
    san_text =cstring_to_text(san->data);

    like_result = DatumGetBool(DirectFunctionCall2(textlike,
                                                PointerGetDatum(san_text),
                                                PointerGetDatum(pattern)));

    pfree(san_text);

    PG_FREE_IF_COPY(san, 0);
    PG_FREE_IF_COPY(pattern, 1);

    PG_RETURN_BOOL(!like_result);
}
```

Indexes

GIN Index

The idea behind this GIN index is indexing and querying on chess game positions by extending the GIN index for the chess game type.

8 functions to try to implement this index:

`fens_from_san`

Extracts FEN strings from a SAN representation of a chess game

Input: (SAN) chessgame

Output: (FEN) chessboard

`gin_compare`

Compares two text values for GIN indexing

Input: Two text values
(keys)

Output: (int) -1, 0, or 1
for the ordering
of the two keys.

```
Datum gin_compare(PG_FUNCTION_ARGS)
{
    text *key1 = PG_GETARG_TEXT_PP(0);
    text *key2 = PG_GETARG_TEXT_PP(1);

    char *key1Str = text_to_cstring(key1);
    char *key2Str = text_to_cstring(key2);

    int32 result = DatumGetInt32(DirectFunctionCall2Coll(btint4cmp,
        PG_GET_COLLATION(),
        PointerGetDatum(key1),
        PointerGetDatum(key2)));

    pfree(key1Str);
    pfree(key2Str);

    PG_RETURN_INT32(result);
}
```

```
Datum fens_from_san(PG_FUNCTION_ARGS){
    int32 *nkeys;
    ArrayBuildState *astate;
    SAN *san, *gameTruncated;

    int i;
    const char *fenConversionStrResult;

    san = (SAN *) PG_GETARG_POINTER(0);
    nkeys = (int32 *) PG_GETARG_POINTER(1);

    i = 0;
    *nkeys = 0;
    astate = NULL;

    while (true) {

        gameTruncated = truncate_san(san, i);

        if (gameTruncated == NULL)
            break;

        fenConversionStrResult = san_to_fen(gameTruncated);

        if (fenConversionStrResult == NULL)
            ereport(ERROR, (errmsg("No FEN result returned from mapping san to fen")));

        astate = accumArrayResult(astate, CStringGetTextDatum(fenConversionStrResult),
            false, TEXTOID, CurrentMemoryContext);

        (*nkeys)++;
        i++;
    }

    if (*nkeys > 0) {

        Datum *keys = (Datum *) palloc(*nkeys * sizeof(Datum));

        for (i = 0; i < *nkeys; i++) {
            keys[i] = astate->dvalues[i];
        }

        PG_RETURN_POINTER(keys);
    } else {
        PG_RETURN_NULL();
    }
}
```

Indexes

GIN Index

gin_extract_value

Prepares indexable keys for GIN indexing from a SAN

Input: (SAN) chessgame

Output: Array of keys (Datum)

gin_extract_query

Extracts a query key from a FEN type for GIN indexing searches

Input: (FEN) chessboard, array of keys (Datum), search mode

Output: Array of keys (Datum)

gin_consistent

Checks if indexed keys are consistent with a query key in GIN index searches

Input: (Boolean Array) check, (Query Key) query, (Number of Keys)

nkeys, (Recheck Flag) recheck, (Array of Query Keys) queryKeys

Output: (Boolean) True if the result of the comparison between the query key and indexed keys is less than 0; otherwise, False

```
Datum gin_extract_query(PG_FUNCTION_ARGS) {
    FEN *itemValue;
    Datum *keys, query;
    int32 *nkeys, *searchMode;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1) || PG_ARGISNULL(2) || PG_ARGISNULL(3) ||
        PG_ARGISNULL(4) || PG_ARGISNULL(5) || PG_ARGISNULL(6)) {
        ereport(ERROR, (errmsg("gin_extract_query: One of the arguments is null")));
    }

    query = PG_GETARG_DATUM(0);
    nkeys = (int32 *) PG_GETARG_POINTER(1);
    itemValue = (FEN *) DatumGetPointer(query);
    searchMode = (int32 *) PG_GETARG_POINTER(6);

    *nkeys = 1;
    keys = (Datum *) palloc(*nkeys * sizeof(Datum));
    keys[0] = CStringGetTextDatum(itemValue->positions);

    *searchMode = GIN_SEARCH_MODE_DEFAULT;

    PG_RETURN_POINTER(keys);
}
```

```
Datum gin_extract_value(PG_FUNCTION_ARGS) {
    SAN *san;
    Datum *keys;
    int32 *nkeys;
    bool **nullFlags;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1) || PG_ARGISNULL(2))
        ereport(ERROR, (errmsg("gin_extract_value: One of the arguments is null")));

    san = (SAN *) PG_GETARG_POINTER(0);
    nkeys = (int32 *) PG_GETARG_POINTER(1);
    nullFlags = (bool **) PG_GETARG_POINTER(2);

    *nkeys = 0;

    keys = (Datum *) DirectFunctionCall4Coll(fens_from_san,
                                              PG_GET_COLLATION(),
                                              PointerGetDatum(san),
                                              PointerGetDatum(nkeys),
                                              BoolGetDatum(false),
                                              PointerGetDatum(NULL));

    *nullFlags = NULL;

    PG_RETURN_POINTER(keys);
}
```

```
Datum gin_consistent(PG_FUNCTION_ARGS)
{
    bool *check = (bool *) PG_GETARG_POINTER(0);
    Datum query = PG_GETARG_DATUM(2);
    int32 nkeys = PG_GETARG_INT32(3);
    bool *recheck = (bool *) PG_GETARG_POINTER(5);
    Datum *queryKeys = (Datum *) PG_GETARG_POINTER(6);

    FEN queryFen;
    char *queryFenStr = text_to_cstring(DatumGetTextP(query));
    parseStr_ToFEN(queryFenStr, &queryFen);
    pfree(queryFenStr);

    for (int i = 0; i < nkeys; i++) {
        if (check[i]) {
            FEN keyFen;
            char *keyFenStr = text_to_cstring(DatumGetTextP(queryKeys[i]));
            parseStr_ToFEN(keyFenStr, &keyFen);
            pfree(keyFenStr);

            if (strcmp(queryFen.positions, keyFen.positions) == 0) {
                *recheck = true;
                PG_RETURN_BOOL(true);
            }
        }
    }

    *recheck = true;
    PG_RETURN_BOOL(false);
}
```

Indexes

GIN Index

`gin_tri_consistent`

Performs a ternary consistency check for GIN index operations

Input: (Array of Ternary Values) `check`, (Query Key) `query`,
Number of Keys) `nkeys`, (Array of Query Keys) `queryKeys`

Output: (Boolean) True if the result of the comparison
between the query key and indexed keys is less than 0;
otherwise, False.

```
Datum gin_tri_consistent(PG_FUNCTION_ARGS)
{
    GinTernaryValue *check = (GinTernaryValue *) PG_GETARG_POINTER(0);
    Datum query = PG_GETARG_DATUM(2);
    int32 nkeys = PG_GETARG_INT32(3);
    Datum *queryKeys = (Datum *) PG_GETARG_POINTER(5);

    GinTernaryValue result = GIN_MAYBE;

    for (int i = 0; i < nkeys; i++) {
        if (check[i] == GIN_FALSE)
            return GIN_FALSE;

        if (check[i] == GIN_TRUE) {
            char *queryFenStr = TextDatumGetCString(queryKeys[i]);
            char *gameFenStr = TextDatumGetCString(query);

            if (strcmp(queryFenStr, gameFenStr) == 0)
                result = GIN_TRUE;
        }
    }

    PG_RETURN_GIN_TERNARY_VALUE(result);
}
```

`has_board_fn_operator` and `fen_in_san_eq`

Both determine if a given FEN type matches any board
state in a SAN type

Input: (FEN) `a`, (SAN) `b`.

Output: (Boolean) True if the result of the comparison
between FEN string `a` and the SAN string from the indexed key
(using `san_to_fen`) is less than 0; otherwise, False.

```
Datum has_board_fn_operator(PG_FUNCTION_ARGS)
{
    FEN *input_fen, *result_fen;
    SAN *san, *gameTruncated;

    int i;
    bool result;
    const char *fenConversionStrResult;

    if (PG_ARGISNULL(0) || PG_ARGISNULL(1))
        ereport(ERROR, (errmsg("has_board_fn_operator: One of the arguments is null\n")));

    result_fen = (FEN *) palloc(sizeof(FEN));
    input_fen = (FEN *) PG_GETARG_POINTER(1);
    san = (SAN *) PG_GETARG_CHESSGAME_P(0);

    i = 0;
    result = false;

    while (true) {
        gameTruncated = truncate_san(san, i);

        if (gameTruncated == NULL)
            break;

        fenConversionStrResult = san_to_fen(gameTruncated);

        if (fenConversionStrResult == NULL)
            ereport(ERROR, (errmsg("has_board_fn_operator: No FEN result returned from mapping san to fen")));

        parseStr_ToFEN(fenConversionStrResult, result_fen);

        if (strcmp(result_fen->positions, input_fen->positions) == 0)
        {
            result = true;
            break;
        }

        i++;
    }

    PG_RETURN_BOOL(result);
}
```


Running Results

The screenshot displays a database IDE interface with the following components:

- Database Explorer:** Shows a tree view of the database structure, including 'ExtensionTest@localhost', 'ExtensionTest', 'public', 'tables', 'routines', 'operators', 'sequences', 'object types', 'operator classes', 'operator families', and 'Database Objects'.
- Console:** Contains SQL code for creating an index and running an explain analyze query.

```
209 CREATE INDEX game_notation_idx ON chess_games(game_notation)
210 WHERE game_notation LIKE '1. Nf3%';
211
212
213
214 EXPLAIN analyze SELECT * FROM chess_games WHERE game_notation LIKE '1. Nf3%';
215
216 /** IndexScan(chess_games game_notation_idx) */
217 explain analyze SELECT count(*) FROM chess_games WHERE has_opening(game_notation, '1. d3'::san);
218
```
- Services:** A list of services including 'Database', 'ExtensionTest@localhost', 'default 17 ms', and 'console 26 ms'.
- Output:** Displays the query plan for the executed query.

```
+ IndexScan(chess_ga...s game_notation_idx) x
7 rows
CSV
QUERY PLAN
1 Aggregate (cost=7047.99..7048.00 rows=1 width=8) (actual time=2.232..2.232 rows=1 loops=1)
2   -> Index Only Scan using game_notation_idx on chess_games (cost=0.66..7039.66 rows=3333 width=0) (actual time=1.145..2.213 rows=491 loops=1)
3     Filter: has_opening(game_notation, '1. d3'::san)
4     Rows Removed by Filter: 9509
5     Heap Fetches: 0
6 Planning Time: 0.058 ms
7 Execution Time: 2.251 ms
```

The status bar at the bottom indicates the current file is 'SUM: Not enough values' with line 217:5, column LF, UTF-8 encoding, and 4 spaces.

Running Results

The screenshot displays a database IDE interface with a dark theme. The top toolbar includes icons for file management, search, and execution. The left sidebar shows the 'Database Explorer' with a tree view of the 'ExtensionTest@localhost' database, including 'public' schema, 'tables', 'routines', 'operators', 'sequences', 'object types', 'operator classes', 'operator families', 'Database Objects', 'access methods', and 'casts'.

The main editor area shows a SQL script with the following content:

```
207 CREATE INDEX game_notation_idx ON chess_games using btree(game_notation san_ops);
208
209 CREATE INDEX game_notation_idx ON chess_games(game_notation)
210 WHERE game_notation LIKE '1. Nf3%';
211
212 EXPLAIN analyze SELECT * FROM chess_games WHERE game_notation LIKE '1. Nf3%';
213
214 /* IndexScan(chess_games game_notation_idx) */
215 explain analyze SELECT count(*) FROM chess_games WHERE has_opening(game_notation, '1. d3'::san);
216
217
```

The 'Services' panel at the bottom left shows a list of services, including 'default' (17 ms) and 'console' (43 ms). The 'Output' panel on the right displays the execution results for the query plan, showing a 'Seq Scan on chess_games' with a filter for 'has_opening(game_notation, '1. d3'::san)'. The results include the number of rows removed by the filter (9509) and the execution time (4.736 ms).

The bottom status bar indicates the current file is 'SUM: Not enough values' and shows the cursor position (217:5) and encoding (UTF-8).

Running Results

The screenshot displays a database IDE interface with a dark theme. The top bar shows the 'Documents' menu and a 'master' branch indicator. The left sidebar contains a 'Database Explorer' showing a tree structure of database objects, including 'ExtensionTest@localhost', 'ExtensionTest', 'public', 'tables', 'routines', 'operators', 'sequences', 'object types', 'operator classes', 'operator families', 'Database Objects', 'access methods', 'casts', 'extensions', 'chess', 'plpgsql', 'languages', 'virtual views', and 'Server Objects'. The main editor area is titled 'console' and shows a SQL query with line numbers 276 to 297. The query includes a filter, row removal, planning and execution times, and an 'EXPLAIN ANALYZE' statement. The bottom panel shows the 'Output' tab for the 'Partial Index Query test', displaying a query plan with 8 steps, including a Bitmap Heap Scan and a Bitmap Index Scan, with associated costs, row counts, and execution times. The status bar at the bottom indicates 'SUM: Not enough values', '293:4', 'LF', 'UTF-8', and '4 spaces'.

```
276      Filter: has_opening(game_notation, '1. c4 e6'::san)"
277      Rows Removed by Filter: 25617
278      Planning Time: 0.185 ms
279      Execution Time: 23.254 ms
280      */
281
282
283      -- Aggregate Queries test
284      -- need to turn off the seq scan to trigger the index scan
285      SET enable_seqscan = off;
286      EXPLAIN ANALYZE
287      SELECT COUNT(*)
288      FROM chess_games
289      WHERE has_opening(game_notation, '1. f4 e5');
290      SET enable_seqscan = on;
291
292      -- Partial Index Query test
293      EXPLAIN ANALYZE
294      SELECT * FROM chess_games
295      WHERE game_notation LIKE '1. Nf3%'
296      AND has_opening(game_notation, '1. Nf3 d5');
297
```

Output: Partial Index Query test

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Loops
1	Bitmap Heap Scan on chess_games	(cost=347.38..1851.38 rows=1667 width=1004)	1667	1004	(actual time=0.525..0.817 rows=23 loops=1)	23	1
2	Recheck Cond: (game_notation ~ '1. Nf3%'::text)						
3	Filter: has_opening(game_notation, '1. Nf3 d5'::san)						
4	Rows Removed by Filter: 466						
5	Heap Blocks: exact=426						
6	-> Bitmap Index Scan on game_notation_idx_partial	(cost=0.00..346.97 rows=5000 width=0)	5000	0	(actual time=0.466..0.466 rows=489 loops=1)	489	1
7	Planning Time: 0.128 ms						
8	Execution Time: 0.828 ms						

Database Consoles > ExtensionTest@localhost > console

SUM: Not enough values 293:4 LF UTF-8 4 spaces

Running Results

The screenshot displays a database IDE interface with a dark theme. The top bar shows the file explorer and a 'console' tab. The left sidebar, 'Database Explorer', shows a tree view of the database structure, including 'ExtensionTest@localhost', 'public', 'tables', 'routines', 'operators', 'sequences', 'object types', 'operator classes', 'operator families', 'Database Objects', 'access methods', 'casts', 'extensions', 'chess', 'plpgsql', 'languages', 'virtual views', and 'Server Objects'. The main editor area shows a SQL script with line numbers 276 to 297. The script includes a filter, row removal, planning and execution times, a comment about aggregate queries, a test to turn off the seq scan, an EXPLAIN ANALYZE statement, and a partial index query test. The bottom panel, 'Services', shows a tree view of the database services, including 'Database', 'ExtensionTest@localhost', 'default', and 'console'. The 'console' service is selected, and its output is displayed in the 'Output' tab. The output shows the query plan for the partial index query test, including the sequence of operations, filter, rows removed, planning time, and execution time.

```
276      Filter: has_opening(game_notation, '1. c4 e6)::san)"
277      Rows Removed by Filter: 25617
278      Planning Time: 0.185 ms
279      Execution Time: 23.254 ms
280      */
281
282      -- Aggregate Queries test
283      -- need to turn off the seq scan to trigger the index scan
284      SET enable_seqscan = off;
285      EXPLAIN ANALYZE
286      SELECT COUNT(*)
287      FROM chess_games
288      WHERE has_opening(game_notation, '1. f4 e5');
289      SET enable_seqscan = on;
290
291      -- Partial Index Query test
292      EXPLAIN ANALYZE
293      SELECT * FROM chess_games
294      WHERE game_notation LIKE '1. Nf3%'
295      AND has_opening(game_notation, '1. Nf3 d5');
```

Output: Partial Index Query test

QUERY PLAN
1 Seq Scan on chess_games (cost=0.00..1579.00 rows=1667 width=1004) (actual time=0.421..3.423 rows=23 loops=1)
2 Filter: ((game_notation ~ '1. Nf3% '::text) AND has_opening(game_notation, '1. Nf3 d5 '::san))
3 Rows Removed by Filter: 9977
4 Planning Time: 0.059 ms
5 Execution Time: 3.437 ms

Database Consoles > ExtensionTest@localhost > console

SUM: Not enough values 293:5 LF UTF-8 4 spaces

Running Results

The screenshot displays a database IDE interface with a dark theme. The top bar shows the file explorer with 'Documents' and 'master' selected. The main window is divided into three panes: Database Explorer, Console, and Services.

Database Explorer: Shows a tree view of the database structure. The 'ExtensionTest' database is expanded, showing 'public' schema with tables, routines, operators, sequences, object types, operator classes, and operator families. The 'Database Objects' section is also expanded, showing 'access methods', 'casts', 'extensions', 'chess', 'plpgsql', 'languages', and 'virtual views'.

Console: Displays the SQL script being executed. The script includes several SELECT statements and SET commands. The current line of execution is highlighted in green.

```
383 SET enable_seqscan = ON;
384
385
386 SELECT * FROM chess_games WHERE game_notation ~ 'e4%';
387 SELECT * FROM chess_games WHERE game_notation ~ '1. d4 d5%';
388
389 SELECT * FROM chess_games WHERE game_notation !~ 'e4%';
390
391 SELECT * FROM chess_games WHERE game_notation !~ '%Bb5%';
392
393 SELECT san_not_like('e4 e5 Nf3 Nc6 Bb5 a6 Ba4 Nf6 O-O Be7 Re1 b5 Bb3 d6 c3 O-O h3 Nb8'::SAN, 'e4%');
394
395 SET enable_seqscan = off;
396
397 explain analyze SELECT g.game_notation
398 FROM chess_games g, favorite_games f
399 WHERE has_opening(g.game_notation, get_firstmoves(f.game_notation, 4));
400
401 SET enable_seqscan = on;
```

Services: Shows the execution results and the query plan. The 'Output' pane displays the query plan for the SELECT statement. The 'Result 93' pane shows the execution time and other statistics.

Output:

```
1 Nested Loop (cost=0.00..176684.50 rows=3333333 width=1000) (actual time=87.225..2646.670 rows=46 loops=1)
2   Join Filter: has_opening(g.game_notation, get_firstmoves(f.game_notation, 4))
3   Rows Removed by Join Filter: 9999954
4   -> Seq Scan on chess_games g (cost=0.00..1529.00 rows=10000 width=1000) (actual time=0.015..1.595 rows=10000 loops=1)
5   -> Materialize (cost=0.00..158.00 rows=1000 width=1000) (actual time=0.001..0.031 rows=1000 loops=10000)
6       -> Seq Scan on favorite_games f (cost=0.00..153.00 rows=1000 width=1000) (actual time=6.984..7.251 rows=1000 loops=1)
7 Planning Time: 0.171 ms
8 JIT:
9   Functions: 7
10  Options: Inlining false, Optimization false, Expressions true, Deforming true
11 Timing: Generation 0.950 ms, Inlining 0.000 ms, Optimization 0.516 ms, Emission 6.477 ms, Total 7.944 ms
12 Execution Time: 2647.701 ms
```

Result 93: Shows the execution time and other statistics.

Database Consoles > ExtensionTest@localhost > console

SUM: Not enough values 316:6 LF UTF-8 4 spaces

Running Results

The screenshot displays a database IDE interface with a dark theme. The top section shows a SQL console with a series of queries executed on a database named 'ExtensionTest'. The queries include setting 'enable_seqscan' to ON, several SELECT statements with WHERE clauses, and an EXPLAIN ANALYZE statement. The EXPLAIN ANALYZE query is highlighted with a green box.

```
383 SET enable_seqscan = ON;
384
385
386 SELECT * FROM chess_games WHERE game_notation ~ 'e4%';
387 SELECT * FROM chess_games WHERE game_notation ~ '1. d4 d5%';
388
389 SELECT * FROM chess_games WHERE game_notation !~ 'e4%';
390
391 SELECT * FROM chess_games WHERE game_notation !~ '%Bb5%';
392
393 SELECT san_not_like('e4 e5 Nf3 Nc6 Bb5 a6 Ba4 Nf6 0-0 Be7 Re1 b5 Bb3 d6 c3 0-0 h3 Nb8'::SAN, 'e4%');
394
395 SET enable_seqscan = off;
396
397 explain analyze SELECT g.game_notation
398 FROM chess_games g, favorite_games f
399 WHERE has_opening(g.game_notation, get_firstmoves(f.game_notation, 4));
400
401 SET enable_seqscan = on;
```

The bottom section shows the 'Services' pane with a tree view of the database structure. The 'console' service is selected, and its output is displayed in the 'Output' pane. The output shows the query plan for the EXPLAIN ANALYZE query, including details about the Nested Loop, Join Filter, Rows Removed by Join Filter, Index Only Scan, Heap Fetches, Materialize, Seq Scan, Planning Time, JIT, Functions, Options, Timing, and Execution Time.

QUERY PLAN

1	Nested Loop (cost=10000000000.66..10000182170.16 rows=3333333 width=1000) (actual time=58.842..2524.482 rows=46 loops=1)
2	Join Filter: has_opening(g.game_notation, get_firstmoves(f.game_notation, 4))
3	Rows Removed by Join Filter: 9999954
4	-> Index Only Scan using game_notation_idx on chess_games g (cost=0.66..7014.66 rows=10000 width=1000) (actual time=0.012..2.853 rows=10000 loops=1)
5	Heap Fetches: 0
6	-> Materialize (cost=10000000000.00..1000000158.00 rows=1000 width=1000) (actual time=0.003..0.032 rows=1000 loops=10000)
7	-> Seq Scan on favorite_games f (cost=10000000000.00..10000000153.00 rows=1000 width=1000) (actual time=28.445..28.577 rows=1000 loops=1)
8	Planning Time: 0.070 ms
9	JIT:
10	Functions: 5
11	Options: Inlining true, Optimization true, Expressions true, Deforming true
12	Timing: Generation 0.195 ms, Inlining 6.092 ms, Optimization 14.835 ms, Emission 7.524 ms, Total 28.646 ms
13	Execution Time: 2524.735 ms

Database Consoles > ExtensionTest@localhost > console

SUM: Not enough values 316:4 LF UTF-8 4 spaces