DNA Sequence Database Extension

INFO-H417 Database System Architecture 2024-2025

Overview

In this project, your team will create a PostgreSQL extension for storing and analyzing DNA sequences, with a particular focus on k-mer analysis. K-mers are subsequences of DNA of length k, which are fundamental in many bioinformatics applications such as genome assembly, sequence alignment, and genome comparison. The extension will be implemented using C-Language functions and will add support for three new data types along with various functions and an index structure.

Data Types

The extension must implement three custom types to handle different aspects of DNA sequence analysis. For each type, favor an internal representation which optimizes the storage space, without jeopardizing the query time.

1. DNA Sequence Type (dna)

The `dna` type represents variable-length DNA sequences with no specified maximum length. It accepts the four standard nucleotides: A, C, G, T (case-insensitive). For example, a DNA sequence might be 'ACGTACGT' or 'GATTACA'.

2. K-mer Type (kmer)

The `kmer` type represents subsequences of DNA, with a maximum length of 32 nucleotides. Like the dna type, it only accepts the four standard nucleotides. For example, a 5-mer might be 'ACGTA' or 'GATTC'.

3. Query K-mer Type (qkmer)

The `qkmer` type represents pattern queries on k-mers using IUPAC nucleotide codes, allowing for ambiguous positions. It has the same length restrictions as the kmer type but accepts additional characters representing multiple possible nucleotides. For example, 'ANGTA' represents any 5-mer that starts with A, followed by any nucleotide, then G, T, and A. Other common IUPAC codes include R (A or G), Y (C or T), and N (any nucleotide).

Functions and Operators

The extension must provide several functions to manipulate these types:

- Basic length functions must be implemented for all three types: `length(dna)`, `length(kmer)`, and `length(qkmer)`. These return the number of nucleotides in the sequence.
- The `generate_kmers(sequence dna, k integer)` is a set-returning function that produces all possible k-mers from a given DNA sequence. Since it returns a set of values, it must be properly aliased when used in a `FROM` clause. For example:

```
-- Correct usage with column alias
SELECT k.kmer
FROM generate_kmers('ACGTACGT', 6) AS k(kmer);

-- Example output:
--    kmer
--  --------
--   ACGTAC
--   CGTACG
--   GTACGT
```

For more detail on how to implement set-returning functions, see the PostgreSQL documentation.

- The `equals(k kmer, k kmer)` function tests whether two kmers are equal. This function should also be available through the `=` operator for more natural syntax in queries. For example:

```
-- These two queries are equivalent:
SELECT * FROM kmers WHERE equals('ACGTA', kmer);
SELECT * FROM kmers WHERE kmer = 'ACGTA';
```

- The `starts_with(k kmer, k kmer)` function tests whether a kmer starts with a given prefix. This function should also be available through the `^@` operator for more natural syntax in queries. For example:

```
-- These two queries are equivalent:
SELECT * FROM kmers WHERE starts_with('ACG', kmer);
SELECT * FROM kmers WHERE kmer ^@ 'ACG';
```

The `starts_with` function should fail if the prefix length is greater than the kmer length.

- The `contains(pattern qkmer, k kmer)` function tests whether a kmer matches a given pattern. This function should also be available through the `@>` operator for more natural syntax in queries. For example:

```
-- These two queries are equivalent:
SELECT * FROM kmers WHERE contains('ANGTA', kmer);
SELECT * FROM kmers WHERE 'ANGTA' @> kmer;
```

The `contains` function should fail if the pattern and kmer lengths do not match.

K-mer Counting Support

The extension must efficiently support k-mer counting operations. For this to work, the kmer type needs to support grouping operations (`GROUP BY kmer`). Consider what this means in terms of PostgreSQL's type system and operator requirements. A typical k-mer counting query looks like:

```
-- Count all 5-mers in a DNA sequence
SELECT k.kmer, count(*)
FROM generate_kmers(dna, 5) AS k(kmer)
GROUP BY k.kmer
ORDER BY count(*) DESC;

-- Return the total, distinct and unique count of 5-mers in a DNA sequence
WITH kmers AS (
    SELECT k.kmer, count(*)
    FROM generate_kmers(dna, 5) AS k(kmer)
    GROUP BY k.kmer
)
SELECT sum(count) AS total_count,
       count(*) AS distinct_count,
       count(*) FILTER (WHERE count = 1) AS unique_count
FROM kmers;
```

Index Structure

The extension must implement a trie-based index using SP-GiST (Space-Partitioned Generalized Search Tree) for efficient k-mer queries. SP-GiST provides a framework for implementing tree-based indexes where the tree structure and search rules can be customized.

Read the SP-GiST documentation to understand:

1. The basic concepts of SP-GiST indexes
2. The required interface functions you need to implement
3. How to register your index with PostgreSQL

Additionaly, you can study the text type implementation in ***src/backend/access/spgist/spgtextproc.c*** as a reference.

The index should support three types of queries:

1. Equality searches (`WHERE kmer = 'ACGTA'`)
2. Prefix searches (`WHERE kmer ^@ 'ACG'`)
3. Pattern matching using qkmer (`WHERE 'ANGTA' @> kmer`)

Datasets

You should test your implementation with both synthetic and real-world data. For development and initial testing, you can generate random DNA sequences of various lengths. Additionally, you can test your implementation using real genomic data from public databases such as the NCBI Sequence Read Archive (SRA). This will ensure your extension can handle the scale and complexity of actual genomic data analysis tasks.

Submission

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

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

Evaluation (40 points total)

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

- Data Types (10 points)
- Functions and Operators (10 points)
- K-mer Counting Support (10 points)
- SP-GiST Index Implementation (10 points)