INZENT

IT Player, it's INZENT

PostgreSQL Advanced

PostgreSQL Advanced

2024.07

데이터플랫폼개발팀
신주한

Contents

01

FULL TEXT SEARCH

PostgreSQL Advanced

01

FULL TEXT SEARCH

1.1. Full Text Search 정의

INZENT

▶ Full text search

- 텍스트 데이터를 빠르고 효율적으로 검색하기 위한 기능
- 쿼리 용어 조건을 만족하는 문서(Document)를 식별하는 기능을 제공
- 쿼리 용어를 포함한 모든 문서들 찾고 이를 유사도(Similarity) 순위로 정렬해주는 기능을 제공

▶ LIKE, ILIKE, ~, ~* 의 한계

- 언어지원이 없음
  - 단수/복수, 시제 등의 처리가 영어조차 지원되지 않음
- 정렬(Ordering) 기능이 없음
  - 검색 결과의 순위에 의한 정렬 기능을 제공하지 않으므로 수많은 문서가 발견될 경우 효과가 떨어짐
- 속도가 느림
  - 모든 문서마다 문서 전체를 비교 검색하기 때문에 성능이 느린 경향이 있음

1.2. 전처리 작업(Preprocessing)

INZENT

▶ 전처리 작업(Preprocessing)

- 모든 문서를 빠르게 검색할 수 있도록 전처리를 거쳐 인덱스로 만들어 놓을 수 있음
- 전처리 단계는 '토큰화 > 정규화 > 저장' 순서로 진행됨

▶ 토큰화(Tokenizing)

- 문서를 토큰 단위로 나눔
- 공백 등을 제거하고, 숫자, 단어, 복합 단어 등을 구분하여 각각을 처리할 수 있도록 분리함

▶ 어휘소(Lexeme)로 정규화(Normalization)

- 토큰을 어휘소로 변환
- 어휘소는 동일한 단어의 다양한 형태가 하나의 단어로 만들어진 정규화 단어
- 관사와 같은 불용어(Stop word, 검색에 불필요한 흔한 단어)는 제거함

▶ 전처리 결과를 저장

- 각 문서는 정규화된 어휘소의 정렬된 배열로 표현될 수 있음
- 어휘소와 함께 위치 정보를 저장하여 근접 순위 평가에 사용할 수 있음
- 쿼리 단어가 더 밀집된 영역을 포함하는 문서에 더 높은 순위를 부여함으로써 이루어짐

▶ Example

'The cute dogs are running.'
> 'The' 'cute' 'dogs' 'are' 'running'

▶ Example

'The' 'cute' 'dogs' 'are' 'running'
> 'cute' 'dog' 'run'

▶ Example

'cute':2 'dog':3 'run':5

1.3. 어휘소 사전(Lexeme Dictionary)

INZENT

▶ 어휘소 사전(Lexeme dictionary)

- 사전을 통해 토큰을 정규화하는 방법을 세밀하게 제어할 수 있음
- 사전을 필요에 맞게 수정하여 아래와 같은 전처리 작업을 제어할 수 있음

▶ 불용어(Stopword) 정의

- 검색에 사용하지 않을 불용어를 정의

▶ Example

'the', 'is', 'at', 'which'

▶ 동의어 매핑

- 동의어를 단일 단어로 매핑

▶ Example

'fast', 'quick' > 'fast'

▶ 유의어 매핑

- 유의어를 단일 단어로 매핑

▶ Example

'heart attack'
> 'myocardial infarction'

▶ 단어 변형 매핑

- 단어의 다양한 변형을 하나의 표준 형태로 매핑

▶ Example

'run', 'runs', 'running'
> 'run'
'connect', 'connected', 'connecting'
> 'connect'

1.4. tsvector 타입

INZENT

▶ tsvector 타입

- 문서를 텍스트 검색에 최적화된 형태를 저장
- 고유한 어휘소들의 정렬된 형태
- 문서를 인덱싱하기 위해 사용되며, 단어의 위치와 빈도를 저장하여 빠른 검색 및 검색 결과에 대한 순위를 매길 수 있게함

▶ to_tsvector 함수

- 문서를 tsvector 형태로 변환시켜주는 함수
- 전처리에 사용할 언어를 정의할 수 있음
  - 기본 값은 'english'

▶ Example

```
SELECT to_tsvector('The cute dogs are running.');
-- 결과 : 'cute':2 'dog':3 'run':5
SELECT to_tsvector('english', 'The cute dogs are running.');
-- 결과 : 'cute':2 'dog':3 'run':5

SELECT to_tsvector('I cani carini stanno correndo');
-- 결과 : 'cani':2 'carini':3 'correndo':5 'stanno':
SELECT to_tsvector('italian', 'I cani carini stanno correndo');
-- 결과 : 'can':2 'carin':3 'corr':5
```

1.4. tsvector 타입

INZENT

▶ tsvector 형변환과 to_tsvector 함수의 차이

- tsvector는 텍스트를 저장하거나 형변환을 할 때, 이미 모든 전처리가 끝나있을 것을 전제로 함
- to_tsvector 함수는 텍스트를 tsvector 타입으로 변환하는 과정에서 전처리를 수행함

▶ Example

```
SELECT 'The cute dogs are running.'::tsvector;
-- 결과 : 'The' 'are' 'cute' 'dogs' 'running.'

SELECT to_tsvector('The cute dogs are running.');
-- 결과 : 'cute':2 'dog':3 'run':5
```

1.4. tsvector 타입

INZENT

▶ setweight 함수

- tsvector의 어휘소에 가중치(weight)를 매길 수 있음
- 가중치는 'A', 'B', 'C', 'D'로 설정할 수 있으며, 'A'가 가장 높은 값이고 기본값은 'D'임
- Lexeme은 가중치를 따로 설정하지 않은 경우 'D'가 기본값으로 주어짐

▶ Example

```
SELECT setweight(to_tsvector('The cute dogs are running.'), 'A');
-- 결과 : 'cute':2A 'dog':3A 'run':5A

SELECT setweight(to_tsvector('The cute dogs are running.'), 'A') ||
setweight(to_tsvector('The old dogs are sleeping.'), 'B');
-- 결과 :  'cute':2A 'dog':3A,8B 'old':7B 'run':5A 'sleep':10B
```

1.5. tsquery 타입

INZENT

▶ tsquery 타입

- 텍스트 검색에 사용할 어휘소와 연산자의 조합
- tsvector에 저장된 데이터를 검색하는데 사용

▶ tsquery 연산자

- Boolean 연산자
  - & : AND
  - | : OR
  - ! : NOT
- Phrase search 연산자
  - <-> : Followed by
  - <N> : Followed by (N : the distance between the two lexems)
- 괄호
  - 괄호를 이용해 연산자들을 그룹화하여 연산 순서를 정할 수 있음

▶ Example

```
SELECT 'dog & run'::tsquery;
SELECT 'dog | cat'::tsquery;
SELECT '(dog | cat) & !rat'::tsquery;
SELECT 'dog <-> run'::tsquery;
SELECT 'dog <2> run'::tsquery;
```

1.5. tsquery 타입

INZENT

▶ 가중치(Weight)

- tsvector의 어휘소에 매겨진 가중치까지 비교해서 검색을 할 수 있음
- 검색 용어를 포함한 tsvector가 있더라도 가중치가 다른 경우 검색 결과에서 배제됨

▶ Example

```
SELECT 'dog:A & run:B'::tsquery;
SELECT setweight(to_tsvector('dog'), 'A') @@ to_tsquery('quick:B');
-- 결과 : false
```

1.5. tsquery 타입

INZENT

▶ 접두사 매칭(Prefix matching)

- *를 단어 뒤에 라벨링하면 해당 단어로 시작하는 모든 단어를 포함한 tsvector를 매칭함

▶ Example

```
SELECT 'do:*'::tsquery;

SELECT to_tsvector('dog') @@ to_tsquery('do:*');
-- 결과 : false
```

1.5. tsquery 타입

INZENT

▶ plainto_tsquery 함수

- 입력된 텍스트를 어휘소로 분해하여 tsquery 형식으로 변환
- 텍스트는 논리 연산자 없이 AND 조건으로 연결된 어휘소로 변환

▶ Example

```
SELECT plainto_tsquery('The cute dogs are running.');
SELECT plainto_tsquery('english', 'The cute dogs are running.');
-- 결과 : 'cute' & 'dog' & 'run'
```

1.5. tsquery 타입

INZENT

▶ to_tsquery 함수

- tsquery 연산자와 단어들로 이루어진 텍스트를 토큰화하고, 논리 연산자를 포함한 tsquery 형식으로 변환
- 논리 연산자(&, |, !)에 대한 처리를 지원
- 텍스트는 전처리되며, 불용어 제거 및 어간 추출이 수행

▶ Example

```
SELECT to_tsquery('The cute dogs are running.');
-- 에러

SELECT to_tsquery('cute | dogs & running');
SELECT to_tsquery('The & cute | dogs & are & running.');
SELECT to_tsquery('english', 'The & cute | dogs & are & running.');
-- 결과 : 'cute' | 'dog' & 'run'
```

1.5. tsquery 타입

INZENT

▶ phraseto_tsquery 함수

- & 연산자 대신 전처리가 완료된 단어들 간의 거리를 고려하여 <-> 연산자를 추가

▶ Example

```
SELECT phraseto_tsquery('The cute dogs are running.');
SELECT phraseto_tsquery('english', 'The cute dogs are running.');
-- 결과 : 'cute' <-> 'dog' <2> 'run'
```

1.6. 매칭 연산자

▶ 매칭 연산자(Matching operator)

- tsquery를 만족하는 tsvector를 찾으면 true를 반환하는 연산자
- @@ 기호를 사용

▶ Example

```
SELECT to_tsvector('The cute dogs are running.') @@ to_tsquery('cute &
running');
-- 결과 : true

SELECT to_tsvector('The cute dogs are running.') @@ to_tsquery('cute <->
running');
-- 결과 : false

SELECT to_tsvector('The cute dogs are running.') @@ to_tsquery('cute <3>
running');
-- 결과 : true
```

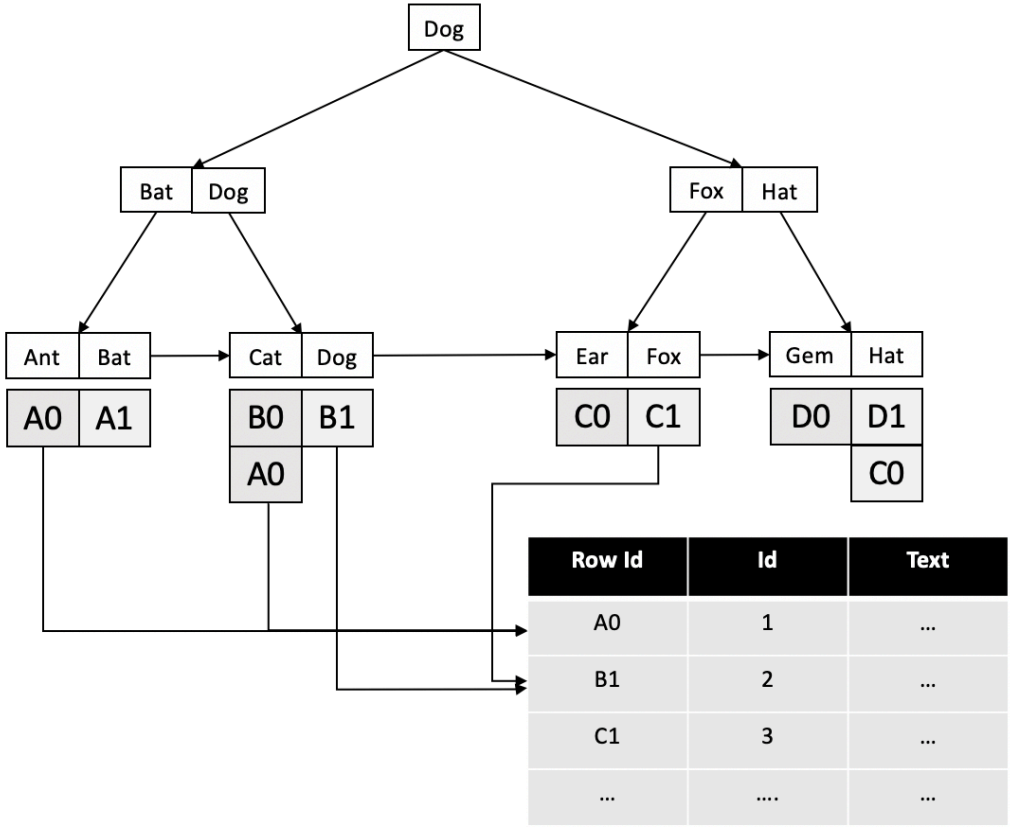
1.7. GIN 인덱스

▶ GIN(Generalized inverted index) 인덱스

- 빠른 텍스트 검색에 적합하게 설계된 인덱스
- 트리의 노드마다 tsvector의 lexeme들을 키로, 해당 단어를 포함시킨 문서들의 ID를 리스트/트리 형태로 저장하고 있으므로 단어의 존재 여부를 신속하게 확인 가능

▶ Example

```
CREATE TABLE employee (
                    id INT PRIMARY KEY,
                    name VARCHAR(100),
                    introduction TEXT,
                    ts_intro tsvector
);
CREATE INDEX idx_ts_intro ON employee USING GIN(ts_intro);
```



1.8. ts_rank 함수

INZENT

▶ ts_rank 함수

- tsvector와 tsquery를 기반으로 문서의 관련성을 계산하여 랭크를 매김
- 각 단어의 위치와 빈도를 고려하여 랭크를 계산
- 높은 점수일수록 검색어와의 관련성이 높음

▶ Example

```
SELECT id, name, ts_rank(ts_intro, to_tsquery('engineer & data')) AS rank
FROM employee
WHERE ts_intro @@ to_tsquery('engineer & data')
ORDER BY rank DESC;
```

1.9. Practice

▶ **자기소개에 study와 hard를 포함한 직원을 조회하는 SQL을 작성하세요. (ts_query)**

▶ **자기소개에 study 뒤어 hard가 바로 따라오는 직원을 조회하는 SQL을 작성하세요. (ts_query)**

▶ **study와 hard가 포함된 자기소개의 관련성을 기준으로 정렬하는 SQL을 작성하세요. (ts_query, ts_rank)**

▶ **study 또는 data가 포함된 자기소개의 관련성을 기준으로 정렬하는 SQL을 작성하세요. (ts_query, ts_rank)**

Contact Us

(주)인젠트
서울 영등포구 국제금융로2길 36, 유화증권빌딩 8층, 9층

Tel : 070.8209.6189 | Fax : 02.787.3699
info@inzent.com

www.inzent.com