PostgreSQL에서의 columnar 지원에 관한 수다

이지호

Columnar DB 너는 누구니?

Columnar 너는 누구니?

Columnar 스토어는 데이터를 전통적인 행 기반(row-oriented) 방식 대신 열(column) 단위로 저장하는 데이터베이스 시스템입니다.

특징:

데이터를 각 열별로 연속적으로 저장

같은 데이터 타입의 값들이 함께 저장되어 압축률 향상

특정 열에 대한 연산을 수행할 때 튜플을 가지고와서 열을 읽는게 아니라 해당 열의 데이터만 읽음

Columnar?

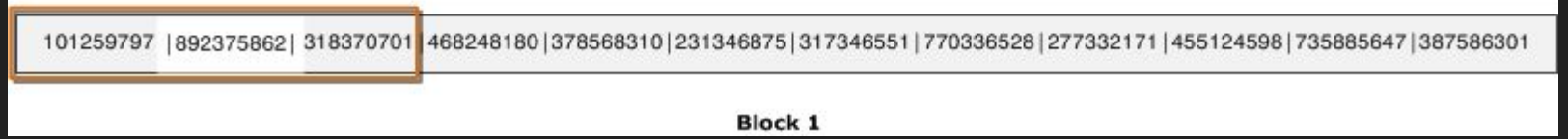
RDB에서의 데이터 보관

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

Columnar?

Columnar 에서의 데이터 보관

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



Block 1

Columnar 장점

a) 쿼리 성능 향상:

분석 쿼리에서 일부 열만 필요할 때 효율적

집계 연산(SUM, AVG, COUNT 등)이 빠름

b) 데이터 압축:

같은 데이터 타입이 연속적으로 저장되어 압축 효율이 높음

c) 디스크 I/O 감소:

필요한 열만 읽어 불필요한 데이터 접근 최소화

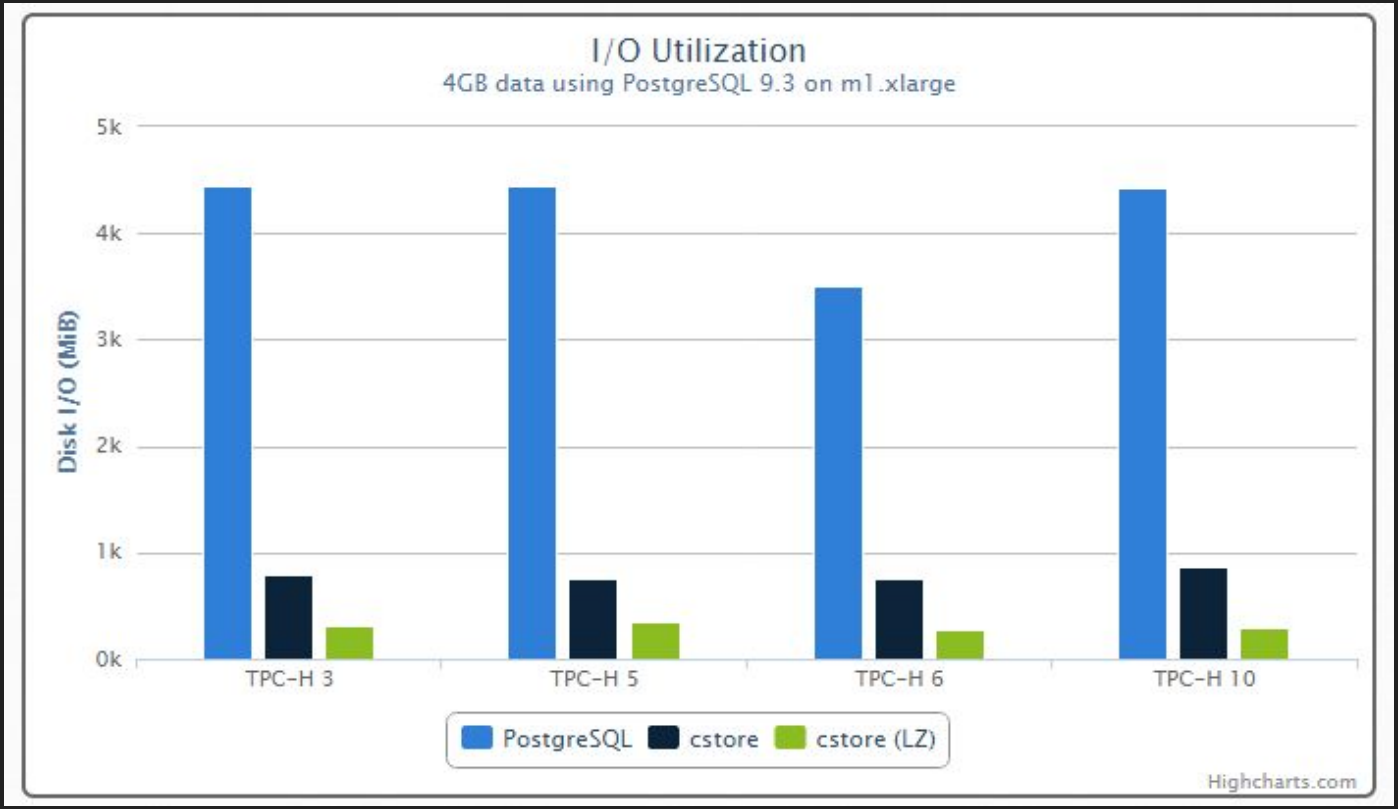
Columnar 단점

a) 데이터 입력/수정 속도가 상대적으로 느림

b) 전체 레코드를 읽는 작업에서는 성능이 떨어질 수 있음

c) 복잡한 인덱싱 구조가 필요할 수 있음

cStore_fdw과 PostgreSQL 성능 비교



Columnar DB에서는 어떤일이 발생할까요?

벡터화 실행: 데이터를 벡터로 처리하여 CPU 캐시 활용도 증가

늦은 구체화(Late Materialization): 필요한 시점까지 전체 레코드 재구성을 지연

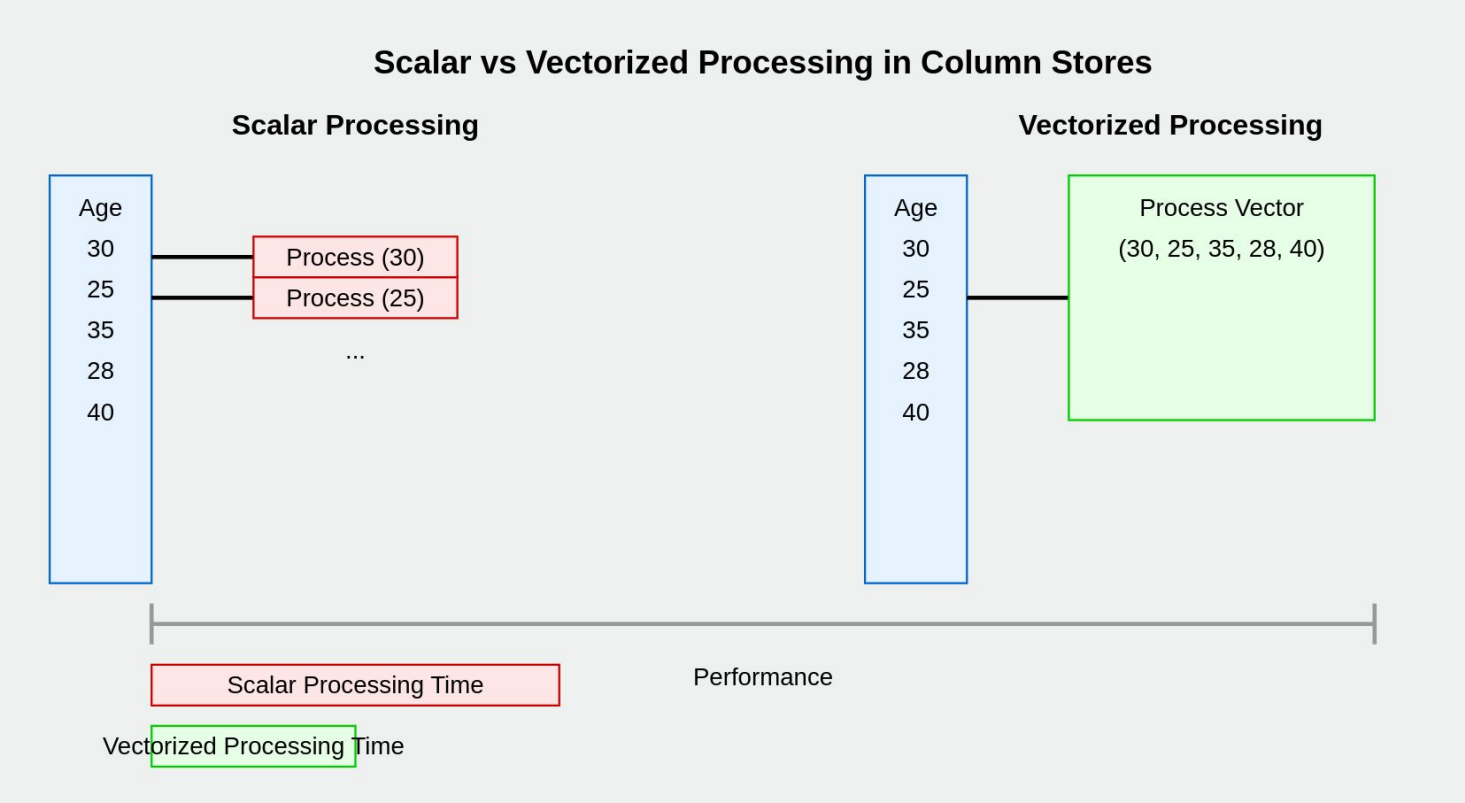
조인 최적화: 열 기반 저장에 최적화된 조인 알고리즘 사용

Columnar DB 기술에 대해 더 알아보기 - 벡터화

벡터화 실행 (Vectorized Execution):

- 개념: 데이터를 작은 벡터(vector)로 처리하여 CPU 캐시 활용도를 높입니다.
- 작동 원리:
  - 데이터를 캐시 라인 크기에 최적화된 벡터로 나눕니다.
  - SIMD(Single Instruction, Multiple Data) 명령어를 사용하여 병렬 처리합니다.
- 이점: CPU 파이프라인 활용도 향상, 브랜치 미스 예측 감소, 전반적인 처리 속도 향상

벡터화

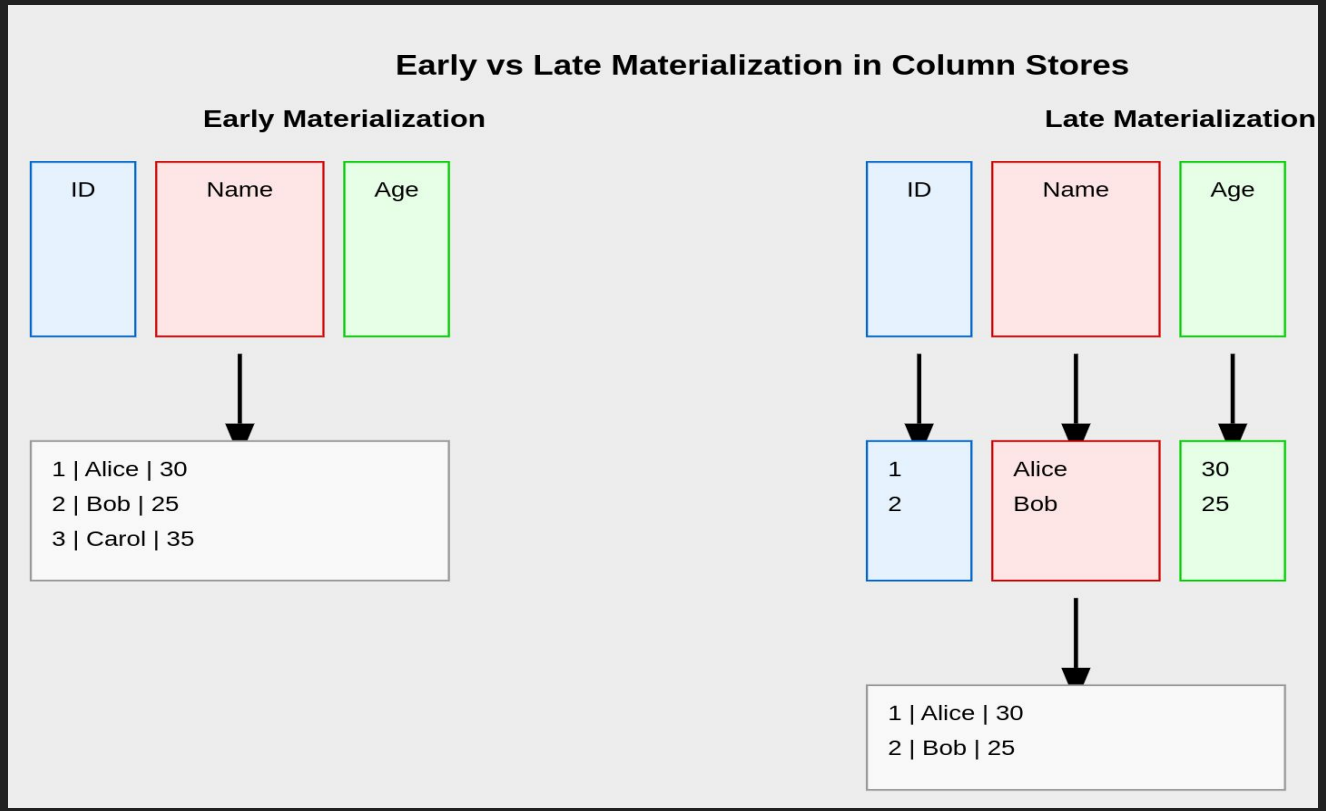

Scalar vs Vectorized Processing in Column Stores
Scalar Processing
Age
30
25
35
28
40
Process (30)
Process (25)
...
Vectorized Processing
Age
30
25
35
28
40
Process Vector
(30, 25, 35, 28, 40)
Performance
Scalar Processing Time
Vectorized Processing Time

Columnar DB 기술에 대해 더 알아보기 - 늦은 구체화

늦은 구체화 (Late Materialization):

- 개념: 필요한 시점까지 전체 레코드의 재구성을 지연시킵니다.
- 작동 원리:
  - 쿼리 처리 중 필요한 열만 메모리에 로드합니다.
  - 최종 결과 생성 직전에 선택된 열들을 조합하여 전체 레코드를 구성합니다.
- 이점: 메모리 사용량 감소, 불필요한 데이터 처리 최소화

늦은 구체화


Early vs Late Materialization in Column Stores
Early Materialization
ID
Name
Age
1 | Alice | 30
2 | Bob | 25
3 | Carol | 35
Late Materialization
ID
Name
Age
1
2
Alice
Bob
30
25
1 | Alice | 30
2 | Bob | 25

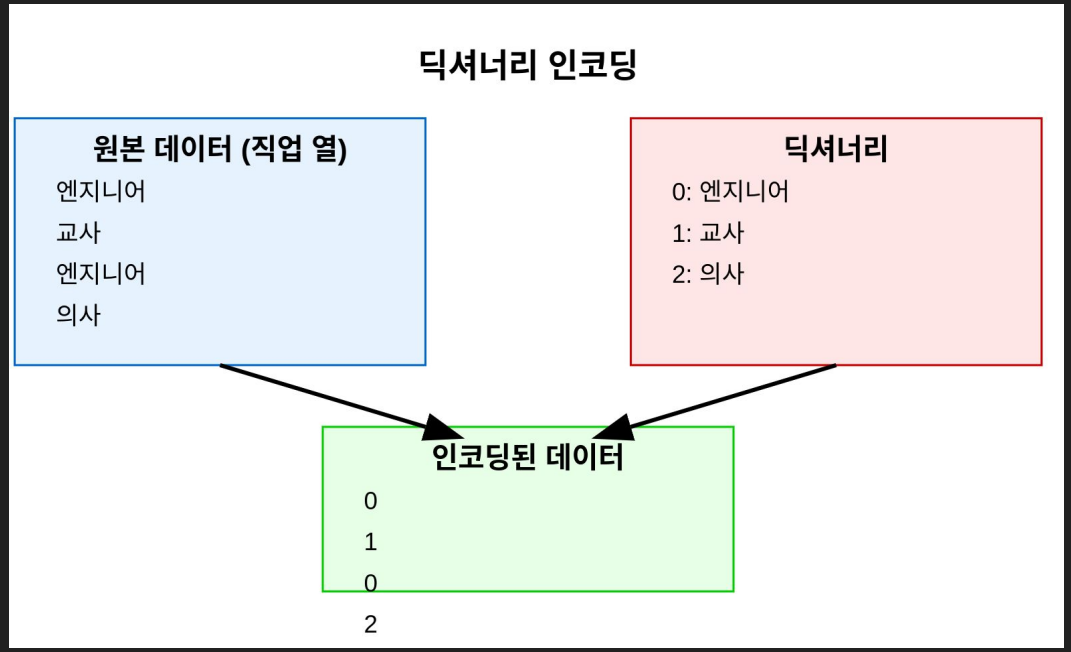
Columnar DB 기술에 대해 더 알아보기 - 압축 기술

- Run-length encoding (RLE): 연속된 같은 값을 값과 반복 횟수로 저장
- Dictionary encoding: 고유한 값에 정수 ID를 할당하여 저장
- Bit-packing: 작은 정수값을 비트 단위로 압축
- Delta encoding: 연속된 값의 차이만 저장

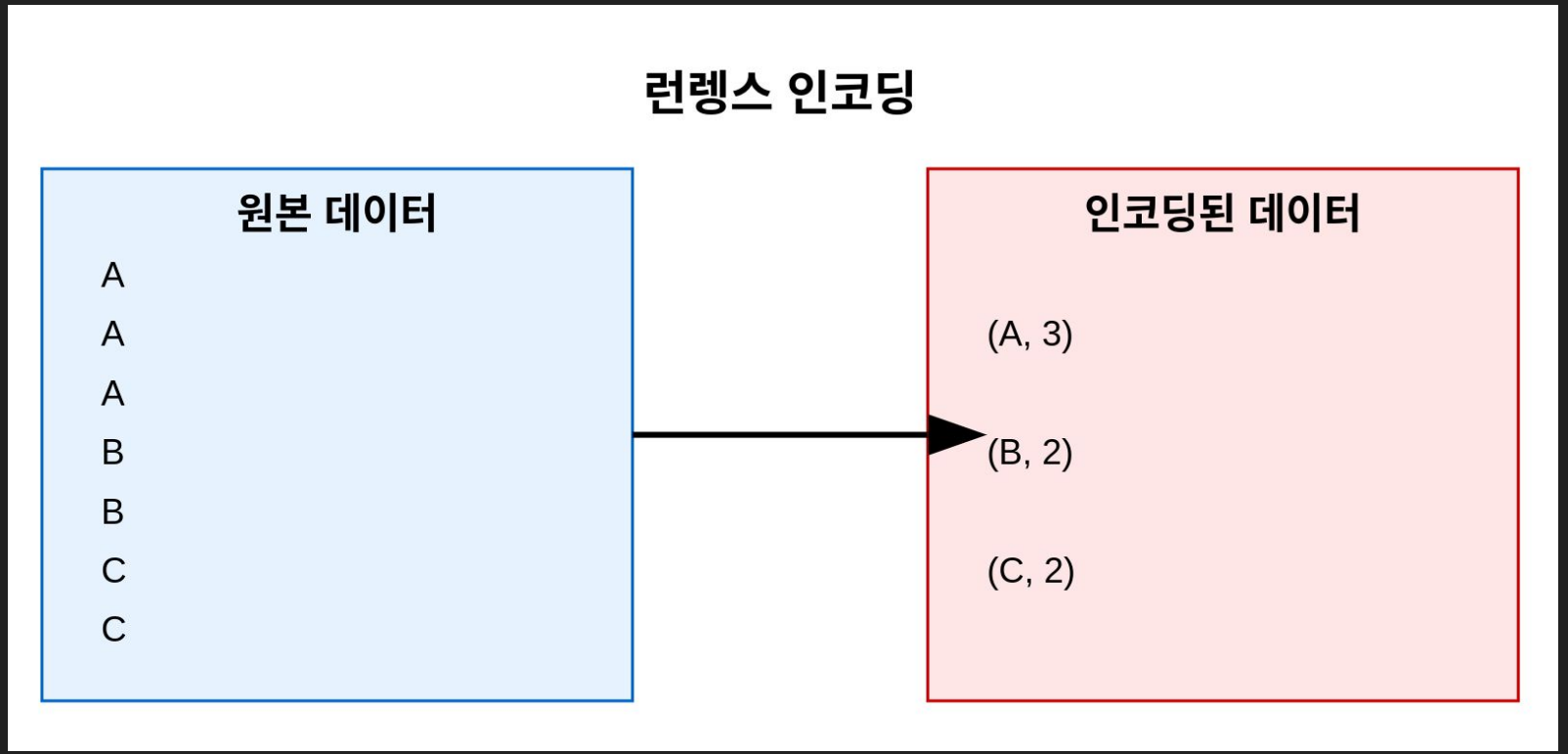
이점: 저장 공간 절약, I/O 감소, 쿼리 성능 향상

Dictionary encoding

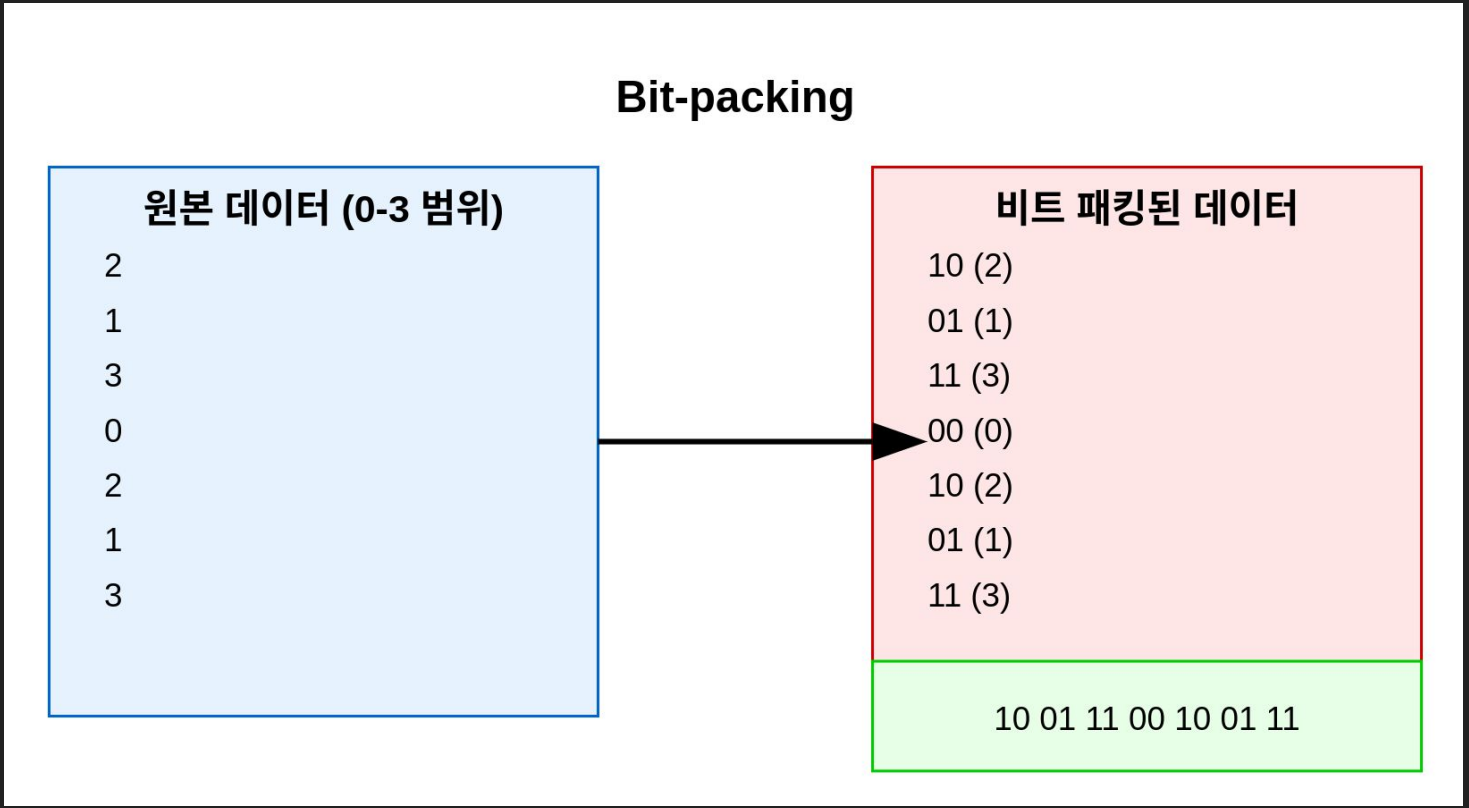
딕셔너리 인코딩은 고유한 값들을 정수 ID에 매핑하는 압축 기법입니다. 특히 카디널리티가 낮은(고유 값의 수가 적은) 열에 효과적입니다.



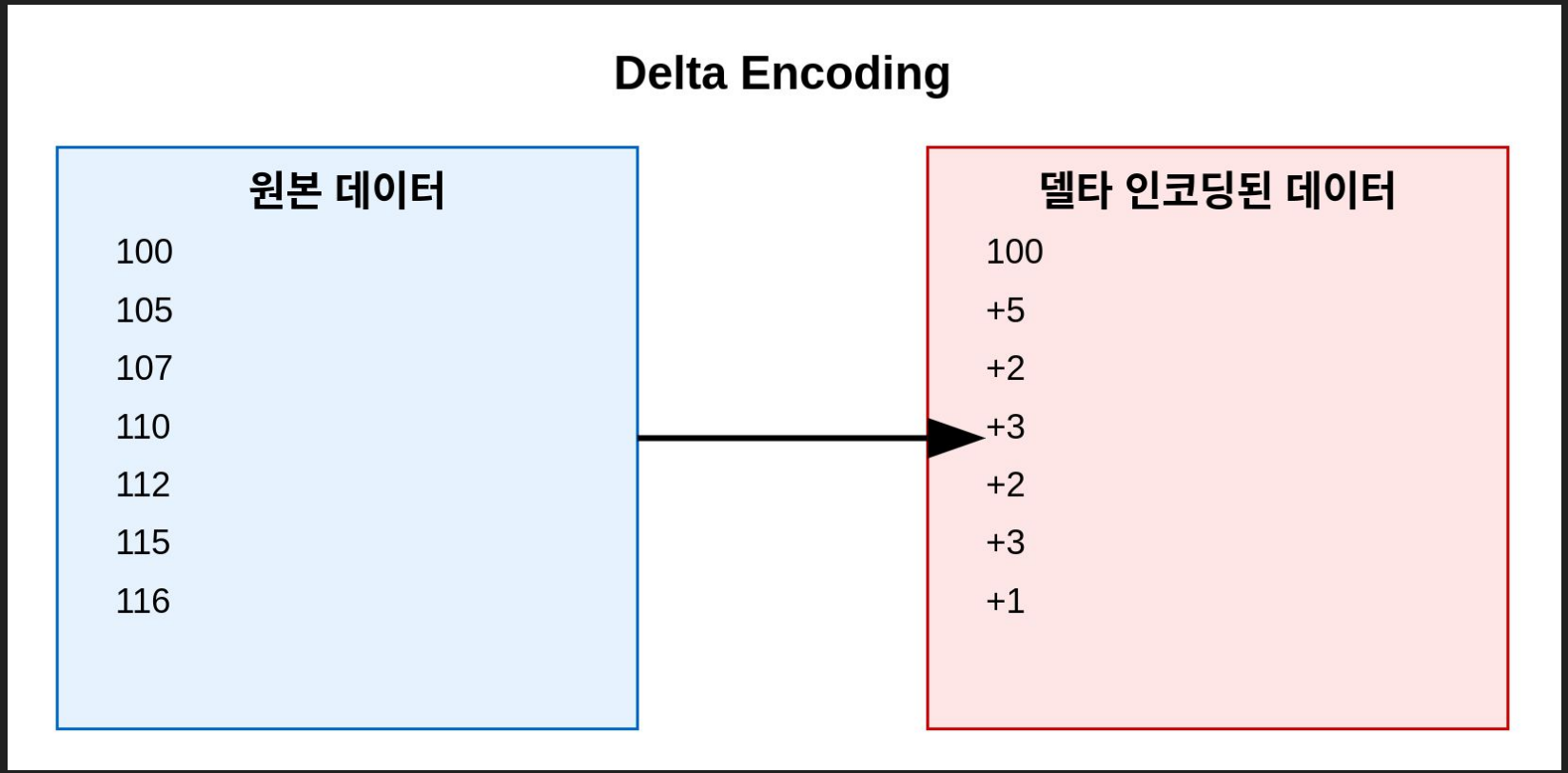
Run-length encoding (RLE)



Bit-packing


Bit-packing
원본 데이터 (0-3 범위)
2
1
3
0
2
1
3
비트 패킹된 데이터
10 (2)
01 (1)
11 (3)
00 (0)
10 (2)
01 (1)
11 (3)
10 01 11 00 10 01 11

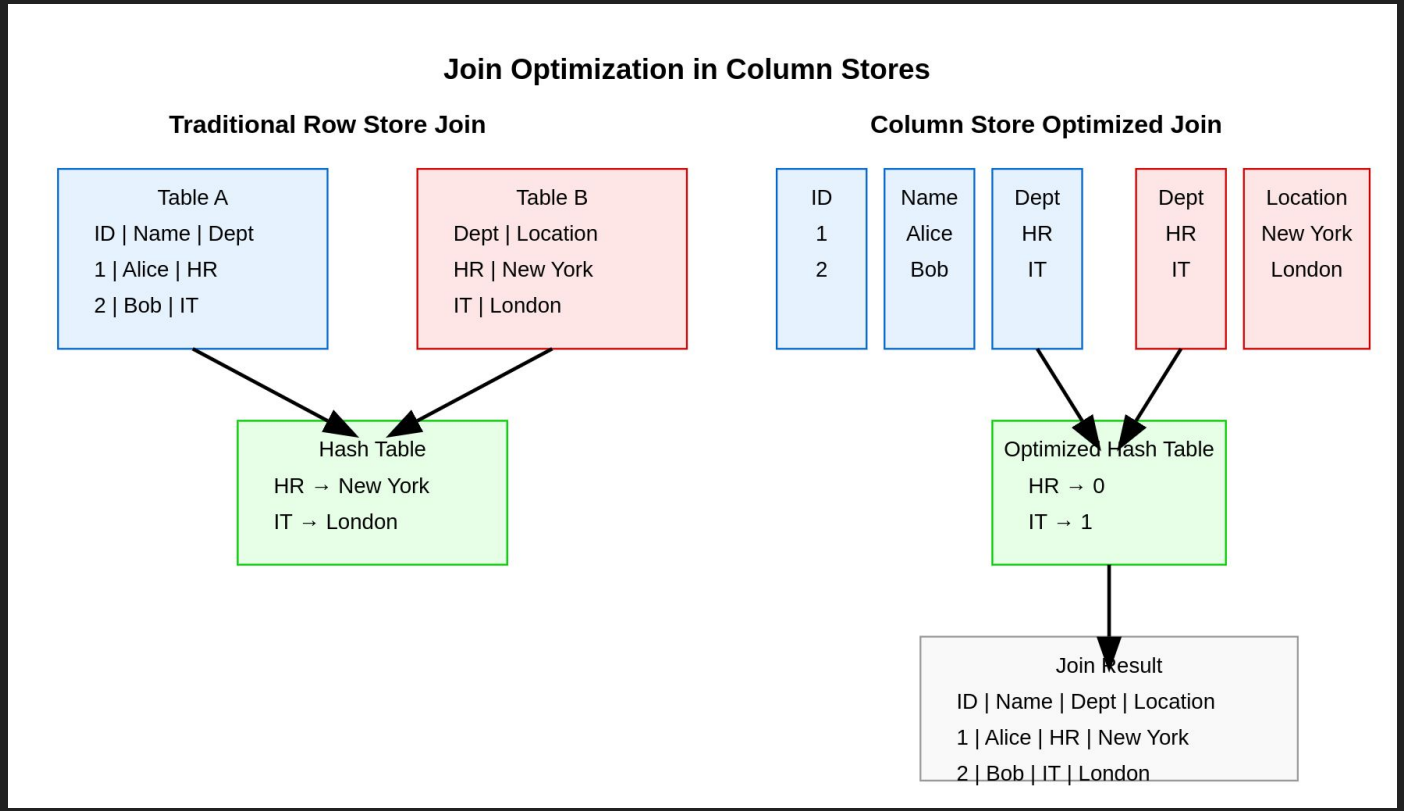
Delta Encoding



Columnar DB 기술에 대해 더 알아보기 - 조인 최적화

- 열 기반 해시 조인: 조인 키 열만 사용하여 해시 테이블 구축
- 벡터화된 조인: 벡터 단위로 조인 연산 수행
- 블룸 필터 사용: 조인 전 불필요한 데이터 필터링

조인 최적화


Join Optimization in Column Stores
Traditional Row Store Join
Table A
ID | Name | Dept
1 | Alice | HR
2 | Bob | IT
Table B
Dept | Location
HR | New York
IT | London
Hash Table
HR → New York
IT → London
Column Store Optimized Join
ID
1
2
Name
Alice
Bob
Dept
HR
IT
Dept
HR
IT
Location
New York
London
Optimized Hash Table
HR → 0
IT → 1
Join Result
ID | Name | Dept | Location
1 | Alice | HR | New York
2 | Bob | IT | London

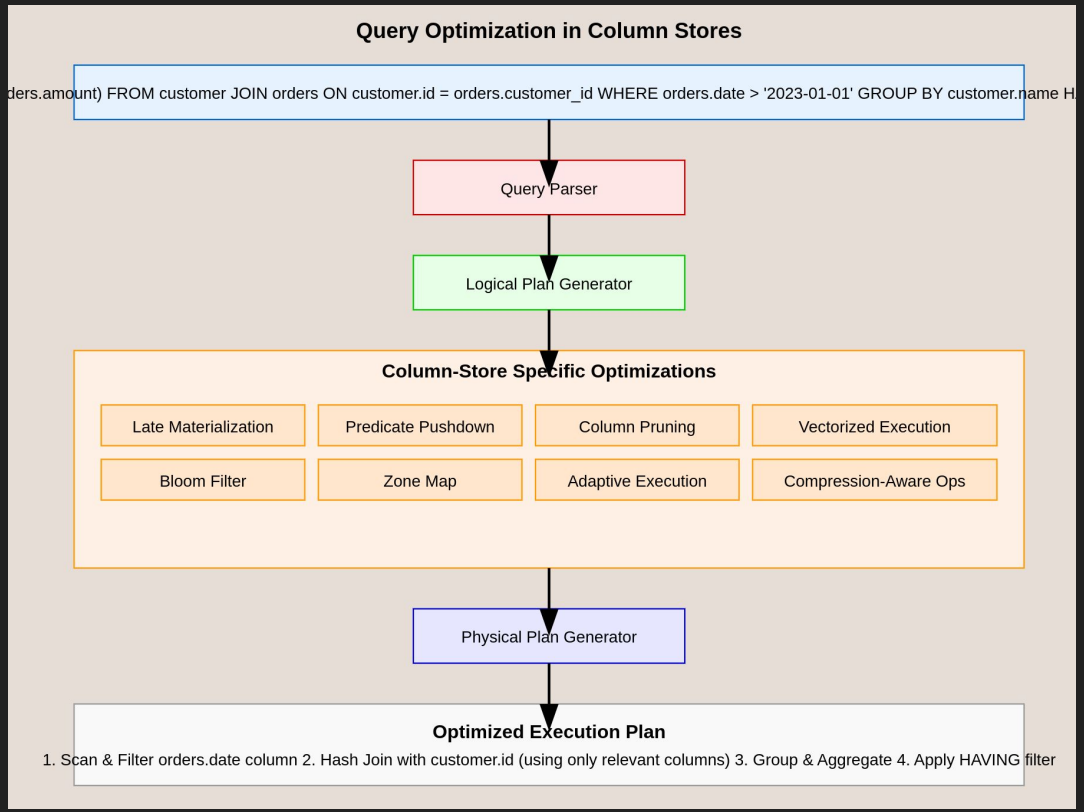
Columnar DB 기술에 대해 더 알아보기 - 쿼리 최적화

열 기반 통계 수집: 각 열의 데이터 분포, 카디널리티 등 정보 활용

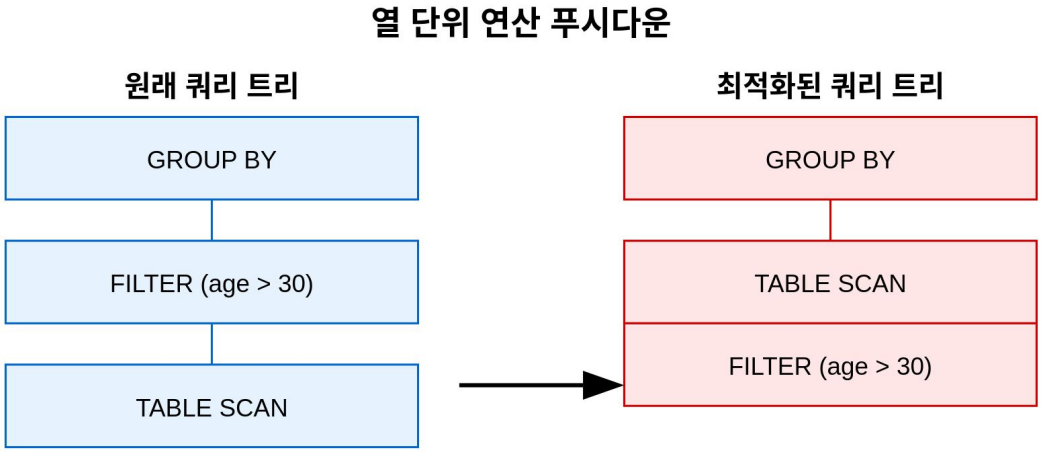
열 단위 연산 푸시다운: 필터링, 집계 등의 연산을 가능한 한 아래로 내림

조건부 실행 계획: 데이터 특성에 따라 동적으로 실행 계획 변경

Columnar DB에서의 쿼리 최적화



Columnar DB에서 열단위 연산의 Pushdown



푸시다운의 이점:

1. 쿼리 실행 초기 단계에서 데이터 양을 줄임
2. 필터링에 컬럼 스토어의 효율성을 활용
3. 존 맵이나 다른 컬럼 수준 메타데이터를 활용 가능
4. 전반적인 쿼리 성능 향상

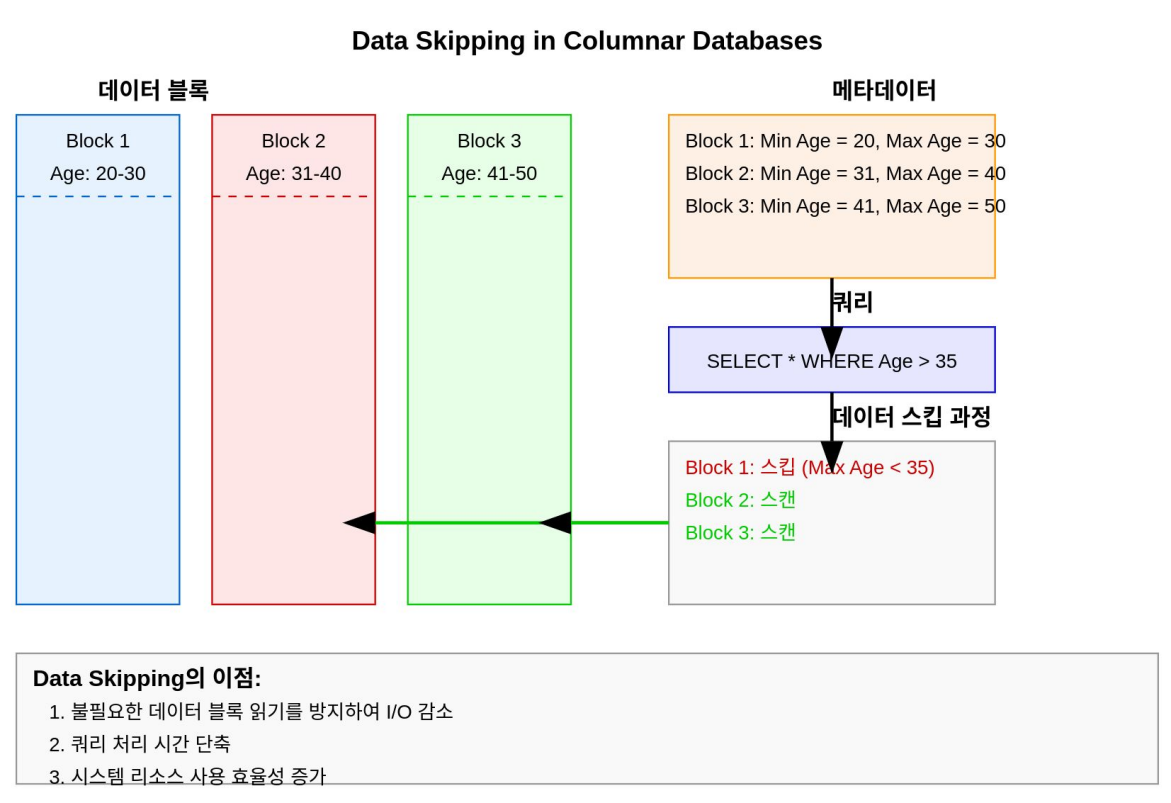
Columnar DB 기술에 대해 더 알아보기 - Data Skipping

개념: 쿼리와 관련 없는 데이터 블록을 빠르게 건너뜁니다.

구현: 메타데이터에 각 데이터 블록의 통계 정보(min, max 등) 저장

이점: 전체 데이터 스캔 없이 필요한 데이터만 빠르게 접근

Data Skipping 시각화



Columnar DB 기술에 대해 더 알아보기 - 분산 처리 최적화

데이터 파티셔닝: 열 값에 기반한 효율적인 데이터 분산

로컬리티 인식 스케줄링: 데이터 위치를 고려한 태스크 할당

병렬 쿼리 실행: 여러 노드에서 동시에 쿼리 처리

Columnar DB 기술에 대해 더 알아보기 - 인메모리 처리

열 데이터를 메모리에 캐시하여 빠른 접근 제공

메모리 내 압축 상태 유지로 효율적인 공간 활용

Columnar DB 기술에 대해 더 알아보기 - 적응형 인덱싱

쿼리 패턴에 따라 동적으로 인덱스 생성 및 관리

자주 접근되는 열에 대해 자동으로 인덱스 구축

그래서 PostgreSQL에서는 어떻다는 거야?

cstore_fdw(Columnar Store Foreign Data Wrapper)

Citus Data에서 개발한 외부 데이터 래퍼(FDW) 확장

컬럼 기반 저장을 제공하여 분석 쿼리 성능 향상

특징:

- 데이터 압축 지원
- 불필요한 열 스캔 방지
- 벡터화된 실행 지원

https://github.com/citusdata/cstore_fdw

그래서 PostgreSQL에서는 어떻다는 거야?

Apache AGE (Automatic Graph Extension):

- 그래프 데이터 모델과 Cypher 쿼리 언어를 PostgreSQL에 통합
- 일부 그래프 데이터를 columnar 형식으로 저장 가능

https://age.apache.org/

그래서 PostgreSQL에서는 어떻다는 거야?

TimescaleDB:

- 시계열 데이터를 위한 확장으로, 일부 columnar 압축 기능 제공
- 최근 버전에서는 시계열 데이터에 대한 자동 압축 및 컬럼 기반 저장 지원

https://github.com/timescale/timescaledb

그래서 PostgreSQL에서는 어떻다는 거야?

PostgreSQL 자체 기능

- BRIN (Block Range INdex) 인덱스: 대용량 테이블에 대해 효율적인 인덱싱을 제공하며, 일부 columnar 저장의 이점 제공
- Table Access Method: PostgreSQL 12부터 도입된 기능으로, 향후 네이티브 columnar 저장 구현의 기반 가능성

그래서 PostgreSQL에서는 어떻다는 거야?

Hydra:

- Hydra 프로젝트는 PostgreSQL에 네이티브 columnar 저장 엔진을 추가하는 것이 목표
- 아직 개발 중인 프로젝트로, 향후 PostgreSQL의 기본 기능이 될 가능성 존재

https://www.hydra.so/

https://github.com/hydradatabase/hydra

여기에서 살펴볼 Columnar 지원

cstore_fdw

```
-- After adding adding shared_preload_libraries = 'citus'
-- to postgresql.conf and restarting:
CREATE EXTENSION IF NOT EXISTS citus;

-- Create a table using the columnar access method, with the same columns
-- as an existing cstore_fdw table
CREATE TABLE customer_reviews_am (
  LIKE customer_reviews_fdw INCLUDING ALL
) USING columnar;

-- Copy data from an old cstore_fdw table to an access method table
INSERT INTO customer_reviews_am SELECT * FROM customer_reviews_fdw;

-- cstore_fdw data size
SELECT pg_size_pretty(cstore_table_size('customer_reviews_fdw'));
```

```
┌─────────────────┐
│ pg_size_pretty  │
├─────────────────┤
│ 100 MB          │
└─────────────────┘
```

```
-- Citus Columnar data size
SELECT pg_size_pretty(pg_table_size('customer_reviews_am'));
```

```
┌─────────────────┐
│ pg_size_pretty  │
├─────────────────┤
│ 64 MB           │
└─────────────────┘
```

Table Access Method 방식

Table Access Method는 PostgreSQL 12에서 도입된 기능으로, 테이블 데이터의 물리적 저장 및 접근 방식을 사용자가 정의할 수 있게 해줍니다.

주요 특징:

- 사용자 정의 저장 방식: 다양한 저장 형식(columnar, compressed 등)을 구현할 수 있습니다.
- 플러그인 아키텍처: 코어 PostgreSQL 코드 수정 없이 새로운 저장 방식을 추가할 수 있습니다.
- 기존 SQL 인터페이스 유지: 새로운 저장 방식을 사용해도 기존 SQL 문은 그대로 사용 가능합니다.

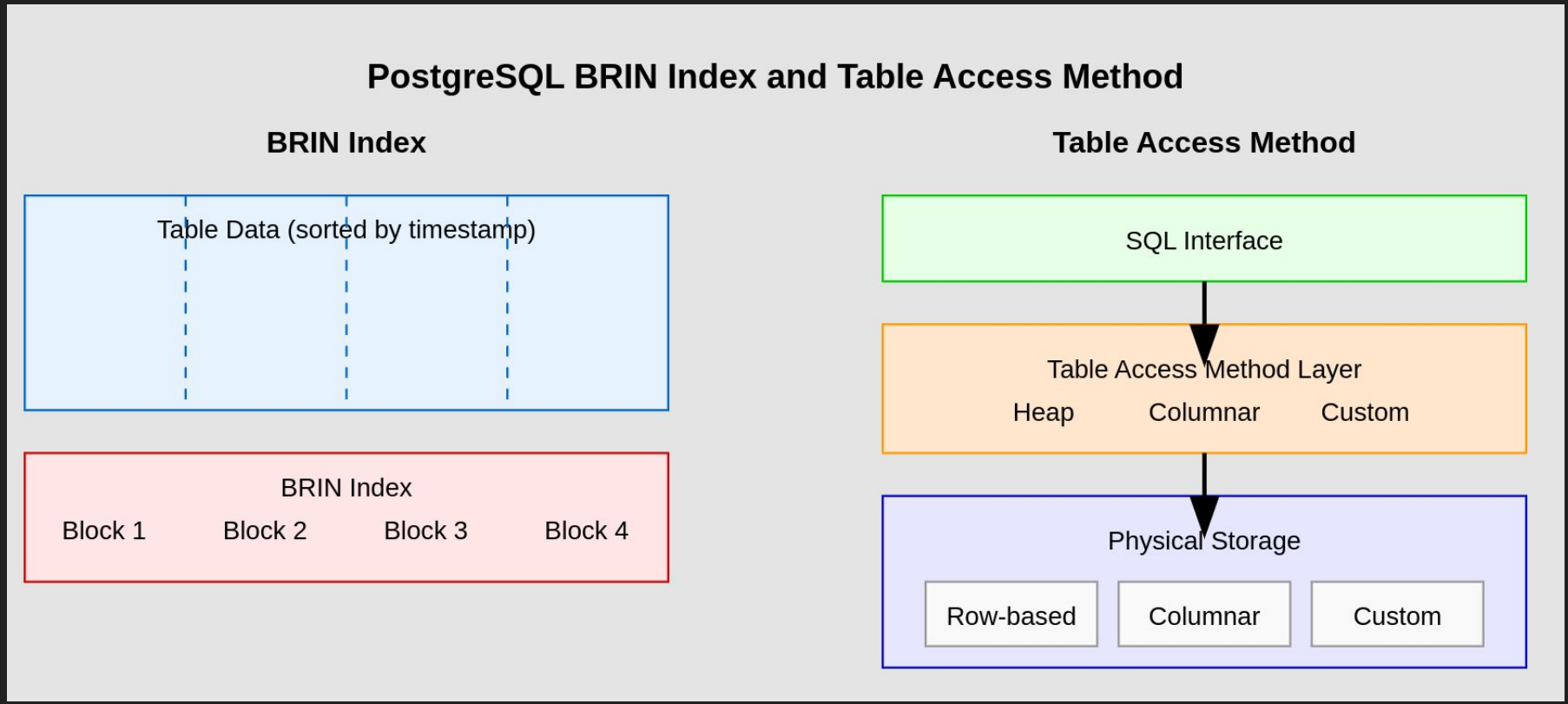
기본 제공 Table Access Methods:

- heap: 기본 row-oriented 저장 방식
- AOCS (Append-Optimized Columnar Storage): 외부 테이블을 위한 columnar 저장 방식

사용 예:

CREATE TABLE columnar_table (id int, data text) USING columnar;

Table Access Method



Hydra

CREATE EXTENSION IF NOT EXISTS columnar;

CREATE TABLE columnar_table (...) USING columnar;

다른건 안 필요해요?

더 알아보는 Columnar DB 이야기

Row-store(전통적인 tuple 기반의 DB) 와 Columnar DB 비교

Row-store: 전체 레코드 읽기/쓰기에 효율적, OLTP 워크로드에 적합

Column-store: 분석 쿼리와 대량 데이터 처리에 효율적, OLAP 워크로드에 적합

Columnar 어디에서 사용하니?

- 데이터 웨어하우스
- 비즈니스 인텔리전스 (BI) 시스템
- 대규모 분석 작업

여기에서 알아본 Columnar 외에 다른 Columnar DB

- Apache Parquet
- Apache Cassandra (부분적으로 columnar)
- Google BigQuery
- Google AlloyDB
- Amazon Redshift
- Vertica

들어주셔서 감사합니다!