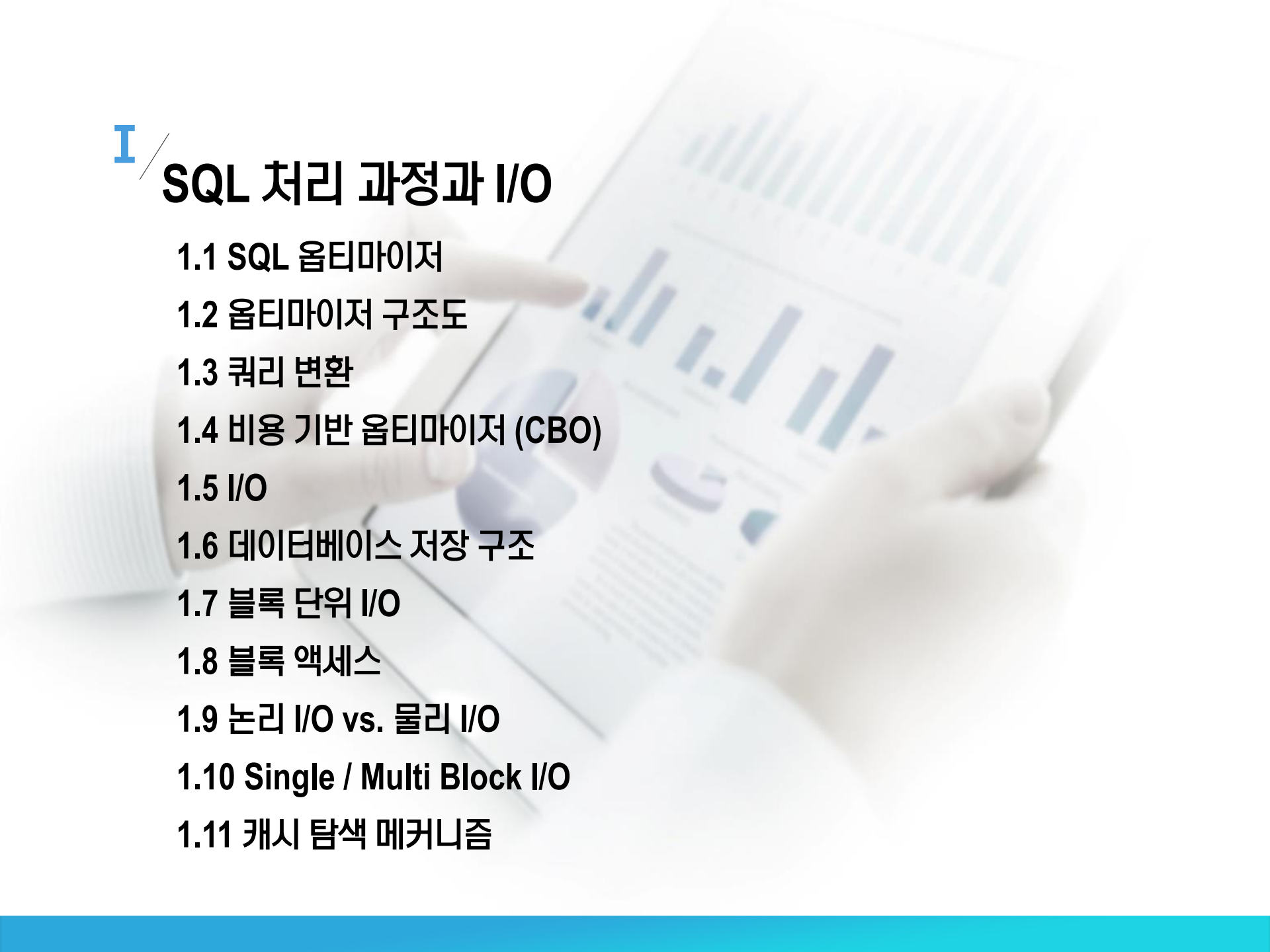
TIBERO 튜닝 기초교육

(1. SQL 처리 과정과 I/O)

YOON



I SQL 처리 과정과 I/O

1.1 SQL 옵티마이저

1.2 옵티마이저 구조도

1.3 쿼리 변환

1.4 비용 기반 옵티마이저 (CBO)

1.5 I/O

1.6 데이터베이스 저장 구조

1.7 블록 단위 I/O

1.8 블록 액세스

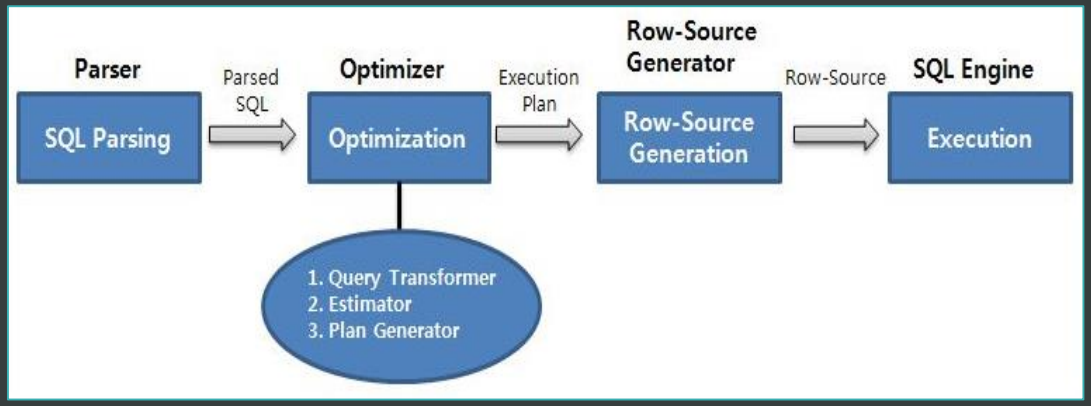
1.9 논리 I/O vs. 물리 I/O

1.10 Single / Multi Block I/O

1.11 캐시 탐색 메커니즘

1.1 SQL 옵티마이저

SQL 처리과정



Row-Source Generator

Row-Source는 레코드 집합을 반복처리 하면서 사용자가 요구한 최종 결과집합을 실제적으로 생성하는데 사용되는 제어구조를 말함.
Generator는 실행 계획을 실행 가능한 코드 또는 프로시저 형태로 포맷하는 작업을 담당

- 사용자가 SQL문을 실행하면 SQL 파싱 단계를 거친 후 해당 SQL이 메모리에 캐싱되어 있는지를 먼저 확인
- 메모리에서 찾으면 곧바로 실행할 수 있지만, 찾지 못했을 때는 최적화 (Optimization) 단계를 거치게 됨
- 파싱 단계에서 SQL 커서를 메모리에서 찾아 곧바로 실행 단계로 넘어가는 것을 소프트 파싱이라고 하고, 찾는데 실패해서 최적화 및 Row-Source 생성 단계를 거치는 것을 하드 파싱이라고 함
- 하드 파싱이 중요한 이유는, 최적화(Optimization) 단계가 그만큼 무거운 처리과정을 거치기 때문임.
  대부분의 SQL 튜닝이 I/O 작업에 집중되는 반면 하드 파싱은 CPU를 많이 소비하는 몇 안되는 작업에 속함

1.2 옵티마이저 구조도

옵티마이저 컴포넌트

1. Query Transformation

   사용자가 던진 SQL을 우선 최적화하기 쉬운 형태로 변환을 시도합니다. 물론 쿼리 변환 전후 결과가 통일함이 보장될 때만 그렇게 함

2. Estimator

   쿼리 오퍼레이션 각 단계의 선택도(Selectivity) 카디널리티 (Cardinality) , 비용(Cost)을 계산하고,
   궁극적으로는 실행계획 전체에 대한 총 비용을 계산함.
   이는 어디까지나 예상치이며, 각 단계를 수행하는데 펼요한 I/0, CPU, 메모리 사용량 등을 예측하기 위해
   데이터베이스 오브젝트(테이블, 인덱스 등) 통계정보와 하드웨어적인 시스템 성능 통계정보도 이용함

3. Plan Generator

   하나의 쿼리를 수행하는 데 있어 후보군이 될만한 실행계획들을 생성해 내는 역할을 함

1.3 쿼리 변환

Query Transformation

리소스를 실제로 사용하지 않고 논리적으로만 판단하기에 Logical Optimizer라고 불리며 2가지 형태로 작동

1. Heuristic Query Transformation : 체험기반의 쿼리 변환
   옵티마이저가 성능을 향상시키기 위해서 비용계산 과정이 필요한 경우도 있지만 특정한 경우에는 비용계산 과정이 필요 없기 때문에 이러한 과정을 생략하고 바로 특정 Rule을 적용하여 최적화를 진행하겠다는 의미

2. Cost Based Query Transformation : 비용기반의 쿼리 변환
   비용기반 쿼리변환은 쿼리 블럭을 최적화 하는 방법을 정할 때 스스로 결정하지 못하고 Cost Estimator에게 어떠한 방법이 제일 좋은 것인지 물어보고 결정하는 방법

1.3 쿼리 변환 (계속)

Query Transformation (예제)

인덱스 구성 - author, title

[Order By 사용안함]

```
select * from book
 where author = 'zrjthoiqpy';
```

10053 Trace 결과

```
**************************
Order-by elimination (OBYE)
**************************
OBYE:     OBYE bypassed: no order by to eliminate.
```

인덱스 구성 - author, title

[Order By 사용]

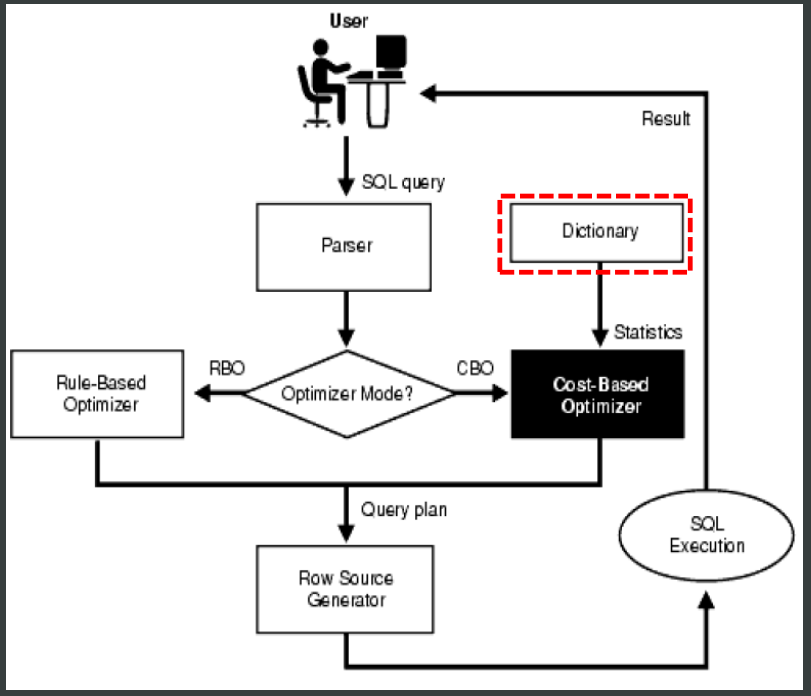
```
select * from book
 where author = 'zrjthoiqpy'
 order by title ;
```

10053 Trace 결과

```
**************************
Order-by elimination (OBYE)
**************************
OBYE:     OBYE performed.
```

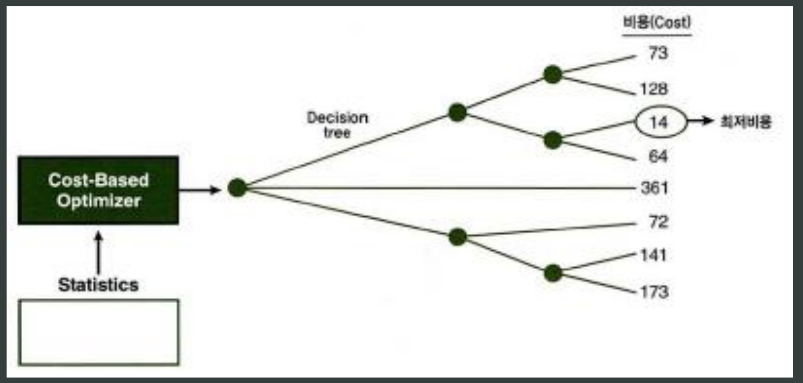
1.4 비용 기반 옵티마이저 (CBO)

방식



- 말 그대로 비용을 기반으로 최적화를 수행하는 것을 말하며 비용(Cost)이란, 쿼리를 수행하는데 소요되는 일량 또는 시간을 뜻함.

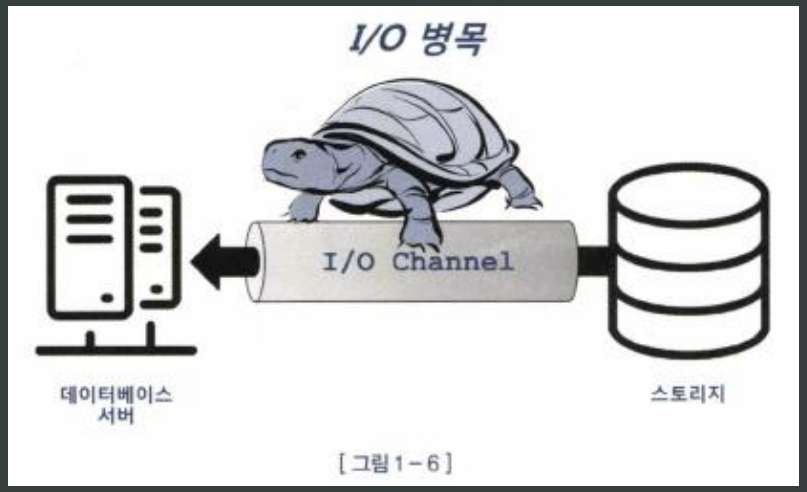
- CBO가 실행계획을 수립할 때 판단 기준이 되는 비용은 어디까지나 예상치이며 미리 구해놓은 테이블과 인텍스에 대한 여러 통계정보를 기초로 각 오퍼레이션 단계별 예상 비용을 산정하고, 이를 합산한 총비용이 가장 낮은 실행계획 하나를 선택함



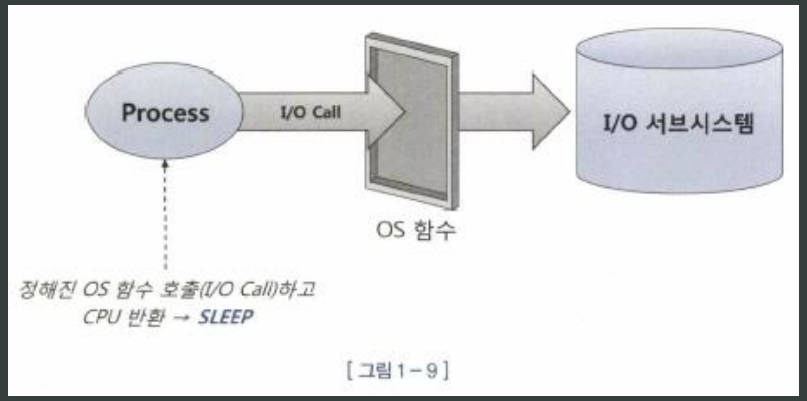
1.5 I/O

SQL이 느린 이유

- SQL 튜닝의 80% 이상은 I/O 튜닝
- SQL의 Slow Performance는 대부분 I/O 때문이며 구체적으로 디스크 I/O
- I/O 채널
  디스크와 DB 서버 간의 한정적인 리소스
- CPU의 대기
  OS 또는 I/O 서브시스템이 I/O를 처리하는 동안 프로세스틀 Sleep 상태에 빠짐



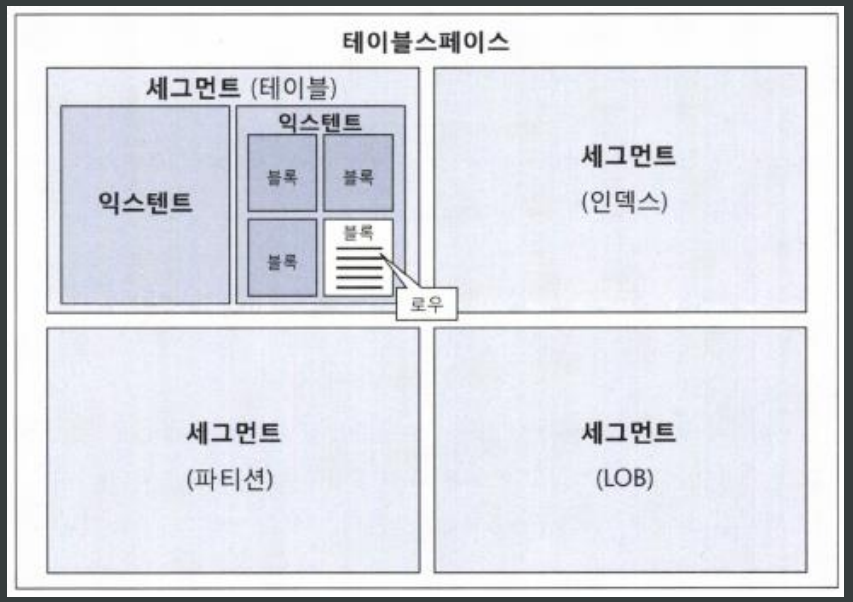
[그림1-6]

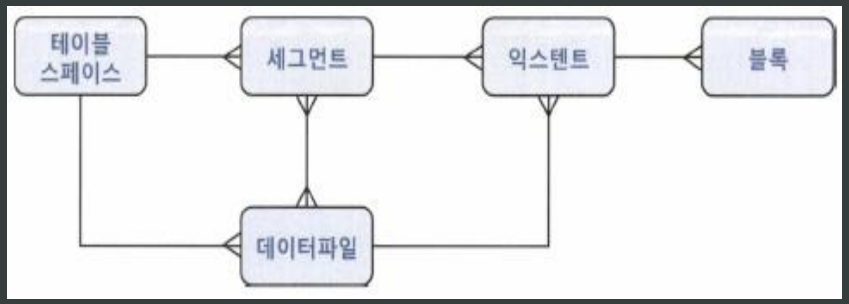


[그림1-9]

1.6 데이터베이스 저장 구조

저장 구조간 관계





용어 정리

- 블록(Block) : 데이터를 읽고 쓰는 I/O 최소 단위
- 익스텐트(Extent) : 공간을 확장하는 단위, 연속된 블록 집합
- 세그먼트(Segment) : 데이터 저장공간이 필요한 오브젝트(테이블, 인덱스, 파티션, LOB, ...)
- 테이블스페이스(Tablespace) : 세그먼트를 담는 콘테이너
- 데이터 파일(Data File) : 디스크 상의 물리적인 OS 파일

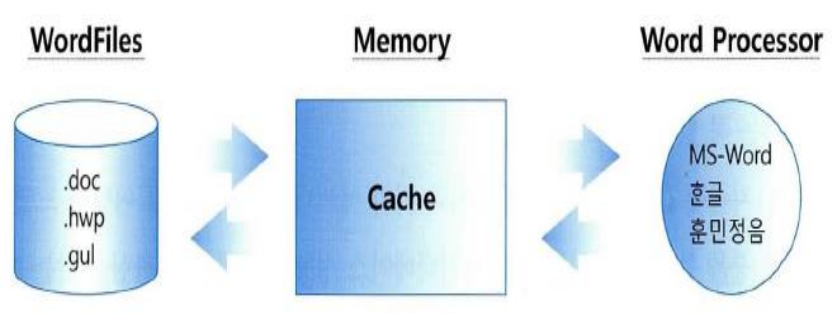
1.6 데이터베이스 저장 구조 (계속)

테이블스페이스

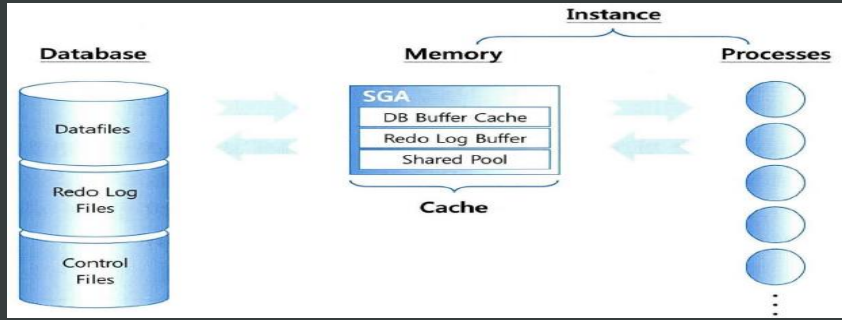
익스텐트 내 블록은 서로 인접한 연속된 공간이지만, 익스텐트끼리는 연속된 공간이 아님

1.7 블록 단위 I/O

파일 단위 I/O



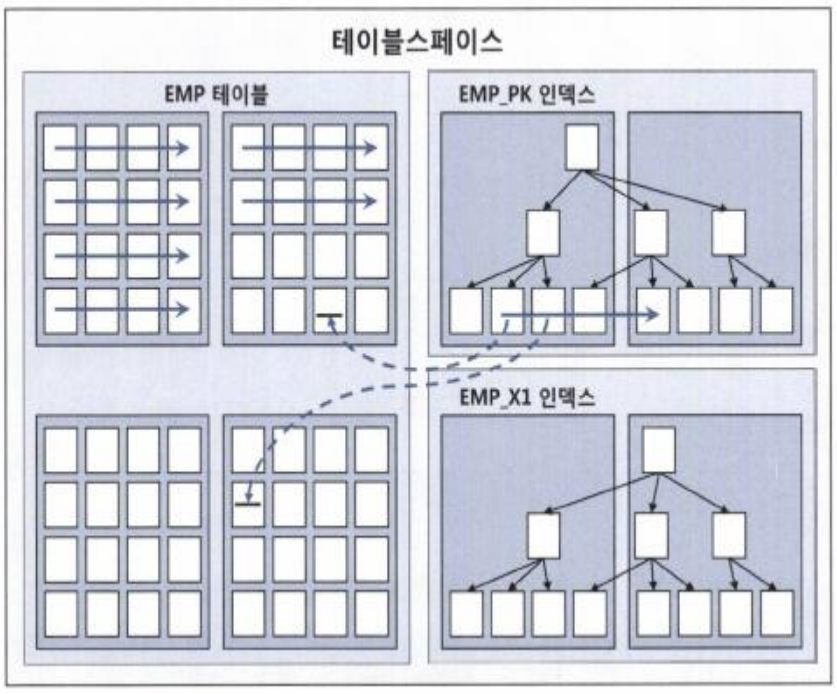
블록 단위 I/O



- 워드프로세서를 통해 입력하는 내용을 파일에 I/O 직접 담당한다면 수 MB의 작은 파일을 편집 하더라도 속도 때문에 큰 불편을 느낌(당연히 메모리 캐시의 도움이 필요함)
- 같은 원리로 데이터베이스(데이터를 저장하는 파일 집합)와 이를 액세스하는 프로세스 사이에 SGA라고 하는 메모리 캐시 영역을 두고 있음
- 수십 MB에 이르는 큰 파일을 열거나 저장할 때 오랜 시간이 소요되는 것을 경험하게 되는데 GB~TB급 데이터를 관리하는 DBMS가 그런 식으로 데이터를 읽고 쓴다는 것은 불가능하기에 블록 단위로 읽기, 저장하고 변경시에도 변경이 발생한 블록만 찾아 블록 단위로 저장
- 최신 워드 프로세서는 캐시와 파일 간에 주기적으로 동기화를 수행해 주는 자동저장 기능이 있는 것처럼 DBMS도 백그라운드에서 DBWR와 CKPT 프로세스가 캐시와 데이터 파일간 동기화를 주기적으로 수행

1.8 블록 액세스

Sequential vs Random



Sequential Access

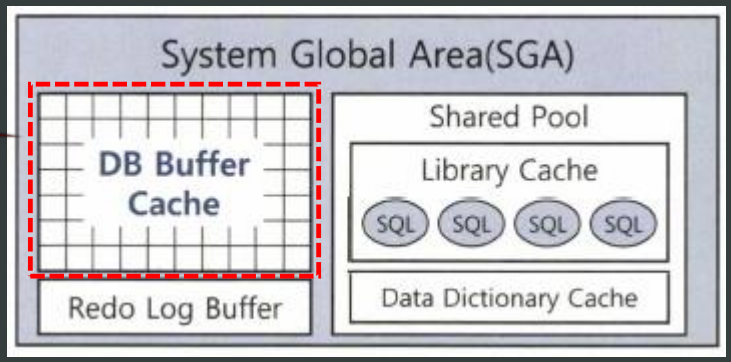
- 논리/물리적으로 연결된 순서에 따라 차례대로 블록을 읽는 방식으로 ①좌측의 인덱스를 스캔하는 굵은 실선 화살표와 ②테이블을 스캔하는 굵은 실선 화살표가 여기에 해당

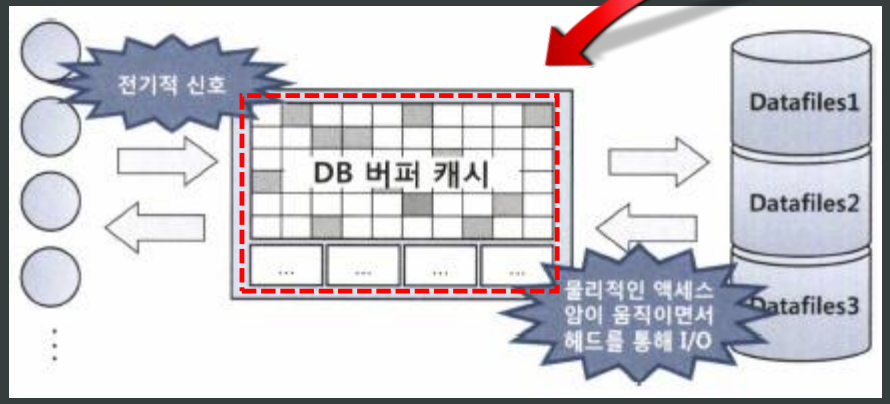
Random Access

- 논리/물리적인 순서를 따르지 않고, 레코드 하나를 읽기 위해 한 블록씩 접근하는 방식이며 죄측의 점선 화살표가 여기에 해당함

1.9 논리 I/O vs. 물리 I/O

DB Buffer Cache

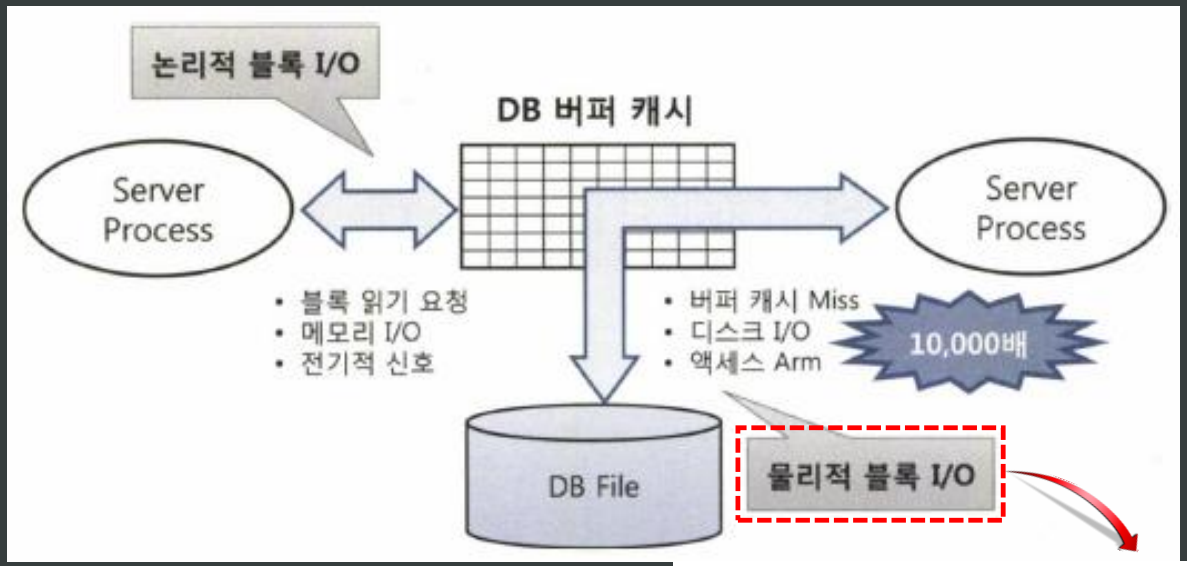


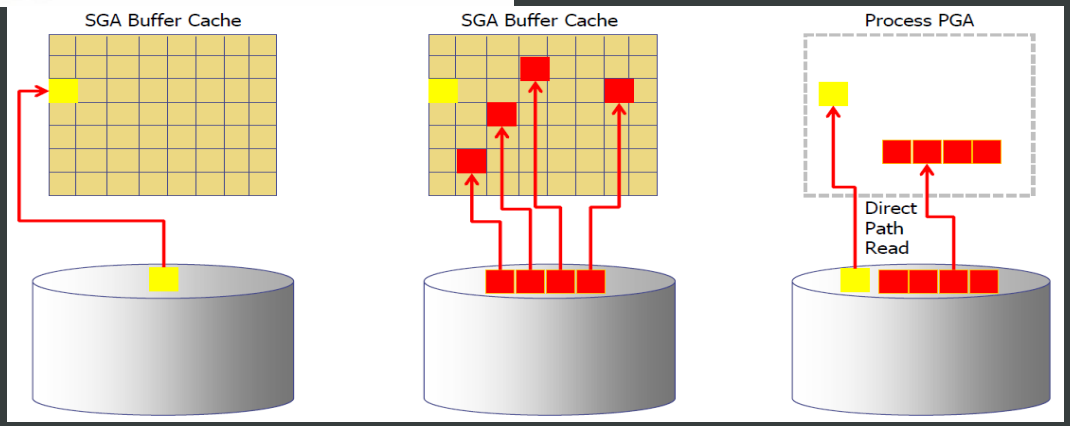


- Library Cache가 SQL과 실행계획, 저장형 함수/프로시저 등을 캐싱하는 “코드 캐시”라고 한다면, DB Buffer Cache는 "데이터 캐시"라고 할 수 있으며, 디스크에서 어렵게 읽은 데이터 블록을 캐싱함으로써 같은 블록에 대한 반복적인 I/O Call을 줄이는 데 목적이 있음
- 서버 프로세스와 데이터 파일 사이에 버퍼캐시가 있으므로 데이터 블록을 읽을 땐 항상 버퍼캐시부터 탐색함

1.9 논리 I/O vs. 물리 I/O (계속)

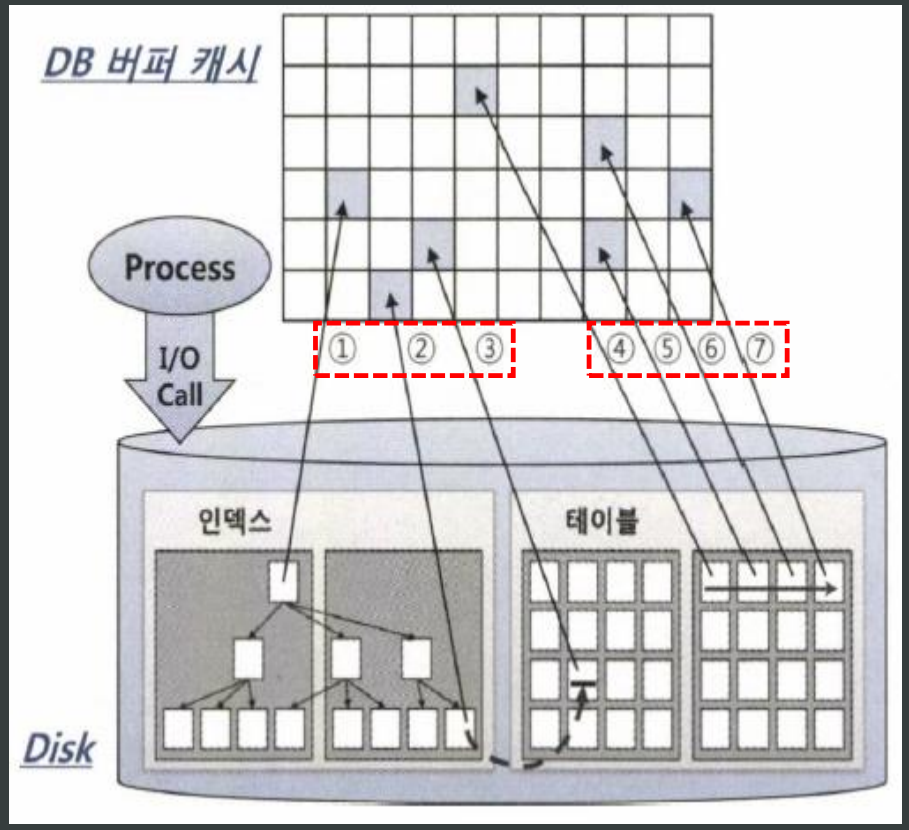
논리적 I/O vs. 물리적 I/O





1.10 Single / Multi Block I/O

Single vs. Multi



캐시에서 찾지 못한 데이터 블록을 I/O Call을 통해 디스크에서 DB 버퍼캐시로 적재시 한 번에 한 블록씩 요청해서 메모리에 적재하는 방식을 Single Block I/O, 여러 블록씩 요청해서 메모리에 적재하는 방식을 Multi Block I/O 라고 함

Single Block I/O

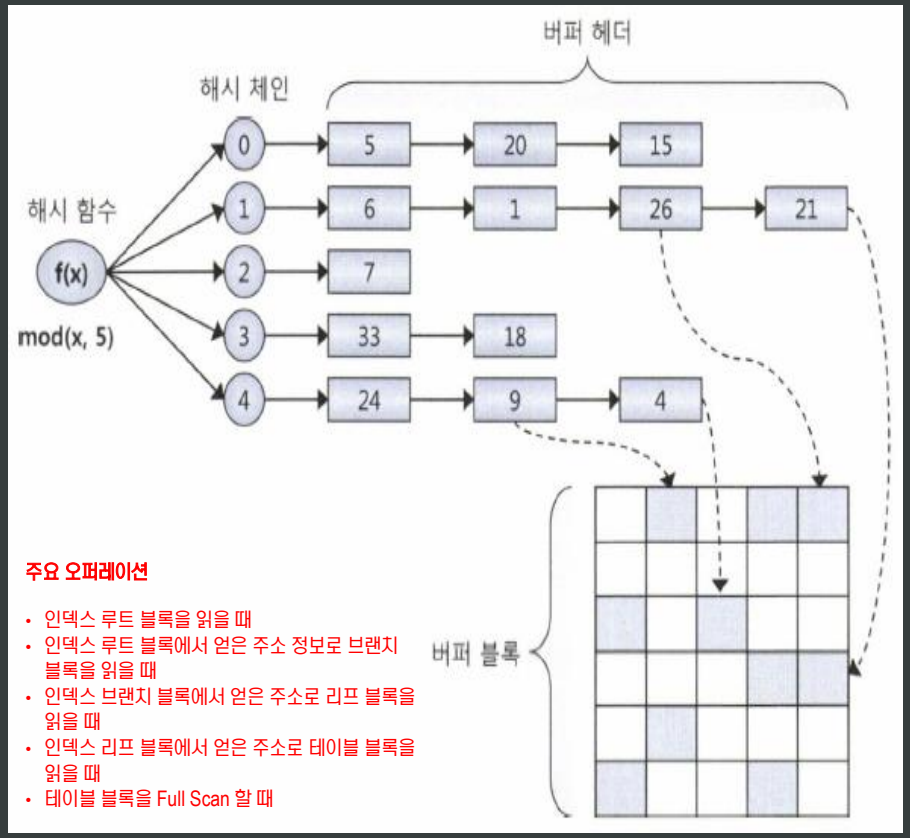
- ① ~ ③이 해당
- 인덱스 스캔, 소량 데이터 스캔시 효율적

Multi Block I/O

- ④ ~ ⑦이 해당
- 테이블 전체 스캔, 대량 데이터 스캔시 효율적

1.11 캐시 탐색 메커니즘

버퍼캐시 탐색 과정



Direct Path I/O를 제외한 모든 블록 I/O는 메모리 버퍼캐시를 경유하고 DBMS는 버퍼캐시를 해시 구조로 관리함

캐싱되어 있는 경우

- 20번 블록을 찾을 경우
- 첫번째 해시 체인(해시 값 '0')을 스캔

캐싱되어 있지 않은 경우

- 27번 블록을 찾을 경우
- 세번째 해시 체인(해시 값 '2')을 스캔
- 체인에 없을 경우, 디스크로부터 읽어서 연결

1.11 캐시 탐색 메커니즘 (계속)

메모리 공유자원에 대한 액세스 직렬화

- 버퍼캐시는 SGA 구성요소이므로 캐싱된 버퍼블록은 공유 자원임
- 모두에게 접근한 권한이 있지만, 하나의 블록을 두 개 이상 프로세스가 동시에 접근하려고 할 때 문제 발생
  - 동시에 접근하면 블록 정합성에 문제가 발생할 수 있음(Hot Block)
- 자원을 공유하기 위해선 내부적으로 한 프로세스씩 순차적으로 접근하도록 구현해야 하는 직렬화 필요
  (이러한 메커니즘이 Latch(Spin)임 – 버퍼캐시부터 탐식하기 때문)
- 캐시버퍼 뿐만 아니라 버퍼블록 자체에도 직렬화 메커니즘이 존재하며 "버퍼 Lock”이라 함
- 이런 직렬화 메커니즘에 의한 캐시 경합을 줄이려면, SQL 튜닝을 통해 논리적 I/O를 줄여야 함