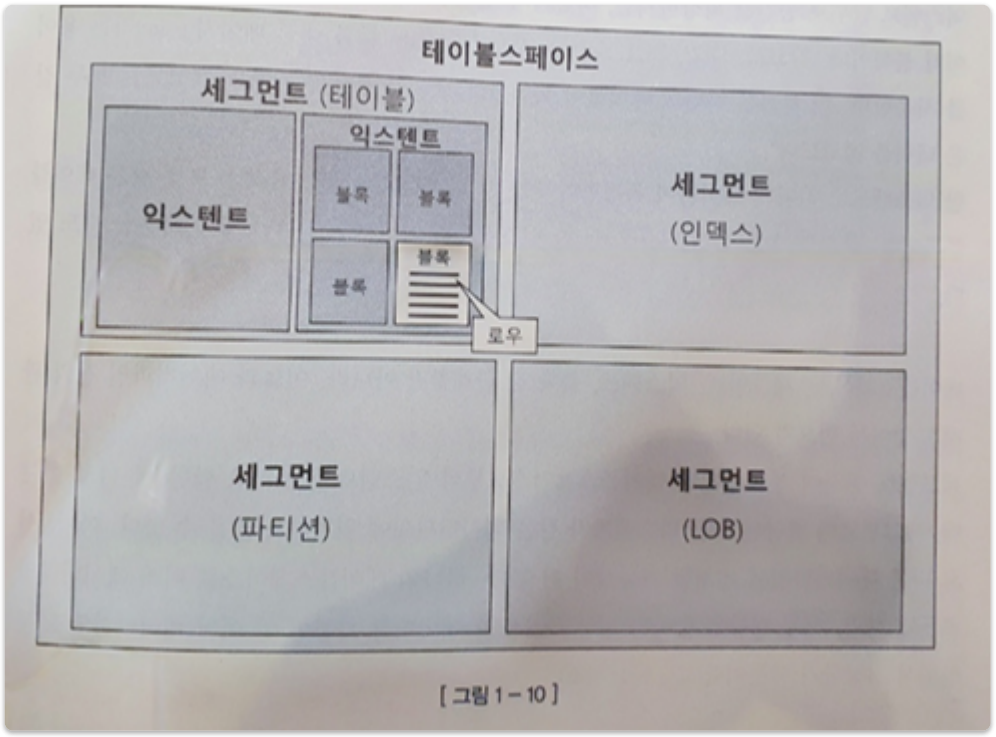
1.3 데이터 저장 구조 및 IO 메커니즘

💡 1.3.1 SQL이 느린 이유

- 디스크 I/O 때문이다
- 디스크 I/O 작업동안 프로세스는 대기하게 된다.
- 따라서 I/O 가 많으면 성능 저하가 이루어진다.
- 디스크 I/O 가 SQL 성능을 좌우한다.

💡 1.3.2 데이터베이스 저장 구조

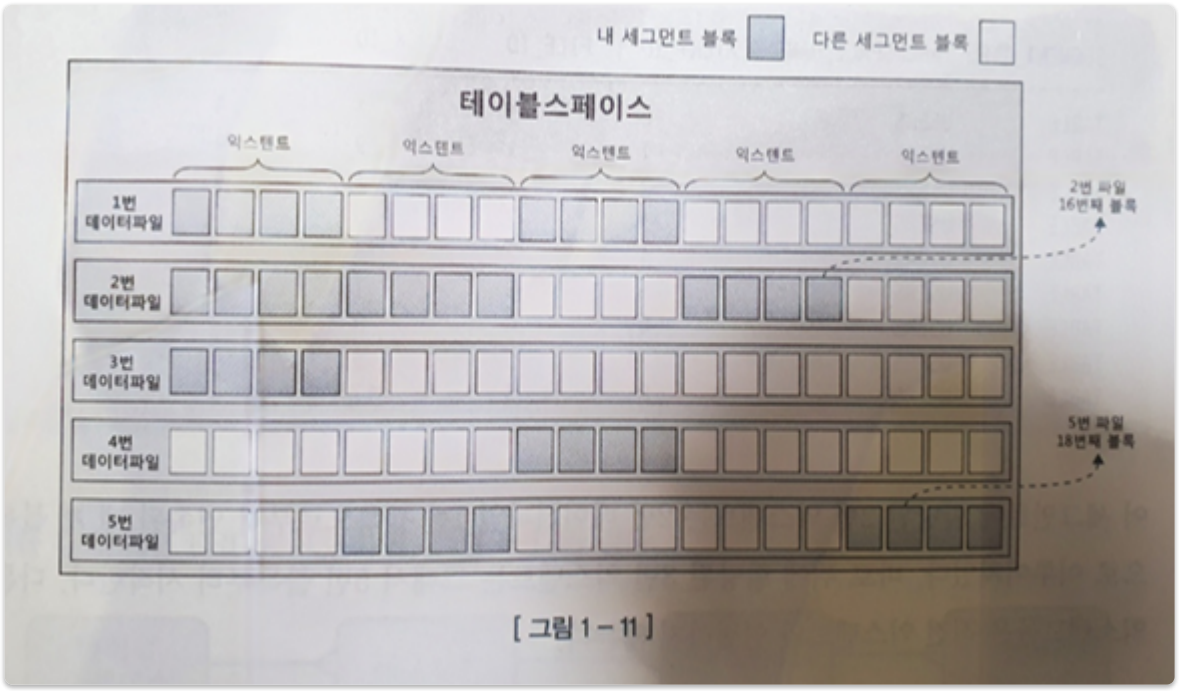
- 블록 : 데이터를 읽고 쓰는 단위
- 익스텐트 : 공간을 확장하는 단위, 연속된 블록 집합
- 세그먼트 : 데이터 저장공간이 필요한 오브젝트(데이블, 인덱스, 파티션, LOB 등)
- 데이블스페이스 : 세그먼트를 담는 콘테이너
- 데이터파일 : 디스크 상의 물리적인 OS 파일



[그림 1-10]

- 세그먼트에 할당된 모든 익스텐트가 같은 데이터파일에 위치하지 않을 수 있다.

- 파일 경합을 줄이기 위해 DBMS 가 데이터를 가능한 여러 데이터파일로 분산해서 저장하기 때문이다.



[그림 1-11]

- 익스텐트 내 블록은 서로 인접한 연속된 공간이지만, 익스텐트끼리는 연속된 공간이 아니다

> - *DBA(Data Block Address) : 데이터 블록의 고유 주소값*
> - *인덱스 ROWID 는 DBA+로우번호(블록내순번) 으로 구성*
> - *테이블 스캔시 테이블 세그먼트 헤더에 저장된 익스텐트 맵을 통해 각 익스텐트의 첫번째 블록 DBA 를 알 수 있다. 그 후 연속적 스캔을 진행*

💡 1.3.3 블록 단위 I/O

- 블록 : DBMS 가 데이터를 읽는 단위
- 특정 레코드 1개를 읽고 싶어도 블록을 통째로 읽는다.
- 오라클은 기본 8KB 크기 블록 사용(1Byte 읽기 위해 8KB 읽음)
  - 인덱스도 블록 단위로 데이터를 읽는다.

💡 1.3.4 시퀀셜 액세스 vs 랜덤 액세스

1. 시퀀셜(Sequential) 액세스

- 논리적, 물리적으로 연결된 순서에 따라 차례대로 불록을 읽는 방식
- 예) 인덱스 리프 블록을 논리적으로 읽는것
- 테이블 블론간에는 서로 논리적인 연결고리가 없지않아?

> - *세그먼트에 할당된 익스텐트 목록을 세그먼트 헤더에 맵으로 관리, 익스텐트 맵은 각 익스텐트의 첫번째 블록 주소값을 가진다. 이를 통해 연속적으로 블록을 읽으면 Full Table Scan 이다.*

2. 랜덤(Random) 액세스

- 논리적, 물리적 순서를 다르지 않고 레코드 하나를 읽기 위해 한 블록씩 접근

💡 1.3.5 논리적 I/O vs 물리적 I/O

1. DB 버퍼캐시

- 자주 읽는 블록을 매번 디스크에서 읽는 것은 비효율적이다
- SGA 구성요소로 DB 버퍼캐시가 있다.
  - 라이브러리 캐시(코드 캐시) : SQL과 실행계획, DB 저장형 함수/프로시저 등을 캐시
  - DB 버퍼 캐시(데이터 캐시) : 디스크에서 어렵게 읽은 데이터 블록을 캐싱하여 같은 블록에 대한 반복적인 I/O Call 을 줄인다.
- 데이터 블록을 읽을 때 항상 버퍼캐시터 탐색
- 공유메모리 영역이므로 같은 블록을 읽는 다른 프로세스도 득을 본다

2. 논리적 블록 I/O vs 물리적 I/O

1) 논리적 블록 I/O

- SQL 처리하는 과정에 발생한 총 블록 I/O
- 메모리상의 버퍼캐시를 경유하므로 메모리 I/O(전기적 신호) 와 동일

2) 물리적 블록 I/O

- 디스크에서 발생한 총 블록 I/O
- 블록캐시에서 찾지 못한 경우 디스크 액세스하므로 논리적블록 I/O 중 일부를 물리적으로 I/O 한다.
- 디스크 I/O 는 물리적 작용이 일어나므로 굉장히 느리다.

3. 버퍼캐시 히트율(Buffer Cache Hit Ratio, BCHR)

> *BCHR = (캐시에서 곧바로 찾은 블록 수 / 총 읽은 블록 수) 100*
> *BCHR = (1 - (물리적 I/O) / (논리적 I/O)) 100*

- 평균 99%를 달성해야 한다.
- 실제 SQL 성능 향상을 위해서는 물리적 I/O가 아닌 논리적 I/O를 줄여야 한다.

> *물리적 IO = 논리적 I/O * (100 - BCHR)*

- 논리적 I/O 는 일정하므로 물리적 I/O 는 BCHR 에 의해 결정된다
- BCHR 은 시스템 환경에 따라 달라진다
- 물리적 I/O 는 결국 통제 불가능한 외생변수에 의해 발생한다.
- SQL 성능은 결국 논리적 I/O 를 줄이는 것이다.

- 논리적 I/O 를 줄임으로 써 물리적 I/O 를 줄이는 것이 곧 SQL 튜닝이다

💡 1.3.6 Single Block I/O vs

1. Single Block I/O

- 한번에 한 블록씩 요청하여 메모리 적재
- 인덱스는 기본적으로 Single Block I/O
- 인덱스는 소량 데이터를 읽을 때 주로 사용

2. Multiblock I/O

- 한번에 여러 블록씩 요청하여 메모리 적재
- 많은 데이터를 읽을때
- 테이블 전체 스캔시 사용
- 단위가 크면 더 효율적이다(한번 I/O 작업으로 프로세스 쉴때 많이가져온다)
- 대용량 테이블 full scan 시 multiblock 단위를 크게 설정하면 성능이 좋다
- OS 단에서는 보통 1MB 단위로 I/O 수행한다
  - (8KB * 128 = 1MB) 오라클에서는 I/O 단위가 8KB 이므로 한번에 128을 가져오는게 최대이다
- 하지만 익스텐트 경계를 넘지는 못한다.

> *왜 multiblock i/o 중간에 single block i/o ? -> 책 59 참조*

💡 1.3.7 Table Full Scan vs Index Range Scan

1. Table Full Scan

- 테이블에 속한 블록 전체를 읽는다

2. Index Range Scan

- 인덱스에서 일정량을 스캔하면서 얻는 ROWID(테이블 레코드가 디스크상 어디 저장되었는지) 로 테이블 레코드를 찾는다

- Table Full Scan 은 Multiblock I/O 이므로 대용량의 경우 index 가 아닌 full scan 을 하는게 효과적이다.
- 또한 인덱스는 블록을 반복적으로 읽으므로 비효율적일수도있다

💡 1.3.8 캐시 탐색 메커니즘

- Direct Path I/O 를 제외한 모든 블록 I/O 는 메모리 버퍼캐시를 경유한다.

- 버퍼캐시는 해시 구조로 관리된다.
- 해시 알고리즘으로 버퍼 헤더를 찾고, 얻은 포인터로 버퍼 블록을 액세스한다.

1. 메모리 공유자원에 대한 액세스 직렬화

- 버퍼캐시는 SGA 구성요소로 공유자원이다.
- 동시성의 문제가 존재한다.
- 한 프로세스씩 순차적 접근이 가능하도록 직렬화 메커니즘이 필요하다

2. 래치(Latch)

> *버퍼 캐시에서 해시 체인을 탐색하면서 대량의 데이터를 읽는데, 그사이에 체인이 수정되면 안된다. 따라서 해시체인 래치가 존재하고 키를 획득한 프로세스만이 진입가능하다.*

- SGA 를 구성하는 서브 캐시마다 별도의 래치 존재
- 래치에 의한 경합이 높으면 성능 저하 발생
- 버퍼블록에도 직렬화 메커니즘이 존재한다 : 버퍼 Lock
- 결국 SQL 튜닝을 통해 쿼리 일량(논리적I/O)를 줄여야 한다