



I. 데이터베이스 아키텍처

1) 아키텍처 개관

+ 모델링의 정의

- 데이터베이스 . 디스크에 저장된 데이터 집합

- 인스턴스 . SGA 교유 메모리 영역과 이를 애플리케이션 프로세스 집합

↳ DB의 구조변경이나 변경사항이 있을 때 자동으로 업데이트,
모든 데이터 파일, 그 파일의 정보(경로, 이름 등)를 가능

⇒ 기본적으로 하나의 인스턴스가 하나의 DB만 액세스하되만, RAC(Real Application Cluster) 환경에서 여러 인스턴스가
한나의 DB를 액세스할 수 있음, 하나의 인스턴스가 여러 DB를 액세스할 수는 없음

t) SGA(System Global Area) : 모든 사용자가 교유 가능하여 사용

PGA(Program Global Area) : 사용자마다 소유하지 않고 개별적으로 사용

④ SQL Server 아키텍처

하나의 인스턴스당 최고 32,767개의 DB 지원해 사용 가능

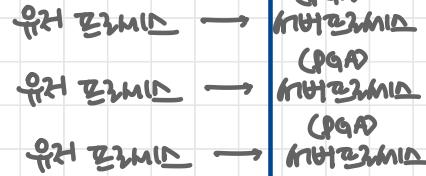
DB 하나를 만들때마다 주 데이터 파일과 트랜잭션 파일과 로그 파일이 생성

DB엔진 관리

모든 변경사항을 기록하는 곳

→ Datafile, Redo Log File, Control File 등

• PGA



- DB에 접속하기 위해 필요로 하는 정보면 내부적으로 유저프로세스가 아니라
 - 해당 유저프로세스는 DB에 접속하는 순간 모든 자원은 Shared Pool 서비스에 전수
 - 해당 유저 프로세스에서 퍼센트 SQL 수행했다고 하면 유저프로세스는 해당 SQL과 기타 정보는 서버 프로세스에게 전달해주고 별도 대기하는 역할
 - 유저 프로세스로부터 퀼리반을 내보내기 위해서 Shared Pool이 필요로 하는 작업은 수행
- 퀼리반은 내부 및 기타 작업을 처리하기 위해 자체 서비스는 자신의 일의 범위로만
- PGA 이용

⇒ PGA: DB에 접속하는 모든 유저에게 할당되는 각 서버프로세스가 퀼리반을 사용하는 외부 메모리 영역,

정렬구간	시작값	커서상태정보	변수저장구간	PGA
------	-----	--------	--------	-----

- * 정렬구간 (Sort Area): Order By 및 Group By 등의 정렬을 위하여 주는 공간
 - 해당 구간내에서 정렬이 완료되면 메모리 정렬이 가능.
 - 해당 정렬 메모리 구간이 부족하면 더 이상 이용

- * 세션정보 (Session Information): 서버프로세스에 의해 추적된 결과값을 저장하기 위한 퍼센트 유저 프로세스의 세션자료 저장

- * 커서상태정보 (Cursor State): 해당 SQL의 실행 정보가 저장되어 있는 구간 저장

- * 변수저장구간 (Stack Space): SQL 문장에 바인드 변수 (Bind Variable)를 사용했을 때 해당 바인드 변수 저장

* 대형풀 : 반드시 지정해야 한 SGA 영역을 이용

→ 지정시 공유풀의 부하 감소

User Global Area : SQL문, 저장 및 변수 등을 저장하는 있는 베이스에 메모리 영역이며

UGA 영역 저장

병렬 프로세스의 자비난 처리 : 여러개의 프로세스를 통하여 하나의 SQL수행

이때 각 프로세스 간 메시지 주고 받음 → 병렬 프로세스 처리

RMAN : RMAN 유한번의 사용에 여러개의 디스크 I/O 속도이번 프로세스 가능

I/O 세션 프로세스 : DBWR 백그라운드 프로세스는 해당 프로세스 아래 솔레이브 프로세스 가능내 더 빠른 디스크 I/O 작업 수행

* 자비난 : 자비연령을 구분분석할 경우 사용하는 메모리 공간

→ 자비나 분석하고 사용할 경우 자정해야 함

2) 프로세스

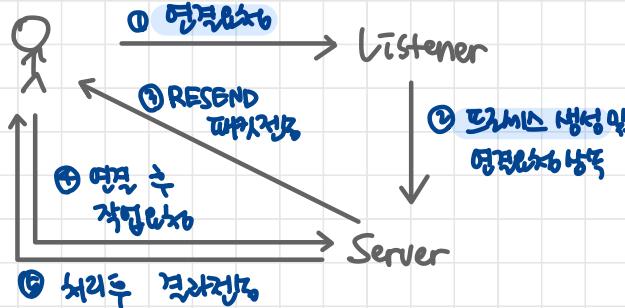
■ 서버프로세스: 사용자가 터미널 명령어를 처리

■ 배포환경 프로세스: 뒤에서 처리

① 서버프로세스: SQL 파트, 필요한 면허증 처리, 구현된 SQL 구문 처리에서 복잡 일정,
있을 때이면 성장과 함께 데이터베이스가 성장한 결과집합 만들어 네트워크를 통해 전송하는 일정의 작업 모두 처리

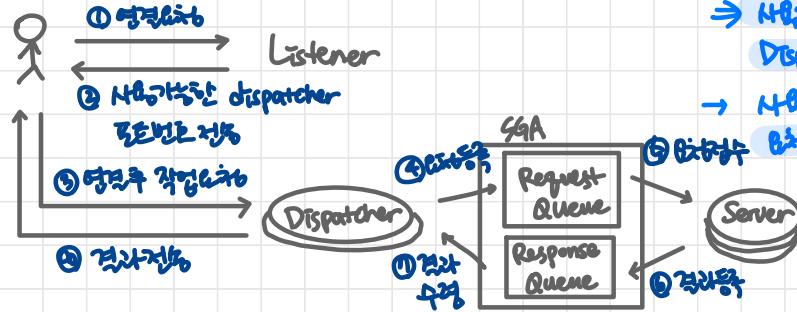
→ 데이터베이스가 서버프로세스와 연결되는 방식

전용데이터방식: 연결할 때만 커넥션이 서버프로세스에 생성되고, 서버프로세스가 데이터를 읽기 전용서비스처럼



⇒ SQL 수행작업과 연결되는 방식으로
서버프로세스의 생성과 해제도 방식 → 서버프로세스
∴ 전용 서버 방식을 사용하는 MySQL은 애플리케이션에는
Connection Pooling 기법 필수로 사용해야함
↳ 초기의 서버 프로세스와 연결된
다개의 사용자 프로세스는 모두에 연결 가능

접근제어방식: 자신의 서비스 프로세스를 여러 사용자에게서의 접근하는 방식 (여기 여러개의 서비스 프로세스는 학생하고 이는 교과목에 따른 별도로 배포, 관리)



→ 사용자 프로세스는 서비스 프로세스와 직접 통신X

Dispatcher 프로세스 거친

→ 사용자 프로세스가 Dispatcher에게 접속하면 Dispatcher는 이를 SGA에 올린다

⑤ 접속한 사용자에게 등록

→ 이후 가능한 사용자를 위한 서비스 프로세스가 응답큐에 있는 사용자에게만 서비스를 제공

→ 그 결과는 응답 큐에 등록

→ 응답 큐는 마지막으로 Dispatcher가 응답 결과를 반영시, 사용자 프로세스에게 전송

* 배경작동 프로세스

System Monitor (SMON) : 장애 발생을 시도해 재가동시, 인스턴스 복구 수립, 일시 세션과 트랜잭션 모니터링

Process Monitor (PMON) : 이상이 생긴 프로세스가 사용하면서 다른 복구

Database Writer (DBWn) : 버퍼캐시에 있는 Dirty 버퍼를 데이타 파일에 기록

Log Writer (LGWR) : 32비트 레코드는 Redo 로그 파일에 기록

Archiver (ARCn) : 막힌 Redo 로그가 덮어쓰여지기 전에 Archive 로그 디렉토리로 백업

Checkpoint (CKPT)

Recoverer (RECO) : 복구 트랜잭션 과정에 발생한 문제를 해결

3) 파일구조

7. 데이터 파일

→ 외국 SQL Server

① **복수(=페이지)**: 대부분 DBMS에서 I/O는 복수 단위로 이루어짐

한국어 2KB, 4KB, …, 64KB의 다양한 복수 크기 사용할 수 있지만

SQL Server는 8KB 단위로 사용함

⇒ SQL 성능지표: 액세스하는 복수 개수(읽어야 하는 레코드 수)

② **리프레인트**: 테이블 스페이스를 뿐만 아니라 고급을 학습하는 단위

테이블이나 인덱스에 데이터 일관성이 있는 고급이 부족하거나 해당 오브젝트가 다른 테이블 스페이스를 뿐만 아니라 고급 학습방법

ex) 복수 크기가 64KB인 상대에서 64KB 단위로 이드센트 학습하도록 전의하면 고급이 복잡한 편이다

테이블 스페이스로부터 8개의 연속된 복수를 찾아서 리프레인트에 학습해온

이스렌트 내 복수는 일관적으로 인접, but 이드센트끼리 서로 인접 X

[SQL Server는 2가지 태입의 이스렌트 사용]

• **균일 (Uniform) 이스렌트**: 64KB 이상의 공간을 필요로 하는 테이블이나 인덱스는 유래 사용됨

8개 페이지 단위로 학습한 이스렌트는 단일 오브젝트가 모두 사용됨

• **혼합 (Mixed) 이스렌트**: 한 이스렌트에 학습된 8개의 페이지는 여러 오브젝트가 나누어 사용하는 형태
모든 테이블 → 처음에는 혼합으로 시작, 64KB를 넘으면서 2번째부터 균일 사용

한국의 제이드에서 한국의 경영인 읽으려고 할 때도
→ 레코드가 딱한 복수 단위로 인접됨

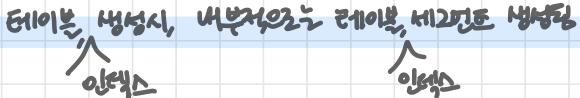
이스렌트 내 복수는 일관성 +
이드센트끼리 인접 X

복수 단위로 학습하는
이스렌트 구조

정비진 이스렌트 초기화
여러번 복수 학습 방식

초기화된 복수 학습 방식

③ 세그먼트: 테이블, 인덱스, Undo 처리를 저장하는 공간을 필요로 하는 데이터베이스 오브젝트
한개 이상의 이스케일 사용

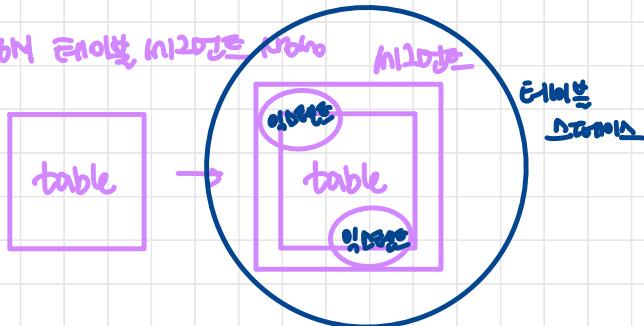


→ 데이터 저장에 레이블, 세그먼트 사용

다른 오브젝트는 세그먼트와 1:1 대응 관계 갖지만, 파일은 1:m 관계 갖음

즉, 파일은 레이블을 만들면, 내부적으로 여러개의 세그먼트가 만들어짐

세그먼트에 확장된 이스케일이 여러 데이터 파일에 흩어져 저장됨



④ 테이블 스페이스: 세그먼트를 담는 컨테이너. 여러 데이터 파일로 구성됨

데이터는 물리적으로 데이터 파일에 저장되지만, 사용자가 데이터 파일의 적절성을 X

사용하는 세그먼트는 유틸리티 툴을 사용해 빼기

문제 없이 저장된 데이터 파일을 선택하고 이스케일을 학파하는 것은 DBMS 을

각 세그먼트는 한 테이블 스페이스안에 있으며, 한 테이블 스페이스에는 여러 세그먼트가 포함될 수 있음

[Oracle 저장구조]

테이블 스페이스 → 세그먼트 → 이스케일 → 블록



[SQL Server 저장구조]

(8GB 페이지)

파일그룹 → 힙/인덱스 → 이스케일 → 페이지
(64KB)



1. 임시 레이터 파일 : 대량의 데이터나 해시작업 후에는 메모리 공간이 부족해지면 충분한 경과 짧은 저장시간으로 임시로 저장했다가 자동으로 삭제됨,

Redo 정보는 병렬화에 있어 후후 파일에 문제 발생을 예방하지 X, ∴ 버려지는 필요 X

오라클에서는 임시 테이블, 스페이스 여러개 사용해두고

내용자마다 별도의 임시 테이블, 스페이스 저장해 줄 수 O

SQL Server는 데이터의 tempdb 데이터베이스 사용

전역리스, 시스템에 연결된 모든 사용자의 임시 레이터 여기에 저장

2. 로그파일 : DB 데이터 개시에 가해지는 모든 변경사항을 기록하는 파일을 오라클은 'Redo Log', SQL Server는 '트랜잭션 로그'

변경된 메모리 버퍼 복제를 디스크상의 데이터복제에 기록하는 작업은 random I/O 방식으로 이루어져 있음

로그기록은 Append 방식으로 이루어져야 함, ∴ 대부분 DBMS는 버퍼복제에 대한 변경사항을 전부 데이터 디스크에 기록하기 보자 우선 로그파일의 Append 방식으로 버퍼에 기록하는 방식 사용

이후, 버퍼복제와 데이터 파일간 동기화는 저장한 수단(DBWR, Checkpoint)을 이용해 나중에 버지 방식으로 일괄 처리 사용자의 그룹화된 메모리의 버퍼 복제에만 기록된 채 아직 디스크에 기록되지 않았더라도 Redo Log는 막고

버퍼에 거칠 외연화하는 의미에서 Fast Commit 대체법이라고 함

→ 인스턴스 종료가 발생해도 로그파일 이전에 기록된 복구 가능함 → Fast Commit은 버퍼에 트랜잭션 저장하는데는 모든 DBMS의 공통적인 대체법

• Online Redo 로그 (온라인)

- : 캐시에 저장된 명령어들이 아직 데이터베이스에 기록되지 않은 상태에서 인스턴스가 비정상 종료하면, 그때까지의 작업 내용을 알게됨
이러한 트랜잭션은 데이터의 유실에 대비하기 위해 Online Redo 로그 사용
- : 아직 체크포인트 이후부터 사고 발생 직전까지 수행되었던 트랜잭션들은 Redo 로그를 이용해 재현 → '캐시 복구'라고 함
Online Redo 로그는 최대 2MB 이상의 파일로 구성됨 → 현재 사용중인 파일이 막 차면 다음 파일로 **로그 스위치** 발생
제작 로그 세 개다가 모두 파일 막차면 다시 첫번째 파일부터 재사용하는 轮回 방식 사용

• 트랜잭션 로그 (SQL Server)

- : SQL Server의 고급파일, 주 데이터 파일마다 속 DB마다 트랜잭션 고급파일이 하나씩 생성
트랜잭션 로그파일은 내부적으로 '가상로그 파일'이나 복기는 더 작은 단위의 세그먼트로 나뉨
→ 가상로그파일의 개수가 너무 많아지지 않도록 (디스크가 부족하거나 많도록) 끊어서 사용
(ex. 로그파일을 애초에 네트워크 카드에 만들어 사용하거나 많도록 자동증가한다면 증가하는 만큼 초기화)

• Archived (= Offline) Redo 로그 (온라인)

- : 온라인에서 Online Redo 로그가 재사용되기 전에 다른 위치로 백업해둔 파일
혹은 파일 저장매체에 백업한 파일을 때 DB 복구를 위해 사용됨

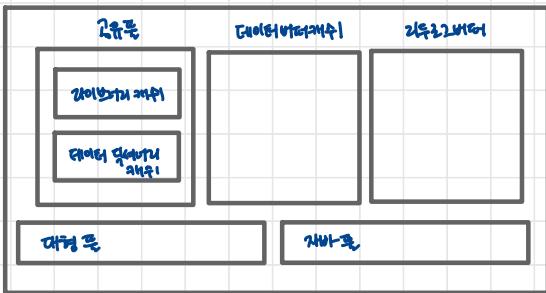
- ④ 메모리 구조 → 시스템 메모리 영역, 프로세스 전용 메모리 영역,
- 시스템 메모리 영역: 여러 프로세스가 동시에 액세스할 수 있는 메모리 영역
 - 오라클에서는 System Global Area(SGA), SQL Server에서는 Memory Pool이라고 함
 - ⇒ DB 버퍼캐시, 끝유辱, 블록버퍼, 러치풀, 자바풀, 시스템 구조와 제어구조는 차별화된 영역,
 - ⊕ 내부적으로 래치, 버퍼 lock, 러기ண더리캐시 lock/pin 같은 액세스 자료구조 대체로 사용됨
 - 프로세스 전용 메모리 영역: 오라클은 프로세스 기반 아키텍처 ∴ 서버 프로세스가 자원의 메모리 영역 가질 수 있음 ⇒ Process Global Area(PGA)
 - 데이터 저장, 세션과 커넥션 관리, 낭비 저해 차별화된 영역
 - 쓰레드 기반 아키텍처를 사용하는 SQL Server는 프로세스 전용 메모리 영역은 가지치 X
 - ↳ 전용 메모리 영역, 가지치 X, 부모 프로세스의 메모리 영역을 사용하기 때문

• SGA (System Global Area)



SGA: 공유메모리 영역

∴ 동일 DB에 접속하는 모든 사용자는 동일 SGA를 사용



⇒ SGA: 고려할 데이터는 있거나 블록화된 후에 사용되는 공유메모리 영역

* 고급 - 고정영역, 동적영역 ⇒ LRU 알고리즘 적용

고정영역. 고정영역은 SGA를 관리하는 메커니즘 및 오라클 최적화의 기본기-장장,
 고정영역의 크기는 고정되어 설정값 등을 바꾸어 자동으로 할당

사용자가 지정할 수 X



동적쿼리 - 라이브러리 캐시, 데이터베이스 캐시

- **라이브러리 캐시**: DB에 접속한 유저가 뉴런은 SQL, 오라클이 내부적으로 사용하는 SQL, SQL에 대한 분석 정보 및 실행계획 저장
- **데이터 디스커버 캐시(로우캐시)**: 테이블, 인데스, 향수 및 트리거 등 오라클 오브젝트 정보 및 구조 등의 정보가 저장

ex)

SELECT * FROM 사원 →

뉴런 쿼리

라이브러리 캐시

데이터 디스커버 캐시

→ SELECT * FROM 사원

→ 사원테이블 정보

* 디스커버 캐시: 테이블, 인데스 등을 오브젝트 + 테이블 스키마, 데이터 타입, 제약조건, - 메타정보는 저장

ex) '주문' 테이블 → 임의한 구문 데이터는 데이터 파일에 저장됐다가 버퍼캐시를 경유해 읽어지면,
테이블 메타 정보는 디스커버 캐시에 저장됐다가 디스커버 캐시를 경유해 읽힘

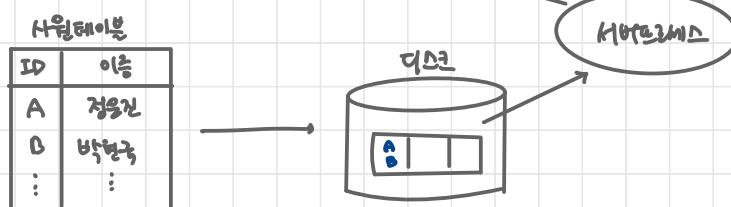
* 라이브러리 캐시: 사용자가 수행한 쿼리를 뉴런에 넘기면, 저장 프로시저는 저장해두는 캐시에서

사용자가 쿼리 명령어를 통해 결과집합 조회하면 이를 헤더으로 (가장 처음 나오는 사용해 빠르게) 수행하기 위해 → 쿠드파마 쿼리 수행을 위해 내부적으로 뉴런에 있는 프로시저

→ 쿼리 구문을 분석해 운영 외부 및 뉴런 구조 등을 체크, 최적화, 쿼리 실행환경이 조건을 만족하는 경우로 맞춰준다.

① SQL이 데이터베이스

- * 데이터 베이스 캐시 구조에서 응답성이 높은 데이터 베이스 캐시
- ex) SELECT ID FROM 사용
WHERE 이름 = '정우진';



② 데이터 베이스 캐시에 해당 데이터는 데이터베이스에

- ③ 디스크 → 리버스 캐시 → 디스크은 데이터베이스에
디스크부터 데이터베이스 캐시로 읽음

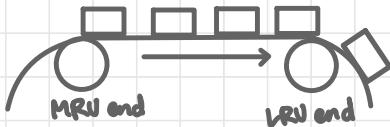
- ▼ 데이터 파일로부터 읽을 데이터 블록은 딥캐시에 있음
- ▼ 인터널에 접속한 모든 사용자 프로세스는 서버프로세스를 통해 DB 베이스 캐시의 블록들을 동시에 (내부적으로는 베이스 캐시를 통해) 읽어온다.
- ▼ 모든 블록 앤드캐시는 통해 이동 → 앤드캐시는 블록은 먼저 데이터베이스에서 찾았고, 찾을 때 디스크에서 읽음
(디스크에서 읽을 때도 베이스 캐시에 고려한 후 읽음)
- ▼ 데이터 변경도 베이스 캐시에 저메인 블록을 통해 이루어짐
→ 데이터 변경 (DML) 베이스 블록은 주기적으로 데이터 파일에 기록하는 작업은 DBWR 프로세스의 몫
- ▼ 디스크 I/O는 물리적으로 액세스 암(Addr)이 물리적으로 헤드로 들어온 이후에才是 베이스
액세스 I/O는 물리적 시점으로 이루어지기 때문에 베이스
- ∴ 디스크에서 읽을 데이터 블록은 예전에 상에 보관해둔 가능성이 모든 DB 시스템에 필수적인 이유

* 버퍼 봉록의 상태

- Free 버퍼: 인스턴스 기동 후 아직 데이타가 있하지 않은 비어있는 상태 (Clean 버퍼)이나, 데이타가 담겼지만 데이타 파일과 서로 동기화된 대는 상태여서 블록을 떠나면서도 유통하고 버퍼 봉록 데이터 파일과 브록 세트를 데리고 부터 세트를 데리고 봉록을 조정하려면 먼저 free 버퍼 확보해야 함
free 상태인 버퍼에 변경이 발생하면 그 순간 dirty 버퍼로 상태 바뀜
- Dirty 버퍼: 버퍼에 개시된 이후 변경이 발생했지만, 아직 디스크에 기록되지 않은 데이타 파일 봉록과 동기화가 필요한 버퍼 봉록
버퍼 봉록들이 다른 데이터 봉록은 위해 재정렬화하면 디스크에 먼저 기록되어야 함
→ 디스크에 기록되는 순간 free 버퍼로 상태 바뀜
- Pinned 버퍼: 앓기 또는 쓰기 작업이 현재 진행 중인 버퍼 봉록

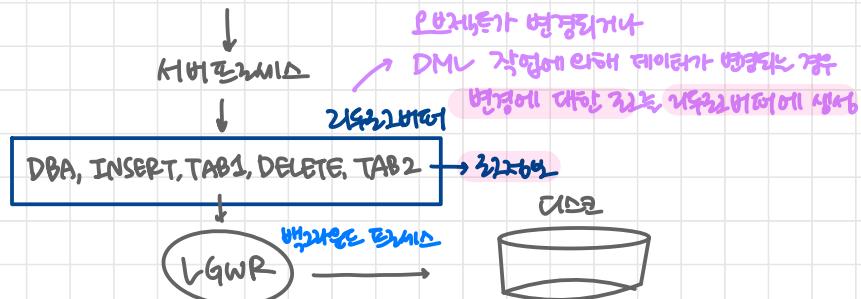
* LRU 알고리즘

DBMS는 사용빈도가 높은 데이터 봉록 위주로 버퍼캐시가 구성되도록 LRU (least recently used) 알고리즘 사용
모든 버퍼 봉록 데이터를 LRU 원칙에 맞춰 사용빈도 순으로 위치를 옮겨간다 free 버퍼가 필요해질 때 어제스 빈도가 높은쪽 (LRU end)
데이터 봉록부터 데이터는 방식



* 거두고 버퍼 : 오브젝트 및 데이터 변경시 사용되는 값을 저장하는 SGA 메모리간

ex) INSERT INTO TAB1 (ID, 이름) VALUES ('A123', '황현')



→ 로그 엔트리도 파일에 곧바로 기록하는 것이 아니라 로그버퍼에 먼저 기록

기록이 디스크에 기록하기보다 일정 czas을 모았다가 기록하면 훨씬 빠르기 때문

⇒ 서버프로세스가 데이터 블록 버퍼에 변경을 가하니 전에 Redo 로그 버퍼에 먼저 기록해두면 주기적으로 LGR 프로세스가

Redo 로그파일에 기록함 → 변경이 가해진 블록의 버퍼를 데이터 파일에 기록하기 전에 항상 로그버퍼는 먼저 로그파일에

기록해야함 → 인스턴스 장애가 발생할 때 로그파일에 기록된 내용은 자동으로 캐시보드 복구, 차급복원 가능하지 않을

트랜잭션을 즉시 해야함, 로그파일에 있는 변경내역이 이미 데이터 파일에 기록되었으면 사용자가 향후 커밋하지 않을

트랜잭션이 커밋되는 경지로 유지되기 때문

→ 버퍼 캐시 복구를 개선하기 전 변경사항은 먼저 로그 버퍼에 기록해야함,

마지막 버퍼를 디스크에 기록하기 전에 해당 로그 블록은 먼저