Oracle Memory Architecture 학습 노트

SGA(System Global Area) 집중 학습

학습 일자: 2025년 12월 24일
학습 범위: Oracle Database 19c Memory Architecture - Overview of the System Global Area (SGA)
참고 문서: https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/memory-architecture.html

---

1. SGA(System Global Area)의 아키텍처적 의의

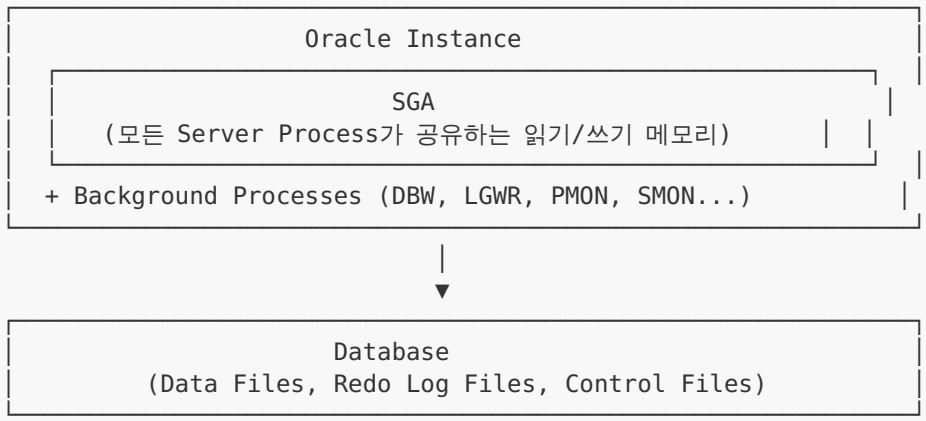
1.1 Why: SGA가 존재하는 구조적 이유

Oracle의 메모리 아키텍처를 이해하려면 먼저 **Instance와 Database의 분리**라는 핵심 설계 원칙을 상기해야 한다. Database는 디스크에 저장된 물리적 파일들의 집합이고, Instance는 이 파일들에 접근하기 위한 **메모리 구조 + 백그라운드 프로세스**의 조합이다.

SGA는 이 Instance의 메모리 부분을 담당하며, 공식 문서의 정의는 다음과 같다:

> "The SGA is a read/write memory area that, along with the Oracle background processes, form a database instance."

여기서 핵심 단어는 **"form"**이다. SGA와 백그라운드 프로세스가 **결합되어야** 비로소 Instance가 된다는 의미다. 즉, SGA 없이는 Instance가 존재할 수 없고, Instance 없이는 클라이언트 Connection이 불가능하다.



설계적 의의: SGA를 Instance 레벨의 공유 메모리로 분리한 이유는 **다중 사용자 환경에서의 효율성** 때문이다. 만약 각 세션이 독립적으로 SQL 파싱 결과나 데이터 블록을 관리한다면, 동일한 정보가 세션 수만큼 중복 저장되어야 한다. SGA는 이러한 중복을 제거하고 모든 Server Process가 동일한 메모리 영역을 공유하도록 설계되었다.

1.2 How: SGA의 핵심 컴포넌트 구조

Instance 시작 시 출력되는 SGA 정보를 살펴보면:

```
SQL> STARTUP
ORACLE instance started.

Total System Global Area  368283648 bytes
Fixed Size                  1300440 bytes   -- 내부 관리용 고정 영역
Variable Size             343935016 bytes   -- Shared Pool, Large Pool 등
Database Buffers           16777216 bytes   -- Buffer Cache
Redo Buffers                6270976 bytes   -- Redo Log Buffer
```

이 구성요소들 중 백엔드 개발자에게 직접적인 영향을 미치는 세 가지를 중심으로 설명한다.

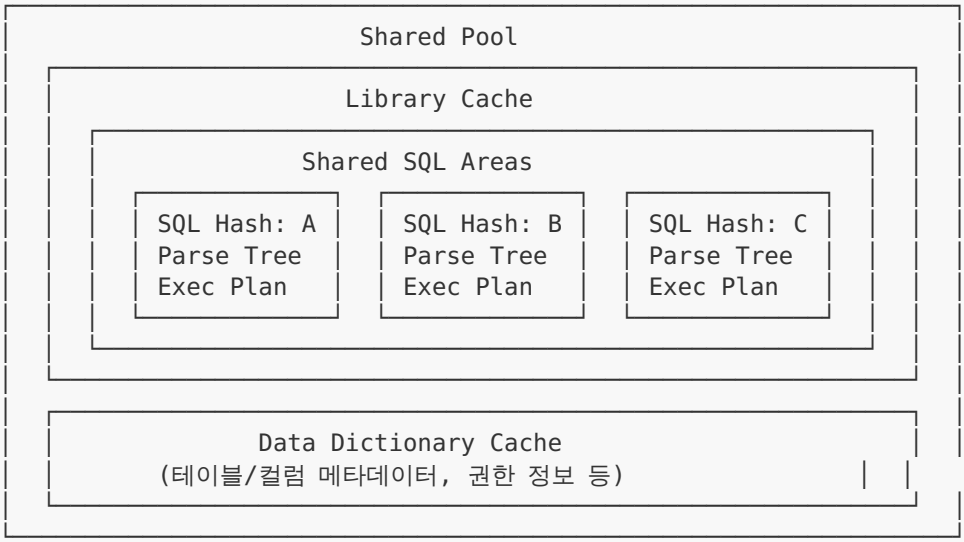
1.2.1 Shared Pool (핵심 - Wherehouse N+1과 직접 연결)

Shared Pool은 **SQL 처리 비용을 결정하는 가장 중요한 SGA 컴포넌트**다. 공식 문서의 정의:

> "The shared pool caches various types of program data. For example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information. The shared pool is involved in almost every operation that occurs in the database."

Shared Pool의 핵심 하위 구조는 **Library Cache**다:

> “The library cache is a shared pool memory structure that stores executable SQL and PL/SQL code. This cache contains the shared SQL and PL/SQL areas and control structures such as locks and library cache handles. When a SQL statement is executed, the database attempts to reuse previously executed code. If a parsed representation of a SQL statement exists in the library cache and can be shared, the database reuses the code. This action is known as a **soft parse** or a **library cache hit**. Otherwise, the database must build a new executable version of the application code, which is known as a **hard parse** or a **library cache miss**.”



Wherehouse 프로젝트와의 연결점:

IN 절의 파라미터 개수가 가변적일 때 성능이 저하되는 원인이 바로 여기에 있다. Oracle은 SQL 문장의 **텍스트 Hash Value**로 동일 여부를 판단한다:

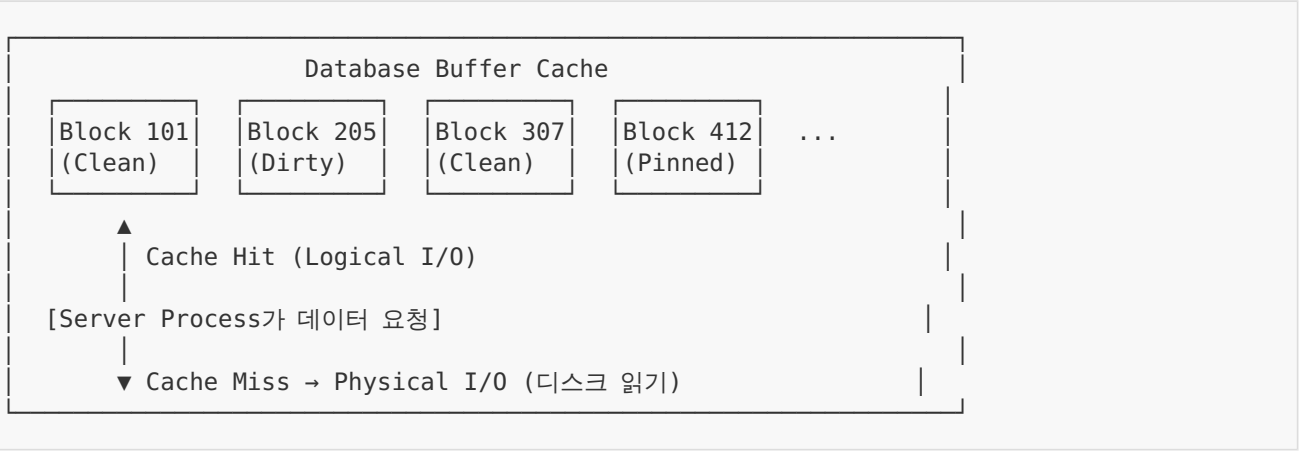
```
-- 이 두 SQL은 다른 Hash Value를 가짐 → 별개의 Shared SQL Area 할당
SELECT * FROM PROPERTY WHERE ID IN (?, ?)          -- Hash: A
SELECT * FROM PROPERTY WHERE ID IN (?, ?, ?)       -- Hash: B
SELECT * FROM PROPERTY WHERE ID IN (?, ?, ?, ?)    -- Hash: C
```

3차 테스트에서 1,000개 단위 Chunking이 효과적이었던 이유는 **SQL 형태를 고정**함으로써 동일한 Shared SQL Area를 재사용(Soft Parse)하도록 유도했기 때문이다.

1.2.2 Database Buffer Cache

> “The database buffer cache, also called the buffer cache, is the memory area that stores copies of data blocks read from data files.”

Buffer Cache는 **디스크 I/O를 최소화**하기 위한 데이터 블록 캐싱 영역이다. 쿼리 실행 시 Oracle은 먼저 Buffer Cache에서 필요한 블록을 검색하고, 없을 때만 디스크에서 읽어온다.



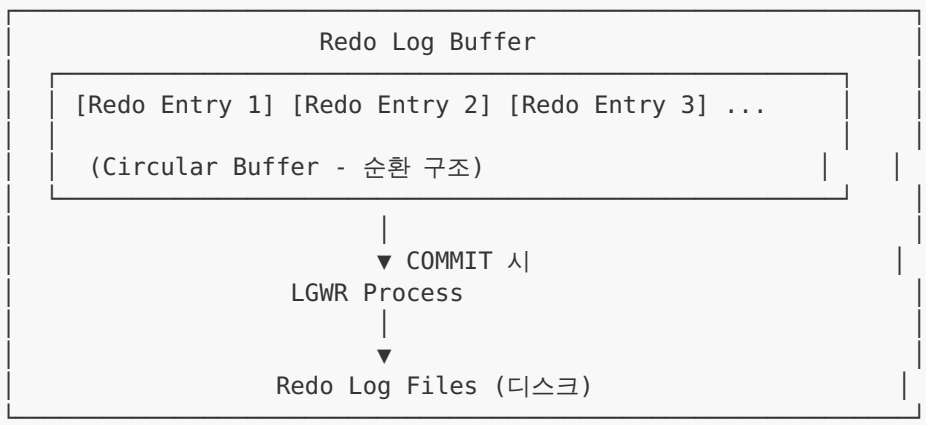
Buffer 상태는 세 가지로 구분된다: - **Unused**: 아직 사용되지 않은 빈 버퍼 - **Clean**: 데이터가 있지만 디스크와 동기화된 상태 (재사용 가능) - **Dirty**: 수정되었지만 아직 디스크에 기록되지 않은 상태 (COMMIT 후에도 즉시 디스크에 쓰이지 않음)

백엔드 관점에서의 의의: `COMMIT` 후에도 데이터가 **즉시 디스크에 기록되지 않는다**는 사실이 중요하다. Oracle은 성능을 위해 Dirty Buffer를 모아뒀다가 DBW(Database Writer) 프로세스가 **비동기적으로** 디스크에 기록한다. 트랜잭션 안전성은 Redo Log를 통해 보장한다.

1.2.3 Redo Log Buffer

> "The redo log buffer is a circular buffer in the SGA that stores redo entries describing changes made to the database."

Redo Log Buffer는 **데이터 변경 사항을 임시 저장**하는 순환 버퍼다. `COMMIT` 시 이 버퍼의 내용이 LGWR(Log Writer) 프로세스에 의해 Redo Log Files로 기록되며, 이것이 Oracle의 장애 복구 메커니즘의 핵심이다.



1.3 So What: 백엔드 개발자 관점에서의 설계적 함의

1.3.1 "같은 Connection에서 같은 SQL을 실행하면 빠르다"의 아키텍처적 근거

이 현상은 SGA의 Shared Pool 구조에서 기인한다:

1. 첫 번째 실행: Library Cache에 SQL이 없음 → **Hard Parse** (파싱 + 실행 계획 수립 + Shared SQL Area 할당)
2. 두 번째 이후 실행: Library Cache에 동일 SQL 존재 → **Soft Parse** (기존 실행 계획 재사용)

Hard Parse는 CPU 집약적 작업이며, 공식 문서에 따르면:

> "A cache miss on the library cache or data dictionary cache is more expensive than a miss on the buffer cache."

즉, **Library Cache Miss의 비용 > Buffer Cache Miss의 비용**이다. 이것이 SQL 형태를 일관되게 유지해야 하는 아키텍처적 이유다.

1.3.2 Wherehouse 테스트 결과 해석

3차 테스트(Chunk 방식)에서 응답 시간이 개선된 원인을 SGA 관점에서 해석하면:

테스트	SQL 형태	Library Cache 동작	영향
1차 (N+1)	동일 SQL 25회 반복	첫 번째만 Hard Parse, 나머지 Soft Parse	Parse 비용은 낮지만 Connection 점유 시간 증가
2차 (Bulk)	IN절 8,660개 파라미터 → OR 9개 분해	매번 다른 SQL로 인식될 가능성	Hard Parse 증가 + 비효율적 실행 계획
3차 (Chunk)	IN절 1,000개 고정 × 9회	SQL 형태 동일 → Soft Parse	Parse 비용 최소화 + 실행 계획 재사용

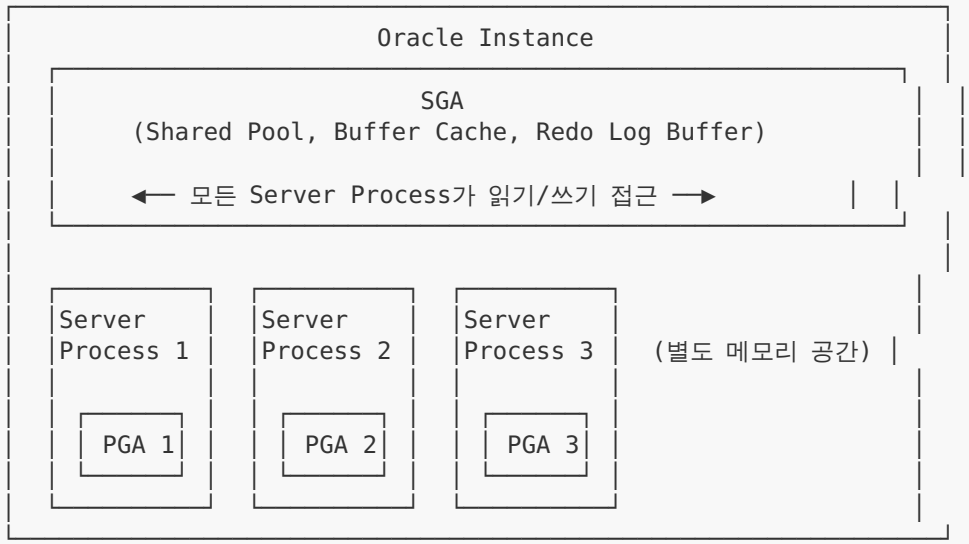
1.3.3 SGA가 "공유 메모리"라는 사실의 설계적 영향

공식 문서의 Note를 주목해야 한다:

> “The server and background processes do not reside within the SGA, but exist in a separate memory space.”

Server Process와 Background Process는 SGA **내부**에 있지 않고, **별도의 메모리 공간**에 존재한다. 이 분리가 의미하는 바는:

- **SGA**: 모든 세션이 **공유**하는 Instance 레벨 메모리
- **PGA**: 각 Server Process가 **독점**하는 프로세스 레벨 메모리



이 구조 때문에 Connection Pool이 효과적이다. Connection을 재사용하면: - Server Process가 이미 존재 (fork 비용 없음) - PGA가 이미 할당됨 (메모리 할당 비용 없음) - SGA의 Library Cache에 SQL이 이미 캐싱되어 있을 가능성 높음 (Soft Parse)

---

2. Server Process vs Background Process 구분

2.1 Oracle Process 유형의 아키텍처적 분류

Oracle에서 “Process”라는 용어는 세 가지 완전히 다른 범주를 가리킨다. 이 구분을 명확히 이해하지 않으면 Connection 생성 비용의 근거를 아키텍처적으로 설명할 수 없다.

프로세스 유형 분류

```
┌──────────────────────────────────────────────────────────────────────┐
│                        Oracle Process 분류                           │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│  1. Client Process (User Process)                                    │
│     └─ 사용자 애플리케이션 측 프로세스                                  │
│        예: Spring Boot App, SQL*Plus, JDBC Driver가 동작하는 프로세스   │
│                                                                      │
│  2. Server Process                                                   │
│     └─ 클라이언트 요청을 처리하기 위해 Oracle이 생성하는 프로세스         │
│        Connection 수립 시 생성됨 (Dedicated Server 모드 기준)          │
│                                                                      │
│  3. Background Process                                               │
│     └─ Instance 운영을 위해 자동으로 존재하는 프로세스                  │
│        Instance 시작 시 생성, 클라이언트와 무관하게 동작                │
│        예: DBW, LGWR, PMON, SMON, CKPT...                            │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

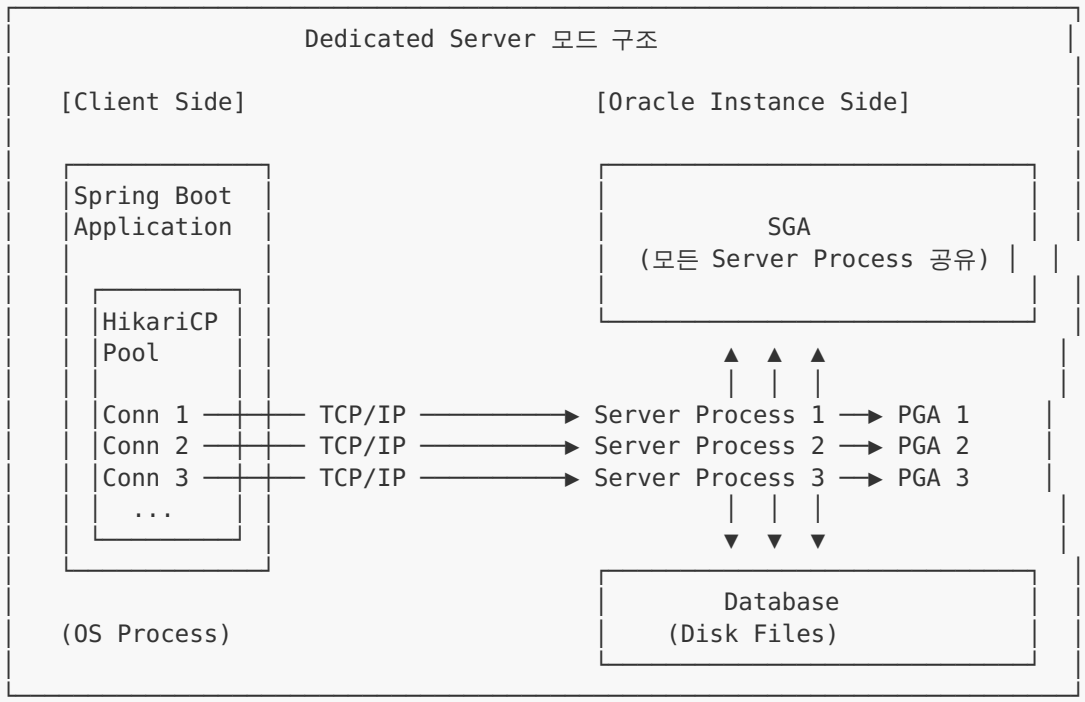
2.2 Server Process의 정체

공식 문서의 정의:

> “When a user runs an application such as a ProC program or SQLPlus, the operating system creates a **client process** (sometimes called a user process) to run the user application.”

> “In **dedicated server connections**, the client connection is associated with one and only one **server process**. On Linux, 20 client processes connected to a database instance are serviced by 20 server processes. Each client process communicates directly with its server process. This server process is **dedicated to its client process for the duration of the session**.”

즉, **Server Process**는: - **클라이언트의 SQL 요청을 실제로 처리하는 Oracle 측 프로세스** - Dedicated Server 모드에서 **Connection 1개 = Server Process 1개** - 클라이언트가 보낸 SQL을 파싱하고, SGA에 접근하여 데이터를 읽고, 결과를 반환하는 역할 - **각 Server Process는 자신만의 PGA를 소유**



2.3 Background Process의 정체

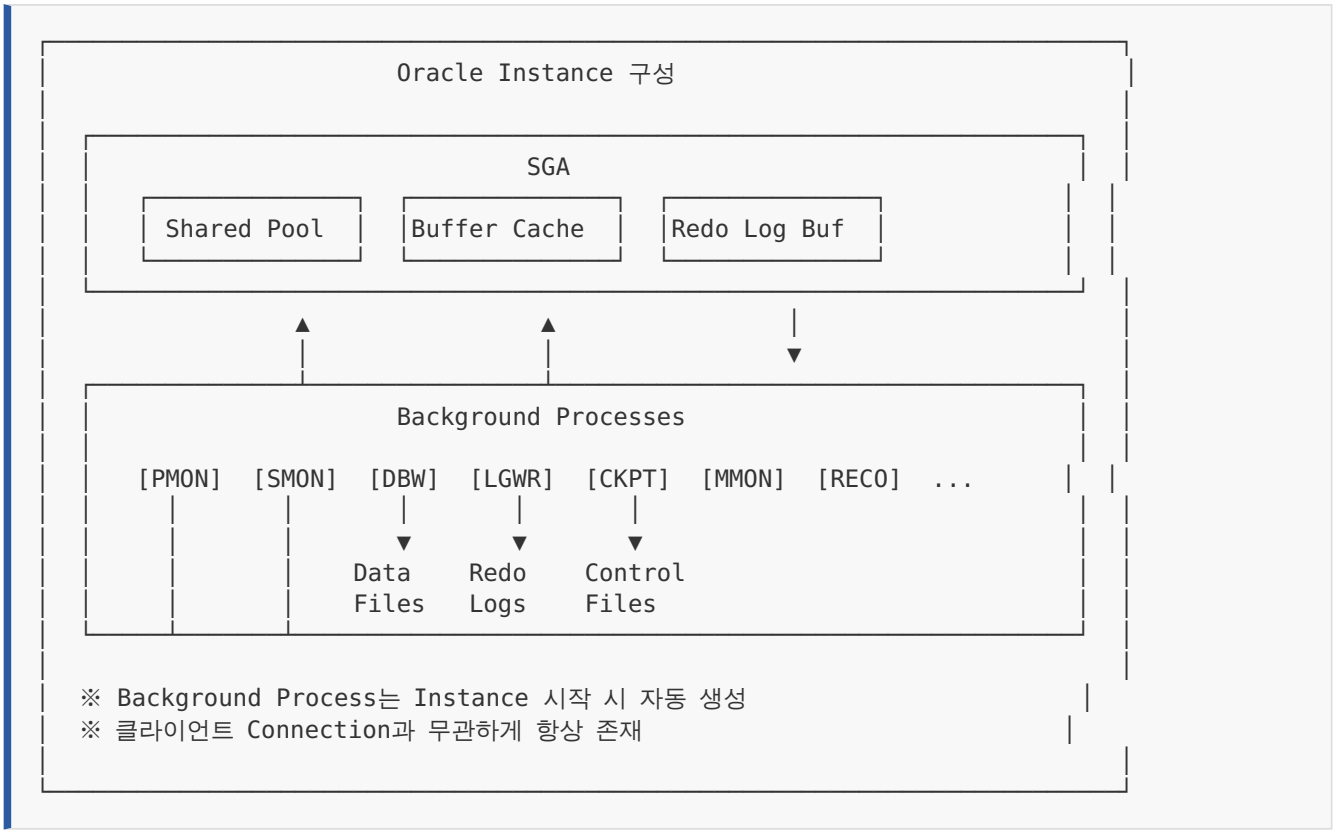
공식 문서의 정의:

> “A multiprocess Oracle Database uses some additional processes called **background processes**. The background processes perform **maintenance tasks required to operate the database** and to maximize performance for multiple users.”

> “Oracle Database creates background processes **automatically when a database instance starts**.”

Background Process는 Instance가 동작하기 위해 **필수적으로 존재해야 하는 프로세스들**이다:

Process	역할	설계적 의의
DBW (Database Writer)	Buffer Cache의 Dirty Buffer를 디스크에 기록	COMMIT 시 즉시 디스크에 쓰지 않고 비동기 기록
LGWR (Log Writer)	Redo Log Buffer를 Redo Log Files에 기록	COMMIT 시 트랜잭션 영속성 보장
PMON (Process Monitor)	비정상 종료된 프로세스 감지 및 정리	자원 누수 방지
SMON (System Monitor)	Instance Recovery, 임시 세그먼트 정리	시스템 레벨 유지보수
CKPT (Checkpoint)	Checkpoint 정보를 Control File에 기록	복구 시작점 관리



2.4 핵심 차이점 정리

구분	Server Process	Background Process
생성 시점	Connection 수립 시	Instance 시작 시
생성 주체	클라이언트 요청에 의해	Oracle이 자동으로
수명	Session 종료 시 사라짐	Instance 종료 시까지 존재
역할	클라이언트 SQL 요청 처리	Instance 운영/유지보수
PGA 소유	각자 독립적 PGA 보유	각자 독립적 PGA 보유
개수	Connection 수에 비례	Instance당 고정 (필수 프로세스 기준)

2.5 Connection 생성 비용의 아키텍처적 근거

이 구분이 중요한 이유는 **Connection 생성 비용**을 설명할 수 있기 때문이다:

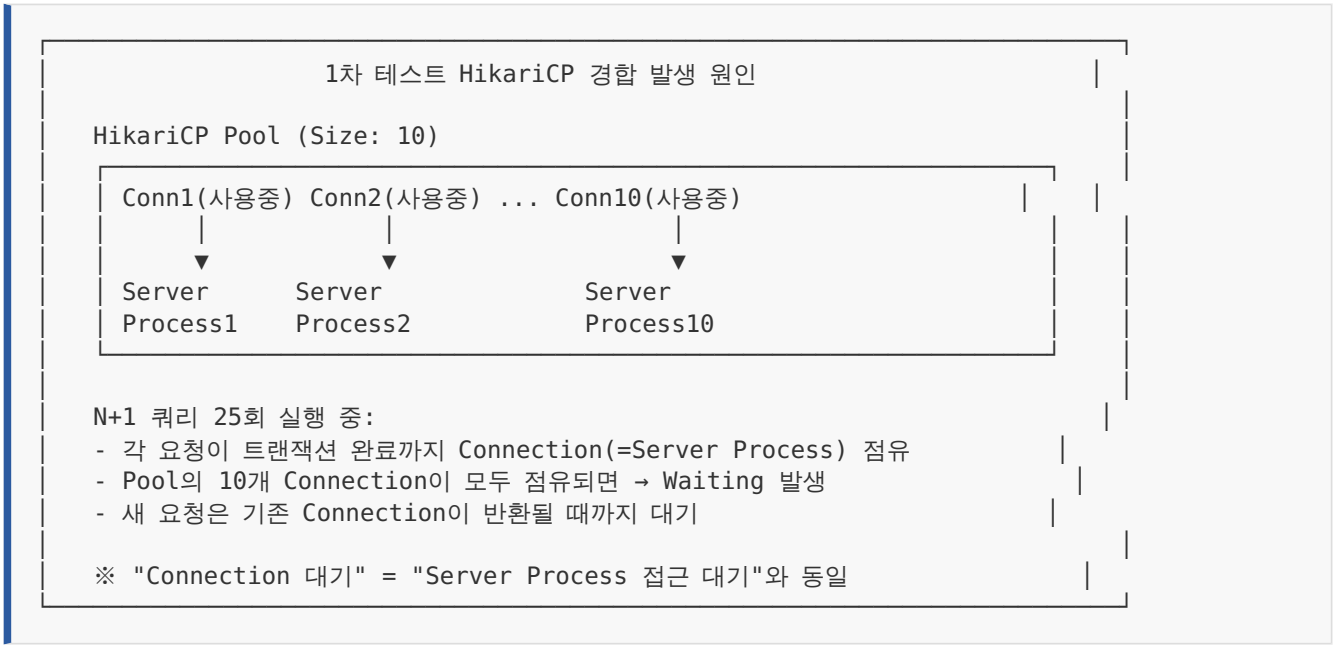
```
┌──────────────────────────────────────────────────────────────┐
│              Connection 생성 시 발생하는 작업 (Dedicated Server)          │
│                                                                  │
│  1. TCP 3-Way Handshake (네트워크)                                 │
│                    ↓                                             │
│  2. Oracle Net Listener가 요청 수신 (1521 포트)                     │
│                    ↓                                             │
│  3. Listener가 Dedicated Server Process 생성 요청                  │
│                    ↓                                             │
│  4. OS 레벨에서 Server Process fork() ◀── 무거운 작업               │
│                    ↓                                             │
│  5. Server Process에 PGA 메모리 할당 ◀── 무거운 작업                │
│                    ↓                                             │
│  6. Oracle 인증 및 세션 수립                                       │
│                    ↓                                             │
│  7. Connection 반환                                               │
│                                                                  │
│  총 소요 시간: 50~200ms                                            │
└──────────────────────────────────────────────────────────────┘
```

Connection Pool(HikariCP)이 효과적인 이유:

Pool에 Connection을 유지한다는 것은 곧 **Server Process와 PGA가 이미 할당된 상태를 유지**한다는 의미다. Connection을 재사용하면 위 1~7단계 중 대부분을 건너뛰고 바로 SQL을 실행할 수 있다.

2.6 Wherehouse 테스트 결과와의 연결

1차 테스트에서 HikariCP Waiting이 최대 9건까지 발생한 것은:



이것이 RDB 시간 비율이 6.2%밖에 안 되는데 전체 응답 시간이 5초를 넘은 이유다. 실제 DB 작업(Server Process가 SGA에 접근하여 쿼리 실행)은 빨랐지만, Server Process를 점유한 Connection을 획득하기 위한 **대기 시간이 누적**되었다.

---

3. Library Cache Miss vs Buffer Cache Miss: 비용 비교의 아키텍처적 근거

3.1 공식 문서의 원문 확인

Oracle Performance Tuning Guide의 원문:

> “A cache miss on the **library cache or data dictionary cache** is **more expensive** than a miss on the **buffer cache**. For this reason, the shared pool should be sized to ensure that frequently used data is cached.”

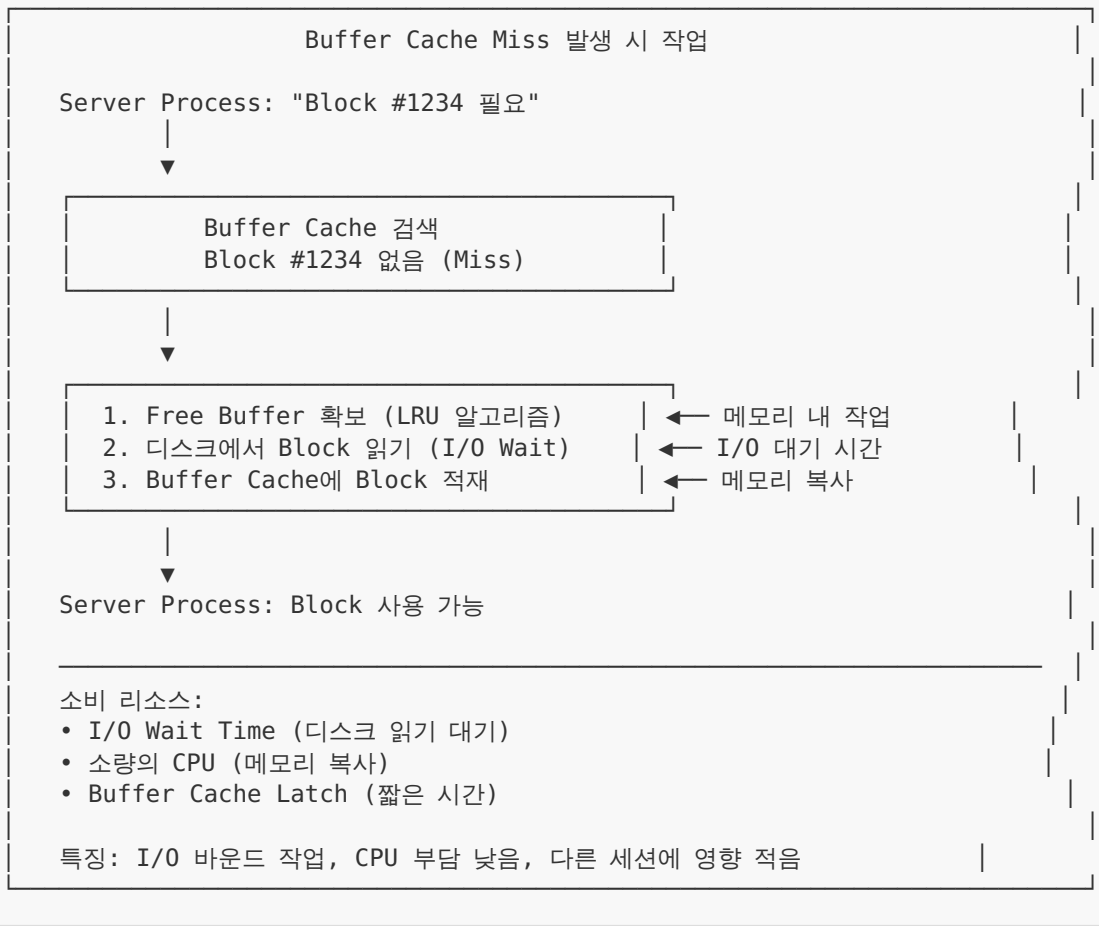
그리고 Hard Parse와 Soft Parse의 리소스 차이:

> “In order to perform a **hard parse**, Oracle Database uses **more resources** than during a soft parse. Resources used for a **soft parse** include CPU and library cache latch gets. Resources required for a **hard parse** include **additional CPU, library cache latch gets, and shared pool latch gets**.”

3.2 각 Cache Miss 발생 시 수행되는 작업 비교

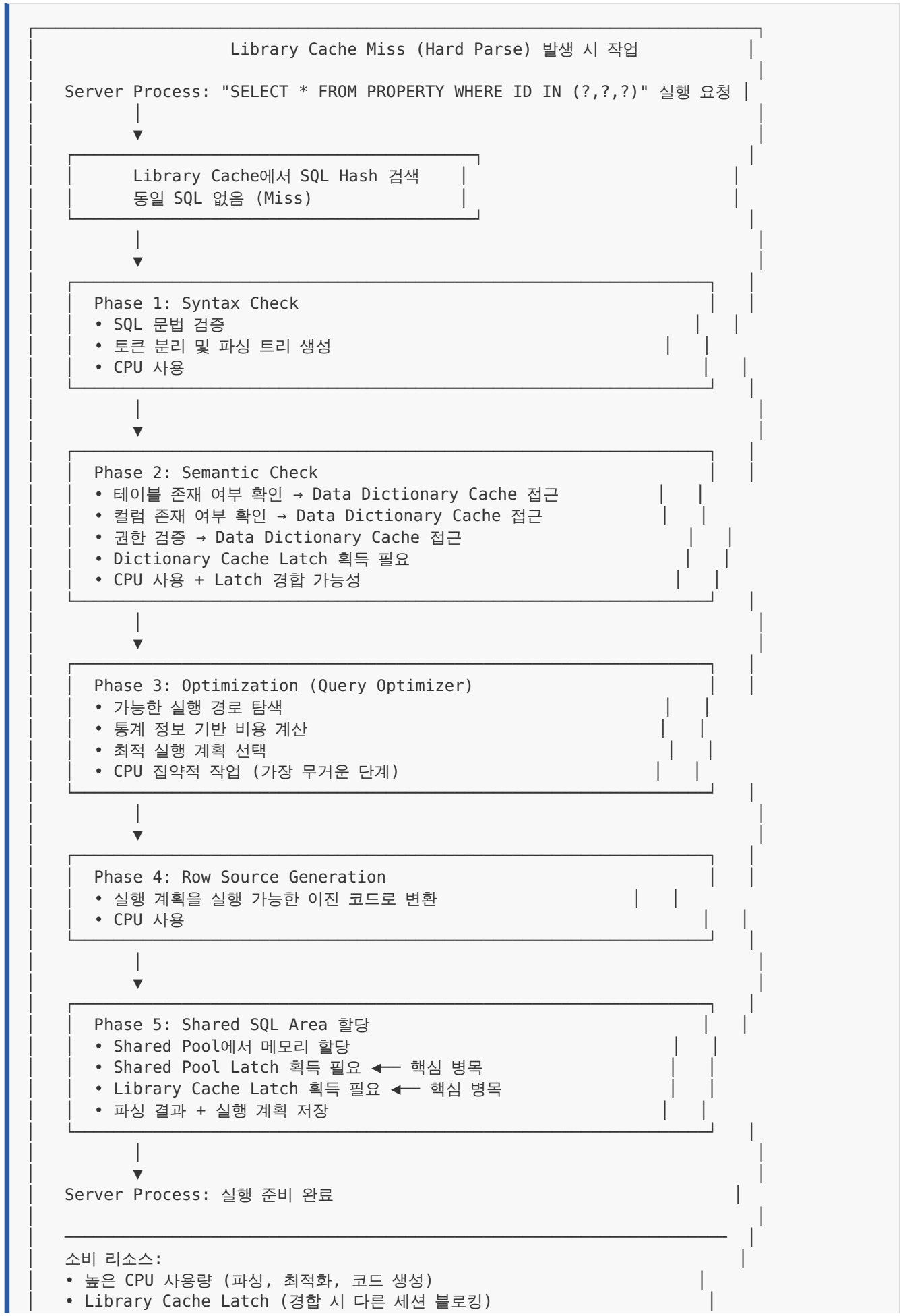
3.2.1 Buffer Cache Miss (Physical I/O)

Buffer Cache Miss가 발생하면 **디스크에서 데이터 블록을 읽어오는 작업**이 수행된다:



3.2.2 Library Cache Miss (Hard Parse)

Library Cache Miss가 발생하면 **SQL 문장의 전체 파싱 + 최적화 + 실행 계획 수립**이 수행된다:


Library Cache Miss (Hard Parse) 발생 시 작업
Server Process: "SELECT * FROM PROPERTY WHERE ID IN (?,?,?)" 실행 요청
Library Cache에서 SQL Hash 검색
동일 SQL 없음 (Miss)
Phase 1: Syntax Check
• SQL 문법 검증
• 토큰 분리 및 파싱 트리 생성
• CPU 사용
Phase 2: Semantic Check
• 테이블 존재 여부 확인 → Data Dictionary Cache 접근
• 컬럼 존재 여부 확인 → Data Dictionary Cache 접근
• 권한 검증 → Data Dictionary Cache 접근
• Dictionary Cache Latch 획득 필요
• CPU 사용 + Latch 경합 가능성
Phase 3: Optimization (Query Optimizer)
• 가능한 실행 경로 탐색
• 통계 정보 기반 비용 계산
• 최적 실행 계획 선택
• CPU 집약적 작업 (가장 무거운 단계)
Phase 4: Row Source Generation
• 실행 계획을 실행 가능한 이진 코드로 변환
• CPU 사용
Phase 5: Shared SQL Area 할당
• Shared Pool에서 메모리 할당
• Shared Pool Latch 획득 필요 ◀── 핵심 병목
• Library Cache Latch 획득 필요 ◀── 핵심 병목
• 파싱 결과 + 실행 계획 저장
Server Process: 실행 준비 완료
소비 리소스:
• 높은 CPU 사용량 (파싱, 최적화, 코드 생성)
• Library Cache Latch (경합 시 다른 세션 블로킹)

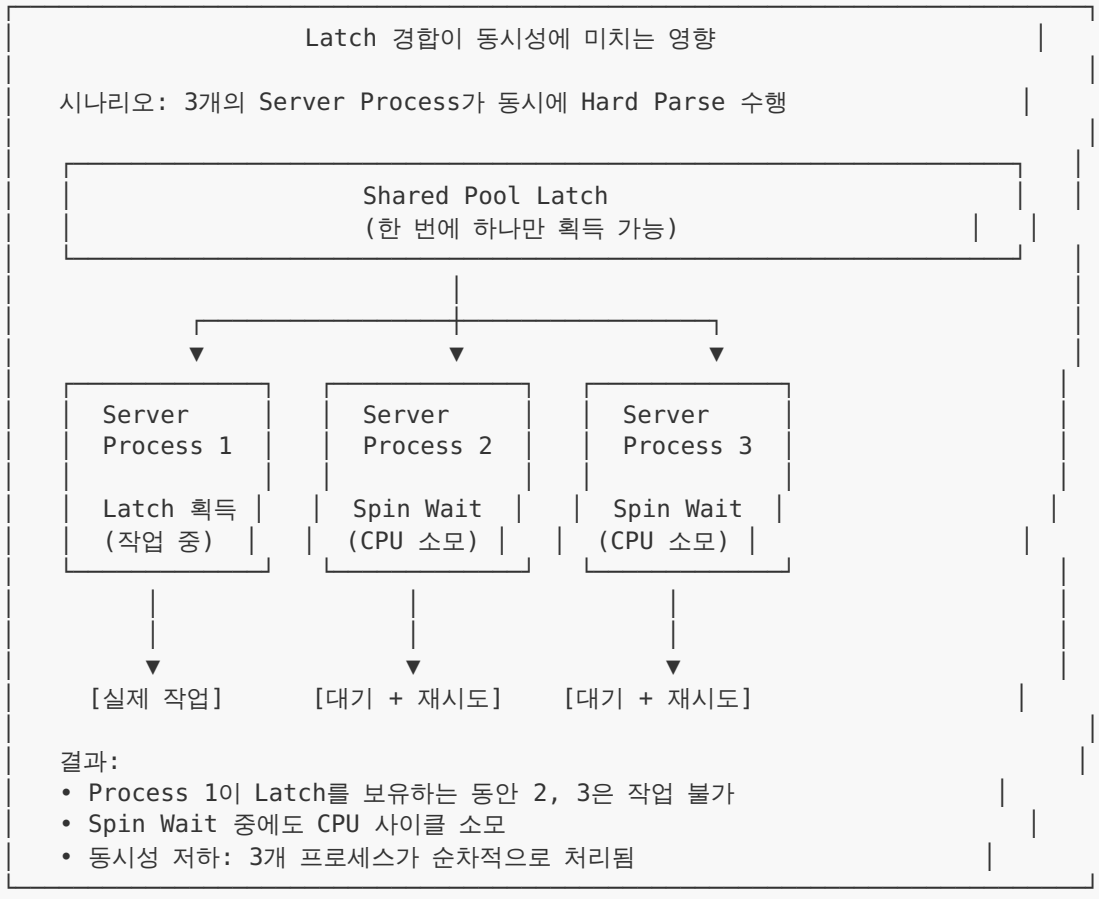
```
│   • Shared Pool Latch (경합 시 다른 세션 블로킹)                          │
│   • Dictionary Cache Latch                                              │
│   • Shared Pool 메모리 할당                                             │
│                                                                         │
│   특징: CPU 바운드 작업, Latch 경합으로 동시성 저하                     │
└─────────────────────────────────────────────────────────────────────────┘
```

3.3 핵심 차이: Latch 경합의 영향

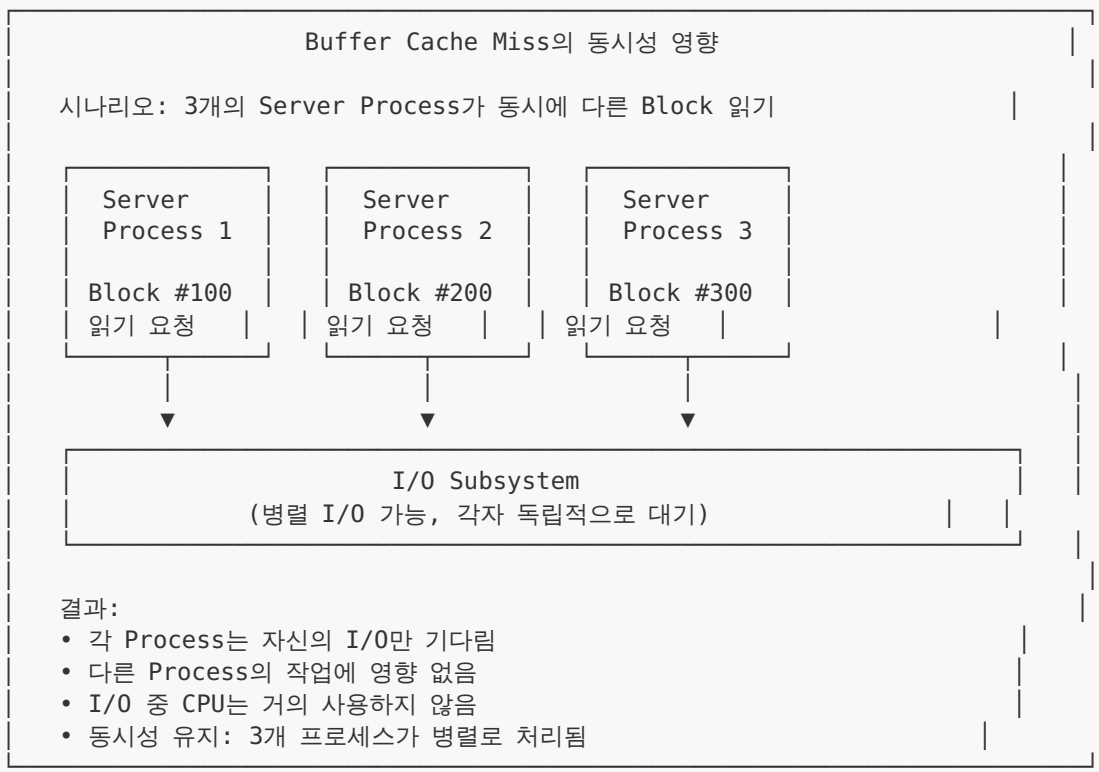
공식 문서의 핵심 구절:

> "During the hard parse, the database accesses the library cache and data dictionary cache **numerous times** to check the data dictionary. When the database accesses these areas, it uses a **serialization device called a latch** on required objects so that their definition does not change. **Latch contention increases statement execution time and decreases concurrency.**"

Latch는 Oracle 내부에서 **공유 메모리 구조를 보호하기 위한 저수준 잠금 메커니즘**이다. 여기서 "serialization device"라는 표현이 중요하다. **직렬화**를 강제한다는 의미다.



반면 **Buffer Cache Miss**의 경우:

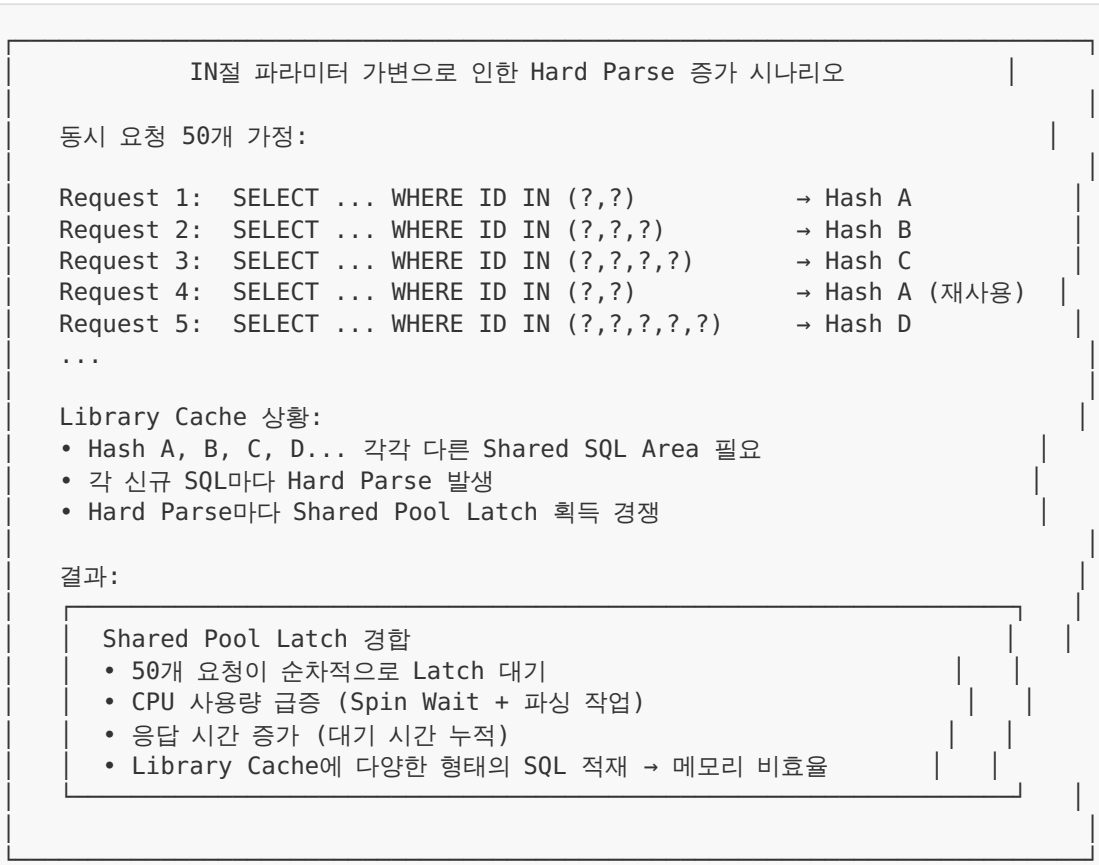


3.4 비용 비교 요약

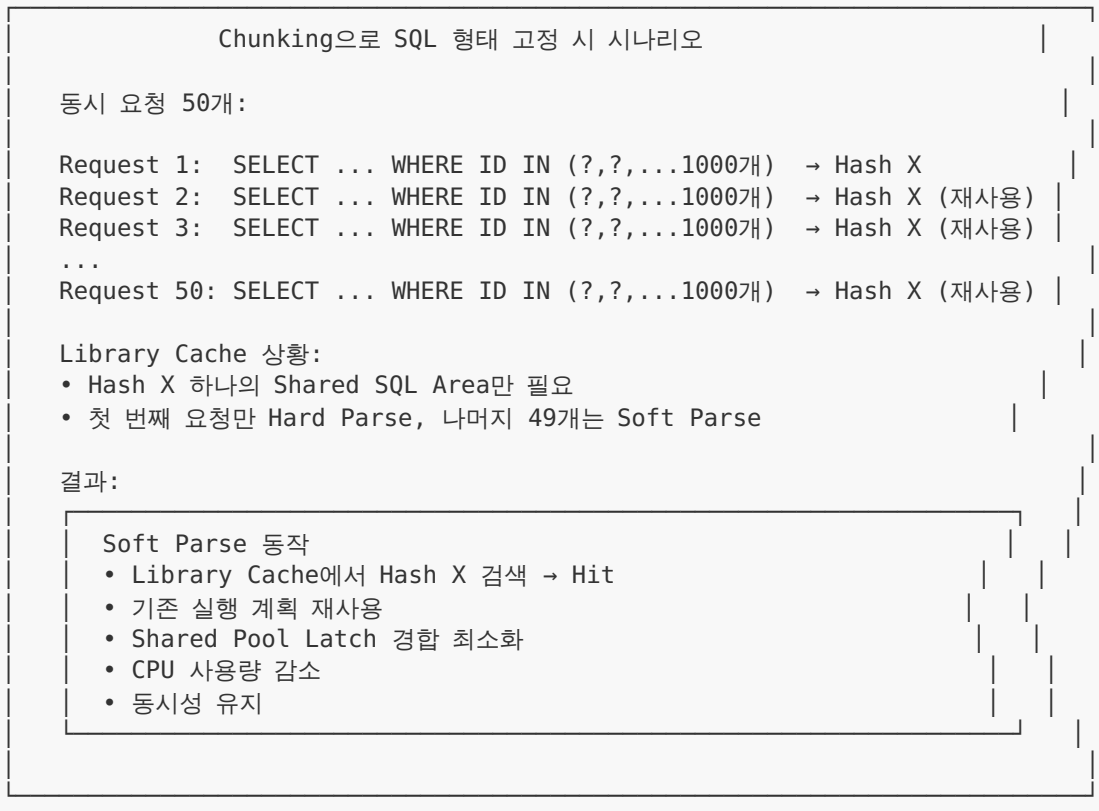
측면	Library Cache Miss (Hard Parse)	Buffer Cache Miss (Physical I/O)
CPU 사용량	높음 (파싱, 최적화, 코드 생성)	낮음 (메모리 복사 정도)
대기 유형	Latch Spin Wait (CPU 소모)	I/O Wait (CPU 거의 미사용)
동시성 영향	**심각** (Latch 경합 → 직렬화)	낮음 (각자 독립적 I/O)
다른 세션 영향	**있음** (Latch 보유 중 블로킹)	거의 없음
확장성	**낮음** (사용자 증가 시 경합 증가)	상대적으로 높음
해결 방법	SQL 재사용 (Soft Parse 유도)	메모리 증설, I/O 최적화

3.5 Wherehouse 프로젝트에서의 의미

IN 절 파라미터 개수가 가변적일 때 발생하는 문제를 이 관점에서 재해석하면:



1,000개 단위 Chunking의 효과:



3.6 결론: 왜 “Library Cache Miss > Buffer Cache Miss”인가

Buffer Cache Miss는 **I/O 대기 시간**이 주된 비용이다. 느리지만 CPU를 거의 사용하지 않고, 다른 세션의 작업을 방해하지 않는다.

Library Cache Miss는 **CPU 집약적 작업 + Latch 경합**이 주된 비용이다. 파싱과 최적화에 CPU를 많이 사용하면서, 동시에 Latch를 획득하기 위해 다른 세션과 경쟁해야 한다. 이 경합은 시스템 전체의 동시성을 저하시킨다.

따라서 **동시 사용자가 많은 환경에서 Hard Parse가 빈번하면 시스템 전체 처리량이 급격히 저하**된다. 이것이 SQL 형태를 일관되게 유지해야 하는 아키텍처적 이유다.

---

4. Latch 학습 범위 판단

4.1 학습 범위 판단 기준

학습 로드맵에서 정의한 수준을 다시 확인하면:

Level	대상	알아야 하는 것
Level 2	**백엔드 개발자 (목표)**	아키텍처 구성요소, SQL 처리 흐름, **성능 영향 요소**
Level 3	DBA	튜닝, 백업/복구, 인스턴스 관리, AWR 분석
Level 4	Oracle 내부 개발자	소스 코드 레벨 이해

Latch의 내부 알고리즘(spin lock, sleep, mutex 구현 등)은 **Level 3~4 영역**이다.

4.2 백엔드 개발자로서 알아야 할 것 vs 알 필요 없는 것

알아야 할 것 (Level 2)

“Hard Parse가 왜 비싼가”를 설명할 수 있는 수준:

```
Hard Parse 발생
    ↓
Shared Pool/Library Cache 접근 시 Latch 필요
    ↓
Latch는 한 번에 하나의 프로세스만 획득 가능
    ↓
동시에 Hard Parse가 많으면 경합 발생
    ↓
동시성 저하, 응답 시간 증가
```

면접에서 이 정도 인과관계를 설명할 수 있으면 충분하다.

알 필요 없는 것 (Level 3~4)

- Latch의 spin lock 알고리즘 (spin count, willing-to-wait 등)
- VLATCH, VLATCH_CHILDREN 뷰 분석
- `_spin_count` 히든 파라미터 튜닝
- Latch vs Mutex vs Enqueue Lock의 내부 구현 차이
- Latch-free 알고리즘 (CAS 연산 등)

4.3 면접 대응 관점

예상 질문: “Hard Parse가 왜 성능에 영향을 주나요?”

적절한 답변 수준: > “Hard Parse 시 Oracle은 Shared Pool에 새로운 SQL 영역을 할당해야 합니다. 이 과정에서 공유 메모리 보호를 위한 Latch라는 내부 잠금을 획득해야 하는데, 동시에 많은 세션이 Hard Parse를 수행하면 Latch 경합이 발생하여 동시성이 저하됩니다. 제 프로젝트에서 IN절 파라미터 개수를 1,000개로 고정한 이유가 SQL 형태를 일관되게 유지하여 Soft Parse를 유도하고, 이런 경합을 줄이기 위해서였습니다.”

과도한 답변 (DBA 영역 침범): > “Latch는 spin lock 기반으로 동작하며, 먼저 CPU에서 spin하다가 일정 횟수 실패하면 sleep합니다. V$LATCH에서 gets, misses, sleeps를 확인하여…”

후자는 백엔드 개발자 면접에서 기대하는 답변이 아니다. 오히려 “본인 역할의 경계를 모른다”는 인상을 줄 수 있다.

4.4 결론

Latch라는 용어와 **“경합이 발생하면 동시성이 저하된다”**는 인과관계만 이해하면 된다. 내부 알고리즘은 학습 대상이 아니다.

부록: 참고 자료

공식 문서 URL

- **Memory Architecture**: https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/memory-architecture.html
- **Process Architecture**: https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/process-architecture.html
- **SQL Tuning Guide**: https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/sql-processing.html

Wherehouse 프로젝트 테스트 결과 요약

테스트	총 소요 시간	RDB 조회 시간	RDB 시간 비율	HikariCP Waiting
1차 (N+1)	5,531ms	343ms	6.2%	최대 9건
2차 (Bulk)	6,001ms	1,748ms	28.8%	0건
3차 (Chunk)	4,434ms	548ms	12.1%	최대 19건