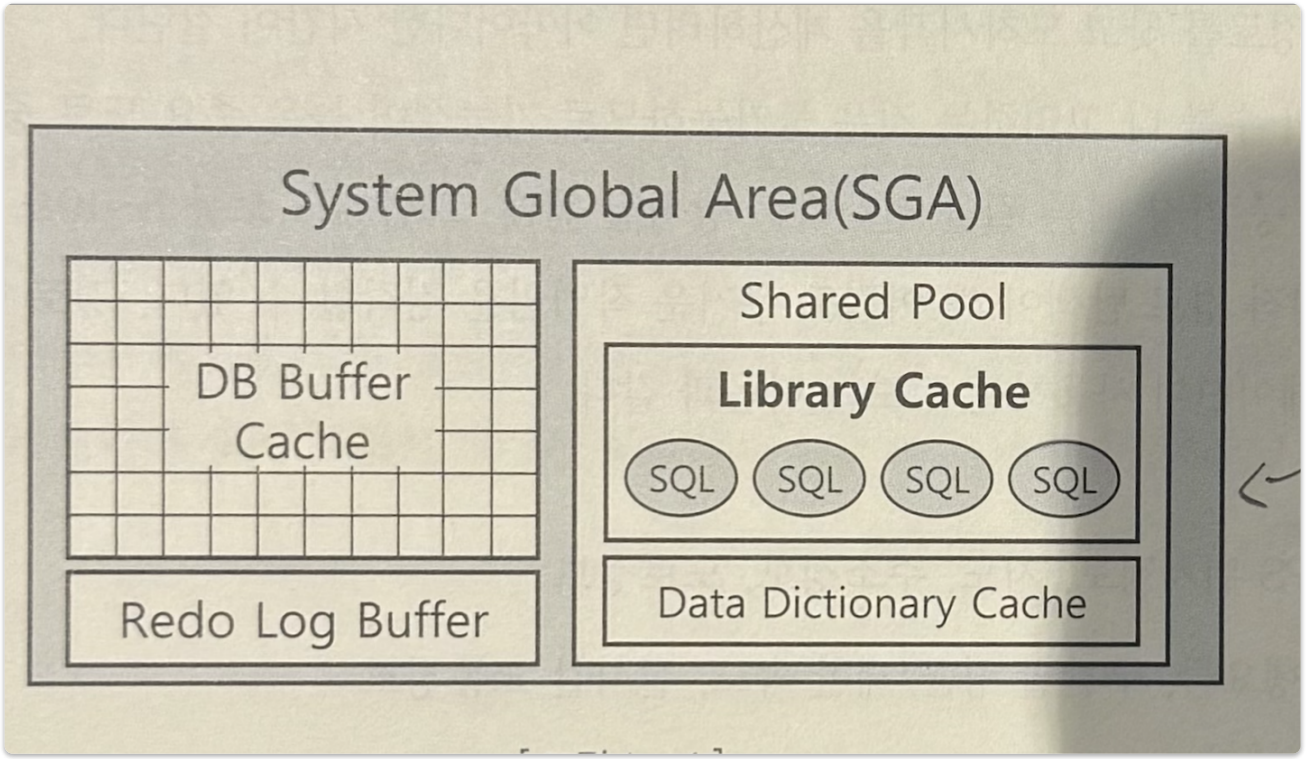
# 1.2 SQL 공유 및 재사용

> *SQL을 짜다보면 우리가 짜는 코드와 크게 다른 점이 있다. 바로 각 쿼리에 이름이 없다는 점이다. 우리가 작성하는 메서드들은 이름을 가지고 있고, 파라미터를 달리하여 재사용할 수 있게 되어있다. 그런데 일반적으로 작성하는 SQL은 이름이 존재하지 않고, 매번 실행할 때마다 문법이 바뀐다거나 변수 이름을 달리 쓸 수 있다.*
> *이런 경우에 이 SQL은 매번 새롭게 생성되는 것일까?*

## 💡 1.2.1 Sort Parsing vs Hard Parsing

- 라이브러리 캐시 (Library Cache)
  - SQL 파싱, 최적화, 로우 소스 생성 과정을 거쳐 생성한 내부 프로시저를 반복 재사용할 수 있도록 캐싱해두는 메모리 공간
  - SGA(System Global Area) 구성 요소
  - 프로시저 (Procedure) : 자주 쓰는 SQL작업을 미리 저장해두고, 필요할 때 마다 불러올 수 있는 명령어 묶음
- SGA (System Global Area)
  - 서버 프로세스와 백그라운드 프로세스가 공통으로 액세스하는 데이터와 제어 구조를 캐싱하는 메모리 공간



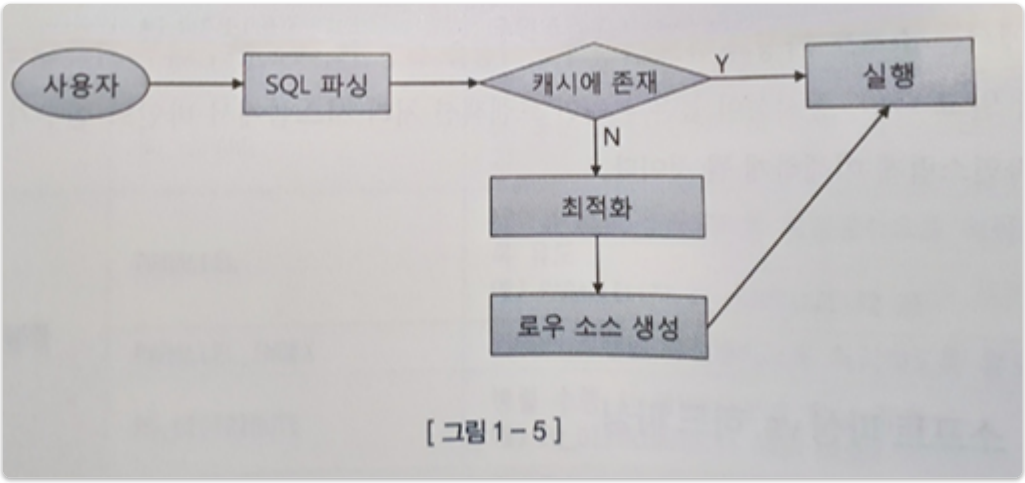

- 사용자가 SQL 문을 전달하면 DBMS는 SQL을 파싱한 후 해당 SQL이 라이브러리 캐시에 존재하는지 확인.

### 1. Soft Parsing

- SQL을 캐시에서 찾아 곧바로 실행단계로 넘어가는 것

### 2. Hard Parsing

- SQL을 캐시에서 찾지 못하여 최적화 및 로우 소스 생성 단계를 모두 거치는 것




- 왜 SQL 최적화 과정을 hard 라고 표현하는가 ?
  - 최적화 시 고려해야 할 부분이 굉장히 많다.
  - 조인 순서만 120(5!) 이다.
  - full scan 할지, index scan 할지, index scan 에도 종류가 굉장히 많다.
  - 그 외에도 테이블, 컬럼, 인덱스 구조에 관한 기본 정보
  - 오브젝트 통계 (테이블, 인덱스, 컬럼 통계)
  - 시스템 통계 (CPU 속도, Single Block I/O 속도, Multi Block I/O 속도 등)
  - 옵티마이저 관련 파라미터

> *데이터 베이스 처리 과정은 대부분 I/O 작업에 집중한다.*
> *하드 파싱은 CPU 를 많이 소비하는 작업이다.*
> *따라서 hard 과정을 거쳐 생성한 내부 프로시저를 한 번만 하고 버리면 낭비이므로 라이브러리 캐시가 필요하다.*

## 💡 1.2.2 바인드 변수의 중요성

### 1. SQL은 이름이 없다.

- SQL ID는 SQL 전체 텍스트를 간략히 표현하려고 오라클이 내부 함수를 이용해 생성한 값
- SQL ID 는 SQL 전체 텍스트와 1:1 대응 관계
- 작은 부분이라도 수정되면 다른 객체로 생성된다.
- SQL이 많아지면 저장공간도 많고 찾는 속도도 느리므로 영구 저장하지 않는다. (Oracle)

### 2. 공유 가능 SQL

```
SELECT * FROM emp
select * FROM emp
SELECT * from emp
```

- 라이브러리 캐시에서 SQL을 찾기 위해 사용하는 키 값이 'SQL문 그 자체'이므로 위는 모두 다른 SQL이다.
- 의미적으로는 모두 같지만, 실행 시 각각 최적화를 진행하고 라이브러리 캐시에서 별도 공간 활용
- 500만 고객을 보유한 어떤 쇼핑몰에서 로그인 모듈 담당 개발자가 프로그램을 아래와 같이 작성했다.

```
public void login(String login_id) throws Exception {
    String SQLStmt = "SELECT * FROM CUSTOMER WHERE LOGIN_ID = '" + login_id
+ "'";
    Statemaent st = con.createStatement();
    ResultSet rs = st.executeQuery(SQLStmt);
    if (rs.next()) {
        // do anything
    }
    rs.close();
    st.close();
}
```

- 동시다발적으로 100만명이 시스템 접속을 할 경우 어떻게 될까 ?

> *DBMS에 발생하는 부하는 대개 과도한 I/O가 원인인데, 이 때는 I/O가 거의 발생하지 않음에도 불구하고 CPU 사용률은 급격히 올라가고, 라이브러리 캐시에 발생하는 여러 종류의 경합 때문에 로그인이 제대로 처리되지 않을 것이다.*
> *각 고객에 대해 동시다발적으로 발생하는 SQL 하드파싱 때문이다.*

- 이 때 라이브러리 캐시를 조회하면 다음과 같다.

```
SELECT * FROM CUSTOMER WHERE LOGIN_ID = '"+login_id+"'
------------------------------------------------------------
-- 라이브러리 캐시 --
SELECT * FROM CUSTOMER WHERE LOGIN_ID = 'oraking';
SELECT * FROM CUSTOMER WHERE LOGIN_ID = 'javaking';
------------------------------------------------------------
-- 로그인 할 때마다 하나씩 프로시저를 거쳐 라이브러리 캐시에 적재 --
create procedure LOGIN_ORAKING() { … }
create procedure LOGIN_JAVAKING() { … }
```

- 위 프로시저의 내부 처리 루틴은 모두 같음.

- 그렇다면 프로시저를 여러 개 생성할 것이 아니라 아래처럼 로그인 ID를 파라미터로 받는 프로시저 하나를 공유하면서 재사용하는것이 마땅하다.

```
create procedure LOGIN (login_id in varchar2) { ··· }
```

```
public void login(String login_id) throws Exception {
    String SQLStmt = "SELECT * FROM CUSTOMER WHERE LOGIN_ID = ?";
    Statemaent st = con.createStatement(SQLStmt);
    st.setString(1, login_id);
    ResultSet rs = st.executeQuery();
    if (rs.next()) {
        // do anything
    }
    rs.close();
    st.close();
}
```

- 바인드 변수를 통해 SQL에 대한 하드파싱은 최초 1번만 이루어지게 해야한다.

```
SELECT * FROM CUSTOMER WHERE LOGIN_ID = :1
```