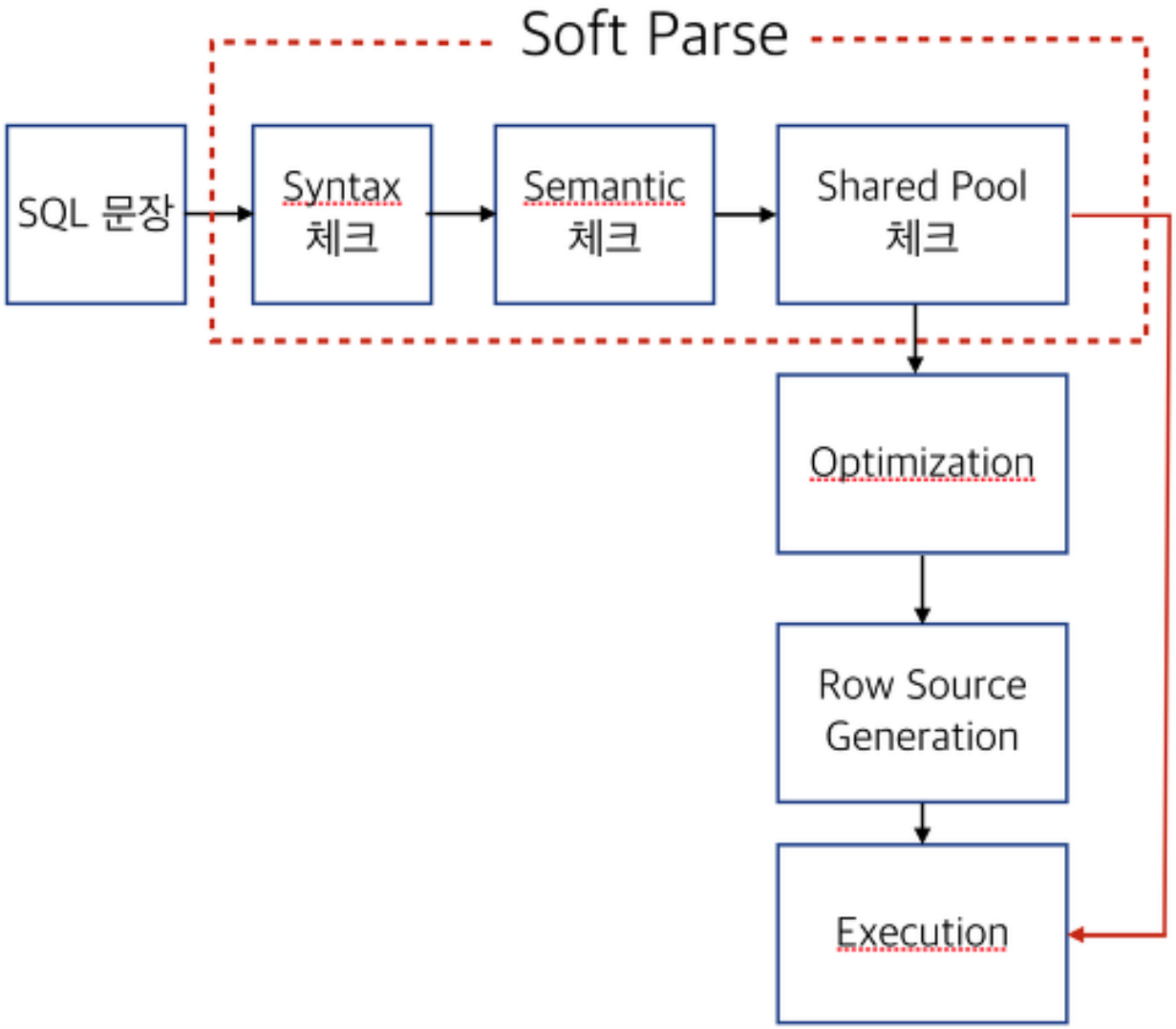
Part 2 . **SQL 문장의 실행 원리**

2. SQL 문장의 실행 원리



2. SQL 문장의 실행 원리

1. 사용자 문장 실행 시 User Process에서 Server Process로 실행한 SQL문 전달

2. User Process로부터 문장을 받은 Server Process가 해당 문장의 세부적 체크 진행
   - Syntax Check : SQL문이 적절한 문법을 사용했는지 검사
   - Semantic Check : SQL문에 포함된 오브젝트들이 실제로 존재하는지 검사

2. SQL 문장의 실행 원리

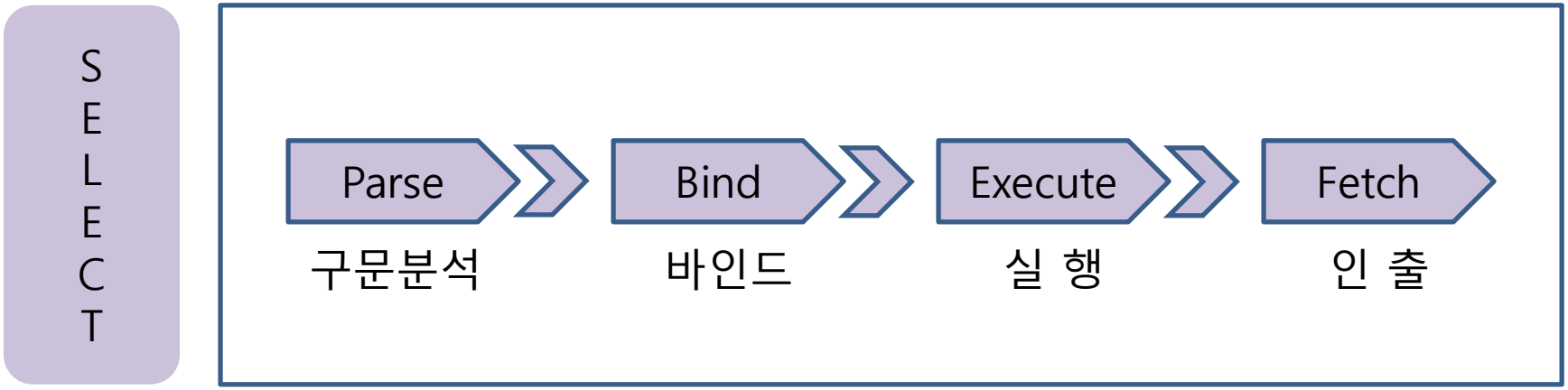
3. Parse과정 후 Shared Pool의 Library Cache에서 공유되어있는 실행계획이 있는지 체크

4. 실행계획이 있을 경우 Execution 진행
   (Soft Parsing)

5. 실행계획이 없을 경우 Optimizer를 통해 Data dictionary 등을 참조하여 실행계획을 새로 생성 후 Library Cache에 저장(Hard Parsing)

2. SQL 문장의 실행 원리

- Select 문장의 실행 원리



2. SQL 문장의 실행 원리

1) Parse(구문분석)
   - User Process로부터 전달받은 SQL문장을 Server프로세스가 SQL문을 수행하기 위해 Parse Tree생성.
   - Parse Tree를 만드는 과정에서 Syntax Check, Semantic Check 진행
   - 데이터 딕셔너리 조회를 통해 문법이 맞는지, 해당 테이블이 있는지 확인.

2. SQL 문장의 실행 원리

1) Parse(구문분석)
   - 자주 사용되는 데이터 딕셔너리는 Shared Pool의 Dictionary Cache에 캐싱해 두어 성능을 높힘

   - 오류가 없을 경우 SQL문장을 Hash 함수로 Hash Value로 변경한 후 Shared Pool의 Library cache에서 Hash Value와 비교하여 동일한 값이 있는지 확인
     (**커서 공유** 혹은 Soft Parsing이라고 함)

2. SQL 문장의 실행 원리

- Cursor : 메모리에 데이터를 저장하기 위해 만든 임시 저장공간
  (공유 커서, 세션커서, 어플리케이션 커서)

- Library Cache안에 있는 커서는 공유 커서를 의미

- 공유커서
  - 한번 수행된 SQL 문장의 실행계획과 관련 정보를 보관
  - 재활용을 통해 Hard Parse의 부담을 줄여 SQL 문장의 수행속도를 빠르게 함

2. SQL 문장의 실행 원리

- 옵티마이저 : SQL의 실행 계획을 생성해주는 네비게이션 역할을 하는 것

- 옵티마이저 모드 : RBO, CBO

  1) RBO : Rule Based Optimizer(11g 부터 사용 불가)

  2) CBO : Cost Based Optimizer
     - 데이터 딕셔너리 정보를 이용하여 판단.
     - 옵티마이저가 참조하는 데이터 딕셔너리 중 대부분은 Static Dictionary, 즉 항상 최신의 정보를 가지고 있지 않음
     - 데이터 딕셔너리를 관리해야 함(통계정보 생성 및 관리)

2. SQL 문장의 실행 원리

2) BIND(바인드)

- SQL문이 정확히 일치해야 Soft Parsing이 가능해짐

- 변수 값이 달라지는 동일 쿼리에 대해 모두 다른 SQL로 인식
  -> Hard Parsing 유발

- 변수 값이 달라지는 부분에 바인드 변수 처리

  ex) select .....
  from emp
  where empno = :emp_num;

- 바인드 처리하면 모든 SQL이 동일 실행계획을 가짐

2. SQL 문장의 실행 원리

3) Execute(실행)
   - Parse와 Bind 단계 후 해당 데이터를 가져오기 위해 데이터가 저장되어있는 데이터 블록을 찾아 DB Buffer Cache에 복사하는 과정
   - 사용자가 찾는 모든 데이터는 SGA의 DB Buffer Cache에 있어야 함
   - 사용자가 찾거나 변경하는 모든 작업은 DB Buffer Cache에서 작업이 수행됨
   - 서버프로세스는 해당블록을 찾기 위해 DB Buffer Cache를 확인 후 없는 경우 데이터 파일로부터 복사해옴
   - 데이터 파일과 DB Buffer Cache의 데이터 이동은 BLOCK 단위
   - DB_BLOCK_SIZE 크기만큼 데이터를 이동(default : 8K)

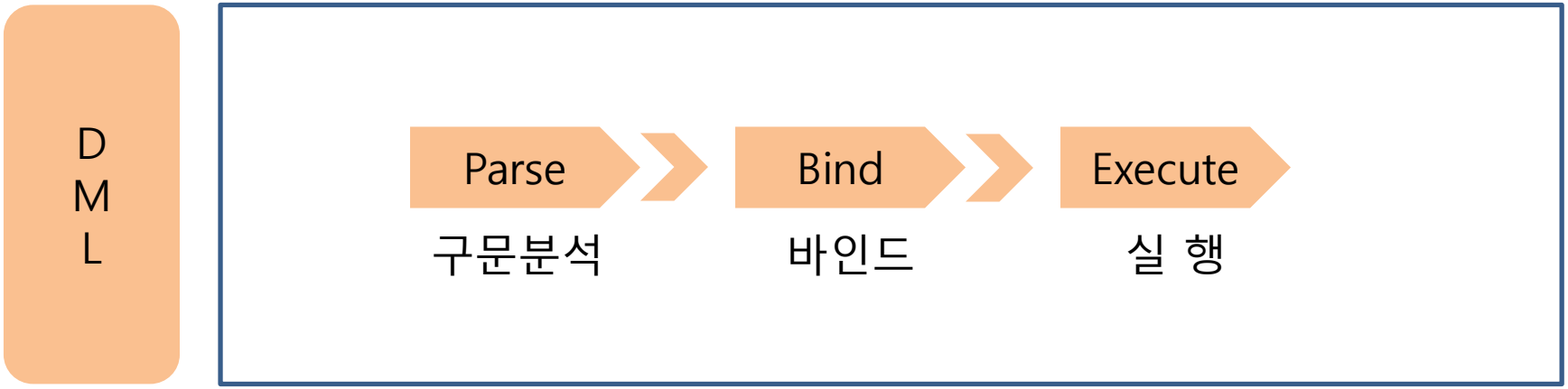
2. SQL 문장의 실행 원리

4) Fetch(인출)

- Execute 단계까지 수행하면, 원하는 데이터가 들어있는 블록이 DB Buffer Cache에 올라오게 됨.
- 데이터의 I/O 최소 단위가 Block이므로, DB Buffer Cache에 원하는 데이터만 있는 것이 아닌 다른 데이터도 존재
- **사용자가 요청한 데이터만 골라내는 과정을 Fetch라고 함**
- 정렬이 필요하거나 추가 작업을 요구하는 경우 Fetch과정에서 Sort를 하여 데이터를 보내주며, 정렬은 PGA영역(Program Global Area)에서 수행

2. SQL 문장의 실행 원리

- DML 문장의 실행 원리
  - 모든 DML의 수행원리는 동일하다.
  - DML의 수행단계는 Fetch과정만 없고 나머지는 동일



2. SQL 문장의 실행 원리

- DML 문장의 실행 원리

  1) 서버프로세스는 Parse과정 수행

  - Library Cache에 실행계획이 있는지 확인
  - 있으면 Soft Parse, 없으면 Hard Parse 수행

  2) 실행계획을 받은 서버프로세스는 Library Cache에 Plan 정보 등록한 후 Execute 수행

  - DB Buffer Cache에 update 하려는 데이터가 있는지 확인
  - 없을 경우 데이터파일의 해당 블록을 DB Buffer Cache에 복사

2. SQL 문장의 실행 원리

3) Execute 단계에서 원하는 데이터가 들어있는 DB Buffer Cache를 가져온 후 Server프로세스는 데이터 변경 내역을 Redo Log Buffer에 먼저 기록

4) 기록 후 Undo Segment에 원본 이미지를 기록한 후 DB Buffer Cache의 내용을 변경

[정리] 데이터 변경이 일어날 경우 순서

Redo Log Buffer에 기록 -> Undo Segment에 기록 -> DB Buffer Cache의 실제 데이터 변경

2. SQL 문장의 실행 원리

Update 문장 수행 과정