

SQLD 시험 대비

Part 2

Section 01. 데이터 모델링 (Data Modeling)

데이터 모델링

- 고객의 비즈니스 프로세스를 이해하고 비즈니스 프로세스의 규칙을 정의
- 추상화: 현실세계 → 데이터베이스
- 단순화: 고객이 이해할 수 있게 복잡하지 않아야 한다.
- 명확: 명확하게 의미가 해석되어야 한다. (한 가지 의미를 가짐)

데이터 모델링 단계

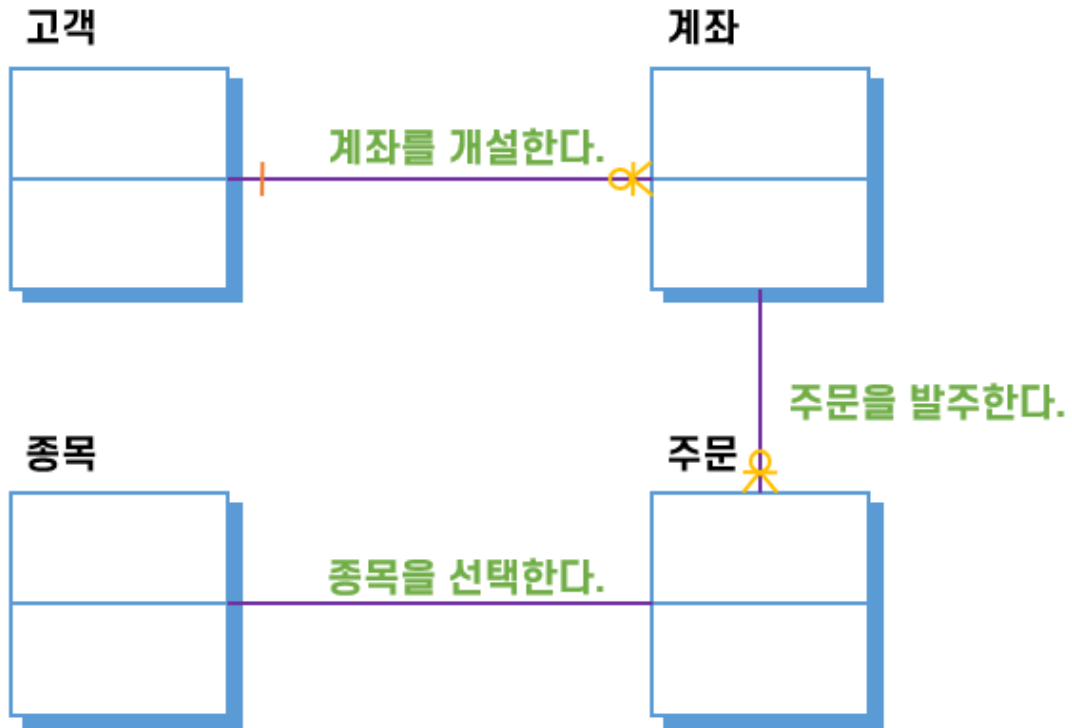
개념적 모델링 (Conceptual Data Modeling)	<ul style="list-style-type: none">• 전사적 관점에서 모델링• 추상화 수준이 가장 높은 수준의 모델• 데이터 베이스 모델 종류 상관 X• 엔터티(Entity)와 속성(Attributes) 도출• 개념적 ER(Entity Relationship Diagram) 작성
논리적 모델링 (Logical Data Modeling)	<ul style="list-style-type: none">• 특정 데이터 베이스 모델에 종속• 식별자 정의, 관계, 속성 등 모두 표현• 정규화 → 재사용성↑
물리적 모델링 (Physical Modeling)	<ul style="list-style-type: none">• 테이블, 인덱스, 함수 등을 생성• 성능, 보안, 가용성 등을 고려하여 데이터베이스 구축

데이터 모델링 과정

데이터	비즈니스 프로세스에서 사용되는 데이터	구조 분석, 정적 분석
프로세스	비즈니스 프로세스에서 수행하는 작업	시나리오 분석, 도메인 분석, 동적 분석
데이터와 프로세스	프로세스와 데이터 간의 관계	CRUD(Create, Read, Update, Delete)분석

ERD(Entity Relationship Diagram)

- entity와 entity 간의 관계를 정의



- ERD 작성 절차
 1. 엔터티를 도출하고 그린다.
 2. 엔터티를 배치한다.
 3. 엔터티 간 관계 설정
 4. 관계명 설정
 5. 관계 참여도를 표현
 6. 관계의 필수 여부를 표현
- 작성 시 고려사항
 - 중요한 엔터티를 왼쪽 상단에 배치
 - ERD는 이해가 쉬워야 하고 너무 복잡하지 않아야 한다.

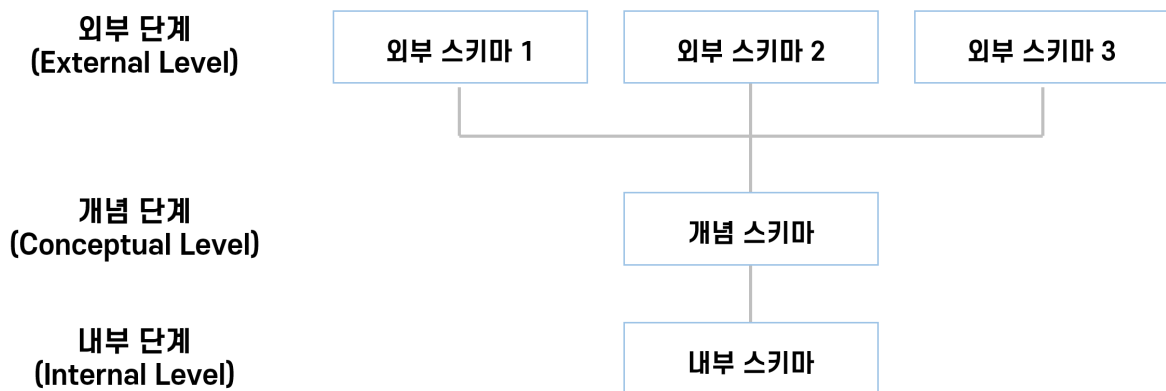
데이터 모델링 고려사항

- 데이터 모델의 독립성
- 고객 요구사항의 표현
- 데이터 품질 확보

3층 스키마 (3-Level Schema)

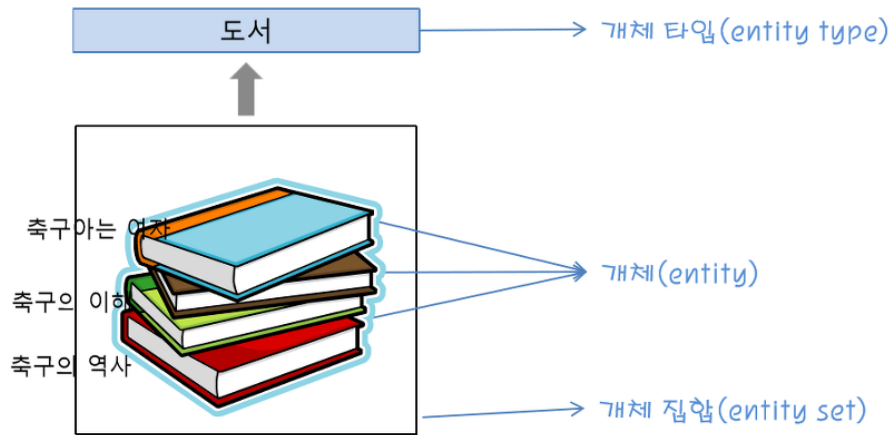
- 사용자, 설계자, 개발자가 데이터베이스를 보는 관점에 따라 데이터베이스를 기술, 관계 정의
- 데이터베이스의 **독립성** 확보 → 복잡도 감소, 중복 제거, 요구사항 변경 대응력 향상, 유지보수 비용 절감
- 각 계층을 뷰(View)라고도 한다.

3층 스키마 구조



구조	설명
외부 스키마 (external schema)	<ul style="list-style-type: none"> • 사용자 관점 • 뷰(View)를 표시 • 응용 프로그램이 접근하는 데이터베이스를 정의
개념 스키마 (conceptual schema)	<ul style="list-style-type: none"> • 설계자 관점 • 규칙, 구조 표현 • 통합 데이터베이스 구조
내부 스키마 (internal schema)	<ul style="list-style-type: none"> • 개발자 관점 • 데이터베이스 물리적 저장 구조 • 데이터 저장 구조, 레코드 구조, 필드 정의, 인덱스 등

엔터티 (Entity)



- 데이터, 업무에서 관리해야 하는 집합으로 저장되고 관리되어야 하는 개념, 사건, 장소 등의 데이터
- 엔터티는 비즈니스 프로세스에서 관리되어야 하는 정보 추출

엔터티의 특징

엔터티 특징	설명
식별자	유일한 식별자
인스턴스 집합	2개 이상의 인스턴스
속성	반드시 속성을 가짐
관계	다른 엔터티와 최소 한 개 이상의 관계
업무	업무에서 관리되어야 하는 집합

엔터티의 종류

유무형

유형 엔터티	지속적으로 사용 물리적 형태 O
개념 엔터티	물리적 형태 X 개념적으로 사용
사건 엔터티	비즈니스 프로세스 실행하며 생성

발생 시점

기본 엔터티	키 엔터티, 독립적으로 생성
중심 엔터티	기본, 행위 중간
행위 엔터티	2개 이상의 엔터티로부터 발생

속성(Attribute)

- 엔터티가 가지는 항목

개체 타입	속성
도서	도서 이름, 출판사, 도서단가

- 일반적으로 하나의 값만 가짐
- 기본키 변경 → 속성값 변경

속성의 종류

분해 여부에 따른 종류

단일 속성	하나의 의미
복합 속성	여러 개의 의미 (주소 → 시, 군, 구)
다중값 속성	여러 개의 값 (상품 리스트), 엔터티로 분해 가능

특성에 따른 속성의 종류

기본 속성	본래의 속성
설계 속성	데이터 모델링 과정에서 발생 유일한 값 부여
파생 속성	다른 속성에 의해서

관계 (Relationship)

- 엔터티 간의 관련성 → 존재 관계 / 행위관계

존재관계

- 엔터티 간의 상태
- ex) 고객 → 지점

행위관계

- 엔터티 간에 어떤 행위가 있음
- ex) 계좌 개설, 주문 발주

관계 차수 (Relation Cardinality)

엔터티에 관계에 참여하는 수

- 1대1 관계: 하나의 엔터티에 하나의 엔터티
 - 완전 1대1: 반드시 존재
 - 선택적 1대1: 없을 수도 있음
- 1대N 관계
 - 하나의 엔터티에 여러개의 엔터티 값
 - ex) 고객 → 계좌1, 계좌 2, ...
- M대N 관계
 - 두 개 엔터티 서로 여러 개의 관계

- ex) 여러 과목 ↔ 여러 학생
- 필수적 관계, 선택적 관계
 - 필수적 관계(|): 반드시 하나 존재
 - 선택적 관계(o): 없을 수도 있는 관계

식별관계, 비식별 관계

- 식별관계
 - 강한 개체: 독립적으로 존재
 - 약한 개체 → 강한 개체의 기본키를 공유 받음
- 비식별관계
 - 강한 개체의 기본키를 다른 엔터티의 기본키가 아닌 일반 칼럼으로 관계

엔터티 식별자

▼ 키의 종류

기본키	후보키 중 엔터티를 대표
후보키	유일성 O, 최소성 O
슈퍼키	유일성 X, 최소성 O
대체키	후보키 중 기본키를 선정하고 남은 키
외래키	참조무결성 확인

- 주식별자 (기본키, Primary Key)
 - 최소성, 대표성, 유일성, 불변성
- 식별자의 종류
 - 대표성

주식별자	엔터티를 대표하는 식별자, 참조관계로 연결
보조식별자	대표성을 만족하지 못하는 식별자

- 생성여부

내부 식별자	엔터티 내부에서 스스로 생성되는 식별자
외부 식별자	다른 엔터티와의 관계로 만들어지는 식별자

- 속성의 수

단일 식별자	하나의 속성으로 구성
복합 식별자	두 개 이상의 속성으로 구성

- 대체여부

본질 식별자	비즈니스 프로세스에서 만들어진 식별자
인조 식별자	인위적으로 만들어진 식별자

Section 02. 데이터 모델과 성능

정규화

- 데이터의 일관성, 최소한의 중복, 최대한의 유연성
- 제1정규화 ~ 제5정규화 → 실질적으로 제3정규화까지 수행

함수의 종속성

- $X \rightarrow Y$ 면 Y는 X에 함수적으로 종속

제1정규화	속성의 원자성 확보 기본키 설정
제2정규화	기본키 2개 이상의 속성 → 부분 함수 종속성을 제거
제3정규화	이행 함수 종속성(기본키 제외 칼럼 간 종속성)을 제거
BCNF	후보키가 기본키를 종속시키면 분해
제4정규화	여러 칼럼들이 하나의 칼럼 종속 → 분해하여 다중값 종속성 제거
제5정규화	조인에 의해 종속성이 발생하는 경우 분해

정규화의 문제점

- 데이터 조회(SELECT)시에 조인(Join)유발 → CPU와 메모리 많이 사용
- 성능 저하 문제 해결 → 반정규화

반정규화

- 데이터베이스 성능 향상 → 데이터 중복 허용, 조인↓

반정규화 수행하는 경우

- 수행 속도가 느려지는 경우
- 다량의 범위 자주 처리
- 특정 범위의 데이터 자주 처리
- 요약/집계 정보 자주 요구

대상 조사 및 검토 → 다른 방법 검토 → 반정규화 수행

반정규화 기법

1. 계산된 칼럼 추가
2. 테이블 수직 분할
3. 테이블 수평 분할

▼ 파티션 기법

- 논리적으로는 하나의 테이블이지만, 여러 개의 데이터 파일이 분산되어 저장
 - Range Partition: 데이터 값 범위 기준
 - List Partition: 특정한 값을 지정
 - Hash Partition: 해시 함수 적용
 - Composite Partition: 범위, 해시 복합 사용
- 액세스 범위 줄어 성능 향상, I/O 성능 향상, 백업 및 복구 가능

4. 테이블 병합

- 1대1 관계 병합, 1대 N 관계 병합 (중복데이터 발생), 슈퍼 타입 + 서브타입

▼ 슈퍼타입 서브타입

- 슈퍼타입 → 서브타입1, 서브타입 2, ... (부모자식관계)
- OneToOne Type: 개별 테이블로 도출
- Plus Type: 슈퍼 타입, 서브 타입 테이블로 도출
- Single Type: 하나의 테이블로 도출

분산 데이터베이스

- 중앙 집중형 데이터베이스
 - 하나의 물리적 시스템에 여러 명의 사용자

- 분산 데이터베이스
 - 물리적으로 떨어진 데이터베이스 네트워크로 연결 → 단일 데이터베이스 이미지

투명성

- 네트워크로 분산되어 있는지 여부 인식X, 자신만의 데이터베이스를 사용하는 것처럼
- 분할: 사본이 여러 시스템에 저장되어 있음 인식 X
- 위치: 데이터 저장 장소 명시 X
- 지역사상: 지역 시스템 이름과 무관한 이름 사용
- 중복: 고객과는 무관하게 데이터의 일관성 유지
- 장애: 시스템, 통신망 이상에도 무결성 보장
- 병행: 응용 프로그램 동시에 트랜잭션 수행해도 결과에 이상X

분산 데이터베이스 설계 방식

- 상향식
 - 지역 스키마 작성 후 전역 스키마 작성
- 하향식
 - 전역 스키마 작성 후 지역 스키마 작성
 - 전사 데이터 모델을 수렴하여 전역 스키마 생성 → 분산 데이터베이스 구축

분산 데이터베이스 장점과 단점

장점	단점
- 신뢰성, 가용성 - 빠른 응답 가능 - 용량 확장 쉬움	- 관리 통제 어려움 - 보안 관리 어려움 - 무결성 관리 어려움 - 설계 복잡

정규화와 성능

정규화의 문제점

```
SELECT 사원번호, 부서코드, 부서명, 이름, 전화번호, 주소
FROM 직원, 부서
```

```
WHERE 직원, 부서코드 = 부서, 부서코드;
```

```
# ANSI Join  
SELECT 사원번호, 부서코드, 부서명, 이름, 전화번호, 주소  
FROM 직원 INNER JOIN 부서  
ON 직원, 부서코드 = 부서, 부서코드
```

- 데이터 조회(SELECT) 시에 조인(JOIN) 유발 → CPU와 메모리 많이 사용
- 중첩된 루프 (Nested Loop) 사용 → 성능저하
- 성능 저하 문제 해결 → 인덱스, 옵티마이저 / 반정규화

정규화를 사용한 성능 튜닝

- 반정규화
 - 데이터 중복 → 또 다른 문제 발생
 - 조인 최소화
 - 장점: 조회 빨라짐
 - 단점: 너무 많은 칼럼 추가 → 여러 개의 블록 → 성능 저하
 - 칼럼의 증가 문제는 테이블 분해만 해결 가능
- 정규화 → 입출력 데이터의 양을 줄여 성능 향상

4. 분산 데이터베이스

데이터베이스

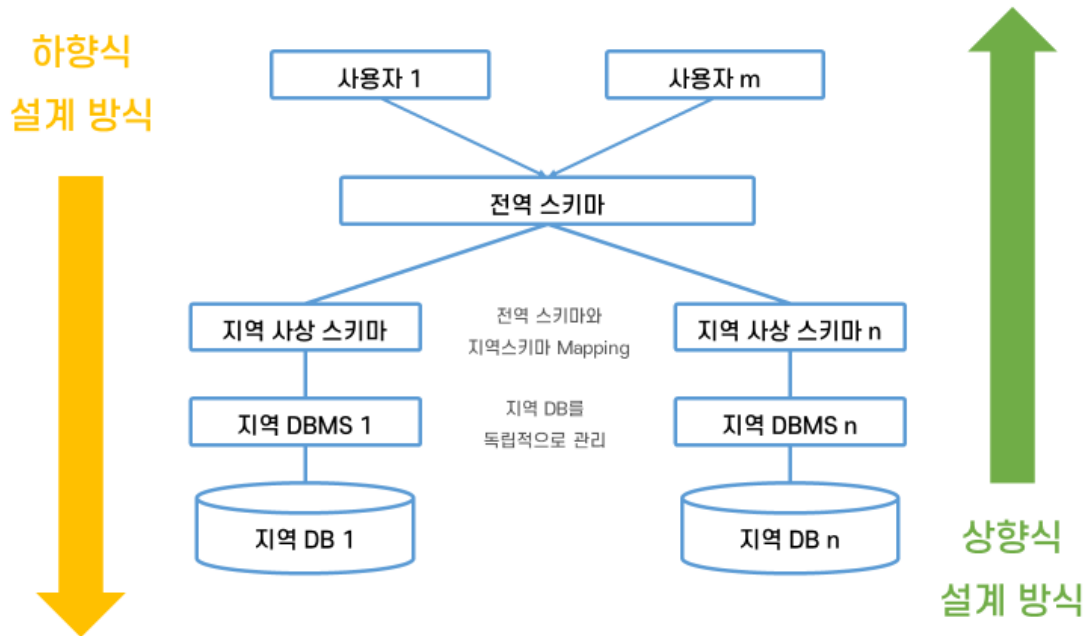
- 중앙 집중형 데이터베이스: 하나의 데이터베이스 관리 시스템
- 분산 데이터베이스: 물리적으로 떨어진 여러 개 데이터베이스

분산 데이터베이스의 투명성

- | | |
|---|--|
| <ul style="list-style-type: none">• 투명성<ul style="list-style-type: none">◦ 네트워크로 분산 여부 인식 X◦ 자신만의 데이터베이스 처럼 사용 | <ul style="list-style-type: none">• 투명성의 종류<ul style="list-style-type: none">◦ 분할 투명성◦ 위치 투명성◦ 지역 사상 투명성 |
|---|--|

- 중복 투명성
- 장애 투명성
- 병행 투명성

분산 데이터베이스 설계 방식



- 하향식 접근 방식
 - 전사 데이터 모델 → 전사 스키마
 - 지역스키마 → 분산 데이터베이스

분산 데이터베이스 장단점

장점	단점
<ul style="list-style-type: none"> • 신뢰성, 가용성 • 빠른 응답 • 용량 확장 	<ul style="list-style-type: none"> • 관리 통제 어려움 • 무결성 관리 어려움 • 설계 복잡

Part 3

Section 01

GROUP 연산

```
SELECT DEPTNO, SUM(SAL) # 그룹별로 SAL의 합계 계산
      FROM EMP
     GROUP BY DEPTNO; // DEPTNO를 기준으로 그룹화

//HAVING - 조건절
HAVING SUM(SAL) > 10000; //합계 10000 초과인 경우만

// ORDER BY - 정렬
ORDER BY ...
```

- 그룹화 → 평균, 최댓값, 최솟값 등 계산 가능

집계 함수 종류

함수명	기능
COUNT()	행 수
SUM()	합계
AVG()	평균
MAX() / MIN()	최댓값 / 최솟값
STDDEV()	표준편차
VARIANCE()	분산

SELECT문 실행 순서

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

명시적 형변환과 암시적 형변환

형변환

- 두 개의 데이터의 데이터타입(형)이 일치하도록 변환
- **명시적(Explicit)** 형변환: 형변환 함수를 사용해서 데이터 타입을 일치
 - TO_NUMBER(문자열): 문자열 → 숫자

- TO_CHAR(숫자 혹은 날짜, [FORMAT]): 숫자, 날짜 → FORMAT의 문자
- TO_DATE(문자열, FORMAT): 문자열을 지정된 FORMAT의 날짜형으로 변환
- **암시적 (Implicit) 형변환**: DBMS 자동으로 형변환
- 인덱스 컬럼에 형변환 → 인덱스 사용 x

내장형 함수

DUAL 테이블

- 오라클에서 사용되는 Dummy 테이블
- 내장형 함수 실행 가능
 - 형변환 함수
 - 문자열 및 숫자형 함수
 - 날짜형 함수

문자열 함수

문자열 함수	설명
ASCII(문자)	문자/숫자 → ASCII
CHR/CHAR(ASCII 코드값)	ASCII코드 → 문자
SUBSTR(문자열, m, n)	문자 m번째~n번째
CONCAT(문자열1, 문자열2)	문자열1 + 문자열2
LOWER(문자열)	영문자 → 소문자
UPPER(문자열)	영문자 → 대문자
LENGTH 혹은 LEN(문자열)	공백 포함 문자열 길이
LTRIM(문자열, 지정문자)	왼쪽에서 지정된 문자 삭제
RTRIM(문자열, 지정문자)	오른쪽에서 지정된 문자 삭제
TRIM(문자열, 지정된 문자)	왼쪽 & 오른쪽 지정된 문자 삭제

숫자형 함수

숫자형 함수	설명
ABS(숫자)	절대값
SIGN(숫자)	양수, 음수, 0 구별
MOD(숫자1, 숫자2)	숫자 1 % 숫자2

날짜형 함수

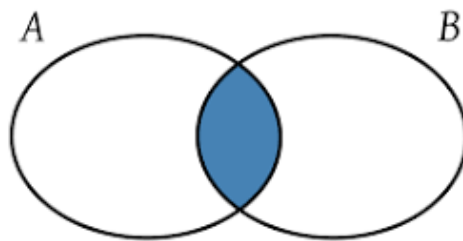
날짜형 함수	설명
SYSDATE	오늘의 날짜 → 날짜 타입
EXTRACT(YEAR FROM SYSDATE)	날짜에서 년, 월, 일 조회

숫자형 함수	설명
CEIL/CEILING(숫자)	크거나 같은 최소의 정수
FLOOR(숫자)	작거나 같은 최대의 정수
ROUND(숫자, m)	소수점 m자리에서 반올림
TRUNC(숫자, m)	소수점 m자리에서 절삭

Section 02

1. 조인(JOIN)

- 여러 개의 릴레이션을 사용하여 새로운 릴레이션만들어냄
- 교집합 (두 개의 테이블 간에 일치하는 것을 조인)



EQUI JOIN (등가조인)

- 두 테이블 사이 공통된 컬럼이 같은 것을 조인
- 해시 함수는 테이블리를 해시 메모리에 적재한 후에 해시 함수로 연결
- 해시 조인은 EQUI JOIN으로만 가능

```
// 1. EQUI JOIN - =을 통해 테이블 연결
SELECT * FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO // =로 두 테이블 연결
      AND EMP.ENAME LIKE '임%' // 조인에 추가 조건 및 정렬 간으
ORDER BY ENAME;
```

```
// 2. INNER JOIN - ON문을 통해 테이블 연결
SELECT / FROM EMP INNER JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

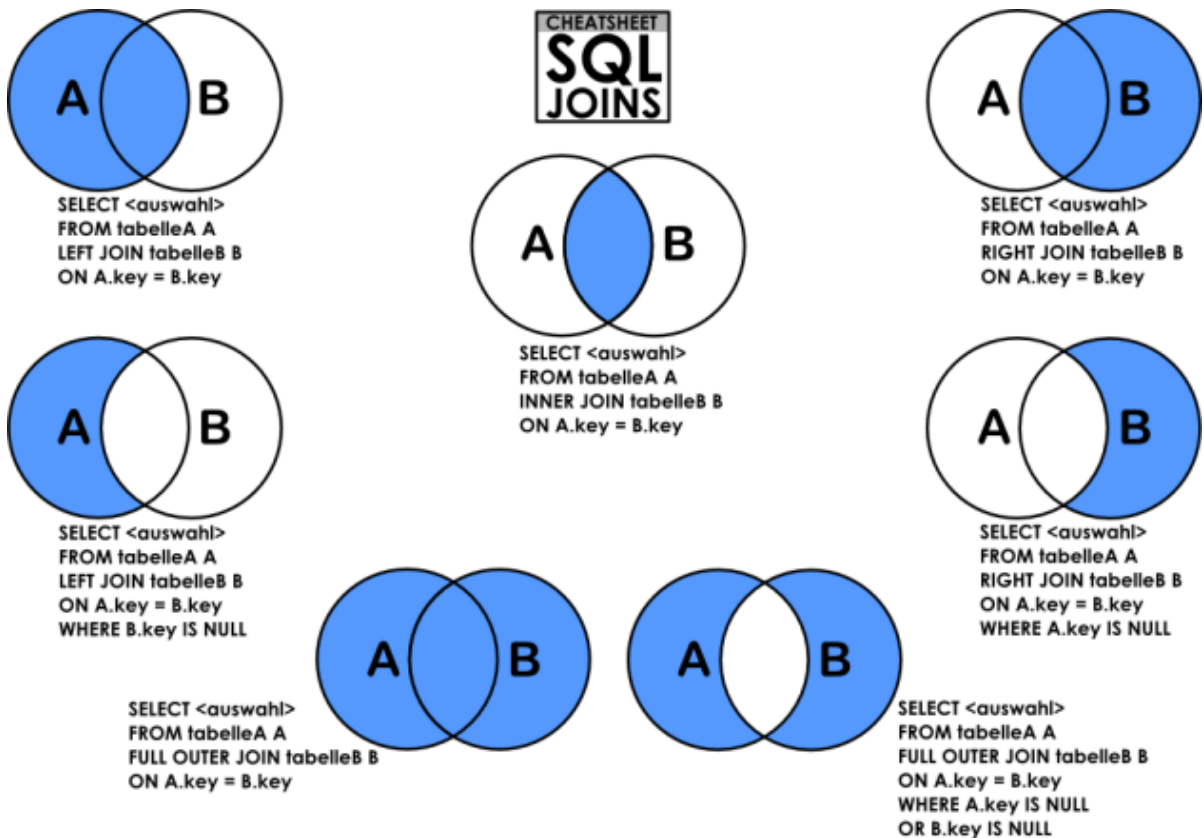
```
// 3. INTERSECT 연산 - 두 테이블에서 공통된 값을 조회
SELECT DEPTNO FROM EMP
INTERSECT
SELECT DEPTNO FROM DEPT;
```

NON-EQUI JOIN (비등가 조인)

- 두 개의 테이블 간에 조인하는 경우 $>$, $<$, \geq , \leq 사용
- 정확하게 일치하지 않는 것을 조인

OUTER JOIN

- 두 개의 테이블 간에 교집합 + 한쪽 테이블에만 있는 데이터도 포



```
SELECT * FROM DPET, EMP
WHERE EMP.DEPTNO (+)= DEPT.DEPTNO;
```

```
// 1. LEFT OUTER JOIN / RIGHT OUTER JOIN
```

```
SELECT * FROM DEPT LEFT OUTER JOIN EMP
      ON EMP.DEPTNO = DEPT.DEPTNO;
```

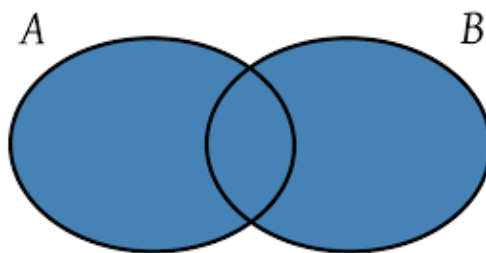
CROSS JOIN

- 조인 조건구 없이 2개의 테이블을 하나로 조인
- 카테시안 곱이 발생

```
SELECT * FROM EMP CROSS JOIN DEPT;
```

UNION

- 두 개의 테이블을 하나로 만드는 연산 (합집합)
- 칼럼 수 데이터 형식이 다르면 오류발생
- 중복 데이터 제거 → 정렬 발생

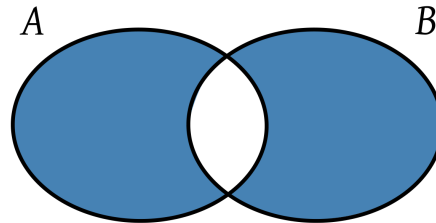


```
// 1. UNION
SELECT DEPTNO FROM EMP
UNION
SELECT DEPTNO FROM EMP;

// 2. UNION ALL - 중복제거, 정렬X
SELECT DEPTNO FROM EMP
UNION ALL
SELECT DEPTNO FROM EMP;
```

MINUS

- 두 개의 테이블에서 차집합을 조회
- MINUS = EXCEPT



```
SELECT DEPTNO FROM DEPT
MINUS
SELECT DEPTNO FROM EMP;
```

2. 계층형 조회(CONNECT BY)

- 계층형으로 데이터 조회 가능
- 트리 (Tree) 형태의 구조로 질의를 수행

```
SELECT MAX(LEVEL) // 최대 계층수 - LEVEL은 계층값을 가짐
FROM Limbest.EMP
START WITH MGR IS NULL // 시작조건
CONNECT BY PRIOR EMPNO = MGR; // 조인 조건, PRIOR = 바로 직전 출력 행
```

키워드	설명
LEVEL	검색 항목의 깊이 (가장 상위 레벨이 1)
CONNECT_BY_ROOT	가장 최상위 값
CONNECT_BY_ISLEAF	가장 최하위
SYS_CONNECT_BY_PATH	전체 전개 경로 표시
NOCYCLE	순환 구조가 발생 지점까지만 전개
CONNECT_BY_ISCYCLE	순환 구조 발생 지점 표시

키워드	설명
START WITH 조건	계층 전개의 시작 위치 지정
PRIOR 자식 = 부모	부모 → 자식 (순방향)

키워드	설명
PRIOR 부모 = 자식	자식 → 부모 (역방향)
NOCYCLE	사이클이 발생되지 않게
Order siblings by 컬럼명	형제 노드 사이에서 정렬

3. 서브쿼리(Subquery)

Main query와 Subquery

- SELECT문 내에 다시 SELECT문 사용
- 인라인 뷰(Inline view) : FROM + SELECT
- 스칼라 서브쿼리(Scala Subquery): SELECT + Subquery
- 서브쿼리(Subquery): WHERE + SELECT

```
SELECT *
  FROM EMP
 WHERE DEPTNO =
        (SELECT DEPTNO FROM DEPT
         WHERE DEPTNO = 10);
```

단일 행 서브쿼리와 다중 행 서브쿼리

- 반환하는 행 수가 한 개인 것과 여러 개인 것에 따라 나눈다.

서브쿼리 종류	설명
단일 행 서브쿼리	결과는 반드시 한 행만 조회 =, <, ≤, ≥, <>
다중 행 서브쿼리	결과 여러 행 조회 IN ANY, ALL, EXISTS

다중 행(Multi row) Subquery)

다중 행 연산	설명
IN	Subquery의 결과 중 하나만 동일하면 참
ALL	Main query, Subquery 모두 결과 동일하면 참
ANY	Subquery하나 이상만 동일하면 참

다중 행 연산	설명
EXISTS	Main query, Subquery결과가 하나라도 존재하면 참

스칼라(Scala) Subquery

- 반드시 한 행과 한 칼럼만 반환
- 여러 행 반환 시 오류 발생

```
SELECT ENAME AS "이름"
      SAL AS "급여",
      (SELECT AVG(SAL)
       FROM EMP
       ) AS "평균급여"
FROM EMP
WHERE EMPNO = 1000;
```

연관(Correlated) Subquery

- Subquery 내에서 Main Query 내의 칼럼을 사용

```
FROM EMP a
WHERE a.DEPTNO =
      (SELECT DEPTNO FROM DEPT b
       WHERE b.DEPTNO = a.DEPTNO);
```

4. 그룹 함수 (Group Function)

ROLLUP

- GROUP BY 칼럼에 Subtotal을 만들어 준다.
- GROUP BY 구에 두 개 이상 오면 순서에 따라서 결과가 달라진다.

```
SELECT DECODE (DEPTNO, NULL, '전체합계', DEPTNO),
      SUM(SAL)
FROM EMP
GROUP BY ROLLUP (DEPTNO);
```

GROUPING

- ROLLUP, CUBE, GROUPING SETS 함께 구분

```
SELECT DEPTNO, GROUPING(DEPTNO),  
        JOB, GROUPING(JOB), SUM(SAL)  
FROM EMP  
GROUP BY ROLLUP(DEPTNO, JOB);
```

GROUPING SETS

- GROUP BY에 나오는 칼럼의 순서와 관계 없이 다양한 소계

```
SELECT DEPTNO, JOB, SUM(SAL)  
FROM EMP  
GROUP BY GROUPING SETS(DEPTNO, JOB);
```

CUBE

- 제시한 칼럼에 대해서 결합 가능한 모든 집계를 계산

```
SELECT DEPTNO, JOB, SUM(SAL)  
FROM EMP  
GROUP BY CUBE(DEPTNO, JOB)
```

5. 윈도우 함수(Window Function)

- 행과 행 간의 관계를 정의
- 순위, 합계, 평균, 행 위치 등 조작 가능

```
SELECT WINDOW_FUNCTION(ARGUMENTS)  
      OVER(PARTITION BY 칼럼  
           ORDER BY WINDOWING 절)  
FROM 컬럼명;
```

윈도우 함수 구조

구조	설명
ARGUMENTS (인수)	0~N개의 인수 설정
PARTITION BY	전체 집합 → 소그룹
ORDER BY	정렬
WINDOWING	ROW: 물리적 결과 범위 RANGE: 논리적 값 범위

WINDOWING

구조	설명
ROWS	부분집합인 윈도우 크기를 물리적 단위로 행의 집합 지정
RANGE	논리적인 주소에 의해 행 집합 지정
BETWEEN ~ AND	윈도우의 시작과 끝 위치를 지정
UNBOUNDED PRECEDING	윈도우의 시작 위치 = 첫 번째 행
UNBOUNDED FOLLOWING	윈도우의 마지막 위치 = 마지막 행
CURRENT ROW	윈도우 시작 위치 = 현재 행

```
// 사용 예시
SELECT EMPNO, ENAME, SAL,
       SUM(SAL) OVER(ORDER BY SAL // 합계 구하기, 데이터 SAL를 기준으로 정
                      ROWS BETWEEN UNOUNDED PRECEDING
                      AND UNBOUNDED FOLLOWING) TOTSAL ,
FROM EMP; // TOTSAL은 EMP 테이블
```

순위 함수 (RANK FUNCTION)

```
SELECT ENAME, SAL,
       RANK() OVER(ORDER BY SAL DESC) ALL RANK, // SAL로 정렬 -> ALL
       RANK() OVER (PARTITION BY JOB ORDER BY SAL // JOB으로 파티션 생
                     DESC) JOB RANK
FROM EMP;
```

함수명	동일한 수 처리
RANK	동일한 값
DENSE_RANK	하나의 건수로
ROW_NUMBER	고유의 순위 부여

집계함수(AGGREGATE FUNCTION)

```
SELECT ENAME, SAL,  
       SUM(SAL) OVER(PARTITION BY MGR) SUM MGR  
FROM EMP;
```

집계함수	설명
SUM	파티션 별 합계
AVG	파티션 별 평균
COUNT	파티션 별 행
MAX / MIN	파티션 별 최댓값, 최솟값

행 순서 관련 함수

```
SELECT DEPTNO ENAME, SAL,  
       FIRST_VALUE(ENAME) OVER (PARTITION BY DEPTNO) // 가장 처음 나오  
       ORDER BY SAL DESC ROWS UNBOUNDED PRECEDING) AS // SAL 정렬  
DEPT_A FROM EMP;
```

행 순서	설명
FIRST_VALUE	가장 처음에 나오는 값 MIN함수와 같음
LAST_VALUE	가장 나중에 나오는 값 MAX함수와 같음
LAG	이전 행
LEAD	특정한 위치의 행 (default: 1)

비율 관련 함수

```
SELECT DEPTNO, ENAME, SAL,  
       PERCENT_RANK() OVER(PARTITION BY DEPTNO  
       ORDER BY SAL DESC) AS PERCENT SAL  
FROM EMP;
```

비율 함수	설명
CUME_DIST	누적 백분율을 조회 (0~1)
PERCENT_RANK	순서별 백분율 조회 (가장 먼저 0, 가장 나중 1)
NTILE	전체 건수를 ARGUMENT값으로 N등분한 결과
RATIO_TO_REPORT	전체 SUMDP 대해 행 별 칼럼값의 백분율을 소수점까지 조회

6. 테이블 파티션(Table Partition)

Partition 기능

- 파티션 → 여러 개의 데이터 파일에 분리하여 저장
- 데이터가 물리적으로 분리된 데이터 파일에 저장 → 입력, 수정, 삭제, 조회 성능이 향상
- 파티션은 독립적으로 관리 가능
- 파티션 별로 백업, 복구 가능 → 파티션 전용 인덱스 생성도 가능
- 테이블 스페이스 간에 이동 가능
- 데이터의 범위를 줄여 성능 향상

Range Partition

- 여러 개의 파티션으로 데이터를 나누어 저장

List Partition

- 특정 값을 기준으로 분할

Hash Partition

- 데이터베이스 관리 시스템이 내부적으로 해시 함수를 사용해서 데이터를 분할
- 데이터베이스 관리 시스템이 알아서 분할하고 관리

파티션 인덱스

구분	주요 내용
Global Index	여러 개의 파티션에 하나의 인덱스
Local Index	해당 파티션 별로 각자의 인덱스 사용

구분	주요 내용
Prefixed Index	파티션 키와 인덱스 키가 동일
NonPrefixed Index	파티션 키와 인덱스 키가 다르다.

Section 3

옵티마이저(Optimizer)와 실행 계획

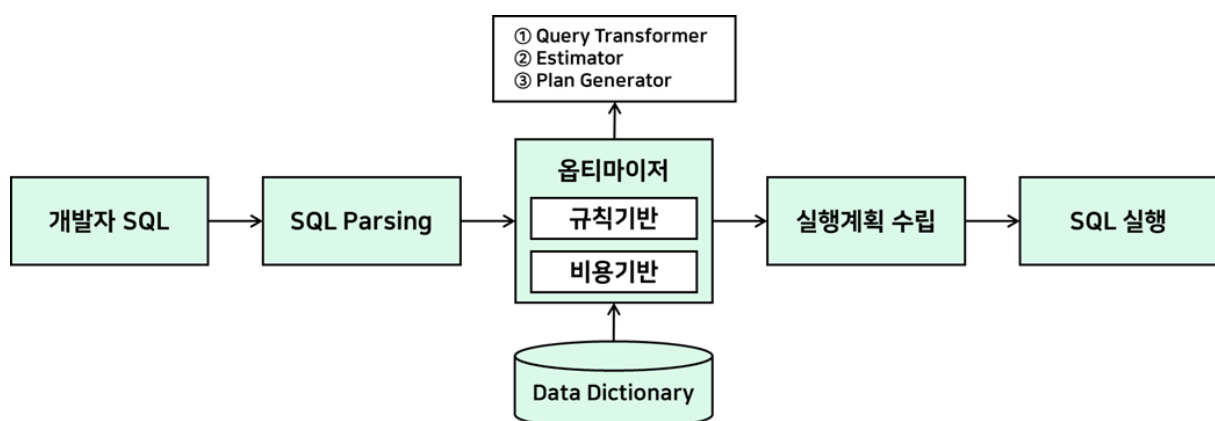
옵티마이저

- SQL 실행 계획을 수립하고 데이터베이스 관리 시스템의 소프트웨어
- 데이터 디렉터리리에 있는 오브젝트 통계, 시스템 통계 → 예상 비용 산정 후 최저 비용 선택
- 옵티마이저 실행 계획 → SQL 성능에 중요한 역할
- 옵티마이저 실행 계획 변경 → 힌트(HINT) 사용
- PLAN_TABLE에 SQL 실행 계획 저장 (TOAD에서 Execution Plan Current SQL 메뉴 선택)
- TABLE ACCESS FULL → 테이블 전체를 읽었다는 의미

옵티마이저 종류

옵티마이저 실행 방법

1. SQL 실행 → 파싱(Parsing) → SQL의 문법 검사 및 구문 분석 수행
2. 옵티마이저가 규칙 기반 혹은 비용 기반으로 실행 계획 수립 (보통 비용 기반 옵티마이저)
3. 실행 계획 수립 완료 → SQL 실행 → 데이터 인출(Fetch)



Query Transformer	- SQL 문을 효율적으로 실행하기 위해서 옵티마이저가 변환 - SQL이 변환되어도 그 결과는 동일
Estimator	- 통계정보를 사용하여 SQL 실행 비용을 계산 - 총비용은 최적의 실행 계획을 수립하기 위해서
Plan Generator	- SQL을 실행할 실행 계획을 수립

옵티마이저 엔진

- 15개의 우선순위를 기준으로 실행 계획 수립
1. ROWID를 사용한 단일 행인 경우
 2. 클러스터 조인에 의한 단일 행인 경우
 3. 유일하거나 기본키를 가진 해시 클러스터 키에 의한 단일 행인 경우
 4. 유일하거나 기본키에 의한 단일 행인 경우
 5. 클러스터 조인인 경우
 6. 해시 클러스터 조인인 경우
 7. 인덱스 클러스터 키인 경우
 8. 복합 칼럼 인덱스인 경우
 9. 단일 칼럼 인덱스인 경우
 10. 인덱스가 구성된 칼럼에서 제한된 범위를 검색하는 경우
 11. 인덱스가 구성된 칼럼에서 무제한 범위를 검색하는 경우
 12. 정렬-병합(Sort Merge) 조인인 경우
 13. 인덱스가 구성된 칼럼에서 MAX 혹은 MIN을 구하는 경우
 14. 인덱스가 구성된 칼럼에서 ORDER BY를 실행하는 경우
 15. 전체 테이블을 스캔(FULL TABLE SCAN)하는 경우

비용 기반 옵티마이저

- 오브젝트 통계, 시스템 통계 → 총 비용 계산
- SQL문 실행을 위해 예상되는 소요시간, 자원의 사용량
- 총 비용이 적은쪽으로 선택
- 통계 정보가 적은 경우 성능 저하 발생 가능

인덱스 (Index)

- 데이터 빠르게 검색 가능
- 오름차순, 내림차순 탐색 가능
- 하나의 테이블에 여러 개의 인덱스 가능
- Leaf Block → Double Linked List 형태로 양방향 탐색 가능
- ROWID를 사용하여 EMP 테이블의 행 직접 읽음

인덱스 생성

```
CREATE INDEX IND_EMP ON  
EMP (ENAME ASC, SAL DESC)
```

인덱스 스캔 (Index Scan)

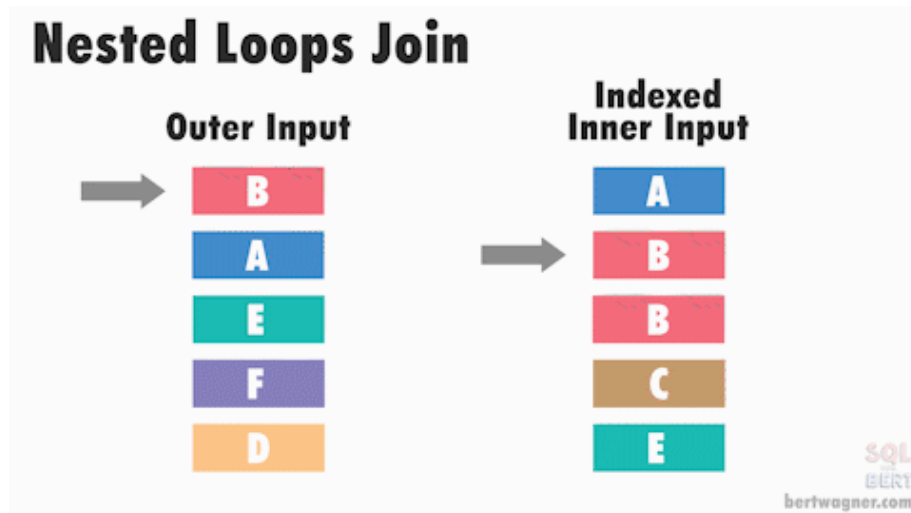
```
# 1. 인덱스 유일 스캔 (Index Unique Scan)  
# 인덱스의 키 값이 중복되지 않는 경우 해당 인덱스를 사용할 때 발생  
SELECT * FROM EMP WHERE EMPNO = 1000;  
  
# 2. 인덱스 범위 스캔 (Index Range Scan)  
# SELECT문에서 특정 범위를 조회하는 WHERE문으로 사용할 경우 발생  
SELECT EMPNO FROM EMP  
WHERE EMPNO >= 1000;  
  
# 3. 인덱스 전체 스캔 (Index Full Scan)  
# 인덱스에서 검색되는 인덱스 키가 많은 경우  
SELECT ENAME, SAL FROM EMP  
WHERE ENAME Like '%' AND SAL > 0;
```

- 인덱스 유일 스캔의 속도가 가장 빠르다.

옵티마이저 조인 (Optimizer Join)

Nested Loop 조인

- 하나의 테이블에서 데이터를 먼저 찾고 그다음 테이블을 조인하는 방식
 - 조인에서 먼저 조회되는 테이블 → 외부 테이블(Outer Table)
 - 그 다음 조회되는 테이블 → 내부 테이블(Inner Table)이라고 한다.
- 외부 테이블(선행 테이블)의 크기가 작은 것을 먼저 찾는 것이 중요 (범위 ↓)
- RANDOM ACCESS발생 → 성능 지연 발생 (RANDOM ACCESS의 양을 줄여야 성능 향상 가능)



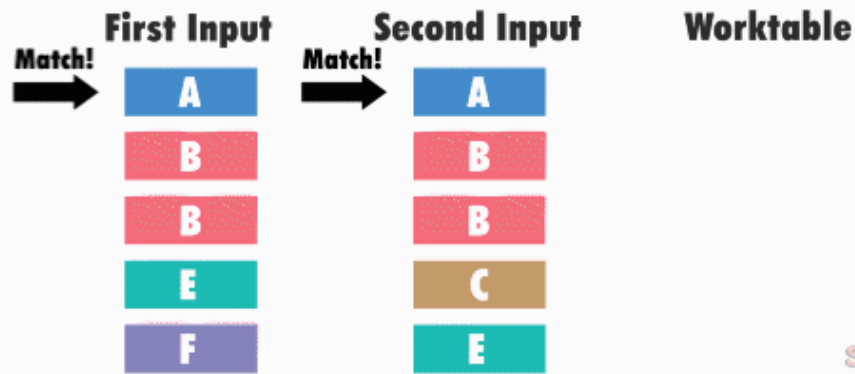
```
SELECT /*+ orderd use nl (B) */ *
FROM EMP a, DEPT b
WHERE a.DEPTNO = b.DEPTNO
      AND a.DEPTNO = 10;
```

- EMP 테이블 FULL SCAN → DEPT 테이블 FULL SCAN → Nested Loop

Sort Merge 조인

- SORT_AREA라는 메모리 공간에 모두 로딩>Loading>하고 SORT를 수행
- 두 테이블에 대해서 SORT가 완료되면 두 개의 테이블을 병합>Merge>)
- 정렬 → 데이터양이 많아지면 성능 떨어짐

Merge Join



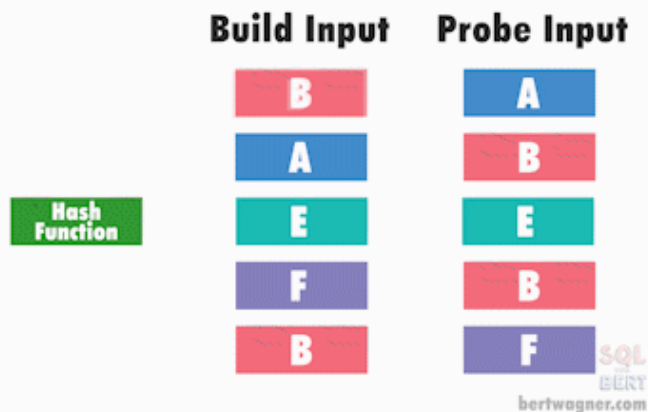
SQL
BERT
bertwagner.com

```
SELECT /*+ ordered use merge (b) */
FROM EMP a, DEPT b
WHERE a.DEPTNO = b.DEPTNO
AND a.DEPTNO = 10;
```

Hash 조인

- 두 개의 테이블 중에서 작은 테이블을 HASH 메모리에 로딩하고 두 개의 테이블의 조인 키를 사용해서 해시 테이블 생성
- 해시 함수로 주소 계산, 해당 주소를 사용해서 테이블 조인 → CPU 연산 ↑
- 선행 테이블이 충분히 메모리에 로딩되는 크기여야 한다.

Hash Match Join



SQL
BERT
bertwagner.com

```
SELECT /*+ ordered use hash(b) */*  
FROM EMP a, DEPT b  
WHERE a.DEPTNO = b.DEPTNO  
      AND a.DEPTNO = 10;
```