SQLD 요약

과목 I: 데이터 모델링의 이해
과목 II: SQL 기본 및 활용

BOAZ 5 기 이은지

참조 사이트

<SQLD 정리 & 카페> -- Q&A

- ✓ http://www.gurubee.net/oracle/advanced
- ✓ http://cafe.naver.com/sqlpd

<SQLD 실습파일>

- ✓ http://www.dbguide.net/logon.db?uri=da.db&cmd=snb9_2_view&boardGroupUid=6&boardConfigUid=59&boardUid=148066

<SQLD 데이터 전문가 가이드 요약 & 기출문제>

- ✓ http://wiki.gurubee.net/pages/viewpage.action?pageId=27426887
- ✓ http://wiki.gurubee.net/pages/viewpage.action?pageId=26743251

<과목 I: 데이터 모델링의 이해>

Chap1. 데이터 모델링의 이해

1) 데이터 모델의 이해

1. 모델링의 이해
- 모델링의 정의
  : 복잡한 현실세계를 일정한 표기법에 의해 표현하는 일
- 모델링의 특징(3)
- 모델링의 관점(3)

2. 데이터 모델의 기본 개념의 이해
- 데이터 모델링의 정의:
  정보시스템을 구축하기 위한 데이터 관점의 업무분석 기법
- 데이터 모델이 제공하는 기능

3. 데이터 모델링의 중요성 및 유의점
- 파급효과 (Leverage)
- 복잡한 정보요구사항의 간결한 표현 (Conciseness)
- 데이터의 품질 (Data Quality)
- 유의점: 중복, 비유연성, 비일관성

4. 데이터 모델링의 3단계 진행: 개.논.물.
- 개념적 데이터 모델링
- 논리적 데이터 모델링
- 물리적 데이터 모델링

5. 프로젝트 생명주기에서 데이터 모델링
데이터 축/ 애플리케이션 축(Process)
- 분석: 개념적&논리적 데이터 모델링 / 프로세스 모델링
→ 설계: 물리적 데이터 모델링 / AP 설계
→ 개발: DB 구축 / AP 개발
→ 테스트: DB 튜닝 / AP 테스트
→ 전환/ 이행: DB 전환 / AP 설치

6. 데이터 모델링에서 데이터독립성의 이해
- 데이터독립성의 필요성
- 데이터베이스 3단계 구조
- 데이터독립성 요소
- 두 영역의 데이터독립성
- 사상 (Mapping)

7. 데이터 모델링의 중요한 세 가지 개념
- 데이터 모델링의 세 가지 요소
- 단수와 집합 (복수)의 명명

8. 데이터 모델링의 이해관계자
- 이해관계자의 데이터 모델링 중요성 인식
- 데이터 모델링의 이해관계자

9. 데이터 모델의 표기법인 ERD의 이해
- 데이터 모델 표기법: Chen/ IE/ Barker
- ERD 표기법을 이용하여 모델링하는 방법

10. 좋은 데이터 모델의 요소
- 완전성 (Completeness)
- 중복 배제 (Non-Redundancy)
- 업무 규칙 (Business Rules)
- 데이터 재사용 (Data Reusability)
- 의사소통 (Communication)
- 통합성 (Integration)

2) 엔터티 (Entity)

1. 엔터티 개념
- 업무에 필요하고 유용한 정보를 저장하고 관리하기 위한 집합적인 것 (Thing)

cf. 인스턴스의 개념: 엔터티의 하나의 값

2. 엔터티와 인스턴스에 대한 내용과 표기법

3. 엔터티 특징 (6): **업무 2 + SQL 4**
- 업무에서 필요로 하는 정보
- 업무프로세스에 의해 활용
- 식별이 가능해야 함
- 인스턴스의 집합
- 속성을 포함
- 관계의 존재

cf. 관계를 생략해 표현해야 하는 경우 (3)

4. 엔터티 분류 (2)
- 유무형에 따른 분류 (3)
- 발생시점에 따른 분류 (3)

5. 엔터티 명명
- 현업업무에서 사용하는 용어 사용
- 약어를 사용하지 않음
- 단수 명사 사용
- 모든 엔터티에서 유일하게 이름 부여되어야 함
- 엔터티 생성의미대로 이름을 부여 (적절한 엔터티명 부여)

3) 속성 (Attribute)

1. 속성 (Attribute)의 개념
-업무에서 필요로 하는 인스턴스에서 관리하고자 하는 더 이상 분리되지 않는 최소의 데이터 단위

2. 엔터티, 인스턴스, 속성, 속성값 내용과 표기법
- 한 개의 엔터티는 두 개 이상의 인스턴스의 집합
- 한 개의 엔터티는 두 개 이상의 속성
- 한 개의 속성은 한 개의 속성값

3. 속성의 특징 (3): **업무1 + SQL2**
- 업무에서 필요로 하는 정보
- 한 개의 속성은 한 개의 속성값만을 가짐
- 정해진 주식별자에 함수적 종속성을 가져야 함

4. 속성의 분류 (4)
- 속성의 특성에 따른 분류 (3)
- 엔터티의 구성방식에 따른 분류 (3)
- 단순형/ 복잡형
- 단일값/ 다중값

5. 도메인(Domain)
- 속성이 가질 수 있는 범위

6. 속성의 명명(Naming)
-해당 업무에서 사용하는 이름 부여
- 서술식 속성명 사용하지 않음
- 약어 사용 제한
- 전체 데이터 모델에서 유일성 확보

4) 관계 (Relationship)

1. 관계의 개념
- 관계의 정의
  : 인스턴스 사이의 논리적인 연관성으로 존재나 행위로서 서로에게 연관성이 부여된 상태
- 관계의 패어링
  : 인스턴스가 개별적으로 관계를 가지는 것

2. 관계의 분류
- 존재적 관계 ← 연관관계
- 행위적 관계 ← 의존관계

3. 관계의 표기법
- 관계명 (Membership)
- 관계차수 (Cardinality)
- 관계선택사양 (Optionality)

4. 관계의 정의 및 읽는 방법
- 관계 체크사항 (4)
- 관계 읽기

5) 식별자 (Identifiers)

1. 식별자(Identifiers) 개념
- 엔터티의 여러 개의 속성 중 엔터티를 대표할 수 있는 속성

2. (주)식별자의 특징: 존.유.최.불 (존나 유명한 최불암)

3. 식별자 분류 및 표기법
- 식별자 분류 (4) 대.대.속.스
: 대표성 여부/ 스스로 생성 여부/ 속성의 수/ 대체 여부
- 식별자 표기법

4. 주식별자 도출기준
- 해당 업무에서 자주 이용되는 속성으로 지정
- 명칭, 내역 등과 같이 이름으로 기술되는 것 지양
- 속성의 수가 많아지지 않도록 함

5. 식별자관계와 비식별자관계에 따른 식별자
- 식별자관계와 비식별자 관계의 결정
- 식별자관계
- 비식별자관계
- 식별자 관계로만 설정할 경우의 문제점
- 비식별자 관계로만 설정할 경우의 문제점

Chap2. 데이터 모델과 성능

1) 성능 데이터 모델링의 개요

1. 성능 데이터 모델링의 정의
- 데이터베이스의 성능향상을 목적으로 설계단계의 데이터 모델링 때부터 성능과 관련된 사항이 데이터 모델링에 반영될 수 있도록 하는 것

2. 성능 데이터 모델링 수행시점

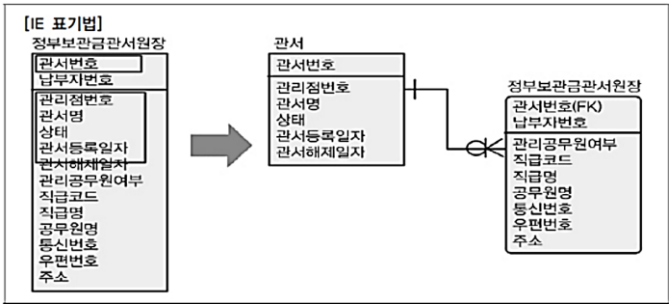
3. 성능 데이터 모델링 고려사항 (6)

2) 정규화와 성능

1. 정규화를 통한 성능 향상 전략
- 정규화: 데이터를 결정하는 결정자에 의해 함수적 종속을 가지고 있는 일반 속성을 의존자로 하여 입력/수정/삭제 이상을 제거하는 것

- 데이터 처리 성능: 조회성능 vs. 입력/수정/삭제 성능
  →반정규화 필요 →정규화 필요

2. 반정규화된 테이블의 성능저하 사례1
- 정규화로 오히려 데이터 조회 성능이 향상된 경우 (PK Unique Key)



"직급코드가 AA인 직급명, 관서번호, 관서명 조회"

- 반정규화

SELECT 직급명, 관서번호, 관서명
FROM 정부보관금관서원장
WHERE 직급코드 = “AA”;

- 정규화

SELECT B. 직급명, A.관서명, A.관서번호
FROM 관서 A, 정부보관금관서원장 B
WHERE 직급코드 = “AA”
AND A.관서번호 = B.관서번호 ;

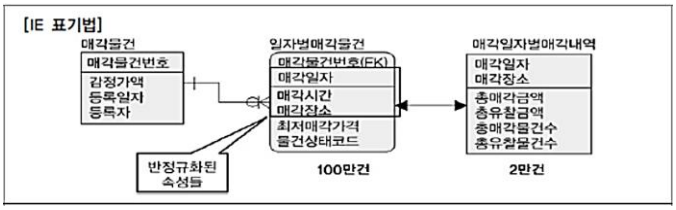
3. 반정규화된 테이블의 성능저하 사례2

- 2NF가 안되어 부분적인 함수적 종속성을 가진 경우

cf. 2NF: 모든 속성은 식별자에 의해 “완전종속”

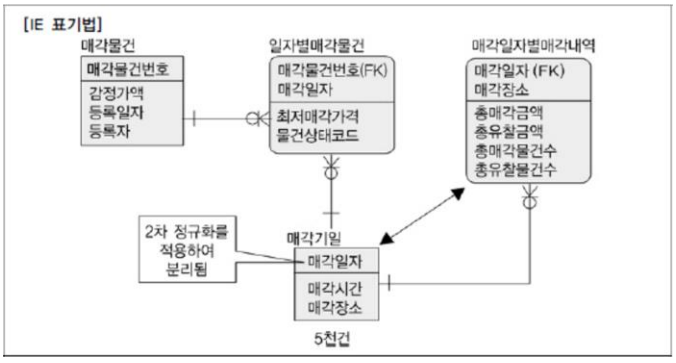
“서울 7호에서 매각된 총매각금액, 총유찰금액 산출”

〈반정규화〉



SELECT B.총매각금액, B.총유찰금액
FROM (**SELECT DISTINCT** 매각일자
FROM 일자별매각물건 **WHERE** 매각장소 = ’서울7호’) A, 매각일자별매각내역 B
WHERE A.매각일자 = B.매각일자
AND A.매각장소 = B.매각장소;

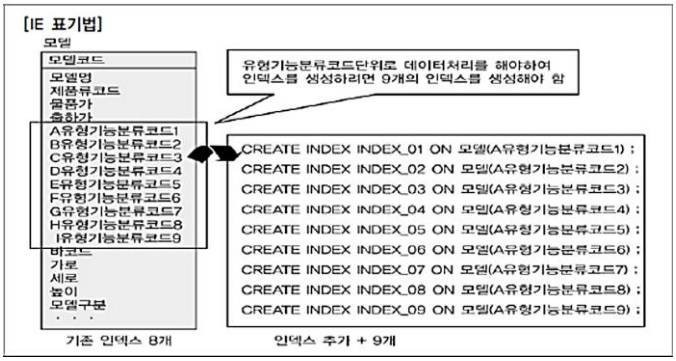
〈제 2 정규화〉

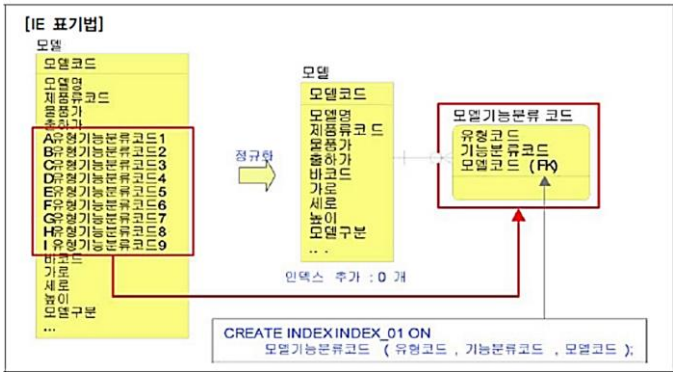


SELECT B.총매각금액, B.총유찰금액
FROM 매각기일 A, 매각일자별매각내역 B
WHERE A.매각장소 = ‘서울7호’
AND A.매각일자 = B.매각일자
AND A.매각장소 = B.매각장소 ;

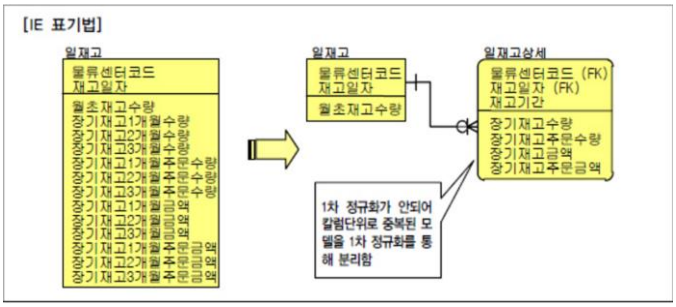
4. 반정규화된 테이블의 성능저하 사례3

-동일한 속성형식을 2개 이상의 속성으로 나열한 경우





5. 반정규화된 테이블의 성능저하 사례4



6. 함수적 종속성에 근거한 정규화 수행 필요

- 함수적 종속성: 데이터들이 어떤 기준값에 의해 종속되는 현상

종속자는 근본적으로 결정자에 함수적으로 종속성을 가지고 있음

이름, 출생지, 주소는 주민등록번호에 함수 종속성을 가지고 있음

- 정규화는 “함수적 종속성”에 근거하여 반복적 데이터를 분리하고, 데이터가 종속된 테이블에 적절하게 배치되도록 하는 것

3) 반정규화와 성능

1. 반정규화를 통한 성능향상 전략

- 반정규화의 정의:
  →협의: 데이터를 중복하여 성능을 향상시키기 위한 기법
  →광의: 성능을 향상시키기 위해 정규화된 데이터모델에서 중복, 통합, 분리 등을 수행하는 모든 과정
- 반정규화의 적용방법/순서 (3)

2. 반정규화의 기법
- 테이블 반정규화 (3)
- 칼럼 반정규화 (5)
- 관계 반정규화 (1)

3. 정규화된 데이터모델의 성능저하 사례1

4. 정규화된 데이터모델의 성능저하 사례2

4) 대량 데이터에 따른 성능

1. 대량 데이터발생에 따른 테이블 분할 개요
- 수평/ 수직 분할
- Row Chaining
- Row Migration

2. 한 테이블에 많은 수의 칼럼을 가지고 있는 경우
- 트랜잭션이 어떤 칼럼에 집중적으로 발생하는지 분석하여 1:1관계로 테이블을 쪼개줘야 함

3. 대량 데이터 저장 및 처리로 인해 성능
- Range Partition
- List Partition
- Hash Partition

※파티셔닝 기법은 반정규화가 아님!

4. 테이블에 대한 수평분할/수직분할의 절차 (4)

5) 데이터베이스 구조와 성능

1. 슈퍼타입/서브타입 모델의 성능고려 방법
- 슈퍼/ 서브타입 데이터 모델의 개요
- 슈퍼/ 서브타입 데이터 모델의 변환
- 슈퍼/ 서브타입 데이터 모델의 변환기술 (3)
- 슈퍼/ 서브타입 데이터 모델의 변환타입 비교

2. 인덱스 특성을 고려한 PK/FK DB 성능향상
- PK순서 고려 사항: 자주 쓰이는 순으로&
  범위가 좁은 순으로

3. FK제약 없을 시 인덱스 미생성으로 성능저하
- 물리적인 테이블에 FK 제약 걸었을 때: FK인덱스 반드시!
  물리적 테이블에 FK 제약 없을 시: FK인덱스 기본!
- Ex.

CREATE INDEX 수강신청_FK1 **ON**수강신청(학사기준번호)

6) 분산 데이터베이스와 성능

1. 분산 데이터베이스의 개요

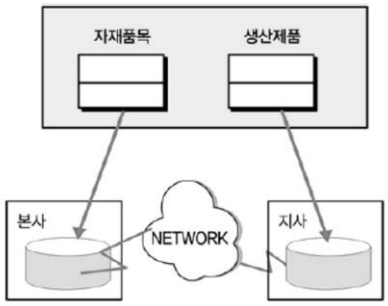
2. 분산 데이터베이스의 투명성(Transparency)
- 분산의 투명성
- 위치의 투명성
- 지역 사상의 투명성
- 중복의 투명성
- 장애의 투명성
- 병행의 투명성

3. 분산 데이터베이스의 적용 방법 및 장단점

4. 분산 데이터베이스의 활용 방향성

5. 데이터베이스 분산구성의 가치

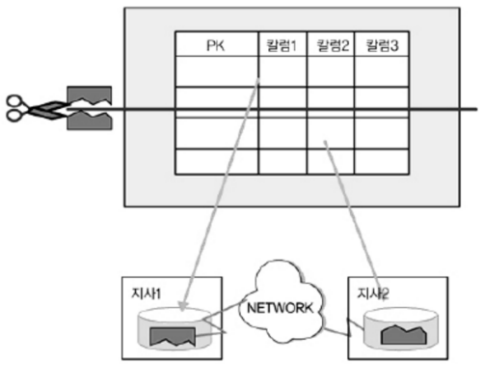
6. 분산 데이터베이스의 적용 기법
- 테이블 위치 분산



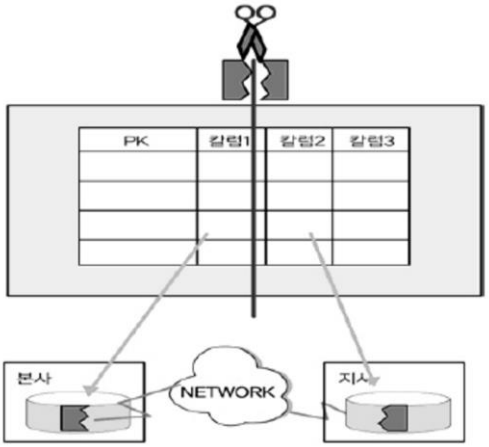
[그림 Ⅰ-2-41] 테이블별 위치 분산

위치 \ 테이블	자재품목	생산제품	협력회사	사원	부서
본사	•		•		•
지사		•		•	

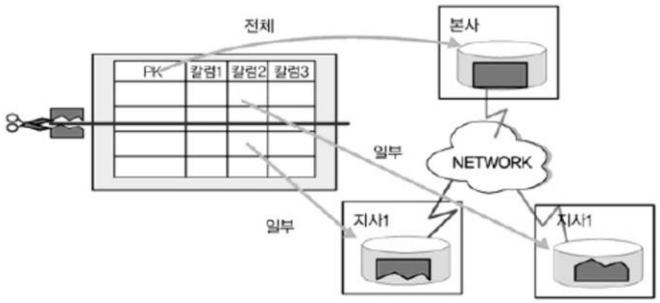
- 테이블 분할(Fragmentation) 분산
  - 수평분할



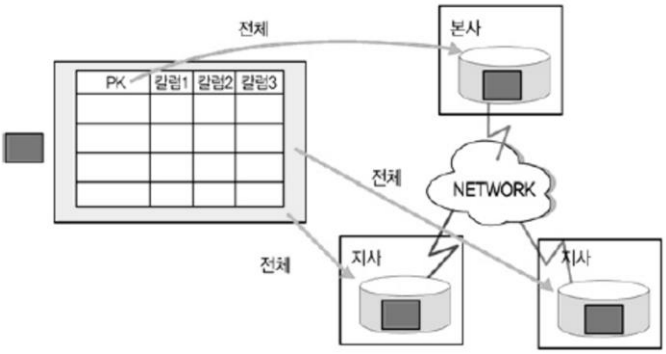
  - 수직분할



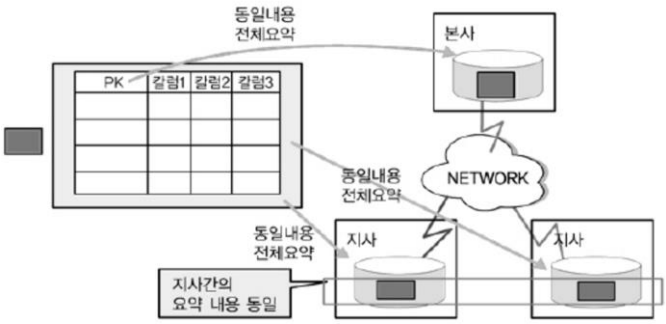
- 테이블 복제 (Replication) 분산
  - 부분복제 (Segment Replication)



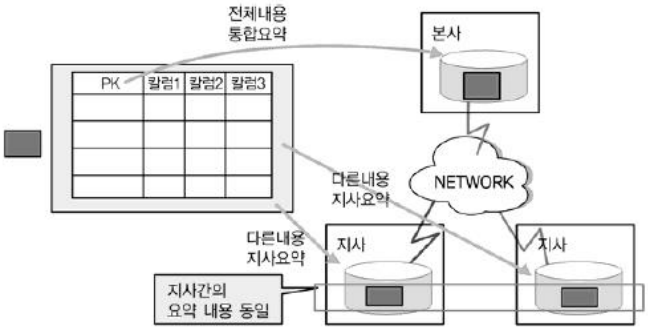
  - 광역복제 (Broadcast Replication)



- 테이블 요약 (Summarization) 분산
  - 분석 요약 (Rollup Relplication)



  -통합 요약 (Consolidation Replication)



7. 분산 데이터베이스를 적용하여 성능이 향상된 사례

<과목 II: SQL 기본 및 활용>

Chap1. SQL 기본

1) 관계형 데이터 베이스 개요

1. 데이터베이스
- 데이터베이스의 발전
- 관계형 데이터베이스

2. SQL(Structured Query Language)
- 명령어의 종류 (4)

3. TABLE
- 테이블 관계 용어: 정규화/ 기본키/ 외부키

4. ERD(Entity Relationship Diagram)

2) DDL (Data Definition Language)

1. 데이터 유형

2. CREATE TABLE
- 테이블과 컬럼 정의
- **CREATE TABLE**
- 제약조건(CONSTRAINT)
- 생성된 테이블 구조 확인: **DESC** 테이블명
- SELECT 문장을 통한 테이블 생성 사례: CTAS

3. ALTER TABLE 테이블 + 명령어
- **ADD** (칼럼명 데이터유형);
- **DROP COLUMN** 칼럼명;
- **MODIFY** (칼럼명1 데이터유형 **DEFAULT__ NOT NULL**);
- **RENAME COLUMN** 칼럼명1 **TO** 칼럼명2:
- **DROP CONSTRAINT** 제약조건명;
- **ADD CONSTRAINT** 제약조건명 제약조건 (칼럼명);

4. RENAME TABLE
- **RENAME** 테이블1 **TO** 테이블2

5. DROP TABLE
- **DROP TABLE** 테이블;

6. TRUNCATE TABLE
- **TRUNCATE TABLE** 테이블;

3) DML (Data Manipulation Language)

1. INSERT
- **INSERT INTO** 테이블 (칼럼명) **VALUES** (___);

2. UPDATE
- **UPDATE** 테이블 **SET** 칼럼명 = 값;

3. DELETE
- **DELETE (FROM)** 테이블 **WHERE ____**;

4. SELECT
- **SELECT (DISTINCT)** 칼럼명 **FROM** 테이블;
- DISTINCT 옵션
- WILDCARD 사용하기: *
- ALIAS 부여하기

5. 산술연산자와 합성연산자
-산술연산자: (), +, -, *, /
-합성연산자: ||
ex. **SELECT** PLAYER_NAME || '선수, ' || HEIGHT || 'cm, ' || WEIGHT || 'kg' 체격정보 **FROM** PLAYER;

[예제] Oracle

```
SELECT PLAYER_NAME || '선수,' || HEIGHT || 'cm,' || WEIGHT || 'kg' 체격정보
FROM   PLAYER;
```

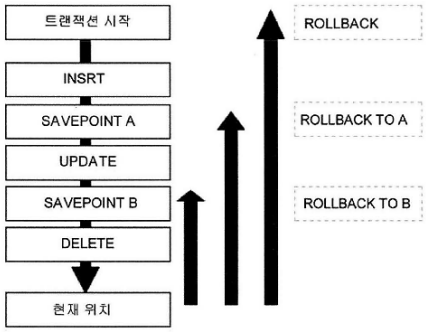
4) TCL (Transaction Control Language)

1. 트랜잭션 개요
- 트랜잭션의 정의
  : 논리적 작업 단위
- 트랜잭션의 특성 (4): 원. 일. 고. 지
- LOCKING 의 의미

2. COMMIT
- "변경된 데이터를 테이블에 영구적으로 반영해라!"
- COMMIT 이전 vs. 이후
- SQL server의 COMMIT(3)

3. ROLLBACK
- "변경된 데이터가 문제가 있으니 변경 전 데이터로 복귀해라!"
- SQL server의 ROLLBACK: **BEGIN TRAN → ROLLBACK**

4. SAVEPOINT
- "데이터변경을 사전에 지정한 저장점까지만 롤백하라!"
- **SAVEPOINT _____ → ROLLBACK TO ____;**



[그림 II-1-11] ROLLBACK 원리 (Oracle 기준)

5) WHERE 절

1. WHERE 조건절 개요

- **SELECT** 칼럼명 **FROM** 테이블 **WHERE** 조건식
- 조건식에 칼럼명/ 연산자/ 문자, 숫자 표현식 순으로 옴

2. 연산자의 종류 (5)

- 비교 연산자
- SQL 연산자
- 논리 연산자
- 부정 비교 연산자
- 부정 SQL 연산자
- 연산자의 우선순위: 괄호〉NOT〉 비교, SQL비교연산자〉AND〉 OR

3. 비교 연산자

비교 연산자	=	같다.
	〉	보다 크다.
	〉=	보다 크거나 같다.
	〈	보다 작다.
	〈=	보다 작거나 같다.

4. SQL 연산자

SQL 연산자	BETWEEN a AND b	a와 b의 값 사이에 있으면 된다.(a와 b 값이 포함됨)
	IN (list)	리스트에 있는 값 중에서 어느 하나라도 일치하면 된다.
	LIKE '비교문자열'	비교문자열과 형태가 일치하면 된다.(%, _ 사용)
	IS NULL	NULL 값인 경우

5. 논리 연산자

논리 연산자	AND	앞에 있는 조건과 뒤에 오는 조건이 참(TRUE)이 되면 결과도 참(TRUE)이 된다. 즉, 앞의 조건과 뒤의 조건을 동시에 만족해야 한다.
	OR	앞의 조건이 참(TRUE)이거나 뒤의 조건이 참(TRUE)이 되어야 결과도 참(TRUE)이 된다. 즉, 앞뒤의 조건 중 하나만 참(TRUE)이면 된다.
	NOT	뒤에 오는 조건에 반대되는 결과를 되돌려 준다.

6. 부정 연산자

부정 비교 연산자	!=	같지 않다.
	^=	같지 않다.
	〈〉	같지 않다.(ISO 표준, 모든 운영체제에서 사용 가능)
	NOT 칼럼명 =	~와 같지 않다.
	NOT 칼럼명 〉	~보다 크지 않다.
부정 SQL 연산자	NOT BETWEEN a AND b	a와 b의 값 사이에 있지 않다. (a, b 값을 포함하지 않는다)
	NOT IN (list)	list 값과 일치하지 않는다.
	IS NOT NULL	NULL 값을 갖지 않는다.

7. ROWNUM, TOP 사용

- 테이블이나 집합에서 원하는 만큼의 행만 가져오고 싶을 때 사용
- **WHERE ROWNUM** = 1, 〈2, 〈=1: 행 1개만 반환
- **WHERE ROWNUM** 〈N+1, 〈=N: 행 N개 반환

6) 함수 (FUNCTION)

1. 내장함수 (Built-In Function) 개요

- SQL 제공 함수의 종류
- 단일행 함수의 특징 (5)

2. 문자형 함수 (10)

- **LOWER** (문자열): 소문자
- **UPPER** (문자열): 대문자
- **ASCII** (문자): 문자→ASCII 부호
- **CHR** (ASCII 번호): ASCII 부호 → 문자
- **CONCAT** (문자열1, 문자열2): 이어 붙이기
- **SUBSTR** (문자열, m, n): m번째부터 n개 문자 잘라내기
- **LENGTH** (문자열): 문자열 길이
- **LTRIM** (문자열, 지정문자): 맨 왼쪽의 지정문자 지우기
- **RTRIM** (문자열, 지정문자): 맨 오른쪽의 지정문자 지우기
- **TRIM** ([**leading | trailing | both**] 지정문자 **FROM** 문자열)

문자형 함수 사용	결과 값 및 설명
LOWER('SQL Expert')	'sql expert'
UPPER('SQL Expert')	'SQL EXPERT'
ASCII('A')	65
CHR(65) / CHAR(65)	'A'
CONCAT('RDBMS',' SQL') 'RDBMS' || ' SQL' / 'RDBMS' + ' SQL'	'RDBMS SQL'
SUBSTR('SQL Expert', 5, 3) SUBSTRING('SQL Expert', 5, 3)	'Exp'
LENGTH('SQL Expert') / LEN('SQL Expert')	10
LTRIM('xxxYYZZxYZ','x') RTRIM('XXYYzzXYzz','z') TRIM('x' FROM 'xxYYZZxYZxx')	'YYZZxYZ' 'XXYYzzXY' 'YYZZxYZ'
RTRIM('XXYYZZXYZ ') → 공백 제거 및 CHAR와 VARCHAR 데이터 유형을 비교할 때 용이하게 사용된다.	'XXYYZZXYZ'

- **SELECT** 문자열함수 () **FROM DUAL**;
- DUAL 테이블: 더미테이블, X라는 행 가짐, 함수에 꼭 필요

3. 숫자형 함수 (9)

- **ABS** (숫자): 절대값
- **SIGN** (숫자): 부호
- **MOD** (숫자1, 숫자2): 나머지
- **CEIL** (숫자): 올림
- **FLOOR** (숫자): 내림
- **ROUND** (숫자, m): 반올림
- **TRUNC** (숫자, m): 버림
- **SIN, COS, TAN:** 삼각함수
- **EXP**(), **POWER**(), **SQRT**(), **LOG**(), **LN**()

숫자형 함수 사용	결과 값 및 설명
ABS(-15)	15
SIGN(-20) SIGN(0) SIGN(+20)	-1 0 1
MOD(7,3) / 7%3	1
CEIL(38.123) / CEILING(38.123) CEILING(-38.123)	39 39 -38
FLOOR(38.123) FLOOR(-38.123)	38 -39
ROUND(38.5235, 3) ROUND(38.5235, 1) ROUND(38.5235, 0) ROUND(38.5235)	38.524 38.5 39 39 (인수 0이 Default)
TRUNC(38.5235, 3) TRUNC(38.5235, 1) TRUNC(38.5235, 0) TRUNC(38.5235)	38.523 38.5 38 38 (인수 0이 Default)

4. 날짜형 함수 (3)

- **SYSDATE**: 현재 날짜, 시각
- **EXTRACT** (**YEAR | MONTH | DAY FROM** 날짜컬럼명)
- **TO_NUMBER**(**TO_CHAR**(날짜컬럼명, **'YYYY'| 'MM'| 'DD'**))

5. 변환형 함수 (3)

- **TO_NUMBER** (문자열)
- **TO_CHAR** (숫자|날짜, 'format')
- **TO_DATE** (문자열, 'format')

6. CASE 표현 (2)

- Simple-Case Expression

```
SELECT
  CASE 칼럼명
  WHEN 칼럼 값 THEN 반환 값
  WHEN 칼럼 값 THEN 반환 값
  ELSE 반환값
  END AS 반환값의 칼럼명
FROM 테이블명;
```

-Searched-Case Expression

```
SELECT
  CASE
  WHEN 칼럼명 = 칼럼 값 THEN 반환 값
  WHEN 칼럼명 = 칼럼 값 THEN 반환 값
  ELSE 반환값
  END AS 반환값의 칼럼명
FROM 테이블명;
```

7. NULL 관련 함수 (3)

- NVL/ ISNULL 함수

NVL (표현식1, 표현식2): NULL → 특정 값 대체
표현식1의 결과값이 NULL이면 표현식2 출력
(표현식1과 2의 결과값의 데이터 타입 같아야 함)

- NULL 과 공집합

공집합: 조건에 맞는 데이터가 한 건도 없는 경우
→ 집계함수 사용하여 공집합을 NULL값으로 바꿔줌
→ NVL로 공집합을 다른 값으로 치환

- NULLIF: 특정 값 → NULL 대체

NULLIF (표현식1, 표현식2)
표현식1이 표현식2와 같으면 NULL, 아니면 표현식1 리턴

- 기타 NULL 관련 함수 (COALESCE)

COALESCE (표현식1, 표현식2, ……):
임의의 개수 표현식에서 NULL이 아닌 최초의 표현식 리턴

7) GROUP BY, HAVING 절

1. 집계 함수 (Aggregate Function)

- 집계함수의 특성 (3)
- **COUNT**(*): NULL 값 포함 행의 수
- **COUNT** (표현식): 표현식에서 NULL값 제외한 행의 수
- **SUM** (DISTINCT| ALL 표현식): NULL 제외 합계
- **AVG** (DISTINCT| ALL 표현식): NULL 제외 평균
- **MAX** (DISTINCT| ALL 표현식): 최대값
- **MIN** (DISTINCT| ALL 표현식): 최소값
- **STDDEV**(DISTINCT| ALL 표현식): 표준편차
- **VARIAN** (DISTINCT| ALL 표현식): 분산

2. GROUP BY 절

- 중요한 특징 (3)

```
SELECT [DISTINCT] 칼럼명 [ALIAS명]
FROM 테이블명 [WHERE 조건식]
GROUP BY 칼럼이나 표현식 ;
```

3. HAVING 절

- WHERE 절 대신에 오는 조건절로, 집계함수를 이용한 조건 표시 가능

```
SELECT [DISTINCT] 칼럼명 [ALIAS명]
FROM 테이블명
GROUP BY 칼럼이나 표현식
HAVING 그룹조건식;
```

← GROUP BY 앞에 와도 무관

[예제]

```
SELECT   POSITION 포지션, ROUND(AVG(HEIGHT),2) 평균키
FROM     PLAYER
GROUP    BY POSITION
HAVING   AVG(HEIGHT) >= 180;
```

4. CASE 표현을 활용한 월별 데이터 집계

[예제] 부서별로 월별 입사자의 평균 급여 구하기

[결과]

DEPTNO	M01	M02	M03	M04	M05	M06	M07	M08	M09	M10	M11	M12
30	0	1425	0	0	2850	0	0	0	1375	0	0	950
20	0	0	0	2975	0	0	2050	0	0	0	0	1900
10	1300	0	0	0	0	2450	0	0	0	0	5000	0

[쿼리]

```
SELECT DEPTNO,
NVL(AVG(CASE MONTH WHEN 1 THEN SAL END),0) M01,
NVL(AVG(CASE MONTH WHEN 2 THEN SAL END),0) M02,
NVL(AVG(CASE MONTH WHEN 3 THEN SAL END),0) M03,
NVL(AVG(CASE MONTH WHEN 4 THEN SAL END),0) M04,
NVL(AVG(CASE MONTH WHEN 5 THEN SAL END),0) M05,
NVL(AVG(CASE MONTH WHEN 6 THEN SAL END),0) M06,
NVL(AVG(CASE MONTH WHEN 7 THEN SAL END),0) M07,
NVL(AVG(CASE MONTH WHEN 8 THEN SAL END),0) M08,
NVL(AVG(CASE MONTH WHEN 9 THEN SAL END),0) M09,
NVL(AVG(CASE MONTH WHEN 10 THEN SAL END),0) M10,
NVL(AVG(CASE MONTH WHEN 11 THEN SAL END),0) M11,
NVL(AVG(CASE MONTH WHEN 12 THEN SAL END),0) M12
FROM (SELECT ENAME, DEPTNO, EXTRACT(MONTH FROM HIREDATE) MONTH, SAL FROM EMP)
GROUP BY DEPTNO;
```

5. 집계함수와 NULL 처리

[예제] 팀별 포지션별 FW, MF, DF, GK 포지션의 인원수와 팀별 전체 인원수를 구하는 문장

[결과]

	TEAM_ID	FW	MF	DF	GK	SUM
1	K06	11	11	20	4	46
2	K14	0	1	1	0	2
3	K13	1	0	1	1	3
4	K15	1	1	1	0	3
5	K04	13	11	18	4	46

[쿼리]

```
SELECT TEAM_ID,
NVL(SUM(CASE POSITION WHEN 'FW' THEN 1 END), 0) FW,
NVL(SUM(CASE POSITION WHEN 'MF' THEN 1 END), 0) MF,
NVL(SUM(CASE POSITION WHEN 'DF' THEN 1 END), 0) DF,
NVL(SUM(CASE POSITION WHEN 'GK' THEN 1 END), 0) GK,
COUNT(*) SUM
FROM PLAYER
GROUP BY TEAM_ID;
```

8) ORDER BY 절

1. ORDER BY 정렬

- 조회된 데이터를 특정 칼럼을 기준으로 정렬하는 함수
- **ORDER BY** + 칼럼명 /ALIAS/ 칼럼번호

2. SELECT 문장 실행 순서

[형식]

5. SELECT 칼럼명 [ALIAS 명]
1. FROM 테이블명
2. WHERE 조건식
3. GROUP BY 칼럼 | 표현식
4. HAVING 그룹조건식
6. ORDER BY 칼럼 | 표현식

[실행 순서]

1. FROM: 발췌 대상 테이블을 참조
2. WHERE: 발췌 대상 데이터 아닌 것 제거
3. GROUP BY: 행들을 소그룹화
4. HAVING: 그룹핑된 값의 조건에 맞는 것만 출력
5. SELECT: 데이터 값을 출력, 계산
6. ORDER BY: 데이터 정렬

3. TOP N 쿼리

- Oracle에서 순위가 높은 N개의 로우 추출하려면
➔ 인라인 뷰로 데이터 정렬 후 [ORDER BY]
메인쿼리에서 ROWNUM! [WHERE절]

[예제] 급여가 상위인 3명 출력

```
SELECT ENAME, SAL FROM EMP
WHERE ROWNUM< 4
ORDER BY SAL DESC;
```

→ 무작위로 추출된 3명에 한해 급여를 내림차순
(**ROWNUM → ORDER BY**)

```
SELECT ENAME, SAL
FROM(SELECT SAL, ENAME FROM EMP ORDER BY SAL DESC)
WHERE ROWNUM <4;
```

→ 인라인 뷰에서 **ORDER BY**로 데이터 정렬해 급여 높은 순으로 뽑고, **ROWNUM**으로 TOP 3 뽑음

9) 조인(JOIN)

1. JOIN 개요

- JOIN: 두 개 이상의 테이블을 연결 또는 결합하여 데이터를 출력하는 것
- 3개 이상의 테이블을 조인해도 SQL에서 데이터를 처리할 때에는 단 두 개의 집합 간에만 조인이 일어남

2. EQUI JOIN

- EQUI JOIN 정의:
두 개의 테이블 간에 칼럼 값들이 서로 정확하게 일치
주로, PK↔FK 관계

SELECT 테이블1.칼럼명, 테이블2. 칼럼명
FROM 테이블1, 테이블2
WHERE 테이블1.칼럼명1 = 테이블2.칼럼명2; (←조인조건)
→ INNER JOIN 이용해도 가능
- 테이블에 ALIAS 사용했다면 WHERE절과 SELECT절에는 무조건 테이블의 ALIAS로 기술해야 함

[예제] 선수테이블과 팀테이블에서 선수 이름과 소속팀 이름 출력

```
SELECT PLAYER.PLAYER_NAME, TEAM.TEAM_NAME
FROM PLAYER, TEAM
WHERE PLAYER.TEAM_ID = TEAM.TEAM_ID;
```

```
SELECT PLAYER.PLAYER_NAME, TEAM.TEAM_NAME
FROM PLAYER INNER JOIN TEAM
ON PLAYER.TEAM_ID = TEAM.TEAM_ID;
```

[예제] 선수 이름, 백넘버, 팀명, 연고지 출력

```
SELECT P.PLAYER_NAME 선수명, P.BACK_NO 백넘버,
T.TEAM_NAME 팀명, T.REGION_NAME 연고지
FROM PLAYER P INNER JOIN TEAM T
ON P.TEAM_ID = T.TEAM_ID;
```

```
SELECT P.PLAYER_NAME 선수명, P.BACK_NO 백넘버,
T.TEAM_NAME 팀명, T.REGION_NAME 연고지
FROM PLAYER P, TEAM T
WHERE P.TEAM_ID = T. TEAM_ID;
```

[예제] 위 SQL문장의 WHERE 절에 포지션이 골키퍼('GK') 인 선수들에 대한 데이터만을 백넘버 순으로 출력

```
SELECT P.PLAYER_NAME 선수명, P.BACK_NO 백넘버,
T.REGION_NAME 연고지, T.TEAM_NAME 팀명
FROM PLAYER P, TEAM T
WHERE P.TEAM_ID = T. TEAM_ID
AND P.POSITION = 'GK'
ORDER BY P.BACK_NO;
```

```
SELECT P.PLAYER_NAME 선수명, P.BACK_NO 백넘버,
T.REGION_NAME 연고지, T.TEAM_NAME 팀명
FROM PLAYER P INNER JOIN TEAM T
ON P.TEAM_ID = T.TEAM_ID
WHERE P.POSITION = 'GK'
ORDER BY P.BACK_NO;
```

[예제] 팀테이블과 구장테이블 이용하여 소속팀이 가지고 있는 전용구장의 정보를 팀의 정보와 함께 출력

```
SELECT T.REGION_NAME,T.TEAM_NAME,T.STADIUM_ID,
S.STADIUM_NAME, S.SEAT_COUNT
FROM TEAM T, STADIUM S
WHERE T.STADIUM_ID = S.STADIUM_ID;
```

```
SELECT T.REGION_NAME,T.TEAM_NAME,T.STADIUM_ID,
S.STADIUM_NAME, S.SEAT_COUNT
FROM TEAM T INNER JOIN STADIUM S
ON T.STADIUM_ID = S.STADIUM_ID;
```

3. Non EQUI JOIN

- 두 개의 테이블 간에 칼럼값들이 서로 정확하게 일치하지 않는 경우
- '=' 대신 다른 연산자 사용: BETWEEN, 〉, 〉=, 〈, 〈= 등

SELECT 테이블1.칼럼명, 테이블2. 칼럼명
FROM 테이블1, 테이블2
WHERE 테이블1.칼럼명1
BETWEEN 테이블2.칼럼명1 **AND** 테이블2.칼럼명2;

[예제] EMP테이블과 SAL GRADE 테이블을 이용해 어떤 사원이 받고 있는 급여가 어느 등급에 속하는지 구하기

```
SELECT E.ENAME 사원명, E.SAL 급여, S.GRADE 급여등급
FROM EMP E, SALGRADE S
WHERE E.SAL BETWEEN S.LOSAL AND S.HISAL;
```

4. 3개 이상 TABLE JOIN

[예제] 선수,팀,운동장 테이블 JOIN하기

```
SELECT P.PLAYER_NAME 선수명, P.POSITION 포지션,
T.REGION_NAME 연고지, T.TEAM_NAME 팀명, S.STADIUM_NAME 구장명
FROM PLAYER P, TEAM T, STADIUM S
WHERE P.TEAM_ID = T.TEAM_ID
AND T.STADIUM_ID = S.STADIUM_ID
ORDER BY 선수명;
```

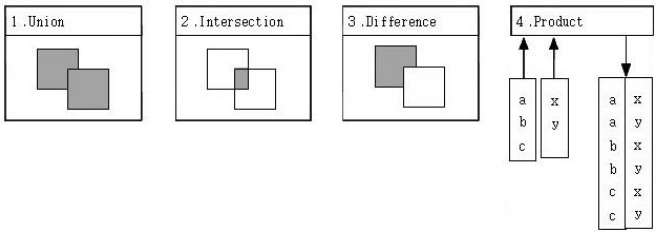
```
SELECT P.PLAYER_NAME 선수명, P.POSITION 포지션,
T.REGION_NAME 연고지, T.TEAM_NAME 팀명, S.STADIUM_NAME 구장명
FROM PLAYER P INNER JOIN TEAM T
ON P.TEAM_ID = T.TEAM_ID
INNER JOIN STADIUM S
ON T.STADIUM_ID = S.STADIUM_ID
ORDER BY 선수명;
```

Chap2. SQL 활용

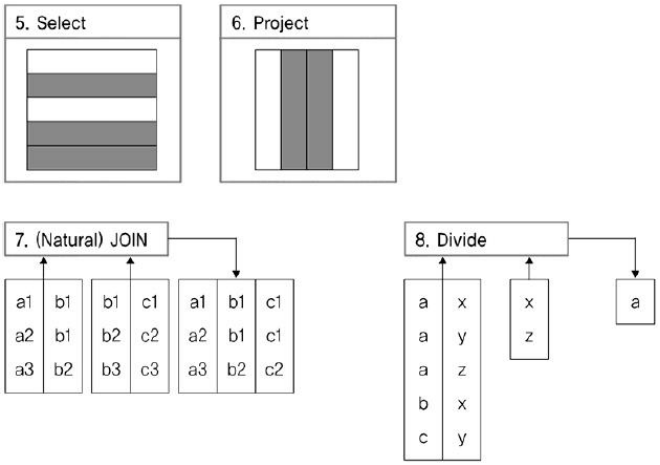
1) 표준 조인 (Standard Join)

1. STANDARD SQL 개요

- 일반 집합 연산자(4)



- 순수 관계 연산자(4)



2. FROM 절 JOIN 형태

- WHERE절을 사용하던 기존 JOIN 방식과 차이를 보임
  → FROM 절에 JOIN 종류를 명시 &
  ON 조건절에 JOIN 조건 명시
- INNER JOIN/ NATURAL JOIN/ USING 조건절/ ON조건절 / CROSS JOIN/ OUTER JOIN

3. INNER JOIN

[예제] 사원 번호, 사원명, 소속부서 코드, 소속부서 이름

```
SELECT E.EMPNO 사원번호, E.ENAME 사원이름,
D.DEPTNO 소속부서코드, D.DNAME 소속부서명
FROM EMP E INNER JOIN DEPT D
ON E.DEPTNO = D.DEPTNO;
```

4. NATURAL JOIN

- 두 테이블 간의 동일한 이름을 갖는 "모든" 칼럼에 대해 EQUI JOIN 수행
- JOIN에 사용된 칼럼은 같은 데이터 유형이어야 함
- ALIAS, 테이블명 접두사 붙이지 않음

```
SELECT DEPTNO, EMPNO, ENAME, DNAME
FROM EMP NATURAL JOIN DEPT;
SELECT EMP.DEPTNO, EMPNO, ENAME, DNAME
FROM EMP NATURAL JOIN DEPT;
```

→2번째쿼리오류

- NATURAL JOIN과 INNER JOIN의 차이점:

와일드카드 (*)을 사용하여 JOIN 수행 시, 전자는 공통칼럼을 하나로 합쳐주지만 후자는 각각 출력

[예제]DEPT와 DEPT_TEMP 테이블

i. NATURAL JOIN 수행결과: 공통된 칼럼을 하나만 출력

```
SELECT * FROM DEPT NATURAL JOIN DEPT_TEMP;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
40	OPERATIONS	BOSTON

ii. INNER JOIN 수행결과: 공통된 칼럼 모두 출력

```
SELECT * FROM DEPT INNER JOIN DEPT_TEMP
ON DEPT.DEPTNO = DEPT_TEMP.DEPTNO
AND DEPT.DNAME = DEPT_TEMP.DNAME
AND DEPT.LOC = DEPT_TEMP.LOC;
```

DEPTNO	DNAME	LOC	DEPTNO_1	DNAME_1	LOC_1
10	ACCOUNTING	NEW YORK	10	ACCOUNTING	NEW YORK
40	OPERATIONS	BOSTON	40	OPERATIONS	BOSTON

5. USING 조건절

- 원하는 칼럼에 대해서 선택적으로 EQUI JOIN을 할 경우
- [INNER] JOIN 후 USING 절을 써주면, USING 절에 쓰인 칼럼은 공통칼럼으로 처리&나머지는 2개 칼럼으로 표시됨
- ALIAS, 테이블명 접두사 쓸 수 없음

[예제]DEPT, DEPT_TEMP테이블을 DEPTNO 칼럼을 이용한 INNER JOIN의 USING조건절

```
SELECT * FROM DEPT INNER JOIN DEPT_TEMP
USING (DEPTNO);
```

DEP...	DNAME	LOC	DNAME_1	LOC_1
10	ACCOUNTING	NEW YORK	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS	R&D	DALLAS
30	SALES	CHICAGO	MARKETING	CHICAGO
40	OPERATIONS	BOSTON	OPERATIONS	BOSTON

6. ON 조건절

- JOIN 서술부 (ON 조건절)과 비서술부 (WHERE절) 구분하기 좋음
- ON 조건절을 사용하는 경우 (3)
- ALIAS, 테이블명 접두사 써줘야 함

ON + WHERE

[예제] 부서코드30인 부서의 소속사원 이름, 소속부서 코드, 부서코드, 부서이름 찾기

```
SELECT E.ENAME, E.DEPTNO, D.DEPTNO, D.DNAME
FROM EMP E JOIN DEPT D
ON E.DEPTNO = D.DEPTNO
WHERE E.DEPTNO = 30;
```

ON에 데이터 검색조건 추가

[예제] 매니저 사원번호가 7698번인 사원들의 이름, 소속부서 코드, 부서 이름 찾기

```
SELECT E.ENAME, E.MGR, D.DEPTNO, D.DNAME
FROM EMP E JOIN DEPT D
ON E.DEPTNO= D.DEPTNO
AND E.MGR = 7698;
```

ON 조건절 예제

[예제] 팀이름, 스타디움 ID, 스타디움이름 찾기

i. ON 조건절 이용

```
SELECT TEAM_NAME, T.STADIUM_ID, STADIUM_NAME
FROM TEAM T JOIN STADIUM S
ON T.STADIUM_ID = S.STADIUM_ID
ORDER BY STADIUM_ID;
```

ii. USING 조건절 이용

```
SELECT TEAM_NAME, STADIUM_ID, STADIUM_NAME
FROM TEAM JOIN STADIUM
USING (STADIUM_ID)
ORDER BY STADIUM_ID;
```

iii. WHERE절 이용

```
SELECT TEAM_NAME, T.STADIUM_ID, STADIUM_NAME
FROM TEAM T, STADIUM S
WHERE T.STADIUM_ID = S.STADIUM_ID
ORDER BY STADIUM_ID;
```

[예제] 팀ID로 조인하여 팀이름, 팀ID, 스타디움이름 찾기

i. ON 조건절 이용

```
SELECT T.TEAM_NAME, T.TEAM_ID, S.STADIUM_NAME
FROM TEAM T INNER JOIN STADIUM S
ON T.TEAM_ID = S.HOMETEAM_ID
ORDER BY TEAM_ID;
```

ii. USING 조건절 이용

: STADIUM에는 팀ID가 HOMETEAM_ID라는 다른 이름의 칼럼으로 표시되어 사용불가

iii. WHERE절 이용

```
SELECT T.TEAM_NAME, T.TEAM_ID, S.STADIUM_NAME
FROM TEAM T, STADIUM S
WHERE T.TEAM_ID = S.HOMETEAM_ID
ORDER BY TEAM_ID;
```

다중 테이블 JOIN

[예제] 사원과 DEPT 테이블의 소속부서명, DEPT_TEMP 테이블의 바뀐 부서명 정보 출력

i. ON 조건절 이용

```
SELECT E.EMPNO, D1.DEPTNO,D1.DNAME, D2.DNAME NEW_DNAME
FROM EMP E JOIN DEPT D1
ON E.DEPTNO = D1.DEPTNO
JOIN DEPT_TEMP D2
ON E.DEPTNO = D2.DEPTNO
ORDER BY EMPNO;
```

ii. WHERE절 이용

```
SELECT E.EMPNO, D1.DEPTNO, D1.DNAME, D2.DNAME NEW_DNAME
FROM EMP E, DEPT D1, DEPT_TEMP D2
WHERE E.DEPTNO = D1.DEPTNO
AND E.DEPTNO = D2.DEPTNO
ORDER BY EMPNO;
```

[예제] 'GK'포지션의 선수별 연고지명, 팀명, 구장명 출력

i. ON 조건절 이용

```
SELECT P.PLAYER_NAME 선수명, T.REGION_NAME 연고지명,
T.TEAM_NAME 팀명, S.STADIUM_NAME 구장명
FROM PLAYER P JOIN TEAM T
ON P.TEAM_ID = T.TEAM_ID
JOIN STADIUM S
ON T.STADIUM_ID = S.STADIUM_ID
WHERE P.POSITION = 'GK'
ORDER BY 선수명;
```

ii. WHERE절 이용

```
SELECT P.PLAYER_NAME 선수명, T.REGION_NAME 연고지명,
T.TEAM_NAME 팀명, S.STADIUM_NAME 구장명
FROM PLAYER P, TEAM T, STADIUM S
WHERE P.TEAM_ID = T.TEAM_ID
AND T.STADIUM_ID = S.STADIUM_ID
AND P.POSITION = 'GK'
ORDER BY 선수명;
```

[예제]홈팀이 3점 이상 차이로 승리한 경기의 경기장 이름, 경기 일정, 홈팀 이름과 원정팀 이름 정보

i. ON 조건절 이용

```
SELECT S.STADIUM_NAME, S.STADIUM_ID, SC.SCHE_DATE, HT.TEAM_NAME,
AT.TEAM_NAME, SC.HOME_SCORE, SC.AWAY_SCORE
FROM SCHEDULE SC JOIN STADIUM S
ON SC.STADIUM_ID = S.STADIUM_ID
JOIN TEAM HT
ON SC.HOMETEAM_ID = HT.TEAM_ID
JOIN TEAM AT
ON SC.AWAYTEAM_ID = AT.TEAM_ID
WHERE SC.HOME_SCORE-SC.AWAY_SCORE>=3;
```

ii. WHERE절 이용

```
SELECT S.STADIUM_NAME, S.STADIUM_ID, SC.SCHE_DATE, HT.TEAM_NAME,
AT.TEAM_NAME, SC.HOME_SCORE, SC.AWAY_SCORE
FROM SCHEDULE SC, STADIUM S, TEAM HT, TEAM AT
WHERE SC.STADIUM_ID = S.STADIUM_ID
AND SC.HOMETEAM_ID = HT.TEAM_ID
AND SC.AWAYTEAM_ID = AT.TEAM_ID
AND SC.HOME_SCORE-SC.AWAY_SCORE>=3;
```

7. CROSS JOIN

- 생길 수 있는 모든 데이터의 조합 출력

[예제] 사원명, 소속부서명 모든 조합 찾기

```
SELECT ENAME, DNAME
FROM EMP CROSS JOIN DEPT
ORDER BY ENAME;
```

8. OUTER JOIN

- LEFT OUTER JOIN: 테이블 A 기준 모든 정보

[예제] STADIUM과 TEAM을 조인하되 홈팀이 없는 경기장 정보도 같이 출력

```
SELECT S.STADIUM_NAME, S.STADIUM_ID,
S.SEAT_COUNT,S.HOMETEAM_ID,T.TEAM_NAME
FROM STADIUM S LEFT JOIN TEAM T
ON S.HOMETEAM_ID = T.TEAM_ID
ORDER BY HOMETEAM_ID;
```

- RIGHT OUTER JOIN: 테이블 B 기준 모든 정보

[예제]DEPT와 EMP를 조인하되 사원이 없는 부서정보도 같이 출력

```
SELECT E.ENAME, D.DEPTNO, D.DNAME, D.LOC
FROM EMP E RIGHT JOIN DEPT D
ON E.DEPTNO = D.DEPTNO;
```

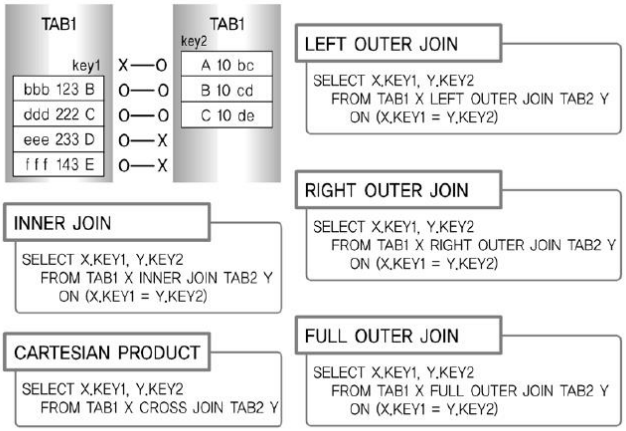
- FULL OUTER JOIN: 테이블 ALL 모든 정보 (중복 X)

[예제] DEPTNO를 기준으로 DEPT와 DEPT_TEMP 데이터를 FULL OUTER JOIN 하기

```
SELECT *
FROM DEPT FULL JOIN DEPT_TEMP
ON DEPT.DEPTNO = DEPT_TEMP.DEPTNO
ORDER BY DEPT.DEPTNO;
```

```
SELECT *
FROM DEPT L LEFT JOIN DEPT_TEMP R
ON L.DEPTNO = R.DEPTNO
UNION
SELECT *
FROM DEPT L RIGHT JOIN DEPT_TEMP R
ON L.DEPTNO = R.DEPTNO;
```

9. INNER vs. OUTER vs. CROSS JOIN 비교



[그림 Ⅱ-2-4] INNER vs OUTER vs CROSS JOIN 문장 비교

2) 집합 연산자 (Set Operator)

- 집합 연산자의 종류 (4)

집합 연산자	연산자의 의미
UNION	여러 개의 SQL문의 결과에 대한 합집합으로 결과에서 모든 중복된 행은 하나의 행으로 만든다.
UNION ALL	여러 개의 SQL문의 결과에 대한 합집합으로 중복된 행도 그대로 결과로 표시된다. 즉, 단순히 결과만 합쳐놓은 것이다. 일반적으로 여러 질의 결과가 상호 배타적인(Exclusive)일 때 많이 사용한다. 개별 SQL문의 결과가 서로 중복되지 않는 경우, UNION과 결과가 동일하다. (결과의 정렬 순서에는 차이가 있을 수 있음)
INTERSECT	여러 개의 SQL문의 결과에 대한 교집합이다. 중복된 행은 하나의 행으로 만든다.
EXCEPT	앞의 SQL문의 결과에서 뒤의 SQL문의 결과에 대한 차집합이다. 중복된 행은 하나의 행으로 만든다. (일부 데이터베이스는 MINUS를 사용함)

[예제] 삼성블루윙즈팀 OR 전남드래곤즈팀인 선수

i. UNION 연산자 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
UNION
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K07';
```

ii. OR/ IN () 연산자 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02' OR TEAM_ID = 'K07';
```

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID IN ('K02','K07');
```

[예제] 삼성블루윙즈팀 OR 포지션이 GK인 선수

i. UNION 연산자 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
UNION
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE POSITION = 'GK';
```

ii. OR 연산자 이용 (IN() 은 불가)

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02' OR POSITION = 'GK';
```

cf. UNION ALL 쿼리 및 결과

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
UNION ALL
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE POSITION = 'GK'
ORDER BY 1,2,3,4,5;
```

TEA...	PLAYER_NAME	POSITION	BACK_NO	HEIGHT
K02	김운재	GK	1	182
K02	김운재	GK	1	182

→ K02에서 GK인 사람은 중복출력됨

[예제] 포지션별 평균키, 팀별 평균키

```
SELECT 'P' 구분코드, POSITION 포지션, ROUND(AVG(HEIGHT),1) 평균키
FROM PLAYER
GROUP BY POSITION
UNION
SELECT 'T' 구분코드, TEAM_ID 팀코드, ROUND(AVG(HEIGHT),1) 평균키
FROM PLAYER
GROUP BY TEAM_ID
ORDER BY 1;
```

[예제] 삼성블루윙즈팀이면서 포지션이 MF가 아닌 선수

i. MINUS 연산자 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
MINUS
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE POSITION = 'MF';
```

ii. 논리연산자 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
AND POSITION<>'MF'
ORDER BY 1,2,3,4,5;
```

iii. NOT EXISTS/ NOT IN 서브쿼리 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER X
WHERE TEAM_ID = 'K02'
AND NOT EXISTS (SELECT 1 FROM PLAYER Y
WHERE Y.PLAYER_ID = X.PLAYER_ID
AND POSITION = 'MF')
ORDER BY 1,2,3,4,5;
```

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
AND PLAYER_ID NOT IN (SELECT PLAYER_ID
FROM PLAYER WHERE POSITION = 'MF')
ORDER BY 1,2,3,4,5;
```

[예제] 삼성블루윙즈팀이면서 포지션이 GK인 선수

i. INTERSECT 연산자 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
INTERSECT
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE POSITION = 'GK'
ORDER BY 1,2,3,4,5;
```

ii. AND 연산자 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE TEAM_ID = 'K02'
AND POSITION = 'GK'
ORDER BY 1,2,3,4,5;
```

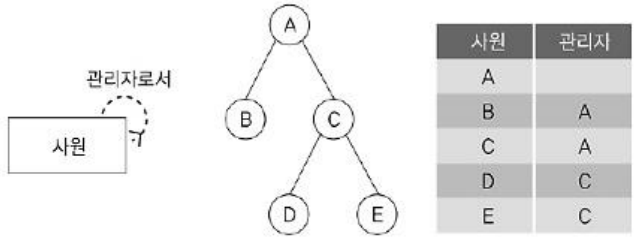
iii. EXISTS/ IN 서브쿼리 이용

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER X
WHERE X.TEAM_ID = 'K02'
AND EXISTS (SELECT 1 FROM PLAYER Y
WHERE Y.PLAYER_ID = X.PLAYER_ID
AND Y.POSITION = 'GK')
ORDER BY 1,2,3,4,5;
```

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER X
WHERE TEAM_ID = 'K02'
AND PLAYER_ID IN (SELECT PLAYER_ID FROM PLAYER
WHERE POSITION = 'GK')
ORDER BY 1,2,3,4,5;
```

3) 계층형 질의와 셀프조인

1. 계층형 질의 (Hierarchical Query)



사원	관리자
A	
B	A
C	A
D	C
E	C

(1) 순환관계 데이터 모델 (2) 계층형 구조 (3) 샘플 데이터

[그림 Ⅱ-2-6] 계층형 데이터

- 계층형 질의 정의:
  동일 테이블에 계층적으로 상위와 하위 데이터가 포함된 데이터
- 형태

```
SELECT…
FROM 테이블
WHERE condition AND condition…
START WITH condition
CONNECT BY [NOCYCLE] condition AND condition…
[ORDER SIBLINGS BY column, column, …]
```

- ✓ **SELECT** + 가상칼럼*
- ✓ **START WITH** + 루트데이터 위치
- ✓ **CONNECT BY + PRIOR** 부모=자식|자식=부모
- ✓ **NOCYCLE**: 데이터를 전개하면서 이미 나타났던 동일한 데이터가 전개 중 다시 나타나면 사이클이 형성되는데 이를 방지하는 옵션
- ✓ **ORDER SIBLING BY**: 형제 노드 사이 정렬

- 계층형 질의에서 사용되는 가상 칼럼

가상 컬럼	설명
LEVEL	루트 데이터이면 1, 그 이후 리프(Leaf) 데이터까지 1씩 증가함
CONNECT_BY_ISLEAF	리프 데이터이면 1, 그렇지 않으면 0.
CONNECT_BY_ISCYCLE	해당 데이터가 조상*으로서 존재하면 1, 그렇지 않으면 0. (*조상: 자신으로부터 루트까지의 경로에 존재하는 데이터) NOCYCLE 옵션을 사용했을 때 사용가능

- 계층형 질의에서 사용되는 함수

함수	설명
SYS_CONNECT_BY_PATH (칼럼, 칼럼분리자)	루트 데이터부터 현재 전개할 데이터까지의 경로를 표시
CONNECT_BY_ROOT(칼럼)	현재 전개할 데이터의 루트 데이터

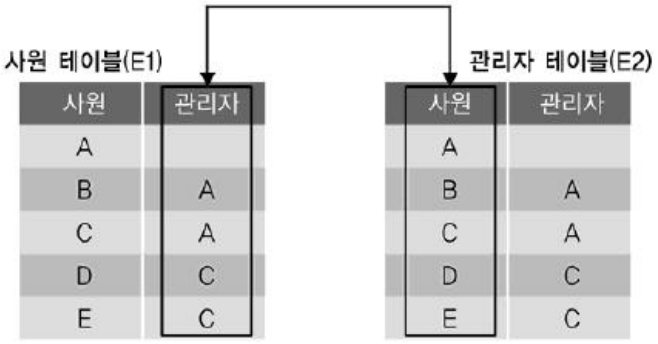
[예제] 관리자→사원 순으로 계층형 쿼리 생성

```
SELECT LEVEL, EMPNO 사원, MGR 관리자,
CONNECT_BY_ISLEAF ISLEAF
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR;
```

[예제] 관리자→사원 순, 경로와 루트데이터 포함해 계층형 쿼리 생성

```
SELECT CONNECT_BY_ROOT(EMPNO) 루트사원,
SYS_CONNECT_BY_PATH(EMPNO,'/') 경로, EMPNO 사원, MGR 관리자
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO= MGR;
```

2. 셀프 조인



- 동일 테이블 사이의 조인으로, 조인 시 식별을 위해 반드시 ALIAS 사용해야 함!
- 형태

SELECT 별칭1.칼럼명, 별칭2.칼럼명
FROM 테이블 별칭1, 테이블 별칭2
WHERE 별칭1.칼럼명2 = 별칭2.칼럼명1;

[예제] EMP에서 자신과 상위, 차상위 관리자를 같은 줄에 표시하기

```
SELECT E1.EMPNO 사원, E1.MGR 상위관리자, E2.EMPNO 차상위관리자
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO
ORDER  BY E2.MGR DESC, E1.MGR, E1.EMPNO;
```

→ 최상위 관리자인 7839 번 사원 정보 누락됨

```
SELECT E1.EMPNO 사원, E1.MGR 상위관리자, E2.MGR 차상위관리자
FROM EMP E1 LEFT OUTER JOIN EMP E2
ON E1.MGR = E2.EMPNO
ORDER BY E2.MGR DESC, E1.MGR, E1.EMPNO;
```

4) 서브쿼리

[그림 Ⅱ-2-12] 메인쿼리와 서브쿼리

- 서브쿼리 vs. 조인 (2)
- 서브쿼리 사용 시 주의사항 (3)
- 서브쿼리의 분류 → 동작방식 (2), 데이터 형태(3)

1. 단일 행 서브쿼리

- 단일행 비교 연산자 (=,<,<=,>,>=,<>)와 함께 사용하고, 서브쿼리의 결과 수가 반드시 1건 이하!

[예제] 정남일 선수가 소속된 팀의 선수들에 대한 정보


```
SELECT TEAM_ID,PLAYER_NAME,POSITION,BACK_NO
FROM PLAYER
WHERE TEAM_ID =
(SELECT TEAM_ID FROM PLAYER WHERE PLAYER_NAME = '정남일')
ORDER BY PLAYER_NAME;
```

[예제] 키가 평균 이하인 선수들의 정보


```
SELECT PLAYER_NAME, POSITION, BACK_NO
FROM PLAYER
WHERE HEIGHT <= (SELECT AVG(HEIGHT) FROM PLAYER)
ORDER BY PLAYER_NAME;
```

→ AVG() 와 같은 그룹함수를 써주기 위해선 서브쿼리 필수!

2. 다중 행 서브쿼리

- 서브쿼리 결과가 2건 이상 반환될 수 있고, 반드시 다중 행 비교 연산자와 사용해야 함

다중 행 연산자	설명
IN(서브쿼리)	서브쿼리의 결과에 존재하는 임의의 값과 동일한 조건 (Multiple OR 조건)
비교연산자 **ALL**(서브쿼리)	서브쿼리의 결과에 존재하는 모든 값을 만족하는 조건
비교연산자 **ANY**(서브쿼리)	서브쿼리의 결과에 존재하는 어느 하나의 값이라도 만족하는 조건
EXISTS(서브쿼리)	서브쿼리의 결과를 만족하는 값의 존재 여부를 확인하는 조건. 조건을 만족하는 건이 여러 건이더라도 1건만 찾으면 더 이상 검색하지 않음.

[예제] '정현수'라는 선수가 소속되어 있는 팀의 정보 (동명이인 있음)

```
SELECT REGION_NAME 연고지명, TEAM_NAME 팀명, E_TEAM_NAME 영문팀명
FROM TEAM
WHERE TEAM_ID IN
(SELECT TEAM_ID FROM PLAYER WHERE PLAYER_NAME = '정현수')
ORDER BY TEAM_NAME;
```

→ IN 연산자로 서브쿼리의 결과가 여러 개인 다중 행 반환

[예제] 부서번호 30번인 사람들이 받는 급여보다 많은 급여를 받는 직원 정보 출력

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ALL (SELECT SAL FROM EMP WHERE DEPTNO = 30);
```

→ ALL 연산자 사용!

[예제]부서번호 30번 사람들이 받는 급여 중에 가장 낮은 사람보다 많은 급여를 직원 정보 출력

→ ANY 연산자 사용

3. 다중 칼럼 서브쿼리

- 서브쿼리의 결과로 여러 개의 칼럼이 반환되어 메인쿼리의 조건과 동시에 비교되는 것

[예제] 소속팀별 키가 가장 작은 사람들의 정보 출력

```
SELECT TEAM_ID, PLAYER_NAME, POSITION, BACK_NO, HEIGHT
FROM PLAYER
WHERE (TEAM_ID, HEIGHT) IN
(SELECT TEAM_ID, MIN(HEIGHT) FROM PLAYER GROUP BY TEAM_ID)
ORDER BY TEAM_ID, PLAYER_NAME;
```

4. 연관 서브쿼리

- 서브쿼리 내에 메인 쿼리 칼럼이 사용된 서브쿼리

〈연관 서브쿼리 프로세스〉

1. 메인 쿼리에서 한 행 값을 가져옴
2. 서브쿼리에서 전달받은 메인 쿼리 값을 이용해 실행
3. 서브쿼리의 실행결과를 메인 쿼리로 넘겨줌
4. 메인 쿼리의 row가 끝날 때까지 1~3 반복

[예제] 소속팀별 평균 키보다 작은 선수들의 정보 출력

```
SELECT TEAM_NAME,PLAYER_NAME,POSITION,BACK_NO,HEIGHT
FROM PLAYER P, TEAM T
WHERE P.TEAM_ID = T.TEAM_ID
AND HEIGHT <= (SELECT AVG(S.HEIGHT) FROM PLAYER S
               WHERE P.TEAM_ID = S.TEAM_ID
               AND S.TEAM_ID IS NOT NULL
               GROUP BY S.TEAM_ID)
ORDER BY PLAYER_NAME;
```

[예제] 소속부서의 평균급여보다 급여를 많이 받는 사원의 모든 정보 출력

```
SELECT * FROM EMP E
WHERE SAL > (SELECT  AVG(SAL)
FROM EMP M WHERE E.DEPTNO = M.DEPTNO
GROUP BY DEPTNO);
```

→ '>'는 단일 행 연산자이지만 메인 쿼리의 하나의 행인 DEPTNO값을 참조하여 서브쿼리에서 SAL의 평균을 내줌 (메인 쿼리의 한 행 별로 한 부서의 평균을 내줘서 단일 행 연산자 가능)

- EXISTS 서브쿼리: 항상 연관 서브쿼리!!!!

[예제] 20120501부터 20120502사이에 경기가 있는 경기장 조회

```
SELECT STADIUM_ID, STADIUM_NAME
FROM STADIUM S
WHERE EXISTS (SELECT 1 FROM SCHEDULE SC
WHERE S.STADIUM_ID = SC.STADIUM_ID
AND SCHE_DATE BETWEEN '20120501' AND '20120502');
```

5. 그밖에 위치에서 사용하는 서브쿼리

- SELECT 절에 서브쿼리 이용: **스칼라 서브쿼리**

1 ROW 1 COLUMN

[예제] 선수정보와 해당 선수가 속한 팀의 평균키 출력

```
SELECT PLAYER_NAME, HEIGHT,
ROUND((SELECT AVG(HEIGHT) FROM PLAYER L
WHERE P.TEAM_ID = L.TEAM_ID),3) 팀별평균키
FROM PLAYER P
ORDER BY 1;
```

- FROM 절에 서브쿼리 이용: **인라인뷰**

FROM절에서 사용되는 서브쿼리로, SQL문이 실행될 때만 임시적으로 생성된 동적인 뷰이기 때문에 DB에 해당 정보가 저장되지 않음

[예제] 포지션이 MF인 선수들의 소속팀명 및 선수정보 출력

i. WHERE절 이용

```
SELECT TEAM_NAME,PLAYER_NAME,BACK_NO
FROM PLAYER P, TEAM T
WHERE P.TEAM_ID = T.TEAM_ID
AND POSITION = 'MF'
ORDER BY 2;
```

ii. 인라인뷰 이용

```
SELECT T.TEAM_NAME,P.PLAYER_NAME,P.BACK_NO
FROM (SELECT TEAM_ID, PLAYER_NAME, BACK_NO FROM PLAYER
WHERE POSITION='MF') P, TEAM T
WHERE P.TEAM_ID =T.TEAM_ID
ORDER BY 2;
```

TOP-N 쿼리

인라인 뷰에서 정렬을 먼저 수행하고 정렬된 결과 중에서 일부 데이터를 추출하는 것

[예제] 키가 가장 큰 5명의 선수정보 출력

옳은 쿼리

```
SELECT PLAYER_NAME,POSITION,BACK_NO,HEIGHT
FROM (SELECT PLAYER_NAME,POSITION,BACK_NO,HEIGHT FROM PLAYER
WHERE HEIGHT IS NOT NULL
ORDER BY HEIGHT DESC)
WHERE ROWNUM<=5;
```

틀린 쿼리

```
SELECT PLAYER_NAME,POSITION,BACK_NO,HEIGHT
FROM PLAYER
WHERE ROWNUM<=5
ORDER BY HEIGHT DESC;
```

- HAVING 절에 서브쿼리 이용: 그룹함수와 함께 사용될 때

그룹핑된 결과에 대해 부가적인 조건을 줄 때
[예시] 평균키가 삼성 블루윙즈팀의 평균키보다 작은 팀의 이름과 해당 팀의 평균키 구하기

```
SELECT T.TEAM_NAME, AVG(P.HEIGHT)
FROM PLAYER P, TEAM T
WHERE P.TEAM_ID = T.TEAM_ID
GROUP BY T.TEAM_NAME
HAVING AVG(P.HEIGHT) < (SELECT AVG(HEIGHT) FROM PLAYER
WHERE TEAM_ID ='K02');
```

- UPDATE문의 SET절에서 이용
[예제] TEAM테이블에 STADIUM_NAME 열을 추가하고, UPDATE문을 이용해 STADIUM_NAME의 값을 STADIUM 테이블의 STADIUM_NAME에서 가져오기

```
ALTER TABLE TEAM
ADD STADIUM_NAME VARCHAR(50);
UPDATE TEAM A
SET A.STADIUM_NAME = (SELECT X.STADIUM_NAME
FROM STADIUM X
WHERE X.STADIUM_ID = A.STADIUM_ID);
```

- INSERT문의 VALUES절에서 이용

```
INSERT INTO PLAYER(PLAYER_ID, PLAYER_NAME, TEAM_ID)
VALUES ((SELECT TO_CHAR(MAX(TO_NUMBER(PLAYER_ID))+1) FROM PLAYER),
'홍길동','K06');
```

6. 뷰 (View)
- 뷰의 장점 (3): 독립성, 편리성, 보안성

5) 그룹함수

1. 데이터 분석 개요
- AGGREGATE FUNCTION
- GROUP FUNCTION: **ROLLUP, CUBE, GROUPING SETS**
- WINDOW FUNCTION

2. ROLLUP 함수

- ROLLUP에 지정된 Grouping Columns의 리스트는 'Subtotal'을 생성하기 위해 사용됨
(그룹핑 컬럼수: N → 부분합: N+1)

Step1. 일반적인 GROUP BY 절
[예제] 부서명과 업무명을 기준으로 사원 수와 급여 합을 집계한 일반적인 GROUP BY 절 수행

```
SELECT DNAME, JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY DNAME,JOB;
```

Step2. GROUP BY + ORDER BY 절

```
SELECT DNAME, JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY DNAME,JOB
ORDER BY DNAME,JOB;
```

Step3. ROLLUP 함수 사용
[결과]

	DNAME	JOB	직원수	급여합
1	SALES	CLERK	1	950
2	SALES	MANAGER	1	2850
3	SALES	SALESMAN	4	5600
4	SALES	(null)	6	9400
5	RESEARCH	CLERK	2	1900
6	RESEARCH	ANALYST	2	6000
7	RESEARCH	MANAGER	1	2975
8	RESEARCH	(null)	5	10875
9	ACCOUNTING	CLERK	1	1300
10	ACCOUNTING	MANAGER	1	2450
11	ACCOUNTING	PRESIDENT	1	5000
12	ACCOUNTING	(null)	3	8750
13	(null)	(null)	14	29025

[쿼리]

```
SELECT DNAME, JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY ROLLUP (DNAME,JOB);
```

→ L1: GROUP BY 수행시 생성되는 표준 집계 (9건)
L2: DNAME별 모든 JOB의 SUBTOTAL(3건)
L3: GRAND TOTAL (마지막 행, 1건)

Step 2-2. ROLLUP + ORDER BY

```
SELECT DNAME, JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY ROLLUP (DNAME,JOB)
ORDER BY DNAME, JOB;
```

Step3. GROUPING 함수 적용: 0과 1로 SUBTOTAL 구분

```
SELECT DNAME,GROUPING(DNAME),
       JOB,GROUPING(JOB),
       COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY ROLLUP (DNAME,JOB)
ORDER BY DNAME, JOB;
```

Step4. GROUPING 함수 + CASE 사용:
- 소계(Subtotal) 필드에 문자열지정
[결과]

	DNAME	JOB	직원수	급여합
1	SALES	CLERK	1	950
2	SALES	MANAGER	1	2850
3	SALES	SALESMAN	4	5600
4	SALES	All Jobs	6	9400
5	RESEARCH	CLERK	2	1900
6	RESEARCH	ANALYST	2	6000
7	RESEARCH	MANAGER	1	2975
8	RESEARCH	All Jobs	5	10875
9	ACCOUNTING	CLERK	1	1300
10	ACCOUNTING	MANAGER	1	2450
11	ACCOUNTING	PRESIDENT	1	5000
12	ACCOUNTING	All Jobs	3	8750
13	All Departments	All Jobs	14	29025

[쿼리]

```
SELECT
CASE GROUPING(DNAME) WHEN 1 THEN 'All Departments' ELSE DNAME END AS DNAME,
CASE GROUPING(JOB) WHEN 1 THEN 'All Jobs' ELSE JOB END AS JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY ROLLUP(DNAME,JOB);
```

```
SELECT
DECODE(GROUPING(DNAME),1,'All Departments',DNAME) DNAME,
DECODE(GROUPING(JOB),1,'All Jobs',JOB) JOB,
COUNT(*) 직원수,SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO=EMP.DEPTNO
GROUP BY ROLLUP(DNAME,JOB);
```

Step4-2. ROLLUP 함수 일부 사용: Grand total 제거

[결과]

	DNAME	JOB	직원수	급여합
1	SALES	CLERK	1	950
2	SALES	MANAGER	1	2850
3	SALES	SALESMAN	4	5600
4	SALES	All Jobs	6	9400
5	RESEARCH	CLERK	2	1900
6	RESEARCH	ANALYST	2	6000
7	RESEARCH	MANAGER	1	2975
8	RESEARCH	All Jobs	5	10875
9	ACCOUNTING	CLERK	1	1300
10	ACCOUNTING	MANAGER	1	2450
11	ACCOUNTING	PRESIDENT	1	5000
12	ACCOUNTING	All Jobs	3	8750

[쿼리]

```
SELECT DNAME,
DECODE(GROUPING(JOB),1,'All Jobs',JOB) JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY DNAME, ROLLUP(JOB);
```

Step4-3. ROLLUP 함수 결합칼럼 사용

[예제] JOB과 MGR은 하나의 집합으로 간주하여 부서별, JOB&MGR에 대한 ROLLUP 결과 출력

[결과]

	DNAME	JOB	MGR	급여합
1	SALES	CLERK	7698	950
2	SALES	MANAGER	7839	2850
3	SALES	SALESMAN	7698	5600
4	SALES	(null)	(null)	9400
5	RESEARCH	CLERK	7788	1100
6	RESEARCH	CLERK	7902	800
7	RESEARCH	ANALYST	7566	6000
8	RESEARCH	MANAGER	7839	2975
9	RESEARCH	(null)	(null)	10875
10	ACCOUNTING	CLERK	7782	1300
11	ACCOUNTING	MANAGER	7839	2450
12	ACCOUNTING	PRESIDENT	(null)	5000
13	ACCOUNTING	(null)	(null)	8750
14	(null)	(null)	(null)	29025

[쿼리]

```
SELECT DNAME, JOB, MGR, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY ROLLUP(DNAME,(JOB,MGR));
```

3. CUBE 함수

- 결합 가능한 모든 값에 대해 다차원 집계 생성

Step5. CUBE 함수 사용

[결과]

	DNAME	JOB	직원수	급여합
1	All Departments	All Jobs	14	29025
2	All Departments	CLERK	4	4150
3	All Departments	ANALYST	2	6000
4	All Departments	MANAGER	3	8275
5	All Departments	SALESMAN	4	5600
6	All Departments	PRESIDENT	1	5000
7	SALES	All Jobs	6	9400
8	SALES	CLERK	1	950
9	SALES	MANAGER	1	2850
10	SALES	SALESMAN	4	5600
11	RESEARCH	All Jobs	5	10875
12	RESEARCH	CLERK	2	1900
13	RESEARCH	ANALYST	2	6000
14	RESEARCH	MANAGER	1	2975
15	ACCOUNTING	All Jobs	3	8750
16	ACCOUNTING	CLERK	1	1300
17	ACCOUNTING	MANAGER	1	2450
18	ACCOUNTING	PRESIDENT	1	5000

[쿼리]

```
SELECT
CASE GROUPING(DNAME) WHEN 1 THEN 'All Departments' ELSE DNAME END DNAME,
CASE GROUPING(JOB) WHEN 1 THEN 'All Jobs' ELSE JOB END JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP,DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO
GROUP BY CUBE(DNAME,JOB);
```

4. GROUPING SETS 함수

- 괄호로 묶은 집합 별로 집계를 구함
- 인수들간에 평등한 관계
  → 인수의 순서 바꿔도 결과 똑같음
- UNION ALL을 사용한 그룹ftn을 사용한 쿼리와 동일

[결과]

	DNAME	JOB	직원수	급여합
1	All Departments	CLERK	4	4150
2	All Departments	SALESMAN	4	5600
3	All Departments	PRESIDENT	1	5000
4	All Departments	MANAGER	3	8275
5	All Departments	ANALYST	2	6000
6	ACCOUNTING	All Jobs	3	8750
7	RESEARCH	All Jobs	5	10875
8	SALES	All Jobs	6	9400

[쿼리]

```
SELECT
DECODE(GROUPING(DNAME),1,'All Departments',DNAME) DNAME,
DECODE(GROUPING(JOB),1,'All Jobs',JOB) JOB,
COUNT(*) 직원수, SUM(SAL) 급여합
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
GROUP BY GROUPING SETS(DNAME,JOB);
```

6) 윈도우함수

1. WINDOW FUNCTION 개요

- 행과 행간의 관계를 쉽게 정의하기 위해 만든 함수
- 종류 (5): 순위/집계/행 순서/비율/통계
- WINDOW FUNCTION 구조

```
SELECT 윈도우함수 (인자)
OVER([PARTITION BY] [ORDER BY] [WINDOWING])
FROM 테이블 명;
```

- ✓ **PARTITION BY**: 전체 집합을 기준에 의해 소그룹화
- ✓ **ORDER BY:** 어떤 항목에 대해 순위를 지정할 지
- ✓ **WINDOWING:** 함수의 대상이 되는 행 기준의 범위 지정 (**ROWS** or **RANGE**)
  →**ROWS**는 물리적인 결과의 행의 수,
  RANGE는 논리적인 값에 의한 범위

2. 그룹 내 순위 함수

- **RANK**: OREDER BY를 포함한 쿼리에서 특정 항목(PARTITION BY)에 대한 순위를 구하는 함수

[예제] 사원 데이터에서 급여가 높은 순서와 JOB별로 급여가 높은 순서 출력

```
SELECT JOB,ENAME,SAL,
RANK() OVER (ORDER BY SAL DESC) ALL_RANK,
RANK() OVER (PARTITION BY JOB ORDER BY SAL DESC) JOB_RANK
FROM EMP;
```

→ JOB별로 정렬되지 않음

- **DENSE_RANK**: 동일한 순위를 하나의 건수로 취급
- **ROW_NUMBER**: 동일한 값이라도 고유한 순위 부여

[예제] RANK, DENSE_RANK, ROW_NUMBER비교

```
SELECT JOB,ENAME,SAL,
RANK() OVER (ORDER BY SAL DESC) RANK,
DENSE_RANK() OVER(ORDER BY SAL DESC) DENSE_RANK,
ROW_NUMBER() OVER(ORDER BY SAL DESC) ROW_NUMBER
FROM EMP;
```

	JOB	ENAME	SAL	RANK	DENSE_RANK	ROW_NUMBER
1	PRESIDENT	KING	5000	1	1	1
2	ANALYST	FORD	3000	2	2	2
3	ANALYST	SCOTT	3000	2	2	3
4	MANAGER	JONES	2975	4	3	4
5	MANAGER	BLAKE	2850	5	4	5
6	MANAGER	CLARK	2450	6	5	6
7	SALESMAN	ALLEN	1600	7	6	7
8	SALESMAN	TURNER	1500	8	7	8
9	CLERK	MILLER	1300	9	8	9
10	SALESMAN	WARD	1250	10	9	10
11	SALESMAN	MARTIN	1250	10	9	11
12	CLERK	ADAMS	1100	12	10	12
13	CLERK	JAMES	950	13	11	13
14	CLERK	SMITH	800	14	12	14

→ **RANK**: 기본값
→**DENSE_RANK**: 동일한 순위를 하나의 건수로
→**ROW_NUMBER**: 동일한 순위 배제, 유니크한 순위

3. 일반 집계 함수

- **SUM**: 파티션별 윈도우의 합

[예제] 사원 이름, 급여 & 같은 매니저를 두고 있는 사원들의 SALARY의 합 출력

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

[결과]매니저별 급여의 합을 MGR_SUM에 저장함

	MGR	ENAME	SAL	MGR_SUM
1	7566	FORD	3000	6000
2	7566	SCOTT	3000	6000
3	7698	JAMES	950	6550
4	7698	ALLEN	1600	6550
5	7698	WARD	1250	6550
6	7698	TURNER	1500	6550
7	7698	MARTIN	1250	6550
8	7782	MILLER	1300	1300
9	7788	ADAMS	1100	1100
10	7839	BLAKE	2850	8275
11	7839	JONES	2975	8275
12	7839	CLARK	2450	8275
13	7902	SMITH	800	800
14	(null)	KING	5000	5000

[예제] 파티션 내 데이터 정렬, 이전 SALARY데이터까지의 누적값

```
SELECT MGR,ENAME,SAL,
SUM(SAL) OVER(PARTITION BY MGR ORDER BY SAL RANGE UNBOUNDED PRECEDING)
AS MGR_SUM
FROM EMP;
```

→ **RANGE UNBOUNDED PRECEDING**:
현재 행을 기준으로 파티션 내의 첫 번 째 행까지의 범위를 지정

[결과]

```
   MGR ENAME      SAL   MGR_SUM
------ ---------- ----- ----------
  7566 SCOTT      3000   6000
  7566 FORD       3000   6000
------------------------------------------------------------
  7698 JAMES       950   950
  7698 WARD       1250   3450 << 950 + 1250 + 1250 = 3450
  7698 MARTIN     1250   3450 << 950 + 1250 + 1250 = 3450
  7698 TURNER     1500   4950 << 950 + 1250 + 1250 + 1500 = 4950
  7698 ALLEN      1600   6550
------------------------------------------------------------
  7782 MILLER     1300   1300
------------------------------------------------------------
  7788 ADAMS      1100   1100
------------------------------------------------------------
  7839 CLARK      2450   2450
  7839 BLAKE      2850   5300
  7839 JONES      2975   8275
------------------------------------------------------------
  7902 SMITH       800   800
------------------------------------------------------------
       KING       5000   5000
```

- **MAX**: 파티션 별 윈도우의 최댓값

[예제] 사원들의 급여&
같은 매니저를 두고 있는 사원들의 SALARY 중 최댓값

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

[결과]

	MGR	ENAME	SAL	MAX_SAL
1	7566	FORD	3000	3000
2	7566	SCOTT	3000	3000
3	7698	JAMES	950	1600
4	7698	ALLEN	1600	1600
5	7698	WARD	1250	1600
6	7698	TURNER	1500	1600
7	7698	MARTIN	1250	1600
8	7782	MILLER	1300	1300
9	7788	ADAMS	1100	1100
10	7839	BLAKE	2850	2975
11	7839	JONES	2975	2975
12	7839	CLARK	2450	2975
13	7902	SMITH	800	800
14	(null)	KING	5000	5000

[예제] 파티션별 최대값을 가진 행만 추출 (INLINE뷰)

```
SELECT MGR,ENAME,SAL
FROM (SELECT MGR,ENAME,SAL, MAX(SAL)
      OVER(PARTITION BY MGR) AS IV_MAX_SAL
      FROM EMP)
WHERE SAL = IV_MAX_SAL;
```

→ i. 인라인 뷰를 통해 테이블의 칼럼들을 제한
(MGR, ENAME, SAL, MAX…)
ii. WHERE절에서 SAL을 급여의 MAX값과 같도록 설정

[결과]

	MGR	ENAME	SAL
1	7566	FORD	3000
2	7566	SCOTT	3000
3	7698	ALLEN	1600
4	7782	MILLER	1300
5	7788	ADAMS	1100
6	7839	JONES	2975
7	7902	SMITH	800
8	(null)	KING	5000

- **MIN**: 파티션 윈도우의 최소값

[예제] 사원들의 급여 & 같은 매니저를 두고 있는 사원들을 입사 일자를 기준으로 정렬 & SAL의 최소값

```
SELECT MGR,ENAME,HIREDATE,SAL,
MIN(SAL) OVER(PARTITION BY MGR ORDER BY HIREDATE) MGR_MIN
FROM EMP;
```

- **AVG**: ROWS윈도우를 이용해 AVG구하기

[예제] 같은 매니저를 두고 있는 사원들의 평균 SAL,
*평균을 구할 때, 같은 매니저 내에서
(자기 바로 앞의 사번 + 자신 + 바로 뒤의 사번의 직원)/3

```
SELECT MGR,ENAME,SAL,HIREDATE,
       ROUND(AVG(SAL)OVER(PARTITION BY MGR ORDER BY HIREDATE
       ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)) MGR_AVG
FROM EMP;
```

→ **ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING**:
현재 행을 기준으로 파티션 내에서 앞의 한 건, 현재 행, 뒤의 한 건을 범위로 지정

- **COUNT**: ROWS 윈도우를 이용해 조건에 맞는 통계값

[예제] 사원들을 급여 기준으로 정렬하고, 본인의 급여보다 50이하가 적거나 150이하로 많은 급여를 받는 인원수 출력

```
SELECT ENAME,SAL,
       COUNT(*) OVER (ORDER BY SAL
       RANGE BETWEEN 50 PRECEDING AND 150 FOLLOWING) SIM_COUNT
FROM EMP;
```

4. 그룹 내 행 순서 함수

- **FIRST_VALUE**: 파티션에서 가장 먼저 나온 값 출력

[예제] 부서별 직원들을 연봉이 높은 순서부터 정렬, 파티션 내에서 가장 먼저 나온 값 출력

```
SELECT DEPTNO,ENAME,SAL,
       FIRST_VALUE(ENAME) OVER(PARTITION BY DEPTNO ORDER BY SAL DESC
       ROWS UNBOUNDED PRECEDING) DEPT_RICH
FROM EMP;
```

→ 공동등수 인정하지 않고 처음 나온 행만 처리

[예제] ORDER BY 정렬 조건 추가하여 공동등수 인정하기

```
SELECT DEPTNO,ENAME,SAL,
       FIRST_VALUE(ENAME)OVER
       (PARTITION BY DEPTNO ORDER BY SAL DESC, ENAME
       ROWS UNBOUNDED PRECEDING) RICH_EMP
FROM EMP;
```

→ ENAME칼럼을 ORDER BY에 추가하여 SCOTT->FORD로 변경됨

- **LAST_VALUE**: 파티션에서 가장 나중값 출력

[예제] 부서별 직원들을 연봉이 높은 순서부터 정렬, 파티션 내 가장 마지막에 나온 값 출력

```
SELECT DEPTNO,ENAME,SAL,
       LAST_VALUE(ENAME)OVER
       (PARTITION BY DEPTNO ORDER BY SAL DESC
       ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) DEPT_POOR
FROM EMP;
```

→ **ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING:**
현재 행을 포함하여 파티션 내의 마지막 행까지의 범위를 지정

- **LAG**: 파티션별 윈도우에서 이전 몇 번째 행의 값

[예제] 직원들을 입사일자가 빠른 기준으로 정렬 & 본인보다 입사일자가 두 명 앞선 사원의 급여를 본인의 급여와 함께 출력

```
SELECT ENAME,HIREDATE,SAL,
       LAG(SAL,2,0)OVER(ORDER BY HIREDATE) AS PREV_SAL
FROM EMP;
```

→ **LAG(SAL, 2, 0)**:
두 행 앞의 SALARY를 가져오고, 가져올 값이 없을 경우 0

- **LEAD**: 파티션별 윈도우에서 이후 몇 번째 행의 값

[예제] 직원들을 입사일자가 빠른 기준으로 정렬 & 다음에 입사한 인력의 입사일자를 함께 출력

```
SELECT ENAME,HIREDATE,SAL,
       LEAD(HIREDATE)OVER(ORDER BY HIREDATE) AS NEXTHIRED
FROM EMP;
```

5. 그룹 내 비율 함수

- **RATIO_TO_REPORT**: $\frac{\text{행별 칼럼값}}{\text{파티션 내 전체 칼럼값}}$ 표현

[예제] JOB이 SALESMAN인 사원의 전체 급여에서 본인의 급여가 차지하는 비율을 출력

```
SELECT JOB,ENAME,SAL,
       ROUND(RATIO_TO_REPORT(SAL) OVER(),2) RATIO
FROM EMP
WHERE JOB = 'SALESMAN';
```

- **PERCENT_RANK:** 파티션 내 순서별 백분율 출력

[예제] 같은 부서 소속 사원들의 집합에서 본인의 급여가 순서상 몇 번째 위치쯤에 있는지 0과 1사이의 값으로 출력

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

- **CUME_DIST:** 파티션별 윈도우의 전체건수에서 현재 행보다 작거나 같은 건수에 대한 누적 백분율을 구함

[예제] 같은 부서 소속사원들의 집합에서 본인의 급여가 "누적 순서상" 몇 번째 위치쯤에 있는지 출력

```
SELECT DEPTNO,ENAME,SAL,
       CUME_DIST() OVER(PARTITION BY DEPTNO ORDER BY SAL DESC) C_D
FROM EMP;
```

- **NTILE:** 파티션별 전체건수를 ARGUMENT값으로 N등분

[예제] 전체 사원을 급여가 높은 순서로 정렬, 급여를 기준으로 4개의 그룹으로 분류

```
SELECT ENAME,SAL,
       NTILE(4) OVER(ORDER BY SAL DESC) AS QUAR_TILE
FROM EMP;
```

7) DCL (Data Control Language)

1. DCL 개요

- DCL의 정의:
 유저를 생성하고 권한을 제어하는 명령어

2. 유저와 권한

Oracle 기본적으로 제공하는 유저: **SYS, SYSTEM, SCOTT**

유저	역할
SCOTT	Oracle 테스트용 샘플 유저 Default 패스워드 : TIGER
SYS	DBA ROLE을 부여받은 유저 (DataBase의 시스템 권한 + 오브젝트 권한)
SYSTEM	데이터베이스의 모든 시스템 권한을 부여받은 DBA 유저 Oracle 설치 완료 시에 패스워드 설정

※ Oracle 권한 종류

* 시스템 권한: 데이터베이스 자체에 대한 Control하는 권한
* 오브젝트 권한: 데이터베이스에 만들어진 특정 오브젝트 (테이블, 뷰 등)을 Control 하는 권한

* Syntax
- 권한 부여: **GRANT** 권한 **TO** 유저명;
- User 생성: **CREATE USER** 유저명 **IDENTIFIED BY** PW;
- 권한 회수: **REVOKE** 권한 **FROM** 유저명;
- 로그인 생성: **CREATE LOGIN** login명 **WITH PASSWORD** = 'PW ',**DEFAULT_DATABASE**= DB명;
- 로그인을 이용한 유저 생성: **CREATE USER** 유저명 **FOR LOGIN** login명 **WITH DEFAULT_SCHEMA**= '스키마명';

- 유저 생성과 시스템 권한 부여

GRANT CREATE USER TO 유저; 유저 생성 권한
CREATE SESSION TO 유저; 로그인후 접속 권한
CREATE TABLE TO 유저; 테이블 생성 권한

- OBJECT에 대한 권한 부여

3. Role 을 이용한 권한 부여

8) 절차형 SQL (생략)

1. 절차형 SQL 개요

2. PL/ SQL 개요
- PL/SQL 특징
- PL/SQL 구조
- PL/SQL 기본 문법

3. T-SQL 개요
- T-SQL 특징
- T-SQL 구조
- T-SQL 기본 문법

4. Procedure의 생성과 활용

5. User Defined Function의 생성과 활용

6. Trigger의 생성과 활용

7. 프로시저와 트리거의 차이점

Chap3. SQL 최적화 기본 원리

1) 옵티마이저와 실행 계획

1. 옵티마이저

- 옵티마이저: 사용자가 질의한 SQL문에 대해 최적의 실행 방법(실행 계획)을 결정하는 역할
- 규칙 기반 옵티마이저:

i. 규칙(우선순위)을 가지고 실행계획 생성

→ 우선순위가 높은 규칙이 적은 일량으로 작업 수행

〈우선 순위〉

1st. Single Row by Rowid

하나의 Row id로 한 행 액세스

다른 데이터를 참조할 필요가 없어 가장 빠름

4th. Single Row by PK or Unique Key

하나의 인덱스로 한 행 액세스

인덱스에 액세스→인덱스의 Rowid 추출→테이블의 한 행 액세스

8th. Composite Index

복합 인덱스에 동등 조건**('=')** 으로 검색

i. 모든 구성칼럼에 동등조건 →ii. 인덱스 구성 칼럼 수 많음

9th. Single Column Index

단일 칼럼 인덱스에 동등 조건 **('=')** 으로 검색

10th. Bounded Range Search on Indexed Columns

인덱스 구성 칼럼에 양쪽 범위 한정된 형태로 검색

BETWEEN, LIKE

11th. Unbounded Range Search on Indexed Columns

인덱스 구성 칼럼에 한쪽 범위만 한정된 형태로 검색

〉, 〉=, 〈, 〈=

15th. Full Table Scan

ii. 규칙기반 옵티마이저가 조인순서를 결정할 때

조인 칼럼에 대한 인덱스의 존재 유무 중요!

1st. 인덱스가 양쪽 테이블에 존재하는 경우

→ 우선순위가 **높은** 테이블을 **선행테이블**로

→ 우선순위가 동일하다면 FROM절에 나열된 테이블의 역순으로

2nd. 한쪽 조인 칼럼에만 인덱스가 존재하는 경우

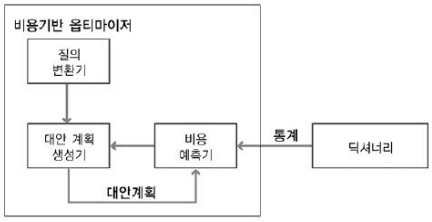
→ 인덱스가 **없는** 테이블을 **선행테이블**로

3rd. 인덱스가 양쪽 테이블에 존재하지 않는 경우

→ from절에 **뒤**에 나열된 테이블을 **선행테이블**로

- 비용 기반 옵티마이저: SQL문을 처리하는데 필요한 비용(소요시간, 자원 사용량)이 가장 적은 실행계획 선택

→ 비용 예측 위해 객체 통계정보, 시스템 통계정보 이용



〈비용 기반 옵티마이저의 모듈 구성〉

질의 변환기: SQL문을 처리하기에 보다 용이한 형태로 변환

대안 계획 생성기: 동일한 결과를 생성하는 다양한 대안 계획을 생성 (eg. 연산적용 순서 변경, 방법 변경, 조인순서 변경)

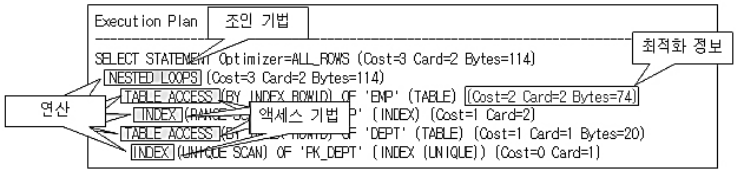
→ 현실적인 제약으로 대안 계획의 수 제약 불가피함

비용 예측기: 대안 계획 생성기에 의해 생성된 대안 계획의 비용을 예측 →통계정보를 토대로 정확한 예측 필수

2. 실행계획 (Execution Plan)

- 실행계획의 정의:

SQL에서 요구한 사항을 처리하기 위한 절차와 방법



〈실행 계획의 구성요소〉

조인 순서: 조인 작업을 수행할 때 참조하는 테이블의 순서

조인 기법: 테이블을 조인할 때 사용할 수 있는 기법

(NS JOIN, HASH JOIN, SORT MERGE JOIN)

액세스 기법: 테이블을 액세스할 때 사용하는 기법

(인덱스 스캔, 전체 테이블 스캔)

최적화 정보: 옵티마이저가 각 단계마다 예상되는 비용 사항 표시

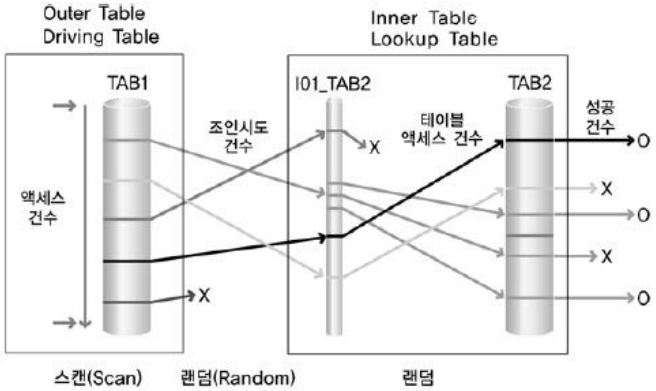
→ 비용사항: Cost/Card/Bytes (통계정보를 바탕으로 한 예상치)

연산: 여러 가지 조작을 통해 원하는 결과를 얻어내는 작업

→ 조인기법, 액세스기법, 필터, 정렬, 집계, 뷰 등 여러 종류

3. SQL 처리 흐름도

- SQL 처리 흐름도: 실행 계획을 시각화한 것



[그림 Ⅱ-3-5] SQL 처리 흐름도

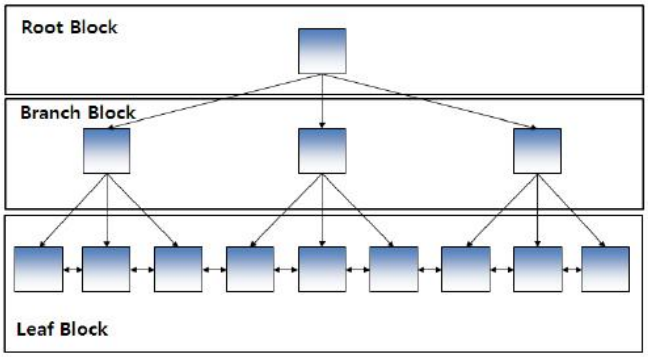
- ✓ TAB1: Outer table(Driving table)
- ✓ TAB2: Inner Table (Lookup table)
- ✓ 조인순서: TAB1-〉TAB2
- ✓ TAB1: Full Table Scan /TAB2: Index Scan
- ✓ 조인방법: NL Join
- ✓ TAB1 액세스: SCAN/ TAB2: 랜덤 액세스

(But, 대량의 데이터를 랜덤 액세스하면 많은 I/O 발생으로 성능에 좋지 않음)

2) 인덱스 기본

1. 인덱스 특징과 종류

- 트리 기반 인덱스



- ✓ DBMS에서 가장 일반적인 인덱스
- ✓ **=,BETWEEN, >** 등과 같은 연산자 검색에 모두 적합
- ✓ **루트 블록**: 브랜치 블록 중 가장 상위에 있는 블록
- ✓ **리프 블록**: 인덱스 구성 칼럼 데이터 +Row id로 구성
  트리의 가장 마지막 단계에 위치
  양방향 링크로 연결 → 오름차순, 내림차순
- ✓ **인덱스 데이터**는 인덱스를 구성하는 칼럼의 값으로 정렬
  →인덱스 데이터의 값이 동일하면 Row id순서로 저장
- ✓ 인덱스를 생성할 때 동일 칼럼으로 구성된 인덱스를 생성할 수 없음
- ✓ 인덱스 구성 칼럼은 동일하지만 칼럼의 순서가 다르면 서로 다른 인덱스로 생성할 수 있음
  (ex. JOB+SAL , SAL+JOB)
- ✓ 인덱스 칼럼의 순서는 성능에 중요한 역할

*Row id = Record Identifier, 행의 위치를 알려주는 id

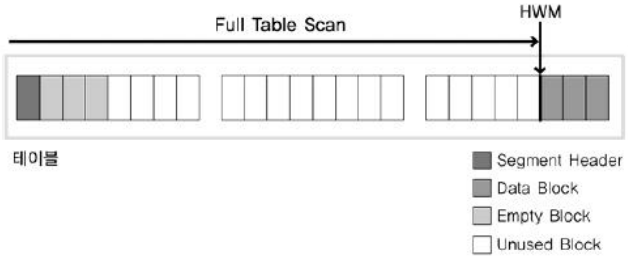
- SQL server의 클러스터형 인덱스

EmployeeID	LastName	FirstName	HireDate
1	Davolio	Nancy	1992-05-01 00:00:00.000
2	Fuller	Andrew	1992-08-14 00:00:00.000
3	Leverling	Janet	1992-04-01 00:00:00.000
4	peacock	Margaret	1993-05-03 00:00:00.000
5	Buchanan	Steven	1993-10-17 00:00:00.000
6	Suyama	Michael	1993-10-17 00:00:00.000
7	King	Robart	1994-01-02 00:00:00.000
8	Callahan	Laura	1994-03-05 00:00:00.000
9	Dodsworth	Anne	1994-11-15 00:00:00.000

- ✓ 인덱스 **리프 페이지** = 데이터 페이지
  → 테이블 탐색에 필요한 레코드식별자가 리프 페이지에 없음
  →리프 페이지를 탐색하면 테이블의 모든 칼럼 값을 곧바로 얻을 수 있음
- ✓ 리프 페이지의 모든 로우(데이터)는 인덱스 키 칼럼 순으로 물리적으로 정렬되어 저장
  → 테이블 로우는 물리적으로 한 가지 순서로만 정렬
  → 클러스터형 인덱스는 테이블 당 한 개만 생성

2. 전체테이블 스캔과 인덱스 스캔

- 전체 테이블 스캔: 검색 조건에 맞는 데이터를 찾기 위해 테이블의 고수위 마크(HWM) 아래의 모든 블록을 읽음

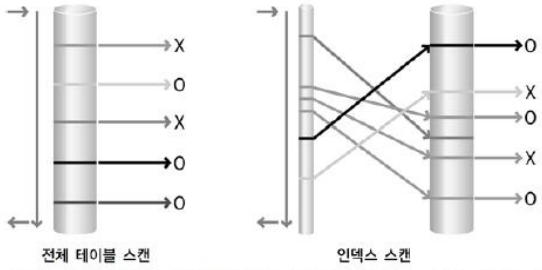


*HWM: 테이블에 데이터가 쓰여졌던 블록 상의 최상위 위치

- ✓ 이렇게 읽은 블록은 재사용성 떨어짐
  → 메모리에서 곧 제거될 수 있도록 관리됨

- ✓ 옵티마이저가 전체 테이블스캔 방식을 선택하는 이유(4)

- 인덱스 스캔 (3)
- 전체 테이블 스캔과 인덱스 스캔 방식 비교

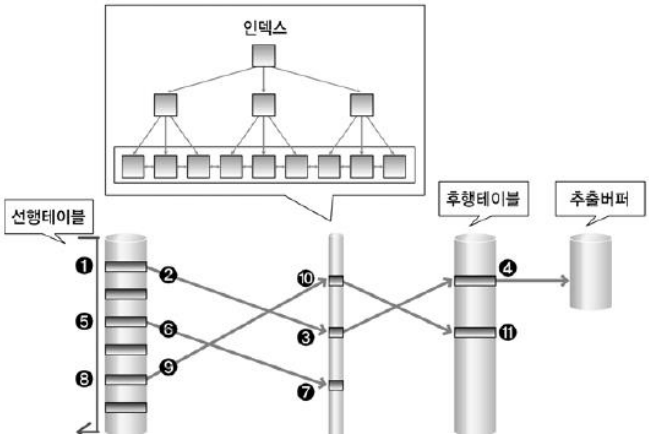


[그림 Ⅱ-3-11] 전체 테이블 스캔과 인덱스 스캔에 대한 SQL 처리 흐름도 표현 예시

→ 대용량 데이터 중 극히 일부의 데이터를 찾을 때 &
인덱스를 구성할 칼럼이 있을 때: 인덱스 스캔방식
→ 대용량 데이터 중 대부분의 데이터를 찾을 때&
한번에 여러 블록씩 읽을 때: 전체 테이블 스캔 방식

3) 조인 수행 원리

1. NL Join: 랜덤액세스! 조인결과 빨리 보여줘!



〈NL 조인의 수행방식〉

step1. 선행 테이블에서 조건을 만족하는 첫 번째 행을 찾음
→ 이때 선행 테이블에 주어진 조건을 만족하지 않는 경우 해당 데이터는 필터링됨

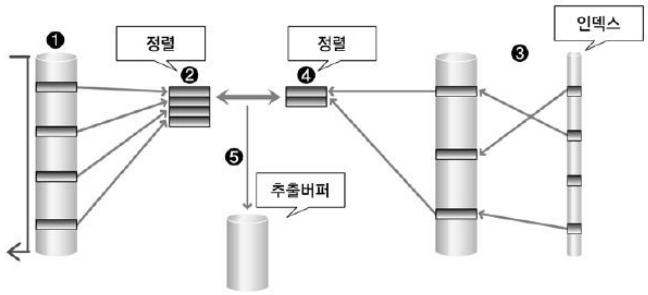
step2. 선행 테이블의 조인 키를 가지고 후행 테이블에 조인 키가 존재하는지 찾으러 감
→ 조인시도

step3. 후행 테이블의 인덱스에 선행 테이블의 조인 키가 존재하는지 확인
→ 선행 테이블의 조인 값이 후행 테이블에 존재하지 않으면 선행 테이블 데이터는 필터링 됨(조인작업 x)

step4. 인덱스에서 추출한 레코드 식별자를 이용하여 테이블을 액세스 & 해당 행을 추출 버퍼에 넣음
→ 인덱스 스캔을 통한 테이블 액세스

step5~11.반복수행

2. Sort Merge Join: 데이터 정렬! 넓은 범위의 데이터!



[그림 Ⅱ-3-13] Sort Merge Join

〈Sort Merge 조인의 수행방식〉

step1. 선행테이블에서 주어진 조건을 만족하는 행 찾음

step2. 선행테이블의 조인키를 기준으로 정렬
→ 조건을 만족하는 모든 행에 대해 step1~2 반복 수행

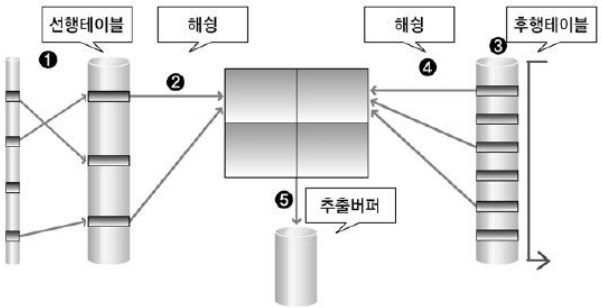
step3. 후행 테이블에서 주어진 조건을 만족하는 행 찾음

step4. 후행테이블의 조인키를 기준으로 정렬
→ 조건을 만족하는 모든 행에 대해 step3~4 반복 수행

step5. 정렬된 결과를 이용하여 조인 수행 & 조인에 성공하면 추출 버퍼에 넣음

- 조인 칼럼의 인덱스가 존재하지 않을 경우 사용

3. Hash Join: 동등조인만! 랜덤액세스&정렬 문제 보완!



〈Hash 조인의 수행방식〉

step1. 선행 테이블에서 주어진 조건을 만족하는 행을 찾음

step2. 선행 테이블의 조인 키를 기준으로 해쉬 함수를 적용하여 해쉬 테이블을 생성
→조인 칼럼과 SELECT 절에서 필요로 하는 칼럼도 함께 저장됨
→ 조건을 만족하는 모든 행에 대해 step1~2 반복 수행

step3. 후행 테이블에서 주어진 조건을 만족하는 행을 찾음

step4. 후행 테이블의 조인 키를 기준으로 해쉬 함수를 적용하여 해당 버킷을 찾음
→ 조인 키를 이용해서 실제 조인될 데이터를 찾음

step5. 조인에 성공하면 추출버퍼에 넣음
→ 조건을 만족하는 모든 행에 대해 step3~5 반복 수행