



SQL

withsky

강의 내용

curriculum

주제	내용
데이터 베이스	<ul style="list-style-type: none"> • 데이터와 데이터베이스 개념 • 데이터베이스 활용 분야 • 데이터베이스 구조 • 실습환경 구축
SQL	<ul style="list-style-type: none"> • SQL 소개 • DDL과 DML 개념
Oracle DB 설치	
SQL 실습	<ul style="list-style-type: none"> • 등록, 수정, 삭제
	<ul style="list-style-type: none"> • 조회 기초 <ul style="list-style-type: none"> - 기본문법 - 정렬 - 조건 검색
	<ul style="list-style-type: none"> • 함수 <ul style="list-style-type: none"> - 문자관련 - 숫자관련 - 날짜관련
	<ul style="list-style-type: none"> • 조회 심화 <ul style="list-style-type: none"> - 그룹핑 - 조인 - 서브쿼리

강의 내용

curriculum

내용
데이터 모델링 데이터 정의어
객체 (테이블, 뷰, 시퀀스, 트랜잭션) 사용자 권한 관리
PL/SQL 기초 문법
프로시저, 함수, 트리거
오라클 아키텍처
인덱스 개념 및 활용

BIG DATA

지금은 데이터가 중심이 되는 빅데이터 시대,
이제는 여러분이 직접 필요한 데이터를 추출하고 가공하는 기술이 필수입니다.



Data

라틴어 dare(주다, give)에서 파생된 명사 **datum**의 복수형(주어진 것)
우리말로 '자료'라는 의미로 사용.

Data(자료)와 Information(정보)의 차이

자료 (Data)	수, 문자(단어) 등의 형태로 이루어진 단위, 추론과 추정의 근거를 이루는 객관적 사실
정보 (Information)	자료를 가공해서 의미를 얻은 것

데이터베이스

Database : 데이터를 구조적으로 모아 저장한 것



DB(DataBase) ! = DBMS(DataBase Management System)



DBMS : 데이터베이스를 관리하는 시스템

DBMS 종류

ORACLE®
DATABASE


MariaDB


Microsoft®
SQL Server®


MySQL®


PostgreSQL

정형
데이터

- **관계형 데이터베이스** (RDB, Relational DB)

- 데이터 구조와 규칙이 정해져 있으며, 체계화 되어 있음
- 대부분의 기업에서 업무용으로 사용

비정형
데이터

- **NoSQL** (Not Only SQL)

- 데이터 구조와 규칙이 정해져 있지 않음
- 빅데이터 처리
- 비정형데이터 : 텍스트, 이미지, 동영상 등

데이터베이스 활용 분야



마케팅 분야

- 방대한 고객 데이터 활용하여 시장변화에 대응
- 다양한 환경에서의 피드백을 활용하여 데이터 추출 분석



커머스 분야

- 사용자 반응 및 사용자 경험 분석을 통한 서비스 개선

전 분야에서 다양하게 활용



프로그래밍 분야

- 어플리케이션 개발 필수 역량



금융 분야

- 고객분석 및 트렌드 예측
- 고객 맞춤형 마케팅 및 컨설팅

고민..

엑셀파일에 백만 건이 넘는 데이터가 들어 있는데, 파일이 잘 열리지도 않고, 빨리 엑셀로 분석결과 제출해야 되는데, 너무 느려요...

우리 회사 고객 데이터를 추출해서 마케팅으로 활용하고 싶은데, 어떻게 해야 할지 모르겠어요...

이번 사업에 대한 매출결과를 다양한 형태로 보고 싶어, 개발팀에 데이터 추출을 요청해야 되는데...

사내 시스템에서 제공하는 분석결과 외에 제가 직접 추출하고 분석도 해 보고 싶어요.

데이터의 진정한 의미



주체는 “나”

데이터는 내가 필요할 때,
내가 필요한 만큼
추출하고 분석할 수 있어야
진정한 의미를 가진다.


데이터베이스의 구조

데이터베이스

아이디	이름	연락처	주소
aaa	홍길동	010-111-1111	서울시
bbb	이순신	010-222-2222	경기도
ccc	강감찬	010-333-3333	인천시
ddd	유관순	010-444-4444	제주도

테이블

데이터베이스 구조

Tablespace (DB)	<div><div></div><div>ABC쇼핑몰.xlsx</div></div>																																																
Table	<div><div>회원관리</div><div>상품관리</div><div>주문내역</div><div>+</div></div>																																																
Data	<table><tr><th></th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th><th>G</th></tr><tr><td>1</td><td>아이디</td><td>회원명</td><td>연락처</td><td>이메일</td><td>성별</td><td>주소</td><td>가입일</td></tr><tr><td>2</td><td>hong</td><td>홍길동</td><td>010-111-1111</td><td>hong@gmail.com</td><td>남</td><td>서울시</td><td>2020-01-01</td></tr><tr><td>3</td><td>lee</td><td>이순신</td><td>010-222-2222</td><td>lee@naver.com</td><td>남</td><td>경기도</td><td>2020-01-02</td></tr><tr><td>4</td><td>kim</td><td>김유신</td><td>010-333-3333</td><td>kim@daum.net</td><td>남</td><td>강원도</td><td>2020-01-03</td></tr><tr><td>5</td><td>ryu</td><td>유관순</td><td>010-444-4444</td><td>ryu@gmail.com</td><td>여</td><td>제주도</td><td>2020-01-04</td></tr></table>		A	B	C	D	E	F	G	1	아이디	회원명	연락처	이메일	성별	주소	가입일	2	hong	홍길동	010-111-1111	hong@gmail.com	남	서울시	2020-01-01	3	lee	이순신	010-222-2222	lee@naver.com	남	경기도	2020-01-02	4	kim	김유신	010-333-3333	kim@daum.net	남	강원도	2020-01-03	5	ryu	유관순	010-444-4444	ryu@gmail.com	여	제주도	2020-01-04
	A	B	C	D	E	F	G																																										
1	아이디	회원명	연락처	이메일	성별	주소	가입일																																										
2	hong	홍길동	010-111-1111	hong@gmail.com	남	서울시	2020-01-01																																										
3	lee	이순신	010-222-2222	lee@naver.com	남	경기도	2020-01-02																																										
4	kim	김유신	010-333-3333	kim@daum.net	남	강원도	2020-01-03																																										
5	ryu	유관순	010-444-4444	ryu@gmail.com	여	제주도	2020-01-04																																										

데이터베이스의 구조

Table

특정한 종류의 데이터를 구조적 목록으로 묶은 것
데이터를 종류별로 구분하는 단위

예) 회원테이블, 상품테이블, 주문테이블...

Member(회원) 테이블

열(컬럼, 속성)

행
로우
레코드

ID	NAME	PHONE	EMAIL	GENDER	ADDRESS	REGDATE
1	hong 홍길동	010-1111-1111	hong@gmail.com	남	서울시	20/01/01
2	lee 이순신	010-2222-2222	lee@naver.com	남	경기도	20/01/02
3	kim 김유신	010-3333-3333	kim@daum.net	남	강원도	20/01/03
4	ryu 유관순	010-4444-4444	ryu@gmail.com	여	제주시	20/01/04

PK (Primary Key)

데이터를 구분하기 위한 유일한 키값

데이터베이스의 특징

통합된 데이터

(Integrated Data)

동일한 내용의 데이터가 중복되지 않음
(중복 데이터 - 관리 어려움)

저장된 데이터

(Stored Data)

디스크 등 저장 매체에 저장

변화되는 데이터

(Changeable Data)

데이터의 삽입, 삭제, 수정을 통한 최신
데이터 유지

공용 데이터

(Shared Data)

여러 이용자(어플리케이션)가 동시에 공유 가능

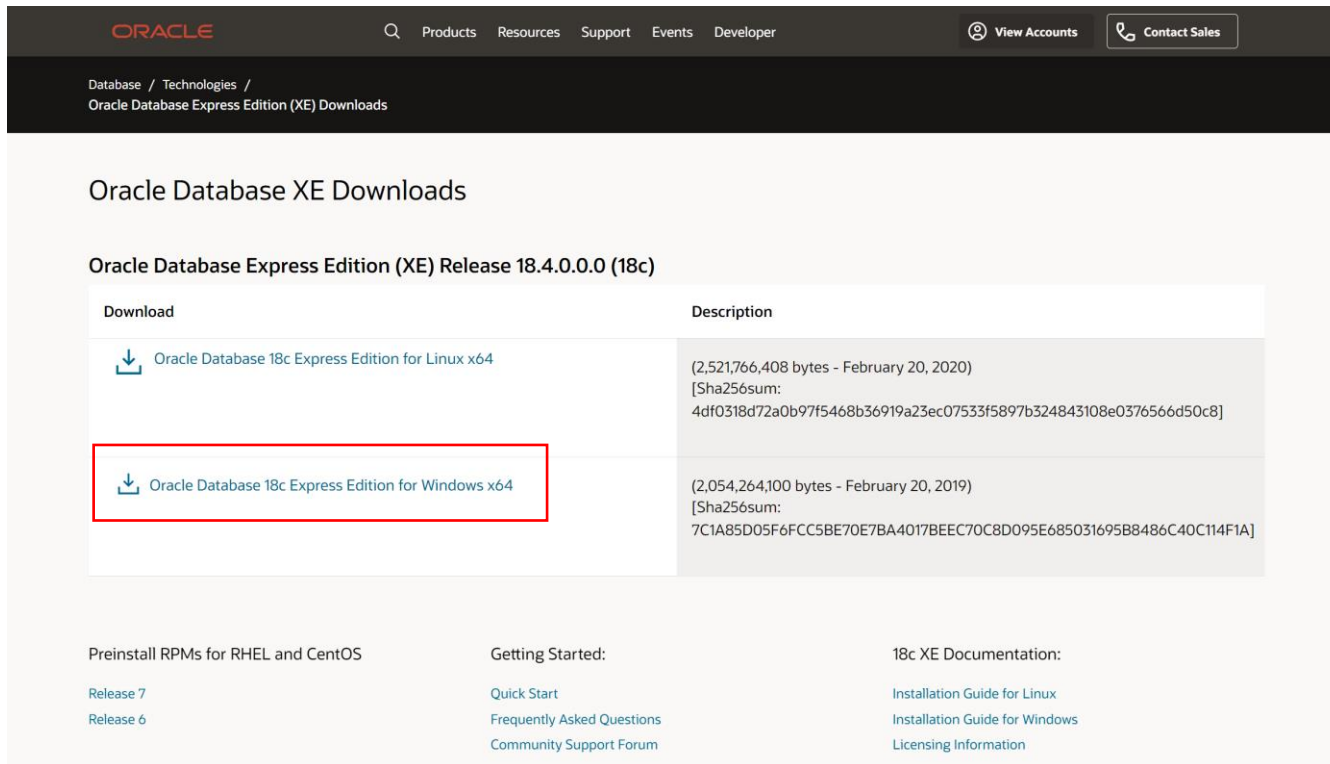
Oracle 설치

Oracle DataBase Express Edition (XE) 다운로드

<https://www.oracle.com/kr/database/technologies/xe-downloads.html>

오라클 계정 생성(회원가입)

윈도우용



ORACLE

Products Resources Support Events Developer

View Accounts Contact Sales

Database / Technologies /
Oracle Database Express Edition (XE) Downloads

Oracle Database XE Downloads

Oracle Database Express Edition (XE) Release 18.4.0.0.0 (18c)

Download	Description
Oracle Database 18c Express Edition for Linux x64	(2,521,766,408 bytes - February 20, 2020) [Sha256sum: 4df0318d72a0b97f5468b36919a23ec07533f5897b324843108e0376566d50c8]
Oracle Database 18c Express Edition for Windows x64	(2,054,264,100 bytes - February 20, 2019) [Sha256sum: 7C1A85D05F6FCC5BE70E7BA4017BEEC70C8D095E685031695B8486C40C114F1A]

Preinstall RPMs for RHEL and CentOS

Release 7
Release 6

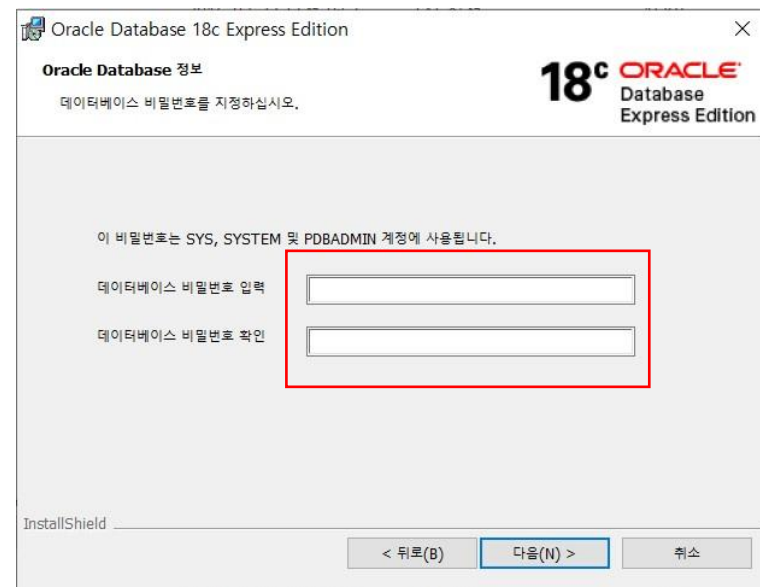
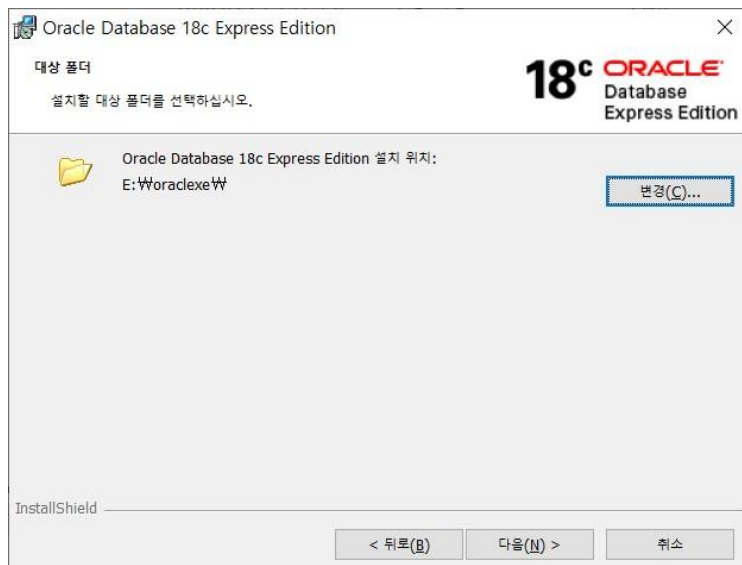
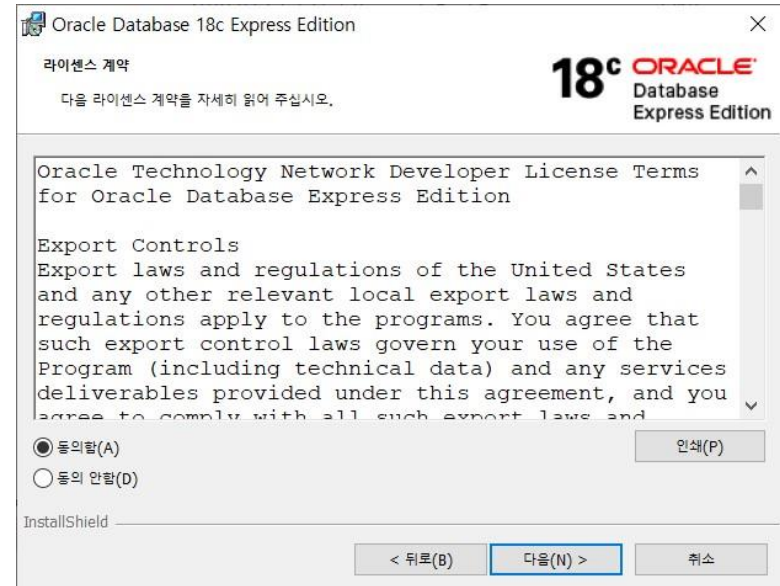
Getting Started:

Quick Start
Frequently Asked Questions
Community Support Forum

18c XE Documentation:

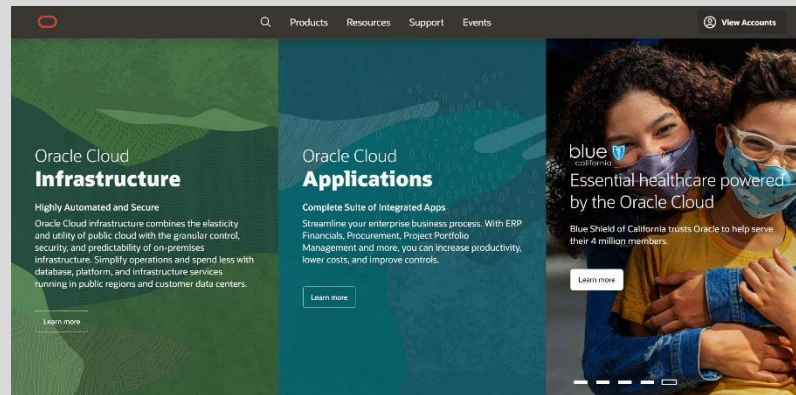
Installation Guide for Linux
Installation Guide for Windows
Licensing Information

Oracle 설치





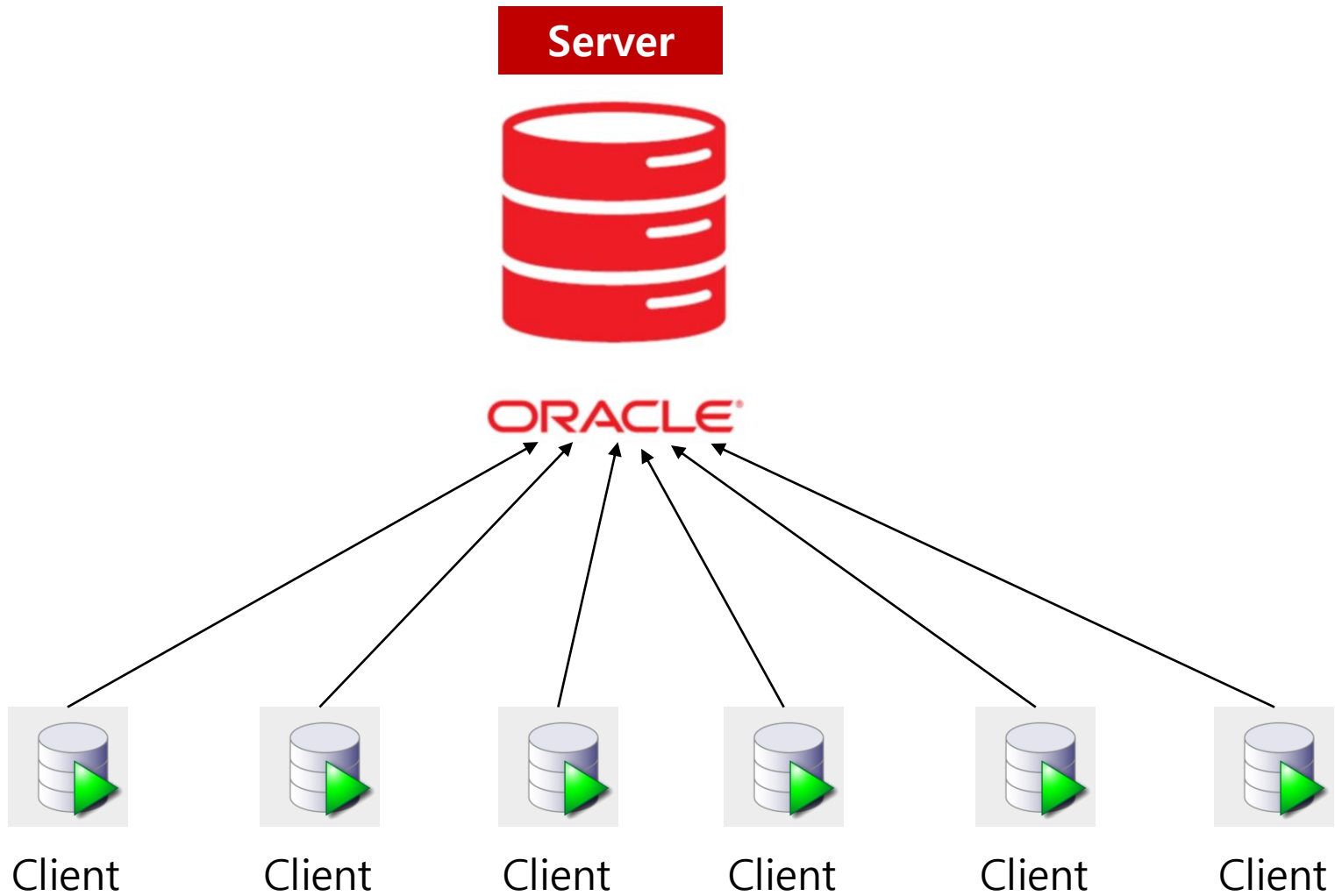
Oracle SQL Developer 설치



<https://www.oracle.com/>

Resources > Software Downloads > Developer Tools > SQL Developer
JDK included

실습환경 구축



실습환경 구축

SQL Developer 접속

새로 만들기/데이터베이스 접속 선택

접속 이름 접속 세부정보

Name local Color

데이터베이스 유형 Oracle

사용자 정보 프록시 사용자

인증 유형 기본값

사용자 이름(U) sys 롤(L) SYSDBA

비밀번호(P) ☐ 비밀번호 저장(V)

접속 유형(Y) 기본

세부정보 고급

호스트 이름(A) localhost

포트(R) 1521

☒ SID(I) xe

☐ 서비스 이름(E)

상태: 성공

도움말(H) 저장(S) 지우기(C) 테스트(T) 취소

Table Space 생성

메뉴 : 보기 > DBA 접속

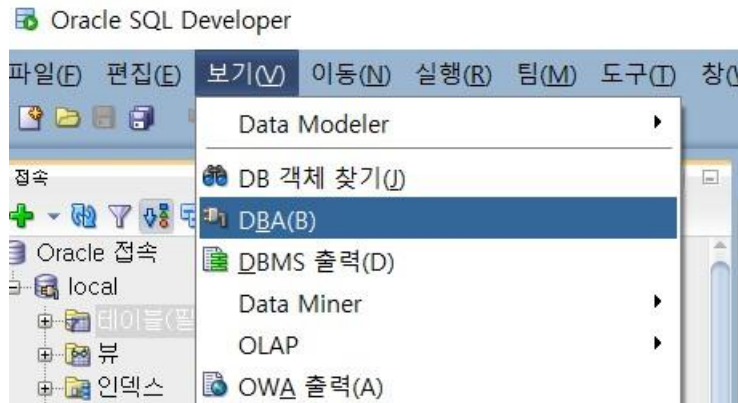


Table Space 생성

테이블스페이스명, 파일크기, 자동확장 설정

테이블스페이스 생성

이름(A): TEST

테이블스페이스 유형(B): 영구

파일 사양 속성 기본 매개변수 DDL

파일 사양(E): + X

FILE_SPECIFICATION1

이름(C): FILE_SPECIFICATION1

디렉토리(D):

파일 크기(E): 1 G

☐ 재사용(G)

☒ 자동 확장 설정(I)

다음 크기(J): 1 G

최대 크기(K): ☒ 제한 없음(L)

도움말(H) 확인 취소

사용자 계정 생성

사용자명, 비밀번호, 테이블 스페이스 설정

부여된 롤 : CONNECT, RESOURCE 체크

할당량 : TEST 테이블 스페이스 무제한 체크

사용자 계정 생성

계정 생성 시 오류 발생하는 경우

```
alter session set "_oracle_script"=true;
```

오라클 최신버전에서는 사용자 추가 시 C##으로 시작하는 이름으로 생성 가능
위 명령어 실행 시 일반 이름으로 생성 가능

사용자 계정 접속 테스트

새 접속 정보 추가

새로 만들기/데이터베이스 접속 선택

접속 이름	접속 세부정보
bigdb	bigdb@//local...
local	sys@//localho...
aangong	bigdb@//221....
shinhan	shinhanuser@...

Name: testuser

데이터베이스 유형: Oracle

사용자 정보: 프록시 사용자

인증 유형: 기본값

사용자 이름(U): testuser

비밀번호(P):

인증(L): 기본값

☒ 비밀번호 저장(V)

접속 유형(Y): 기본

세부정보: 고급

호스트 이름(A): localhost

포트(R): 1521

☒ SID(I): xe

☐ 서비스 이름(E):

상태: 성공

도움말(H) | 저장(S) | 지우기(C) | 테스트(T) | 접속(O) | 취소

실습데이터 생성

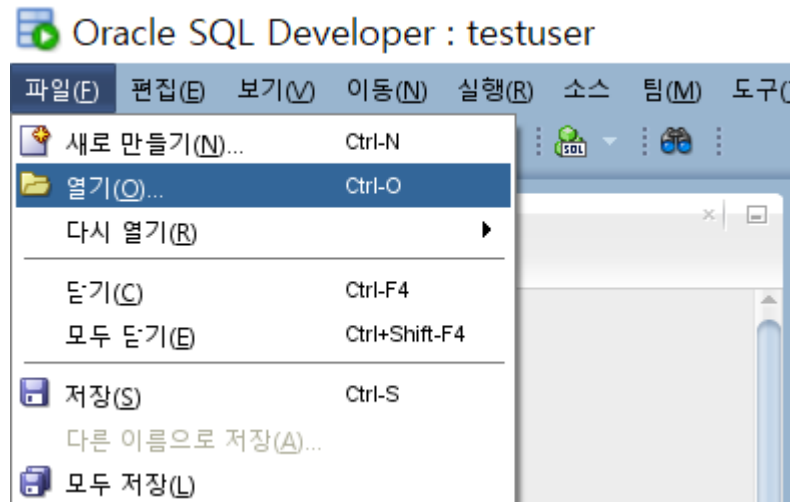
클래스룸 접속

실습데이터.sql 다운로드

SQL Developer 파일 > 열기

실습데이터.sql 열기

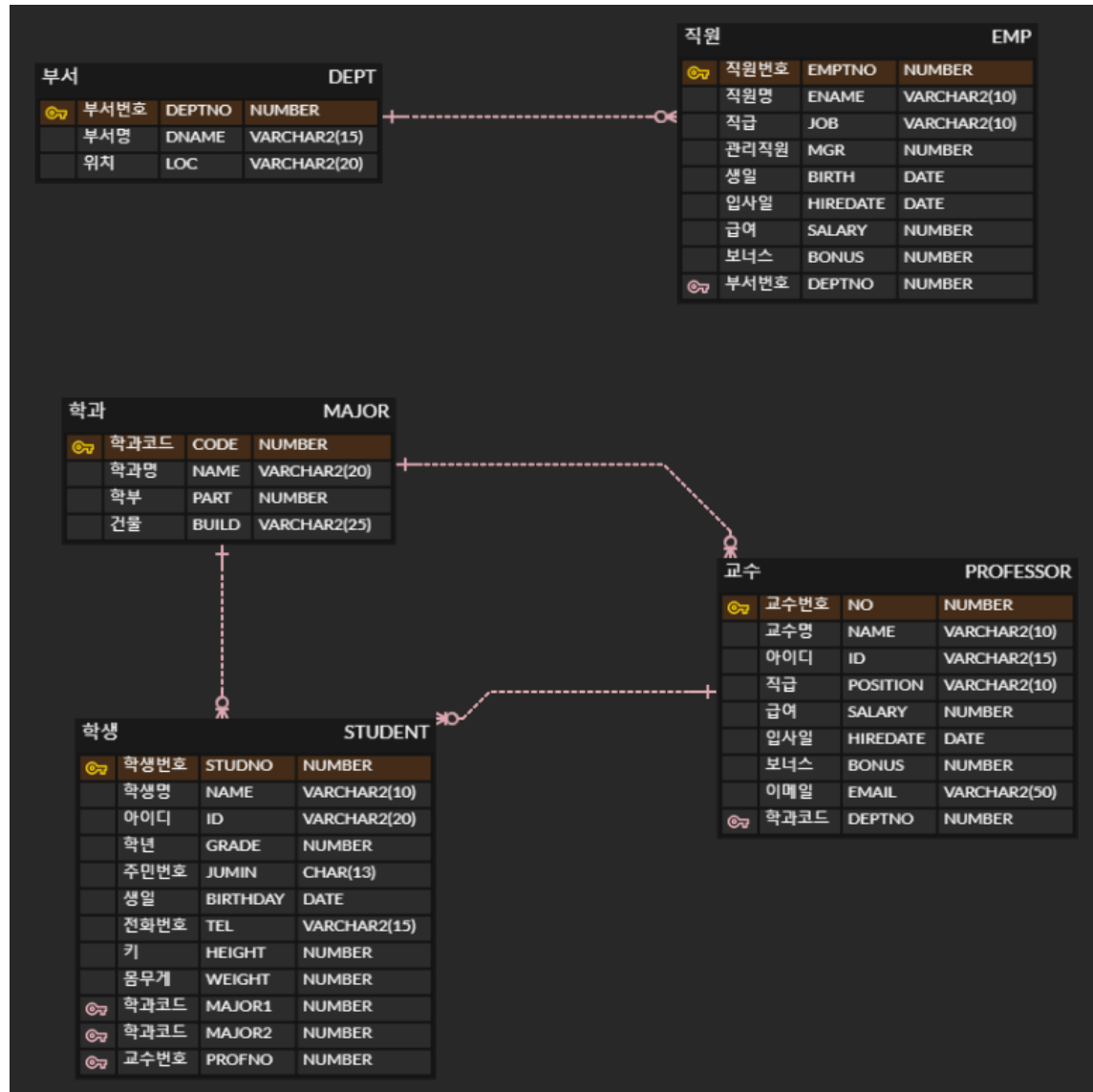
실행



실습환경 ERD

ERD

(Entity Relationship Diagram)



데이터 타입

Data Type (자료형)

구분	데이터 타입	설명
문자	CHAR(n)	n 크기만큼 고정된 문자 최대 2,000바이트
문자	VARCHAR(n) VARCHAR2(n)	n 크기만큼 가변길이의 문자 최대 4,000바이트
숫자	NUMBER(p,s)	숫자 타입 (p:정수 자리수, s:소수점 자리수)
날짜	DATE	날짜+시간 타입 최대 9999년 12월 31일

Structured Query Language

- ✓ 구조적 질의 언어
 - ✓ 데이터베이스와의 통신을 위한 언어
 - ✓ 데이터 조회, 정의, 조작하기 위한 언어
-
- 표준 언어 : 여러 데이터베이스에서 사용 가능
 - 간단하고, 사용하기 쉬워 배우기 쉽다.
 - 과거 프로그래머나 엔지니어들이 주로 사용했으나, 최근에는 데이터 활용성의 증가로 여러 분야에서 다양한 형태로 사용되고 있음.

! 언어를 가장 쉽게 배울 수 있는 방법은 직접 해보는 것 !

DML

(Data Manipulation Language)

데이터 조작 언어
데이터를 조회하거나, 검색, 등록, 수정, 삭제

DDL

(Data Definition Language)

데이터 정의 언어
테이블 생성, 수정, 변경, 삭제

DCL

(Data Control Language)

데이터베이스 접근 권한 제어 언어

TCL

(Transaction Control Language)

트랜잭션 (논리적 작업단위) 제어를 위한 언어

SQL 실습

SQL Developer의 워크시트 창에서 SQL문 작성

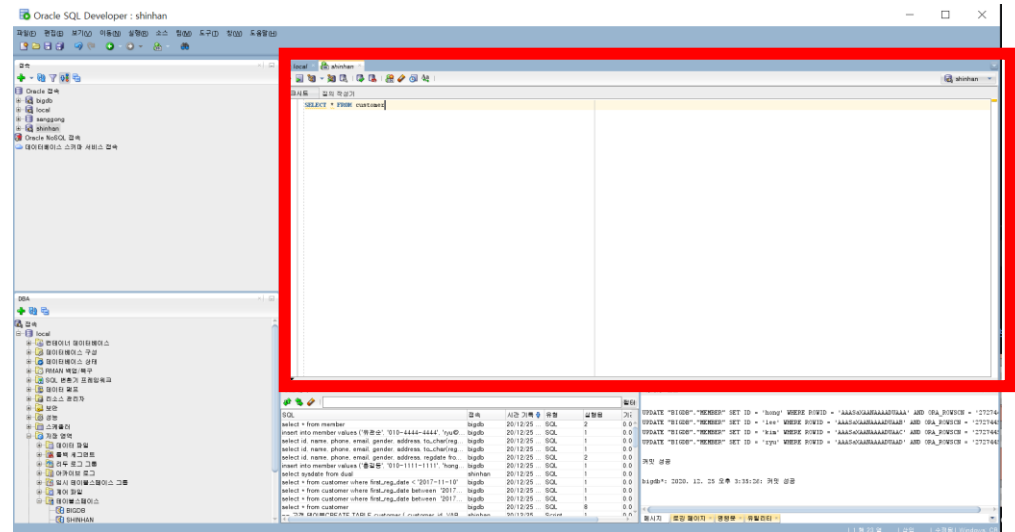
SQL문 실행

ctrl+enter 또는 F9

- 현재 커서가 있는 SQL문
- 드래그하여 선택된 SQL문

SQL문 작성 시

- ✓ 대소문자 구분 없음
- ✓ 띄어쓰기, 줄 바꿈 제한 없음
- ✓ 하나의 문장은 마지막에 ; 기호로 구분
- ✓ 특정 값 입력 시 문자열인 경우 '(작은따옴표)로 감싸서 입력



테이블 생성

테이블명 : MEMBER

컬럼명	타입
ID	VARCHAR2(10)
NAME	VARCHAR2(20)
AGE	NUMBER(5)
EMAIL	VARCHAR2(20)

Data Manipulation Language

- 데이터 조작 언어
- 테이블에 있는 데이터를 조회하거나 등록, 수정, 삭제할 때 사용

명령어	설 명
INSERT	테이블에 새로운 행 삽입
UPDATE	테이블에 있는 행의 내용 수정
DELETE	테이블의 행을 삭제
SELECT	테이블의 행을 조회

INSERT

테이블에 새로운 행 삽입

INSERT문의 구조

```
INSERT INTO 테이블명 [(열이름1, 열이름2, ...)]  
VALUES (값1, 값2, ...);
```

실습

MEMBER 테이블에 데이터(본인 정보)를 추가하시오.
컬럼명 : ID, NAME, AGE, EMAIL

```
INSERT INTO member (id, name, age, email)  
VALUES ('hong', '홍길동', 30, 'hong@gmail.com');
```

UPDATE

테이블에 있는 행의 내용 수정

UPDATE문의 구조

UPDATE 테이블명 SET

열이름1 = 값1, 열이름2 = 값2...

[WHERE 조건식];

실습

MEMBER 테이블에 데이터(본인 정보)를 수정하시오.

수정할 값 : AGE를 20으로 수정

조건식 : ID = '본인이추가했던아이디 '

```
UPDATE member SET age=20
```

```
WHERE ID='hong';
```

DELETE

테이블에 행 삭제

DELETE문의 구조

DELETE FROM 테이블명 [WHERE 조건식];

실습

STUDENT 테이블에 데이터(본인 정보)를 삭제하시오.

조건식 : ID = '본인이추가했던아이디 '

DELETE FROM member WHERE ID='hong';

COMMIT과 ROLLBACK

COMMIT

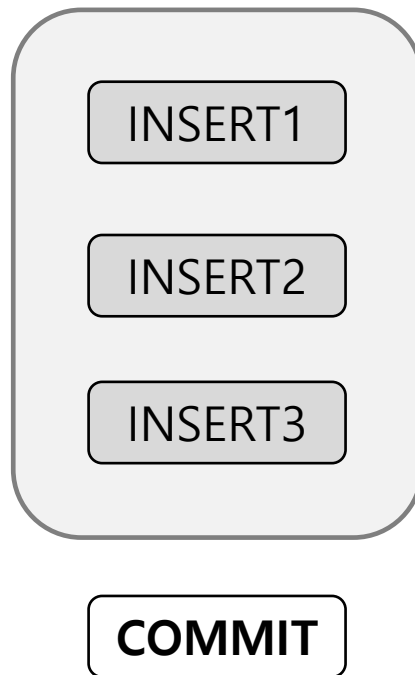
트랜잭션(작업 단위)를 정상적으로 데이터베이스에 적용
작업(INSERT, UPDATE, DELETE) 내용을 DB에 저장

ROLLBACK

작업 중 문제 발생 시 현재 트랜잭션의 변경 내역을 취소하고, 종료
트랜잭션 발생 이전 시점으로 되돌림
작업(INSERT, UPDATE, DELETE) 내용을 취소

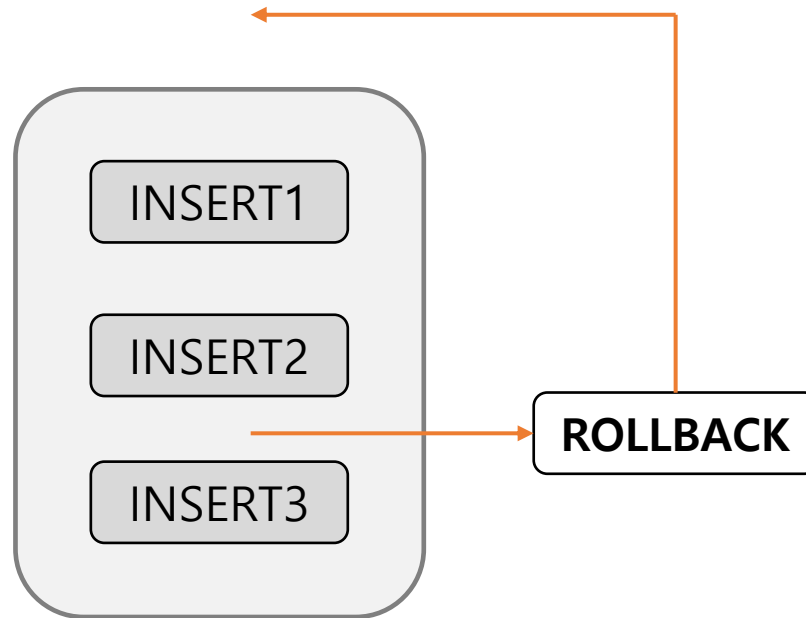
COMMIT

하나의 트랜잭션



INSERT1, INSERT2, INSERT3 모두 적용
(COMMIT 하지 않으면 모든 작업적용 안됨)

ROLLBACK



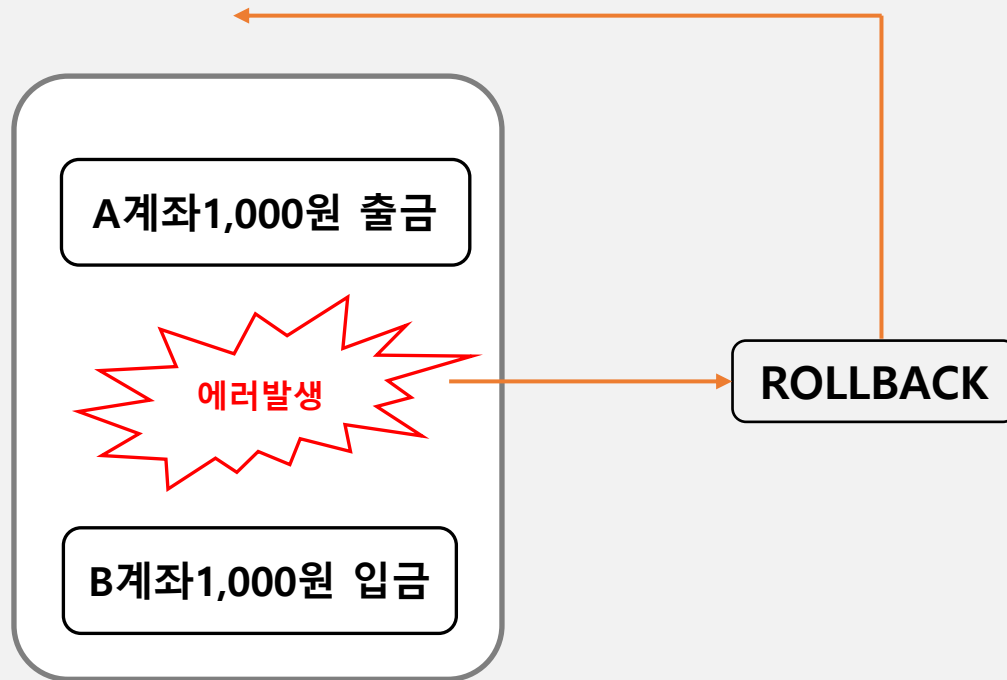
INSERT1, INSERT2 처리 후 에러 발생 시 **ROLLBACK**

INSERT1, INSERT2 작업 취소됨

(이전 시점으로 돌아감)

ROLLBACK 예시

A가 B에게 1,000원으로 이체해야 하는 경우



A계좌에 1000원 출금되는 작업과
B계좌에 1000원 입금되는 작업이 하나의 작업으로 처리 필요

실습데이터 준비

SELECT문 실습을 위한 데이터 준비

데이터 복구

실습데이터_오라클.sql 파일 다운로드

편집기로 파일 열기

내용 복사 후 SQL Developer에 붙여 넣기

전체 코드 실행


데이터 조회

SELECT문은 가장 자주 사용하게 되는 구문이며 데이터를 조회할 때 사용.

SELECT문을 사용할 때 반드시 먼저 고민해야 할 부분

- 어디서 가져올 것인가
- 어떻게 가져올 것인가
- 어떤 값을 출력할 것인가

SELECT문 구조

SELECT 열이름 [as 별칭]...  as(ALIAS) : 별명, 별칭
FROM 테이블명
[WHERE 조건식]
[ORDER BY 열이름 [ASC or DESC]];

SQL문 작성 규칙

Tip

문법적인 규칙은 아니지만, 실무에서 관례적으로 사용되는 규칙

1. SQL은 대소문자를 구분하지 않지만, 예약어는 대문자로, 나머지 (열명, 테이블명 등)은 소문자로 표기
2. 가독성을 위해 여러 줄, 공백을 이용해 작성(명령어의 일부는 나눌 수 없음)
3. 구조적(Structured) 코드는 들여쓰기 이용(가독성 향상)

전체 데이터 조회

SELECT문의 구조를 이용해 전체 데이터 조회

다른 DML과는 다르게 데이터가 변경되지 않으니 자유롭게 실행 가능

어디에서? → 직원 테이블에서

어떤 값을 출력? → 모든 데이터를

***** : 모든 열(컬럼)을 의미

```
SELECT * FROM emp;
```

Tip

실행 : ctrl+enter 또는 F9

컬럼명 지정

```
SELECT empno, ename, deptno FROM emp;
```

전체 데이터 조회

ALIAS(별칭) : 생략가능

```
SELECT empno AS 직원번호, ename AS 직원명, deptno 부서명 FROM  
emp;
```

리터럴값으로 출력 열 추가

```
SELECT empno, ename, deptno, '회사명' AS company FROM emp;
```

중복 제거

```
SELECT DISTINCT deptno FROM emp;
```

에러 메시지 확인

SQL문 작성 후 실행 시 에러가 발생 하는 경우(디버깅 하는 방법)

1. 데이터베이스에서 응답하는 에러 메시지 확인
2. 작성했던 SQL문에서 에러가 발생한 위치를 찾아 수정
3. 재실행

```
SELECT name FROM emp;
```

ORA-00904: "NAME": 부적합한 식별자

"컬럼명을 name이라고 잘못 입력했구나"

```
SELECT ename FROM emp;
```

전체 데이터 정렬해서 조회

ORDER BY 컬럼명 [ASC or DESC]

ASC : ascending, 오름차순 정렬(생략 시 기본값)

DESC : descending, 내림차순 정렬

SELECT문의 마지막에 위치하며, 여러 개 지정 시 순서대로 적용

어디에서? —————> 직원 테이블에서

어떻게? —————> 입사일순으로 정렬해서

어떤 값을 출력? —> 모든 데이터를

```
SELECT * FROM emp ORDER BY hiredate;
```

데이터 필터링

WHERE 키워드를 이용하여 특정 조건에 해당하는 행을 조회

WHERE 조건문이 없으면 모든 데이터 조회

어디에서? —————> 학생 테이블에서

어떻게? —————> 1학년 학생의

어떤 값을 출력? —> 모든 데이터를

```
SELECT * FROM student WHERE grade = 1;
```

퀴즈

기획부 직원의 이름과 부서번호를 조회 하시오.

데이터 필터링

급여가 800 보다 많은 직원 조회

어디에서? —————> 직원 테이블에서

어떻게? —————> 급여가 800보다 큰

어떤 값을 출력? —> 직원명과 급여

```
SELECT ename, salary FROM emp WHERE salary > 800;
```

WHERE 조건식의 비교 연산

- $A > B$: A가 B보다 크다
- $A \geq B$: A가 B보다 크거나 같다
- $A < B$: A가 B보다 작다
- $A \leq B$: A가 B보다 작거나 같다
- $A = B$: A와 B가 같다
- $A \neq B$: A와 B가 다르다

데이터 필터링

급여가 800 보다 많고, 보너스가 200보다 많은 직원 조회

어디에서? —————> 직원 테이블에서

어떻게? —————> 급여가 800보다 크고, 보너스도 200보다 큰

어떤 값을 출력? —> 직원명과 급여

```
SELECT ename, salary FROM emp WHERE salary > 800 AND  
bonus > 200;
```

WHERE 조건식의 논리 연산

- A and B : A도 참이고 B도 참
- A or B : A가 참이거나 B가 참

데이터 필터링

급여가 500 이상, 1000 이하 직원 조회

어디에서? —————> 직원 테이블에서

어떻게? —————> 급여가 500 이상, 1000 이하

어떤 값을 출력? —> 직원명과 급여

```
SELECT ename, salary FROM emp WHERE salary >= 500 AND  
salary <= 1000;
```

500 <= salary <= 1000

```
SELECT ename, salary FROM emp WHERE salary BETWEEN 500  
AND 1000;
```

데이터 필터링

기획부, 영업부 직원 조회

어디에서? —————> 직원 테이블에서

어떻게? —————> 부서코드가 20이거나, 40인

어떤 값을 출력? —> 직원명과 부서코드

```
SELECT ename, deptno FROM emp WHERE deptno = 20 OR deptno = 40;
```

```
SELECT ename, deptno FROM emp WHERE deptno IN (20,40);
```

부서코드가 20, 40이 아닌 직원

```
SELECT ename, deptno FROM emp WHERE deptno NOT IN (20,40);
```

데이터 필터링

- LIKE 연산자**
- 특정 단어를 포함하는 경우 조건
 - % : 0개 이상의 문자
 - _ : 1개의 문자

'김'씨 학생 조회

```
SELECT * FROM student WHERE name LIKE '김%';
```

'인'자로 끝나는 학생 조회

```
SELECT * FROM student WHERE name LIKE '%인';
```

데이터 필터링

- LIKE 연산자**
- 특정 단어를 포함하는 경우 조건
 - % : 0개 이상의 문자
 - _ : 1개의 문자

이름이 두 자인 이씨 학생 조회

```
SELECT * FROM student WHERE name LIKE '이|_';
```

이름이 김씨가 아닌 학생 조회

```
SELECT * FROM student WHERE name NOT LIKE '김%';
```

퀴즈

전화번호가 서울지역번호인 학생의 이름, 전화번호 조회

산술연산

숫자로된 값을 계산할 때 사용

+ (더하기), - (빼기), * (곱하기), / (나누기)

FROM 절을 제외한 모든 절에서 사용 가능

우선순위는 (), *, / , +, - 순

연산결과는 새로운 열로 출력

전체 직원의 이름과 급여, 연봉(급여 * 12) 조회

```
SELECT ename, salary, salary*12 FROM emp;
```

전체직원의 이름, 현재급여, 100만원 인상 급여, 10% 인상 급여 조회

```
SELECT ename, salary, salary+100, salary*1.1 FROM emp;
```

집합 연산

UNION(중복 제거)

UNION ALL (중복 허용)

```
SELECT
    studno, name, major1
FROM student
WHERE major1 = 202;
```

** 전공코드가 202인 학생정보 조회*

	STUDNO	NAME	MAJOR1
1	981213	장영	202
2	971211	유진성	202
3	961215	박재원	202
4	951215	하정환	202

```
SELECT
    studno, name, major1
FROM student
WHERE major2 = 101;
```

** 전공코드가 101인 학생정보 조회*

	STUDNO	NAME	MAJOR1
1	951215	하정환	202

집합 연산

UNION ALL (중복 허용)

SELECT

studno, name, major1

FROM student

WHERE major1 = 202

UNION ALL

SELECT

studno, name, major1

FROM student

WHERE major2 = 101;

*전공코드가 202인 학생정보와
부전공코드가 101인 학생정보 한꺼번에 조회*

	STUDNO	NAME	MAJOR1
1	981213	장영	202
2	971211	유진성	202
3	961215	박재원	202
4	951215	하정환	202
5	951215	하정환	202

집합 연산

UNION (중복 제거)

```
SELECT
    studno, name, major1
FROM student
WHERE major1 = 202
UNION
SELECT
    studno, name, major1
FROM student
WHERE major2 = 101;
```

*전공코드가 202인 학생정보와
부전공코드가 101인 학생정보 한꺼번에 조회*

	STUDNO	NAME	MAJOR1
1	951215	하정환	202
2	961215	박재원	202
3	971211	유진성	202
4	981213	장영	202

NULL

NULL

- 값이 없다는 의미
- 값 자체가 존재하지 않음
- 숫자의 0, 문자의 " 과는 다름
- 값이 없으므로 연산 불가

보너스가 없는 직원 조회

```
SELECT * FROM emp WHERE bonus IS NULL;
```

보너스가 있는 직원 조회

```
SELECT * FROM emp WHERE bonus IS NOT NULL;
```

NULL

NULL

- 값이 없다는 의미
- 값 자체가 존재하지 않음
- 숫자의 0, 문자의 " 과는 다름
- 값이 없으므로 연산 불가

직원의 이름과, 현재급여, 연봉(급여*12+보너스) 조회

```
SELECT ename, salary, salary*12+bonus FROM emp;
```

-보너스가 null인 경우 연산불가-

NULL인 경우 다른 값으로 처리해주는 함수 사용

```
SELECT ename, salary, salary*12+NVL(bonus,0) FROM emp;
```

-보너스가 null인 경우 0으로 처리-

NULL

NVL, NVL2 함수

- 값이 null인 경우 처리하는 함수
- NVL은 null인 경우만 체크
- NVL2는 null인 경우와 그렇지 않은 경우 따로 처리

NVL2(컬럼, 값이 null이 아닌 경우, 값이 null인 경우)

보너스가 없으면(null) 'X', 있으면 'O' 라고 출력

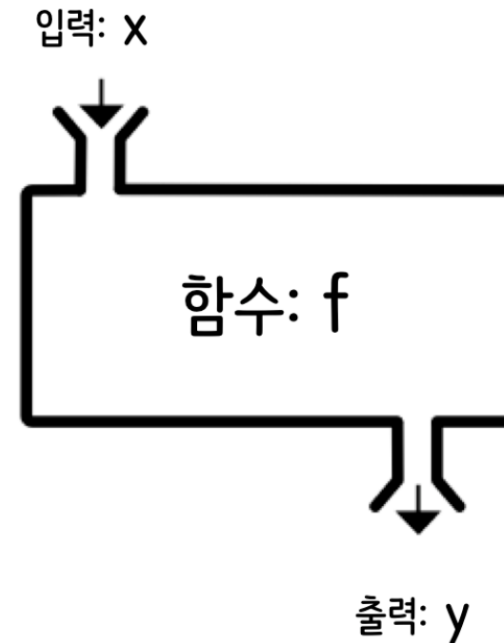
```
SELECT ename, salary, bonus, NVL2(bonus,'O', 'X') FROM emp;
```

함수

입력값을 넣어

어떤 작업을 처리한 결과를 반환

$$y = f(x)$$



함수는 사용하는 목적

- 데이터의 값을 계산하거나 조작 (단일 행 함수)
- 행을 그룹핑해서 계산한 요약값 (그룹 함수)
- 데이터 타입 변환

문자관련 함수

문자 타입 값에 사용되는 함수

자주 사용되는 문자열 함수

함수명	설 명
LOWER	소문자로 변경
UPPER	대문자로 변경
SUBSTR	문자열 일부 추출
REPLACE	문자열 치환
CONCAT	문자열 결합
LENGTH	문자열의 길이
INSTR	문자열 위치
TRIM	앞/뒤 문자열 제거
LTRIM	왼쪽 문자열 제거
RTIRM	오른쪽 문자열 제거

문자관련 함수

학생 이름, 아이디, 대문자 아이디, 소문자 아이디 출력

```
SELECT name, UPPER(id), LOWER(id) FROM student;
```

학생 이름, 성(첫 문자) 출력

```
SELECT name, SUBSTR(name, 1, 1) FROM student;
```

학생 이름, 성(첫 문자)를 #으로 치환해서 출력

```
SELECT name, REPLACE(name, SUBSTR(name, 1, 1), '#') FROM student;
```

문자관련 함수

직원 명, 직급, 직원명과 직급 결합 된 값 출력

```
SELECT ename, job, CONCAT(ename, job) FROM emp;
```

```
SELECT ename, job, ename || job FROM emp;
```

직원명과 직원명의 길이 출력

```
SELECT ename, LENGTH(ename) FROM emp;
```

학생명, 전화번호, ')'의 위치 출력

```
SELECT name, tel, INSTR(tel, ')') FROM student;
```


문자관련 함수

퀴즈

학생명, 전화번호, 지역번호 출력

```
SELECT name, tel, SUBSTR(tel, 1, INSTR(tel, '-')-1) FROM student;
```

교수명, 홈페이지주소, 'http://'를 제거한 주소 출력

```
SELECT name, url, LTRIM(url, 'http://') FROM professor;
```

두번째 값을 넣지 않으면 공백을 제거
이름에 공백이 들어간 경우 ' 홍길동 '

```
SELECT ' 홍길동 ', TRIM(' 홍길동 ') FROM dual;
```

숫자관련 함수

숫자 타입 값에 사용되는 함수

자주 사용되는 숫자 함수

함수명	설 명
ROUND	반올림
TRUNC	버림
CEIL	정수로 올림
FLOOR	정수로 내림
POWER	거듭제곱
SQRT	제곱근

숫자관련 함수

ROUND(컬럼명, 소수점 자리수)

```
SELECT  
    ROUND(13.141592), ROUND(13.141592,2),  
    ROUND(13.141592,-1)  
FROM dual;
```

TRUNC(컬럼명, 소수점자리수)

```
SELECT ROUND(3.141592,4) , TRUNC(3.141592,4) FROM dual;
```

숫자관련 함수

CEIL, FLOOR

```
SELECT CEIL(3.1415), FLOOR(3.1415) FROM dual;
```

POWER, SQRT

```
SELECT POWER(3,3) , SQRT(100) FROM dual;
```

날짜관련 함수

현재 날짜(시간) 가져오기

```
SELECT SYSDATE FROM dual;
```

날짜 사이의 일자 구하기

```
SELECT TRUNC(SYSDATE - TO_DATE('2021-01-01')) FROM dual;
```

날짜관련 함수

MONTHS_BETWEEN : 두 날짜 사이의 개월 수 구하기

학생명, 생일, 나이 출력

```
SELECT name, birthday, MONTHS_BETWEEN(SYSDATE, birthday)
FROM student;
```

TO_CHAR : 문자열로 변환하는 함수

```
SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD') FROM dual;
```

랭킹 함수

ROW_NUMBER() : 중복없이 전체 순서 지정

RANK() : 중복 랭킹 제외 후 순서 지정

DENSE_RANK() : 중복 랭킹 하나의 순서로 지정

```
SELECT
```

```
    salary,
```

```
    ROW_NUMBER() OVER(ORDER BY salary DESC) as rank1,
```

```
    RANK() OVER(ORDER BY salary DESC) as rank2,
```

```
    DENSE_RANK() OVER(ORDER BY salary DESC) as rank3
```

```
FROM emp;
```

** 급여 순으로 랭킹 출력*

공통 : OVER (정렬 기준)

조건 함수

DECODE

- ✓ 데이터의 값이 조건과 일치하는지 그렇지 않은지에 따라 다르게 출력
- ✓ 프로그래밍 언어에서 IF~ELSE와 같은 구조
- ✓ SQL문에서 정말 자주 사용되는 함수

DECODE(컬럼명, 조건값, 조건값과 같은 경우, 조건값과 다른 경우)

학생명과 학년이 1학년이면 신입생이라고, 그렇지 않으면 재학생이라고 출력

```
SELECT name, DECODE(grade, 1, '신입생', '재학생') FROM student;
```


조건 함수

CASE

- ✓ 특정 조건에 따라 출력할 데이터 설정
- ✓ 프로그래밍 언어에서 SWITCH 문과 유사한 구조
- ✓ 비교연산과 같은 조건 사용 가능

CASE 컬럼|데이터

WHEN 조건1 THEN 조건1이 참인 경우 출력값

WHEN 조건2 THEN 조건2가 참인 경우 출력값

...

ELSE 모든 조건이 일치하지 않는 경우 출력값

END

조건 함수

CASE

- ✓ 특정 조건에 따라 출력할 데이터 설정
- ✓ 프로그래밍 언어에서 SWITCH 문과 유사한 구조
- ✓ 비교연산과 같은 조건 사용 가능

학생명과 학년이 1학년이면 신입생이라고, 2학년이면 2학년, 3학년이면 3학년, 4학년이면 4학년이라고 출력

```
SELECT name,  
       CASE grade  
         WHEN 1 then '신입생' WHEN 2 then '2학년'  
         WHEN 3 then '3학년' ELSE '4학년'  
       END  
FROM student;
```

그룹핑

그룹함수

함수명	설 명
COUNT	행 개수
SUM	합계
AVG	평균
MAX	최대값
MIN	최소값
VARIANCE	분산
STDDEV	표준편차

학생수 출력

```
SELECT COUNT(*) FROM student;
```

교수 급여 합계

```
SELECT SUM(salary) FROM professor;
```

퀴즈

교수 연봉 합계

```
SELECT SUM(salary*12+NVL(bonus,0)) FROM professor;
```

교수 평균급여와 평균보너스 출력

```
SELECT AVG(salary), AVG(NVL(bonus,0)) FROM professor;
```

최대, 최소

```
SELECT MAX(salary), MIN(salary) FROM professor;
```

분산, 표준편차

```
SELECT VARIANCE(salary), STDDEV(salary) FROM professor;
```

GROUP BY

데이터 값이 같은 행끼리 묶어서 그룹화

전공별 학생수 출력

```
SELECT major1, COUNT(*) FROM student  
GROUP BY major1;
```

교수 직급별 인원수를 인원수가 많은 순으로 출력

```
SELECT position, COUNT(*) FROM professor  
GROUP BY position  
ORDER BY COUNT(*) DESC;
```

그룹 함수

ROLLUP : 하위 합계 생성

```
SELECT deptno, SUM(salary)
FROM emp GROUP BY ROLLUP(deptno); * 부서별 전체 합계
```

부서별 직급별 합계

```
SELECT deptno, job, SUM(salary)
FROM emp GROUP BY ROLLUP(deptno, job); * 부서별, 직급별 합계
```

GROUPING() 함수를 사용하면 합계는 1, 그렇지 않으면 0 출력

그룹 함수

GROUPING SETS : 컬럼별 별도 처리

```
SELECT deptno, job, SUM(salary)
FROM emp GROUP BY GROUPING SETS(deptno, job);
```

** 부서별, 직업별 별도 합계*

CUBE : 결합 가능한 모든 그룹핑(모든 조합의 수)

```
SELECT deptno, job, SUM(salary)
FROM emp GROUP BY CUBE(deptno, job);
```

** 부서와 직급 조합별 합계*

HAVING

GROUP BY 결과의 조건을 지정할 때 사용

전공별 평균 키 출력, (평균키가 170 이상인 학생만 조회하려면?)

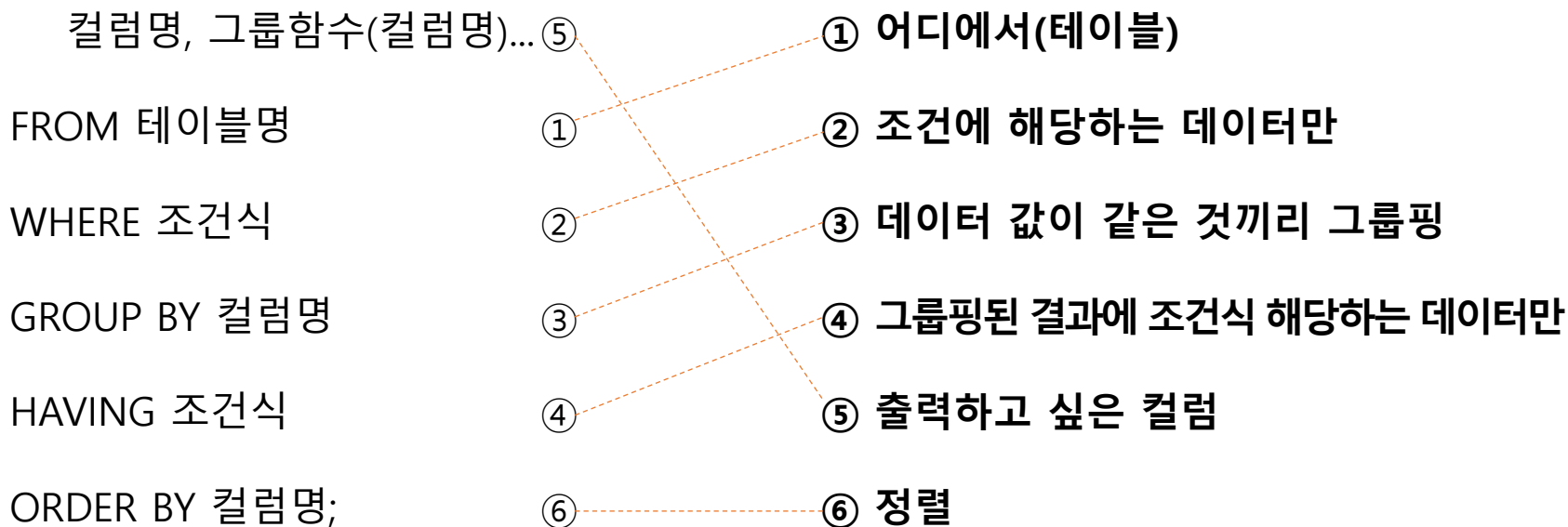
```
SELECT major1, AVG(height) FROM student  
GROUP BY major1  
HAVING AVG(height) >= 170;
```

교수 직급별 평균급여가 300이상인 부서코드와 평균 급여 출력

```
SELECT deptno, AVG(salary) FROM professor  
GROUP BY deptno  
HAVING AVG(salary) >= 300;
```

SELECT 실행 순서

SELECT



JOIN

한 개 이상의 테이블과 테이블을 연결하여 데이터를 조회하는 기법

종류	설 명
크로스(cross join)	가능한 모든 행 조인
등가 조인(equi join)	조건이 일치하는 결과
비등가 조인(non equi join)	조건이 일치하지 않는 결과
외부 조인(outer join)	양쪽 테이블의 한쪽만 조건이 일치해도 출력
자체 조인(self join)	자체 테이블에서 조인

equi join

가장 자주 사용되는 조인 방식으로,
테이블 간의 **PK(Primary Key)**를 이용해서 조인



- ✓ 직원 테이블에는 부서명 값이 없고, 부서 테이블에 존재함
- ✓ 이 부서테이블의 PK값인 부서번호를 직원테이블에 저장하고 있음
- ✓ 따라서 부서번호를 이용해서 직원테이블과 부서테이블을 조인

조인 SQL문의 기본구조

SELECT

테이블명1.컬럼명...,

테이블명2.컬러명...

FROM 테이블명1, 테이블명2

WHERE

테이블명1.컬럼명 = 테이블명2.컬럼명

직원명, 부서번호, 부서명 출력

```
SELECT emp.ename, dept.deptno, dept.dname  
FROM emp, dept  
WHERE emp.deptno = dept.deptno;
```

조인 SQL문의 기본구조 (ANSI 방식)

SELECT

테이블명1.컬럼명...,

테이블명2.컬러명...

FROM 테이블명1 JOIN 테이블명2

ON

테이블명1.컬럼명 = 테이블명2.컬럼명

직원명, 부서번호, 부서명 출력

```
SELECT emp.ename, dept.deptno, dept.dname  
FROM emp JOIN dept  
ON emp.deptno = dept.deptno;
```

퀴즈

학생명, 전공학과 코드, 학과명 출력

```
SELECT s.name, s.major1, m.name  
FROM student s JOIN major m  
ON s.major1 = m.code;
```

학생명, 학과명, 지도교수명 출력

```
SELECT s.name, m.name, p.name  
FROM student s JOIN major m  
ON s.major1 = m.code  
JOIN professor p  
ON s.profno = p.no;
```

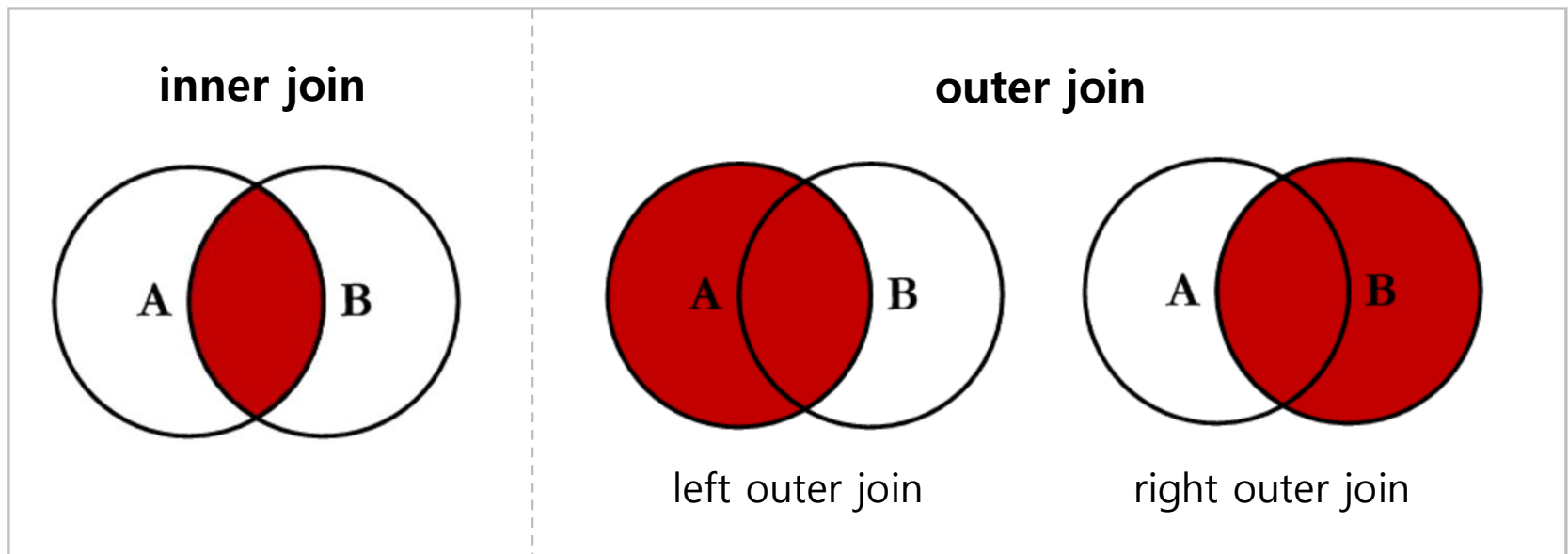
outer join

outer join

양쪽 테이블 모두 조건이 만족하지 않아도, 한쪽 테이블의 데이터를 모두 출력 해야 하는 경우 사용

예) 쇼핑몰에서 모든 회원의 주문건수 출력

- *equi join*은 *inner join*으로 동작하며, 주문했던 회원만 출력됨
- 우리가 원하는 결과는 주문하지 않은 고객은 주문건수가 0으로 출력되길 원함



left outer join (oracle)

학생명, 지도교수명 출력 (지도교수가 없는 학생도 출력)

```
SELECT s.name, p.name  
FROM student s, professor p  
WHERE s.profno = p.no(+);
```

학생명, 지도교수명 출력 (지도학생이 없는 교수도 출력)

```
SELECT s.name, p.name  
FROM student s, professor p  
WHERE s.profno(+) = p.no;
```

left outer join(ansi)

학생명, 지도교수명 출력 (지도교수가 없는 학생도 출력)

```
SELECT s.name, p.name  
FROM student s LEFT JOIN professor p  
ON s.profno = p.no;
```

학생명, 지도교수명 출력 (지도학생이 없는 교수도 출력)

```
SELECT s.name, p.name  
FROM student s RIGHT JOIN professor p  
ON s.profno = p.no;
```

connect by(계층형 조회)

트리구조로 조회 (Oracle 전용)

```
SELECT level, empno, mgr, ename
FROM emp
START WITH MGR IS NULL      * 시작 조건
CONNECT BY PRIOR empno = mgr; * 조인 조건
```

LPAD 함수를 사용해서 출력

```
SELECT level, LPAD(' ', 4*(level-1)) || empno AS empno,
        mgr, ename
FROM emp
START WITH MGR IS NULL
CONNECT BY PRIOR empno = mgr;
```

sub-query

select 안에 select 구문이 포함된 구문

- 단일 행 서브쿼리 : 서브쿼리의 결과가 1개인 경우
- 다중 행 서브쿼리 : 서브쿼리의 결과가 2개 이상인 경우

서브쿼리에 사용되는 연산자 종류

구 분	종 류
단일 행	=, !=, >, >=, <, <=
다중 행	IN, NOT IN, ANY, ALL

sub-query(단일 행)

양준혁 직원보다 급여가 많은 직원을 출력

- 양준혁 직원의 급여 확인
- 이 급여보다 많은 직원 출력

위 두개의 SQL을 하나의 SQL문으로 처리

```
SELECT ename, salary
```

```
FROM emp
```

```
WHERE salary > (SELECT salary FROM emp WHERE ename = '양준혁');
```

sub-query

다중 행 서브쿼리 (IN, NOT IN)

1학년 학생들의 키와 같은 키를 가지고 있는 2학년 학생명과 학년, 키 출력

```
SELECT name, grade, height  
FROM student  
WHERE height IN (SELECT height FROM student WHERE grade=1)  
AND grade=2;
```

1학년 학생들의 키와 같은 키와 다른 2학년 학생명과 학년, 키 출력

NOT IN 사용

sub-query

다중 행 서브쿼리 (ANY, ALL)

2학년 학생들의 가장 작은 키보다 큰 2학년 학생명과 학년, 키 출력
하나라도 만족하면 조회 (OR와 같은 결과)

```
SELECT name, grade, height
FROM student
WHERE height >= ANY (SELECT height FROM student WHERE grade=2)
AND grade=2;
```

서브쿼리의 모든 결과를 만족해야 할 경우 **ALL** 사용

sub-query

스칼라 서브쿼리

컬럼자리에 서브쿼리가 들어가는 경우

직원명, 부서코드, 부서명 출력

```
SELECT ename, d.deptno, dname  
FROM emp e JOIN dept d  
ON e.deptno = d.deptno;
```

```
SELECT ename, deptno,  
       (SELECT dname FROM dept WHERE deptno=emp.deptno) dname  
FROM emp;
```


sub-query

inline view 서브쿼리

- FROM 절 뒤에 들어가는 서브쿼리
- view : 실제 물리적인 테이블이 아니라 가상의 테이블

직원명, 급여, 직급별 평균 급여 출력

```
SELECT ename, e.deptno, e.salary, v.salary  
FROM emp e JOIN  
      (SELECT deptno, AVG(salary) salary FROM emp GROUP BY deptno) v  
ON e.deptno = v.deptno;
```

sub-query

페이징 처리

- 어플리케이션 목록 출력 시 해당 페이지 데이터 출력 용도
- 해당 페이지의 데이터 목록 조회

예) 회원 목록 2페이지 조회

ROWNUM 키워드를 통해 서브쿼리 사용

```
SELECT *  
FROM (  
    SELECT ROWNUM as rnum, a.*  
    FROM (  
        SELECT *  
        FROM student order by grade desc  
    ) a  
    ) b where b.rnum between 1 and 10;
```

페이지당 10개씩 출력되는 학생목록 첫 페이지 조회

*시작값 : (페이지-1) * 페이지당개수 + 1*

*끝값 : 페이지 * 페이지당개수*

sub-query

서브쿼리 사용 가능 위치

- WHERE 조건문
- 컬럼 (스칼라 서브쿼리)
- HAVING 구문
- FROM 구문 (inline view)

Tip

서브쿼리는 데이터 처리 속도에 영향을 줄 수 있어 사용 시 유의

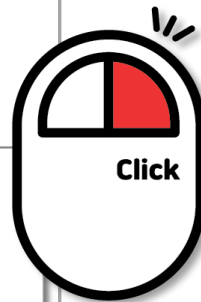
엑셀 출력 가능

**SELECT문
실행 결과**

**엑셀 파일로
출력 가능**
(내보내기 이용)

엑셀파일로 생성 후
별도 보고서 양식이나
차트 등의 추가 작업
가능

ENAME	DEPTNO	SALARY	SALARY_1
1 서민구	10	1800	1800
2 류현진	30	1000	680
3 손흥민	20	900	700
4 오승환	40	900	650
5 선동열	20	700	700
6 최동원	40		
7 박세리	20		
8 양준혁	50		
9 장종훈	40		
10 이종범	30		
11 송진우	30		



출력화면에서
마우스 오른쪽 클릭(익스포트)

강의 내용

curriculum

내용

데이터 모델링
데이터 정의어

객체 (테이블, 뷰, 시퀀스, 트랜잭션)
사용자 권한 관리

PL/SQL 기초 문법

프로시저, 함수, 트리거

오라클 아키텍처

인덱스 개념 및 활용

데이터베이스 설계

데이터 모델링

- 현실 세계에서 일어나는 사건들을 데이터화하는 과정
- 실제 현실은 너무 복잡하므로 개념화(추상화)하여 단순하게 표현
- 모델링을 하기 위해서는 고객과의 의사소통을 통해 고객의 업무 프로세스를 이해
- 업무 프로세스를 추상화하고, 분석/설계하면서 점점 상세하게 설계
- 업무 프로세스를 이해하고, 규칙을 정의해서 데이터 모델로 표현

추상화, 단순화, 명확화

데이터 모델링 단계

개념적 모델링



논리적 모델링



물리적 모델링

개념적 모델링 / 논리적 모델링

개념적 모델링

- 현실 세계에서 일어나는 사건들을 데이터 관점으로 표현
- 개념적 모델링의 산출물인 개념적 ERD를 만드는 과정
- 복잡하지 않고, 중요한 부분 위주로 모델링
- 엔티티와 속성 도출

논리적 모델링

- 개념적 모델링을 논리적 모델링으로 변환
- 식별자 도출, 필요한 릴레이션 정의
- 정규화 수행

엔티티와 속성

엔티티 (Entity)

- 엔티티는 업무에서 관리해야 하는 데이터 집합의 의미
 - 저장되고 관리되어야 하는 데이터
 - 엔티티는 개념, 사건, 장소 등의 명사
- 예) 부서, 사원

속성 (Attribute)

- 데이터의 가장 작은 논리적 단위
- 하나의 엔티티는 한 개 이상의 속성으로 구성
- 엔티티의 특성, 상태 등을 기술

예) 부서 엔티티의 속성 : 부서번호, 부서명

사원 엔티티의 속성 : 사원번호, 사원명

엔티티 도출

엔티티는 고객의 비즈니스 프로세스에서 관리되어야 하는 정보 추출

비즈니스 프로세스

- 고객은 회원가입을 한다.
- 회원가입 시 아이디, 비밀번호, 이름, 전화번호 입력
- 회원가입을 하기위해 반드시 하나의 계좌 개설
- 고객은 여러 개의 계좌 개설 가능
- 계좌를 개설할 때는 계좌번호, 계좌명, 예금, 개설지점, 계좌 담당자 입력

고객

회원ID

비밀번호
이름
전화번호

계좌

계좌번호

계좌명
예금
개설지점
계좌 담당자

엔티티 특징

특징	설명
식별자	<ul style="list-style-type: none">- 엔티티는 유일한 식별자가 있어야 함.- 예) 회원ID, 계좌번호
인스턴스 집합	<ul style="list-style-type: none">- 2개 이상의 인스턴스가 있어야 함.- 예) 2명 이상의 고객 정보
속성	<ul style="list-style-type: none">- 엔티티는 반드시 속성을 가지고 있어야 함.- 예) 고객 엔티티에 ID, 비밀번호, 이름, 전화번호 속성
관계	<ul style="list-style-type: none">- 엔티티는 다른 엔티티와 최소 한 개 이상의 관계- 예) 고객은 계좌를 개설
업무	<ul style="list-style-type: none">- 엔티티는 업무에서 관리되어야 하는 집합- 예) 고객, 계좌

속성 (Attribute)

엔티티가 가지는 항목 (테이블의 컬럼)



강사

엔티티는 속성들에 의해 설명된다.

속성들

- 이름
- 주소
- 생년월일
- 계약일자
- 전문분야

속성은 업무에서 필요로 하는 인스턴스에서 관리하고자 하는
의미상 더 이상 분리되지 않는 최소의 데이터 단위

속성 (Attribute)

- 분해여부에 따른 속성의 종류

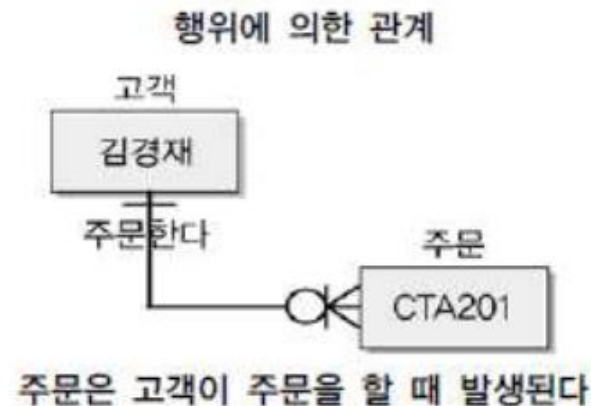
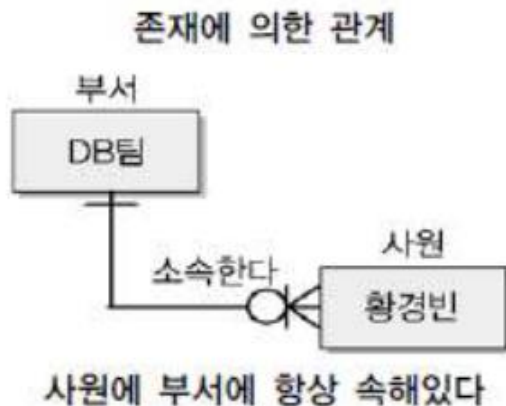
종류	설명
단일 속성	하나의 의미로 구성된 것으로 ID, 이름 등
복합 속성	<ul style="list-style-type: none">- 여러 개의 의미가 있는 것 (주소 등)- 주소는 시, 군, 동 등으로 분해 가능
다중값 속성	<ul style="list-style-type: none">- 속성에 여러 개의 값을 가질 수 있는 것 (취미)- 다중값 속성은 엔티티로 분해

- 특성에 따른 속성의 종류

종류	설명
기본 속성	<ul style="list-style-type: none">- 비즈니스 프로세스에서 도출되는 본래의 속성
설계 속성	<ul style="list-style-type: none">- 데이터 모델링 과정에서 발생하는 속성- 유일한 값을 부여한다.- 예) 상품코드, 지점코드 등
파생 속성	<ul style="list-style-type: none">- 다른 속성에 의해 만들어지는 속성- 예) 합계, 평균 등

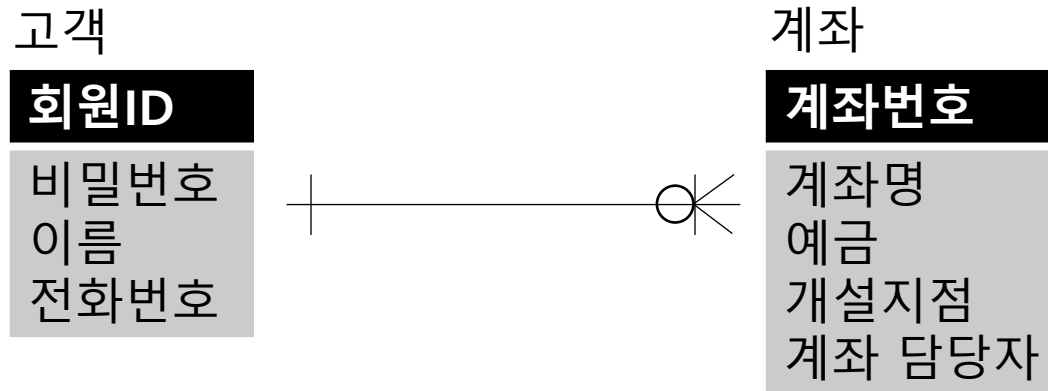
관계 (Relationship)

- 관계는 엔티티 간의 관련성을 의미하며, 존재관계와 행위관계로 분류
- 존재 관계 : 엔티티 간의 상태를 의미. 고객이 은행에 회원가입을 하면, 관리점이 할당되고, 그 할당점으로 관리점에서 고객을 관리
- 행위 관계 : 엔티티 간의 어떤 행위가 있는 것. 계좌를 사용해서 주문을 발주하는 관계. 예) 보험회사에서 보험상품을 만들고, 가입하는 것



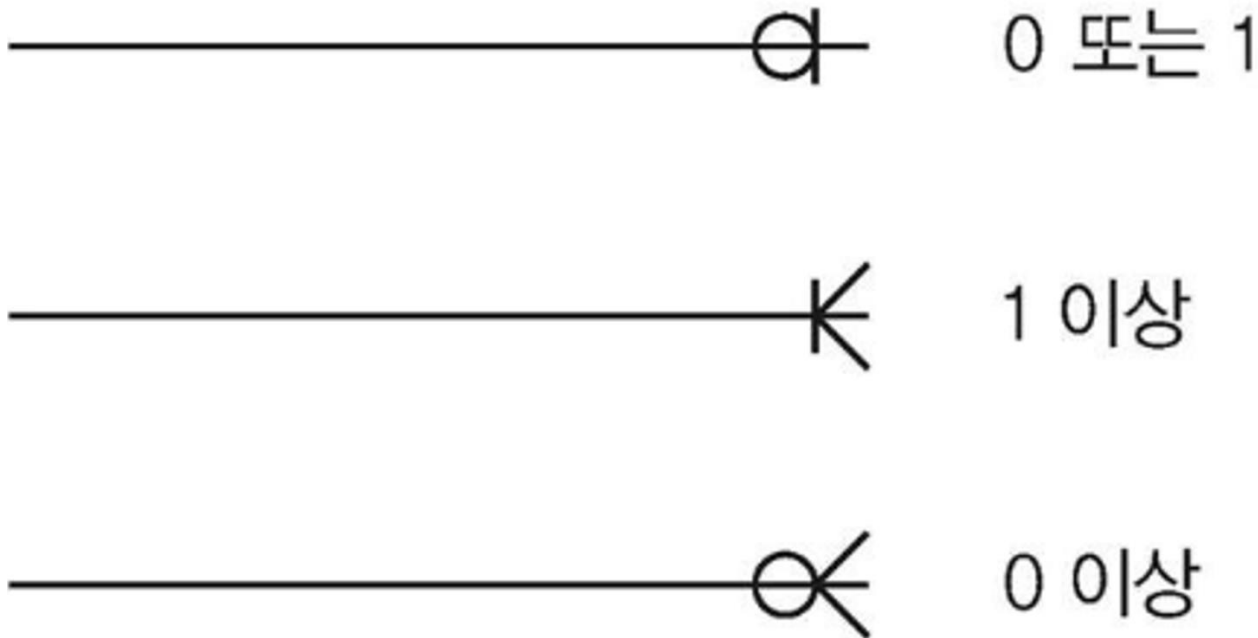
관계 차수(Relationship Cardinality)

- 두 개의 엔티티 간의 관계에 참여하는 수
- 예) 한 명의 고객은 여러 개의 계좌 개설 가능 (1대N의 관계)



관계 차수(Relationship Cardinality)

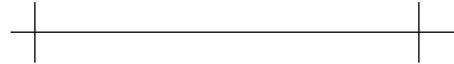
- 필수적 관계와 선택적 관계



관계 차수(Relationship Cardinality)

일대일 (1 : 1)

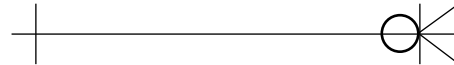
사원



병역

일대다 (1 : N)

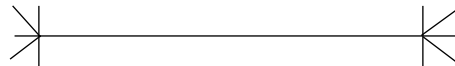
부서



사원

다대다 (N : M)

주문



제품

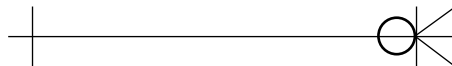
식별관계(Identification Relationship)

- 식별관계란 고객 엔티티의 기본키인 회원ID를 계좌 엔티티의 기본키의 하나로 공유하는 것
- 고객과 계좌 엔티티에서 고객은 독립적으로 존재할 수 있는 강한 개체 (Strong Entity)
- 강한 개체는 어떤 다른 엔티티에게 의존하지 않고 독립적으로 존재 가능하며, 다른 엔티티와 관계를 가질 때 다른 엔티티에게 기본키를 공유
- 식별관계란 고객 엔티티의 기본키인 회원ID를 계좌 엔티티의 기본키의 하나로 공유하는 것

고객

회원ID

비밀번호
이름
전화번호



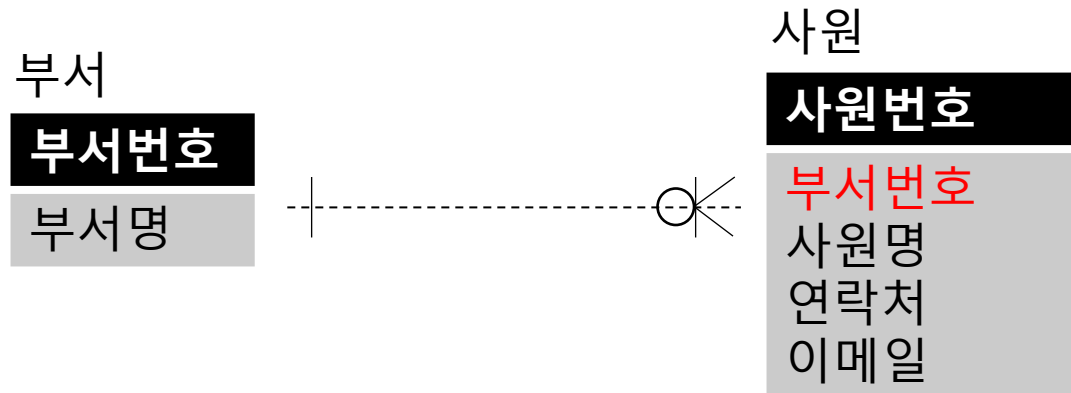
계좌

계좌번호

회원ID
예금
개설지점
계좌 담당자

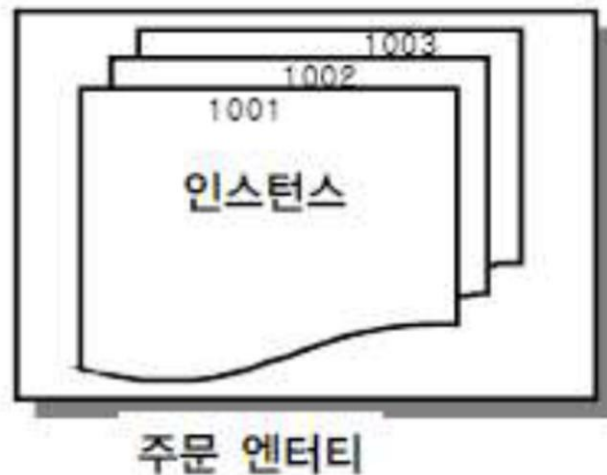
비식별관계(Non-Identification Relationship)

- 강한 개체의 기본키를 다른 엔티티의 기본키가 아닌 일반 컬럼으로 관계를 가짐
- 실선으로 표현



식별자(Identifier)

- 엔티티 내에서 인스턴스들을 구분할 수 있는 구분자
 - 엔티티를 대표할 수 있는 유일성을 만족하는 속성
- 예) 회원아이디, 계좌번호, 직원번호, 주문번호 등



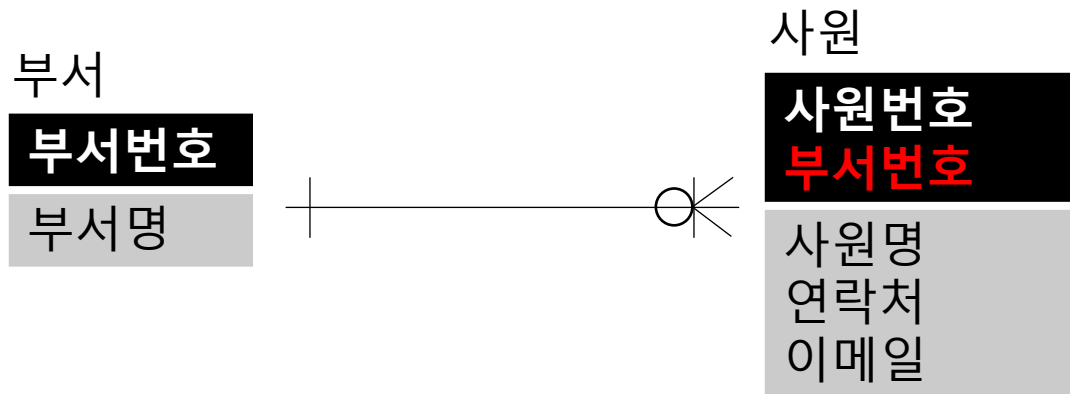
주식별자(Primary Key, PK, 기본키)

- 주식별자는 유일성과 최소성을 만족하는 키
- 엔티티를 대표할 수 있어야 함
- 엔티티의 인스턴스를 유일하게 식별
- NOT NULL + UNIQUE 속성 기본 부여

특징	내용	비고
유일성	주식별자에 의해 엔티티내에 모든 인스턴스들을 유일하게 구분함	예) 사원번호가 주식별자가 모든 직원들에 대해 개인별로 고유하게 부여됨
최소성	주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 함	예) 사원번호만으로도 고유한 구조인데 사원분류코드+사원번호로 식별자가 구성될 경우 부절한 주식별자 구조임
불변성	주식별자가 한 번 특정 엔티티에 지정되면 그 식별자의 값은 변하지 않아야 함	예) 사원번호의 값이 변한다는 의미는 이전기록이 말소되고 새로운 기록이 발생하는 개념임
존재성	주식별자가 지정되면 반드시 데이터 값이 존재 (Null 안됨)	예) 사원번호 없는 회사직원은 있을 수 없음

외래식별자(Foreign Key, FK, 외래키)

- 관계가 있는 두 엔티티를 부모, 자식 엔티티로 구분
- 부모 엔티티의 주식별자 속성을 자식 엔티티의 속성으로 추가



정규화(Normalization)

- 데이터의 일관성, 최소한의 데이터 중복, 최대한의 데이터 유연성을 위한 방법이며 데이터를 분해하는 과정
- 데이터 중복을 제거하고 데이터 모델의 독립성을 확보하기 위한 방법
- 정규화를 수행하면 비즈니스에 변화가 발생하여도 데이터 모델의 변경을 최소화 할 수 있음
- 제1정규화부터 제5정규화까지 있지만, 실제로 제3정규화까지만 수행

**사원번호
부서코드**

이름
전화번호
주소
부서명

- 직원정보 테이블은 정규화를 수행하지 않은 상태로 부서와 직원정보를 하나로 합쳐 둔 상태
- 새 직원이 추가되어 부서코드가 없거나, 부서만 추가되는 경우 이상 현상 (Anomaly) 발생

정규화(Normalization)

직원정보

사원번호

부서코드(FK)
이름
전화번호
주소

부서정보

부서코드

부서명

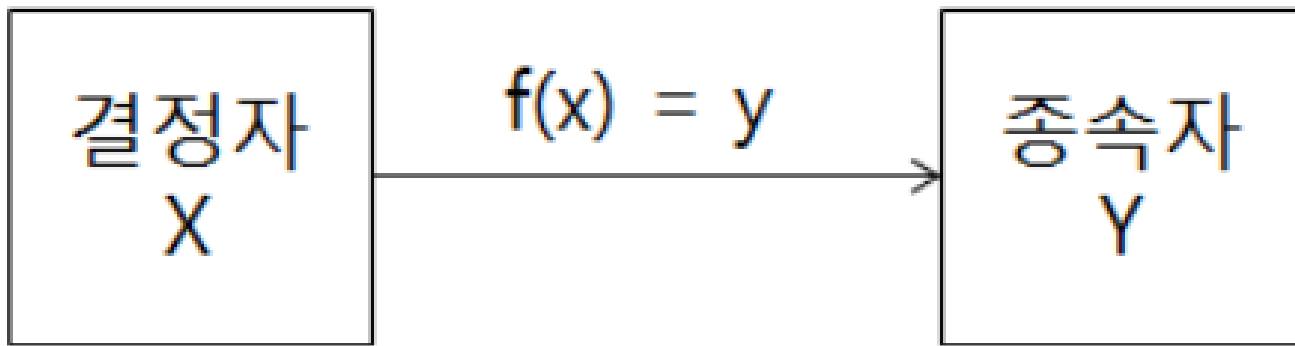
- 정규화된 모델은 테이블이 분해됨.
- 직원 테이블과 부서 테이블 간에 부서코드로 조인(join)을 수행하여 하나의 합집합으로 조회
- 정규화를 수행하면 불필요한 데이터를 입력하지 않아도 되기 때문에 중복 데이터가 제거

정규화 절차

정규화 절차	설명
제1정규화	<ul style="list-style-type: none"> - 속성(Attribute)의 원자성을 확보한다. - 기본키(Primary Key)를 설정한다.
제2정규화	기본키가 2개 이상의 속성으로 이루어진 경우 부분 함수 종속성을 제거(분해)한다.
제3정규화	<ul style="list-style-type: none"> - 기본키를 제외한 컬럼 간에 종속성을 제거한다. - 이행 함수 종속성을 제거한다.
BCNF	기본키를 제외하고 후보키가 있는 경우, 후보키가 기본키를 종속시키면 분해한다.
제4정규화	여러 칼럼들이 하나의 칼럼을 종속시키는 경우 분해하여 다중 값 종속성을 제거한다.
제5정규화	조인에 의해 종속성이 발생하는 경우 분해한다.

함수적 종속성 (Functional Dependency)

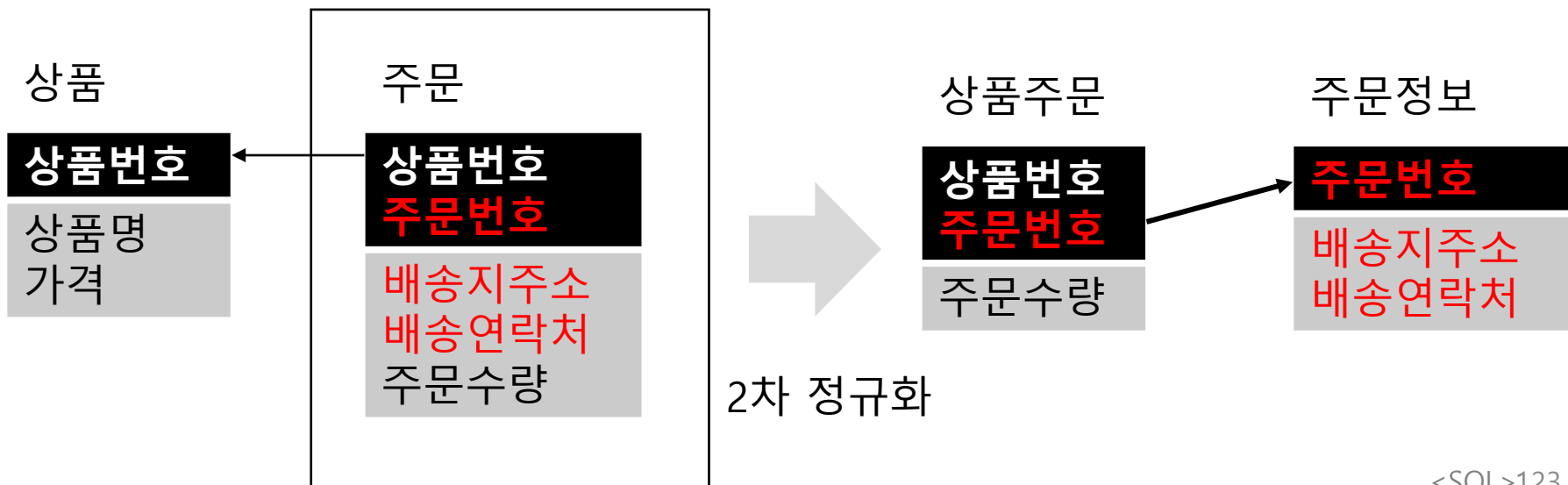
- 데이터들이 어떤 기준값에 의해 종속되는 현상
- 결정자(X)가 변화하면 종속자(Y)도 변화함



제1정규화는 함수적 종속관계에서 기본키를 작성하는 것

제2정규화

- 부분 함수 종속성 : 기본키가 2개 이상의 컬럼으로 이루어진 경우
- 부분 함수 종속성이 발생하면 분해(테이블 분리)
- 기본키가 하나의 컬럼으로 이루어지면 제2정규화는 생략
- 특정 컬럼에만 종속된 컬럼이 없어야 함
- 예) 주문내역에 존재하는 상품명, 상품금액



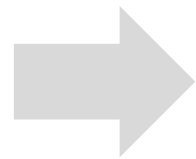
제3정규화

- 이행 함수 종속성 : 기본키를 제외하고 컬럼간에 종속성이 발생하는 것
- 제3정규화는 이행 함수 종속성을 제거
- 제1정규화와 제2정규화를 수행한 다음 수행

주문정보

주문번호

주문금액
주문수량
주문일자
고객번호
고객아이디
고객명



3차 정규화

주문정보

주문번호

주문금액
주문수량
주문일자
고객번호

고객정보

고객번호

고객아이디
고객명

BCNF (Boyce-Codd Normalization Form)

- 복수의 후보키가 있고, 후보키들이 서로 중첩된 상태.
- 교수가 후보키(최소성과 유일성을 만족)이고 교수가 과목을 함수적으로 종속하는 경우 분해
- 새로운 교수 테이블 생성하여 기본키는 교수, 컬럼은 과목

수강신청

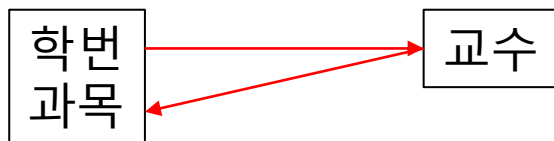
학번	과목	교수
1	컴퓨터공학	홍길동
2	멀티미디어	이순신
2	컴퓨터공학	홍길동

수강신청

학번	교수
1	홍길동
2	이순신
2	홍길동

교수

교수	과목
홍길동	컴퓨터공학
이순신	멀티미디어



정규화와 성능

- 테이블을 분해해서 데이터 중복을 제거하기 때문에 모델의 유연성을 높임
- 데이터 조회 시 join을 유발하기 때문에 리소스(CPU, 메모리) 사용률이 높아짐.
- 테이블간의 조인 시 건수 * 건수의 비교 필요(데이터 양이 증가하면 효율성 떨어짐)
- 비효율 문제를 해결하기 위해 인덱스(index)와 옵티마이저(Optimizer)를 통해 해결 (완벽한 해결은 아님)

반정규화 (De-Normalization)

데이터베이스의 성능 향상을 위하여, 데이터 중복을 허용하고 조인을 줄이는 데이터베이스 성능 향상 방법

조회 속도는 향상되지만, 모델의 유연성은 낮아짐

반정규화를 수행해야 하는 경우

- 정규화에 충실하면 종속성, 활용성은 향상되지만 수행 속도가 느려지는 경우
- 다량의 범위를 자주 처리해야 하는 경우
- 특정 범위의 데이터만 자주 처리하는 경우
- 요약/집계 정보가 자주 요구되는 경우

반정규화 기법

계산된 컬럼 추가

- 총주문금액, 일자별 판매금액 등을 미리 계산하고, 그 결과를 특정 컬럼 추가

테이블 수직분할

- 하나의 테이블을 컬럼기준으로 두 개 이상의 테이블로 분할

테이블 수평분할

- 하나의 테이블에 있는 값을 기준으로 분할

테이블 병합

- 1:1, 1:N 관계의 테이블을 병합하여 성능 향상 (데이터 중복 발생) <SQL>128

물리적 모델링

- 데이터베이스를 선정하여 ERD의 요소들을 실제 데이터베이스의 요소들로 변환

논리적 모델링
엔티티
속성
주식별자
외래식별자
관계



물리적 모델링
테이블
컬럼
기본키
외래키
관계

ERD Cloud

<https://www.erdcloud.com/>

- ✓ 고객과 계좌에 대한 엔티티 도출
- ✓ 각 엔티티의 속성 정의
- ✓ 엔티티간의 관계 정의

- 데이터베이스를 선정하여 ERD의 요소들을 실제 데이터베이스의 요소들로 변환

CREATE TABLE문 구조

```
CREATE TABLE [스키마명.] 테이블명  
(컬럼명, 데이터타입 [기본값] [제약조건] , ...);
```

테이블 생성

컬럼명	데이터 타입	크기
empno	number	5
ename	varchar2	10
salary	number	5

```
CREATE TABLE emp_ex (  
    empno number(5),  
    ename varchar2(10),  
    salary number(5)  
);
```

테이블 변경

- 새로운 컬럼을 추가하거나, 수정, 삭제

emp_ex 테이블에 날짜타입의 birthday 컬럼 추가

```
ALTER TABLE emp_ex ADD (birthday date);
```

emp_ex 테이블에 ename 컬럼 varchar2(20)으로 수정

```
ALTER TABLE emp_ex MODIFY ename varchar2(20);
```

테이블 변경

emp_ex 테이블에 birthday 컬럼 제거

```
ALTER TABLE emp_ex DROP COLUMN birthday;
```

하나의 컬럼만 삭제 가능

emp_ex 테이블명 변경

```
ALTER TABLE emp_ex RENAME TO emp_ex2;
```

또는

```
RENAME emp_ex TO emp_ex2;
```

테이블 삭제

해당 테이블의 모든 데이터를 제거

```
TRUNCATE TABLE emp_ex;
```

emp_ex 테이블 제거

```
DROP TABLE emp_ex;
```

다른 테이블에서 참조되는 경우 삭제 불가

emp_ex 테이블에 데이터 등록

사원번호	사원명	급여
1	홍길동	300
2	이순신	400
3	강감찬	500

```
INSERT INTO 테이블명 [(열이름1, 열이름2, ...)]  
VALUES (값1, 값2, ...);
```


Transaction

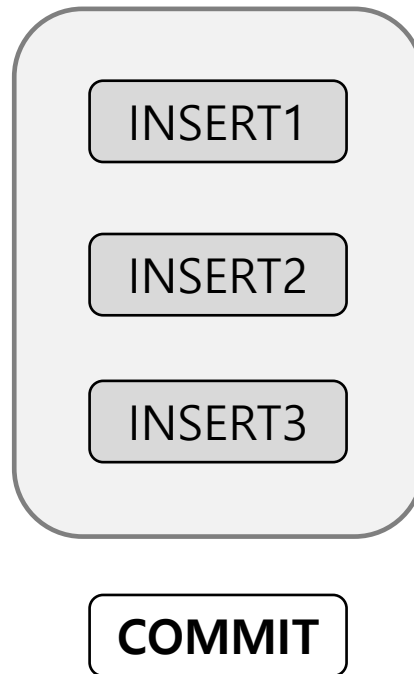
COMMIT

트랜잭션(작업 단위)를 정상적으로 데이터베이스에 적용
작업(INSERT, UPDATE, DELETE) 내용을 DB에 저장

ROLLBACK

작업 중 문제 발생 시 현재 트랜잭션의 변경 내역을 취소하고, 종료
트랜잭션 발생 이전 시점으로 되돌림
작업(INSERT, UPDATE, DELETE) 내용을 취소

하나의 트랜잭션



INSERT1, INSERT2, INSERT3 모두 적용
(COMMIT 하지 않으면 모든 작업적용 안됨)

ROLLBACK

실수로 모든 데이터 삭제

```
DELETE FROM emp_ex;
```

[데이터 확인]

```
SELECT * FROM emp_ex;
```

[ROLLBACK 적용]

```
ROLLBACK;
```

[데이터 재확인]

```
SELECT * FROM emp_ex;
```

지정된 위치 이후 ROLLBACK

```
INSERT INTO emp_ex VALUES (4, '김유신', 600);  
SAVEPOINT sp1;  
INSERT INTO emp_ex VALUES (5, '유관순', 700);  
SAVEPOINT sp2;  
INSERT INTO emp_ex VALUES (6, '안중근', 800);  
ROLLBACK TO sp2;  
COMMIT;
```

[데이터 확인]

```
SELECT * FROM emp_ex;
```

시퀀스

- 테이블 내에서 기본키가 유일한 값을 갖도록 자동으로 지정해주는 객체
- 사용자가 직접 기본키의 값을 생성해 등록하게 되면 중복이 발생할 가능성이 생기며, 직접 로직을 구현해야 한다.
- 기본키(PK)의 중복값을 방지하기 위해 사용

```
CREATE SEQUENCE 시퀀스명  
    START WITH [시작값]  
    INCREMENT BY [증가값]  
    MINVALUE [최소값]  
    MAXVALUE [최대값];
```

시퀀스

- emp_ex 테이블의 empno 컬럼 값을 처리하기 위한 시퀀스 생성

```
CREATE SEQUENCE emp_ex_seq  
  START WITH 1  
  INCREMENT BY 1  
  MINVALUE 1  
  MAXVALUE 99999;
```

시퀀스

CURRVAL

시퀀스의 현재값 확인
CURRENT VALUE의 줄임말

초기 1회 NEXTVAL 실행 후 사용 가능

```
SELECT 시퀀스명.CURRVAL FROM dual;
```

NEXTVAL

시퀀스의 다음 값 확인
NEXT VALUE의 줄임말

```
SELECT 시퀀스명.NEXTVAL FROM dual;
```

시퀀스 실제 사용 사례

- emp_ex 테이블에 새로운 데이터 등록 시 시퀀스 사용
- 직원명 : 유관순, 급여 : 300
- 현재 시퀀스 3으로 지정

```
INSERT INTO emp_ex (empno, ename, salary)
VALUES (emp_ex_seq.nextval, '유관순', 300);
```


- 뷰(View)란 실제로 존재하지 않는 논리적인 가상 테이블

```
CREATE VIEW 뷰명 [(컬럼명...)]
```

```
AS
```

```
SELECT 문;
```

뷰를 사용하는 이유

1. 복잡한 SELECT문을 자주 사용해야 하는 경우
2. 조회 속도를 높이기 위해

뷰

- 직원 중 직급이 사원인 데이터만 뷰 생성

```
GRANT create view TO 사용자계정; -- 뷰 생성 권한 추가
```

```
CREATE VIEW emp_view  
AS  
SELECT * FROM emp WHERE job = '사원';
```

뷰 제거

```
DROP VIEW emp_view;
```

무결성 == 정확성 유지

데이터 무결성 제약조건

테이블에 부적절한 데이터가 입력되는 것을 방지하기 위해,
테이블을 생성할 때, 각 컬럼에 적용하는 규칙

- 테이블 생성 시, 또는 ALTER TABLE로 제약조건 지정 가능

무결성 제약조건 종류

제약조건	설명
NOT NULL	NULL값 비허용
UNIQUE	중복 불가(유일한 값으로 유지)
PRIMARY KEY	NOT NULL + UNIQUE
FOREIGN KEY	참조되는 테이블의 컬럼값
CHECK	데이터 값의 범위나 조건 지정
DEFAULT	기본값 지정

- dept 테이블에 deptno컬럼에 PRIMARY KEY 제약조건 추가

```
ALTER TABLE dept MODIFY deptno number(2) primary key;
```

무결성 제약조건 종류 (PRIMARY KEY)

```
INSERT INTO dept (deptno, dname, loc)
VALUES (50, '영업부', '서울');
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
insert into dept (deptno, dname, loc) values (50, '영업부', '서울')
```

오류 보고 -

ORA-00001: 무결성 제약 조건 (TESTUSER.SYS_C0017776)에 위배됩니다

```
INSERT INTO dept (dname, loc)
VALUES ('영업부', '서울');
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
insert into dept (dname, loc) values ('영업부', '서울')
```

오류 보고 -

ORA-01400: NULL을 ("TESTUSER"."DEPT"."DEPTNO") 안에 삽입할 수 없습니다

사용자 관리

- Oracle DMBS에 접속할 수 있는 사용자 계정 추가

```
CREATE USER 사용자명 IDENTIFIED BY 비밀번호;
```

사용자를 추가할 수 있는 권한 필요

- 시스템 권한을 가지고 있는 SYS로 접속
- 사용자명 : user00, 비밀번호 user1234!@#\$

사용자 권한

- 사용자 생성 후 새로운 접속 시도 시 접속 안됨
- 사용자만 생성되고, 접속 권한이 없기 때문

권한	설명
CREATE USER	사용자 생성 권한
DROP USER	사용자 삭제 권한
CREATE SESSION	데이터베이스 접속 권한
CREATE TABLE	테이블 생성 권한
CREATE VIEW	뷰 생성 권한
CREATE SEQUENCE	시퀀스 생성 권한
CREATE PROCEDURE	프로시저, 함수 생성 권한

사용자 권한 부여

```
GRANT 권한  
TO 사용자명  
[WITH ADMIN OPTION];
```

- SYS로 로그인 후 user00 사용자에게 CREATE SESSION 권한 부여

```
GRANT create session  
TO user00;
```

WITH ADMIN OPTION : 해당 권한을 다른 사용자에게도 부여할 수 있는 권한도 함께 부여

사용자 권한 부여

- CREATE SESSION 권한 부여 후 user00 사용자 접속 가능
- 하지만 데이터 조회 불가 (객체 권한 미부여 상태)

```
ORA-00942: 테이블 또는 뷰가 존재하지 않습니다
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
1행, 24열에서 오류 발생
```

객체별로 DML (SELECT, INSERT, UPDATE, DELETE)을 사용할 수 있는 권한 부여

GRANT 권한
ON 객체명
TO 사용자;

사용자 권한 부여

- user00 사용자에게 dept 테이블 SELECT 권한 부여

```
GRANT select ← 여러개 동시에 지정 가능  
ON 스키마명.dept  
TO user00;
```

- user00 사용자로 접속 후 dept 테이블 조회하려면 스키마 추가

```
SELECT * FROM 스키마명.dept;
```

스키마(Schema)

- 스키마란 객체를 소유한 사용자명
- 객체 앞에 사용자명 기술

```
SELECT * FROM 스키마명.dept;
```

- 지금까지 스키마를 작성하지 않고, 객체명만으로 사용.
- user00 사용자로 접속하면 자신 소유의 객체는 생략 가능하기 때문
- 자신 소유가 아닌 객체를 사용하는 경우 반드시 스키마 지정 필요

사용자 권한 회수

- user00 사용자에게 dept 테이블 SELECT 권한 회수

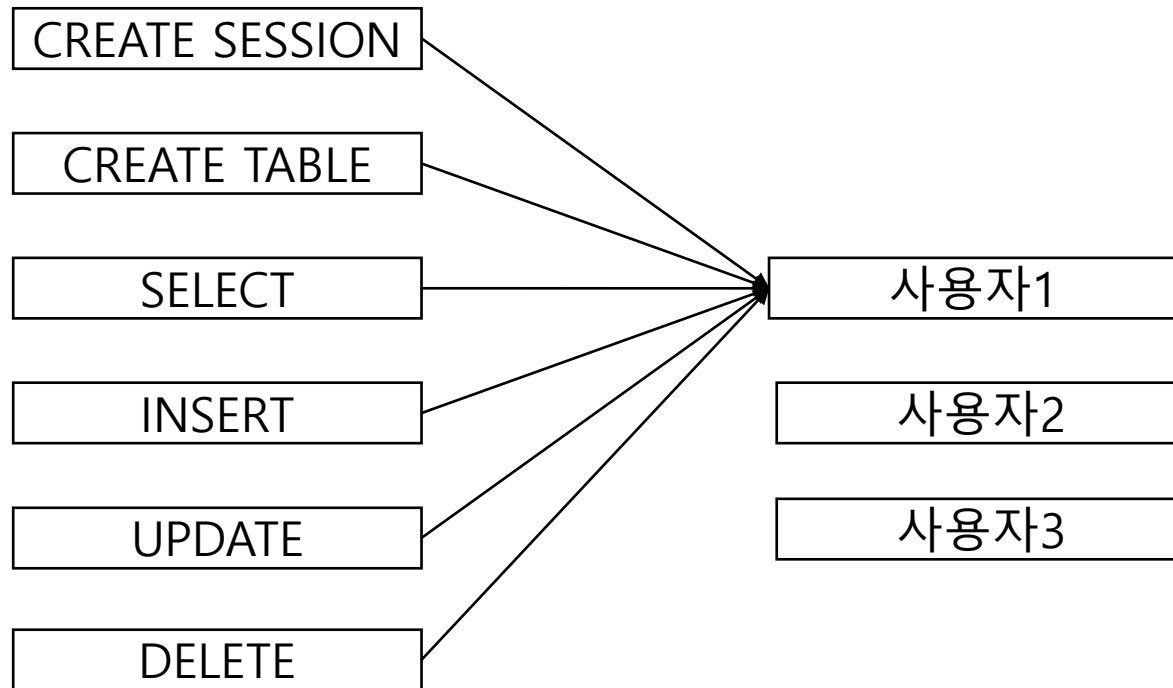
```
REVOKE select  
ON 스키마명.dept  
FROM user00;
```

- user00 사용자로 접속 후 dept 테이블 조회하면 에러

```
ORA-01031: 권한이 불충분합니다  
01031. 00000 - 'insufficient privileges'  
*Cause: An attempt was made to perform a database operation without  
the necessary privileges.  
*Action: Ask your database administrator or designated security  
administrator to grant you the necessary privileges  
1행, 24열에서 오류 발생
```

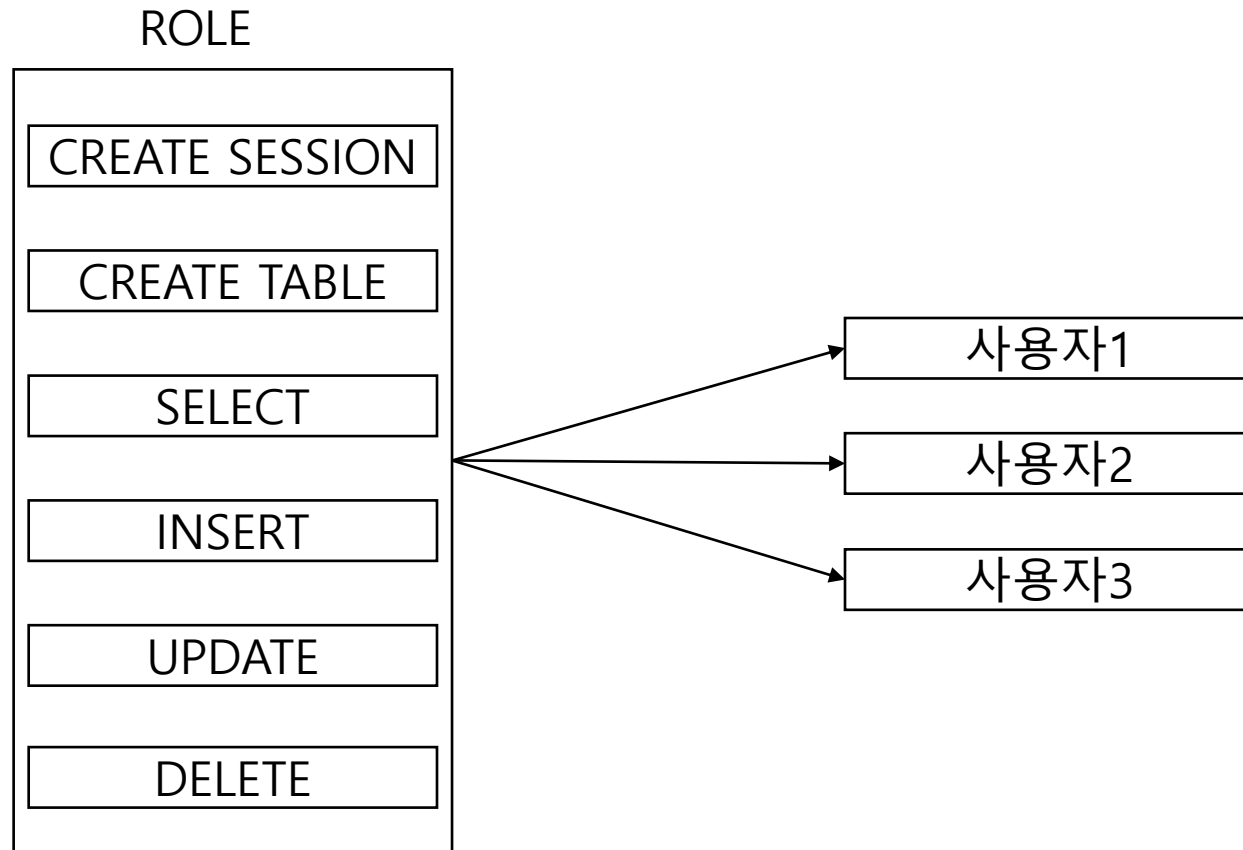
롤(ROLE)

- 롤이란 오라클에서 여러 권한을 묶어 놓은 것
- 개별 권한들을 모든 사용자에게 각각 부여하는 것은 너무 번거로움



롤(ROLE)

- 개별 권한들을 모든 사용자에게 각각 부여하는 것이 아니라, 롤에 하나의 그룹으로 묶어, 사용자에게 해당 롤에 대한 권한을 부여



사전 정의된 롤(ROLE)

CONNECT

DB 접속에 관련된 권한

CREATE SESSION, CREATE TABLE, CREATE VIEW...

RESOURCE

객체(테이블, 뷰, 인덱스)를 생성할 수 있는 권한

CREATE TABLE, CREATE VIEW, CREATE PROCEDURE...

DBA

데이터베이스 객체를 관리하고, 사용자 관리 등 모든 권한

데이터베이스 관리자(Administrator) 권한

사용자 롤 권한 부여/회수

- user00 사용자에게 CONNECT, RESOURCE 롤 권한 부여

```
GRANT connect, resource TO user00;
```

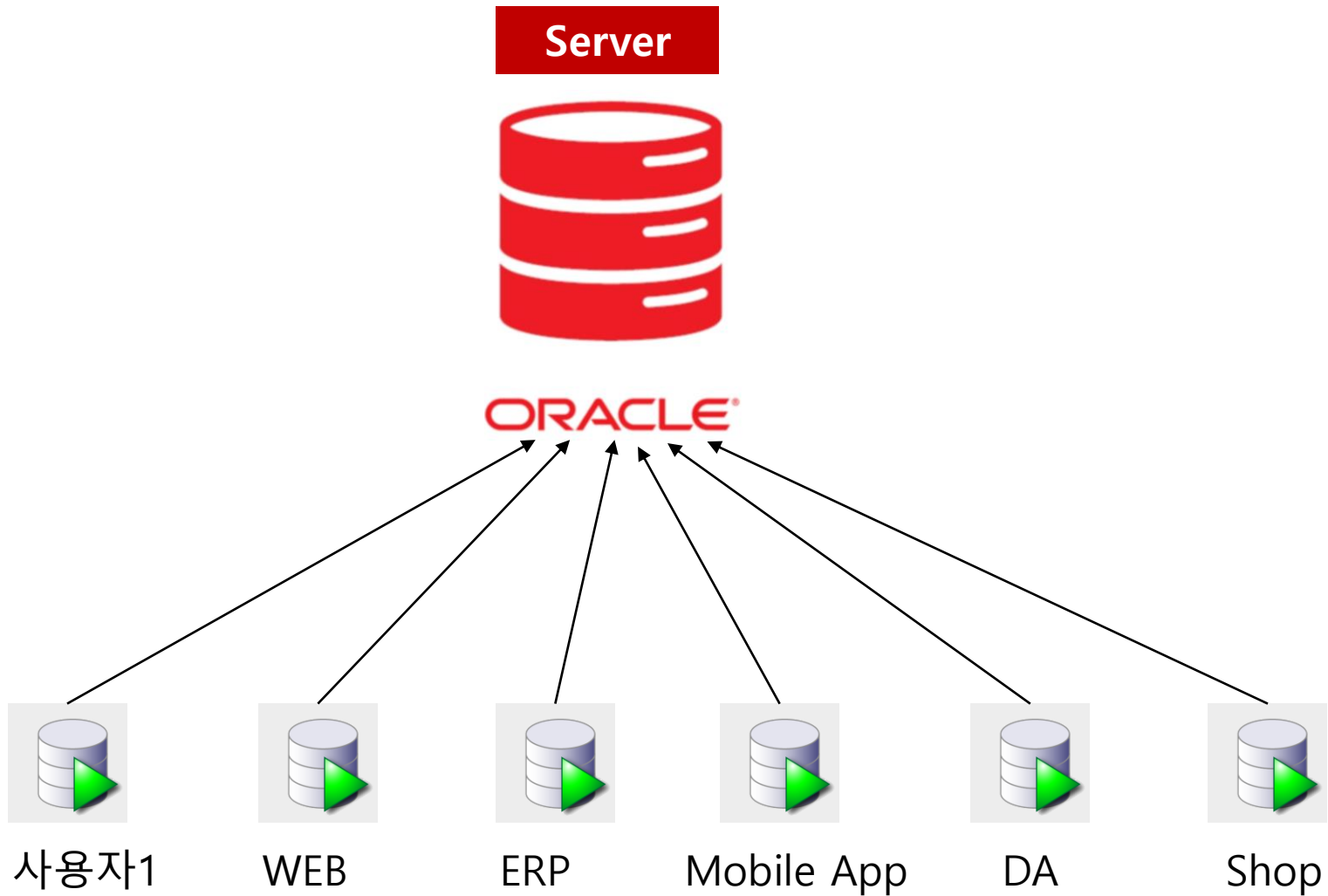
- user00 부여 받은 롤 권한 확인

```
SELECT * FROM user_role_privs
```

- user00 부여 받은 롤 권한 회수

```
REVOKE connect, resource FROM user00;
```


운영환경



- Procedural Language / SQL
- 비절차적 언어인 SQL을 확장한 절차적인 프로그래밍 언어
- 오라클에서만 사용 가능

- 변수, 상수 등을 이용하여 SQL과 절차형 언어 사용
- IF 문을 사용하여 조건에 따른 문장 분기 실행
- LOOP문을 사용하여 반복 실행
- 커서를 사용하여 여러 행을 검색

PL/SQL

- PL/SQL은 세 블록으로 구분

블록	설명
DECLARE (선언부)	변수나 상수 선언
BEGIN (실행부)	SQL, 절차적 언어 (제어문, 반복문, 함수 등)를 이용한 로직
EXCEPTION (예외처리부)	예외가 발생할 가능성이 있는 코드 처리

DECLARE

변수, 상수 선언;

BEGIN

명령문;

필수

EXCEPTION

예외처리;

END;

- hello world 출력

```
SET SERVEROUTPUT ON;  -- 실행결과를 화면 출력 설정
BEGIN
    DBMS_OUTPUT.PUT_LINE('hello world');
END;
```

```
hello world
```

```
PL/SQL 프로시저가 성공적으로 완료되었습니다.
```

변수 선언

DECLARE

변수명 타입 [:= 값/표현식];

- 사원번호, 사원명 변수 선언

DECLARE

VEMPNO NUMBER(4);

VENAME VARCHAR2(10);

BEGIN

VEMPNO := 1;

VENAME := '홍길동';

DBMS_OUTPUT.PUT_LINE('사원번호 : ' || VEMPNO);

DBMS_OUTPUT.PUT_LINE('사원명 : ' || VENAME);

END;

변수 선언

DECLARE

변수명 **CONSTANT** 타입 := 값/표현식;

- 상수는 한번 저장한 값 변경 불가(에러 발생)

DECLARE

VEMPNO CONSTANT NUMBER(4) := 100;

BEGIN

VEMPNO := 1;

DBMS_OUTPUT.PUT_LINE('사원번호 : ' || VEMPNO);

END;

SELECT 문

- SELECT문을 사용해 데이터를 추출해서 변수에 대입
- 이름이 '손흥민'인 사원의 사원번호와 직급 출력

```
DECLARE
```

```
    VEMPNO EMP.EMPNO%TYPE;
```

```
    VJOB EMP.JOB%TYPE;
```

```
BEGIN
```

```
    SELECT EMPNO, JOB INTO VEMPNO, VJOB
```

```
    FROM EMP WHERE ENAME='손흥민';
```

```
    DBMS_OUTPUT.PUT_LINE('사원번호 : ' || VEMPNO);
```

```
    DBMS_OUTPUT.PUT_LINE('직급 : ' || VJOB);
```

```
END;
```

조건문

종류	설명
IF ~ THEN ~ END IF	조건을 만족하는 경우 실행
IF ~ THEN ~ ELSE ~ END IF	조건이 만족하는 경우와 그렇지 않은 경우 구분해서 실행
IF ~ THEN ~ ELSIF ~ ELSE ~ END IF	여러 조건에 따라 지정해서 실행

IF ~ THEN ~ END IF

```
IF 조건 THEN  
    실행문;  
END IF;
```

```
DECLARE
```

```
    VNUMBER NUMBER := 10;
```

```
BEGIN
```

```
    IF MOD(VNUMBER, 2) = 0 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('짝수');
```

```
    END IF;
```

```
END;
```


조건문

IF ~ THEN ~ ELSE ~ END IF

```
IF 조건 THEN
    실행문1;
ELSE
    실행문2;
END IF;
```

```
DECLARE
    VNUMBER NUMBER := 11;
BEGIN
    IF MOD(VNUMBER, 2) = 0 THEN
        DBMS_OUTPUT.PUT_LINE('짝수');
    ELSE
        DBMS_OUTPUT.PUT_LINE('홀수');
    END IF;
END;
```

조건문

IF ~ THEN ~ ELSIF ~ ELSE ~ END IF

```
IF 조건1 THEN
    실행문1;
ELSIF 조건2 THEN
    실행문2;
ELSE
    실행문3;
END IF;
```

```
DECLARE
    VSCORE NUMBER := 70;
BEGIN
    IF VSCORE >= 90 THEN
        DBMS_OUTPUT.PUT_LINE('A');
    ELSIF VSCORE >= 80 THEN
        DBMS_OUTPUT.PUT_LINE('B');
    ELSE
        DBMS_OUTPUT.PUT_LINE('C');
    END IF;
END;
```

반복문

종류	설명
LOOP	기본 반복문
FOR LOOP	반복횟수를 정하여 반복
WHILE LOOP	조건식의 결과에 의해 반복

LOOP

```
LOOP
    실행문;
END LOOP;
```

```
DECLARE
    VNUMBER NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE(VNUMBER);
        VNUMBER := VNUMBER+1;
        EXIT WHEN VNUMBER > 10;
    END LOOP;
END;
```

조건에 해당하면
반복종지

FOR LOOP

```
FOR i IN 시작값..종료값 LOOP  
    실행문;  
END LOOP;
```

```
BEGIN  
    FOR i IN 1..10 LOOP  
        DBMS_OUTPUT.PUT_LINE(i);  
    END LOOP;  
END;
```

WHILE LOOP

```
WHILE 조건식 LOOP  
    실행문;  
END LOOP;
```

```
DECLARE  
    VNUMBER NUMBER := 1;  
BEGIN  
    WHILE VNUMBER <= 10 LOOP  
        DBMS_OUTPUT.PUT_LINE(VNUMBER);  
        VNUMBER := VNUMBER+1;  
    END LOOP;  
END;
```

커서

SELECT문의 실행결과가 여러개인 경우 행별로 작업을 수행 가능
사원테이블의 사원번호, 사원명 출력

```
DECLARE
    CURSOR cur IS
        SELECT empno, ename FROM emp;
BEGIN
    FOR r IN cur LOOP
        DBMS_OUTPUT.PUT_LINE(r.EMPNO || r.ENAME);
    END LOOP;
END;
```

PL/SQL 실습

- PL/SQL문으로 직원테이블에서 급여가 600이하인 직원 조회
- 조회한 데이터의 사원번호, 사원명, 급여(10%인상)를 emp_ex 테이블에 등록

```
DECLARE
```

```
    CURSOR cur IS
```

```
        [SQL문]
```

```
BEGIN
```

```
    FOR r IN cur LOOP
```

```
        [INSERT 문]
```

```
    END LOOP;
```

```
    COMMIT;
```

```
END;
```

프로시저

프로시저 (저장 프로시저, stored procedure)

- 특정 작업 처리를 위해 만들어 놓은 PL/SQL을 미리 저장해 두고, 필요할때 호출해서 사용
- SQL문만으로 처리하는 것보다 성능 향상

```
CREATE [OR REPLACE] PROCEDURE 프로시저명[(파라미터)]  
IS  
    선언부  
BEGIN  
    실행부;  
END;
```


프로시저 생성

emp 테이블을 이용해 emp_test 생성

```
CREATE TABLE emp_test AS SELECT * FROM emp;
```

emp_test 테이블의 전체 데이터를 삭제하는 프로시저 생성

```
CREATE OR REPLACE PROCEDURE deleteAll  
IS  
BEGIN  
    DELETE FROM emp_test;  
    COMMIT;  
END;
```

프로시저 실행

프로시저를 생성만 한 상태에서는 아직 실행되기 전이며,
EXECUTE 명령어로 실행

deleteAll 프로시저 실행

```
EXECUTE deleteAll;
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.

프로시저(파라미터) 생성

emp 테이블을 이용해 emp_test 생성

```
DROP TABLE emp_test;  
CREATE TABLE emp_test AS SELECT * FROM emp;
```

파라미터가 해당 직급인 데이터를 삭제하는 프로시저 생성

```
CREATE OR REPLACE PROCEDURE deleteJob(vjob varchar2)  
IS  
BEGIN  
    DELETE FROM emp_test WHERE JOB = vjob;  
    COMMIT;  
END;
```

프로시저 파라미터 모드

모드	설명
IN	값 입력 (기본값)
OUT	값 반환
IN OUT	값입력 후 결과값 반환

사원번호를 입력 받아, 이름, 직급 반환하는 프로시저

```
CREATE OR REPLACE PROCEDURE selectEmp
  (vempno in emp.empno%type,
   vename out emp.ename%type, vjob out emp.job%type)
IS BEGIN
  SELECT ename, job INTO vename, vjob
  FROM emp WHERE empno = vempno;
END;
```

프로시저 파라미터 모드

-- 변수 선언

variable vename varchar2;

variable vjob varchar2;

-- 프로시저 실행

execute selectEmp(1000, :vename, :vjob);

-- 변수 출력

print vename;

print vjob;

함수

프로시저와 유사한 용도로 사용되며, 프로시저와 다르게 실행 결과를 리턴할 수 있음

```
CREATE [OR REPLACE] FUNCTION 함수명[(파라미터)]  
RETURN 리턴타입  
IS  
    선언부  
BEGIN  
    실행부;  
    RETURN 리턴값;  
END;
```

함수

사원번호를 입력받아 해당 사원의 연봉을 리턴하는 함수 생성

```
CREATE OR REPLACE FUNCTION calSal (vempno in number)
RETURN number
IS
    vsal number;
    vbonus number;
BEGIN
    SELECT salary, bonus INTO vsal, vbonus
    FROM emp WHERE empno=vempno;
    RETURN vsal*12+nvl(vbonus,0);
END;
```

함수 실행

```
variable vsalary varchar2;  
execute :vsalary := calSal(1000);  
  
print vsalary;
```

VSALARY

22100

프로시저와 함수

구분	프로시저	함수
실행	execute 명령어, 다른 PL/SQL 내	execute 명령어, 다른 PL/SQL 내, SQL문에서 직접 실행
파라미터 모드	있을 수도, 없을 수도 있음 IN, OUT, IN OUT	있을 수도, 없을 수도 있음 IN
값 반환	없을 수도 있고, OUT 모드의 개 수에 따라 여러개 반환 가능	RETURN 절을 이용해 하나의 값만 반환 가능

트리거

- 특정 조건을 만족하거나, 어떤 동작이 실행되면, 자동으로 수행되는 객체
- 프로시저, 함수는 사용자가 직접 호출하지만, 트리거는 자동으로 실행
- 사용자가 직접 실행 불가
- SQL문의 복잡도 감소

```
CREATE OR REPLACE TRIGGER 트리거명
```

```
BEFORE | AFTER
```

```
INSERT | UPDATE | DELETE
```

```
ON 테이블명
```

```
BEGIN
```

```
    실행부;
```

```
END;
```

트리거

- 사원 테이블(emp2)에 새로운 데이터가 등록되면 '신입사원 입사' 출력

```
CREATE TABLE emp2 AS SELECT * FROM emp;
```

```
CREATE OR REPLACE TRIGGER emp_trigger
```

```
before insert
```

```
ON emp2
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('신입사원 입사');
```

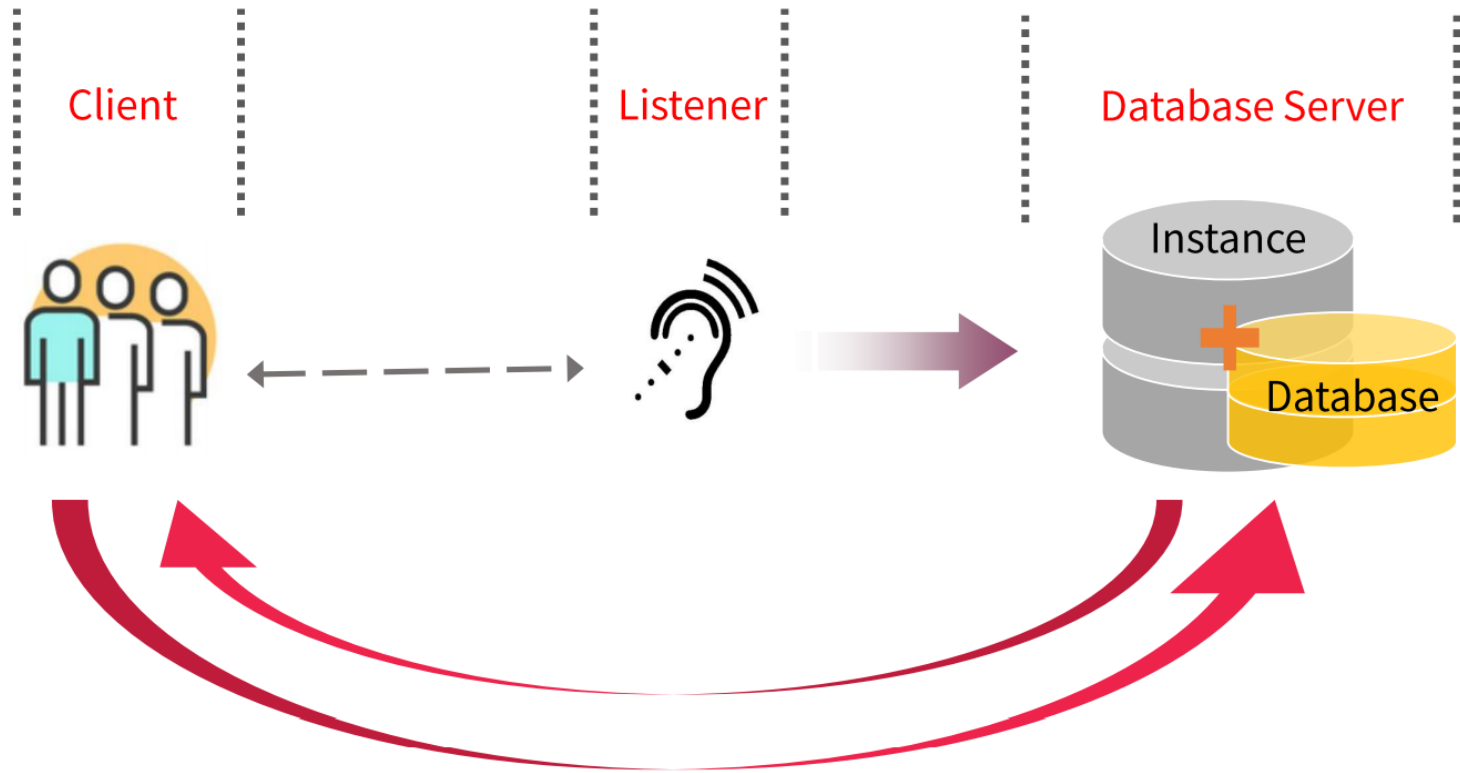
```
END;
```

```
INSERT INTO emp2 VALUES
```

```
(2013, '홍길동', '사원', 2002, '2000-01-01', SYSDATE, 300, 0, 40);
```

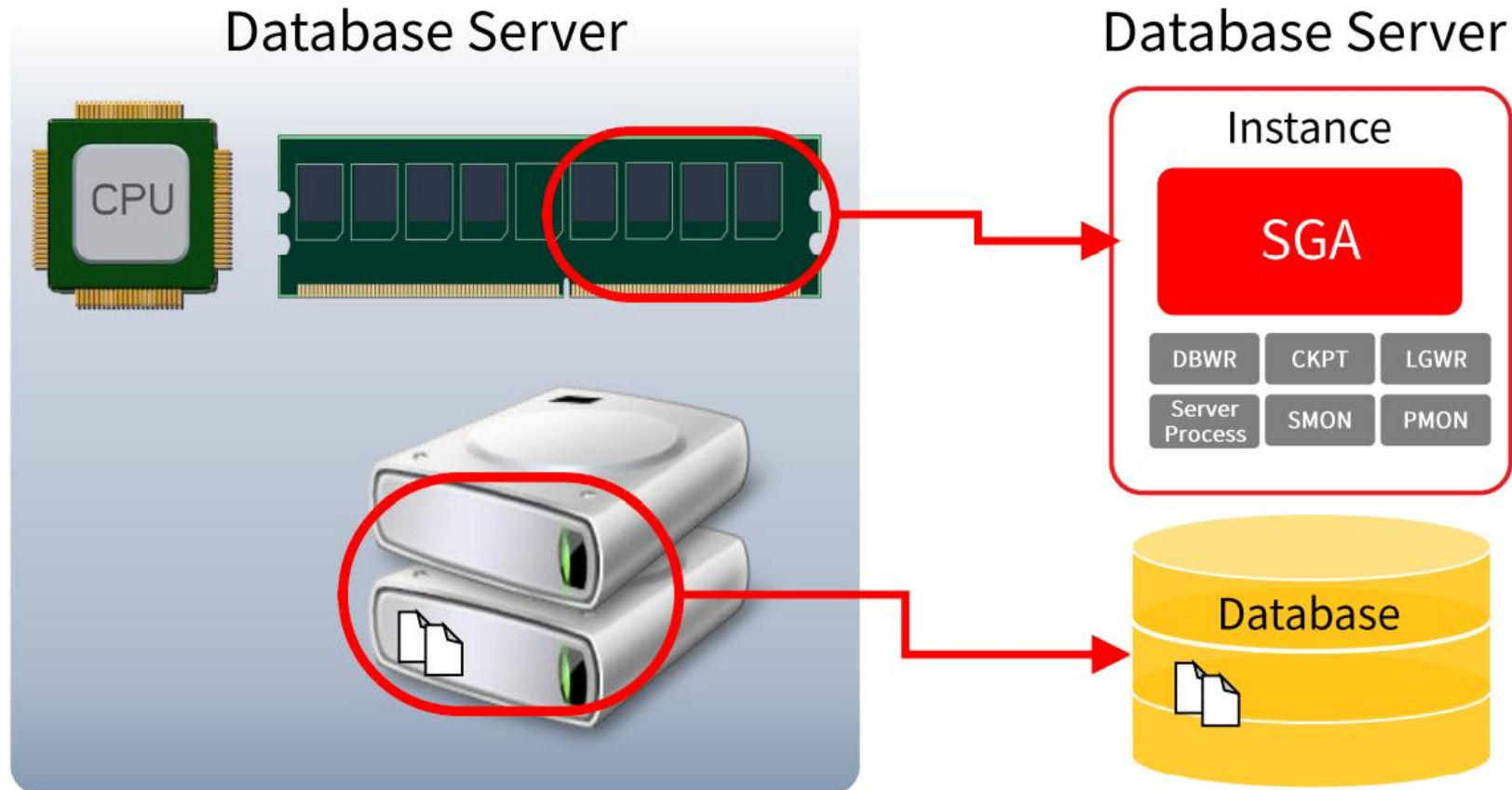
오라클 아키텍처

Oracle Server 접근 방식



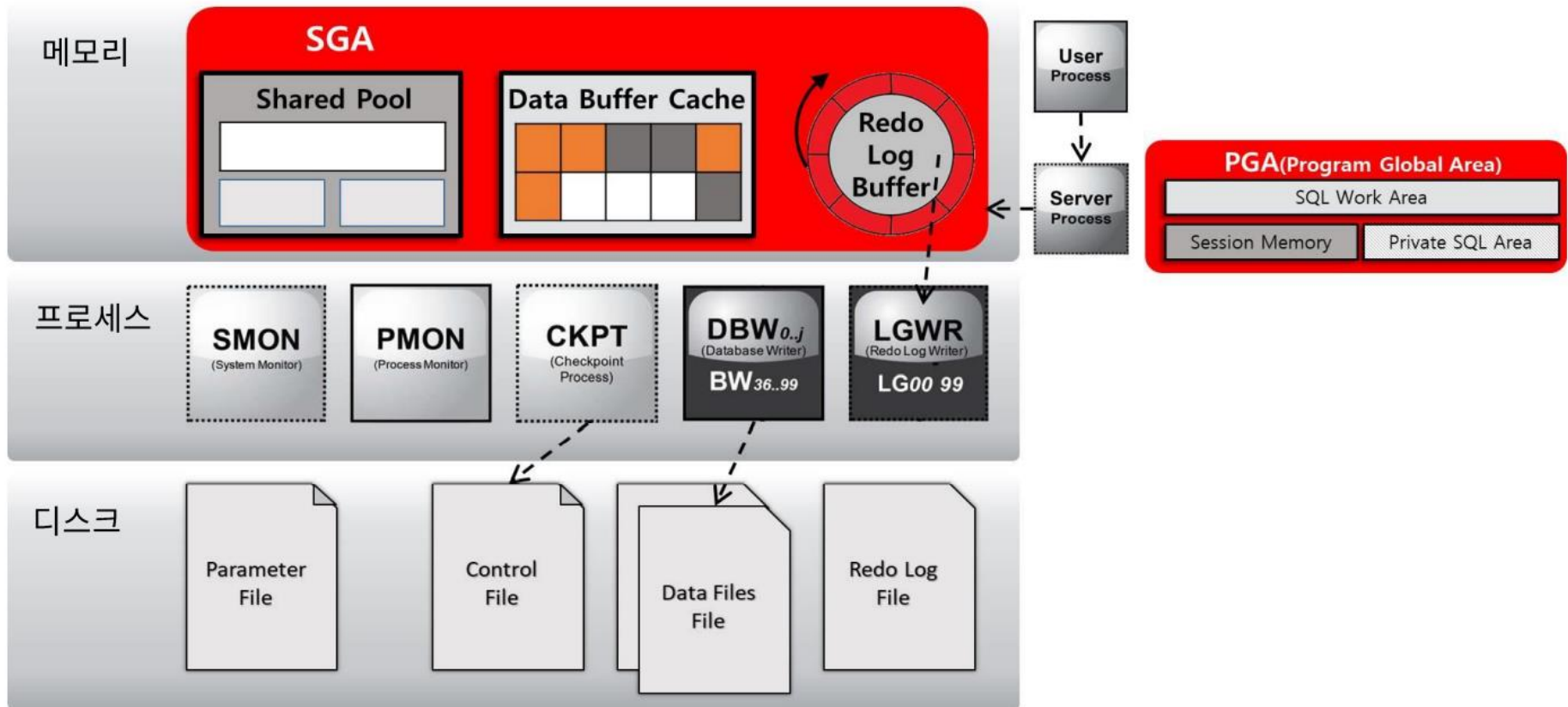
오라클 아키텍처

Oracle Server 접근 방식



오라클 아키텍처

Instance, Database



오라클 아키텍처 (메모리 구조)

SGA (System Global Area)

- 인스턴스 시작 시 SGA 영역 할당
- 인스턴스별로 하나만 생성
- 모든 사용자가 공유하는 영역

PGA (Program Global Area)

- 사용자 요청 데이터 정보가 처리되는 영역
- 사용자 세션별로 독립적으로 생성

SQL 실행 순서

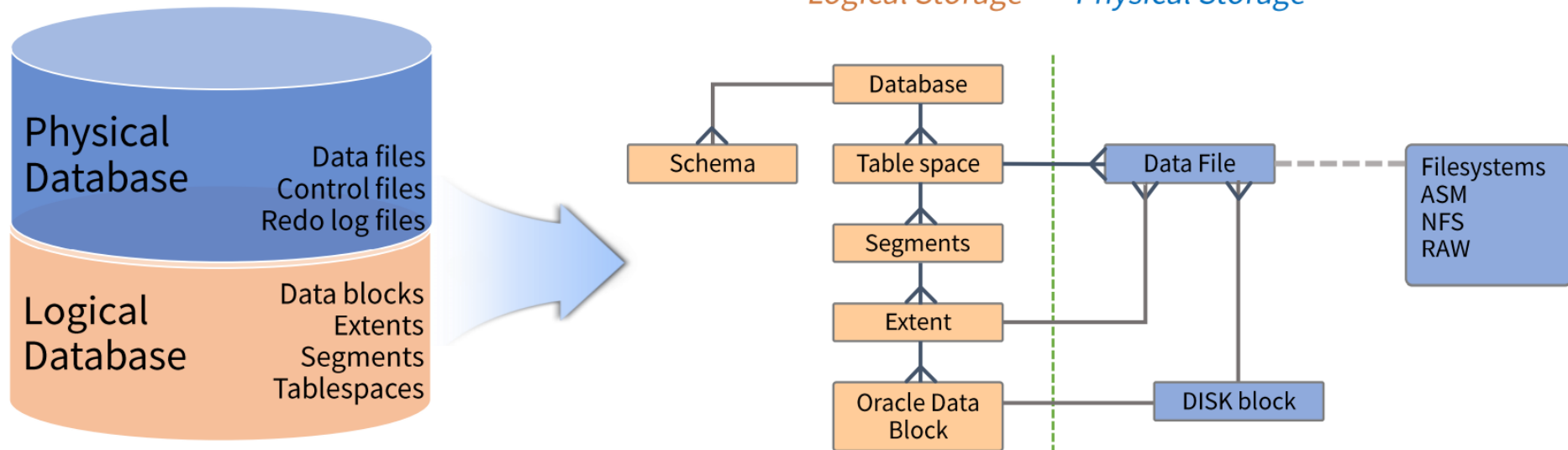
INSERT, UPDATE, DELETE

1. 문법 확인
2. 의미 검사
3. LGWR이 Redo Log Buffer에 저장
4. DBWR이 버퍼캐시에 기록 -> 데이터 파일에 저장

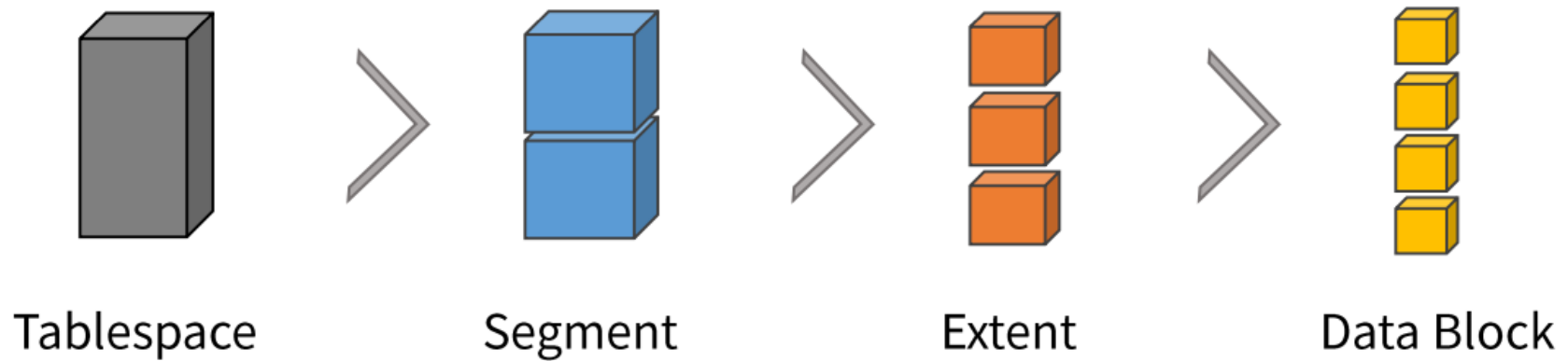
SELECT

1. 문법 확인
2. 의미 검사
3. DB 버퍼 캐시에서 확인 (없으면 디스크에서 찾아 버퍼 캐시로 복사)
4. 조회결과 데이터를 사용자 프로세스로 전송

데이터 저장소 구조



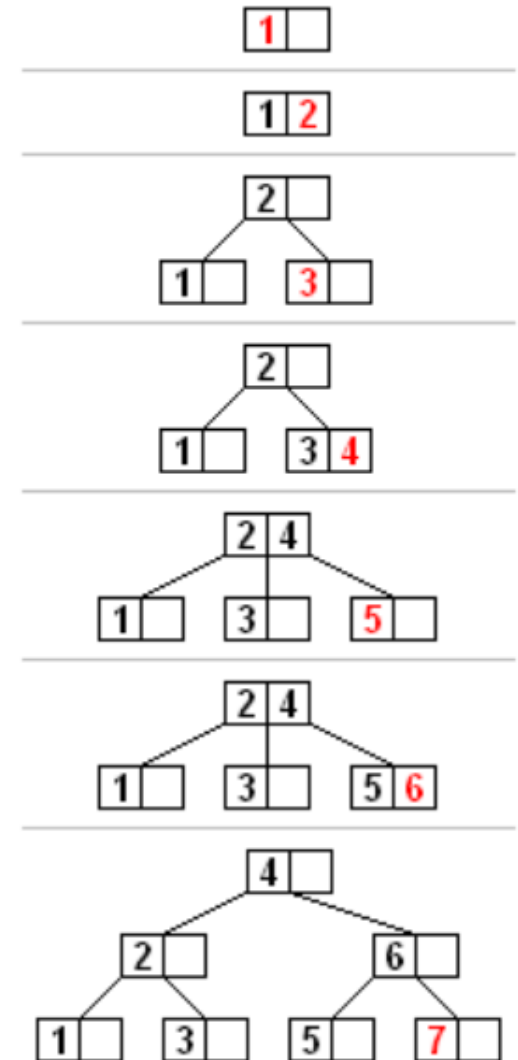
논리적 저장소



단위	설명
Tablespace	테이블이 모여 있는 단위로, 하나 이상의 Segment를 포함하는 저장 단위
Segment	테이블, 인덱스 등 모든 데이터를 보유하고 있는 Extent의 집합
Extent	Segment에 할당된 연속적인 Data Block의 집합
Data Block	데이터 파일에 할당되는 최소의 구조 단위

인덱스

- 데이터의 빠른 검색을 위해 사용하는 색인 기술
- B-tree 자료구조를 이용하여 검색 속도 향상
- B-tree(Balanced Tree)란 이진트리의 변형된 알고리즘으로, 데이터들을 빠르게 찾을 수 있도록 트리구조에 정렬한 상태로 보관하는 방식



인덱스

인덱스 장/단점

장점	단점
<ul style="list-style-type: none">• 검색 속도가 빠름• 시스템 부하를 줄여 성능 향상	<ul style="list-style-type: none">• 추가 공간 필요• 인덱스 생성 시간 소요• INSERT/UPDATE/DELETE 작업 시 성능 저하

인덱스 생성

```
CREATE INDEX 인덱스명 ON 테이블명 (컬럼);
```

인덱스 실습

인덱스 실행 전 실행 계획 출력

```
EXPLAIN PLAN FOR SELECT * FROM emp WHERE ename='손흥민';  
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

1	Plan hash value: 3956160932							
2								
3	-----							
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
5	-----							
6	0	SELECT STATEMENT		1	50	3 (0)	00:00:01	
7	* 1	TABLE ACCESS FULL	EMP	1	50	3 (0)	00:00:01	
8	-----							
9								
10	Predicate Information (identified by operation id):							
11	-----							
12								
13	1 - filter("ENAME"='손흥민')							

인덱스 실습

인덱스 생성

emp 테이블의 직원명(ename) 컬럼에 인덱스 생성

```
CREATE INDEX empindex ON emp (ename);
```

인덱스 실습

실행 계획 출력

```
EXPLAIN PLAN FOR SELECT * FROM emp WHERE ename='손흥민';  
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

PLAN_TABLE_OUTPUT									
1	Plan hash value: 2181900149								
2									
3	-----								
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time		
5	-----								
6	0	SELECT STATEMENT		1	96	1 (0)	00:00:01		
7	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP3	1	96	1 (0)	00:00:01		
8	* 2	INDEX RANGE SCAN	EMPINDEX	1		1 (0)	00:00:01		
9	-----								
10									
11	Predicate Information (identified by operation id):								
12	-----								
13									
14	2 - access("ENAME"='손흥민')								
15									
16	Note								
17	-----								
18	- dynamic statistics used: dynamic sampling (level=2)								

인덱스를 사용해야할 컬럼은?

WHERE절 조건 에서 자주 사용되는 컬럼

조인 시 조건에 사용되는 컬럼

두 개 이상의 컬럼이 포함되는 조건인 경우 복합 인덱스로 지정

복합 인덱스인 경우 조회 조건에 모든 컬럼이 동시에 사용되도록 지정

주의 사항

PRIMARY KEY, UNIQUE KEY 제약조건에서는 자동 생성

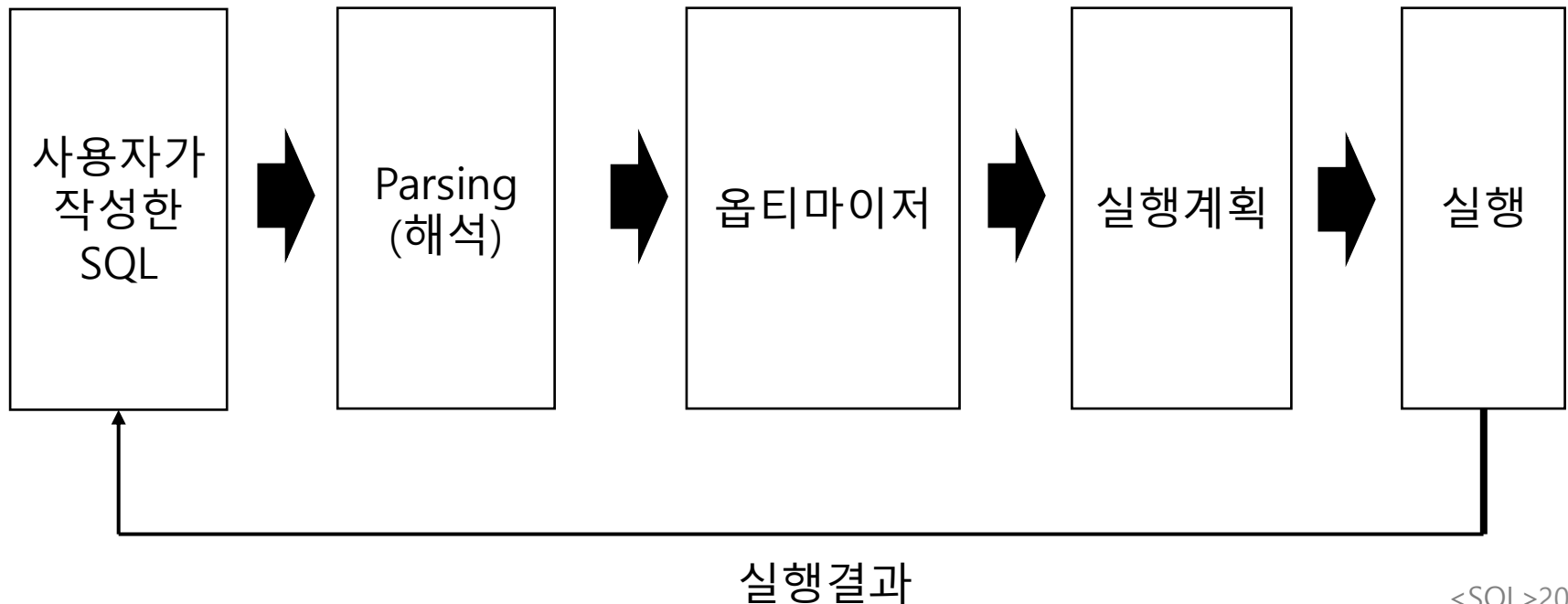
해당 컬럼을 가공하기 전 상태에서 조건 지정

인덱스 여부까지 데이터 모델링 시 고려 필요

LIKE 연산은 인덱스 미적용 가능성 높음 (% 앞뒤로 포함된 경우)

옵티마이저란?

- 사용자가 질의한 SQL문에 대한 최적의 실행 방법을 결정하는 역할
- 옵티마이저가 선택한 실행 방법에 따라 수행 속도 영향



규칙기반 옵티마이저 (Rule Base Optimizer)

규칙(우선순위)를 통한 실행계획 생성

인덱스 유무와 종류

연산자의 종류

우선순위가 높은 규칙으로 실행계획 생성

순위	액세스 기법	설명
1	Single Row By Rowid	단일 로우
4	Single Row By Unique or Primary Key	유니크/PK 단일 로우
8	Composite Index	복합 인덱스
9	Single-Column Index	단일 컬럼 인덱스
10	Bounded Range Search on Indexed Columns	제한된 범위 검색
11	Unbounded Range Search on Indexed Columns	비제한 범위 검색
15	Full Table Scan	전체 테이블 스캔

비용기반 옵티마이저 (Cost Base Optimizer)

규칙기반 옵티마이저의 미측정 사례 단점을 극복하기 위해 사용
SQL문을 처리하는데 필요한 비용이 가장 적은 실행계획 선택
테이블, 인덱스, 컬럼의 통계정보와 시스템 통계정보를 사용
Oracle7 버전 이후 사용 가능

최신 버전에서는 규칙기반 보다는 비용기반 옵티마이저를 기본 사용

```
CREATE INDEX deptno_emp_index ON emp(deptno);
```

```
CREATE INDEX emp_emp_index ON emp(empno);
```

```
SELECT ename FROM emp
```

```
WHERE deptno = 20
```

```
AND empno BETWEEN 2001 AND 2005
```

```
ORDER BY ename ASC
```

실행계획

SQL에서 질의한 내용을 처리하기 위한 순서와 절차
어떤 순서로 어떻게 처리할지 결정하는 방법

구성요소 : 조인순서, 조인기법, 액세스기법, 최적화 정보, 연산

```
SELECT ename FROM emp JOIN dept  
ON emp.deptno = dept.deptno
```

조인순서 : emp->dept

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				3
NESTED LOOPS				3
TABLE ACCESS	EMP	FULL		3
INDEX	SYS_C0017776	UNIQUE SCAN		0
Access Predicates				
EMP.DEPTNO=DEPT.DEPTNO				

액세스기법

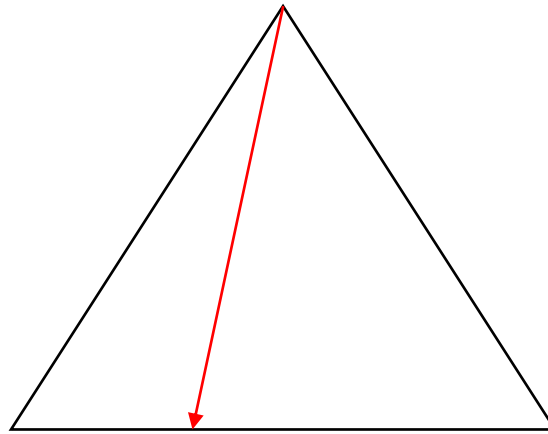
최적화 정보

인덱스를 구성하는 컬럼의 값을 기반으로 데이터를 추출하는 기법

인덱스 스캔 종류

- Index Unique Scan
- Index Range Scan [Descending]
- Index Full Scan
- Index Skip Scan
- Index Fast Full Scan

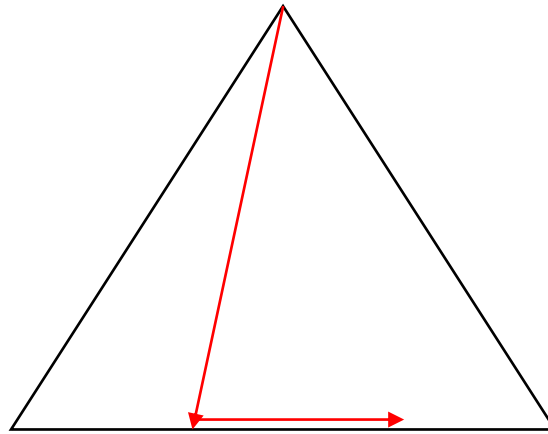
Index Unique Scan



- unique 인덱스를 사용하여 하나의 데이터를 추출
- unique 인덱스는 중복 불가
- '=' 연산자 사용

```
SELECT * FROM dept WHERE deptno=20
```

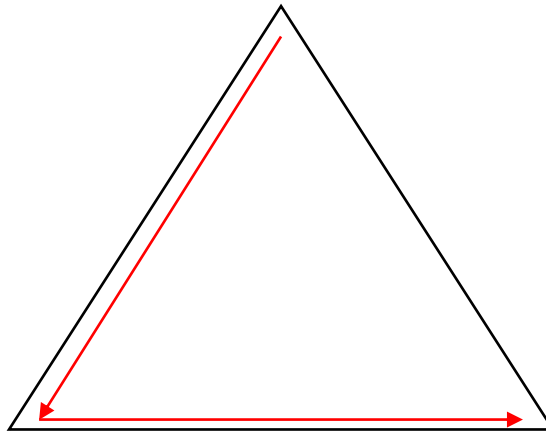
Index Range Scan



- 인덱스를 이용하여 한 건 이상의 데이터 추출

```
SELECT * FROM emp WHERE deptno=20
```

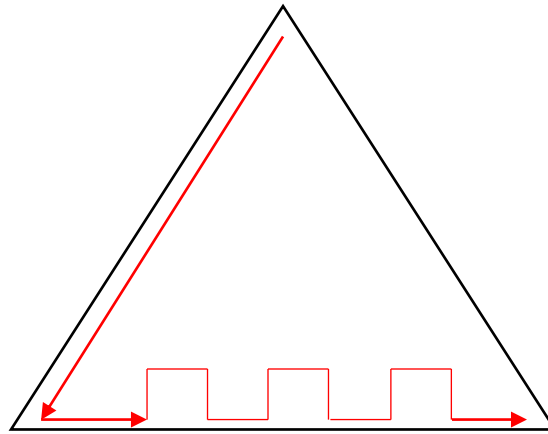
Index Full Scan



- 처음 블록부터 마지막 블록까지 순차적으로 탐색

```
SELECT * FROM EMP WHERE deptno IS NOT NULL
```


Index Range Scan



- 복합 인덱스로 구성된 경우 WHERE 조건절에 생략된 경우(속도저하 위험)

```
CREATE INDEX ENAME_MGR_INDEX ON emp(ename,mgr);
```

```
SELECT * FROM emp WHERE ename='손흥민' and mgr = 1000;
```

인덱스 스캔 유도

실행 계획 생성 시 통계정보를 이용하는데, 잘못된 실행계획을 만들 수 있음

원하는 실행계획을 유도하기 위해 힌트를 사용

```
SELECT /*+ Hint명(테이블명 인덱스명) */ ...
```

힌트의 종류

- INDEX : 인덱스 스캔 유도 (스캔 방식은 오라클에서 선택)
- INDEX_RS : Index Range Scan 유도 (불가능한 경우 무시)
- INDEX_SS : Index Skip Scan 유도 (불가능한 경우 무시)
- INDEX_FFS : Index Fast Full Scan

옵티마이저 조인

Nested Loop

Full Access가 발생하므로, 건수가 작은 테이블부터 스캔하도록 조절 필요 (Random Access가 많이 발생함)

```
SELECT /*+ ordered use_nl(d) */  
    e.deptno, d.dname  
FROM emp e, dept d  
WHERE e.deptno = d.deptno;
```

Sort Merge

메모리의 SORT_AREA 영역에 로딩 후 정렬하므로 데이터가 많아질수록
성능 저하

```
SELECT /*+ ordered use_merge(d) */  
    e.deptno, d.dname  
FROM emp e, dept d  
WHERE e.deptno = d.deptno;
```

옵티마이저 조인

Hash

조인 키를 사용해서 해시 테이블 생성(높은 메모리 사용량)

해시 함수 연산시 CPU 연산

```
SELECT /*+ ordered use_hash(d) */  
    e.deptno, d.dname  
FROM emp e, dept d  
WHERE e.deptno = d.deptno;
```

인덱스 컬럼 가공 금지

```
WHERE SUBSTR(ename, 1, 1) = '손'
```



```
WHERE ename LIKE '손%'
```

복합인덱스와 개별 인덱스

```
WHERE empno = 200 AND deptno = 20
```

인덱스 (empno, deptno)



인덱스 (empno)

인덱스 (deptno)

PK (Primary Key) 활용

- PK 인덱스를 기본적으로 추가
- PK만 지정하더라도 기본적인 인덱스 스캔 가능
- 복합 인덱스 생성 시 PK 컬럼을 앞쪽에 지정

```
CREATE INDEX EMP_INDEX ON emp (empno, deptno);
```


인덱스 스캔범위 비교

- WHERE 절의 범위 지정 순서에 따라 인덱스 스캔 범위 차이
- 예) 주문내역 데이터 중
- 주문일자 : 2020년 1월 1일 ~ 2020년 1월 30일 -> 10,000건
- 상품번호 : A0001 -> 100건

WHERE 주문일자 BETWEEN '2020-01-01' AND '2020-01-30'
AND PRODUCT_NO = 'A0001'



WHERE PRODUCT_NO = 'A0001'
AND 주문일자 BETWEEN '2020-01-01' AND '2020-01-30'

인덱스 설계 전략

- '=' 조건 사용되며, 자주 사용되는 컬럼 앞쪽에 지정
- '=' 조건 사용되며, 값의 개수가 적은 컬럼 앞쪽에 지정
- 자주 사용되는 컬럼으로 생성
- 테이블 랜덤 액세스 줄이자.
- 정렬하는 컬럼에도 인덱스 지정



수고하셨습니다!

감사합니다.