

데이터베이스개론

2023년 2학기 실습자료

ex_db

실습 1

제 3 장 오라클 소개

- 오라클 소개
- 오라클 설치 방법
- 오라클 구조

오라클(Oracle)의 역사

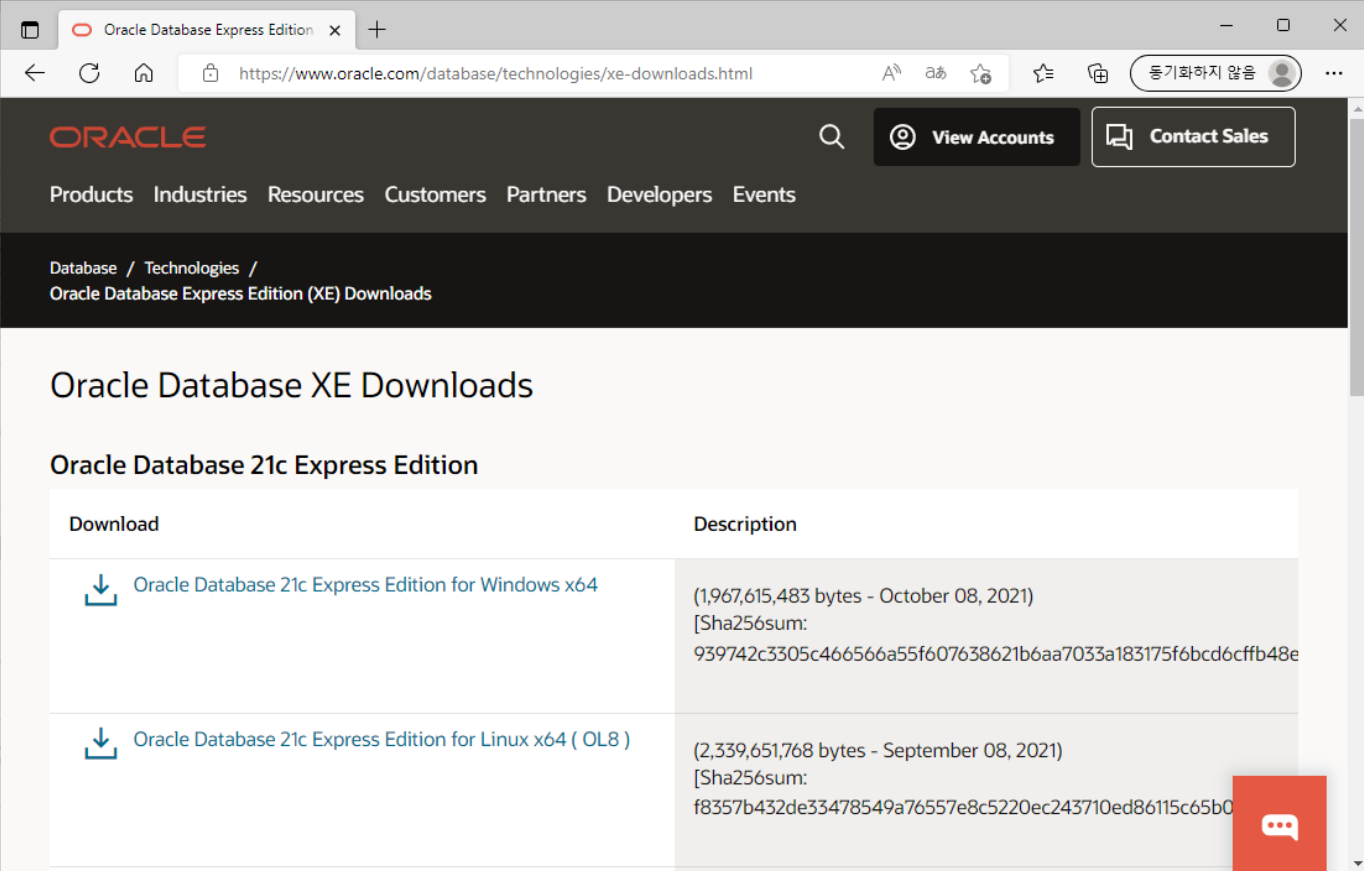
- 1978년
 - 로렌스 J. 엘리슨(현 회장)이 관계형 DBMS인 오라클 첫 번째 버전(Version 1)을 개발
- 1979년
 - 회사명을 RSI(Relational Software Inc.)로 바꾸고 첫 번째 상용 DBMS인 오라클 두 번째 버전(Version 2)을 개발
- 1983년
 - 회사 이름을 지금의 오라클로 바꾸고 C언어로 개발된 오라클 세 번째 버전(Version 3)을 출시
- 1999년
 - 오라클 8i 출시 (i는 인터넷의 약자)
- 2003년
 - 오라클 10g 출시 (g는 그리드 컴퓨팅의 약자)
- 현재
 - 오라클 11g가 최신 버전**

오라클 DBMS



- 클라이언트/서버 및 분산 처리 환경 지원
- 다양한 운영체제 지원
 - MS Windows, Unix(Solaris, HP-UX 등), Linux
- 대용량 데이터 처리 지원
 - Petabyte 크기의 데이터 저장 관리
- 많은 사용자의 동시 접속 지원
- 신뢰성 높은 보안 기능 제공
 - 인증, 권한 관리, 암호화 등
- 오류 및 장애에 대한 대비책 지원
 - backup, recovery 기능
- 다양한 데이터 및 응용 분야 지원
 - Multimedia objects, XML
 - Data Warehouse/OLAP, Mobile, Grid Computing, Cloud Computing

Oracle Express Edition

- 다운로드



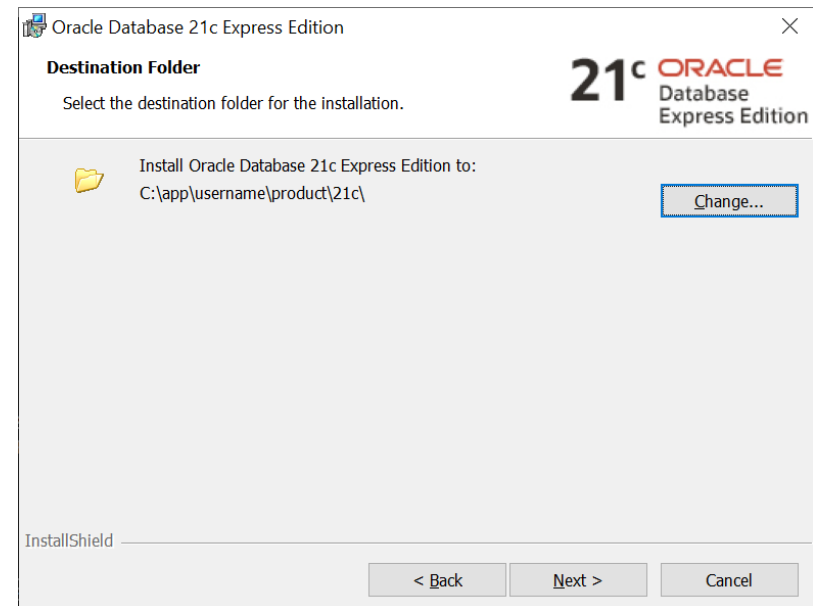
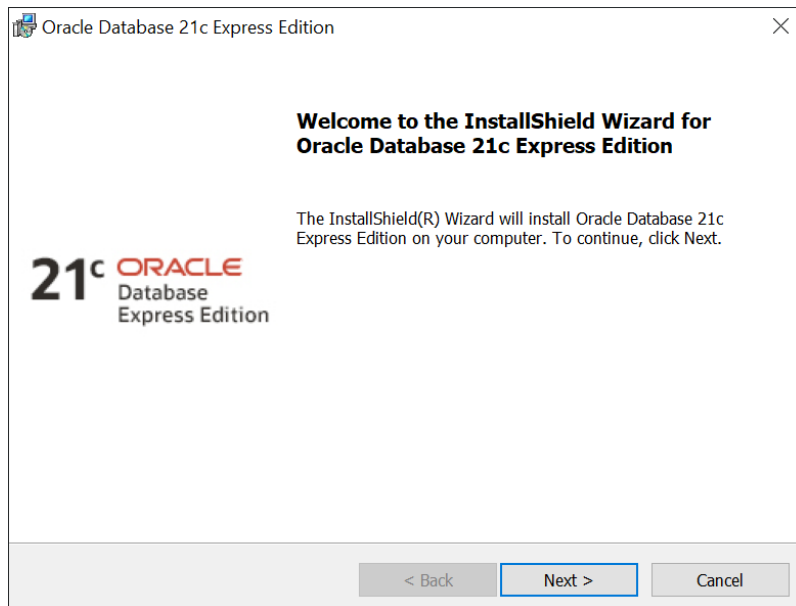
The screenshot shows the Oracle Database XE Downloads page. The browser address bar displays <https://www.oracle.com/database/technologies/xe-downloads.html>. The page header includes the Oracle logo, a search icon, and buttons for 'View Accounts' and 'Contact Sales'. The navigation menu lists 'Products', 'Industries', 'Resources', 'Customers', 'Partners', 'Developers', and 'Events'. The breadcrumb trail indicates the path: 'Database / Technologies / Oracle Database Express Edition (XE) Downloads'. The main heading is 'Oracle Database XE Downloads'. Below this, the section 'Oracle Database 21c Express Edition' is highlighted. A table lists the available downloads:

Download	Description
 Oracle Database 21c Express Edition for Windows x64	(1,967,615,483 bytes - October 08, 2021) [Sha256sum: 939742c3305c466566a55f607638621b6aa7033a183175f6bcd6cffb48e
 Oracle Database 21c Express Edition for Linux x64 (OL8)	(2,339,651,768 bytes - September 08, 2021) [Sha256sum: f8357b432de33478549a76557e8c5220ec243710ed86115c65b0

Oracle Express Edition

- 설치

- Administrator(윈도우즈 계정) 권한이 있는 계정으로 로그인 후 다운로드 실행
- Zip 파일을 압축 해제하고 database/setup.exe 파일을 실행
- 설치 폴더 지정

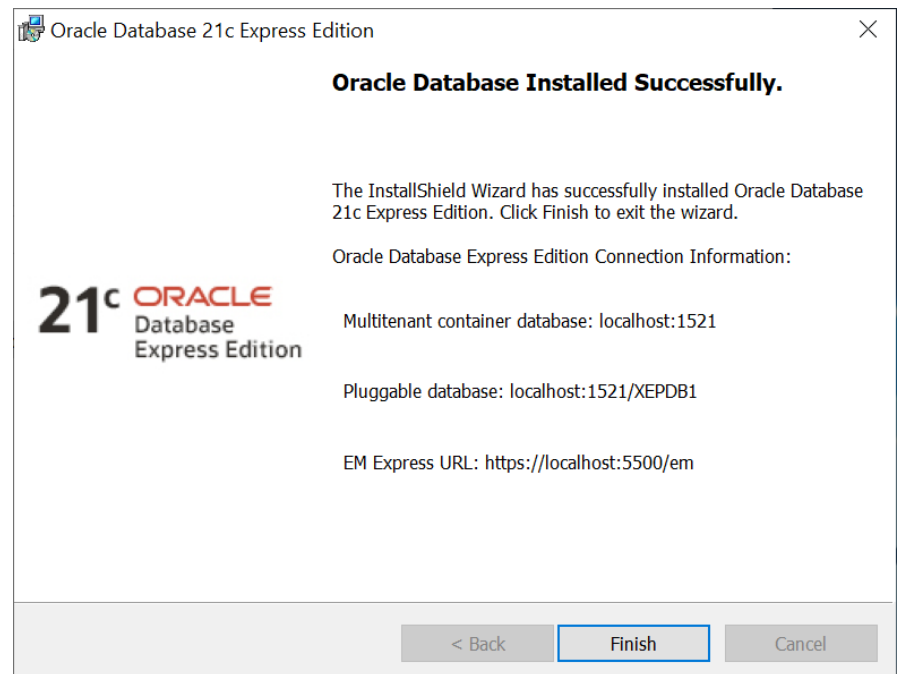
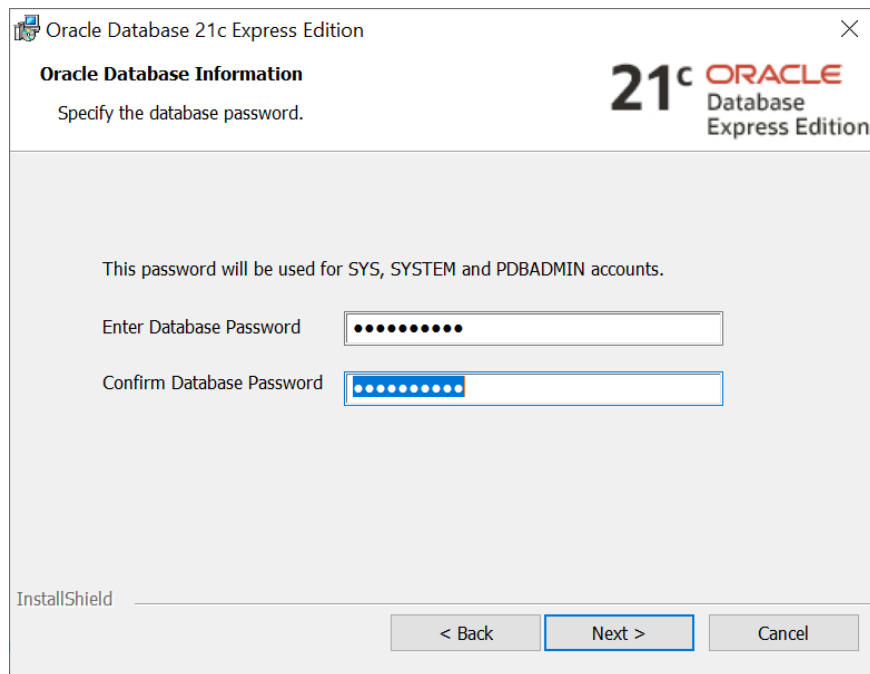


Oracle Express Edition

- 설치

- 시스템 관리자 계정(SYS, **SYSTEM**)을 위한 암호 설정
- 자세한 사항은 Installation Guide 참조

▶ <https://docs.oracle.com/en/database/oracle/oracle-database/21/xeinw/index.html>



Oracle Express Edition

- Windows 메뉴 및 Service 등록 확인
 - OracleServiceXE, OracleOraDB21Home1TNSListener 서비스 실행 중

The screenshot shows the Windows Task Manager 'Services' tab. The left sidebar lists various services, with 'Oracle - OraDB21Home3' expanded. The main pane shows a list of services with columns for Name, Description, Status, and Startup Type. The 'OracleJobSchedulerXE' service is highlighted in blue. Below the list, the 'Startup' tab is selected, showing the 'Startup' and 'Standard' tabs.

이름	설명	상태	시작 유형
OpenSSH Authentication Agent	Agent to hold private keys used...		사용 안 함
Optimize drives	저장소 드라이브의 파일을 최적...		수동
OracleJobSchedulerXE			사용 안 함
OracleOraDB21Home3MTSRecoveryService		실행 중	자동
OracleOraDB21Home3TNSListener		실행 중	자동
OracleServiceXE		실행 중	자동
OracleVssWriterXE		실행 중	자동
Peer Name Resolution Protocol	서버를 사용하지 않고 PNRP(피...		수동
Peer Networking Grouping	피어 네트워킹 그룹을 사용한 다...		수동
Peer Networking Identity Manager	PNRP(피어 이름 확인 프로토콜)...		수동
Performance Counter DLL Host	원경 사용자 및 64-비트 프로세...		수동
Performance Logs & Alerts	서버 로그 및 경고는 미리 그려...		수동

Oracle Express Edition

- 설치 폴더

File Name and Location	Purpose
<INSTALL_DIR>	Oracle Base This is the root of the Oracle Database XE directory tree.
<INSTALL_DIR>\dbhomeXE	Oracle Home This home is where the Oracle Database XE is installed. It contains the directories of the Oracle Database XE executables and net work files.
<INSTALL_DIR>\oradata\XE	Database files
<INSTALL_DIR>\diag\rdbms\XE\XE\trace	Diagnostic logs The database alert log is <INSTALL_DIR>\diag\rdbms\XE\XE\trace\alert_XE.log
<INSTALL_DIR>\cfgtoollogs\	Database installation, creation, and configuration logs. The <INSTALL_DIR>\cfgtoollogs\dbca\XE\XE.log file contains the results of the database creation script execution.

Oracle Express Edition

- 설치 폴더

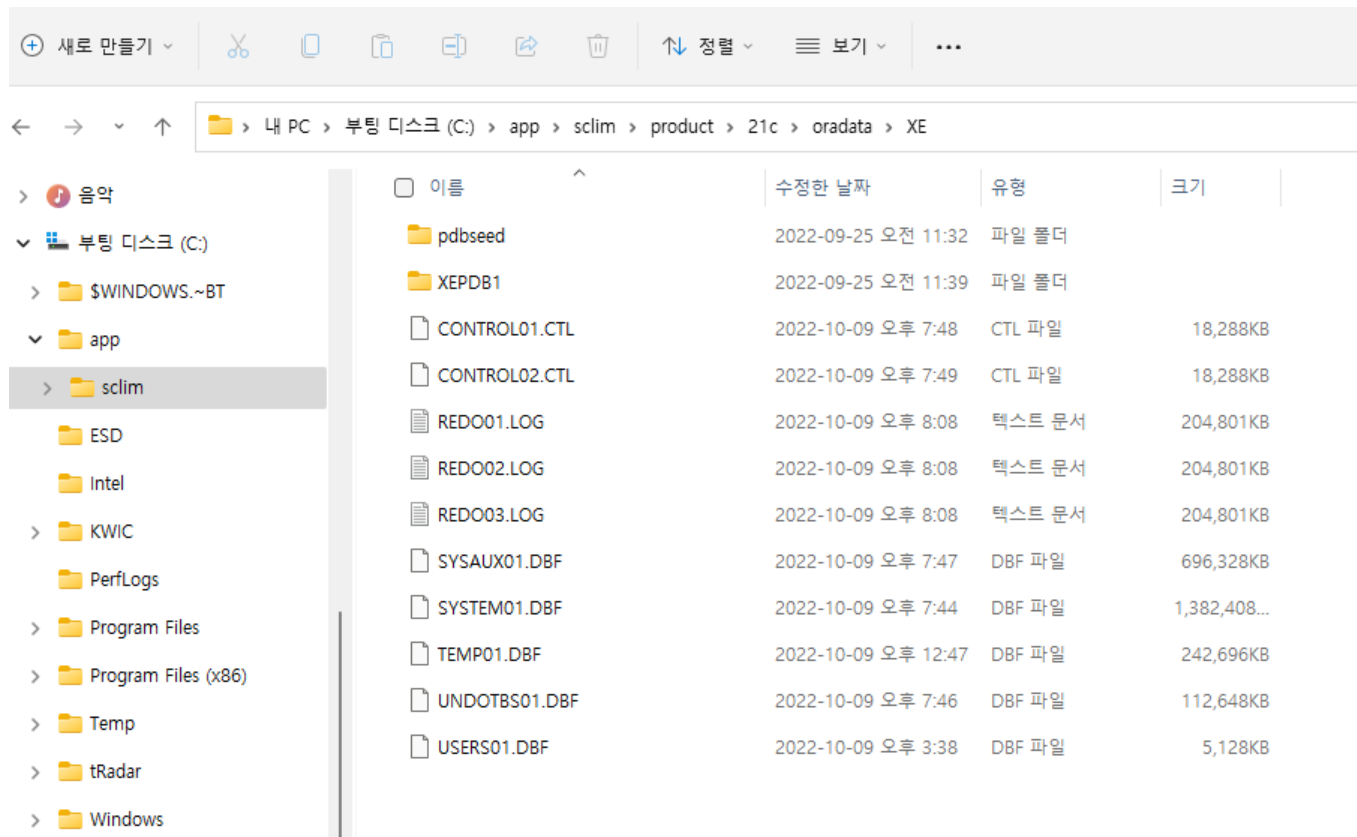
Windows File Explorer window showing the directory structure for Oracle Express Edition installation. The path is C:\app\scim\product\21c\dbhomeXE\bin. The file sqlplus.exe is selected.

이름	수정된 날짜	유형	크기
rman.exe	2021-09-27 오후 4:00	응용 프로그램	3,615KB
sbttest.exe	2021-09-27 오후 4:00	응용 프로그램	85KB
schagent.exe	2021-09-27 오후 4:06	응용 프로그램	90KB
schema.exe	2021-07-09 오후 3:18	응용 프로그램	33KB
sclsspawn.exe	2021-09-23 오후 12:32	응용 프로그램	29KB
sql.exe	2021-06-23 오후 6:21	응용 프로그램	138KB
sqlldr.exe	2021-09-27 오후 3:59	응용 프로그램	1,536KB
sqlplus.exe	2021-07-12 오전 1:23	응용 프로그램	1,358KB
tkprof.exe	2021-09-27 오후 3:59	응용 프로그램	26KB
tnslsnr.exe	2021-07-09 오후 12:44	응용 프로그램	939KB
tnsping.exe	2021-07-09 오후 12:44	응용 프로그램	73KB
uidrvci.exe	2021-09-27 오후 4:06	응용 프로그램	32KB

Oracle Express Edition

- 주요 파일

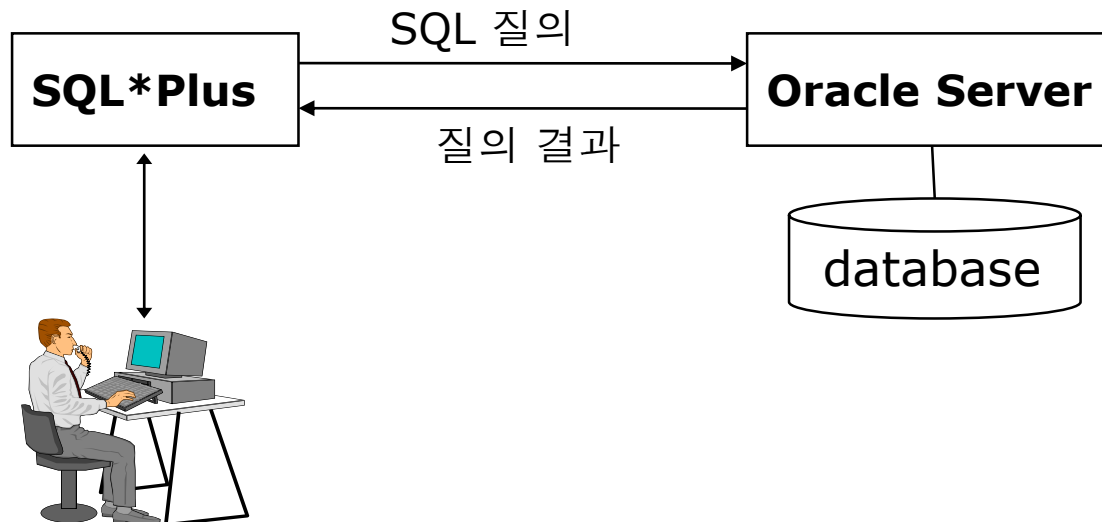
- CONTROL.CTL, REDO.LOG: 시스템 관리와 복구를 위한 정보 저장
- UNDOTBS.DBF: 데이터 복구를 위한 정보 저장
- USERS.DBF: 사용자가 생성한 테이블(데이터) 저장



이름	수정한 날짜	유형	크기
pdbseed	2022-09-25 오전 11:32	파일 폴더	
XEPDB1	2022-09-25 오전 11:39	파일 폴더	
CONTROL01.CTL	2022-10-09 오후 7:48	CTL 파일	18,288KB
CONTROL02.CTL	2022-10-09 오후 7:49	CTL 파일	18,288KB
REDO01.LOG	2022-10-09 오후 8:08	텍스트 문서	204,801KB
REDO02.LOG	2022-10-09 오후 8:08	텍스트 문서	204,801KB
REDO03.LOG	2022-10-09 오후 8:08	텍스트 문서	204,801KB
SYSAUX01.DBF	2022-10-09 오후 7:47	DBF 파일	696,328KB
SYSTEM01.DBF	2022-10-09 오후 7:44	DBF 파일	1,382,408...
TEMP01.DBF	2022-10-09 오후 12:47	DBF 파일	242,696KB
UNDOTBS01.DBF	2022-10-09 오후 7:46	DBF 파일	112,648KB
USERS01.DBF	2022-10-09 오후 3:38	DBF 파일	5,128KB

SQL*Plus 개요

- Oracle DBMS에서 사용자가 SQL 질의를 작성 및 실행할 수 있도록 인터페이스를 제공하는 클라이언트 도구
 - Oracle Server에 접속하여 대화식으로 이용
 - ▶ local 뿐만 아니라 remote에 있는 서버에도 접근 가능
 - SQL문과 Oracle PL/SQL문을 수행하며, 자체 명령어를 제공
 - 주의: SQL은 데이터베이스를 접근하기 위해 사용되는 언어이고, SQL*Plus는 SQL문과 PL/SQL 코드를 실행시키기 위한 도구임

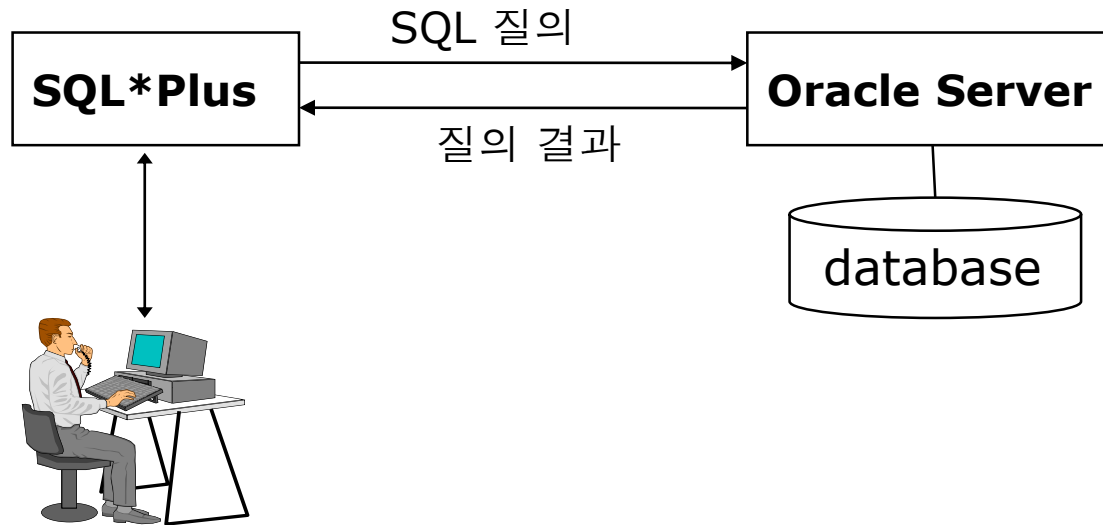


SQL*Plus 개요

- 주요 기능
 - 데이터베이스 접속 및 종료
 - 테이블 생성, 편집, 저장 및 검색
 - SQL 문 또는 PL/SQL 프로그램 코드를 작성 및 실행 가능
 - 질의 결과를 정렬하고 지정된 형식으로 출력 (보고서 작성)
 - 데이터베이스 간의 데이터 복사 등 데이터베이스 관리 작업 수행

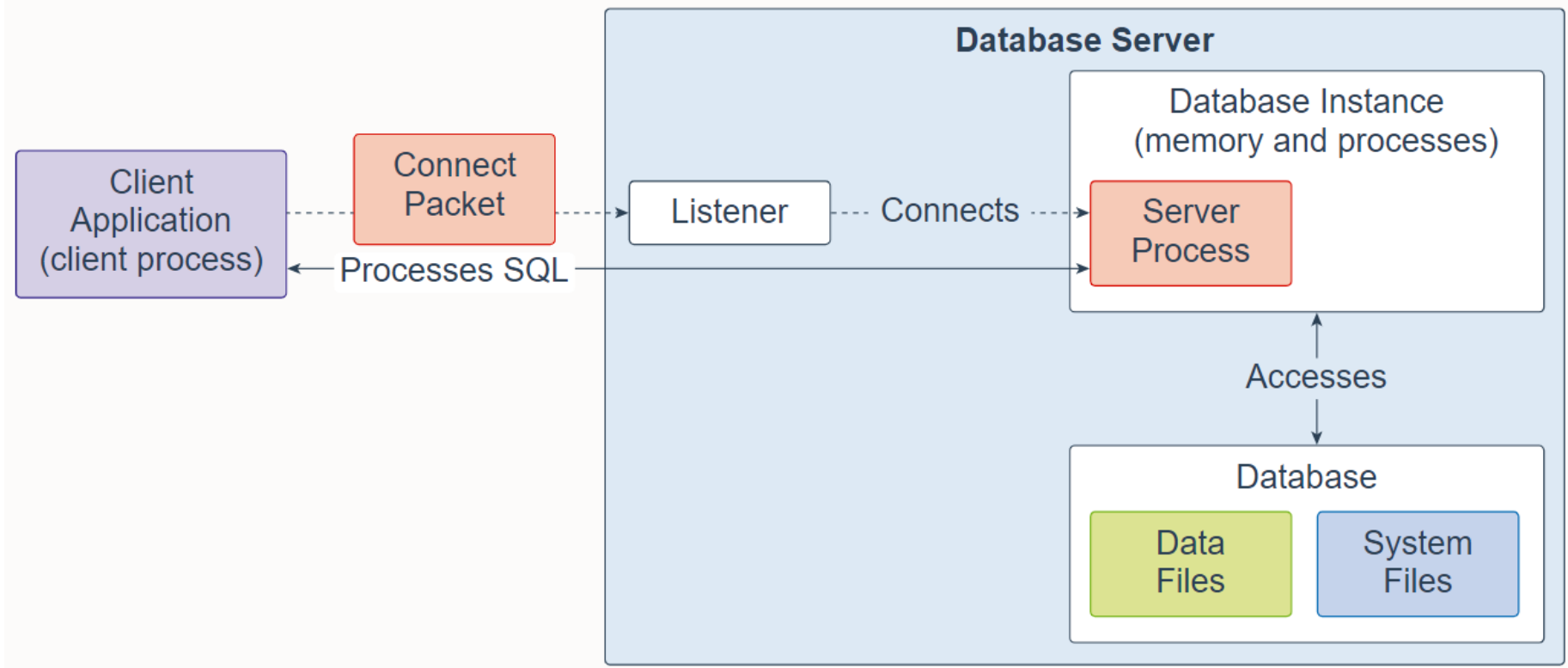
SQL*Plus 개요

- Oracle DBMS에서 사용자가 SQL 질의를 작성하고 실행할 수 있도록 하는 콘솔 프로그램
 - Oracle Server에 접속하여 대화식으로 이용
 - ▶ local 뿐만 아니라 remote에 있는 서버에도 접근 가능
 - SQL문과 Oracle PL/SQL문을 수행하며, 자체 명령어를 제공



Architecture of Oracle DB

- 우리가 사용하는 실습환경인 sqlplus는 아래 그림의 client app.에 해당



Oracle SQL Developer

- 다운로드 및 설치
 - 검색: "oracle sql developer download"
 - Windows 64-bit with JDK 11 included 버전을 선택하여 다운로드
 - Zip 파일을 압축 해제 후 sqldeveloper.exe 실행

The screenshot shows the Oracle website's 'SQL Developer Downloads' page. The header includes the Oracle logo and navigation links like Products, Industries, Resources, Customers, Partners, Developers, Events, and Company. Below the header, the breadcrumb 'Database / SQL Developer / SQL Developer Downloads' is visible. The main heading is 'SQL Developer 22.2.1 Downloads', with a subtext 'Version 22.2.1.234.1810 - September 12, 2022'. There are links for 'Release Notes' and 'Documentation'. A table lists two download options: 'Windows 64-bit with JDK 11 included' (444 MB) and 'Windows 32-bit/64-bit' (490 MB). Each row provides a download link, the file size, and a list of notes including MD5, SHA1, and installation instructions.

Platform	Download	Notes
Windows 64-bit with JDK 11 included	Download (444 MB)	<ul style="list-style-type: none">• MD5: c917657ae52a31af66b3cd16f100cf56• SHA1: 9da21417bcd8a8fab4bba985ffd1d9087be16b2• Installation Notes
Windows 32-bit/64-bit	Download (490 MB)	<ul style="list-style-type: none">• MD5: 92fe45220660c178ad8656c43ea6e9e2• SHA1: b0a6e46b314bb5f976b42564c361d589cc2795a9• Installation Notes• JDK 11 required

Oracle SQL Developer

- SQL 질의 작성 및 실행

The screenshot displays the Oracle SQL Developer interface with the following components:

- Left Panel:** Contains the 'Oracle 접속' (Oracle Connections) tree and the '보고서' (Reports) section. A red arrow points to the '실행' (Execute) button in the toolbar.
- SQL Editor:** Displays the query: `SELECT * FROM emp WHERE deptno = '10';`. A red arrow points to the text with the label '1. 질의 작성(편집)'.
- Results Panel:** Shows the execution results in a table format. A red arrow points to the table with the label '3. 결과 확인'.

2. 실행

1. 질의 작성(편집)

3. 결과 확인

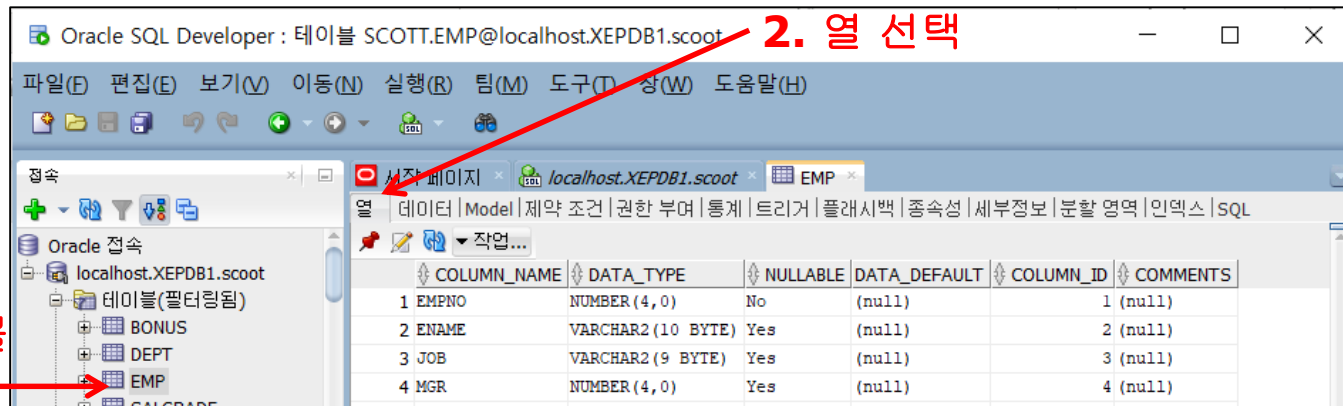
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7782 CLARK	MANAGER	7839	81/06/09	2450	(null)	10
2	7839 KING	PRESIDENT	(null)	81/11/17	5000	(null)	10
3	7934 MILLER	CLERK	7782	82/01/23	1300	(null)	10

http://www.oracle.com/technetwork/developer-tools/sqlcl/overview/index.html | 3 행 21 열 | 삽입 | 수정됨 | Windows: CR

Oracle SQL Developer

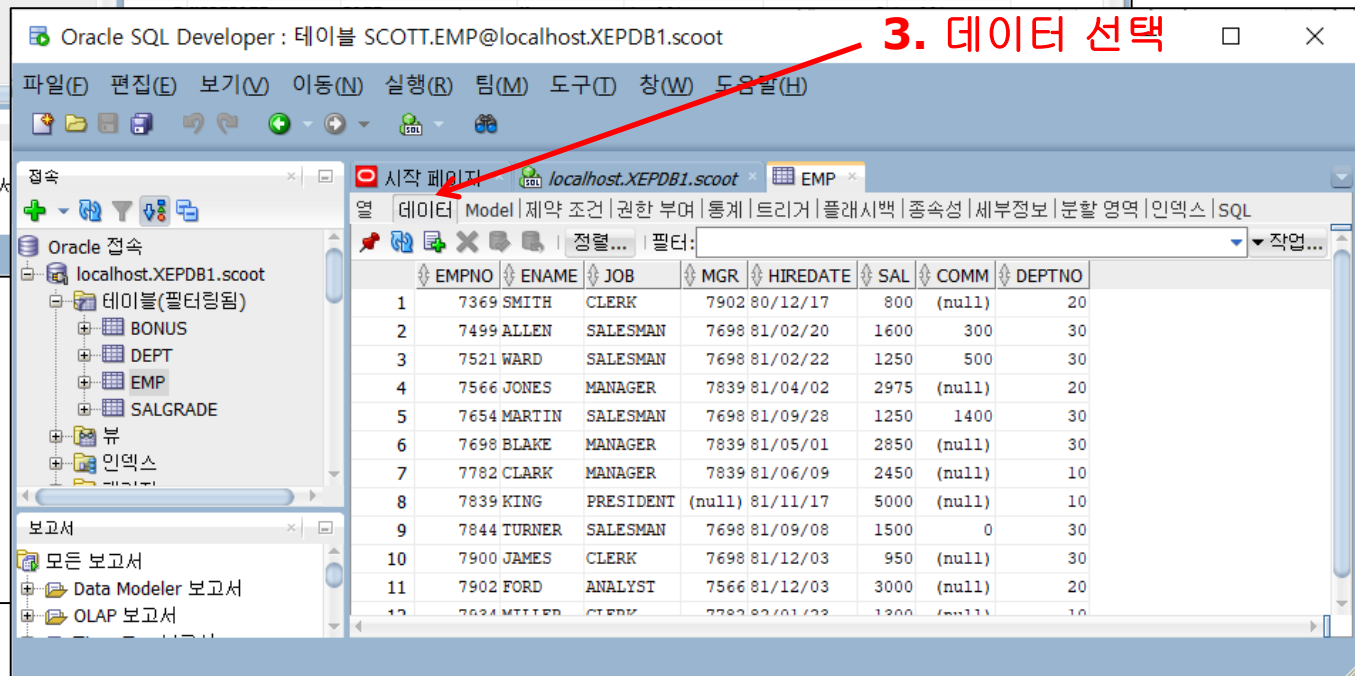
- 테이블 구조 및 데이터 확인

1. 테이블
선택



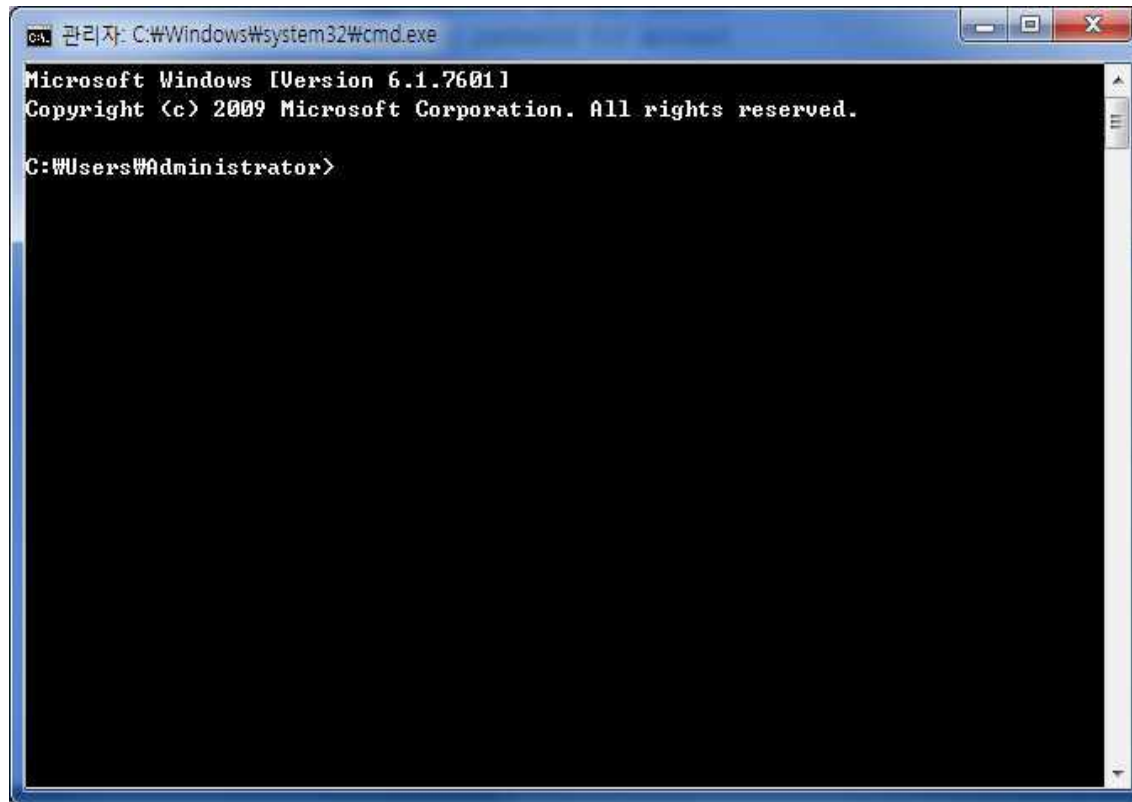
2. 열 선택

3. 데이터 선택



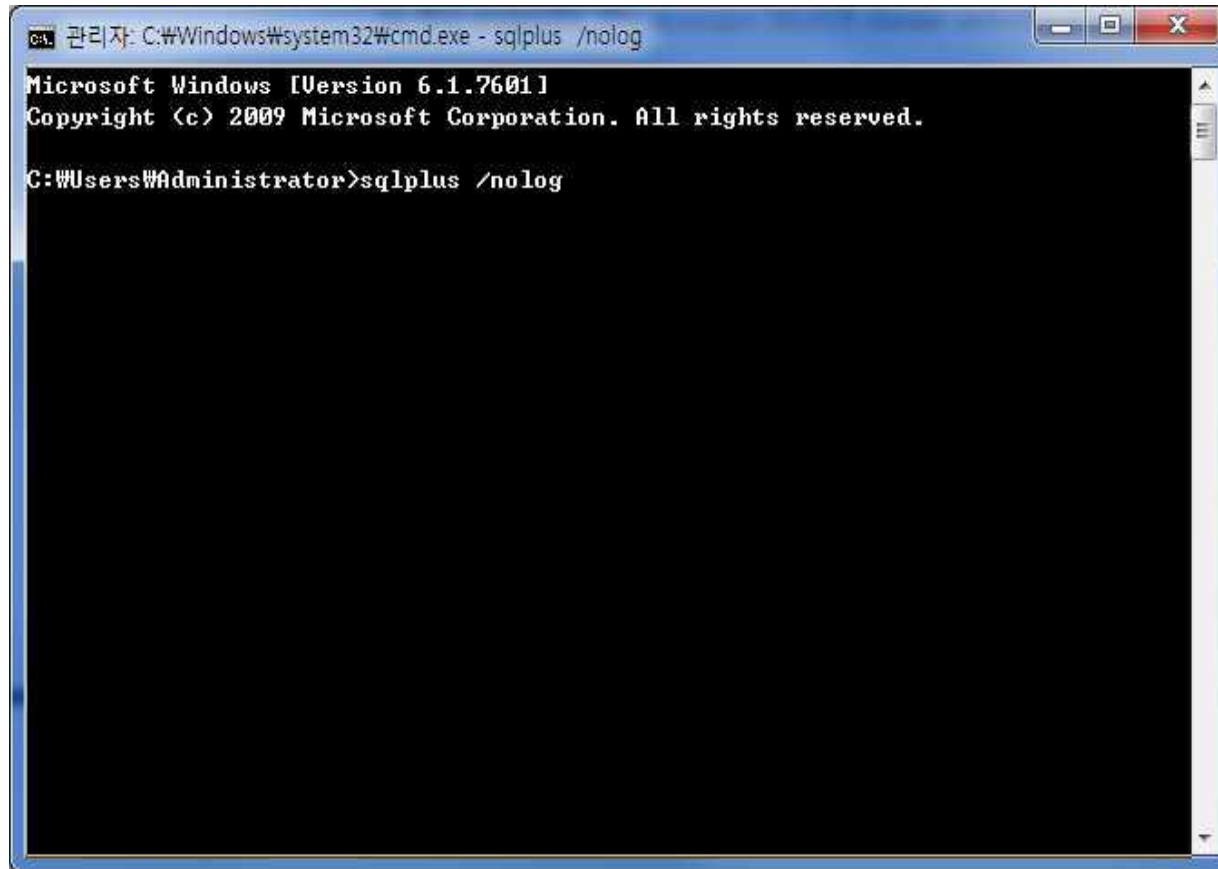
사용자 계정의 잠금 해제 방법

- 실행창에서 **cmd** 명령어를 입력해서 명령어 입력창을 실행



사용자 계정의 잠금 해제 방법

- `sqlplus /nolog` 명령으로 SQL*Plus 를 실행



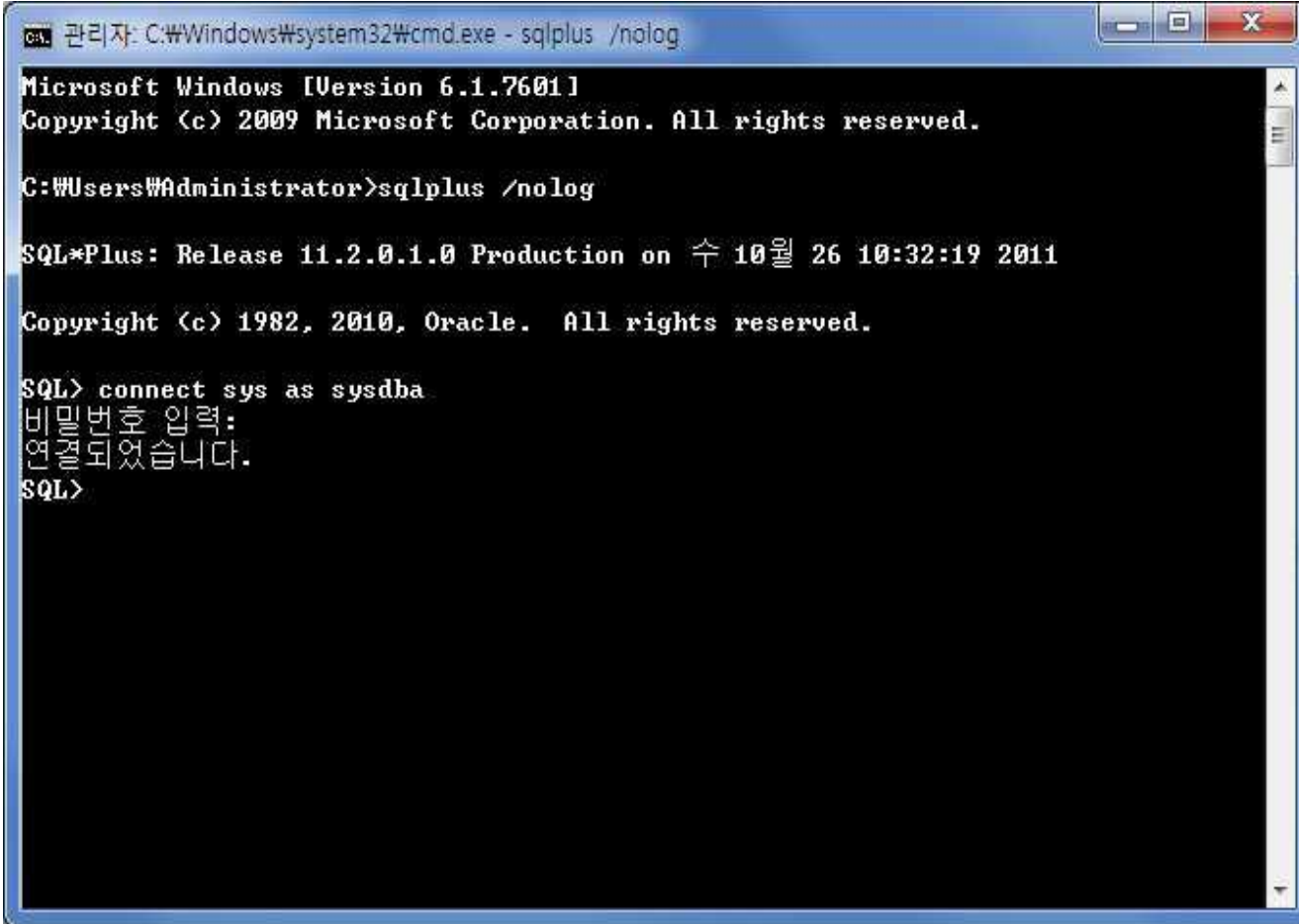
A screenshot of a Windows command prompt window. The title bar reads "관리자: C:\Windows\system32\cmd.exe - sqlplus /nolog". The window content shows the following text:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>sqlplus /nolog
```

사용자 계정의 잠금 해제 방법

- **connect sys as sysdba** 명령으로 시스템 관리자인 **sysdba** 계정으로 접속



```
관리자: C:\Windows\system32\cmd.exe - sqlplus /nolog

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>sqlplus /nolog

SQL*Plus: Release 11.2.0.1.0 Production on 수 10월 26 10:32:19 2011

Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> connect sys as sysdba
비밀번호 입력:
연결되었습니다.
SQL>
```

접속의 시작

- **sqlplus /nolog** 명령으로 SQL*Plus 를 실행
 - “/nolog” 옵션 사용시 계정 입력 화면이 나오지 않고 수행됨
- “**connect sys as sysdba**” 명령으로 시스템 관리자인 **sysdba** 계정으로 접속

```
C:\Users\Administrator>sqlplus /nolog

SQL*Plus: Release 11.2.0.1.0 Production on 수 10월 26 10:32:19 2011

Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> connect sys as sysdba
비밀번호 입력:
연결되었습니다.
SQL>
```

PL/SQL 실행 ** 이후 수업에서 다루겠음

- PL/SQL

- 오라클에서 DBMS의 표준 질의어인 SQL을 확장하여 개발한 고급 프로그래밍 언어
- SQL이 가진 편리함과 유연함에 '제어문', '반복문' 등과 같은 구조적 프로그래밍 언어의 요소가 결합
- 기본 단위는 블록(block)
 - ▶ 변수를 선언하는 부분인 **선언부**와 실행코드가 나오는 **실행부**, 실행 중 에러가 발생했을 때 실행되는 **예외처리부**로 구성

PL/SQL 실행하기

- PL/SQL의 기본 형식

```
DECLARE
    변수 선언문;
BEGIN
    실행문;
EXCEPTION
    예외처리문;
END;
```

예) 변수 n을 생성하고 10을 저장한 후 출력

1	DECLARE
2	n INTEGER;
3	BEGIN
4	n := 10;
5	dbms_output.put_line(n);
6	END;

테이블스페이스 관리

- 테이블스페이스 생성
- 테이블스페이스 변경
- 테이블스페이스 조회

테이블 스페이스 생성

- 오라클에서 테이블을 생성하려면 테이블 스페이스를 사용해야 함
- 테이블 스페이스 사용
 - 오라클을 설치할 때 만들어지는 기본 테이블스페이스를 사용
 - 또는 새로운 테이블스페이스를 생성하여 사용
- 테이블 스페이스는 오라클 관리자만이 생성
- 생성할 때 실제 데이터가 저장될 디스크 상의 파일인 데이터파일을 지정

테이블스페이스 생성

- 형식

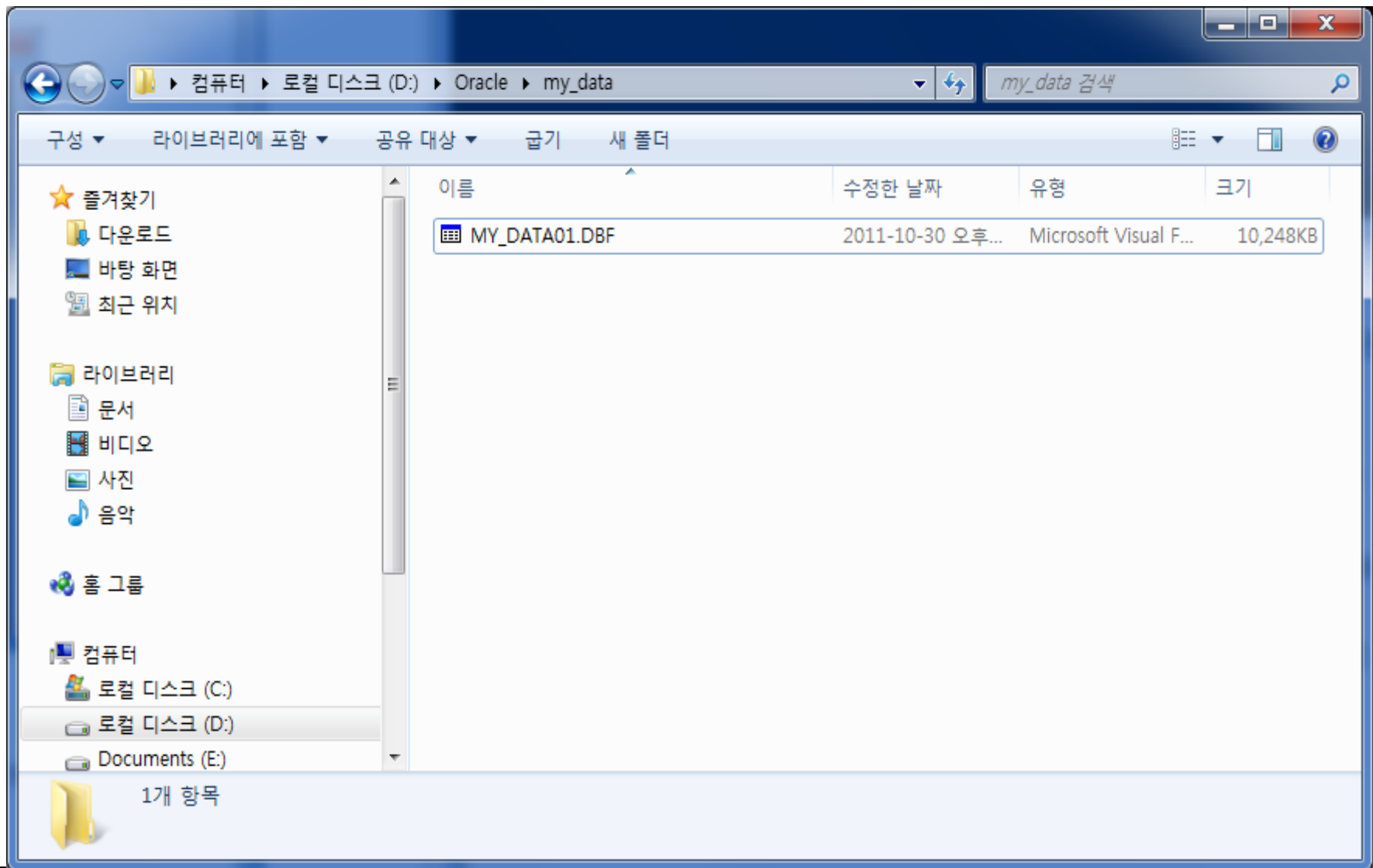
create tablespace <테이블스페이스 이름>

datafile '<데이터파일 경로명>' **size** <데이터파일 크기>

```
SQL> create tablespace my_space  
2 datafile 'd:\#Oracle\my_data\my_data01.dbf' size 10M;
```

테이블스페이스가 생성되었습니다.

생성된 데이터파일 확인



테이블스페이스 변경

- 테이블스페이스에 새로운 데이터파일을 추가

- 형식

alter tablespace <테이블스페이스 이름>

add datafile '<데이터파일 경로명>' **size** <데이터파일 크기>

```
SQL> alter tablespace my_space  
2 add datafile 'd:\Oracle\my_data\my_data02.dbf' size 10M;
```

테이블스페이스가 변경되었습니다.

테이블스페이스 변경

- 테이블스페이스를 삭제할 때에는 **drop tablespace** 명령을 사용
- 형식
drop tablespace <삭제할 테이블스페이스 이름>

테이블스페이스의 사용

- 사용자 계정 생성
- 형식

create user <사용자 계정>

identified by <비밀번호>

default tablespace <사용할 테이블스페이스 이름>

quota <용량> **on** <사용할 테이블스페이스 이름>

테이블스페이스의 사용

무제한(unlimited)
또는 20K, 5M

```
SQL> create user jimmy  
2 identified by jimmy_pass  
3 default tablespace my_space  
4 quota unlimited on my_space;
```

사용자가 생성되었습니다.

사용자 계정 scott 으로 접속 – 현재는 없음

SQL명령입력

(SQL을 실행하려면 반드시 ;으로 실행문을 끝내야 함)

```
SQL> select ename, sal from emp;
```

세미콜론 없이 enter를 치면 여러 줄로 입력 가능

```
SQL> select ename, sal  
2 from emp  
3 where sal > 10000;
```

SQL*Plus 명령어 실행

- 테이블 구조 보기 – **desc(ribe)**
 - 테이블에 어떤 필드들이 정의되어 있는지 확인
 - 형식
 - ▶ **desc** <테이블 명>

```
SQL> desc emp
```

이름

NULL?

유형

	NULL?	유형
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(2)

SQL*Plus 명령어 실행

- 직전에 실행했던 명령문 보기 - **list**
 - 바로 직전에 실행시켰던 명령을 출력
 - 형식
 - ▶ list

```
SQL> select empno, ename, job, sal
2  from emp
3  where sal > 1500;
```

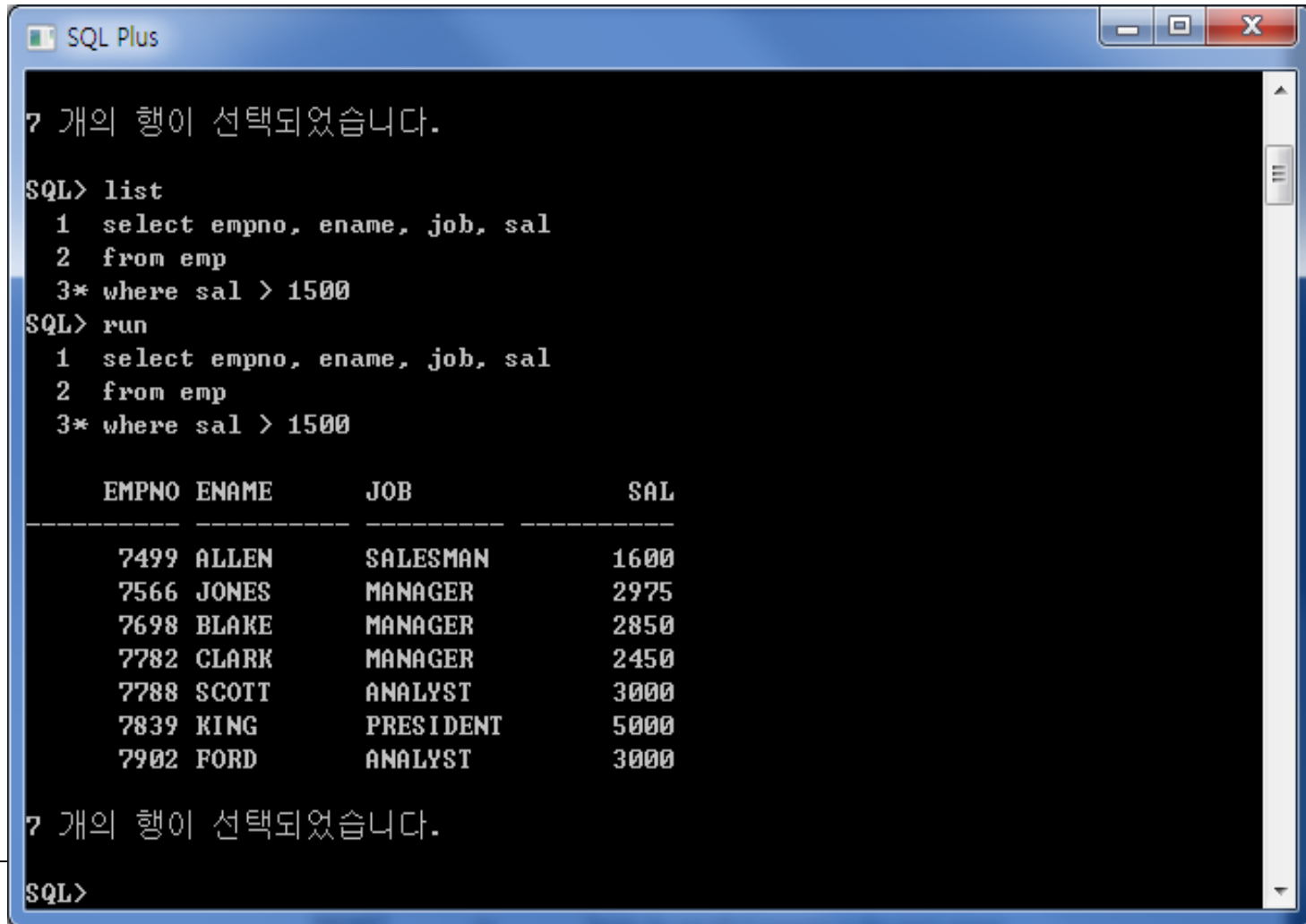
EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	1600
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7788	SCOTT	ANALYST	3000
7839	KING	PRESIDENT	5000
7902	FORD	ANALYST	3000

7 개의 행이 선택되었습니다.

```
SQL> list
1  select empno, ename, job, sal
2  from emp
3* where sal > 1500
SQL>
```

SQL*Plus 명령어 실행

- 직전에 실행했던 명령문 다시 실행하기 – **run**
 - 직전에 실행했던 명령문을 다시 입력하지 않고 반복해서 실행
 - 형식
 - ▶ **run**



The screenshot shows a SQL*Plus window with a blue title bar. The text inside the window is as follows:

```
7 개의 행이 선택되었습니다.  
  
SQL> list  
  1  select empno, ename, job, sal  
  2  from emp  
  3* where sal > 1500  
SQL> run  
  1  select empno, ename, job, sal  
  2  from emp  
  3* where sal > 1500  
  
      EMPNO ENAME      JOB      SAL  
-----  
      7499 ALLEN      SALESMAN    1600  
      7566 JONES      MANAGER    2975  
      7698 BLAKE      MANAGER    2850  
      7782 CLARK      MANAGER    2450  
      7788 SCOTT      ANALYST    3000  
      7839 KING      PRESIDENT  5000  
      7902 FORD      ANALYST    3000  
  
7 개의 행이 선택되었습니다.  
  
SQL>
```

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	1600
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7788	SCOTT	ANALYST	3000
7839	KING	PRESIDENT	5000
7902	FORD	ANALYST	3000

SQL*Plus 명령어 실행

- 직전에 실행했던 명령문 파일로 저장하기 - **save**
 - SQL*Plus에서 실행시킨 명령문을 종류에 상관없이 파일로 저장
 - 형식
 - ▶ **save** <파일 이름>

```
SQL> select empno, ename, job, sal  
2 from emp  
3 where sal > 1500;
```

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	1600
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7788	SCOTT	ANALYST	3000
7839	KING	PRESIDENT	5000
7902	FORD	ANALYST	3000

7 개의 행이 선택되었습니다.

```
SQL> save emp  
file emp.sql<이>가 생성되었습니다
```

저장된 파일은 오라클
시스템 폴더에 저장

C:\wappw[윈도우즈 사용자 이름]\wproduct\11.2.0\wdbhome_1\BIN

SQL*Plus 명령어 실행

- 저장된 명령문 파일 불러오기 - **get**
 - **save**로 저장된 명령문을 불러올 때에는 **get** 명령어를 사용
 - 형식
 - ▶ **get** <파일 이름>

```
SQL> get emp
1 select empno, ename, job, sal
2 from emp
3* where sal > 1500
SQL> run
1 select empno, ename, job, sal
2 from emp
3* where sal > 1500
```

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	1600
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7788	SCOTT	ANALYST	3000
7839	KING	PRESIDENT	5000
7902	FORD	ANALYST	3000

SQL*Plus 명령어 실행

- 운영체제 명령어 실행시키기 - **host**
 - SQL*Plus 실행 도중 운영체제 명령을 실행
 - 형식
 - ▶ **host**
 - 복귀는 exit

```
SQL> host
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\app\Administrator\product\11.2.0\dbhome_1\BIN>exit

SQL> _
```


SQL*Plus 명령어 실행

- 기타 명령어
 - **connect** 계정명
 - ▶ 계정명으로 DB 연결(로그인)
 - ▶ conn으로 명령어 축약 가능
 - ▶ Sql> conn tiger
 - ▶ Sql> conn system
 - ▶ Sql> conn sys as sysdba
 - cl(ear) src
 - ▶ 화면 지우기
 - quit, exit
 - ▶ SQL*Plus를 종료

System 계정 사용 연습

- 앞서 수행한 sqlplus 콘솔에서 아래 명령어 입력
 - > conn(ect) system
 - System이란 Id로 DBMS에 연결(통신 연결을 통해 연결). 패스워드 입력
 - > disc(onnect)
 - 로그인 연결에서 빠져 나옴
- 앞에서 한 실습을 다시 한번 반복(암기해서...)
 - conn/disc
 - cl scr
 - host/exit
- System으로 로그인 한 후 아래 sql 문 수행
 - > **select** username **from** dba_users;
 - 현재 이미 DBMS에 생성되어 있는 계정의 이름을 리스팅

계정 및 테이블의 생성

- 계정 생성하는 SQL 문
 - **주의**) system으로 로그인한 상태에서 수행 (로그인 상태 확인? "show user")
 - Sql> **create user sclim identified by 1;**
 - 계정 생성 확인:
select username from dba_users;
 - 위의 sql 문 수행의 결과 시스템에 sclim 계정이 생성
 - 계정이 생성되었지만 **로그인할 수 있는 권한 및 테이블을 생성할 수 있는 권한이 없음**
 - 이런 권한을 추가로 부여해야 함 (by system)

계정 및 테이블의 생성

- 로그인 할 수 있는 권한(네트워크 접속 즉, session 생성 권한) 부여
 - System 계정자가 아래 sql문 수행
 - Sql> grant connect to sclim;
- 테이블을 생성할 수 있는 권한 주기
 - Sql> grant resource to sclim;
- 보통은 아래처럼 한번에 권한을 부여함(권한 리스트 사용)
 - Sql> grant connect, resource to sclim;

주의) 18c 버전부터는 아래처럼 계정 생성

- SQL> **create user** c##sclim identified by 1;
- SQL> **grant** connect, resource to c##sclim ; // 권한 부여
- SQL> conn c##sclim ; // 로그인

- SQL> **drop user** c##sclim ; // 계정 삭제 시

- **C##을 붙이고 싶지 않다면**
 - SQL> ALTER SESSION SET "_ORACLE_SCRIPT"=true;

계정 및 테이블의 생성

- 테이블을 만들어 보자 (주의!!!! Sclim으로 로그인한 상태에서 수행)
 - **SQL> create table test_table (name varchar2(10), age int);**
 - 위의 문에서 붉은색 표시된 이름이 바로 필드 명
 - 위의 sql 문을 일반 사용자가 수행하도록 한다.
 - 절대 system(or sys)로는 수행하지 않는다
- 실습 중에 내가 어떤 계정 상태인지 헛갈림. 그렇다면 아래와 같이 입력
 - ▶ SQL> show user
 - ▶ 현재 어떤 계정으로 접속 중인지 확인 바람
 - ▶ 질문) 위의 입력에서 ';' 없어도 되나?

실습 종료

- 실습이 끝나면 자신이 만든 계정은 삭제 --- **system 계정에서 수행해야 함**
 - SQL> conn system (계정 변경)
 - System> **drop user** 생성한-계정 **cascade**;
 - 생성 계정을 삭제
 - 뒤에 붙이는 cascade 키워드를 통해 생성된 테이블과 색인이 함께 삭제됨
- 삭제가 되었는지 확인
 - System> select username from dba_users;
 - 읊

실습 과제 **

- 계정 ex_db을 생성한다. 그리고 이 계정에 **적절히 권한을 부여한다** (로그인 연결 및 테이블 생성 가능하도록)
- 다시 ex_db **계정으로 로그인**
- 그리고 ex_db의 권한으로 아래와 같은 구조의 레코드(구조체)와 유사한 필드를 갖는 테이블 person를 생성해 보자("show user"를 통해 사용자 확인)

```
struct person {  
    char name[10];  
    int age;  
    char gender;  
    char address[30];  
};
```

- **계정 생성, 권한 부여, 그리고 테이블 생성하는 과정을 화면 캡처하여 제출한다**

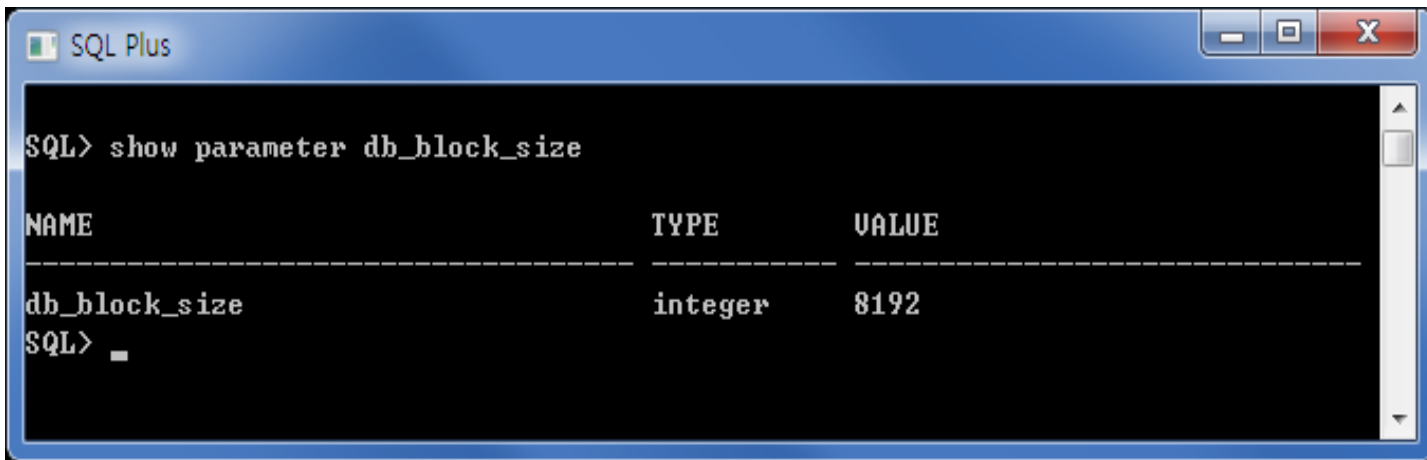
실습 2

오라클 데이터 구조

- 논리적 구성요소
 - 데이터 블록(data block)
 - 익스텐트(extent)
 - 세그먼트(segment)
 - 테이블스페이스(tablespace)

데이터 블록 - 논리적 구성요소

- 데이터가 저장되는 가장 작은 단위
- 저장해야 할 데이터가 늘어나면 데이터 블록의 배수로 저장 공간을 확보하여 저장
- 데이터 블록 표준 크기는 **db_block_size**라는 설정 값에 저장
- 블록 크기 확인 명령
 - **show parameter db_block_size**



```
SQL> show parameter db_block_size
```

NAME	TYPE	VALUE
db_block_size	integer	8192

```
SQL> _
```

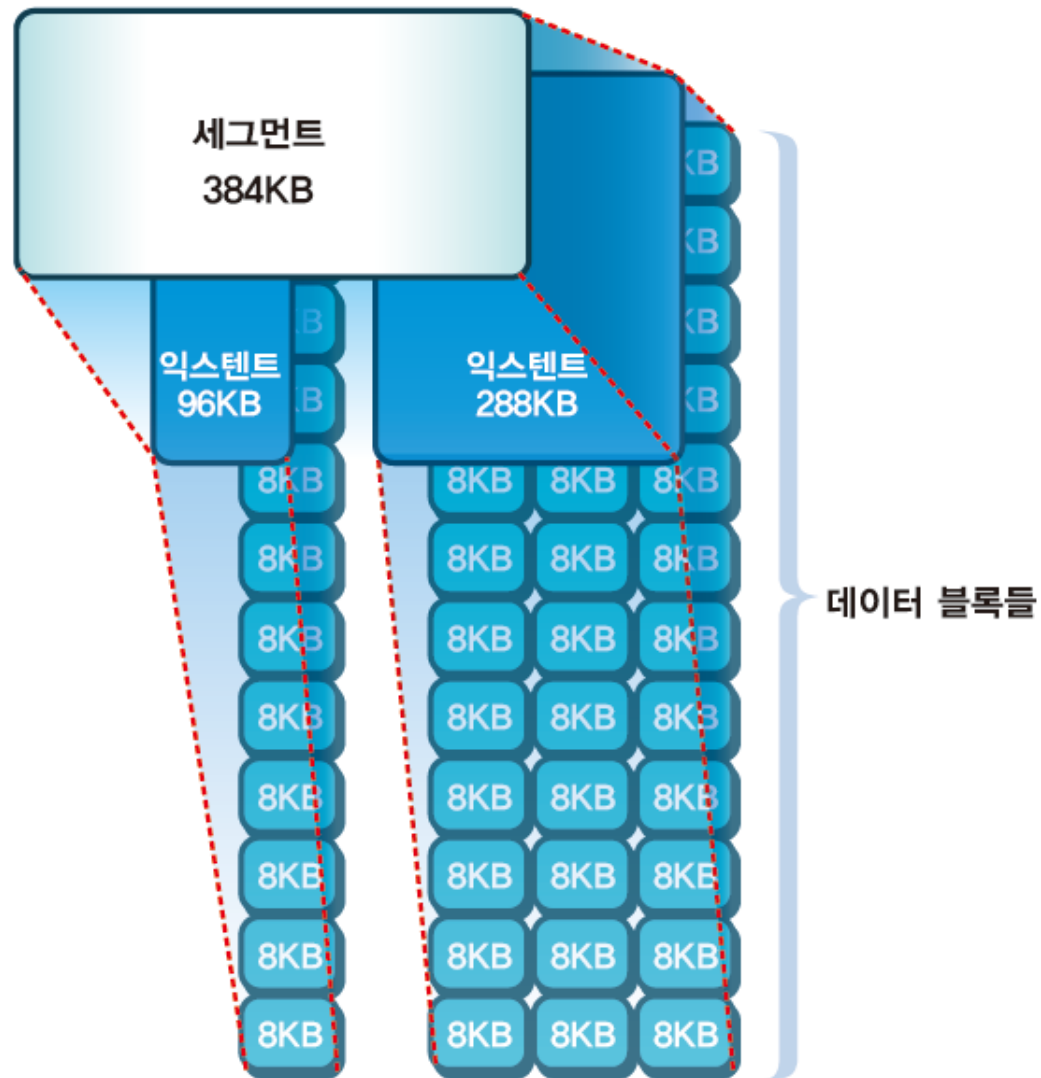
익스텐트(extent)– 논리적 구성요소

- 데이터 블록 다음 단계의 논리적 데이터 저장 공간
- 연속적인 여러 개의 데이터 블록이 모여서 하나의 익스텐트를 구성
- 익스텐트가 모여 다음에 설명할 세그먼트를 구성
 - 하나의 세그먼트에 할당된 공간이 모두 사용되면 오라클은 새로운 익스텐트를 만들어 그 세그먼트에 할당

세그먼트(segment)- 논리적 구성요소

- 여러 개의 익스텐트들이 모여 하나의 세그먼트를 구성
- 하나의 세그먼트에는 같은 종류의 데이터가 저장
 - 데이터 세그먼트
 - ▶ 테이블이 저장되는 세그먼트
 - 인덱스 세그먼트
 - ▶ 인덱스(index) 정보가 저장되는 세그먼트
- 하나의 세그먼트는 뒤에 설명할 하나의 테이블스페이스에 저장
- 하나의 세그먼트를 구성하는 익스텐트들은 디스크상에 연속적으로 저장되지 않을 수도 있음

데이터 블록, 익스텐트, 세그먼트의 관계

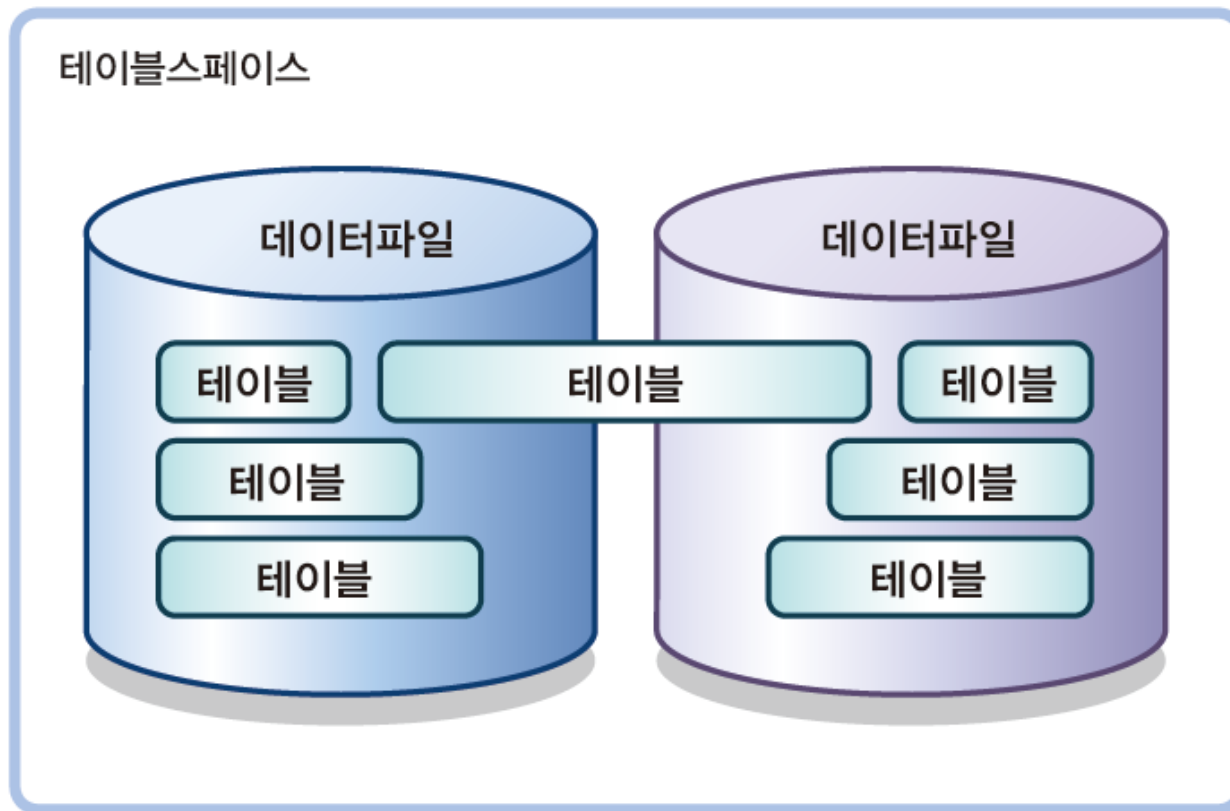


테이블 스페이스(table space)- 논리적 구성요소

- 하나의 데이터베이스는 오라클의 논리적 저장 단위인 테이블 스페이스들로 구성
- 하나의 테이블스페이스에는 하나 이상의 세그먼트를 포함

데이터파일(datafile)- 물리적 구성요소

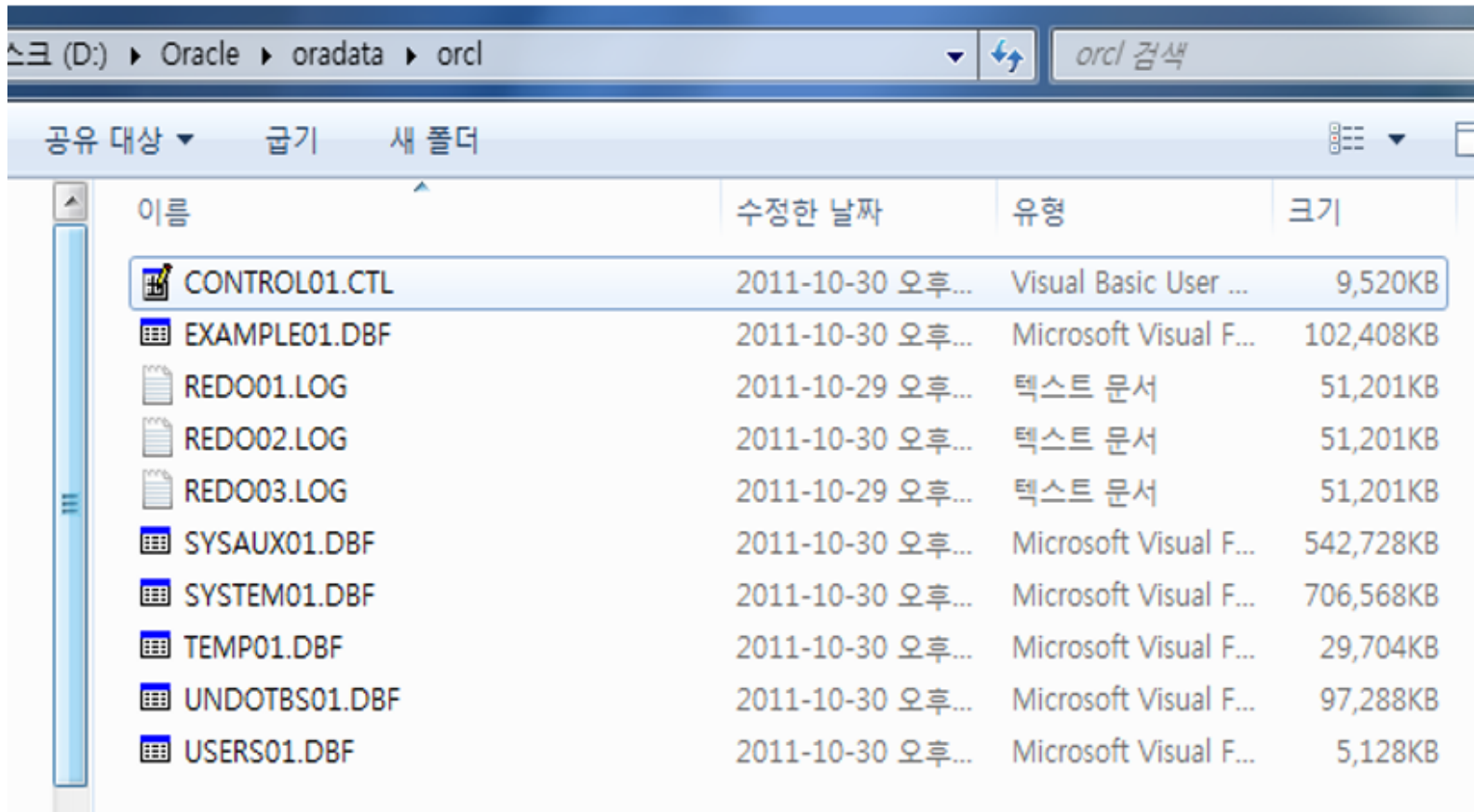
- 오라클에서 관리하는 데이터가 실제로 저장되는 디스크 상의 파일



데이터파일과 테이블스페이스의 관계

데이터파일(datafile)- 물리적 구성요소

- 오라클에서 관리하는 데이터가 실제로 저장되는 디스크 상의 파일
- oradata** 폴더의 데이터파일들



이름	수정한 날짜	유형	크기
CONTROL01.CTL	2011-10-30 오후...	Visual Basic User ...	9,520KB
EXAMPLE01.DBF	2011-10-30 오후...	Microsoft Visual F...	102,408KB
REDO01.LOG	2011-10-29 오후...	텍스트 문서	51,201KB
REDO02.LOG	2011-10-30 오후...	텍스트 문서	51,201KB
REDO03.LOG	2011-10-29 오후...	텍스트 문서	51,201KB
SYS_AUX01.DBF	2011-10-30 오후...	Microsoft Visual F...	542,728KB
SYSTEM01.DBF	2011-10-30 오후...	Microsoft Visual F...	706,568KB
TEMP01.DBF	2011-10-30 오후...	Microsoft Visual F...	29,704KB
UNDOTBS01.DBF	2011-10-30 오후...	Microsoft Visual F...	97,288KB
USERS01.DBF	2011-10-30 오후...	Microsoft Visual F...	5,128KB

데이터파일(datafile)- 물리적 구성요소

SYSAUX01.DBF, SYSTEM01.DBF

오라클 시스템 관리를 위해 만들어진 데이터파일

TEMP01.DBF

임시 데이터들을 저장하기 위한 데이터파일

USER01.DBF

사용자 계정을 위해 만들어진 데이터파일

EXAMPLE01.DBF

예제 테이블들을 저장하고 있는 데이터파일

UNDOTBS01.DBF

데이터에 문제가 발생했을 때 복구를 위한 정보

기타 물리적 구성요소

- 컨트롤 파일(control file)
 - 데이터베이스의 물리적 구조, 데이터베이스 이름, redo 로그 파일들의 위치 정보, 데이터베이스 생성 시간, 현재 로그 번호, 체크포인트 정보 등이 저장
- Redo 로그 파일
 - 데이터베이스의 변경 내역을 저장하는 파일
 - 데이터 변경 과정에서 장애가 발생하여 변경내용이 데이터베이스에 반영되지 못했을 경우 온라인 redo 로그 파일을 이용하여 복구
- 설정 파일(parameter file)
 - 데이터베이스와 데이터베이스 서버와 관련된 설정 정보들이 저장
- alert/trace 로그 파일
 - 오라클 서버 내부에서 오류가 발생할 경우 그 오류에 대한 정보나 메시지를 저장하는 파일

SQL 문의 연습

- 오라클 SQL 문을 공부한다

질의어와 SQL

- SQL: Structured Query Language
- 1974년 IBM의 System R project에서 개발된 Sequel이란 언어에 기초
- 표준 질의어로 채택되어 널리 쓰이는 관계형 질의언어
 - 1986년 ANSI와 ISO에서 표준 질의어로 채택
 - 1992년 SQL2(SQL-92) 발표
 - 2003년 SQL3발표 (최신)
- 관계 대수나 관계 해석은 이론적 배경을 제공하나 상용으로 쓰이기에는 어렵고 적절치 않음
 - SQL은 자연어와 유사하고 상대적으로 비절차적 언어 특성이 있음
 - 습득과 이용이 용이함

SQL의 구성: DDL & DML

- SQL은 크게 DDL과 DML로 구성됨
- 데이터 정의 언어 (DDL: Data Definition Language)
 - 데이터 저장 구조를 명시하는 언어
 - 테이블 스키마의 정의, 수정, 삭제
- 데이터 조작 언어 (DML: Data Manipulation Language)
 - 사용자가 데이터를 접근하고 조작할 수 있게 하는 언어
 - 레코드의 검색(search), 삽입(insert), 삭제(delete), 수정(update)

데이터 정의 언어

- 테이블 생성 (create table)
- 기본키, 외래키 설정
- 테이블 삭제 (drop table)
- 테이블 수정 (alter table)

데이터 정의 언어

- 종류
 - 테이블 생성
 - 테이블 삭제
 - 테이블 수정
- 필드의 Data type 종류

분류	표준 SQL	오라클	설명
문자	char(n)	char(n)	길이가 n byte인 고정길이 문자열 오라클의 경우 최대 2000byte까지 지정 가능
	varchar(n)	varchar2(n)	최대 길이가 n byte인 가변길이 문자열 오라클의 경우 최대 4000byte까지 지정 가능
숫자	int	int	정수형
	float	float	부동 소수
날짜 시간	date	date	년, 월, 일을 갖는 날짜형 오라클의 경우 날짜의 기본 형식은 'yy/mm/dd'이다.
	time timestamp	timestamp	년, 월, 일, 시, 분, 초를 갖는 날짜시간형

테이블 생성

- 형식

```
create table <테이블이름> (<필드리스트>)
```

- ▶ <필드리스트>는 '필드명 데이터타입'

- department 테이블을 생성하는 SQL문

(질의 1)

```
create table department
(
    dept_id          varchar2(10)    not null,
    dept_name        varchar2(14)    not null,
    office            varchar2(10)
)
```

- ▶ 키워드 **not null**은 해당 필드에 널을 허용하지 않는다는 것을 의미함

기본키, 외래키 설정

- 테이블을 생성할 기본키 역할을 하는 필드를 지정

(질의 2)

```
create table department
(
    dept_id          varchar2(10) ,
    dept_name        varchar2(20) not null,
    office           varchar2(20),
    constraint pk_department primary key(dept_id)
)
```

테이블 생성(student table)

- not null과 기본키를 지정한 student 테이블 생성 예

(질의 3)

```
create table student
(
    stu_id                varchar2(10),
    resident_id           varchar2(14) not null,
    name                  varchar2(10) not null,
    year                  int,
    address               varchar2(10),
    dept_id               varchar2(10),
    constraint pk_student primary key(stu_id)
)
```

테이블 생성(student table)

- 외래키까지 포함된 student 테이블 생성 예

(질의 3)

```
create table student
(
    stu_id                varchar2(10),
    resident_id           varchar2(14) not null ,
    name                  varchar2(10) not null,
    year                  int,
    address                varchar2(10),
    dept_id                varcahr2(10),
    constraint pk_student primary key(stu_id),
    constraint fk_student foreign key(dept_id) references
                        department(dept_id)

)
```

테이블 생성

- professor 테이블

(질의 5)

```
create table professor
```

```
(
```

```
    prof_id          varchar2(10) ,
```

```
    resident_id      varchar2(14)      not null,
```

```
    name             varchar2(10)      not null,
```

```
    dept_id          varchar2(10),
```

```
    position         varchar2(10),
```

```
    year_emp         int,
```

```
    constraint       pk_professor      primary key(prof_id),
```

```
    constraint       fk_professor      foreign key(dept_id)
```

```
    references department(dept_id)
```

```
)
```

테이블 생성

- course 테이블

(질의 6)

```
create table course
```

```
(
```

```
    course_id
```

```
    varchar2(10) ,
```

```
    title
```

```
    varchar2(14)
```

```
    not null,
```

```
    credit
```

```
    int,
```

```
    constraint
```

```
    pk_course
```

```
    primary key(course_id)
```

```
)
```

테이블 생성

- class 테이블

(질의 7)

create table class

(

class_id	varchar2(10) ,
course_id	varchar2(10),
year	int,
semester	int,
division	char(1),
prof_id	varchar2(10),
classroom	varchar2(9),
enroll	int,

constraint

constraint

constraint

)

pk_class

fk_class1

references

fk_class2

references

primary key(class_id),
foreign key(course_id)
course(course_id),
foreign key(prof_id)
professor(prof_id)

테이블 생성

- takes 테이블

(질의 8)

```
create table takes
```

```
(
```

```
    stu_id
```

```
    class_id
```

```
    grade
```

```
    constraint
```

```
    constraint
```

```
    constraint
```

```
)
```

```
    varchar2(10) ,
```

```
    varchar2(10),
```

```
    char(5),
```

```
    pk_takes
```

```
    fk_takes1
```

```
    references
```

```
    fk_takes2
```

```
    references
```

```
    primary key(stu_id, class_id),
```

```
    foreign key(stu_id)
```

```
    student(stu_id),
```

```
    foreign key(class_id)
```

```
    class(class_id)
```


기본키, 외래키 관련 사항

- 기본키
 - 테이블에는 한 개의 기본키 생성 가능(기본키 생성이 필수는 아님)
 - 기본키 : 하나 혹은 2개 이상의 필드에 대해 생성 가능
 - ▶ 보통 하나의 필드로 기본키 생성
- 기본키 생성에 따른 제약(constraint) 사항
 - 기본키 값은 반드시 서로 달라야 함 (즉, unique해야 함)
 - 따라서, 기본키 값으로 각 레코드를 유일하게 식별할 수 있음
 - 예. 주민번호. 주민번호가 같은 사람을 존재할 수 없음. 데이터 무결성과 연관을 가짐 있음

기본키, 외래키 관련 사항

- 외래키
 - 두 개의 서로 다른 테이블 사이에 존재하는 의미성을 표현하기 위해 사용됨
- (예) 두 개의 Table T1, T2가 있다고 가정하자
 - T1에 존재하는 필드 Fx(즉, T1.Fx)를 T1의 기본키라고 가정하자
 - T2를 생성할 때, T1.Fx를 참조하는 외래키를 생성할 수 있음
 - T2.Fy를 외래키로 정의: $T2.Fy \rightarrow T1.Fx$ (참조 관계)
 - 위와 같은 외래키가 존재한다면, T2에 새로운 레코드 R을 삽입하려 한다면, R.Fy의 값과 같은 T1.Fx 값에 이미 존재하고 있어야 함
 - ▶ 학생.dept_id 와 같은 학과.dept_id 값이 있어야...
 - 또한, T1에서 레코드 R을 삭제할 때, R.Fx 값을 Fy 필드값으로 가진 레코드가 T2에 존재한다면 외래키 제약 위반으로 인해 레코드 삭제 안됨

테이블 생성: 외래키/기본키

```
create table dept (  
    dept_id int primary key,  
    name varchar2(20)  
);
```

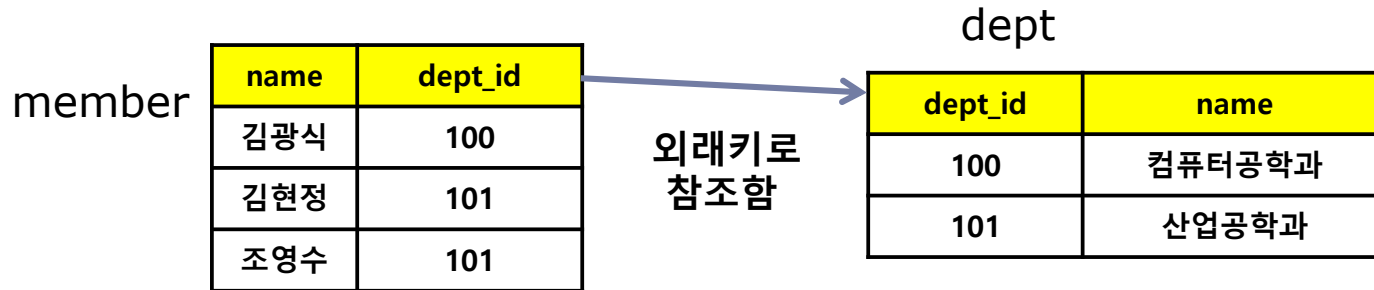
```
create table member (  
    person_name varchar2(20),  
    dept_id int references dept(dept_id)  
);
```

테이블 생성시: **member** 테이블을 생성하기 전에, **dept** 테이블이 이미 존재하고 있어야 함.

****** 외래키로 인해 참조 관계가 있는 테이블 간에는 생성/삭제 순서에 대한 제약 사항이 발생

기본키, 외래키 관련 주의사항

- 예) (주의: 필드명이 반드시 일치할 필요는 없음)



- 만일 department 테이블이 존재하지 않는 상태에서 student 테이블을 생성하려 한다면 오류발생
- member 테이블이 존재하는 동안에는 dept 테이블을 삭제할 수 없음
 - ▶ dept 테이블을 삭제하려면 member 테이블을 먼저 삭제하던지, 외래키 제약조건을 삭제해야 함

테이블 삭제

- 형식

```
drop table <테이블 이름> ;
```

- 주의

- 다른 테이블에서 외래키로 참조되는 경우에는 삭제할 수 없음
- 예)
 - ▶ class 테이블은 takes 테이블에서 외래키로 참조됨

테이블 수정

- 기존 테이블에 새로운 필드를 추가하거나 기존 필드를 삭제
- 필드 추가 형식

```
alter table <테이블이름> add <추가할 필드>
```

- 예) student 테이블에 age 필드를 추가

(질의 9)

```
alter table student  
add age int ;
```

테이블 수정

- 필드 삭제 형식

```
alter table <테이블이름> drop column <삭제할 필드> ;
```

- 예)

(질의 10)

```
alter table student  
drop column age;
```

데이터 조작 언어(DML)

- 레코드 삽입
- 레코드 수정
- 레코드 삭제
- 레코드 검색

레코드 삽입

- 형식

insert into <테이블이름> (<필드리스트>) **values** (<값 리스트>)

- <필드리스트>
 - ▶ 삽입에 사용될 테이블의 필드들
- <값 리스트>
 - ▶ <필드리스트>의 순서에 맞춰 삽입될 값
- <필드 리스트>에 나열되지 않은 필드에 대해서는 널 값이 입력됨
- <필드 리스트>를 생략할 경우 <값 리스트>에는 테이블에 존재하는 **필드의 순서에 맞춰서 값을** 나열해 줘야 함

18c 버전에서의 차이

- 테이블 생성 후 레코드 삽입
 - Insert 문 실행 시 다음 같은 오류 발생
 - ORA-01950 : 테이블스페이스 'USERS'에 대한 권한이 없습니다.
 - 이전 버전에서는 이런 오류 발생하지 않았음.
 - 18c 버전에서는 레코드가 삽입될 저장 공간(table space)에 대한 사용권한을 따로 부여해줘야 한다
- System 계정에서 아래처럼 권한을 부여해 줘야 함
 - SQL> alter user c##cs default tablespace users quota unlimited on users;

레코드 삽입

(질의 11)

```
insert into department (dept_id, dept_name, office)  
values ('920', '컴퓨터공학과', '201호') ;
```

학사 데이터베이스의 데이터 삽입 예

```
insert into department values('920', '컴퓨터공학과', '201호')
```

```
insert into department values('923', '산업공학과', '207호')
```

```
insert into department values('925', '전자공학과', '308호')
```

```
insert into student
```

```
    values('1292001', '900424-1825409', '김광식', 3, '서울', 920)
```

```
insert into student
```

```
    values('1292002', '900305-1730021', '김정현', 3, '서울', 920)
```

```
insert into student
```

```
    values('1292003', '891021-2308302', '김현정', 4, '대전', 920)
```

```
insert into student
```

```
    values('1292301', '890902-2704012', '김현정', 2, '대구', 923)
```

```
insert into student
```

```
    values('1292303', '910715-1524390', '박광수', 3, '광주', 923)
```

```
insert into student
```

```
    values('1292305', '921011-1809003', '김우주', 4, '부산', 923)
```

```
insert into student
```

```
    values('1292501', '900825-1506390', '박철수', 3, '대전', 925)
```

```
insert into student
```

```
    values('1292502', '911011-1809003', '백태성', 3, '서울', 925)
```

레코드 삽입

- 필드 이름을 나열한다면, 그 순서는 테이블을 생성할 때 지정한 순서와 일치할 필요 없음
- 예) (질의 11)과 동일한 SQL

(질의 12)

```
insert into department (office, dept_id, dept_name)  
values ('201호', '920', '컴퓨터공학과')
```

- department 테이블의 필드들 중에서 office 필드를 생략하는 경우
 - 생략된 필드에는 널이 입력

(질의 13)

```
insert into department (dept_id, dept_name)  
values ('920', '컴퓨터공학과')
```

- 단, **not null**로 설정된 필드는 널 값이 들어갈 수 없는 필드이기 때문에 **insert**문의 <필드리스트>에서 생략할 수 없음

레코드 삽입

- <필드리스트>를 사용하지 않고 데이터를 삽입하는 예

(질의 14)

**insert into
values**

department
('923', '산업공학과', '207호')

Department 테이블의 필드 명이나 순서는 어떻게 확인?

레코드 수정

- 형식

```
update <테이블이름>  
set <수정내역>  
where <조건>
```

- <수정내역>

- 대상 필드에 들어가는 값을 수정하기 위한 산술식
- ','를 이용해서 여러 필드에 대한 수정 내역을 지정

- <조건>

- 대상이 되는 레코드에 대한 조건을 기술
- 테이블의 **모든 레코드에 대해 수정하려면 where 절을 생략**

레코드 수정

- 예) student 테이블에서 모든 학생들의 학년을 하나씩 증가

(질의 16)

```
update student
set year = year + 1;
```

- 예) professor 테이블에서 '고희석' 교수의 직위를 '교수'로 수정하고 학과번호를 '923'으로 수정

(질의 17)

```
update professor
set position='교수', dept_id='923'
where name='고희석';
```


레코드 삭제

- 형식

delete from	<테이블이름>
where	<조건>

- **where**절에 지정된 조건을 만족하는 레코드를 삭제
- **where**절이 생략되면 테이블에서 모든 레코드를 삭제

- 예) professor 테이블에서 이름이 '김태석'인 교수를 삭제

(질의 18)

delete from	professor
where	name='김태석';

- **delete**문을 이용하여 테이블의 모든 레코드를 삭제하더라도 테이블은 삭제되지 않음
 - ▶ 예) delete from professor;

레코드 삽입 시 주의사항

- 외래키로 사용되는 필드에 대해 데이터를 삽입할 때
 - 참조하는 테이블의 해당 필드에 그 값을 먼저 삽입해야 함
 - 예) department 테이블이 생성되긴 했지만 아직 레코드가 삽입되지 않은 상태에서 다음질의의 실행 결과

(질의 19)

```
insert into student
values ('1292002', '900305-1730021', '김정현2', 3, '서울', '920')
```

```
SQL> insert into student
2 values('1292002', '900305-1730021', '김정현', 3, '서울', '920');
insert into student
*
1행에 오류:
ORA-02291: 무결성 제약조건(SCOTT.FK_STUDENT)이 위반되었습니다- 부모 키가 없습니다
```

레코드 수정/삭제 시 주의사항

- 외래키 필드의 값을 수정할 때
 - 외래키가 참조하는 테이블에 존재하는 값으로만 수정 가능
- 외래키로 참조되는 필드를 가지고 있는 테이블에서 레코드를 삭제할 경우에도 오류가 발생할 수 있음
 - student 테이블에서 외래키로 참조하는 department 테이블의 레코드에 대한 삭제 시도
 - 해당 dept_id를 가진 학생이 존재할 경우, 오류 발생

연습

- 6개의 테이블(학사 DB)을 생성해 보자
 - Department 테이블에 레코드를 삽입해 보자
 - 자신의 이름으로 학생 레코드를 하나 삽입하자. 학번은 적당히. 소속 학과는 컴퓨터공학과로..
 - ▶ 검색되는지 확인!
- 고재 실습내용 수행
 - 에러 발생 가능(제약 사항때문에) 적절히 수행하도록 하자
- 앞의 실습 후 "drop table"을 사용하여 6개의 테이블을 모두 삭제해 보자(
 - 삭제 순서를 미리 잘 생각한 후 수행한다
 - 테이블 간의 참조 관계를 표시해 보자.
- (테이블 삭제 가능 순서를 하나 기록:)

과제

- 사용자 계정에서 다음을 수행해 보자
- 다음 sql 문을 통해 테이블을 생성한다 (기본키 존재)
 - **Sql> create table** stu
 (stu_id int,
 stu_name varchar2(20) **not null**,
 address varchar2(20),
 constraint stu_pk **primary key** (stu_id)
);
- 위의 테이블에 아래와 같은 레코드를 삽입한다 (잘 생성되었는지 확인은 "select * from stu;"를 수행).
 - [100, '이태규', '부산'] /// 100은 정수 타입
 - [101, '최성희', '대전']
 - [102, '강만희', '부산']
- 위의 테이블에 대해서 sql 문 작성
 - Sql> '강만희' 학생의 주소를 출력

과제(continued)

- 이어서 아래의 레코드를 삽입해 보자
 - [101, '홍길동', '서울']
 - 어떤 문제가 발생하는가 살펴보자.
 - ▶ 에러 메시지 체크하여, 제출한다.
- Stu 테이블에 "tel_num"이란 필드를 추가해 보자. 타입은 varchar2(10)
 - 수정된 테이블에 레코드 하나를 추가해 보자(레코드 값은 적당히)
 - 이태규의 주소를 '인천'으로 수정해 본다
 - 이태규 레코드를 삭제한다
- 추가로 friend라는 테이블을 생성한다
 - ```
Sql> create table friend (stu_id int not null,
friend_id int references stu(stu_id));
```

  
(friend 테이블은 stu 테이블을 referencing.
  - friend 테이블에 레코드를 하나 삽입하여, 최성희는 강만희를 친구로 가짐을 표현해 보자

# 실습 3

## 6. SQL 문 연습 2

---

- 오라클 SQL 문을 공부한다



## 실습: 테이블 생성 문을 돌린다

- 게시판의 sql 문 사용 앞 수업에서 테이블을 이미 만들었다면 에러 메시지 발생할 것임
  - 테이블이 이미 생성되었는지 확인
  - Sql> desc student
- 6개의 테이블의 생성 여부를 확인(sql문 사용)
  - Sql> select table\_name from tabs;     // tabs라는 테이블의  
                                              // table\_name 필드를 출력
  - Select 문은 테이블에서 정보를 보기 위한 SQL 문
  - tabs(시스템이 자동적으로 생성해 둔 테이블이라고 생각해도 좋음. 사실은 view임)에 사용자 생성 테이블에 대한 정보가 저장됨
  - table\_name은 tabs라는 테이블(view)의 필드 명
  - 추가) "desc tabs"를 수행하여 전체 필드를 확인해 보자

# 레코드 검색

---

- SQL에서 **가장 많이 사용하고, 중요하며, 복잡함**
- 종류
  - 기본 구조
  - 재명명 연산
  - LIKE 연산자
  - 집합 연산
  - 외부조인
  - 집계 함수
  - 널의 처리
  - 중첩 질의

# 기본 구조

---

- 형식

|               |          |
|---------------|----------|
| <b>select</b> | <필드리스트>  |
| <b>from</b>   | <테이블리스트> |
| <b>where</b>  | <조건>     |

- select

- 출력할 필드 명을 나열할 수 있음

- from

- 질의할 테이블 들을 나열. 관계 대수의 카티션 프로덕트에 해당
- 예를 들어 "**from student, department**"라면 두 테이블의 카티션 프로덕트

- where

- 선택할 레코드 집합을 정하기 위한 조건을 표현
- 조건 절 기술

# 기본 구조

---

- 예 ; 학생 이름과 학생의 소속 학과를 출력

$$\pi_{name, dept\_name}(\sigma_{department.dept\_id = student.dept\_id}(student \times department))$$

(질의 20)

```
select name, dept_name
from department, student
where department.dept_id = student.dept_id ;
```

- 의미
  - department 테이블과 student 테이블을 카티션 프로덕트
  - where절에 지정된 조건식을 만족하는 레코드만 선택
    - ▶ 같은 이름의 필드가 두 개 이상의 테이블에 나타날 때 혼동을 피하기 위해 '테이블이름.필드이름'으로 표현

# 기본 구조

---

- student 테이블에서 모든 학생들의 주소를 출력(중복 주소 출력 가능)
  - 실행 해보자

(질의 21)

```
select address
from student ;
```

- 중복된 레코드를 제거하고 검색하려면 distinct를 사용
  - student 테이블에서 모든 학생들의 주소를 출력

(질의 22)

```
select distinct address
from student ;
```

# 기본 구조

- 모든 필드의 값을 출력하려 한다면 select 절에 필드를 모두 나열할 필요 없이 '\*'를 사용
- 예) student 테이블에서 모든 레코드의 모든 필드 값을 추출

(질의 23)

```
select *
from student ;
```

- Select 절에 필드 이름 외에 산술식, 상수 사용도 가능
- 예) professor 테이블에서 교수의 이름과 현재까지의 재직연수를 검색

(질의 24)

```
select name, 2022-year_emp+1
from professor
```

# 기본 구조

---

- from 절에 두 개 이상의 테이블을 포함하는 질의
  - 적절한 조건 없이 수행한 카티션 프로덕트는 의미 없는 레코드를 생성

| 이름  | 주민번호 | 주민번호 | 전화번호         |
|-----|------|------|--------------|
| 홍길동 | 9441 | 9441 | 010 411 5111 |
| 김봉구 | 8777 | 8777 | 010 511 7777 |

- 따라서 where 절에 적절한 조건을 부여해야 함. 이를 통해 의미가 있는 레코드 만을 생성
- 즉, 조인 질의나 자연 조인이 필요

# 기본 구조

---

- 예) 컴퓨터공학과 3학년 학생들의 학번을 검색

(질의 26)

```
select student.stu_id
from student, department
where student.dept_id = department.dept_id and
student.year = 3 and
department.dept_name='컴퓨터공학과' ;
```

where 절에서 “student.dept\_id = department.dept\_id”을 통해 의미 있는 레코드만이 생성되도록 함



# 출력 레코드의 순서 지정(order by)

- 검색 결과를 출력 시 필드 간 우선 순위를 부여할 수 있음
  - Select 문 맨 마지막에 다음과 같은 order by절을 추가

**order by** <필드리스트>

- 오름차순을 기본으로 하며 <필드리스트>에 여러 개의 필드를 나열할 경우 나열된 필드 순서에 맞춰 정렬
- 예) student 테이블에서 3, 4학년 학생들의 이름과 학번을 검색

(질의 27)

```
select name, stu_id
from student
where year = 3 or year = 4
order by name, stu_id ;
```

- 학생 이름(name 필드)과 학번에 대해서 오름차순 정렬
  - ▶ Name 필드 우선 오름차순 정렬
  - ▶ 학번, 이름 순서로 정렬하도록 해보자

## 레코드의 순서 지정

---

```
SQL> select name, stu_id
 2 from student
 3 where year = 3 or year = 4
 4 order by name, stu_id;
```

| NAME | STU_ID  |
|------|---------|
| 김관식  | 1292001 |
| 김우주  | 1292305 |
| 김정현  | 1292002 |
| 김현정  | 1292003 |
| 김광수  | 1292303 |
| 박철수  | 1292501 |
| 박태성  | 1292502 |

# 레코드의 순서 지정

---

- 내림차순은 해당 필드 이름 뒤에 **desc** 라는 키워드를 삽입

(질의 28)

```
select name, stu_id
from student
where year = 3 or year = 4
order by name desc, stu_id ;
```

## 재명명 연산 \*\*\*

- 테이블/필드에 대한 재명명
  - 실제 테이블에 변화가 있는 것은 아님
  - 질의를 처리하는 과정 동안만 일시적으로 사용
  - 필드 명이 동일할 때 테이블을 서로 구별하려면 테이블 재명명 필요
- 예) student 테이블과 department 테이블을 조인하여 학생들의 이름과 소속학과 이름을 검색

(질의 29)

```
select student.name, department.dept_name
from student, department
where student.dept_id = department.dept_id
```



(질의 30)

```
select s.name, d.dept_name
from student s, department d
where s.dept_id = d.dept_id
```

# 재명명 연산

---

- 동일 테이블이 두 번 사용되는 예
- 예) student 테이블에서 '김광식' 학생과 주소가 같은 학생들의 이름과 주소를 검색 --- 중요 질의

(질의 31)

```
select s2.name
from student s1, student s2
where s1.address = s2.address and s1.name = '김광식' ;
```

# 필드의 재명명

---

- 질의 실행 결과를 출력할 때 원래 필드 이름 대신 재명명된 이름으로 출력시키고자 할 때 사용
- 예) 교수들의 이름과 직위, 재직연수를 출력

(질의 32)

```
select name, position, 2012-year_emp
from professor ;
```



(질의 33)

```
select name 이름, position 직위, 2022-year_emp 재직연수
from professor ;
```

## LIKE 연산자\*\*

---

- 문자열에 대해서는 일부분만 일치하는 경우를 찾아야 할 때 사용
- '=' 연산자 대신에 'like' 연산자를 이용함
  - '='는 정확히 일치하는 경우에만 사용
- 형식

**where** <필드이름> **like** <문자열패턴>

▶ <필드이름>에 지정된 <문자열패턴>이 들어 있는지를 판단

- 문자열 패턴 종류
  - \_ : 임의의 한 개 문자를 의미한다
  - % : 임의의 여러 개 문자를 의미한다
  - 예)
    - '%서울%' : '서울'이란 단어가 포함된 문자열
    - '%서울' : '서울'이란 단어로 끝나는 문자열
    - '서울%' : '서울'이란 단어로 시작하는 문자열
    - '\_\_\_' : 정확히 세 개의 문자로 구성된 문자열
    - '\_\_\_%' : 최소한 세 개의 문자로 구성된 문자열

# LIKE 연산자

---

- student 테이블에서 김씨 성을 가진 학생들을 찾는 질의

(질의 34)

```
select *
from student
where name like '김%';
```

- student 테이블에서 여학생들만을 검색

(질의 35)

```
select *
from student
where resident_id like '%-2%';
```



# 연습

---

- 교재 73 페이지 질의 36, 37, 38에 맞는 SQL을 작성한다

**예제 2** '컴퓨터공학과' 학생들의 학번과 이름을 검색하라.

**질의 36**  $\pi_{stu\_id, name} (\sigma_{dept\_name = '컴퓨터공학과'} (department \bowtie student))$

**예제 3** 2012년 2학기에 학생들이 수강한 과목에 대해 과목번호와 학번, 성적을 검색하라.

**질의 37**  $\pi_{stu\_id, course\_id, grade} (\sigma_{year = 2012 \wedge semester = 2} (class \bowtie takes))$

**예제 4** 2012년 1학기에 '전산개론'을 가르친 교수의 이름은 무엇인가?

**질의 38**  $\pi_{name} (\sigma_{title = '전산개론' \wedge year = 2012 \wedge semester = 1} (class \bowtie couse \bowtie professor))$

# 과제

---

- Professor 테이블을 이용하여 다음을 수행해 보자
  - 부교수들의 이름과 주민번호를 함께 출력해보자
  - 부교수와 조교수들의 이름을 출력해보자
  - 부교수들의 이름과 학과를 출력해보자. name 필드명은 "교수명"으로, dept\_name 필드는 "소속학과명"으로 바꿔 출력한다(필드 재명명)
  - 70년대에 출생한 부교수들의 이름을 출력해보자(오름차순, 내림차순으로 각각)
- 조인 연산을 사용하여 다음을 수행해 보자
  - 2학년 학생 중 A학점을 받은 학생의 stu\_id와 A 학점을 받은 해당 class\_id를 함께 출력해 보시오
  - 2012년 209호 강의실에서 강의를 한 교수의 이름과 직위(position)를 출력해 보시오
  - 2012년 김광식 학생이 수업한 교과목명, 해당 수업 id(class\_id), 받은 학점을 함께 출력해 보시오

# 실습 4

## 6. SQL 문 연습 3

---

- 오라클 SQL 문을 공부한다

# 집합연산

---

- 관계대수의 집합 연산인 합집합, 교집합, 차집합에 해당하는 연산자
  - union
  - intersect
  - minus

- 형식

`<select문 1> <집합연산자> <select문 2>`

- 조건
  - <select문 1> 과 <select문 2>의 필드의 개수와 데이터 타입이 서로 같아야 함

# UNION

---

- 예) student 테이블의 학생 이름과 professor 테이블의 교수 이름을 합쳐서 출력

(질의 36)

```
select name from student
union
select name from professor ;
```

# UNION ALL

---

- **union** 연산자는 연산 결과에 중복되는 값이 들어갈 경우 한번만 출력
- 중복을 제거하고 싶지 않다면 **union** 연산자 대신 **union all** 연산자를 사용한다.
- 예) student 테이블과 professor 테이블에서 학과번호를 중복을 허용하여 출력

# INTERSECT

- 예) 컴퓨터공학과 학생들 중에서 교과목에 상관없이 학점을 'A+' 받은 학생들의 학번을 검색

(질의 37)

```
select s.stu_id
from student s, department d, takes t
where s.dept_id = d.dept_id and
 t.stu_id = s.stu_id and
 dept_name='컴퓨터공학과' and grade = 'A+' ;
```

- 발상의 전환
  - '컴퓨터공학과'에 다니는 학생들의 학번과 takes 테이블에서 학점이 'A+'인 학생들의 학번의 교집합

(질의 40)

```
select stu_id
from student s, department d
where s.dept_id = d.dept_id and dept_name='컴퓨터공학과'
intersect
select stu_id
from takes
where grade = 'A+';
```



# MINUS

---

- 예) 산업공학과 학생들 중에서 한번이라도 'A+'를 받지 못한 학생들의 학번을 검색

(질의 41)

```
select stu_id from student s, department d
where s.dept_id = d.dept_id and dept_name='산업공학과'
minus
select stu_id from takes
where grade = 'A+' ;
```

## 외부조인(outer join)

- 예) 모든 교과목들에 대해 교과목명, 학점수, 개설 년도, 개설 학기를 검색

(질의 42)

```
select title, credit, year, semester
from course, class
where course.course_id = class.course_id ;
```

| TITLE  | CREDIT | YEAR | SEMESTER |
|--------|--------|------|----------|
| 전산개론   | 3      | 2012 | 1        |
| 자료구조   | 3      | 2012 | 1        |
| 데이터베이스 | 4      | 2012 | 1        |
| 데이터베이스 | 4      | 2012 | 1        |
| 운영체제   | 3      | 2012 | 2        |
| 컴퓨터구조  | 3      | 2012 | 2        |
| 컴퓨터지식1 | 3      | 2012 | 1        |
| 컴퓨터지식2 | 3      | 2012 | 1        |
| 고리     | 2      | 2012 | 2        |
| 고리     | 2      | 2012 | 2        |

10 개의 행이 선택되었습니다.

# 외부조인(outer join)

---

- 강좌로 개설된 적이 있는 교과목에 대해서만 검색됨
  - '이산수학', '객체지향언어' 교과목들은 class 테이블에 저장되어 있지 않기 때문에 검색 결과에 포함되지 못함
  - 모든 교과목을 나열하면서도, class 정보도 출력하고 싶을 경우가 있음
  - → 외부조인 사용

# 왼쪽 외부조인(left outer join)

---

- 연산자의 왼쪽에 위치한 테이블의 각 레코드에 대해서 오른쪽 테이블에 조인 조건에 부합하는 레코드가 없을 경우에도 검색 결과에 포함
- 임의로 생성되는 레코드의 오른쪽 테이블 필드에는 널이 부여

## (질의 43)

```
select title, credit, year, semester
from course left outer join class
using (course_id) ;
```

- ' course **left outer join** class'
  - ▶ Course 테이블과 class 테이블에 대해 왼쪽 외부조인을 적용
- ' **using** (course\_id)'
  - ▶ 조인 조건이 'course.course\_id = class.course\_id ' 라는 것을 의미

참고)

select title from course **minus** select title from course c, class c2 where c.course\_id = c2.course\_id;

# 왼쪽 외부조인(left outer join)

```
SQL> select title, credit, year, semester
2 from course left outer join class
3 using (course_id);
```

| TITLE  | CREDIT | YEAR | SEMESTER |
|--------|--------|------|----------|
| 전산개론   | 3      | 2012 | 1        |
| 자료구조   | 3      | 2012 | 1        |
| 데이터베이스 | 4      | 2012 | 1        |
| 데이터베이스 | 4      | 2012 | 1        |
| 운영체제   | 3      | 2012 | 2        |
| 컴퓨터구조  | 3      | 2012 | 2        |
| 이산수학   | 4      |      |          |
| 객체지향언어 | 4      |      |          |
| 인공지능   | 3      | 2012 | 1        |
| 인공지능   | 3      | 2012 | 1        |
| 알고리즘   | 2      | 2012 | 2        |
|        |        |      |          |
| TITLE  | CREDIT | YEAR | SEMESTER |
| 알고리즘   | 2      | 2012 | 2        |

- 동일한 표현

(질의 44)

```
select title, credit, year, semester
from course, class
where course.course_id = class.course_id (+)
```

# 오른쪽 외부조인(right outer join)

---

(질의 45)

```
select title, credit, year, semester
from course right outer join class
using (course_id)
```

(질의 46)

```
select title, credit, year, semester
from course, class
where course.course_id (+) = class.course_id
```

## 완전 외부조인(full outer join)

---

- 양쪽 테이블에서 서로 일치하는 레코드가 없을 경우, 해당 레코드들도 결과 테이블에 포함시키며 나머지 필드에 대해서는 모두 널을 삽입

(질의 47)

```
select title, credit, year, semester
from course full outer join class
using (course_id)
```

## 7. SQL 집계함수 및 group by 절

---

- 오라클 SQL 문을 공부한다
- 집계 함수를 사용한다
- Group by 절을 이해한다



# 집계 함수(aggregate function)

---

- 통계연산 기능 제공
- 예)
  - 컴퓨터공학과 학생들은 모두 몇 명인가?
  - 교수들의 평균 재직연수는 몇 년인가?
- 종류
  - **count** : 데이터의 개수를 구한다.
  - **sum** : 데이터의 합을 구한다.
  - **avg** : 데이터의 평균 값을 구한다.
  - **max** : 데이터의 최대 값을 구한다.
  - **min** : 데이터의 최소 값을 구한다.
- sum, avg는 숫자형 데이터 타입의 필드에만 적용 가능 (max는?)

# COUNT

---

- 형식

**count( distinct <필드이름> )**

- 해당 필드에 값이 몇 개인지 출력
- distinct: 동일한 값이 존재한다면 제외하고 계산
- NULL은 계산에서 제외됨
- <필드이름>에는 필드 이름 대신 ' \* ' 가 사용된 경우에는 레코드의 개수를 계산
- student 테이블에서 3학년 학생이 몇 명인지 출력

**(질의 48)**

```
select count(*)
from student
where year = 3 ;
```

# COUNT

---

- student 테이블을 사용하여 dept\_id 필드에 값이 몇 개인지를 출력

(질의 49)

```
select count(dept_id)
from student
```

```
SQL> select count<dept_id>
 2 from student;

COUNT<DEPT_ID>

 8
```

# COUNT

---

- 앞의 질의는 중복된 dept\_id를 모두 카운트 했음
- distinct 키워드를 사용하면 중복되는 데이터를 제외한 개수를 리턴
  - **count**(dept\_id) → **count**(distinct dept\_id)를 사용

(질의 49)

```
select count(distinct dept_id)
from student ;
```

# COUNT

---

- 컴퓨터공학과 학생 수를 출력
  - 컴퓨터공학과 소속 학생에 해당하는 레코드의 숫자를 카운트함으로써 학생 수 계산한 경우

(질의 50)

```
select count(*)
from student s, department d
where s.dept_id = d.dept_id and d.dept_name = '컴퓨터공학과' ;
```

# SUM

---

- 형식

```
sum(<필드이름>)
```

- 예) 전체 교수들의 재직연수 합

(질의 51)

```
select sum(2012 - year_emp)
from professor ;
```

# AVG

---

- 형식

**avg**(<필드이름>)

- 예) 전체 교수의 평균 재직연수를 출력

(질의 55)

```
select avg(2012 - year_emp)
from professor
```

# SUM

- 직원 DB 사용

emp

| 필드 이름    | 설명    |
|----------|-------|
| EMPNO    | 사원번호  |
| ENAME    | 사원이름  |
| JOB      | 업무    |
| MGR      | 관리자번호 |
| HIREDATE | 입사날짜  |
| SAL      | 급여    |
| COMM     | 커미션   |
| DEPTNO   | 부서코드  |

dept

| 필드 이름  | 설명   |
|--------|------|
| DEPTNO | 부서코드 |
| DNAME  | 부서이름 |
| LOC    | 위치   |



# SUM

---

- emp 테이블에 저장된 모든 직원들의 급여 합을 출력

(질의 52)

```
select sum(sal)
from emp
```

- 업무(job 필드)가 'ANALYST'인 직원들의 급여의 합을 출력

(질의 53)

```
select sum(sal)
from emp
where job = 'ANALYST'
```

- 부서 이름이 'RESEARCH'인 직원들의 급여의 합을 출력

(질의 54)

```
select sum(sal)
from emp e, dept d
where e.deptno = d.deptno and dname = 'RESEARCH'
```

# MIN, MAX

---

- 형식

```
max(<필드이름>)
min(<필드이름>)
```

- 부서 이름이 'ACCOUNTING'인 직원들 중에서 최대 급여가 얼마인지 출력

(질의 56)

```
select max(sal)
from emp e, dept d
where e.deptno = d.deptno and dname = 'ACCOUNTING' ;
```

- 컴퓨터공학과 교수 중 가장 최근 임용 년도는?
  - 질의를 작성해 보시오

# GROUP BY

- **select** 절에 집계 함수가 사용될 경우, 다른 필드는 **select** 절에 나올 수 없음
- 다음 질의는 오류 \*\*\*
  - 왜 아래와 같은 질의는 의미적으로 성립하지 않을까?

```
SQL> select ename, max(sal)
 2 from emp;
select ename, max(sal)
 *
```

1행에 오류:  
ORA-00937: 단일 그룹의 그룹 함수가 아닙니다

- **GROUP BY**를 이용하면 그룹별로 집계함수 적용 가능
  - 예) '학과별 학생 수', '부서별 최대 급여'
  - 이런 경우 select 문에 집계 함수 적용된 필드 외의 다른 필드가 나올 수 있음

# GROUP BY

---

- 형식

**group by** <필드리스트>

- group by 절은 select문에서 **where 절 다음에 위치**
- group by 에 지정된 필드의 값이 같은 레코드 집합에 대해서, 집계 함수를 적용할 수 있음

- 예) student 테이블에서 학과번호(dept\_id 필드)별로 레코드의 개수를 출력

**(질의 57)**

```
select dept_id, count(*)
from student where ...
group by dept_id ;
```

# GROUP BY

---

```
SQL> select dept_id, count(*)
2 from student
3 group by dept_id;
```

| DEPT_ID | COUNT(*) |
|---------|----------|
| 925     | 2        |
| 920     | 3        |
| 923     | 3        |

group by 절에 사용된 필드는 select 절에 집계 함수가 사용되고 있더라도, 함께 출력될 수 있음

# 과제

- 다음과 같은 테이블 A, B를 만들고 레코드를 삽입(name\_a, name\_b 필드는 varchar2(20) 자료형, id는 int 자료형으로 정의한다)

| name_a | id  |
|--------|-----|
| James  | 300 |
| John   | 400 |
| Joon   | 500 |
| Jungsu | 600 |

Table A

| name_b | id  |
|--------|-----|
| Duck   | 100 |
| Dongho | 200 |
| Duksu  | 300 |
| Dooli  | 500 |

Table B

- 위에서 만든 테이블 A, B에 대해서 left outer join, right outer join, full outer join을 수행해 본다(id 필드 이용).
  - 출력 필드는 name\_a, name\_b, id 순으로.
  - 수행 전에 outer join의 결과를 미리 예상하여 적어본다(꼭!!)
- 과제 제출 시 outer join 검색어와 결과만 캡처(테이블 생성 및 레코드 삽입 문은 제출 필요 없음)

# 과제(계속)

---

- 다음의 sql 문을 작성해 본다
  - 교수 테이블에서 "부교수"의 숫자를 출력해보자
  - Course 테이블을 사용하여 전체 과목 수를 출력해보자
  - 2023년을 기준으로, 부교수 수와 그들의 평균 근무년수를 출력해보자(출력시 컬럼 명을 "평균근무년수", "부교수명수"로 한다)
  - 2012년 개설된 class 중에서 최대 등록학생수와 최소 등록학생수를 출력해 보자
  - 교수 직위(position)와 각 직위에 속하는 교수 수를 출력해 보자(group by 사용) - 교재 58참조

# 실습 5



## 7. SQL 집계함수 및 group by 절

---

- Group by 절을 이해한다
  - 난이도가 높음

# GROUP BY -- 전 주 수업에 연속

---

- 형식

**group by** <필드리스트>

- group by 절은 select문에서 **where 절 다음에 위치**
- group by 에 지정된 필드의 값이 같은 레코드 집합에 대해서, 집계 함수를 적용할 수 있음

- 예) student 테이블에서 학과번호(dept\_id 필드)별로 레코드의 개수를 출력

**(질의 57)**

```
select dept_id, count(*)
from student where ...
group by dept_id ;
```

# GROUP BY

- 학과 이름과 해당 학과에 소속된 학생 수를 출력
  - (질의 57)을 다소 수정

(질의 58)

```
select dept_name, count(*)
from student s, department d
where s.dept_id = d.dept_id
group by dept_name ;
```

| DEPT_NAME | COUNT(*) |
|-----------|----------|
| 전자공학과     | 2        |
| 컴퓨터공학과    | 3        |
| 산업공학과     | 3        |

# SUM

- 직원 DB 사용

emp

| 필드 이름    | 설명    |
|----------|-------|
| EMPNO    | 사원번호  |
| ENAME    | 사원이름  |
| JOB      | 업무    |
| MGR      | 관리자번호 |
| HIREDATE | 입사날짜  |
| SAL      | 급여    |
| COMM     | 커미션   |
| DEPTNO   | 부서코드  |

dept

| 필드 이름  | 설명   |
|--------|------|
| DEPTNO | 부서코드 |
| DNAME  | 부서이름 |
| LOC    | 위치   |

# GROUP BY

---

- 예) emp, dept 테이블에서 부서별 직원수, 평균급여, 최대급여, 최소급여를 출력

(질의 59)

```
select dname, count(*), avg(sal), max(sal), min(sal)
from emp e, dept d
where e.deptno = d.deptno
group by dname ;
```

# GROUP BY

- 학과별 교수 숫자와 평균 재직연수, 최대 재직연수를 출력 \*\*\*

(질의 60)

```
select dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
from professor p, department d
where p.dept_id = d.dept_id
group by dept_name ;
```

| DEPT_NAME | COUNT(*) | AUG<2012-YEAR_EMP> | MAX<2012-YEAR_EMP> |
|-----------|----------|--------------------|--------------------|
| 전자공학과     | 2        | 6                  | 7                  |
| 컴퓨터공학과    | 2        | 12                 | 15                 |
| 산업공학과     | 2        | 10                 | 13                 |

- 다음의 sql 문을 작성해 본다(group by 사용)
  - 학점과 학점별 학생 수를 출력해 보자
  - 교수명과 해당 교수가 강의한 전체 학점을 출력해 보자

```
select grade, count(*) 학생수 from takes group by grade order by grade;
```

```
select p.name, sum(c2.credit) from professor p, class c, course c2 where p.prof_id = c.prof_id and c.course_id =
c2.course_id group by p.name order by p.name;
```

# HAVING

- 각 그룹에 대한 조건을 명시할 때 사용
- 예) 평균 재직연수가 10년 이상인 학과에 대해서만, 학과명, 교수 숫자, 평균 재직연수, 최대 재직연수를 출력
- 주의
  - Group에 대한 조건은 where 절에 사용하지 못함
  - HAVING 절을 이용해야 함

## (질의 61)

```
Select d.dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
From professor p, department d
Where p.dept_id = d.dept_id and avg(2012 - year_emp) >= 10
Group by d.dept_name;
```

- 형식

**having** <집계함수 조건>



# HAVING

---

- 앞의 질의를 맞게 수정: having 절을 이용하여 다시 작성

## (질의 62)

```
select dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
from professor p, department d
where p.dept_id = d.dept_id
group by dept_name
having avg(2012 - year_emp) >= 10 ;
```

# HAVING

---

- 직원 숫자가 5명 이상인 부서에 대해서 부서별 직원수, 평균급여, 최대급여, 최소급여를 출력

## (질의 63)

```
select dname, count(*), avg(sal), max(sal), min(sal)
from emp e, dept d
where e.deptno = d.deptno
group by dname
having count(*) >= 5 ;
```

- (연습) 소속 학생수가 3명 이상인 학과의 학과명과 학생수를 출력한다
- (연습) 소속 교수 수가 2명 이상인 학과의 학과명, 평균 재직년수, 최대 재직년수를 출력한다

# HAVING

---

- where절과 having절, group by절이 사용될 때 아래와 같은 순서로 질의가 처리된다고 생각할 수 있다
  1. where절에 명시된 조건을 만족하는 레코드들을 추린다
  2. Group by 절에 명시된 필드에 대해 값이 같은 레코드 집합을 생성
  3. Having 절에서 위에 생성한 레코드 집합에 집계 함수를 적용. 조건을 만족시키는 집합에 대해서만 출력 수행

# 과제

---

- 다음의 sql 문을 작성해 본다(group by 및 having 사용)
  - 수강한 강의에서 B학점을 2개 이상 받은 학생의 stu\_id를 출력해보자
  - 수강한 강의에서 B학점을 2개 이상 받은 학생의 이름을 출력해보자(동명이인 없음)
  - 2012년도에 수업을 2개 이상 맡았던 교수명을 출력해보자(교수도 동명이인 없음)

## 8. 중첩질의의와 View의 사용

---

- 중첩 질의를 이해한다

# 중첩 질의(nested query)

---

- SQL문을 다른 SQL문 안에 중첩하여 사용하는 질의
  - 복잡한 질의를 쉽게 표현할 수 있는 수단을 제공
- 내부질의(inner query), 부질의(subquery)
  - 내부에 포함된 SQL문
- 외부질의(outer query)
  - 부 질의를 내부적으로 갖는 SQL문
- 부 질의는 외부 질의의 from 절이나 where 절에 위치
- 종류
  - in, not in
  - =some, <=some, <some, >some, >=some, <>some (some 대신 any를 사용해도 됨)
  - =all, <=all, <all, >all, >=all, <>all
  - exists, not exists

# IN, NOT IN

---

- 예) '301호' 강의실에서 개설된 강좌의 과목명을 출력

(질의 66)

```
select title
from course
where course_id in
 (select distinct course_id
 from class
 where classroom = '301호')
```

- 부 질의
  - ▶ 키워드 **in** 뒤에 나오는 SQL문으로서 class 테이블에서 강의실이 '301호'인 교과목 번호를 검색
- 외부 질의
  - ▶ course 테이블에서 course\_id 필드의 값이 부 질의의 검색 결과에 포함되는 경우(**in**)에만 과목명을 출력

# IN, NOT IN

---

- 동일한 표현

(질의 67)

```
select distinct title
from course c1, class c2
where c1.course_id = c2.course_id and
classroom = '301호' ;
```



# IN, NOT IN

---

- 예) 2012년 2학기에 개설되지 않은 과목명을 검색

(질의 68)

**select**  
**from**  
**where**

title  
course  
course\_id

**not in**

(**select distinct** course\_id  
**from** class

**where** year = 2012 **and** semester = 2);

# SOME, ALL

---

- =some
  - 지정된 필드의 값이 부 질의 검색 결과에 존재하는 임의의 값과 같은지를 나타낼 때 사용
  - in과 같은 의미
- <=some
  - 부 질의의 검색 결과에 존재하는 임의의 값보다 작거나 같은지를 나타낼 때 사용
- =all
  - 지정된 필드의 값이 부 질의 검색 결과에 포함된 모든 값과 같은지를 판단
- <=all
  - 지정된 필드의 값이 부 질의 검색 결과에 포함된 모든 값보다 작거나 같은지를 판단

# 간단한 연습

---

- 가장 빨리 학교에 부임한 교수 이름과 부임 년도를 출력
  - "<= all" 사용
  - "in" 사용

```
select name, year_emp from professor where year_emp <= all (select year_emp from professor);
```

## 중첩문을 이용 예

---

- 컴퓨터학과 학생들의 year 필드를 1씩 증가 시킨다

```
SQL> commit;
```

```
SQL> select name, year from student;
```

```
SQL> update student s set s.year = s.year+1 where s.dept_id in
 (select dept_id
 from department
 where dept_name = '컴퓨터공학과');
```

```
SQL> select name, year from student;
```

```
SQL> rollback;
```

```
SQL> select name, year from student;
```

# SOME, ALL

---

- 예) 가장 많은 수강 인원을 가진 강좌를 검색

(질의 69)

```
select c1.course_id, title, year, semester, prof_id
from class c1, course c2
where c1.course_id = c2.course_id and enroll >= all
 (select enroll from class);
```

# EXISTS, NOT EXISTS

---

- 부 질의 검색 결과에 최소한 하나 이상의 레코드가 존재하는지의 여부를 표현
- exists
  - 최소한 한 개의 레코드가 존재하면 참이 되고 그렇지 않으면 거짓
- not exists
  - 부 질의의 결과에 레코드가 하나도 없으면 참이 되고 하나라도 존재하면 거짓
- 예) '301호' 강의실에서 개설된 강좌의 과목명을 출력

(질의 70)

```
select title
from course
where exists
(select *
 from class
 where classroom = '301호' and
 course.course_id = class.course_id)
```

# EXISTS, NOT EXISTS

---

- (질의 68)을 not exists로 표현 가능

(질의 71)

```
select title
from course
where not exists
(select *
 from class
 where year = 2012 and
 semester = 2 and
 course.course_id = class.course_id)
```

# 실습 6



## 8. 중첩질의의와 View의 사용

---

- View에 대해서 학습한다

# 중첩 질의 연습

---

- 최소 1개의 B학점을 받은 학생의 stu\_id (having 절 사용시?)
- 최소 1개의 B학점을 받은 학생의 stu\_id와 이름
- 서울에 주소를 둔 '전자공학과' 학생 레코드를 삭제
- 학생수가 가장 많은 학과의 dept\_id 출력 (join 불필요)
- 아래 질의의 의미는/
  - select count(\*) from takes where grade = ' A ' ;
  - select count(\*) from takes where grade = ' A ' group by class\_id ;
- A학점이 가장 많은 class id 찾기

```
select class_id from takes where grade = 'A' group by class_id having
count(*) >=all (select count(*) from takes where grade = 'A' group by class_id);
```

# 뷰 (view)

---

- 기존 테이블들로부터 생성되는 **가상의 테이블**
  - 인식되기는 일반 테이블과 같이 보이지만 실제 물리적으로 존재하는 것은 아님
  - **뷰를 사용해서 사용자 별로 세밀한 데이터 통제가 가능: 보안 상의 장점 존재 \*\***
  - 다소 복잡한 질의를 통해 얻어지는 데이터를 view로 만들어 놓을 수 있음. 이를 통해 조인 질의에 익숙하지 않은 사용자들도 쉽게 원하는 데이터를 볼 수 있음
  - 실제 저장된 것이 아니기에 DBMS가 해당 SQL 문을 수행시켜 데이터를 얻음

# 뷰 생성

---

- 생성된 뷰는 테이블과 동등하게 취급
- 형식

```
create or replace view <뷰 이름> as
 <select문>
```

- **or replace** 키워드를 추가하면 <뷰 이름>과 같은 뷰가 이미 존재하는 경우 기존의 뷰를 삭제하고 생성
- <select문>
  - ▶ 뷰 생성에 사용될 **select문**

# 뷰 생성

---

- 대부분의 DBMS에서는 사용자 계정에는 뷰 생성 권한이 부여되지 않음
- 관리자 계정이 아닌 사용자 계정은 관련 권한 필요
- 오라클 sql문

```
Sql> grant create view to <사용자 계정>
```

# 뷰 생성

---

- 예) takes 테이블에서 grade 필드를 제외한 나머지 필드만으로 구성된 뷰를 생성

(질의 72)

```
create or replace view v_takes as
select stu_id, class_id
from takes ;
```

- 예) student 테이블에서 컴퓨터공학과 학생들 레코드만 추출하여 뷰를 생성

(질의 73)

```
create or replace view cs_student as
select s.stu_id, s.resident_id, s.name, s.year, s.address, s.dept_id
from student s, department d
where s.dept_id = d.dept_id and
d.dept_name = '컴퓨터공학과' ;
```

# 뷰 사용

---

- 뷰에 대해서 **insert, update, delete**문을 실행
- 예) v\_takes 뷰에 대해 레코드를 삽입

(질의 74)

```
insert into v_takes
values ('1292502', 'C101-01') ;
```

- v\_takes 뷰에 포함되지 않은 grade 필드에는 널이 삽입
  - ▶ what????
- 아래 질의를 통해 확인 바람
- Sql> select \* from takes where stu\_id = '1292502'

# 뷰 삭제 및 삽입이 안되는 뷰 만들기

- 형식 `drop view <뷰이름>`
- 읽기 전용 뷰
  - 뷰는 갱신을 위한 용도가 아니다. 검색용으로 만드는 것이 일반적
  - 따라서 뷰를 생성할 때 insert, update, delete문과 같은 데이터 조작 언어의 사용을 불가능하게 하는 것이 좋음
  - 뷰 생성시 "**with read only**" 키워드를 추가 -- 아래 문 수행

```
SQL> create or replace view v_takes as
 2 select stu_id, class_id
 3 from takes
 4 with read only;
```

뷰가 생성되었습니다.

```
SQL> insert into v_takes
 2 values ('1292502', 'C101-01');
insert into v_takes
```

\*

1행에 오류:

ORA-42399: 읽기 전용 뷰에서는 DML 작업을 수행할 수 없습니다.



# 무결성 제약과 권한의 부여

---

- 보안과 무결성

## 제 5 장 무결성과 보안

- 무결성 제약
- 데이터베이스 보안
- 오라클에서의 무결성과 보안

# 무결성 제약

---

- 무결성 제약(integrity constraint)
  - 데이터베이스에 저장된 데이터는 실제 세계에 존재하는 정보들을 모순 없이 반영 – 데이터 삽입이나 갱신 시 제약 필요
  - 데이터베이스 설계자나 프로그래머는 의미 있는 제약을 DB에 반영시켜야 함
- 무결성 제약의 예
  - 학생은 하나의 학과에 소속된다
  - 학생의 나이는 필드는 언제나 양수이고 100을 넘지 않는다
  - 테이블 생성시에 이런 제약을 함께 정의할 수 있으면 좋을 것임
  - 학생은 한 학기에 20학점 이상 수강할 수 없다 ← 이런 제약은 DBMS에서 제공하는 기본적인 제약 정의 방식으로 구현하기 어려움

## 무결성 제약의 유형 \*\*

| 분류              | 의미                               | 무결성 제약의<br>종류 및 방식   |
|-----------------|----------------------------------|----------------------|
| 기본적 무결성 제약      | 관계형 데이터 모델에서 정의한 무결성 제약          | 기본키 무결성 제약           |
|                 |                                  | 참조 무결성 제약            |
| 테이블의 무결성 제약     | 테이블을 정의하거나 변경 과정에서 설정 가능한 무결성 제약 | NOT NULL             |
|                 |                                  | UNIQUE               |
|                 |                                  | CHECK                |
|                 |                                  | DEFAULT              |
| 복잡한 무결성 제약(추가적) | 위의 방법으로 정의될 수 없는 복잡한 형태의 무결성 제약  | 주장(assertion)        |
|                 |                                  | 트리거(trigger)         |
|                 |                                  | 프로그래밍을 이용한 무결성 제약 설정 |

# 오라클에서의 무결성 제약 : 테이블 생성 예

(질의 49)

```
create table student
```

```
(
```

```
 stu_id varchar2(10),
```

```
 resident_id varchar2(14) not null,
```

```
 name varchar2(10),
```

```
 year int default 1,
```

```
 address varchar2(10),
```

```
 dept_id varchar2(10),
```

```
 constraint pk_student primary key (stu_id),
```

```
 constraint fk_dept foreign key (dept_id)
```

```
 references department (dept_id),
```

```
 constraint uc_rid unique (resident_id),
```

```
 constraint chk_year check (year >= 1 and year <= 4)
```

```
)
```

필드 명 옆에 제약을 부여할 수도 있음.

# 기본키 무결성 제약

---

- primary key integrity constraint
- 테이블에서 레코드들이 반드시 유일하게 식별될 수 있어야 한다는 조건

## 정의: 기본키 무결성 제약

기본키는 널 값을 가질 수 없으며 기본키의 값이 동일한 레코드가 하나의 테이블에 동시에 두 개 이상 존재할 수 없다.

dbms는 어떻게 이 기본적 제약을 유지할까?

# 기본키 무결성 제약

- 설정 방법

- 형식

**constraint** <제약식 명> **primary key** (<필드리스트>)

- ▶ <제약식명>

- 기본키를 정의하는 제약식에 주어진 이름이고

- ▶ <필드리스트>

- 기본키로 정의할 필드들의 리스트

- 예)

**(질의 1)**

```
create table student
(
 stu_id varchar2(10),
 resident_id varchar2(14),
 name varchar2(10),
 year int,
 address varchar2(10),
 dept_id varchar2(10),
 constraint pk_student primary key (stu_id)
)
```

# 기본키 무결성 제약

---

- 테이블 생성 당시에 기본키를 설정하지 않았다면?
  - 나중에 **alter table**문을 이용하여 **기본키 제약을 추가할** 수 있음

(질의 4)

```
alter table student
add constraint pk_student primary key (stu_id) ;
```

- 생성되어 있던 **제약 사항을 삭제할** 수도 있음

(질의 5)

```
alter table student
drop constraint pk_student ;
```

# 참조 무결성 제약

---

- 한 테이블의 레코드가 다른 테이블을 참조
  - 외래키로 우리가 배웠음
  - 실제 존재하지 않는 잘못된 값에 의존하는 레코드가 저장되지 않도록 하는 제약

- 형식

|                                  |                   |                                             |
|----------------------------------|-------------------|---------------------------------------------|
| <b>constraint<br/>references</b> | <제약식명><br><테이블이름> | <b>foreign key</b> (<필드리스트1>)<br>(<필드리스트2>) |
|----------------------------------|-------------------|---------------------------------------------|

- ▶ <제약식명>: 외래키를 정의하는 제약식에 주어진 이름
- ▶ <필드리스트1> : 외래키로 정의하는 필드들의 리스트
- ▶ <테이블이름> : 참조 대상인 테이블의 이름
- ▶ <필드리스트2> : <테이블이름>의 기본키



## 참조 무결성 제약

---

- 외래키가 하나의 필드로만 구성되거나 제약식에 이름을 부여하지 않을 때 다음과 같이 정의 가능
- 참조 무결성 제약은 외래키 정의에 의해 DBMS에서 자동적으로 검증
  - DBMS는 이 조건을 위반하게 되는 연산의 실행을 거부

### (질의 7)

```
create table student
(
 stu_id varchar2(10),
 resident_id varchar2(14),
 name varchar2(10),
 year int,
 address varchar2(10),
 dept_id varchar2(10) foreign key
 department (dept_id)
 references
)
```

# 참조 무결성 제약

---

- 테이블 생성 당시에 외래키를 설정하지 않았다면?
  - **alter table**문을 이용하여 외래키를 별도로 설정
  - 예)

(질의 8)

|                       |                   |                              |
|-----------------------|-------------------|------------------------------|
| <b>alter table</b>    | student           |                              |
| <b>add constraint</b> | fk_dept           | <b>foreign key</b> (dept_id) |
|                       | <b>references</b> | department (dept_id)         |

- 외래키 삭제
  - 예)

(질의 9)

|                        |         |
|------------------------|---------|
| <b>alter table</b>     | student |
| <b>drop constraint</b> | fk_dept |

# 테이블의 무결성 제약

---

- 테이블 스키마를 정의할 때 지켜야 하는 무결성 제약
  - **not null**
  - **unique**
  - **check**
  - **default**

# NOT NULL

---

- 특정 필드에 대해서 널 값의 입력을 허용하지 않아야 되는 경우
- 예)

(질의 10)

```
create table student
(
 stu_id varchar2(10),
 resident_id varchar2(14) not null,
 name varchar2(10),
 year int,
 address varchar2(10),
 dept_id varchar2(10)
)
```

- 기본키로 정의된 필드에 대해서는 명시적으로 **not null** 조건을 설정하지 않아도 됨

# UNIQUE

---

- 해당 필드가 테이블 내에서 중복된 값을 갖지 않고 유일하게 식별되도록 하는 제약 조건
- 형식 `constraint <제약식명> unique (<필드리스트>)`
  - <제약식명> : 제약식의 이름
  - <필드리스트> : **unique**을 설정할 필드들의 리스트

# UNIQUE

---

- 예)

(질의 11)

```
create table student
(
 stu_id varchar2(10),
 resident_id varchar2(14),
 name varchar2(10),
 year int,
 address varchar2(10),
 dept_id varchar2(10),
 constraint uc_rid unique (resident_id)
)
```

# UNIQUE

- 예) 두 개 이상의 필드에 동시에 설정 가능

## (질의 12)

```
create table student
```

```
(
```

```
 stu_id varchar2(10),
```

```
 resident_id varchar2(14),
```

```
 family_name varchar2(10),
```

```
 given_name varchar2(10),
```

```
 Year int,
```

```
 Address varchar2(10),
```

```
 dept_id varchar2(10)
```

```
 constraint
```

```
 uc_name
```

```
 unique (family_name, first_name)
```

```
)
```

- 두 필드 각각에 대해서 유일해야 한다는 조건이 아니고 이름과 성이 동시에 같은 경우에 대해 중복을 허용하지 않음

# UNIQUE

---

- 예) 제약식에 이름을 부여하지 않거나 하나의 필드에 대해서만 설정할 경우 다음과 같은 표현이 가능

## (질의 13)

```
create table student
(
 stu_id varchar2(10),
 resident_id varchar2(14) unique,
 name varchar2(10),
 year int,
 address varchar2(10),
 dept_id varchar2(10)
)
```



# CHECK

---

- 도메인 제약(domain constraint)
  - 각 필드의 값은 정의된 도메인에 속한 값만 허용하는 성질
  - CHECK
    - ▶ 필드를 정의할 때 주어진 데이터 타입 이외에도 좀 더 세부적으로 허용할 수 있는 값의 범위를 지정
- 형식

**constraint** <제약식명> **check** (<조건식>)

- ▶ <조건식> : 만족해야할 필드들의 조건

# CHECK

---

- 예)

## (질의 16)

```
create table student
```

```
(
```

```
 stu_id varchar2(10),
```

```
 resident_id varchar2(14),
```

```
 name varchar2(10),
```

```
 year int,
```

```
 address varchar2(10),
```

```
 dept_id varchar2(10),
```

```
 constraint chk_year check (year >= 1 and year <= 4)
```

```
)
```

- 예)

)

## CHECK \*\*

---

- 각 필드에 대한 조건식을 분리하여 명시 가능

### (질의 18)

```
create table student
```

```
(
```

```
 stu_id varchar2(10),
```

```
 resident_id varchar2(14),
```

```
 name varchar2(10),
```

```
 year int
```

```
 address varchar2(10)
```

```
 dept_id varchar2(10)
```

```
)
```

```
check (year >= 1 and year <= 4),
```

```
check (address in ('서울', '부산'));
```

# CHECK

---

- 테이블 수정을 통해 check 제약을 추가

## (질의 19)

```
alter table student
add constraint chk_year check (year >= 1 and year <= 4) ;
```

→ student를 레코드를 하나 추가해 보자(에러 체크)

```
insert into student(stu_id, year) values ('1200001', 5);
```

check를 해제 → student를 레코드를 하나 추가해 보자(에러 체크)

## (질의 20)

```
alter table student
drop constraint chk_year ;
```

삭제 후 다시 수행:

```
insert into student(stu_id, year) values ('1200001', 5);
```

삽입된 레코드 검색: 작성해 보자(dept\_id 필드에 조건을 걸어서...)

# DEFAULT

---

- 레코드를 삽입할 때, 필드에 대한 값이 정해지지 않았을 경우 사전에 정해놓은 값으로 입력하도록 지정
  - 널 값 대신에 지정된 값이 자동적으로 입력

- 예)

(질의 21)

```
create table student
(
 stu_id varchar2(10),
 resident_id varchar2(14),
 name varchar2(10),
 year int default 1,
 address varchar2(10),
 dept_id varchar2(10)
)
```

# DEFAULT

---

- default를 별도로 설정

(질의 22)

```
alter table student
alter column year set default 1
```

- default 해제

(질의 23)

```
alter table student
alter column year drop default
```

## 참고(DEFAULT) x

---

- 오라클은 default에 대한 별도의 설정 및 해제에 SQL을 표준을 따르지 않음
- 오라클에서는 (질의 22)와 (질의 23)대신에 다음과 같은 형식을 사용해야 함

```
alter table student
modify (year int default 1)
```

```
alter table student
modify (year int default null)
```



# 무결성 제약 설정의 유의점

---

- 지나치게 많은 제약조건들이 존재할 경우 예외적인 데이터가 발생할 때 문제가 발생할 수 있음
  - 예) 외국인의 경우 주민등록번호가 없으므로 student 테이블의 resident\_id에 값을 입력할 수 없음
  - 그러나 이 필드에는 **not null** 조건이 있으므로 삽입이 불가능
- 각 레코드에 대한 삽입, 삭제, 수정 연산을 할 때 정의된 무결성 제약들이 모두 만족하는지 검증함
  - 무결성 제약의 만족 여부를 검사하는 부담이 가중되어 전체적인 성능이 떨어지는 문제가 발생할 수 있음

# 기타 무결성 제약

---

- 지금까지의 방법만으로 모든 무결성 제약들을 표현할 수 없음
- 현실 세계의 제약들은 다양하고 복잡함
- 기타 무결성 제약 방법
  - 주장(Assertion)
  - 트리거(Trigger)
  - 응용 프로그램을 이용한 제약 준수

# 응용 프로그램에서의 무결성 제약

---

- 데이터베이스 사용자가 응용 프로그램에서 직접 무결성을 보장하는 코드를 삽입
- **check**를 사용하지 않는다면? (year에 대해서)
  - 새로운 레코드를 삽입하는 프로그램에서 **insert**나 **update**문을 실행하기 전에 year 필드의 값이 1에서 4까지의 정수인지를 판단
  - 만족하지 못할 경우 에러 처리를 하는 코드를 삽입
- 번거롭고 어렵지만, DBMS가 제공하는 기능으로 해결하지 못할 경우 응용 프로그램에서 무결성을 만족시켜야 함

# 데이터베이스 보안

---

- 데이터베이스 관리자(DBA)는 외부의 권한을 갖지 못한 사용자로부터 데이터를 안전하게 관리할 의무가 있음
- DBMS에서 제공하는 보안
  - 허가 받지 않은 사용자, 즉, 권한이 없는 사용자로부터 데이터의 접근을 사전에 차단

# 과제 (view)

---

- 교재 194 페이지 (질의 72) 수행
  - "select \* from v\_takes;"를 수행해 본다
  - Insert 문을 사용하여 v\_takes에 레코드를 하나 삽입해 보자(학생 id와 class id는 자신이 적절히 새로 선택)
  - 삽입 시 constrain violation에 주의한다
  - 레코드 삽입 후 "select \* from v\_takes; "를 수행해 본다
  - "select \* from takes where grade is null; "를 수행시켜 본다(의미를 생각해 보길 ...)
  - 질의를 만들어 본다) "grade가 널값인 학생의 이름과 stu\_id를 출력"
- 뷰는 기본적으로 정보를 보는 용도로 사용되는 것이지 갱신 용은 아님
  - 뷰를 생성할 때 insert, update, delete와 같은 데이터 조작 언어의 사용을 제한하는 것이 좋을 때가 많음
  - 이때, "with read only" 옵션을 사용 (197 페이지 참조)
  - v2\_takes를 "with read only" 옵션을 부여하여 생성한다. 앞에서 수행했던 레코드 삽입 문을 다시 수행해 보자 (오류를 확인)
  - 생성된 두 개의 view를 삭제한다

# 실습 7

# 데이터베이스에서의 사용권한 \*\*

---

- 사용자가 특정 객체에 대해 특정 연산을 실행할 수 있는 권리
- 특정 객체
  - 테이블, 뷰, 필드 등 데이터베이스의 구성 요소
- 권한 제어가 가능한 연산의 종류
  - ▶ 데이터 접근관련 연산(DML)
    - SQL의 **select, insert, delete, update** 등
  - ▶ 스키마 관련 연산(DDL)
    - **create table, alter table, drop table, create index** 등

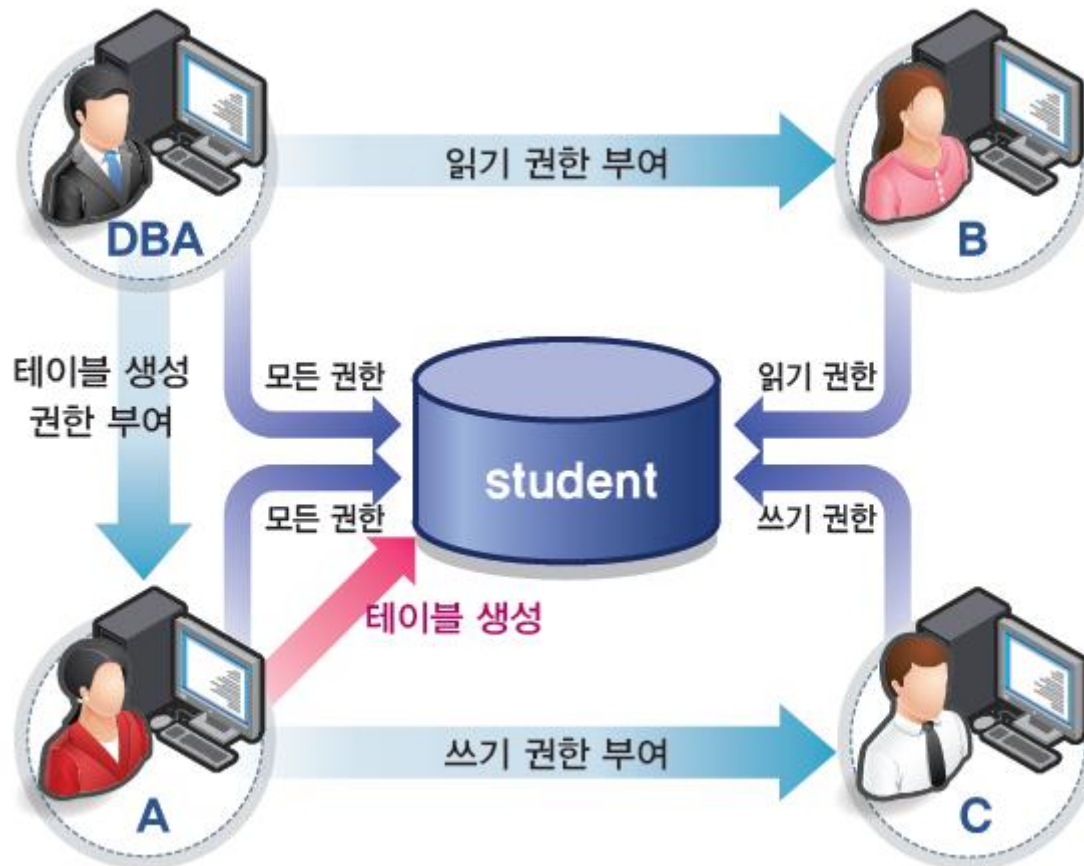
# 권한에 따른 사용자 분류

---

- 데이터베이스 관리자(DBA)
  - DBMS내의 모든 객체에 대해 모든 권한
  - 객체에 대한 연산, 객체의 제거와 변경을 포함
  - 다른 사용자에게 해당 객체에 대한 권한을 부여하거나 회수
- 객체 소유자(owner)
  - 특정 사용자가 객체를 생성한 사용자
  - 생성한 객체에 대해 모든 권한(해당 객체에 대한 권한의 부여나 회수)
- 기타 사용자
  - 기본적으로 객체에 대한 일체의 사용권한이 없음
  - 다만 데이터베이스 관리자나 객체 소유자로부터 일부 또는 모든 권한을 별도로 부여받을 수 있음



# 권한 부여의 예



# 오라클의 대표적인 시스템 권한

| 대상       | 시스템 권한            | 내용                  |
|----------|-------------------|---------------------|
| 사용자      | create user       | 사용자 계정을 생성할 수 있는 권한 |
|          | drop user         | 사용자 계정을 삭제하는 권한     |
| 세션       | create session    | 데이터베이스의 접속(로그인) 권한  |
| 테이블      | create table      | 테이블을 생성할 수 있는 권한    |
|          | drop any table    | 임의의 테이블을 삭제하는 권한    |
|          | select any table  | 임의의 테이블에 대한 읽기 권한   |
| 뷰        | create view       | 뷰를 생성할 수 있는 권한      |
|          | drop any view     | 임의의 뷰에 대한 삭제 권한     |
| 테이블 스페이스 | create tablespace | 테이블 스페이스를 생성하는 권한   |
|          | drop tablespace   | 테이블 스페이스에 대한 삭제 권한  |
| 롤        | create role       | 롤을 생성할 수 있는 권한      |
|          | drop any role     | 임의의 롤을 삭제하는 권한      |

# SQL에서의 권한제어 - GRANT

---

- GRANT
  - 권한을 부여하는 명령
  - 형식

**grant** <권한리스트> **on** <객체명> **to** <사용자리스트>

- ▶ <권한리스트> : 권한의 종류에 대한 리스트
  - select, insert, delete, update, references 중 한 개 이상
- ▶ <객체명> : 대상이 되는 객체
- ▶ <사용자리스트> : 권한을 부여받는 사용자들의 리스트
- ▶ <사용자리스트>에게 <객체명>에 대한 <권한리스트>를 실행할 권리를 부여한다는 의미

- 예) ID **kim**인 사용자에게 student 테이블에 대해 **select** 연산을 수행할 수 있는 권한을 부여

(질의 25)

**grant select on student to kim**

# 사용자 계정 생성과 GRANT

---

- lee에게 create session과 create table 권한을 부여(system 계정에서)

(질의 60)

```
grant create session to lee ;
grant create table to lee ;
```

# SQL에서의 권한제어 - GRANT

---

- 예) **kim**에게 **select**와 **delete** 권한을 부여

(질의 26)

```
grant select, delete on student to kim ;
```

- 예) ID **kim**인 사용자에게 student 테이블에 대해 **select** 연산을 수행할 수 있는 권한을 부여

(질의 25)

```
grant select on student to kim ;
```

- 예) student 테이블에서 특정 필드 stu\_id에 대해서만 **select** 연산을 허용 (**오라클은 허용 안함**)

(질의 27)

```
grant select (stu_id) on student to kim ;
```

# SQL에서의 권한제어 - GRANT

---

- 모든 사용자에게 권한을 부여
  - <사용자리스트>에 **public**을 사용
- student에 대해 모든 사용자들에게 select 권한을 부여

(질의 29)

```
grant select on student to public ;
```

- 모든 종류의 권한을 하나의 명령으로 부여하는 방법
  - all privileges라는 키워드를 사용
- student 테이블에 대한 모든 권한을 lee에게 부여

(질의 30)

```
grant all privileges on student to lee ;
```

# WITH GRANT OPTION

- SQL에서는 부여받은 권한을 다른 사용자에게 전파할 수 있는 특별한 옵션
- 예)

(질의 31)

```
grant select on student to kim with grant option
```

- student 테이블에 대한 select 권한을 kim에게 부여함과 동시에 이 권한을 다른 사용자에게 다시 전파할 수 있는 자격까지 부여
- 예) kim이 다음 명령 실행 가능

(질의 32)

```
grant select on student to chang
```

- ▶ 단, chang은 더 이상 다른 사용자에게 권한의 전파가 불가능

- 권한을 전파할 수 있는 강한 수단
  - 하지만 이 옵션이 남용될 경우 데이터 보안에 심각한 문제를 야기할 수 있음

# REVOKE

---

- 다른 사용자에게 부여한 권한을 회수하기 위한 명령
- 형식

**revoke** <권한리스트> **on** <객체명> **from** <사용자리스트>

- ▶ <사용자리스트>로 ♣부터 <객체명>에 대한 <권한리스트> 연산에 대한 권한을 회수

- 예) kim에게 부여되었던 student 테이블에 대한 select 권한을 회수

(질의 33)

**revoke select on student from kim ;**



# SQL에서의 권한제어 - GRANT

---

- SQL 표준에서는 주로 읽기, 삽입, 삭제, 수정 등 데이터 접근에 관련된 권한만을 정의
- 오라클을 비롯한 일부 DBMS들은 create table, alter table, drop table과 같은 스키마 관련 연산에 대한 권한도 grant문으로 부여할 수 있는 기능을 제공

## 롤(ROLE) \*\*\*

---

- 특정 테이블에 대한 권한을 부여할 사용자가 많다면, 반복적으로 grant 문이나 revoke를 실행해야 하는 문제가 발생
- 단순 반복되는 작업을 줄이기 위해서 권한별로 사용자 그룹을 만들어 그룹에 권한을 부여하는 방법이 필요
- 회사의 구성원들이 사원과 임원으로 구분
  - 각각의 보안등급이 다르면 사원 그룹과 임원 그룹으로 나누고 각 그룹에 해당 권한을 부여
- 롤(role) 생성
  - 롤은 데이터베이스 관리자만이 생성 가능

# 롤의 생성 \*\* system 계정에서 수행

---

- 롤 생성 형식

```
create role <롤이름>
```

- 사원(employee)과 임원(manager)을 위한 롤 생성

(질의 36)

```
create role employee ;
```

(질의 37)

```
create role manager ;
```

# 롤을 사용하여 권한 주기

---

- 롤 배정 형식

**grant** <롤 리스트> **to** <사용자리스트>

- 예) 사용자 lee와 kim은 사원, chang과 choi이 임원일 경우

(질의 38)

**grant** employee **to** lee, kim ;

(질의 39)

**grant** manager **to** chang, choi ;

- 예) 사용자 **park**을 employee와 manager에 모두 배정

(질의 40)

**grant** employee, manager **to** park ;

**\*\* grant connect to c##sclim; ← 여기서 connect는 role의 이름**

# 롤에 권한을 부여

---

- **grant**문의 형식과 동일
- employee에게는 student 테이블에 대해 **select** 연산을 허용하고, manager에게는 **select**와 **insert** 연산을 허용

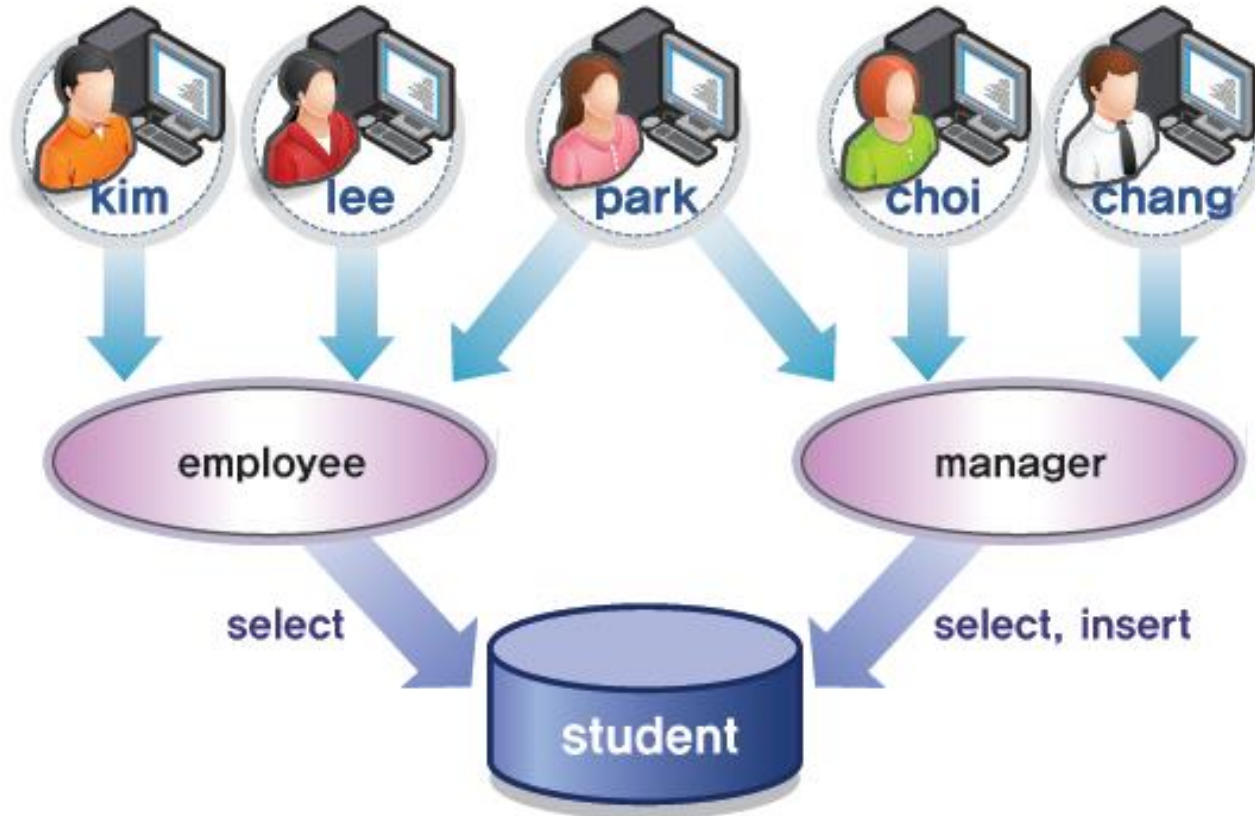
(질의 41)

```
grant select on student to employee ;
```

(질의 42)

```
grant select, insert on student to manager ;
```

# 롤과 권한 부여 상태



# 롤을 사용한 권한 회수

---

- 롤에 부여된 **권한을 회수**

(질의 43)

```
revoke insert on student from manager ;
```

- 사용자로부터 부여된 **롤을 회수**

```
revoke <롤리스트> from <사용자리스트>
```

- 예) 사용자 **choi**를 manager로부터 배제

(질의 44)

```
revoke manager from choi ;
```

# 롤 삭제

---

- 형식

```
drop role <롤이름>
```

- 롤 manager를 데이터베이스에서 삭제

(질의 45)

```
drop role manager ;
```



# 뷰를 이용한 권한 제어

---

- 매우 flexible한 보안 수단 제공
- 특정 테이블에서 일부 필드 혹은 일부 레코드에 대해서만 접근을 허용할 경우 뷰를 사용
  - 필드 뿐만 아니라 테이블 일부에 대해서도 접근제어 가능

(질의 46)

```
create view junior
as select stu_id, name, year, dept_id
from student
where year = 3 ;
```

(질의 47)

```
grant select on junior to kim ;
```

- 위의 질의를 통해 **kim**은 3학년 학생에 대해서만 접근 권한을 가짐

# 오라클에서의 사용자 권한제어

---

- 오라클의 권한 종류
  - 시스템 권한(system privileges)
    - ▶ 사용자의 생성, 테이블이나 테이블 스페이스의 생성 등과 같이 주로 시스템의 자원을 관리할 수 있는 권한
    - ▶ 데이터베이스 관리자 계정(sys, system)은 모든 시스템 권한을 가짐
  - 객체 권한(object privileges)
    - ▶ 해당 객체에 대해 select, insert, delete, update 등과 같은 DML을 실행할 수 있는 권한
    - ▶ 객체
      - 테이블이나 뷰 또는 사용자 정의 함수 등

# 롤의 생성과 사용자 배정

---

- 롤 생성
  - create role문은 데이터베이스 관리자 또는 데이터베이스 관리자로부터 롤을 생성하는 권한을 부여 받은 사용자만이 실행
- Student 테이블의 소유자가 lee라고 가정
  - system 계정에서 다음을 실행

(질의 70)

```
create role stu_role
```

(질의 71)

```
grant select, insert on lee.student to stu_role
```

(질의 72)

```
grant stu_role to park
```

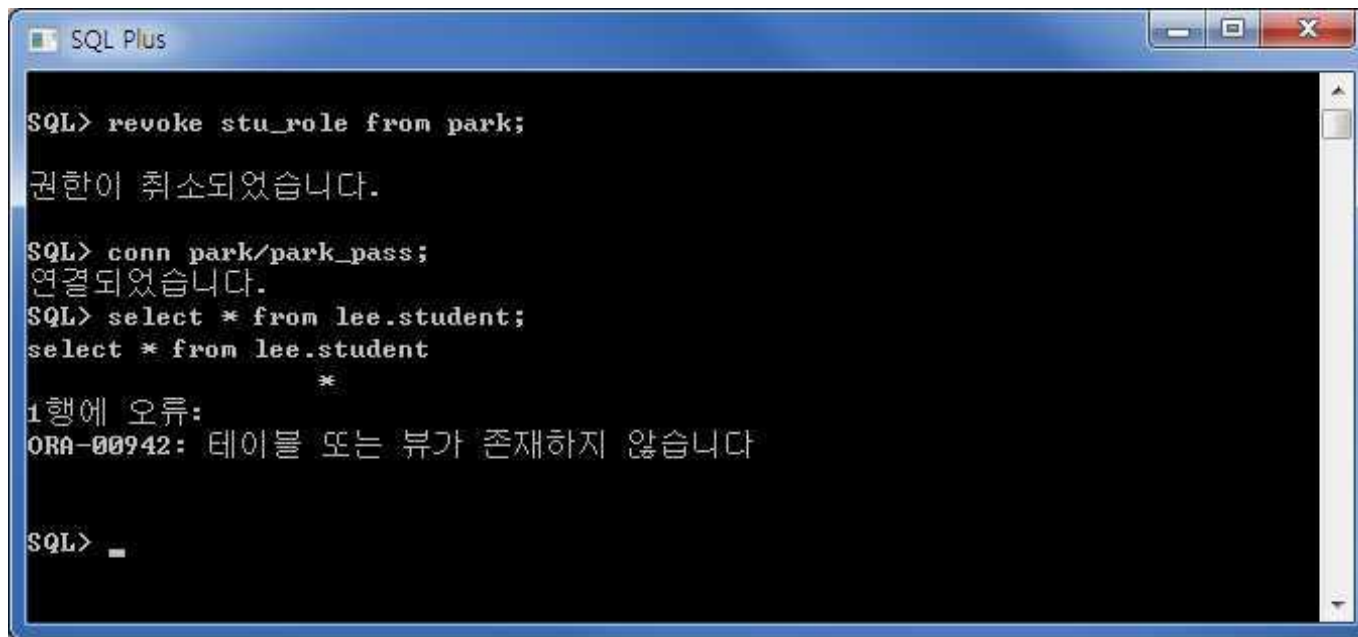
# 롤의 생성과 사용자 배정

- 롤에서 사용자 배제
  - 예) stu\_role에 배정된 park을 다음과 같이 revoke문으로 배제

(질의 73)

**revoke** stu\_role **from** park

(system 계정)



```
SQL> revoke stu_role from park;
권한이 취소되었습니다.

SQL> conn park/park_pass;
연결되었습니다.
SQL> select * from lee.student;
select * from lee.student
 *
1행에 오류:
ORA-00942: 테이블 또는 뷰가 존재하지 않습니다

SQL> _
```

# 과제

---

- 교재 215-216 내용을 참조하여 다음을 수행해 보자
  - **new\_student** 테이블의 name 필드에는 중복된 이름이 생기지 않도록 s\_name\_unique라는 제약을 생성해 보자(수업 사용 자료 이용. student 생성 문 수정하여 실습, 참조 무결성 제약은 없애고...)
  - 생성된 constrain을 확인해 본다  

```
Sql> select constraint_name, table_name from user_constraints
where table_name = 'NEW_STUDENT';
```
  - 제약 생성 후, 같은 이름의 학생 레코드를 삽입해 보자(constraint 위반 메시지 확인, 2번의 insert 문, 이름 필드 값이 같은 레코드 삽입)
  - 생성한 s\_name\_unique 제약을 삭제해 보자
  - 동일한 이름의 학생 레코드를 다시 삽입해 본다
  - 다시 s\_name\_unique 제약을 생성해 보자 (constraint 위반 발생. 왜?)

# 과제

---

- 다음의 롤을 만들어 보자(system 계정에서)
  - System> create role c##sel\_class ;      // class에 대한 select
  - System> create role c##del\_student;      // student에 대한 delete
- 생성한 롤에 권한을 부여해 보자
  - System> grant select on user.class to c##sel\_class;
  - System> grant delete on user.student to c##del\_student;
- 생성한 두 개 role을 사용하여 kim에게 권한을 부여(물론 c##kim)
  - 질의 72를 참조하여 앞에서 생성한 두 개의 role을 kim에게 부여
  - role이 잘 부여되었는지 확인 (select 문과 delete 문 사용하여 확인. Kim 이 수행해야함) (delete 문에 where 절 사용 금지, 왜?)
- 롤을 통해 부여한 권한을 다시 취소해 보자
  - System 계정에서 kim으로부터 role을 취소
  - 취소 후 실제 select문과 delete 문이 수행되지 않음을 확인한다

# 실습 8

# PL/SQL 추가

---

- PL/SQL 연습
- PL/SQL stands for "Procedural Language extensions to the Structured Query Language".
  - combines database language and procedural programming language
  - used in Oracle DBMS



# TRIGGER -- 참고

---

- 다음의 trigger를 생성해 보자. 트리거 코드를 보면서 무슨 동작을 할지 생각해 본다

```
SQL> CREATE OR REPLACE TRIGGER T_TEST_COURSE
 BEFORE UPDATE OR INSERT ON COURSE
 FOR EACH ROW
 BEGIN
 IF INSERTING THEN
 raise_application_error(-20001, '레코드 추가 못합니다');
 END IF;
 IF UPDATING THEN
 IF :NEW.CREDIT > 4 THEN
 raise_application_error(-20002, '크레딧 값이 너무 큼니다');
 END IF;
 END IF;
 END;
/
```

- 실습) 레코드 삽입과 credit 값을 변경하여 위에 정의된 두 가지 경우의 action이 발생하도록 해보자 \*\* 작은 따옴표 입력 필요

# TRIGGER

---

- 정의
  - A procedure that runs automatically when a certain **event** occurs in the DBMS
  - 이벤트 종류: 테이블에 발생하는 insert, update, delete 연산
- Trigger 프로시저의 구성 요소
  - **Event**: When this event happens, the trigger is activated
  - **Condition** (optional): If the condition is true, the trigger executes, otherwise skipped
  - **Action**: The actions performed by the trigger
  - 이벤트 발생시 트리거 조건을 보고 true이면 액션을 수행

# PL/SQL 실행

---

- PL/SQL
  - **오라클**에서 DBMS의 표준 질의어인 SQL을 확장하여 개발한 고급 프로그래밍 언어
  - Procedural Language extension to SQL의 약어
- SQL 만으로는 DB 사용 프로그래밍에 제한적인 부분도 있기 때문에 절차적인 프로그래밍 요소를 가미한 것이라 할 수 있음
  - 변수 정의, 조건 처리, 반복문 등을 지원
  - 예) 앞에서 본 트리거 코드

# PL/SQL을 실습해 보자

---

- 기존 프로그래밍 언어를 고려한 언어이기 때문에 프로그래밍 경험이 있는 개발자라면 쉽게 사용할 수 있음
- 기본 단위는 블록(block)
  - 변수를 선언하는 부분인 **선언부**와 실행코드가 나오는 **실행부**, 실행 중 에러가 발생했을 때 실행되는 **예외처리부**로 구성
- 실습 시 화면에 출력이 보이도록 하기 위해 아래와 같이 환경 변수 설정
- – Set SERVEROUTPUT on
- 슬라이드 예제를 실행해 보자
  - 코드를 노트 패드에 일단 입력한 후, sql 화면에 붙여 넣기한다

# PL/SQL 실행하기

---

- PL/SQL의 기본 형식

```
DECLARE
 변수 선언문;
BEGIN
 실행문;
EXCEPTION
 예외처리문;
END;
```

예) 변수 **n**을 생성하고 **10**을 저장한 후 출력

|   |                          |
|---|--------------------------|
| 1 | DECLARE                  |
| 2 | n INTEGER;               |
| 3 | BEGIN                    |
| 4 | n := 10;                 |
| 5 | dbms_output.put_line(n); |
| 6 | END;                     |

# PL/SQL 실행하기

---

- PL/SQL을 실행하려면 우선 **set serveroutput on** 명령으로 출력을 활성화해야 함
- 프로그램 입력을 마치고 실행시키라는 의미로 맨 마지막에 **/'** 문자를 입력해야 함

```
SQL> set serveroutput on
SQL> declare
 2 n integer;
 3 begin
 4 n := 10;
 5 dbms_output.put_line(n);
 6 end;
 7 /
10
PL/SQL 처리가 정상적으로 완료되었습니다.
```

# Hello World 작성

---

- PL/SQL로 "Hello World!" 출력
- 다음을 입력(실습 코드, 변수 선언은 없음)

```
BEGIN
```

```
 DBMS_OUTPUT.PUT_LINE('Hello World!'); /* 메시지 */
```

```
END;
```

```
/
```

- **코멘트:** - 혹은 `/* */`
- 실행을 위해서는  `'/'`을 입력해 줘야 함

# 추가 PL/SQL

---

- 앞의 SQL/PL 코드를 프로시저로 만들어 반복 수행할 수 있다. 이를 저장 프로시저라고 한다
- 아래 **저장 프로시저(stored procedure)** 생성 (만약 동일 이름의 프로시저가 있다면 자동 수정됨) (view의 생성과 유사한 개념)

Create or replace **procedure proc\_test**

Is

```
NUM_ID NUMBER(4);
```

```
CHAR_NAME VARCHAR(20);
```

```
BEGIN
```

```
NUM_ID := 2000;
```

```
CHAR_NAME := '래리 엘리슨';
```

```
DBMS_OUTPUT.PUT_LINE(' ' || NUM_ID || ' ' || CHAR_NAME);
```

```
END;
```

```
/
```



## 추가 PL/SQL 실습

---

- 앞의 슬라이드의 저장 프로시저를 아래와 같이 수행시켜 보자
  - Sql> **execute** proc\_test
- 저장 프로시저도 테이블과 같이 DBMS의 자원으로 저장된다. 따라서 질의를 통해 확인해 볼 수 있다
  - Sql> select **text** from **user\_source** where name = upper('프로시저 명');
  - **프로시저 이름으로 proc\_test를 대입시켜 수행해 보자**
  - 혹은
  - Sql> select text from **user\_source** where name = 'PROC\_TEST';
  - DBMS는 내부 테이블 데이터를 대부분 대문자로 저장한다, 따라서 name 역시 대문자로 저장되며, 대/소문자가 구분됨

## 간단한 FOR LOOP 문의 예

---

```
DECLARE
 N NUMBER(4);
BEGIN
 N := 1;
 LOOP
 DBMS_OUTPUT.PUT_LINE(N);
 N := N + 1;
 IF N > 5 THEN
 EXIT;
 END IF;
 END LOOP;
END;
/
```

- 실습)  
옆의 코드를 stored procedure로 저장  
이름은 print\_num으로 한다.  
그리고 저장 프로시저를 수행시켜 본다.

## 사용자 입력 가능 \*\*

---

- 앞에서 작성한 프로시저를 사용자 입력 가능하도록 수정해 보자
- 코드를 수정하고, 아래와 같이 실행해 본다
  - Sql> `execute print_num2(20);`

Create or replace procedure print\_num2( N **IN Integer** )

IS

    N2 integer;

BEGIN

    N2 := 1;

    LOOP

        DBMS\_OUTPUT.PUT\_LINE(N2);

        IF N2 > N THEN

            EXIT;

        END IF;

        N2 := N2 + 1;

    END LOOP;

END;

## 테이블 데이터를 사용한 PL/SQL 문의 예 \*\*

---

- 변수 선언의 예.
  - 레코드의 **특정 필드** 값을 가져오거나 저장할 때 사용
    - `VAR_STU_ID    STUDENT.STU_ID%TYPE;    /* student 테이블 레코드의 stu_id 필드에 들어 있는 값을 사용하기 위한 방법. */`
    - `VAR_NAME    DEPARTMENT.NAME%TYPE;`
  - **레코드 전체** 데이터 처리에 사용
    - `VAR_REC    STUDENT%ROW_TYPE;`  
/\* student 테이블 레코드에 있는 값을 사용하기 위한 방법.  
VAR\_REC은 이 레코드 값을 저장하는 변수. \*/

# 테이블 데이터를 사용한 PL/SQL 문의 예 \*\*

---

```
DECLARE
 SNAME STUDENT.NAME%TYPE;
 SADDRESS STUDENT.ADDRESS%TYPE;
BEGIN
 select name, address into SNAME, SADDRESS
 from student
 where stu_id = '1292003'; /* sql 문 */

 dbms_output.put_line('1292003의 이름과 주소는 ' || SNAME || ',
' || SADDRESS || ' 입니다. ');
END;
/
```

- 위의 코드를 수행해보자 (작은 따옴표 주의)
- 위의 코드를 참조하여 다음과 같은 저장 프로시저를 생성해 보자
  - '이태규' 교수의 주민번호를 출력한다 (professor 테이블 이용)

# Rowtype을 사용한 예

---

- 아래 코드는 rowtype을 이용하여 레코드 전체 데이터 처리
- 필드 별로 변수를 만드는 것보다 편한 측면이 있음

```
DECLARE
```

```
 SREC STUDENT%ROWTYPE;
```

```
BEGIN
```

```
 select * into SREC from student
```

```
 where stu_id = '1292003';
```

```
 dbms_output.put_line('1292003의 이름과 주소는 ' || SREC.name || ', ' ||
SREC.address || ' 입니다.');
```

```
END;
```

```
/
```

## 사용자 입력 값의 데이터 타입에 적용

---

```
Create or replace procedure del_test(del_grade IN takes.grade%type)
IS
 BEGIN
 delete takes where grade = del_grade; /* 레코드 삭제 sql 문 */
 END;
/
```

- 위의 코드를 읽고 이해해 본다
- 아래처럼 실행해 본다

```
Sql> commit;
```

```
Sql> select * from takes where grade = 'B'; /// 현재 레코드 확인
```

```
Sql> execute del_test('B');
```

```
Sql> select * from takes where grade = 'B'; /// 다시 확인
```

```
Sql> rollback; // 삭제 취소
```

```
Sql> select * from takes where grade = 'B'; /// 다시 확인
```

# 연습

---

- 수업 시간에 배운 질의를 연습해 보자
- 아래 질의를 수행해 보고 의미를 이해해 보자

```
SQL> select name, stu_id
2 from (select * from student order by name)
3 where rownum <= 4;
```

- 위의 질의를 참조하여 stu\_id가 가장 앞선 3명의 학생 이름을 출력해본다
- Commit과 rollback에 대해서 공부해 보자 (전에 한번 수업 했었음)
  - 아래와 같이 질의를 입력해보자. 이 과정에서 commit/rollback의 의미에 대해서 생각해 본다
  - Sql> commit;
  - Sql> update table set name = '김광식2' where name='김광식';
  - Sql> select name, stu\_id from student; // 김광식 있는지 확인
  - Sql> rollback; // commit 이후 수행된 갱신 연산을 무효화 시킴
  - Sql> select name, stu\_id from student; // 김광식 있는지 확인



# 과제

---

- 1부터 X까지의 합을 구해 출력하는 저장 프로시저를 만들고, 이를 수행하시오.

- \* 출력 예시: "1부터 100까지의 합 = 5050"

100은 X(입력) 값으로 부터 출력. 5050은 합의 결과 값으로 출력됨

- (참고)

Loop 문을 사용해야함. 아래 코드를 참조 (N이 6이 되면 loop 문 탈출)

```
N := 1;
```

```
LOOP
```

```
 N := N + 1;
```

```
 EXIT WHEN N > 5;
```

```
END LOOP;
```

# 과제

---

- 주소값을 매개변수로 하여, 해당 주소를 가진 학생들의 year를 1씩 증가 시키는 저장 프로시저를 코딩하고 수행한다
  - 수행 예) `execute add_year('부산')`
- 교수 id를 매개변수로 하여 해당 id를 가진 교수의 이름과 소속 학과를 출력하는 저장 프로시저를 코딩하여 수행한다
  - 수행 예) `execute prof_name('92001')`