**<DB & SQL>**

* **일시) 2025년 2월 XX**
* **장소) 한국 IT 교육원**

**/\***

DBMS(Database Mangement System)란?

데이터베이스 관리 시스템의 약자로 데이터를 효율적으로 저장, 검색, 관리하는 소프트웨어 도구

저장은 <파일시스템에 각 DBMS 제품의 자료구조에서 정의한 형태>로 한다.

Oracle은?

전 세계에서 가장 성능이 좋고 많이 사용되는 상용 DBMS

....

8i: i=> internet : 인터넷 환경 지원

....

10g: g=> grid: 빅데이터 처리를 위한 분산 환경을 지원

....

12c: c=> cloud: 클라우드 환경의 데이터베이스 지원

.....

18c ~ : 기능의 확장

\*/

-- SQL에서 한 줄 주석: SQL 실행에는 영향을 미치지 않는다.

/\*

SQL에서 여러줄 주석

\*/

/\*

SQL(Structed Query Languae) : 데이터베이스 관리 시스템에서 데이터를 조작하고 관리하기 위해 표준화된 언어

<SQL 문의 종류>

1. DCL(Data Control Language): 데이터베이스 사용자의 권한을 관리하기 위한 SQL문

예) grant connect to open\_source

2. DDL(Data Definition Language): 데이터베이스 구조를 정의하고 관리하기 위한 SQL문

예)CREATE TABLE Student\_Scores1(

student\_name varchar2(50),

KOREAN\_SCORE number,

english\_score number,

math\_score number

);

3. DML(Data Manipulation Langugae): 데이터를 조작하기 위한 SQL문

- 데이터 조작의 유형: CRUD (Create, Read, Update, Delete)

=> insert, select, update, delete

4. TCL(Transaction Control Language): 데이터베이스 트랜잭션을 제어하기위한 SQL문

-COMMIT, ROLLBACK...

\*/

/\*

table: 저장할 데이터의 논리적인 구조

필드 또는 컬럼: 저장할 데이터의 특성을 표현

create table [테이블명](

[필드명] [필드타입],

..

)

\*/

-- create table student\_scores(id number, student\_name varchar2(50) .....); <- SQL 종료는 ';'

-- SQL문은 대소문자를 구분하지 않는다. (대소문자 Rule개발팀의 정책에 따름)

-- 테이블명, 컬럼(필드)명은 통상 snake 룰을 따른다.

-- 카멜(Camel)방식: 두 단어 이상으로 작명시 단어와 단어사이 첫번째 문자를 대문자로 표현

-- ex) StudentScores : 클래스명 또는 테이블명일때에는 첫단어의 첫글자도 대문자

-- studentName: 컬럼(필드)명일때 첫단어의 첫글자는 소문자.

-- 스네이크(Snake)방식: 두 단어 이상으로 작명시 단어는 모두 소문자로 작성하고 연결은 '\_'로 함

-- ex) student\_scores, student\_name

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**단계별 학생 점수 테이블 생성**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**-- Version1]학생 점수를 관리하는 테이블 생성**

**-- 학생 정보: 학생이름, 국/영/수 점수**

**CREATE TABLE Student\_Scores1(**

**student\_name varchar2(50),**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number**

**);**

**describe Student\_Scores1;**

**INSERT INTO Student\_Scores1 (student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES ('홍길동', 90, 80, 95);**

**INSERT INTO Student\_Scores1 (student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES ('김철수', 80, 85, 95);**

**-- 중복되지 않는 학생데이터만 처리할 경우에는 문제없이 처리 가능**

**INSERT INTO Student\_Scores1 (student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES ('홍길동', 90, 100, 95);**

**-- 위와 같이 중복된 학생데이터가 있는 경우 중복학생의 데이터에 대해 구분할 수 없다.**

**-- Version2] 중복데이터를 구분하기 위해 자연키 도입**

**-- 자연키(Natural Key): 실생활에서 발생하는 데이터를 구분지을 수 있는 값**

**-- 학생 정보: 주민번호, 학생이름, 국/영/수 점수**

**CREATE TABLE Student\_Scores2(**

**resistration\_number varchar2(50),**

**student\_name varchar2(50),**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number**

**);**

**INSERT INTO Student\_Scores3 (id, resistration\_number,student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (1,'200120-1011212','홍길동', 90, 100, 95);**

**INSERT INTO Student\_Scores3 (id, resistration\_number,student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (2,'980120-1017282','홍길동', 95, 90, 85);**

**INSERT INTO Student\_Scores2 (resistration\_number,student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES ('200120-1011212','홍길동', 90, 100, 95);**

**INSERT INTO Student\_Scores2 (resistration\_number,student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES ('980120-1017282','홍길동', 95, 90, 85);**

**-- 자연키를 ID로 하여 중복 학생에 대한 구분이 가능**

**update student\_scores2**

**set resistration\_number = '980120-2017282'**

**where student\_name='홍길동' and korean\_score=95;**

**select \* from student\_scores2;**

**-- 자연키는 항상 절대 불변의 값은 아니다.**

**-- 자연키를 ID(Primary Key)로 사용할 경우 외부에서 해당 데이터를 참조하는 경우**

**-- 더이상 예전데이터를 접근할 수 없다.**

**-- Version3] 중복데이터를 구분하기 위해 대리키 도입**

**-- 대리키(Surrogate Key): 데이터베이스에서 사용하는 인공적인 식별자로 테이블의 기본키**

**--Primary 키로 사용한다.**

**-- 학생 정보: 주민번호, 학생이름, 국/영/수 점수**

**CREATE TABLE Student\_Scores3(**

**id number,**

**resistration\_number varchar2(50),**

**student\_name varchar2(50),**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number**

**);**

**INSERT INTO Student\_Scores3 (id, resistration\_number,student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (1,'200120-1011212','홍길동', 90, 100, 95);**

**INSERT INTO Student\_Scores3 (id, resistration\_number,student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (2,'980120-1017282','홍길동', 95, 90, 85);**

**select \* from student\_scores3;**

**update student\_scores3**

**set resistration\_number = '980120-2017282'**

**where id=2;**

**-- 대리키를 기본키로 사용하여 중복의 문제를 해결할 수 있다.**

**-- 그러나 대리키를 insert할 때 똑같은 대리키를 입력하게 되면 문제가 발생된다.**

**INSERT INTO Student\_Scores3 (id, resistration\_number,student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (2,'210320-1017282','강감찬', 100, 90, 85); -- 실수로 id가 중복되게 입력한다.**

**-- Version4] 대리키가 실수로 중복입력이 되지 않도록 하기 위하여 primary key를 사용**

**-- 학생 정보: id, 학생이름, 국/영/수 점수 (주민번호는 더 이상 PK의 역할을 하지 않으므로 생략)**

**CREATE TABLE Student\_Scores4(**

**id number,**

**student\_name varchar2(50),**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number,**

**primary key (id)**

**);**

**INSERT INTO Student\_Scores4 (id, student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (1,'홍길동', 90, 100, 95);**

**INSERT INTO Student\_Scores4 (id, student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (2,'홍길동', 95, 90, 85);**

**INSERT INTO Student\_Scores4 (id, student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (2,'강감찬', 100, 90, 85); -- 실수로 id가 중복되게 입력하는 것을 시도**

**INSERT INTO Student\_Scores4 (id, student\_name, KOREAN\_SCORE, english\_score, math\_score)**

**VALUES (3,'강감찬', 100, 90, 85); -- 정상 id로 입력하는 것을 시도**

**-- 대리키가 중복 입력이 되지 않게 문제를 해결**

**-- 여전히 남아 있는 문제: PK번호가 중복되지 않게 입력하는 것은 누구의 책임? 입력하는 사람의 책임**

**-- 사람이 하는일은 실수가 가능**

**CREATE TABLE Student\_Scores(**

**ID number, -- number: 숫자 처리 타입**

**student\_name varchar2(50), -- varchar2(문자열 길이): 문자 처리 타입**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number,**

**primary key (id)**

**);**

**describe student\_scores;**

**drop table student\_scores;**

**-- 시퀀스 생성**

**CREATE SEQUENCE student\_scores\_seq**

**START WITH 1 -- 시작을 1부터**

**INCREMENT BY 1 -- 1씩 증가**

**NOMAXVALUE -- 최대값 제한 없음**

**NOCYCLE; -- 사이클 없음**

**-- 시퀀스 조회**

**select \* from USER\_SEQUENCES**

**where SEQUENCE\_NAME='STUDENT\_SCORES\_SEQ';**

**drop sequence student\_scores\_seq;**

**<SEQUENCE 상태확인 쿼리>**

SELECT \*

FROM USER\_SEQUENCES

WHERE SEQUENCE\_NAME = 'STUDENT\_ID';

SELECT \* FROM USER\_SEQUENCES WHERE UPPER(SEQUENCE\_NAME) = 'STUDENT\_ID';

**-- 데이터 입력**

**-- insert into [테이블명] (컬럼명1, ... ,컬럼N) values (입력값1, ... ,입력값N);**

**INSERT INTO Student\_Scores (id, student\_name, korean\_score, english\_score, math\_score)**

**VALUES (student\_scores\_seq.NEXTVAL,'홍길동', 90, 80, 95);**

**/\* Oracle 12c 이후 버전의 간소화 방식**

**CREATE TABLE Student\_Scores (**

**id NUMBER GENERATED ALWAYS AS IDENTITY,**

**student\_name VARCHAR2(50),**

**korean\_score NUMBER,**

**english\_score NUMBER,**

**math\_score NUMBER,**

**PRIMARY KEY (id));**

**INSERT INTO Student\_Scores (student\_name, korean\_score, english\_score, math\_score)**

**VALUES ('홍길동', 90, 80, 95);**

**\*/**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* <테이블 스키마> 변경**

**\* - 컬럼(열) 이름 변경, 컬럼 추가, 컬럼 삭제 ..**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**-- 컬럼(열) 이름 변경**

**-- alter table [테이블명]**

**-- rename column [바꿀 컬럼명] to [바꾸고자하는 컬럼명]**

**describe student\_scores;**

**ALTER TABLE student\_scores**

**RENAME COLUMN korean\_score TO kor\_score;**

**ALTER TABLE student\_scores**

**RENAME COLUMN english\_score TO eng\_score;**

**describe student\_scores;**

**/\* 상위 버전에는 아래와 같이 간소화도 가능**

**ALTER TABLE student\_scores**

**RENAME COLUMN korean\_score TO kor\_score,**

**RENAME COLUMN english\_score TO eng\_score;**

**\*/**

**-- 테이블 열 추가**

**alter table student\_scores**

**add (sci\_score number);**

**describe student\_scores;**

**-- 테이블의 열 삭제**

**alter table student\_scores**

**drop column sci\_score;**

**describe student\_scores;**

**-- 시간 데이터 열 추가**

**alter table student\_scores**

**add (enroll\_date date); -- date: 오라클에서 지원하는 날짜 타입**

**describe student\_scores;**

**student\_scores => snake(스네이크)방식 (DB테이블, 컬럼)**

**studentScores => Camel(카멜)방식 Java(변수, 클래스명)**

**참고) StduentScores: 첫글자 대문자인 카멜표기법은**

**클래스명일 경우에**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**-- TO\_DATE([날짜문자열],[날짜포멧]): 날짜 문자열을 오라클의 DATE 으로 변환**

**VALUES (student\_scores\_seq.NEXTVAL, '홍길동', 90, 80, 95, TO\_DATE('28/05/2023', 'DD/MM/YYYY'));**

**-- 2번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김철수', 85, 90, 92, TO\_DATE('29/05/2023', 'DD/MM/YYYY'));**

**-- 3번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이영희', 78, 92, 88, TO\_DATE('30/05/2023', 'DD/MM/YYYY'));**

**-- 4번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '박지민', 87, 85, 90, TO\_DATE('31/03/2023', 'DD/MM/YYYY'));**

**-- 5번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김영호', 92, 87, 82, TO\_DATE('01/04/2023', 'DD/MM/YYYY'));**

**-- 6번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이현주', 95, 88, 79, TO\_DATE('02/04/2023', 'DD/MM/YYYY'));**

**-- 7번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '정민우', 82, 91, 86, TO\_DATE('03/03/2023', 'DD/MM/YYYY'));**

**-- 8번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김민지', 89, 84, 93, TO\_DATE('04/04/2023', 'DD/MM/YYYY'));**

**-- 9번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '박준호', 76, 90, 87, TO\_DATE('05/04/2023', 'DD/MM/YYYY'));**

**-- 10번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '최서연', 93, 89, 91, TO\_DATE('05/03/2023', 'DD/MM/YYYY'));**

**-- 11번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '박현지', 85, 90, 92, TO\_DATE('01/05/2023', 'DD/MM/YYYY'));**

**-- 12번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김동현', 78, 92, 88, TO\_DATE('02/05/2023', 'DD/MM/YYYY'));**

**-- 13번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '정윤아', 87, 85, 90, TO\_DATE('03/05/2023', 'DD/MM/YYYY'));**

**-- 14번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이민수', 92, 87, 82, TO\_DATE('04/05/2023', 'DD/MM/YYYY'));**

**-- 15번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김서영', 95, 88, 79, TO\_DATE('05/05/2023', 'DD/MM/YYYY'));**

**-- 16번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '박진호', 82, 91, 86, TO\_DATE('06/05/2023', 'DD/MM/YYYY'));**

**-- 17번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이민지', 89, 84, 93, TO\_DATE('07/05/2023', 'DD/MM/YYYY'));**

**-- 18번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '최준호', 76, 90, 87, TO\_DATE('08/05/2023', 'DD/MM/YYYY'));**

**-- 19번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김수연', 93, 89, 91, TO\_DATE('09/05/2023', 'DD/MM/YYYY'));**

**-- 20번 학생 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이예준', 93, 100, 91, TO\_DATE('09/05/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '신동엽', 82, 88, 95, TO\_DATE('01/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이하나', 78, 95, 92, TO\_DATE('02/02/2023', 'DD/MM/YYYY'));**

**CREATE TABLE Student\_Scores(**

**ID number, -- number: 숫자 처리 타입**

**student\_name varchar2(50), -- varchar2(문자열 길이): 문자 처리 타입**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number,**

**primary key (id)**

**);**

**위 쿼리는 오라클 11g에서 만들었어.**

**위 테이블에 필요한 초기데이터 10개 만들어줘.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**CREATE TABLE Student\_Scores(**

**ID number, -- number: 숫자 처리 타입**

**student\_name varchar2(50), -- varchar2(문자열 길이): 문자 처리 타입**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number,**

**primary key (id)**

**);**

**CREATE SEQUENCE student\_scores\_seq**

**START WITH 1 -- 시작을 1부터**

**INCREMENT BY 1 -- 1씩 증가**

**NOMAXVALUE -- 최대값 제한 없음**

**NOCYCLE; -- 사이클 없음**

**위 쿼리는 오라클 11g에서 만들었어.**

**위 테이블에 필요한 초기데이터를 시퀀스를 활용해서 10개 만들어줘.**

**CREATE TABLE Student\_Scores(**

**ID number, -- number: 숫자 처리 타입**

**student\_name varchar2(50), -- varchar2(문자열 길이): 문자 처리 타입**

**KOREAN\_SCORE number,**

**english\_score number,**

**math\_score number,**

**primary key (id)**

**);**

**CREATE SEQUENCE student\_scores\_seq**

**START WITH 1 -- 시작을 1부터**

**INCREMENT BY 1 -- 1씩 증가**

**NOMAXVALUE -- 최대값 제한 없음**

**NOCYCLE; -- 사이클 없음**

**위 쿼리는 오라클 11g에서 만들었어.**

**위 테이블에 필요한 초기데이터를 시퀀스를 활용해서 10개 만들어줘.**

**남학생의 국어점수 평균은 25점, 여학생의 국어점수 평균은 90점이 되게 해줘.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**같은 조건으로 1000개의 데이터를 csv 형태로 만들고 다운로드 링크줘**

**/\***

**제약조건(constraint)**

**- NOT NULL: 값을 반드시 가져야 하는 조건**

**\*/**

**CREATE TABLE person (**

**Name VARCHAR2(50) NOT NULL,**

**Age NUMBER**

**);**

**drop table person;**

**insert into person (Name,Age) values ('홍길동', 50); -- OK**

**insert into person (Name) values ('홍길동'); -- OK**

**insert into person (Age) values (30); -- 에러**

**CREATE TABLE Products (**

**ProductName VARCHAR2(50),**

**Price NUMBER,**

**CONSTRAINT CHK\_Price CHECK (Price > 0)**

**);**

**insert into products (productName, price) values('누가바',600); -- OK**

**insert into products (productName) values('빠삐코'); -- OK**

**insert into products (productName, price) values('수박바',0); -- 에러**

**/\***

**테이블 컬럼의 키 종류:**

**1. 기본 키 (Primary Key):**

**- 테이블 내에서 각 행을 고유하게 식별하는 열 또는 열의 집합**

**- 테이블에 하나만 존재하며 중복된 값을 허용 X**

**- 테이블의 레코드를 식별하고 다른 테이블과의 관계를 설정하는 데 사용**

**- NULL 값을 허용 X**

**2. 외래 키 (Foreign Key):**

**- 다른 테이블의 기본 키를 참조**

**- 관계형 데이터베이스에서 테이블 간의 관계를 설정**

**- 외래 키는 참조된 테이블의 기본 키 값과 일치해야 하거나 NULL 값을 가질 수 있습니다.**

**3. 고유 키 (Unique Key):**

**- 고유 키는 테이블 내에서 중복된 값을 허용하지 않는 열 또는 열의 집합**

**- 기본 키와 마찬가지로 각 행을 고유하게 식별합니다.**

**- 하지만 고유 키는 기본 키가 아닌 열에 적용. 테이블 내에서 여러 개의 고유 키를 가질 수 있다.**

**4. 대체 키 (Alternate Key):**

**- 대체 키는 기본 키가 아닌 후보 키입니다.**

**- 기본 키로 사용될 수 있는 열 또는 열의 집합 중 하나입니다.**

**- 대체 키는 테이블에서 각 행을 고유하게 식별하는 데 사용될 수 있지만 기본 키로 선택되지 않은 경우입니다.**

**이러한 키들은 데이터의 무결성을 보장하고 테이블 간의 관계를 설정하는 데 사용됩니다.**

**각 키는 데이터베이스 설계에 따라 적절하게 선택되고 정의되어야 합니다.**

**\*/**

**-- "Customers" 테이블 생성**

**CREATE TABLE Customers (**

**Customer\_ID NUMBER PRIMARY KEY,**

**Customer\_Name VARCHAR2(50),**

**Email VARCHAR2(50)**

**);**

**CREATE TABLE Orders (**

**Order\_ID NUMBER PRIMARY KEY,**

**Customer\_ID NUMBER,**

**Order\_Date DATE,**

**FOREIGN KEY (Customer\_ID) REFERENCES Customers(Customer\_ID)**

**-- Orders 테이블에서는 Customers 테이블의 customer\_id를 외래키로 사용할 것을 명시적으로 지정**

**-- Not null, customer\_id는 Customers 테이블의 customer\_id 값으로만 입력이 되어야 한다.**

**);**

**-- "Customers" 테이블에 데이터 삽입**

**INSERT INTO Customers (Customer\_ID, Customer\_Name, Email)**

**VALUES (1, '홍길동', 'hong@example.com');**

**INSERT INTO Customers (Customer\_ID, Customer\_Name, Email)**

**VALUES (2, '김철수', 'kim@example.com');**

**INSERT INTO Customers (Customer\_ID, Customer\_Name, Email)**

**VALUES (3, '이상은', 'lee@example.com');**

**-- "Orders" 테이블에 데이터 삽입**

**INSERT INTO Orders (Order\_ID, Customer\_ID, Order\_Date)**

**VALUES (1001, 1, TO\_DATE('2023-06-04', 'YYYY-MM-DD'));**

**INSERT INTO Orders (Order\_ID, Customer\_ID, Order\_Date)**

**VALUES (1002, 2, TO\_DATE('2023-06-05', 'YYYY-MM-DD'));**

**INSERT INTO Orders (Order\_ID, Customer\_ID, Order\_Date)**

**VALUES (1003, 3, TO\_DATE('2023-06-06', 'YYYY-MM-DD'));**

**INSERT INTO Orders (Order\_ID, Customer\_ID, Order\_Date)**

**VALUES (1004, 10, TO\_DATE('2023-06-05', 'YYYY-MM-DD'));**

**select \* from customers;**

**select \* from orders;**

**-- customers테이블과 orders 테이블을 customer\_id로 Left join 해보세요.**

**select c.Customer\_ID, c.Customer\_Name, c.email, o.Order\_ID, o.order\_date**

**from customers c**

**left join orders o**

**on c.customer\_id = o.customer\_id;**

**<JOIN>**

**--------------------------------------------------------------------------**

**--------------------------------------------------------------------------**

**-- 테이블 1: 회원 정보 테이블**

**CREATE TABLE members (**

**member\_id NUMBER PRIMARY KEY, -- 기본 키 (Primary Key)**

**user\_id VARCHAR2(10) UNIQUE NOT NULL, -- 대체 키 (Alternate Key)**

**email VARCHAR2(100) UNIQUE, -- 고유 키 (Unique Key) (광의의 개념으로 기본키/협의로 중복불가키)**

**full\_name VARCHAR2(100) NOT NULL, -- NOT NULL 제약조건 추가**

**birth\_date DATE, -- 제약조건 없이 추가**

**role\_id number,**

**FOREIGN KEY (role\_id) REFERENCES roles(role\_id)**

**);**

**drop table members;**

**-- 테이블 2: 역할 정보 테이블**

**CREATE TABLE roles (**

**role\_id NUMBER PRIMARY KEY, -- 기본 키 (Primary Key)**

**role\_name VARCHAR2(50) UNIQUE not null -- 고유 키 (Unique Key)**

**);**

**drop table roles;**

**drop sequence member\_seq;**

**-- 시퀀스 생성**

**CREATE SEQUENCE member\_seq START WITH 1 INCREMENT BY 1;**

**-- roles 테이블에 레코드 삽입**

**INSERT INTO roles (role\_id, role\_name)**

**VALUES (1, '관리자');**

**INSERT INTO roles (role\_id, role\_name)**

**VALUES (2, '사용자');**

**INSERT INTO roles (role\_id, role\_name)**

**VALUES (3, '손님');**

**INSERT INTO roles (role\_id, role\_name)**

**VALUES (4, '매니저');**

**INSERT INTO roles (role\_id, role\_name)**

**VALUES (5, '직원');**

**-- members 테이블에 레코드 삽입**

**INSERT INTO members (member\_id, user\_id, email, full\_name, birth\_date, role\_id)**

**VALUES (member\_seq.NEXTVAL, 'user001', 'user1@example.com', '홍길동', TO\_DATE('1990-01-01', 'YYYY-MM-DD'), 1);**

**INSERT INTO members (member\_id, user\_id, email, full\_name, birth\_date, role\_id)**

**VALUES (member\_seq.NEXTVAL, 'user002', 'user2@example.com', '이순신', TO\_DATE('1985-05-10', 'YYYY-MM-DD'), 2);**

**INSERT INTO members (member\_id, user\_id, email, full\_name, birth\_date, role\_id)**

**VALUES (member\_seq.NEXTVAL, 'user003', 'user3@example.com', '김철수', TO\_DATE('1993-11-25', 'YYYY-MM-DD'), 2);**

**INSERT INTO members (member\_id, user\_id, email, full\_name, birth\_date, role\_id)**

**VALUES (member\_seq.NEXTVAL, 'user004', 'user4@example.com', '박영희', TO\_DATE('1988-07-15', 'YYYY-MM-DD'), 1);**

**INSERT INTO members (member\_id, user\_id, email, full\_name, birth\_date, role\_id)**

**VALUES (member\_seq.NEXTVAL, 'user005', 'user5@example.com', '정민지', TO\_DATE('1992-04-20', 'YYYY-MM-DD'), 3);**

**select \* from members;**

**select \* from roles;**

**SELECT m.member\_id, m.user\_id, m.email, m.full\_name, m.birth\_date, r.role\_name**

**FROM members m**

**LEFT JOIN roles r**

**ON m.role\_id = r.role\_id;**

**<문제>**

**다음과 같은 두 개의 테이블을 만들고 FOREIGN KEY를 설정한 후,**

**ID 값을 자동 증가하도록 \*\*시퀀스(Sequence)\*\*를 활용하여 데이터를 삽입하시오.**

**1. Departments 테이블**

**- Dept\_ID (NUMBER, PRIMARY KEY, 시퀀스로 자동 증가)**

**- Dept\_Name (VARCHAR2(50))**

**2. Employees 테이블**

**- Emp\_ID (NUMBER, PRIMARY KEY, 시퀀스로 자동 증가)**

**- Emp\_Name (VARCHAR2(50))**

**- Dept\_ID (NUMBER, FOREIGN KEY - Departments 테이블의 Dept\_ID 참조)**

**주어진 조건**

**- 3개의 부서를 Departments 테이블에 추가하세요.**

**- 각 부서에 최소 2명 이상의 직원을 Employees 테이블에 추가하세요.**

**- ID 값은 시퀀스를 사용하여 자동 증가하도록 구현하세요.**

**<JOIN>Foreign KEY>**

**/\***

**<문제>**

**다음과 같은 두 개의 테이블을 만들고 FOREIGN KEY를 설정한 후,**

**ID 값을 자동 증가하도록 \*\*시퀀스(Sequence)\*\*를 활용하여 데이터를 삽입하시오.**

**1. Departments 테이블**

**- Dept\_ID (NUMBER, PRIMARY KEY, 시퀀스로 자동 증가)**

**- Dept\_Name (VARCHAR2(50))**

**2. Employees 테이블**

**- Emp\_ID (NUMBER, PRIMARY KEY, 시퀀스로 자동 증가)**

**- Emp\_Name (VARCHAR2(50))**

**- Dept\_ID (NUMBER, FOREIGN KEY - Departments 테이블의 Dept\_ID 참조)**

**주어진 조건**

**- 3개의 부서를 Departments 테이블에 추가하세요.**

**- 각 부서에 최소 2명 이상의 직원을 Employees 테이블에 추가하세요.**

**- ID 값은 시퀀스를 사용하여 자동 증가하도록 구현하세요.**

**\*/**

**CREATE TABLE Departments (**

**Dept\_ID NUMBER PRIMARY KEY,**

**Dept\_Name VARCHAR2(50)**

**);**

**CREATE TABLE Employees (**

**Emp\_ID NUMBER PRIMARY KEY,**

**Emp\_Name VARCHAR2(50),**

**Dept\_ID NUMBER,**

**FOREIGN KEY (Dept\_ID) REFERENCES Departments(Dept\_ID)**

**);**

**-- 부서 ID 자동 증가 시퀀스 생성**

**CREATE SEQUENCE dept\_seq**

**START WITH 1 -- 1부터 시작**

**INCREMENT BY 1 -- 1씩 증가**

**NOMAXVALUE -- 최대값 제한 없음**

**NOCYCLE; -- 반복 없음**

**-- 직원 ID 자동 증가 시퀀스 생성**

**CREATE SEQUENCE emp\_seq**

**START WITH 1**

**INCREMENT BY 1**

**NOMAXVALUE**

**NOCYCLE;**

**-- 부서 데이터 삽입 (시퀀스 사용)**

**INSERT INTO Departments (Dept\_ID, Dept\_Name) VALUES (dept\_seq.NEXTVAL, '인사부');**

**INSERT INTO Departments (Dept\_ID, Dept\_Name) VALUES (dept\_seq.NEXTVAL, '개발부');**

**INSERT INTO Departments (Dept\_ID, Dept\_Name) VALUES (dept\_seq.NEXTVAL, '마케팅부');**

**-- 직원 데이터 삽입 (시퀀스 사용)**

**INSERT INTO Employees (Emp\_ID, Emp\_Name, Dept\_ID) VALUES (emp\_seq.NEXTVAL, '김철수', 1);**

**INSERT INTO Employees (Emp\_ID, Emp\_Name, Dept\_ID) VALUES (emp\_seq.NEXTVAL, '이영희', 1);**

**INSERT INTO Employees (Emp\_ID, Emp\_Name, Dept\_ID) VALUES (emp\_seq.NEXTVAL, '박민수', 2);**

**INSERT INTO Employees (Emp\_ID, Emp\_Name, Dept\_ID) VALUES (emp\_seq.NEXTVAL, '최지훈', 2);**

**INSERT INTO Employees (Emp\_ID, Emp\_Name, Dept\_ID) VALUES (emp\_seq.NEXTVAL, '정하늘', 3);**

**INSERT INTO Employees (Emp\_ID, Emp\_Name, Dept\_ID) VALUES (emp\_seq.NEXTVAL, '한서연', 3);**

**COMMIT;**

**select \* from departments;**

**select \* from employees;**

**<개별 실습 과제>**

**\* 실습 목표**

**- 두 개의 테이블을 직접 생성하고 Foreign key로 관계를 설정한다.**

**예) 도서관 - 책, 학교 - 학생**

**- 데이터를 삽입하여 관계형 데이터베이스의 개념을 이해한다.**

**- 시퀀스를 활용하여 ID 값을 자동 증가하도록 설정한다.**

**- 수동으로 데이터를 만들어본다.**

**- 두개의 테이블을 통합한 하나의 테이블도 만들어본다.**

**-> 하나의 테이블에 데이터를 직접 삽입해 보고 테이블이 분리됬을때와 비교해본다.**

**- 인공지능을 활용하여 데이터를 생성해본다.**

[**\\Desktop-rv7v5lo\공유폴더**](file:///\\Desktop-rv7v5lo\공유폴더) **// 공유폴더 설정하는 법**

[**hklee0724@gmail.com**](mailto:hklee0724@gmail.com)

**<2\_1\_DML\_basic\_practice.sql>**

**-- 전체 레코드 조회**

**-- select \* => 전체 컬럼명**

**-- \*을 사용할 때에는 신중해야 한다. 테이블내에 컬럼수, 레코드수를 고려해야한다.**

**select \* from Student\_Scores;**

**describe student\_scores;**

**-- select [컬럼명1], .. ,[컬럼명N] from [테이블명]**

**-- => select [열필터링] from [테이블명]**

**select student\_name from student\_scores;**

**select student\_name, enroll\_date from student\_scores;**

**select student\_name,kor\_score, enroll\_date from student\_scores;**

**select student\_name, kor\_score, eng\_score, math\_score, enroll\_date from student\_scores;**

**ALTER TABLE**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '신동엽', 82, 88, 95, TO\_DATE('01/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이하나', 78, 95, 92, TO\_DATE('02/02/2023', 'DD/MM/YYYY'));**

**-- 추가 13개 데이터**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '박민준', 87, 92, 85, TO\_DATE('15/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이예슬', 92, 89, 88, TO\_DATE('16/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김태희', 95, 86, 82, TO\_DATE('17/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '정희진', 82, 90, 87, TO\_DATE('18/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '김수민', 89, 85, 93, TO\_DATE('19/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '박준희', 76, 92, 79, TO\_DATE('20/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '최하윤', 93, 87, 86, TO\_DATE('21/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이가은', 93, 84, 91, TO\_DATE('22/02/2023', 'DD/MM/YYYY'));**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '이은서', 85, 90, 92, TO\_DATE('23/02/2023', 'DD/MM/YYYY'));**

**-- 중복이름**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '정민우', 85, 100, 92, TO\_DATE('23/02/2023', 'DD/MM/YYYY'));**

**-- 4글자 이름**

**INSERT INTO Student\_Scores (id, student\_name, kor\_score, eng\_score, math\_score, enroll\_date)**

**VALUES (student\_scores\_seq.NEXTVAL, '남궁은서', 85, 90, 100, TO\_DATE('23/02/2023', 'DD/MM/YYYY'));**

**CREATE TABLE Student\_Scores(**

**ID number, -- number: 숫자 처리 타입**

**student\_name varchar2(50), -- varchar2(문자열 길이): 문자 처리 타입**

**KOR\_SCORE number,**

**eng\_score number,**

**math\_score number,**

**enroll\_date date,**

**primary key (id)**

**);**

**SELECT \* FROM Student\_Scores WHERE student\_name like '\_민\_'; -- 가운데에 '민'자가 들어간 사람**

**-- 예) 정민우(O), 김민지(O), 김상민(X), 강민철구 (테스트)**

**SELECT \* FROM Student\_Scores WHERE student\_name like '\_\_수'; -- 예) 김철수(O), 김수철(X)**

**-- 성이 김으로 시작해서 수로 끝나는 사람**

**SELECT \* FROM Student\_Scores WHERE student\_name like '김%수'; -- 예) 김수, 김철수, 김철수상수**

**-- 아래 쿼리는 안된다. total\_score 참조가 안된다.**

**-- where 조건이 select 보다 먼저 내부적으로 수행된다.**

**select student\_name, kor\_score, eng\_score, math\_score,**

**(kor\_score + eng\_score + math\_score) as total\_score, enroll\_date**

**from student\_scores**

**-- select에서 사용한 별칭은 where에서 사용할 수 없음**

**where student\_name like '김%' or student\_name like '박%' and total\_score >= 270**

**order by enroll\_date desc;**

**-- date 타입은 YY/MM/DD 형식으로 보여줌**

**-- 데이터가 정의되지 않을 경우에는 null로 표시한다.**

**-- 한글 별칭 설정**

**select first\_name as 이름, salary 급여**

**from employees;**

**select first\_name, last\_name, salary, salary\*1.1 as 급여인상분**

**from employees;**

**-- 주의사항: DB 클라이언트 프로그램에서 조회용으로 한글 별칭 사용하는 것은 문제없으나**

**-- 조회의 결과를 다른 프로그램에서 사용할 경우 인코딩 문제가 발생할 수 있으므로**

**-- 가급적 별칭은 영문으로 작성한다.**

**select first\_name, last\_name, salary, salary\*1.1 as raised\_income**

**from employees;**

**-- 리터럴(literal)문자 스트링과의 연결**

**-- || : 연결 연산자**

**-- '문자를 표현하기 위해서 ''를 사용**

**select first\_name || ' ' || last\_name || '''s salary is $' || salary**

**as 사원상세정보**

**from employees;**

**-- 중복 행 제거**

**-- DISTINCT: 중복 레코드 제거**

**-- 부서ID 조회(중복제거)**

**select department\_id from employees; -- 중복 허용**

**select DISTINCT department\_id from employees; -- 중복 제거**

**-- 행 필터링(행의 제한) => Selection**

**-- 성이 'King'인 직원 정보 조회**

**select \* from employees**

**where last\_name = 'King';**

**-- 비교 연산**

**-- 급여가 15000 이상인 직원정보**

**select \* from employees**

**where salary >= 15000;**

**-- 날짜비교 간소화**

**-- 2004년 1월 30일에 고용된 직원 정보 조회**

**select \* from employees**

**where hire\_date = '04/01/30'; -- 저장되는 형식의 문자열로 조회시 암묵적 형변환이 일어난다.**

**-- between 연산자**

**-- between [시작조건] and [종료조건]**

**-- 시작조건과 종료조건은 숫자인 경우 모두 포함되는 조건**

**-- 급여가 10000 ~ 12000 인 직원**

**select \* from employees**

**where salary between 10000 and 12000;**

**select \* from employees**

**where salary >= 10000 and salary <= 12000;**

**-- 조건 Set**

**-- in 연산자**

**-- in (조건1, ... , 조건N) : 해당 조건을 or로 처리**

**-- 매니저ID가 101, 102, 103인 직원 이름, 급여, 매니저id 조회**

**-- 참고 반대 조건은 not in**

**select first\_name, salary, manager\_id**

**from employees**

**where manager\_id in (101,102,103);**

**select first\_name, salary, manager\_id**

**from employees**

**where manager\_id = 101 or manager\_id = 102 or manager\_id = 103;**

**-- LIKE 연산자**

**-- %: 아무 문자 0번에서 N번 매치**

**-- 직무ID가 IT로 시작하는 직원정보**

**select \* from employees**

**where job\_id like 'IT%';**

**-- 2003년도에 입사한 직원 정보 조회**

**select \* from employees**

**where hire\_date like '03%'; -- 날짜 타입에 대해서 like 연산을 사용하면 암묵적 형변환이 일어난다.**

**-- 1월에 입사한 직원 정보 조회**

**select \* from employees**

**where hire\_date like '\_\_/01%';**

**-- 1일에 입사한 직원 정보 조회**

**select \* from employees**

**where hire\_date like '\_\_/\_\_/01';**

**-- 이메일의 두 번째 문자가 'A'인 직원 정보**

**select \* from employees**

**where email like '\_A%';**

**-- null 데이터를 찾을 때 => 정의되지 않은 데이터(결측치)를 찾을 때**

**-- is null**

**-- 매니저 id가 null인 직원이름, 매니저id 조회**

**-- 참고] 반대조건은 not null**

**select first\_name, manager\_id from employees**

**where manager\_id is null;**

**-- 논리 연산자**

**-- and 연산**

**-- 직무 id가 IT\_PROG이고 급여가 5000 이상인 직원 정보**

**select \* from employees**

**where job\_id='IT\_PROG' and salary >= 5000;**

**-- Q] 직무 아이디가 'IT\_PROG' 또는 'FI\_MGR' 이고 급여가 6000 이상인**

**-- 직원의 이름, 직무id, 급여를 출력하세요.**

**-- 오라클에서 and 연산자의 우선순위가 or 보다 높다.**

**-- 따라서 의미를 분명하게 하기 위해서 () 연산자를 활용할 것**

**select first\_name, job\_id, salary**

**from employees**

**where (job\_id = 'IT\_PROG' or job\_id='FI\_MGR') and salary >=6000;**

**-- 데이터 정렬**

**-- order by (기본 오름차순 정렬)**

**-- select**

**-- from**

**-- where**

**-- order by <- 이 위치에서 사용**

**select first\_name, hire\_date**

**from employees**

**order by hire\_date;**

**select first\_name, hire\_date**

**from employees**

**order by hire\_date asc; -- 명시적으로 오름차순임을 지정**

**-- 내림차순 정렬 (desc)**

**select first\_name, hire\_date**

**from employees**

**order by hire\_date desc;**

**-- 열 별칭 응용**

**select first\_name, salary**

**from employees**

**order by salary;**

**select first\_name, salary\*1.2 -- <= salary\*1.2 : 급여의 20% 인상이 반영된 금액**

**from employees**

**order by salary\*1.2;**

**select first\_name, salary\*1.2 as 인상된급여 -- <= salary\*1.2 : 급여의 20% 인상이 반영된 금액**

**from employees**

**order by 인상된급여;**

**select first\_name, salary\*1.2 as raise\_salary -- <= salary\*1.2 : 급여의 20% 인상이 반영된 금액**

**from employees**

**order by raise\_salary;**

**--1. 성이 'Bull'인 사람의 이름과 성을 출력**

**select first\_name, last\_name from employees**

**where last\_name = 'Bull';**

**select email from employees;**

**--2. 이메일에 'LEE'를 포함하는 사원의 급여는?**

**select salary from employees**

**where email like '%LEE%';**

**-- \* 함수**

**-- - 단일행 함수: 단일 행에서만 적용 가능하고 행별로 하나의 결과를 리턴**

**-- - 다중행 함수: 복수의 행을 조작하여 행의 그룹당 하나의 결과를 리턴 (예: count())**

**-- > 단일행 함수: 문자함수, 숫자함수, 날짜 함수, 변환함수, 일반함수**

**-----------------------------------**

**-- 문자함수**

**-- dual 테이블: 오라클 자체에서 값을 확인하기 용도로 제공하는 테이블**

**select \* from dual;**

**-- initcap: 단어를 기준으로 첫 문자만 대문자로 나머지는 소문자로 변환하는 함수**

**select initcap(email) from employees;**

**select count(email), initcap(email) from employees;**

**<subquery, 서브함수>**

**-- lower: 전체문자를 소문자로 만드는 함수**

**select lower('DATABASE SPECIALIST') from dual;**

**-- database specialist**

**-- upper: 전체문자를 대문자로 만드는 함수**

**select upper('database specialist') from dual;**

**-- DATABASE SPECIALIST**

**-- length: 문자열의 길이를 반환하는 함수**

**select length('database specialist') from dual;**

**-- 19**

**select length('데이터베이스 전문가') from dual;**

**-- 10: 한글인 경우 문자를 1로 카운트**

**-- concat: 두 문자열을 연결하여 반환**

**select concat('데이터베이스', '전문가') from dual;**

**-- 데이터베이스전문가**

**select concat('데이터베이스 ', '전문가') from dual;**

**select concat('데이터베이스', ' 전문가') from dual;**

**-- 데이터베이스 전문가**

**-- substr: substr([문자열],[시작인덱스],[시작인덱스로부터 Offset])**

**-- 문자열을 기준으로 시작인덱스에서 끝 인덱스의 문자열 반환**

**-- 주의사항: 문자열의 시작인덱스 1**

**select substr('database specialist',5,8) from dual;**

**select substr('database specialist',0,8) from dual;**

**select substr('database specialist',1,8) from dual;**

**select substr('database specialist',10,7) from dual;**

**-- instr: instr([문자열],[찾고자 하는 문자열]) => 찾고자하는 문자열이 있다면 해당 문자열의 인덱스**

**select instr('데이터베이스 전문가','전') from dual;**

**-- 8**

**select instr('데이터베이스 전문가','정') from dual; => 찾고자하는 문자열이 없으면 0 반환**

**-- 0**

**-- rpad/lpad: 주어진 자릿수 만큼 오른쪽(rpad)/왼쪽(lpad)에 지정한 문자열을 채운다.**

**-- rpad/lpad([문자열],[채울자리수],[채울문자])**

**select rpad('James',10,'\*') from dual;**

**-- James\*\*\*\*\***

**select rpad('홍길동',10,'\*') from dual;**

**-- 홍길동\*\*\*\*\*\*\* (X)**

**-- 홍길동\*\*\*\* : 한글인 경우 한글자당 2로 계산**

**select lpad('홍길동',10,'\*') from dual;**

**-- \*\*\*\*홍길동**

**-- Q] 직원테이블에서 이름 10자리중 나머지 오른쪽을 '-'로 채우고**

**-- 급여 10자리중 나머지 왼쪽은 '\*'로 채워서 출력**

**select rpad(first\_name, 10, '-') as name, lpad(salary, 10, '\*') as sal**

**from employees;**

**-- ltrim/rtrim/trim: 문자열을 기준으로 제거할 문자를 각각 왼쪽, 오른쪽, 양쪽으로 제거**

**-- ltrim/rtrim([문자열],[제거할문자열]) 제거할 문자열 기본값은 공백문자**

**-- trim([문자열]) 양쪽 공백 문자 제거**

**select ltrim(' 데이터베이스 전문가 ') as name from dual;**

**select ltrim(' 데이터베이스 전문가') as name from dual;**

**select ltrim('####데이터베이스####전문가####','#') as name from dual;**

**-- 데이터베이스####전문가####**

**select rtrim(' 데이터베이스 전문가 ') as name from dual;**

**select rtrim('####데이터베이스####전문가####','#') as name from dual;**

**select trim(' 데이터베이스 전문가 ') as name from dual;**

**-- 데이터베이스 전문가**

**select trim('####데이터베이스####전문가####','#') as name from dual; -- trim 함수 매개변수와 미일치**

**-- replace**

**select replace('JavaSpecailist','Java','BigData') from dual;**

**-- BigDataSpecailist**

**select replace('Java Specailist',' ','') from dual; -- 공백 제거용으로 replace 사용**

**select replace('데이터베이스 전문가',' ','') as name from dual;**

**-- 데이터베이스전문가**

**-- translate([원본 문자열],[매칭문자열],[변환문자열])**

**select translate('1234abcd','abcd','ABC') from dual;**

**-- 1234ABC**

**select translate('hello world!!!','hw','HW') from dual;**

**-- Hello World!!!**

**select translate('hello world!!!','!','?') from dual;**

**-- hello world???**

**select translate('$100', '$', ' ') from dual;**

**-- 100**

**select translate('$100', '$', '') from dual; -- 변환문자열은 최소한 매칭문자열 길이와 일치해야한다.**

**-- null**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**-- Concat함수를 사용해서 first\_name, last\_name 연결하기 (사이에 공백 포함)**

**-- STEP1]**

**select concat(first\_name,' ') from employees;**

**-- STEP2]**

**select concat( concat(first\_name,' '), last\_name ) from employees;**

**num1, num2 => 파라메터, 매개변수**

**int sum(int num1, int num2){**

**return num1+num2;**

**}**

**sum(1,2); => 1,2 => 인자(Argument)**

**-- 마지막 글자 3자리가 필요한 ID 정보일경우**

**select substr('dfjlsjdflsjdflsjdlfsdsklfsdff 984',-3) from dual;**

**select substr('dfjlsjdflsjdflsf 744',-3) from dual;**

**SELECT TRANSLATE('Nike@Air-Max#2024!', '!@#$', '') FROM dual;**

**-- 반올림: round**

**select round(45.962,2),round(45.962,0),round(45.962,-1)**

**from dual;**

**-- 올림: ceil (ceil 함수는 소수점 이하를 올리는 기능만 제공한다.)**

**select ceil(7.3)from dual;**

**-- 8**

**-- 버림: floor (floor함수는 소수점 이하를 버리는 기능만 제공한다.)**

**select floor(7.3)from dual;**

**-- 7**

**-- select floor(7.33434343, 2)from dual;**

**-- 절삭: trunc([데이터],[자리수]) 자리수 만큼 데이터의 값을 절삭**

**-- 340,343,056,234,656 원**

**select trunc(45.923,2), trunc(45.923), trunc(45.923, -1) from dual;**

**-- to\_date([날짜스트링],[포멧])**

**select to\_date('2023-06-01','YYYY-MM-DD') from dual;**

**-- trunc 함수는 날짜타입에도 적용가능하다.**

**-- 매월 첫째날 구하기**

**select trunc(to\_date('2023-06-23','YYYY-MM-DD'),'MM') from dual;**

**-- 새해 첫째날 구하기**

**select trunc(to\_date('2023-06-23','YYYY-MM-DD'),'YYYY') from dual;**

**-- 23/01/01**

**select trunc(to\_date('2023-06-23','YYYY-MM-DD'),'DD') from dual;**

**-- 23/06/23 : 날짜 이후로는 절삭할 단위가 없기 때문에 현재 날짜를 반환**

**-- 그 주의 첫번째 일요일 반환**

**select trunc(to\_date('2023-06-23','YYYY-MM-DD'),'DAY') from dual;**

**-- 절대값**

**select abs(-1000) from dual;**

**-- 제곱값**

**select power(2,2) from dual;**

**--4**

**select power(2,3) from dual;**

**--8**

**-- 제곱근**

**select sqrt(100) from dual;**

**-- 나머지**

**select mod(10,3) from dual;**

**-- 1**

* **4.1 문자함수**
* **4.2.1 숫자함수**
* **4.3.1 날짜함수**
* **4.5 변환함수**
* **4.6 그룹함수 함수 전 최종연습 문제**

**문제 6번을 아래와 같이 수정합니다.**

**--문제 6:**

**--employees 테이블에서 각 사원의 입사일(hire\_date)이 2005년 6월 전이면 '창립멤버', 2010년 이후이면 '신규멤버'으로 표시하여 조회하세요.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**--employees 테이블에서 각 사원의 보너스(commission\_pct)의 등급을 표시하세요.**

**-- 등급조건**

**-- 0초과 0.2미만: 최소등급**

**-- 0.2이상 0.3미만 일반등급**

**-- 0.3이상 0.4미만 우수등급**

**-- 0.4이상 최고등급**

**<5.1 그룹함수>**

**/\***

**그룹함수란?**

**복수의 행을 조작하여 행의 그룹당 하나의 결과를 리턴하는 데이터 분석/집계 함수**

**주요함수**

**count, sum, avg, min, max, variance, stddev**

**\*/**

**select count(salary), sum(salary), avg(salary), min(salary), max(salary),**

**variance(salary), stddev(salary)**

**from employees;**

**-- 날짜인 경우 min(): 가장 오래된 날짜/max(): 가장 최근 날짜.**

**select min(hire\_date), max(hire\_date)**

**from employees;**

**-- 문자인경우:min(): 문자숫서가 가장 앞인 경우/max(): 문자 숫서가 가장 뒤인 경우**

**select min(first\_name), max(first\_name)**

**from employees;**

**-- Q] 가장 큰 급여액은?**

**select max(salary) from employees;**

**-- Q] 사원들의 급여의 총합, 평균, 표준편차, 그리고 분산을 구하세요.**

**-- 단 소수점 이하 두 번쨰 자리까지 반올림할것**

**select sum(salary) 합계,**

**round(avg(salary),2) 평균,**

**round(stddev(salary),2) 표준편차,**

**round(variance(salary),2) 분산**

**from employees;**

**-- count 함수 사용시 주의사항**

**-- count(\*) 함수는 null인 값도 센다.**

**select commission\_pct from employees;**

**select count(commission\_pct) from employees;**

**select**

**round(sum(salary\*commission\_pct),2) 총인상금액,**

**round(sum(salary\*commission\_pct) / count(\*) ,2) 인상분평균\_by\_count,**

**round(avg(salary\*commission\_pct),2) 인상분평균\_by\_avg**

**-- avg함수는 값이 있는 레코드를 기준으로 하여 평균값**

**from employees;**

**union all까지 실습**

**15분 드립니다.**

**drop table visit\_statistics;**

**CREATE TABLE visit\_statistics (**

**id NUMBER PRIMARY KEY, -- 고유 ID**

**visit\_date DATE NOT NULL, -- 방문한 날짜**

**page\_category VARCHAR2(50), -- 페이지 카테고리 (예: 홈, 제품, 블로그)**

**device\_type VARCHAR2(20), -- 접속 기기 (예: 모바일, PC, 태블릿)**

**visit\_count NUMBER -- 방문 횟수**

**);**

**CREATE SEQUENCE visit\_statistics\_seq**

**START WITH 1**

**INCREMENT BY 1**

**NOCACHE**

**NOCYCLE;**

**INSERT INTO visit\_statistics (id, visit\_date, page\_category, device\_type, visit\_count) VALUES**

**(visit\_statistics\_seq.NEXTVAL, TO\_DATE('2024-05-01', 'YYYY-MM-DD'), '홈', '모바일', 150);**

**SELECT visit\_date, page\_category, device\_type, SUM(visit\_count) AS total\_visits**

**FROM visit\_statistics**

**GROUP BY ROLLUP (visit\_date, page\_category, device\_type)**

**ORDER BY visit\_date, page\_category, device\_type;**

**SELECT visit\_date, page\_category, device\_type, SUM(visit\_count) AS total\_visits**

**FROM visit\_statistics**

**GROUP BY CUBE (visit\_date, page\_category, device\_type)**

**ORDER BY visit\_date NULLS LAST, page\_category NULLS LAST, device\_type NULLS LAST;**

**INSERT INTO visit\_statistics (id, visit\_date, page\_category, device\_type, visit\_count) VALUES**

**(visit\_statistics\_seq.NEXTVAL, TO\_DATE('2024-05-01', 'YYYY-MM-DD'), '제품', '모바일', 120);**

**<Join의 종류>**

**-- 조인의 종류**

**-- 관점에 따라 여러가지로 나눌 수 있으나 주로 아래 관점으로 분류**

**-- 일반 분류: Inner Join(내부조인) / Outer Join(외부조인) / Cross Join(교차조인) => 조인의 범위에 따라 분류**

**-- 표준 여부: Ansi Join(안시 조인) vs Oracle Join(오라클 조인) => Ansi 조인: SQL 표준 문법**

**-- 조인 구문: Explicit Join(명시적 조인) vs Implicit Join(암시적 조인)**

**-- 조인 방향:(Inner/Left/Right/Full Outer/합집합(Union)/차집합(Difference)/교집합(Intersection)**

**-- 복잡성: Simple Join(단순 조인) / Complex Join(복합 조인)**

**drop table employees;**

**-- 테이블 A 생성**

**CREATE TABLE employees (**

**employee\_id number PRIMARY KEY,**

**employee\_name VARCHAR2(50),**

**department\_id number**

**);**

**-- 테이블 A 데이터 삽입**

**INSERT INTO employees (employee\_id, employee\_name, department\_id)**

**VALUES (1, '홍길동', 101);**

**INSERT INTO employees (employee\_id, employee\_name, department\_id)**

**VALUES (2, '김철수', 102);**

**INSERT INTO employees (employee\_id, employee\_name, department\_id)**

**VALUES (3, '이영희', 101);**

**INSERT INTO employees (employee\_id, employee\_name, department\_id)**

**VALUES (4, '박민지', 102);**

**INSERT INTO employees (employee\_id, employee\_name, department\_id)**

**VALUES (5, '최영호', 103);**

**INSERT INTO employees (employee\_id, employee\_name, department\_id)**

**VALUES (6, '나사장', 201);**

**drop table departments;**

**-- 테이블 B 생성**

**CREATE TABLE departments (**

**department\_id number PRIMARY KEY,**

**department\_name VARCHAR2(50)**

**);**

**-- 테이블 B 데이터 삽입**

**INSERT INTO departments (department\_id, department\_name)**

**VALUES (101, '개발');**

**INSERT INTO departments (department\_id, department\_name)**

**VALUES (102, '영업');**

**INSERT INTO departments (department\_id, department\_name)**

**VALUES (103, '마케팅');**

**INSERT INTO departments (department\_id, department\_name)**

**VALUES (104, '재무');**

**select \* from employees;**

**select \* from departments;**

**select department\_id from employees;**

**select distinct department\_id from employees order by department\_id;**

**--101**

**--102**

**--103**

**--201**

**select distinct department\_id from departments;**

**--101**

**--102**

**--103**

**--104**

**<7.2 연습문제 >**

**-- Inner Join (ANSI Style): 조인의 결과는 두 집합에서 조인되는 키의 교집합범위에서 일어난다.**

**-- 문제] 각 직원의 id, 이름, 부서 id, 부서명을 검색하세요.**

**-- 단, 직원이 부서에 속한 경우만 조회하세요.**

**-- Left Join (ANSI Style): Left 테이블을 기준으로 병합(그렇기 때문에 Left 테이블의 조인키 누락은 없다.)**

**-- 조인시 첫번째로 지정하는 테이블을 Left, 두번?로 지정하는 테이블이 Right**

**-- 문제] 각 직원의 ID(employee\_id), 이름(employee\_name), 부서 ID(department\_id), \*\*부서명(department\_name)\*\*을 검색하세요.**

**--단, 직원이 소속된 부서가 없는 경우에도 모든 직원을 포함하여 조회하세요.**

**-- right join (ANSI Style)**

**--문제:**

**--각 직원의 ID(employee\_id), 이름(employee\_name), 부서 ID(department\_id), \*\*부서명(department\_name)\*\*을 검색하세요.**

**--단, 직원이 속하지 않은 부서도 포함하여 조회하세요.**

**<…>**

**-- 서브쿼리: 쿼리의 각 절에 의존관계를 활용하여 중첩된 쿼리를 작성해 하나의 검색결과를 자동으로 완성**

**-- 종류:**

**-- where, having: 일반 서브쿼리,**

**-- from: 인라인뷰(Inline View),**

**-- select: 스칼라 서브쿼리(Scalar Subquery)**

**-- Nancy의 급여보다 많은 급여를 받는 사원의 이름과 급여를 출력**

**-- Step1] Nancy의 급여를 구한다.**

**-- Step2] Nancy의 급여(12008)보다 많은 급여를 받는 사원의 이름과 급여를 출력**

**-- 서브쿼리의 유형:**

**-- + 단일행 서브쿼리(where, having절에 >, >=, <, <=, = 단일 값을 비교하는 연산에 주로 사용된다.),**

**-- + 다중행 서브쿼리, 다중컬럼 서브쿼리**

**-- 서브쿼리 활용(단일행 서브쿼리: 하나의 값을 대체)**

**-- WHERE 뒤에 값(리터럴)에 의한 비교는 하나의 값과 매치가 되어야 한다. 복수개의 값과 매치를 하려고 한다면**

**-- 에러를 발생하게 된다.**

**-- 예)**

**-- 최고 연봉을 받는 직원 조회**

**-- 서브쿼리와 큰 성능 차이를 보이지 않으며 서브쿼리 사용시 코드가 더 간결해 졌다.**

**-- 따라서 단일 조건을 찾을 경우 조인 보다 비교적 쉽고 간편하게 쿼리 작성이 가능하다.**

**--부서별 평균 급여가 전체 평균 급여보다 높은 부서 조회**

**< >**

**-- 다중행 서브쿼리**

**-- ANY/ALL**

**-- ANY: 하나라도 만족하는 조건 / ALL: 모든 것을 만족하는 조건**

**-- > ANY: 가장 작은 값보다 큰 조건(MIN 조건)**

**-- < ANY: 가장 큰 값보다 작은 조건(MAX 조건)**

**-- > ALL: 가장 큰 값보다 큰 조건(MAX 조건)**

**-- < ALL: 가장 큰 값보다 작은 조건(MIN 조건)**

**--4800**

**--9500**

**--6800**

**-- 다중행 서브쿼리 Min 조건**

**-- 그룹함수의 결과를 where의 검색조건으로 활용**

**-- Q] 20번 부서에 근무하는 사원의 평균보다 많은 급여를 받는 사원의 이름과 급여를 출력**

**-- Q] 직원 테이블에서 가장 높은 급여를 받는 직원의 모든 정보**

**-- 다중행 서브쿼리 Max 조건**

**-- IN 연산을 활용한 다중행 서브쿼리**

**-- In 연산은 where 뒤에 조건을 여러개의 값으로 매치시킬수 있다.**

**-- 따라서 in 연산 안에 다중행을 리턴하는 서브쿼리를 작성할 수 있다.**

**-- 다중 컬럼 서브쿼리: 다중 컬럼 값을 반환하는 서브쿼리**

**-- 부서별 최고 급여를 받는 직원 정보**

**--Q] 부서별 최고급여 받는 사람들 이름이랑 최고급여만 조회**

**-- select 절에서 서브쿼리 사용 -> Scalar Sub Query**

**-- Select절에서 Join의 조건을 명시하여 Join할 대상 범위를 줄임**

**-- 상황에 따라서 Join 보다 좋은 성능을 보이기 때문에 사용 (단, 항상 성능이 좋은 것은 아님)**

**-- 하지만 어떤 식으로 쿼리를 작성하여도 DBMS Optimizer가 최적의 성능을 낼 수 있도록 쿼리를 최적화한다.**

**-- 인라인 뷰**

**-- from 절에 적용하는 서브쿼리. from 절에는 테이블 또는 뷰가 올 수 있음**

**-- 서브쿼리도 독립적인 뷰라고 볼수 있기때문에 인라인 뷰라 칭함**

**-- ? 급여를 가장 많이 받는 상위 10위까지 사원의 이름과 급여를 출력**

**-- 시도1: 급여순으로 정렬을 한다.**

**-- 시도2: 의사열로 Top 10을 필터링해본다.**

**-- 급여 Top10으로 필터링 되지 않는다.**

**-- 시도3: inline**

**-- from에 정렬한 테이블 결과를 가져오고 추후에 Top 10을 필터링한다.**

**-- order by 절이 가장 마지막에 수행되기 때문이다.**

**10. 트랜잭션. SQL**

**09. 뷰. SQL**

**08. 서브쿼리 연습문제. SQL**

**이후부터는 2\_pandas\_basic 으로 진도가 넘어감.**

**<2\_Pandas\_basic >**

**DB/WEB PROGRAMING/ ASP( ) /**