

# 3장. 관계(Releation)

오라클 DB – Releation

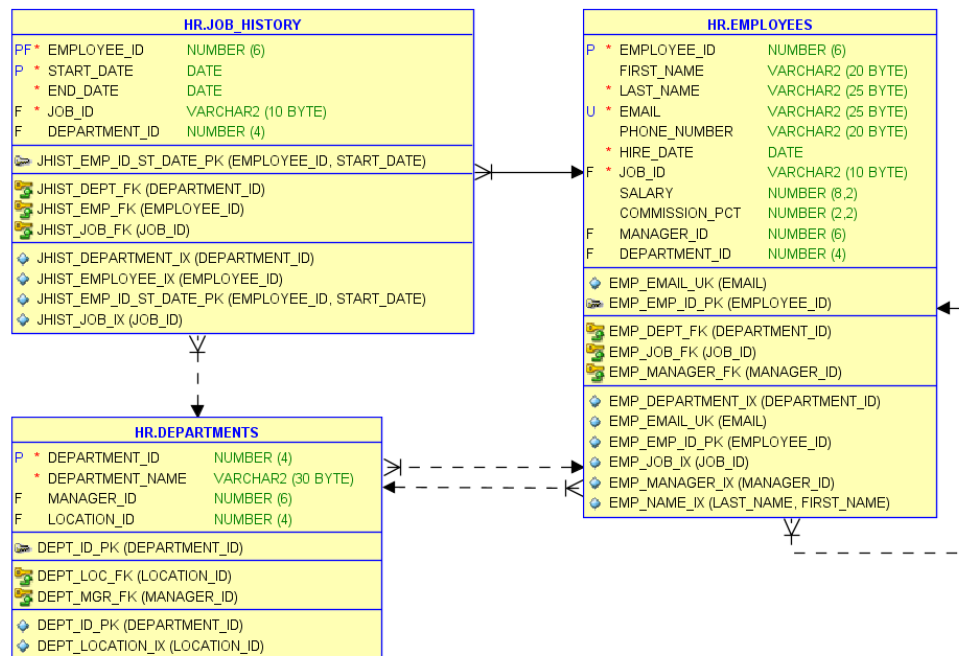


# 개체 관계도(ERD)

## 개체 관계도(ERD-Entity Relation Diagram)

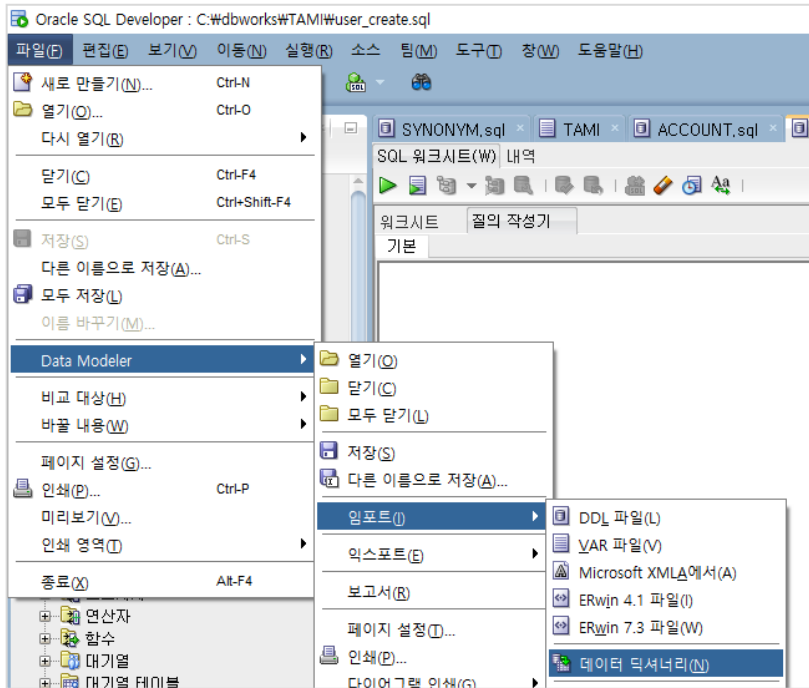
ERD는 ER 다이어그램이라고 부르는데 E는 개체라는 의미이고, R은 관계라는 의미이다.

개체는 정보를 저장하고 관리하기 위한 집합이자 식별 가능한 것이다.



# 개체 관계도(ERD)

## 개체 관계도(ERD-Entity Relation Diagram)



원하는 데이터베이스에 접속하기 위한 데이터베이스 접속을 선택합니다.  
목록이 비어 있으면 "추가" 단추를 사용하여 생성하십시오.

### 1. 데이터베이스에 접속

### 2. 스키마/데이터베이스 선택

### 3. 임포트할 객체 선택

### 4. 디자인 생성

#### 접속 이름

BOOK  
HRdb  
HRDB-XE  
JWEBDB  
OracleORCL  
OracleXE  
SYS  
TAMI

#### 접속 세부정보

book\_ex@//localhost:1521/orcl  
HR@//localhost:1521/orcl  
HR@//localhost:1522/xs  
jweb@//localhost:1522/xs  
system@//localhost:1521/orcl  
system@//localhost:1522/xs  
system@//localhost:1521/orcl  
TAMI@//localhost:1521/orcl

# 개체 관계도(ERD)

선택됨	스키마
<input type="checkbox"/>	ANONYMOUS
<input type="checkbox"/>	APEX_030200
<input type="checkbox"/>	APEX_PUBLIC_USER
<input type="checkbox"/>	APPQOSSYS
<input type="checkbox"/>	BI
<input type="checkbox"/>	CTXSYS
<input type="checkbox"/>	DBSNMP
<input type="checkbox"/>	DIP
<input type="checkbox"/>	EXFSYS
<input type="checkbox"/>	FLWS_FILES
<input checked="" type="checkbox"/>	HR
<input type="checkbox"/>	IX
<input type="checkbox"/>	MDDATA
<input type="checkbox"/>	MDSYS
<input type="checkbox"/>	MGMT_VIEW
<input type="checkbox"/>	MINA
<input type="checkbox"/>	OE
<input type="checkbox"/>	OLAPSYS
<input type="checkbox"/>	ORACLE_OCM
<input type="checkbox"/>	ORDDATA
<input type="checkbox"/>	ORDPLUGINS
<input type="checkbox"/>	ORDSYS

☒ ☐ 필터:  ☐ 모두 선택됨

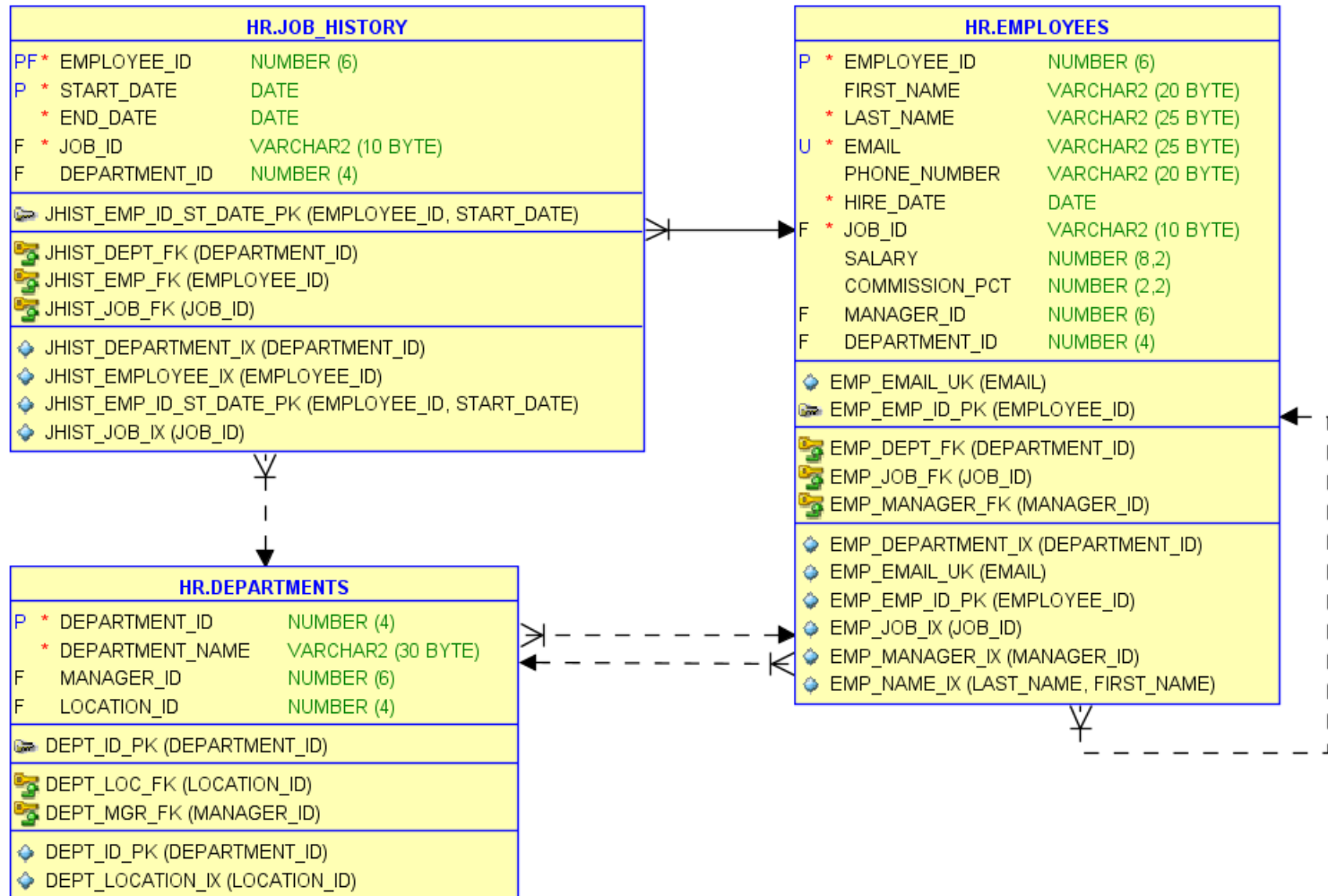
임포트 대상:   
 Relational\_1 ☐ 대상 모델 교체 Oracle Database 11g

임포트하려는 객체를 선택합니다.

선택됨	스키마	객체 이름
<input type="checkbox"/>	HR	COUNTRIES
<input checked="" type="checkbox"/>	HR	DEPARTMENTS
<input checked="" type="checkbox"/>	HR	EMPLOYEES
<input type="checkbox"/>	HR	EX2_4
<input type="checkbox"/>	HR	EX3_1
<input type="checkbox"/>	HR	EX4
<input type="checkbox"/>	HR	JOBS
<input checked="" type="checkbox"/>	HR	JOB_HISTORY
<input type="checkbox"/>	HR	LOCATIONS
<input type="checkbox"/>	HR	REGIONS



# 개체 관계도(ERD)



# 개체 관계도(ERD)

기본 키  
(primary key)

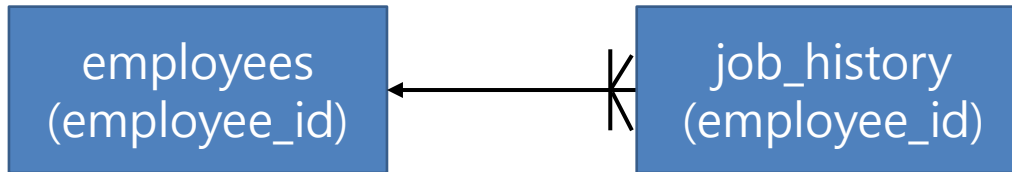
고유 키  
(unique key)

외래 키  
(foreign key)

HR.EMPLOYEES		
P *	EMPLOYEE_ID	NUMBER (6)
	FIRST_NAME	VARCHAR2 (20 BYTE)
	* LAST_NAME	VARCHAR2 (25 BYTE)
U *	EMAIL	VARCHAR2 (25 BYTE)
	PHONE_NUMBER	VARCHAR2 (20 BYTE)
	* HIRE_DATE	DATE
F *	JOB_ID	VARCHAR2 (10 BYTE)
	SALARY	NUMBER (8,2)
	COMMISSION_PCT	NUMBER (2,2)
F	MANAGER_ID	NUMBER (6)
F	DEPARTMENT_ID	NUMBER (4)



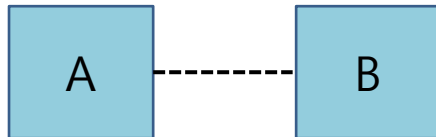
# 개체 관계도(ERD)



직원은 여러 개의 업무(경력)를 가질 수 있다. -> 1 대 다(多)



**실선** : 필수 관계, B가 존재하려면 A가 반드시 존재해야 함



**점선** : 선택적 관계, B는 A가 없어도 존재할 수 있다.



# 조인(JOIN)

## 조인이란?

조인은 한 개 이상의 테이블과 테이블을 서로 연결하여 사용하는 기법을 말한다.  
동등조인, 외부 조인, 자체 조인등이 있다.

조인 기법	개념
동등 조인(equi join)	조인 조건이 정확히 일치하는 경우에 결과를 출력
외부 조인(outer join)	조인 조건이 정확히 일치하지 않아도 모든 결과를 출력
자체 조인(self join)	자체 테이블에서 조인하고자 할 때 사용

## 문법 규칙

```
SELECT 테이블이름1.열 이름1, 테이블이름2.열 이름2
FROM 테이블 이름 1, 테이블 이름2
WHERE 테이블 이름 1.열 이름1 = 테이블 이름2.열 이름 2
```

두 테이블의 열이 갖고 있는 데이터 값을 논리적으로 연결



# 조인(JOIN)

## 동등 조인(equi join or inner join)

양쪽 테이블에서 조인 조건이 일치하는 행만 가져오는 조인으로 기본키와 외래키의 관계를 이용하여 조인하기도 하고 키가 아니더라도 다양한 조건으로 조인할 수 있다.

```
-- 동등 조인 (employees와 departments 테이블은 department_id로 연결되어 있다.)  
SELECT a.employee_id, a.last_name, b.department_id, b.department_name  
FROM employees a, departments b  
WHERE a.department_id = b.department_id;  
--AND a.employee_id = 101;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200	Whalen	10	Administration
201	Hartstein	20	Marketing
202	Fay	20	Marketing
114	Raphaely	30	Purchasing
115	Khoo	30	Purchasing
116	Baida	30	Purchasing
117	Tobias	30	Purchasing
118	Himuro	30	Purchasing



# 조인(JOIN)

## 동등 조인(equi join or inner join)

```
-- employee_id가 101인 직원의 job_history 출력
SELECT a.first_name, a.last_name, b.*
FROM employees a, job_history b
WHERE a.employee_id = b.employee_id
AND a.employee_id = 101;
```

FIRST_NAME	LAST_NAME	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
Neena	Kochhar	101	1997/09/21	2001/10/27	AC ACCOUNT	110
Neena	Kochhar	101	2001/10/28	2005/03/15	AC MGR	110

```
-- 각 직원이 어느 부서에 속하는지와 부서의 소재지가 어디인지 조회
SELECT a.employee_id, a.department_id, b.department_name, c.location_id, c.city
FROM employees a, departments b, locations c
WHERE a.department_id = b.department_id
AND b.location_id = c.location_id;
```

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
100	90	Executive	1700	Seattle
101	90	Executive	1700	Seattle
102	90	Executive	1700	Seattle
103	60	IT	1400	Southlake
104	60	IT	1400	Southlake



# 조인(JOIN)

## 외부 조인(equi join or inner join)

양쪽 테이블에서 데이터 값이 일치하지 않는 경우에도 모든 데이터를 연결하기

```
SELECT COUNT(*) 조인된건수
FROM employees A, departments B
WHERE A.department_id = B.department_id;
```

조인...
1 106

```
-- department_id가 NULL인 직원 찾기
SELECT employee_id, first_name, department_id
FROM employees
WHERE department_id IS NULL;
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
178	Kimberely	(null)

```
-- 양 테이블을 department_id로 외부 조인하여 department_id가 null인 값 출력
SELECT a.employee_id, a.last_name, b.department_id, b.department_name
FROM employees a, departments b
WHERE a.department_id = b.department_id(+);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
109	Faviet	100	Finance
108	Greenberg	100	Finance
206	Gietz	110	Accounting
205	Higgins	110	Accounting
178	Grant	(null)	(null)

# 조인(JOIN)

locations 테이블이 부모이고, departments 테이블이 자식테이블인 경우

```
INSERT INTO departments (department_id, department_name, manager_id, location_id)
VALUES (273, 'Sample Dept', 200, 1110);
```

참조값 없음

오류 보고 -

ORA-02291: integrity constraint (HR.DEPT\_LOC\_FK) violated - parent key not found

LOCATION_ID	STREET_ADDRESS
1	10001297 Via Cola di Rie
2	110093091 Calle della Testa
3	12002017 Shinjuku-ku
4	13009450 Kamiya-cho
5	14002014 Jabberwocky Rd

LOCATIONS 테이블

DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10 Administration	200	1700
20 Marketing	201	1800
30 Purchasing	114	1700
40 Human Resources	203	2400
50 Shipping	121	1500

DEPARTMENTS 테이블



# 데이터 베이스 모델링

## 데이터베이스 스키마와 상태

데이터베이스 스키마는 잘 변하지 않지만, 데이터베이스 상태는 수시로 데이터가 입력되고 삭제되므로 자주 변한다. 그래서 인스턴스라고 부르기도 함

### 학과

학과명	전화번호	사무실 위치
소프트웨어학과	02-1234-1234	B동 3층
전기전자공학과	02-1234-4567	B동 4층
화학공학과	02-1234-5678	B동 5층

### 학생

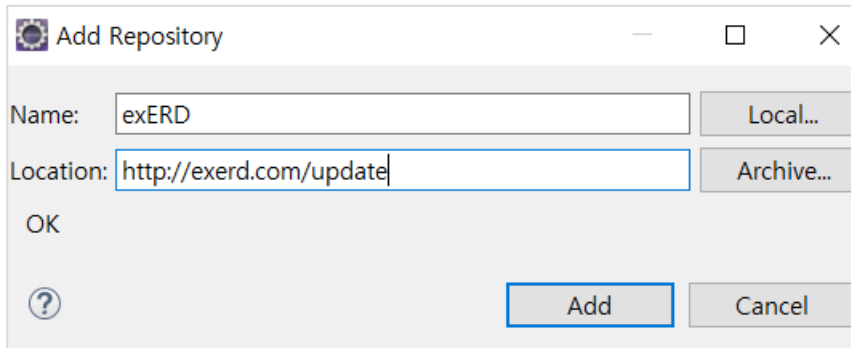
학번	학생 이름	나이	성별	주소
20211234	이강	22	여	서울시 구로구
20211235	박대양	25	남	서울시 성동구
20211236	한비야	23	여	서울시 강남구
20211237	정산들	27	남	경기도 수원시



# 데이터베이스 모델링

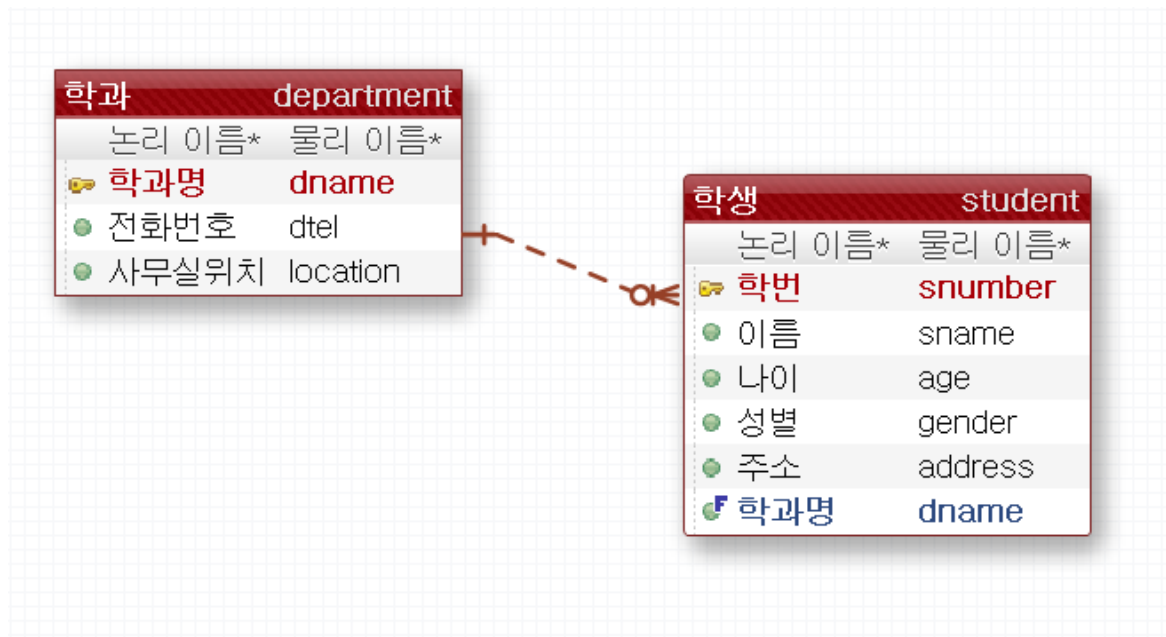
## exERD – 데이터베이스 모델링 소프트웨어

이클립스 – help – install NewSoftware – add 버튼 클릭



# 데이터베이스 모델링

## 개체 관계도(ERD-Entity Relation Diagram)



# 데이터베이스 구축하기

## 대학교 업무 관계

```
-- 학과 테이블
CREATE TABLE department(
    dname VARCHAR2(30),
    dtel VARCHAR2(20),
    location VARCHAR2(20),
    PRIMARY KEY(dname)
);

-- 학생 테이블
CREATE TABLE student(
    snumber NUMBER(10),
    sname VARCHAR2(20),
    age NUMBER(3),
    gender VARCHAR2(4),
    address VARCHAR2(30),
    dname VARCHAR2(30),
    pnumber NUMBER(3),
    PRIMARY KEY(snumber),
    CONSTRAINT FK_dept_std FOREIGN KEY(dname)
    REFERENCES department(dname)
);
```



# 데이터베이스 구축하기

## 대학교 업무 관계

-- 학과 추가 --

```
INSERT INTO department VALUES ('소프트웨어학과', '02-1234-1234', 'B동 3층');
```

```
INSERT INTO department VALUES ('화학공학과', '02-1234-4567', 'B동 4층');
```

```
INSERT INTO department VALUES ('전기전자공학과', '02-1234-5678', 'B동 5층');
```

-- 학생 추가 --

```
INSERT INTO student VALUES (20211234, '이강', 22, '여', '서울시 구로구', '소프트웨어학과');
```

```
INSERT INTO student VALUES (20211235, '박대양', 25, '남', '서울시 성동구', '전기전자공학과');
```

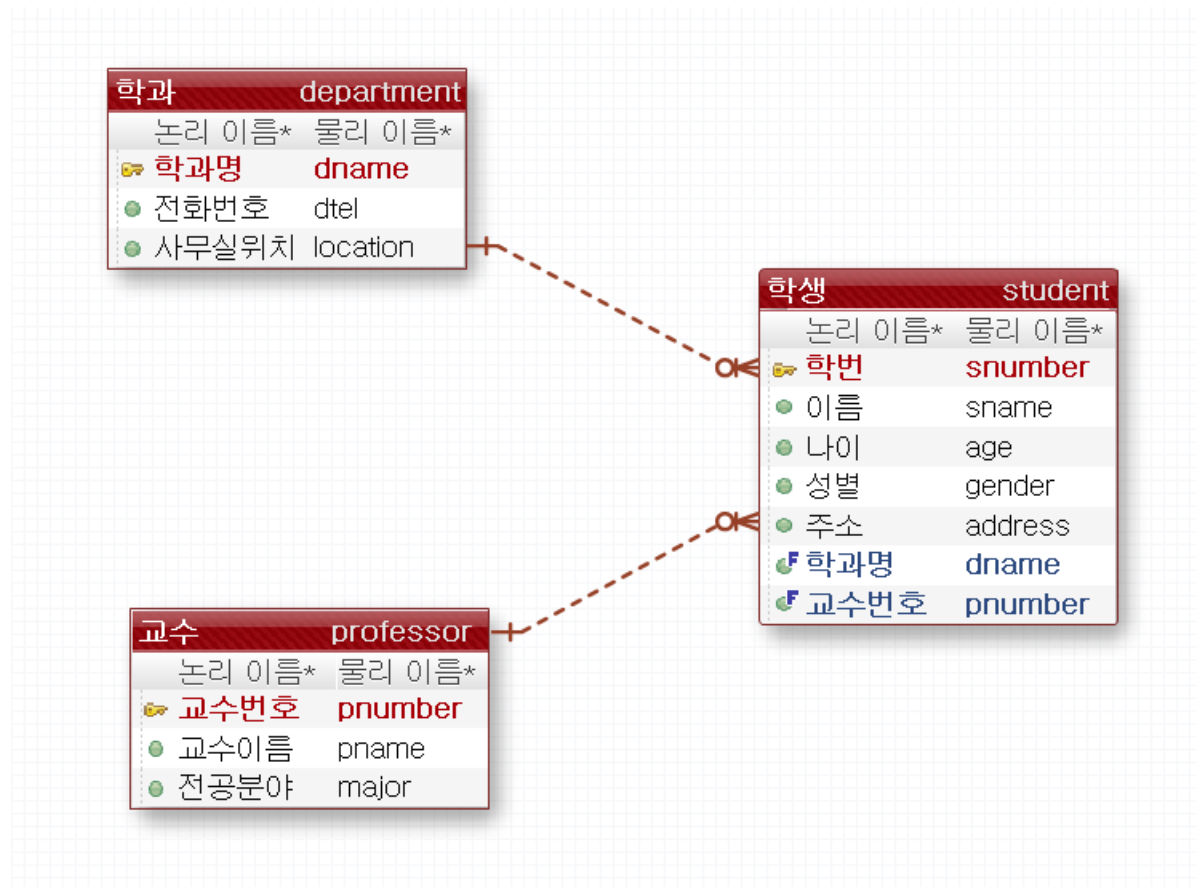
```
INSERT INTO student VALUES (20211236, '한비아', 23, '여', '서울시 강남구', '소프트웨어학과');
```

```
INSERT INTO student VALUES (20211237, '정산들', 27, '남', '경기도 수원시', '화학공학과');
```



# 개체 관계도(ERD)

## 교수 테이블이 있는 경우



# 데이터베이스 구축하기

## 교수 테이블이 있는 경우 - - 학생테이블 변경

```
-- 교수 테이블
CREATE TABLE professor(
    pnumber NUMBER(3),
    pname VARCHAR2(20),
    major VARCHAR2(30),
    PRIMARY KEY(pnumber)
);

-- 학생 테이블
CREATE TABLE student(
    snumber NUMBER(10),
    sname VARCHAR2(20),
    age NUMBER(3),
    gender VARCHAR2(4),
    address VARCHAR2(30),
    dname VARCHAR2(30),
    pnumber NUMBER(3),
    PRIMARY KEY(snumber),
    CONSTRAINT FK_dept_std FOREIGN KEY(dname)
REFERENCES department(dname),
    CONSTRAINT FK_prof_std FOREIGN KEY(pnumber)
REFERENCES professor(pnumber)
);
```



# 데이터베이스 구축하기

## 교수 테이블이 있는 경우 - 학생테이블 변경

```
-- 교수 추가
INSERT INTO professor VALUES (301, '박은종', 'JAVA 프로그래밍');
INSERT INTO professor VALUES (402, '송미영', 'JSP 웹프로그래밍');
INSERT INTO professor VALUES (501, '오용철', '데이터베이스');

-- 학생 추가 --
INSERT INTO student VALUES (20211234, '이강', 22, '여', '서울시 구로구', '소프트웨어학과', 301);
INSERT INTO student VALUES (20211235, '박대양', 25, '남', '서울시 성동구', '전기전자공학과', 501);
INSERT INTO student VALUES (20211236, '한비야', 23, '여', '서울시 강남구', '소프트웨어학과', 402);
INSERT INTO student VALUES (20211237, '정산들', 27, '남', '경기도 수원시', '화학공학과', 501);
```



# 외래키 실습

## 외래키(Foreign Key) 실습

```
-- 부모 테이블 생성 --
CREATE TABLE parent(
    p_id VARCHAR2(2) NOT NULL
);

ALTER TABLE PARENT ADD CONSTRAINT P_PK PRIMARY KEY(p_id);

-- 자식 테이블 생성 --
CREATE TABLE child(
    c_id VARCHAR2(2) NOT NULL,
    p_id VARCHAR2(2)
);

ALTER TABLE CHILD ADD CONSTRAINT C_PK PRIMARY KEY(p_id);
--ALTER TABLE CHILD DROP CONSTRAINT C_PK;

ALTER TABLE CHILD ADD CONSTRAINT C_FK FOREIGN KEY(p_id) REFERENCES PARENT (p_id);
--ALTER TABLE CHILD DROP CONSTRAINT C_FK;
```



# 외래키 실습

## 외래키(Foreign Key) 실습

```
-- 부모 테이블에 자료 입력
INSERT INTO PARENT VALUES ('A');
INSERT INTO PARENT VALUES ('B');

-- 자식 테이블에 자료 입력
INSERT INTO CHILD VALUES ('a', 'A');
INSERT INTO CHILD VALUES ('b', 'B');

-- 부모테이블에 DATA가 없으면 무결성 제약 위반 --
INSERT INTO CHILD VALUES ('c', 'C');
```

P_ID
1 A
2 B

C_ID	...
1 a	A
2 b	B

```
INSERT INTO CHILD VALUES ('c', 'C')
```

오류 보고 -

ORA-02291: 무결성 제약조건 (TAMI.C\_FK) 이 위배되었습니다- 부모 키가 없습니다



# 외래키 실습

## 외래키(Foreign Key) 실습

-- 부모레코드의 자료 삭제시 오류

```
DELETE FROM PARENT WHERE P_ID = 'A';
```

-- 외래키 삭제

```
ALTER TABLE CHILD DROP CONSTRAINT C_FK;
```

-- CASCADE : PARENT 삭제시 CHILD 함께 삭제 -

```
ALTER TABLE CHILD ADD CONSTRAINT C_FK FOREIGN KEY(p_id) REFERENCES PARENT (p_id)  
ON DELETE CASCADE;
```

명령의 29 행에서 시작하는 중 오류 발생 -

```
INSERT INTO CHILD VALUES ('c', 'C')
```

오류 보고 -

ORA-02291: 무결성 제약조건 (TAMI.C\_FK)이 위배되었습니다- 부모 키가 없습니다

P_ID
1 B

# 서브 쿼리

## 서브 쿼리(Sub-Query)란

최종 결과를 출력하는 쿼리를 메인 쿼리라고 한다면, 이를 위한 중간단계 혹은 보조 역할을 하는 SELECT문을 서브 쿼리라한다.

```
-- 전 사원의 평균 급여 이상을 받는 사원 수 조회
SELECT * FROM employees;

SELECT ROUND(AVG(salary), 2) 평균급여 FROM employees;

SELECT count(*) 평균급여이상_사원수
FROM employees
WHERE salary >= (SELECT AVG(salary) FROM employees);
```

	평균급여
1	6461.83

	평균급여이상_사원수
	51

# 서브 쿼리

## 서브 쿼리(Sub-Query)

```
-- job_history 테이블에 있는 employee_id와 job_id가 같은 사원 조회
SELECT * FROM job_history;

SELECT employee_id, first_name, job_id
FROM employees
WHERE (employee_id, job_id) IN (SELECT employee_id, job_id
                                FROM job_history);
```

EMPLOYEE_ID	FIRST_NAME	JOB_ID
200	Jennifer	AD ASST
176	Jonathon	SA REP



# 서브 쿼리

## 조인 조건이 걸린 서브 쿼리(Sub-Query)

```
-- 부서 location_id가 1800인 department_id를 가지는 사원 검색
SELECT * FROM departments;

-- 조인 (기본키, 외래키 관계)
SELECT * FROM employees a, departments b
  WHERE a.department_id = b.department_id
        AND b.location_id = 1800;

-- 서브 쿼리
SELECT * FROM employees a
  WHERE a.department_id = (SELECT b.department_id
                          FROM departments b
                          WHERE b.location_id = 1800);
```



DEPARTMENT_ID
1
20

LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	DEPARTMENT_ID.1
Hartstein	MHARTSTE	515.123.5555	04/02/17	MK MAN	13000	(null)	100	20	20
Fay	PFAY	603.123.6666	05/08/17	MK REP	6000	(null)	201	20	20



# 서브 쿼리

## 서브 쿼리

-- 서브쿼리 : 다중 자료(행)인 경우 --

```
SELECT * FROM employees A
WHERE A.department_id IN (SELECT B.department_id
                          FROM departments B
                          WHERE B.location_id = 1700);
```

	DEPARTMENT_ID
1	10
2	30
3	90
4	100
5	110
6	120
7	130
8	140
9	150
10	160
11	170
12	180
13	190
14	200

LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
Whalen	JWHALEN	515.123.4444	03/09/17	AD ASST	4400	(null)	101	10
Raphaely	DRAPHEAL	515.127.4561	02/12/07	PU MAN	11000	(null)	100	30
Khoo	AKHOO	515.127.4562	03/05/18	PU CLERK	3100	(null)	114	30
Baida	SBIDA	515.127.4563	05/12/24	PU CLERK	2900	(null)	114	30
Tobias	STOBIAS	515.127.4564	05/07/24	PU CLERK	2800	(null)	114	30
Himuro	GHIMURO	515.127.4565	06/11/15	PU CLERK	2600	(null)	114	30
Colmenares	KCOLMENA	515.127.4566	07/08/10	PU CLERK	2500	(null)	114	30
King	SKING	515.123.4567	03/06/17	AD PRES	24000	(null)	(null)	90
Kochhar	NKOCHHAR	515.123.4568	05/09/21	AD VP	17000	(null)	100	90
De Haan	LDEHAAN	515.123.4569	01/01/13	AD VP	17000	(null)	100	90



# 실무 : 게시판 페이징 처리

## ➤ 페이징 처리 : DB – 쿼리 작성

```
-- getBoardList()
-- ROWNUM을 왜 쓰는가? 순번 (bnum) 은 게시글이 삭제되므로 레코드수 계산이 부정확함
SELECT * FROM t_board ORDER BY regDate DESC;

SELECT ROWNUM, t_board.* FROM t_board;

SELECT ROWNUM, t_board.* FROM t_board ORDER BY regDate DESC;

SELECT ROWNUM, board.*      -- t_board로 중복 검색이 되어 오류 - 새로운 별칭을 붙여줌
FROM (SELECT * FROM t_board ORDER BY regDate DESC) board
--WHERE ROWNUM BETWEEN 1 AND 5;
WHERE ROWNUM BETWEEN 6 AND 10;  -- ROWNUM은 1번부터 시작하지 않으면 검색 안됨.

-- paging 처리 적용 쿼리
SELECT * FROM (
    SELECT ROWNUM num, board.*      -- ROWNUM이 SELECT 바깥에서 또 검색되므로 오류 - 별칭을 붙여줌
    FROM (SELECT * FROM t_board ORDER BY regDate DESC) board
)
WHERE num BETWEEN 6 AND 10;
```

