A. SQL Practice

A.1 데이터 검색 : SELECT

[샘플 문제] 사원정보(EMPLOYEE) 테이블에서 사원번호, 이름, 급여, 업무, 입사일, 상사의 사원번호를 출력하시오. 이때 이름은 성과 이름을 연결하여 Name이라는 별칭으로 출력하시오(107행).

```
SELECT employee_id, concat(first_name, ' ', last_name) as "Name",
        salary, job_id, hire_date, manager_id
FROM Employees;
```

[문제 1] 사원정보(EMPLOYEES) 테이블에서 사원의 성과 이름은 Name, 업무는 Job, 급여는 Salary, 연봉에 $100 보너스를 추가하여 계산한 값은 Increased Ann_Salary, 급여에 $100 보너스를 추가하여 계산한 연봉은 “Increased Salary”라는 별칭으로 출력하시오(107행).

```
SELECT concat(first_name, ' ', last_name) as "Name", job_id as "Job", salary as "Salary",
           12*salary+100 as "Increased Ann_salary", 12*(salary+100) as "Increased
Salary"
FROM Employees;
```

[문제 2] 사원정보(EMPLOYEE) 테이블에서 모든 사원의 이름(last_name)과 연봉을 “이름: 1 Year Salary = $연봉” 형식으로 출력하고, “1 Year Salary”라는 별칭을 붙여 출력하시오(107행).

```
SELECT concat(last_name, ': 1 Year Salary = $', salary*12) as "1 Year Salary"
FROM Employees;
```

[문제 3] 부서별로 담당하는 업무를 한 번씩만 출력하시오(20행).

```
SELECT DISTINCT department_id, job_id
FROM Employees;
```

A.2 데이터 제한 및 정렬 : WHERE, ORDER BY

[샘플 문제] HR 부서에서 예산 편성 문제로 급여 정보 보고서를 작성하려고 한다. 사원정보(EMPLOYEES) 테이블에서 급여가 $7,000~$10,000 범위 이외인 사람의 성과 이름(Name으로 별칭) 및 급여를 급여가 작은 순서로 출력하시오(75행).

```
SELECT concat(first_name, ' ', last_name), salary
FROM   Employees
WHERE salary NOT BETWEEN 7000 AND 10000
ORDER BY salary ;
```

[문제 1] 사원의 이름(last_name) 중에 'e' 및 'o' 글자가 포함된 사원을 출력하시오. 이때 머리글은 'e and o Name'라고 출력하시오(10행).

```
SELECT last_name as "e AND o Name"
FROM   Employees
WHERE last_name LIKE '%e%' AND last_name LIKE '%o%';
```

[문제 2] 현재 날짜 타입을 날짜 함수를 통해 확인하고, 1995년 05월 20일부터 1996년 05월 20일 사이에 고용된 사원들의 성과 이름(Name으로 별칭), 사원번호, 고용일자를 출력하시오. 단, 입사일이 빠른 순으로 정렬하시오(8행).

```
SELECT SYSDATE()  ;

select last_name name, employee_id, hire_date
from employees
where hire_date between STR_TO_DATE('1995/05/20', '%Y/%m/%d')
                and STR_TO_DATE('1996/05/20', '%Y/%m/%d')
order by hire_date;
```

[문제 3] HR 부서에서는 급여(salary)와 수당율(commission_pct)에 대한 지출 보고서를 작성하려고 한다. 이에 수당을 받는 모든 사원의 성과 이름(Name으로 별칭), 급여, 업무, 수당율을 출력하시오. 이때 급여가 큰 순서대로 정렬하되, 급여가 같으면 수당율이 큰 순서대로 정렬하시오(35행).

```
SELECT   CONCAT(first_name,' ',last_name) as "Name", salary, job_id, commission_pct
FROM       Employees
WHERE      commission_pct IS NOT NULL
ORDER BY   salary DESC, commission_pct DESC;
```

A.3 단일 행 함수 및 변환 함수

[샘플 문제] 이번 분기에 60번 IT 부서에서는 신규 프로그램을 개발하고 보급하여 회사에 공헌하였다. 이에 해당 부서의 사원 급여를 12.3% 인상하기로 하였다. 60번 IT 부서 사원의 급여를 12.3% 인상하여 정수만(반올림) 표시하는 보고서를 작성하시오. 출력 형식은 사번, 이름과 성(Name으로 별칭), 급여, 인상된 급여(Increased Salary로 별칭)순으로 출력한다(5행).

```
SELECT   employee_id, concat(last_name, ' ', first_name) as "Name", salary
         , ROUND(salary * 12.3, 0) "Increased Salary"
FROM     Employees
WHERE    department_id = 60;
```

[문제 1] 각 이름이 ‘s’로 끝나는 사원들의 이름과 업무를 아래의 예와 같이 출력하고자 한다. 출력 시 성과 이름은 첫 글자가 대문자, 업무는 모두 대문자로 출력하고 머리글은 Employee JOBs.로 표시하시오(18행).

예) Sigal Tobias is a PU_CLERK

```
SELECT   concat(first_name, ' ', last_name, ' is a ', UPPER(job_id)) as "Employee JOBs."
FROM     Employees
WHERE    SUBSTR(last_name, -1, 1) = 's';
```

[문제 2] 모든 사원의 연봉을 표시하는 보고서를 작성하려고 한다. 보고서에 사원의 성과 이름(Name으로 별칭), 급여, 수당여부에 따른 연봉을 포함하여 출력하시오. 수당여부는 수당이 있으면 “Salary + Commission”, 수당이 없으면 “Salary only”라고 표시하고, 별칭은 적절히 붙인다. 또한 출력 시 연봉이 높은 순으로 정렬한다(107행).

<Hint> IFNULL() 함수, IF() 함수를 이용하여 NULL이었는지 아니었는지를 식별할 수 있다.

```
SELECT   concat(first_name, ' ', last_name) as "Name", salary
       , (salary * 12) + (salary * 12 * IFNULL(commission_pct, 0)) "Annual Salary"
       , IF(commission_pct IS NULL, 'Salary + Commission', 'Salary only') "Commission ?"
FROM     Employees
ORDER BY 2 DESC;
```

[문제 3] 모든 사원들 성과 이름(Name으로 별칭), 입사일 그리고 입사일이 어떤 요일이였는지 출력하시오. 이때 주(week)의 시작인 일요일부터 출력되도록 정렬하시오(107행).

```
SELECT       concat(first_name, ' ', last_name) as "Name",
             hire_date, DATE_FORMAT(hire_date, '%W') as "Day of the week"
FROM         Employees
ORDER BY   DATE_FORMAT(hire_date, '%w');
```

A.4 집계된 데이터 보고 : 집계 함수

[샘플 문제] 모든 사원은 직속 상사 및 직속 직원을 갖는다. 단, 최상위 또는 최하위 직원은 직속 상사 및 직원이 없다. 소속된 사원들 중 어떤 사원의 상사로 근무 중인 사원의 총 수를 출력하시오(1행).

```
SELECT      COUNT(DISTINCT manager_id) "Count Managers"
FROM        Employees;
```

[문제 1] 각 사원이 소속된 부서별로 급여 합계, 급여 평균, 급여 최대값, 급여 최소값을 집계하고자 한다. 계산된 출력값은 6자리와 세 자리 구분기호, $ 표시와 함께 출력하고 부서번호의 오름차순 정렬하시오. 단, 부서에 소속되지 않은 사원에 대한 정보는 제외하고 출력시 머리글은 아래 예시처럼 별칭(alias) 처리하시오(11행).

```
SELECT    department_id,
          CONCAT('$', FORMAT(SUM(salary), 0)) as "Sum Salary",
          CONCAT('$', FORMAT(AVG(salary), 1)) as "Avg Salary",
          CONCAT('$', FORMAT(MAX(salary), 0)) as "Max Salary",
          CONCAT('$', FORMAT(MIN(salary), 0)) as "Min Salary"
FROM      Employees
WHERE     department_id is NOT NULL
GROUP BY  department_id;
```

[문제 2] 사원들의 업무별 전체 급여 평균이 $10,000보다 큰 경우를 조회하여 업무, 급여 평균을 출력하시오. 단 업무에 사원(CLERK)이 포함된 경우는 제외하고 전체 급여 평균이 높은 순서대로 출력하시오(7행).

```
SELECT    job_id, avg(salary) as "Avg Salary"
FROM      Employees
WHERE     job_id NOT LIKE '%CLERK%'
GROUP BY  job_id
HAVING    avg(salary) > 10000
ORDER BY  avg(salary) DESC;
```

A.5 여러 테이블의 데이터 표시 : JOIN

[샘플 문제] hr 스키마에 존재하는 Employees, Departments, Locations 테이블의 구조를 파악한 후 Oxford에 근무하는 사원의 성과 이름(Name으로 별칭), 업무, 부서명, 도시명을 출력하시오. 이때 첫 번째 열은 회사명인 'Han-Bit'이라는 상수값이 출력되도록 하시오(34행).

```
DESC Employees ;
DESC Departments ;
DESC Locations ;

(ANSI 표준 SQL)
SELECT  'Han-Bit', CONCAT(e.first_name, ' ', e.last_name) as "Name",
        e.job_id, d.department_name, l.city
FROM  Employees e JOIN Departments d ON (e.department_id = d.department_id)
        JOIN     locations l ON     (d.location_id = l.location_id)
        AND    l.city LIKE 'Oxford';

(MySQL)
SELECT  'Han-Bit', CONCAT(e.first_name, ' ', e.last_name) as "Name",
        e.job_id, d.department_name, l.city
FROM     Employees e , Departments d ,  Locations l
WHERE    e.department_id = d.department_id
        AND d.location_id = l.location_id
        AND l.city LIKE 'Oxford';
```

[문제 1] HR 스키마에 있는 Employees, Departments 테이블의 구조를 파악한 후 사원수가 5명 이상인 부서의 부서명과 사원수를 출력하시오. 이때 사원수가 많은 순으로 정렬하시오(5행).

```
DESC Employees;
DESC Departments;

SELECT d.department_name, COUNT(e.employee_id)
FROM Employees e, Departments d
WHERE  e.department_id=d.department_id
GROUP BY d.department_name
HAVING COUNT(e.employee_id) >= 5
ORDER BY count(*) DESC;
```

[문제 2] 각 사원의 급여에 따른 급여 등급을 보고하려고 한다. 급여 등급은 JOB_GRADES 테이블에 표시된다. 해당 테이블의 구조를 살펴본 후 사원의 성과 이름(Name으로 별칭), 업무, 부서명, 입사일, 급여, 급여등급을 출력하시오(106행).

```
SELECT   CONCAT(e.first_name, ' ', e.last_name) as "Name",
         e.job_id, d.department_name, e.hire_date, e.salary, j.grade_level
FROM     Employees e , Departments d , job_grades j
WHERE    e.department_id = d.department_id
AND      e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

[문제 3] 각 사원과 직속 상사와의 관계를 이용하여 다음과 같은 형식의 보고서를 작성하고자 한다.

예) 홍길동은 허균에게 보고한다 → Eleni Zlotkey report to Steven King

어떤 사원이 어떤 사원에서 보고하는지를 위 예를 참고하여 출력하시오. 단, 보고할 상사가 없는 사원이 있다면 그 정보도 포함하여 출력하고, 상사의 이름은 대문자로 출력하시오(107행).

```
SELECT CONCAT(e.first_name, ' ', e.last_name, ' report to ',
         UPPER(CONCAT(IFNULL(m.first_name,' '), ' ', IFNULL(m.last_name,' '))) AS
"Employee vs Manager"
FROM Employees e LEFT OUTER JOIN Employees m
         ON (e.manager_id = m.employee_id)
    ;
```

A.6 부속질의를 사용하여 복잡한 질의 해결

[샘플 문제] HR 부서의 어떤 사원은 급여정보를 조회하는 업무를 맡고 있다. Tucker 사원(last_name)보다 급여를 많이 받고 있는 사원의 성과 이름(Name으로 별칭), 업무, 급여를 출력하시오(15행).

```
SELECT CONCAT(first_name, ' ', last_name) as "Name", job_id, salary
FROM Employees
WHERE salary > (SELECT salary
                FROM Employees
                WHERE last_name = 'Tucker');
```

[문제 1] 사원의 급여 정보 중 업무별 최소 급여를 받고 있는 사원의 성과 이름(Name으로 별칭), 업무, 급여, 입사일을 출력하시오(21행).

```
SELECT CONCAT(first_name, ' ', last_name) as "Name",
        job_id, salary, hire_date
FROM Employees e1
WHERE salary    IN
        (SELECT min(salary)
        FROM Employees e2
        WHERE e1.job_id=e2.job_id
        GROUP BY job_id);
```

[문제 2] 소속 부서의 평균 급여보다 많은 급여를 받는 사원에 대하여 사원의 성과 이름(Name으로 별칭), 급여, 부서번호, 업무를 출력하시오(38행).

```
SELECT CONCAT(first_name, ' ', last_name) as "Name",
                salary, department_id, job_id
FROM Employees e
WHERE salary  >       (SELECT AVG(salary)
                       FROM Employees
                       WHERE department_id = e.department_id);
```

[문제 3] 사원들의 지역별 근무 현황을 조회하고자 한다. 도시 이름이 영문 'O' 로 시작하는 지역에 살고 있는 사원의 사번, 이름, 업무, 입사일을 출력하시오(34행).

```
SELECT          employee_id,CONCAT(first_name, ' ', last_name) as "Name",
                job_id, hire_date
FROM   Employees
WHERE department_id in (
                SELECT          department_id
                FROM   Departments
                WHERE location_id in (
                                SELECT          location_id
                                FROM   Locations
                                WHERE city like 'O%'));
```

[문제 4] 모든 사원의 소속부서 평균연봉을 계산하여 사원별로 성과 이름(Name으로 별칭), 업무, 급여, 부서번호, 부서 평균연봉(Department Avg Salary로 별칭)을 출력하시오(107행).

```
SELECT          CONCAT(first_name, ' ', last_name) as "Name",
        job_id, salary, department_id,
           (SELECT ROUND(AVG(salary))
            FROM Employees
            WHERE department_id = e.department_id) as "Department Avg Salary"
FROM   Employees e;
```

A.7 집합 연산자 사용

[샘플 문제] hr 스키마에 있는 JOB_HISTORY 테이블은 사원들의 업무 변경 이력을 나타내는 테이블이다. 이 정보를 이용하여 모든 사원의 현재 및 이전의 업무 이력 정보를 출력하고자 한다. 중복된 사원정보가 있을 경우 한 번만 표시하여 출력하시오(115행).

```
SELECT employee_id, job_id
FROM Employees
UNION
SELECT employee_id, job_id
FROM    job_history;
```

[문제 1] 위 샘플 문제에서 각 사원의 업무 이력 정보를 확인하였다. 하지만 모든 사원의 업무 이력 전체를 보지는 못했다. 여기에서는 모든 사원의 업무 이력 변경 정보 및 업무 변경에 따른 부서정보를 사번이 빠른 순서대로 출력하시오(117행).

```
SELECT employee_id, job_id, department_id
FROM Employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM    job_history
ORDER BY employee_id;
```

[문제 2] 사원정보(Employees) 테이블에 JOB_ID는 사원의 현재 업무를 뜻하고, JOB_HISTORY에 JOB_ID는 사원의 이전 업무를 뜻한다. 이 두 테이블을 교차해보면 업무가 변경된 사원의 정보도 볼 수 있지만 이전에 한번 했던 같은 업무를 그대로 하고 있는 사원의 정보도 볼 수 있다. 이전에 한번 했던 같은 업무를 보고 있는 사원의 사번과 업무를 출력하시오(2행).

```
SELECT e.employee_id, e.job_id
FROM    Employees e, job_history j
WHERE e.employee_id=j.employee_id AND e.job_id=j.job_id;
```

[문제 2-계속] 위 결과를 이용하여 출력된 176번 사원의 업무 이력의 변경 날짜 이력을 조회하시오(3행).

```
SELECT employee_id, job_id, NULL AS "Start Date", NULL AS "End Date"
FROM    Employees
WHERE employee_id = 176
UNION
SELECT employee_id, job_id, start_date, end_date
FROM    job_history
WHERE employee_id = 176;
```

[문제 3] 우리 회사는 1년에 한 번 업무를 변경하여 전체적인 회사 업무를 사원들이 익히도록 하는 Role change 정책을 시행하고 있다. 이번 인사이동 때 아직 업무가 변경된 적이 없는 사원들을 적합한 업무로 이동시키려고 한다. HR 부서의 사원정보 테이블과 업무이력정보 테이블을 이용하여 한 번도 업무가 변경되지 않은 사원의 사번을 출력하시오(100행).

```
SELECT employee_id
FROM    Employees
WHERE employee_id NOT IN
        (SELECT employee_id FROM    job_history);
```

A.8 조건부 논리 표현식 제어 : CASE

[샘플 문제] HR 부서에서는 신규 프로젝트의 성공으로 해당하는 각 업무 자들에 대한 급여 인상안을 결정하고, 다음과 같이 업무 별 급여 인상에 대해 적용하고자 한다. 현재 107명의 사원은 19개의 업무에 소속되어 근무 중이다. (Distinct job_id) 이 중 5개의 업무 자들에 대한 급여 인상이 각각 결정되었고, 나머지 업무에 대해서는 인상이 동결되었다 (107행). HR_REP(10%), MK_REP(12%), PR_REP(15%), SA_REP(18%), IT_PROG(20%)

<Hint> CASE와 DECODE를 이용하여 위 조건을 만족하는 구문을 작성해 본다.

```
SELECT employee_id, CONCAT(last_name, ' ', first_name) as "Name", job_id, salary
          , CASE job_id WHEN 'HR_REP' THEN 1.10 * salary
                        WHEN 'MK_REP' THEN 1.12 * salary
                        WHEN 'PR_REP' THEN 1.15 * salary
                        WHEN 'SA_REP' THEN 1.18 * salary
                        WHEN 'IT_PROG' THEN 1.20 * salary
            ELSE        salary
            END    "New Salary"
FROM   Employees;
```

[샘플 문제] HR 부서에서는 최상위 입사일에 해당하는 2001년부터 2003년까지 입사자들의 급여를 각각 5%, 3%, 1% 인상하여 지분에 따른 배당금으로 지급하고자 한다. 전체 사원들 중 해당 년도에 해당하는 사원들의 급여를 검색하여 적용하고, 입사일자에 따른 오름차순 정렬을 수행하시오.

<Hint> CASE 구문을 이용한 검색 조건 비교를 이용하여 해당 년도 별 조건 처리를 수행 할 수 있다.

```
SELECT employee_id, CONCAT(last_name, ' ', first_name) as "Name",
                hire_date, salary,
        CASE WHEN hire_date < STR_TO_DATE('1998-01-01', '%Y-%m-%d')
                THEN salary * 1.05
                WHEN hire_date < STR_TO_DATE('1999-01-01', '%Y-%m-%d')
                THEN salary * 1.03
                WHEN hire_date < STR_TO_DATE('2000-01-01', '%Y-%m-%d')
                THEN salary * 1.01
                ELSE    salary
        END    "New Salary"
FROM   Employees
ORDER BY hire_date;
```

[문제 1] 부서별 급여 합계를 구하고, 그 결과를 가지고 다음과 같이 표현하시오(12행).
Sum Salary > 100000 이면, “Excellent"
Sum Salary > 50000 이면, “Good"
Sum Salary > 10000 이면, “Medium"
Sum Salary <= 10000 이면, “Well"

<Hint> case 문을 사용하는 보통 방법을 사용할 수도 있고, inline view를 이용하여 우선 부서별 급여 합계를 구하고, 상위 쿼리에서 CASE 구문을 이용하여 위의 조건 비교를 통해 급여 합계에 따른 표현을 할 수 있다.

```
SELECT department_id, SUM(salary) sum_sal,
        CASE WHEN SUM(salary) > 100000 THEN 'Excellent'
                WHEN SUM(salary) > 50000 THEN 'Good'
                WHEN SUM(salary) > 10000 THEN 'Medium'
                WHEN SUM(salary) <= 10000 THEN 'Well'
        END "Department Grade Salary"
FROM   Employees
GROUP BY department_id;
```

[문제 2] 2005년 이전에 입사한 사원 중 업무에 “MGR"이 포함된 사원은 15% 급여를 인상하고, ”MAN"이 포함된 사원은 20% 급여 인상이 정해졌고, 또한 2005년부터 근무를 시작한 사원 중 “MGR"이 포함된 사원은 25% 급여 인상을 수행하는 쿼리를 작성하시오. 해당되는 사원들만 출력한다. (13행).

```
SELECT employee_id, CONCAT(last_name, ' ', first_name) as "Name", job_id, hire_date, salary,
        CASE
                WHEN hire_date < STR_TO_DATE('20000101', '%Y%m%d')
                        and job_id LIKE '%MGR%' THEN salary*1.15
                WHEN hire_date < STR_TO_DATE('20000101', '%Y%m%d')
                        and job_id LIKE '%MAN%' THEN salary*1.20
                WHEN hire_date >= STR_TO_DATE('20000101', '%Y%m%d')
                        and job_id LIKE '%MGR%' THEN salary*1.25
        END "Department Grade Salary"
FROM   Employees
WHERE ((hire_date < STR_TO_DATE('20000101', '%Y%m%d') AND
                (job_id like '%MGR' OR job_id like '%MAN')))
        OR
        (((hire_date >= STR_TO_DATE('20000101', '%Y%m%d')) AND
                (job_id like '%MGR')));
```

(2중 case 문)

```
SELECT employee_id, CONCAT(last_name, ' ', first_name) as "Name", job_id, hire_date,
salary,
        CASE WHEN hire_date < STR_TO_DATE('20000101', '%Y%m%d')
                THEN CASE WHEN job_id like '%MGR' THEN salary * 1.15
                          WHEN job_id like '%MAN' THEN salary * 1.20
                             ELSE salary END
                ELSE CASE  WHEN job_id like '%MGR' THEN salary * 1.25
                             ELSE salary END
            END "Job Salary"
FROM employees
WHERE ((hire_date < STR_TO_DATE('20000101', '%Y%m%d') AND
                (job_id like '%MGR' OR job_id like '%MAN')))
        OR
        (((hire_date >= STR_TO_DATE('20000101', '%Y%m%d')) AND
                (job_id like '%MGR')));
```

[문제 3] 월별로 입사한 사원 수 출력
(Step 1) 월별로 입사한 사원수가 아래와 같이 각 행별로 출력되도록 하시오(12행).
(Step 2) 첫 행에 모든 월별 입사 사원수가 출력되도록 하시오(1행).

(STEP1)

```
SELECT
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '01'
                THEN count(*) ELSE 0 END "1 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '02'
                THEN count(*) ELSE 0 END "2 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '03'
                THEN count(*) ELSE 0 END "3 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '04'
                THEN count(*) ELSE 0 END "4 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '05'
                THEN count(*) ELSE 0 END "5 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '06'
                THEN count(*) ELSE 0 END "6 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '07'
                THEN count(*) ELSE 0 END "7 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '08'
                THEN count(*) ELSE 0 END "8 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '09'
                THEN count(*) ELSE 0 END "9 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '10'
                THEN count(*) ELSE 0 END "10 Month",
```

```
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '11'
                THEN count(*) ELSE 0 END "11 Month",
        CASE WHEN DATE_FORMAT(hire_date, '%m') = '12'
                THEN count(*) ELSE 0 END "12 Month"
FROM Employees
GROUP BY DATE_FORMAT(hire_date, '%m')
ORDER BY DATE_FORMAT(hire_date, '%m');
```

(STEP2)

```
SELECT SUM(M1) "1 Month",SUM(M2) "2 Month",SUM(M3) "3 Month",
       SUM(M4) "4 Month",SUM(M5) "5 Month",SUM(M6) "6 Month",
       SUM(M7) "7 Month",SUM(M8) "8 Month",SUM(M9) "9 Month",
       SUM(M10) "10 Month",SUM(M11) "11 Month",SUM(M12) "12 Month" FROM (
        SELECT
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '01'
                        THEN count(*) ELSE 0 END M1,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '02'
                        THEN count(*) ELSE 0 END M2,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '03'
                        THEN count(*) ELSE 0 END M3,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '04'
                        THEN count(*) ELSE 0 END M4,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '05'
                        THEN count(*) ELSE 0 END M5,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '06'
                        THEN count(*) ELSE 0 END M6,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '07'
                        THEN count(*) ELSE 0 END M7,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '08'
                        THEN count(*) ELSE 0 END M8,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '09'
                        THEN count(*) ELSE 0 END M9,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '10'
                        THEN count(*) ELSE 0 END M10,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '11'
                        THEN count(*) ELSE 0 END M11,
                CASE WHEN DATE_FORMAT(hire_date, '%m') = '12'
                        THEN count(*) ELSE 0 END M12
        FROM Employees
        GROUP BY DATE_FORMAT(hire_date, '%m')
) E
;
```

A.9 다차원 그룹 데이터 검색 : ROLLUP

[샘플 문제] HR 부서에서는 부서와 업무별 급여 합계를 구하여 신년도 급여 수준 레벨을 지정하고자 한다. 부서번호와 업무를 기준으로 전체 행을 그룹별로 나누어 급여 합계와 인원수를 출력하시오(20행).

```
SELECT       department_id, job_id
               , CONCAT('$', FORMAT(SUM(salary), 0)) as "Salary SUM"
               , COUNT(employee_id) as "COUNT EMPs"
FROM         Employees
GROUP BY   department_id, job_id
ORDER BY   department_id;
```

[샘플 문제-계속] 위 결과에서 부서번호와 업무를 기준으로 전체 행을 그룹별로 나누어 급여 합계와 인원수를 출력하고, 부서번호와 업무로 그룹화된 총합계, 총 인원수를 출력하시오(33행).

```
SELECT       department_id, job_id
               , CONCAT('$', FORMAT(SUM(salary), 0)) as "Salary SUM"
               , COUNT(employee_id) as "COUNT EMPs"
FROM         Employees
GROUP BY   department_id, job_id WITH ROLLUP
;
```

[문제 1] 위 샘플 문제의 결과를 근거로 부서에 대한 집계 결과가 아니면 (ALL-DEPTS) 라고 출력하고, 업무에 대한 집계 결과가 아니면 (ALL-JOBS)라고 출력하며, 해당 집계에 대한 사원수와 평균연봉을 구하시오(33행).

<Hint> ROLLUP, GROUPING 함수를 사용한다. GROUPING(e.department_id) 값은 그룹값일 경우는 1, 아닐 경우는 0 값을 갖는다.1

```
SELECT
    CASE WHEN GROUPING(e.department_id)=1 THEN '(All-DEPTS)'
                ELSE e.department_id END as "Dept#",
    CASE WHEN GROUPING(e.job_id)=1 THEN '(All-JOBS)'
                ELSE e.job_id END as "Jobs",
        COUNT(*)  as "Count Emps",
                ROUND(AVG(salary) * 12) as "Avg Ann_sal"
 FROM Employees e
 GROUP BY e.department_id, e.job_id with rollup ;
```

A.10 분석함수 : NTILE, RANK

[샘플 문제] 분석함수 NTILE()을 이용하여 부서별 급여합계를 구하시오. 단, 급여 합계가 제일 큰 것이 1, 제일 작은 것이 4가 되도록 등급을 나누어 출력하고, 등급을 기준으로 오름차순 정렬하시오(12행).

```
SELECT   department_id, SUM(salary) as "Sum Salary"
            , NTILE (4) over (order by SUM(salary) DESC) as "Bucket#"
FROM    Employees
GROUP BY department_id
ORDER BY 3;
```

[문제 1] 각 사원들이 소속된 부서별로 급여를 기준으로 내림차순 정렬하시오. 이때 다음 세 가지 함수를 이용하여 순위를 그림과 같이 출력하시오(107행).

- RANK() : 동등 순위가 발생하면 다음 순위는 중복된 값만큼 증가시키고 매긴다.
- DENSE_RANK() : 동등 순위가 발생하면 다음 순위는 중복된 값을 무시하고 바로 다음 번호로 매긴다.
- ROW_NUMBER() : 동등 순위 자체를 인식하지 않고 매번 번호가 증가하여 매긴다.

```
SELECT   employee_id, last_name, salary, department_id,
          RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) as "Rank",
          DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) as "Dense_Rank",
          ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) as "Row_Number"
FROM     Employees;
```

[문제 2] 지정된 개수의 이전, 이후 행의 값을 가져오는 LAG(), LEAD() 함수를 이용하여 50번 부서의 사원 정보를 급여순으로 내림차순 정렬하고 이전, 다음 행의 급여를 함께 출력하시오(45행).

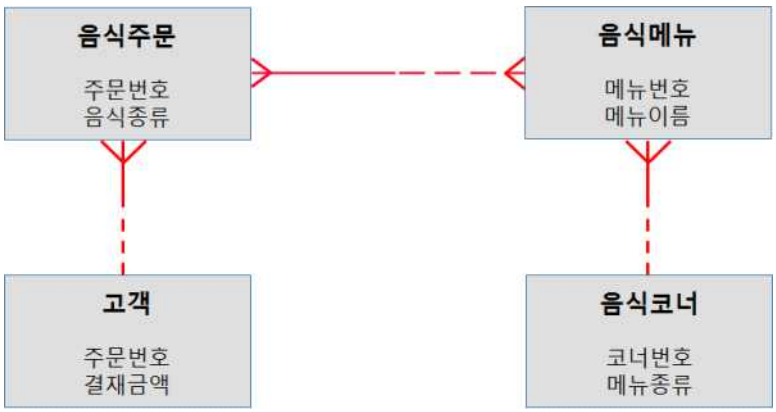
```
SELECT   employee_id, last_name, salary,
         LAG (salary,1,0) over (order by salary DESC) as "Prev_Sal",
         LEAD (salary,1,0) over (order by salary DESC) as "Next_Sal"
FROM     Employees
WHERE    department_id = 50;
```

B. 데이터 모델링

B.1 푸드코트 ERD

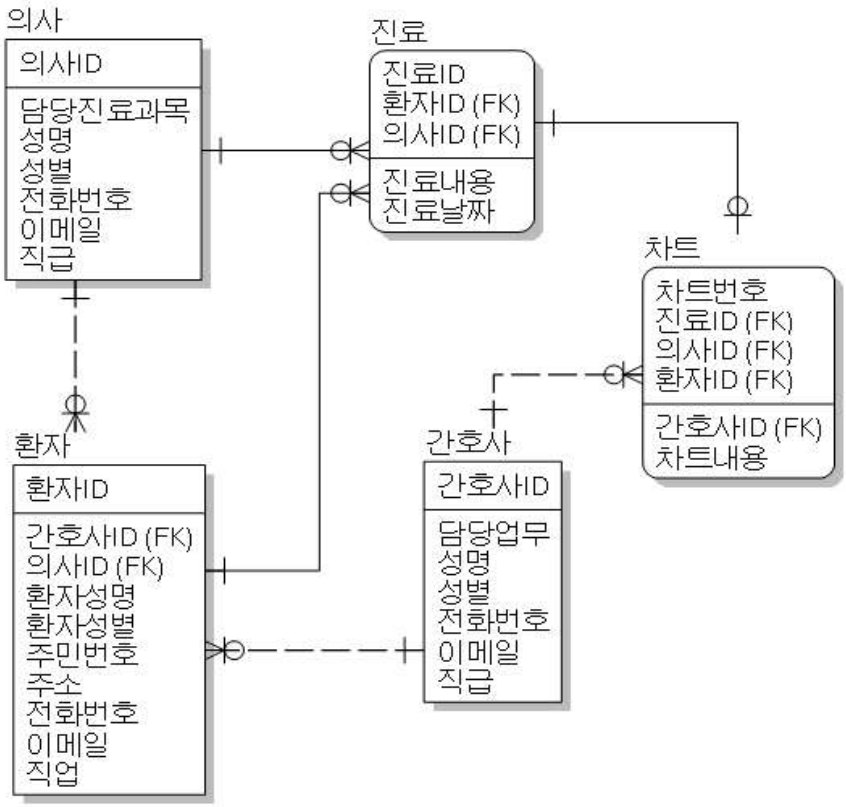
[문제1]

- 우리가족(고객) 각자 음식주문기를 통해 반드시 하나 이상의 음식메뉴를 선택할 수 있다.
- 우리가족(고객) 각자 선택한 음식주문 메뉴에는 하나 이상의 메뉴를 포함한 주문일 수 있다.
- 각각의 음식메뉴 주문은 반드시 한 명의 우리가족(고객)에 속해 있다.
- 우리가족(고객) 각자는 하나 이상의 음식메뉴를 주문할 수 있다.
- 우리가족(고객) 각자가 주문한 음식메뉴는 반드시 하나의 음식코너에서 조리하여 제공한다.
- 각각의 음식코너는 하나 이상의 음식메뉴를 보유할 수 있다.



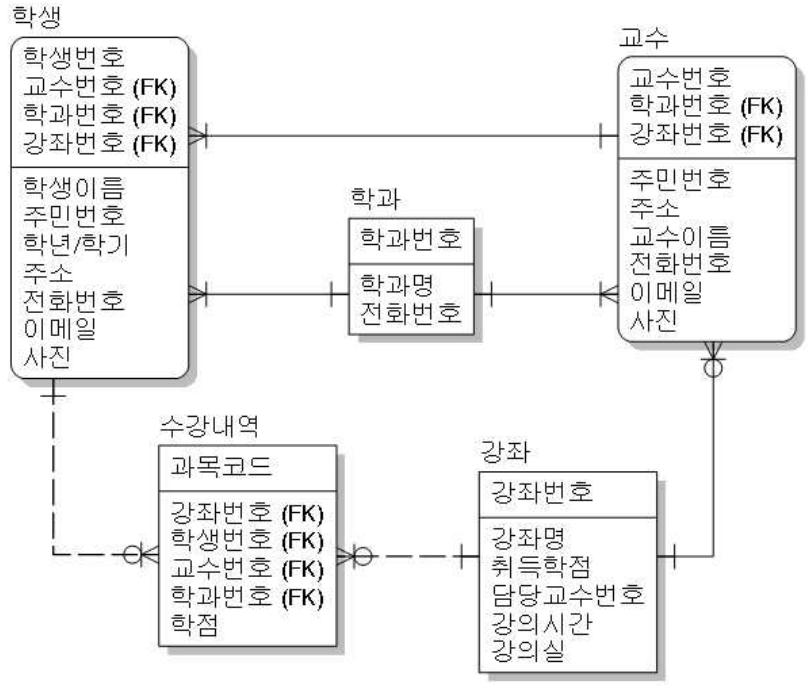
B.2 병원 업무 관리 ERD

[문제1]

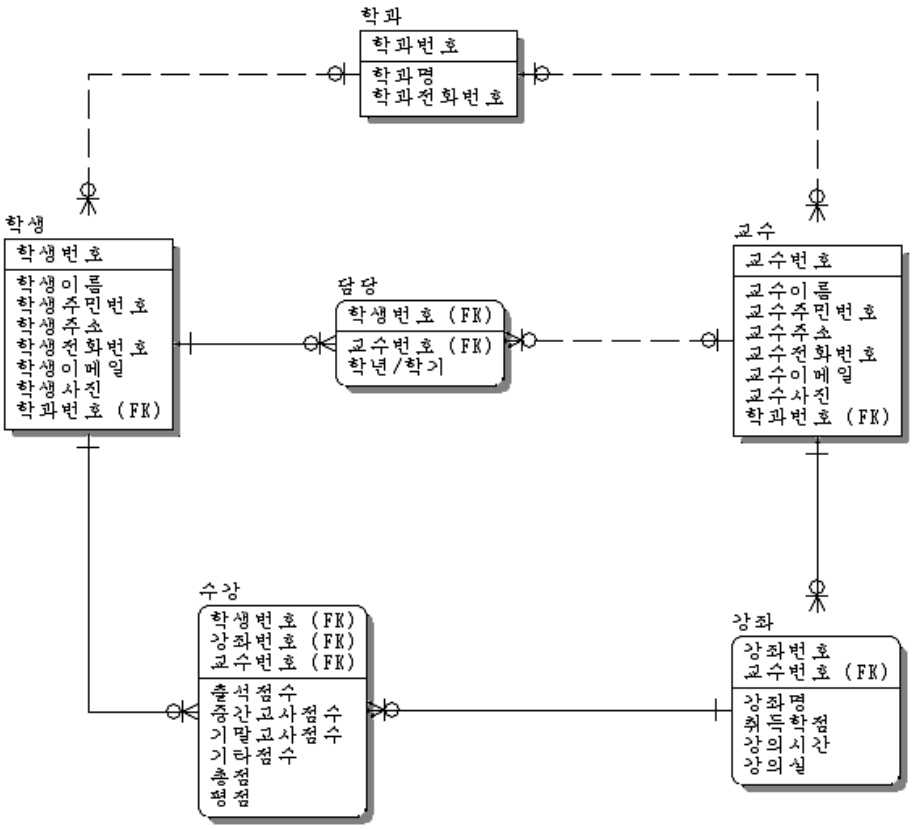


B.3 학사 관리 ERD

[문제1]

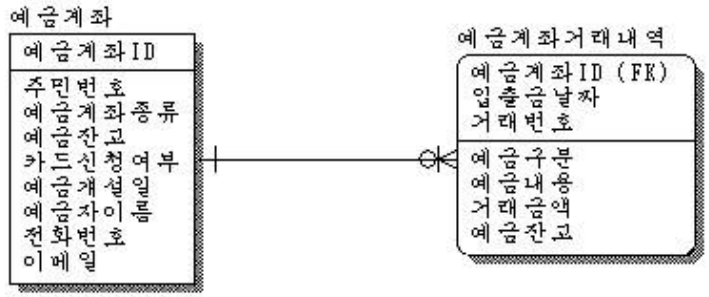


(또는)

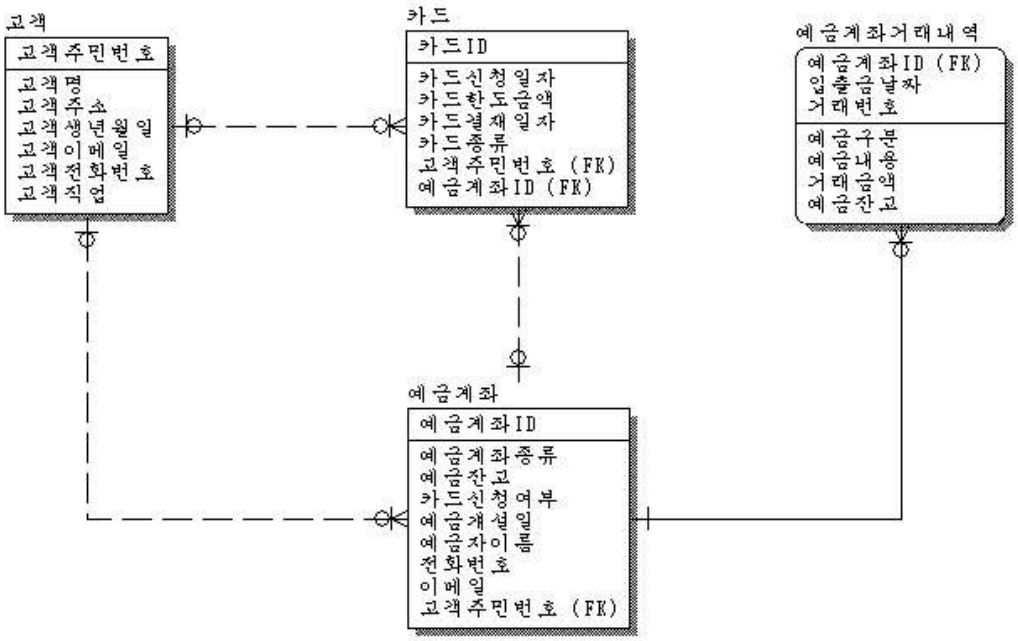


B.4 은행 업무 관리 ERD

[문제1]

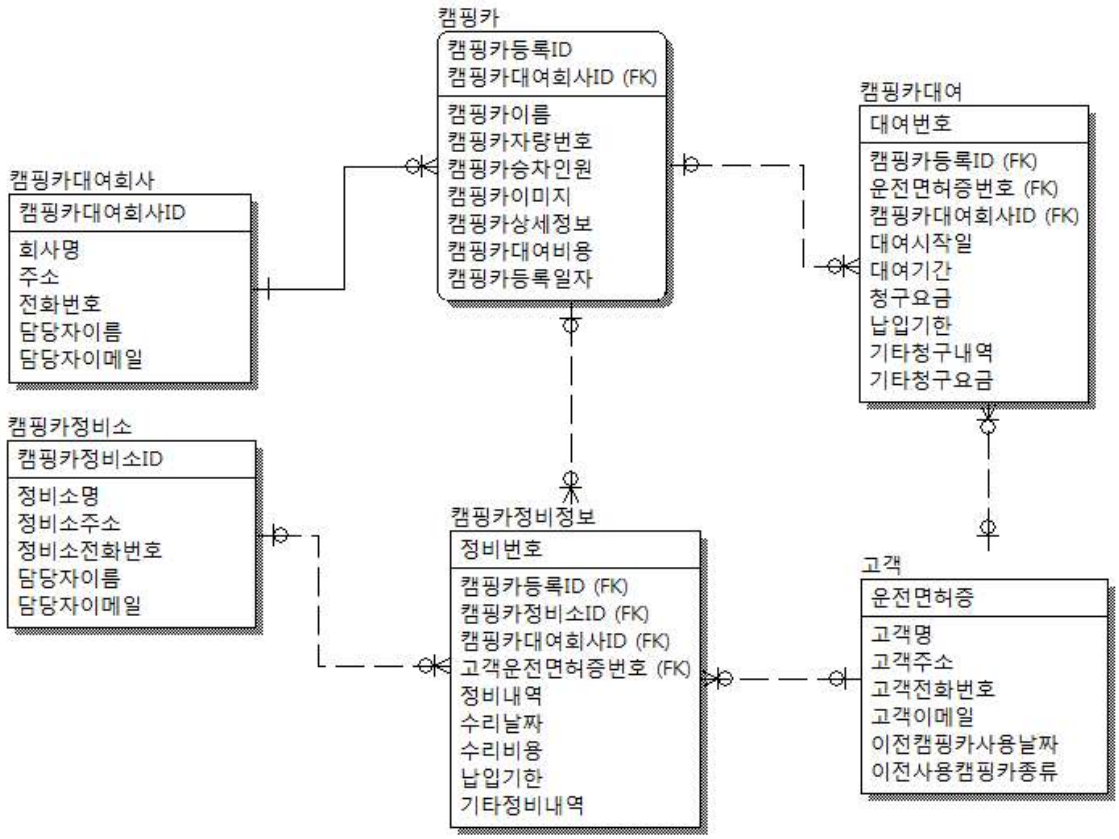


[문제2]



B.5 캠핑카 대여 관리 ERD

[문제1]



B.6 정규화 수행 연습 : 도서주문

[문제1]

(1) 제 1 정규화

주문번호	주문날짜	회원 이름	회원 번호	회원 주소	전화번호	도서 SNO	도서명	수 량	단가
20140213123	20140213	Lee	L-123	서울 강남구	010-3664-1234	9788 9123	NoSQL의 진수를 만나다	1	25000
20140509238	20140509	Park	P-234	서울 성동구	010-5301-3456	9788 9234	데이터베이스 개론과 실습	1	27000
20140509238	20140509	Park	P-234	서울 성동구	010-5301-3456	9788 9235	Java for Beginner	2	18000
20140721376	20140721	Kim	K-345	경기도 용인	010-7341-2345	9788 9345	정보보안 개론과 실습	5	24000
20140729401	20140729	Nam	N-456	강원도 속초	010-9279-3456	9788 9234	데이터베이스 개론과 실습	1	27000
20141005456	20141005	Park	P-234	서울 성동구	010-5301-3456	9788 9567	오라클 프로그래밍	4	27000

(2) 제 2 정규화

주문회원

주문번호	주문날짜	회원 이름	회원 번호	회원 주소	전화번호
20140213123	20140213	Lee	L-123	서울 강남구	010-3664-1234
20140509238	20140509	Park	P-234	서울 성동구	010-5301-3456
20140509238	20140509	Park	P-234	서울 성동구	010-5301-3456
20140721376	20140721	Kim	K-345	경기도 용인	010-7341-2345
20140729401	20140729	Nam	N-456	강원도 속초	010-9279-3456
20141005456	20141005	Park	P-234	서울 성동구	010-5301-3456

주문내역

주문번호	도서 SNO	수 량	단가
20140213123	97889123	1	25000
20140509238	97889234	1	27000
20140509238	97889235	2	18000
20140721376	97889345	5	24000
20140729401	97889234	1	27000
20141005456	97889567	4	27000

주문도서

도서 SNO	도서명
97889123	NoSQL의 진수를 만나다
97889234	데이터베이스 개론과 실습
97889235	Java for Beginner
97889345	정보보안 개론과 실습
97889567	오라클 프로그래밍

(3) 제 3 정규화

주문회원

주문번호	주문날짜	회원 번호
20140213123	20140213	L-123
20140509238	20140509	P-234
20140509238	20140509	P-234
20140721376	20140721	N-345
20140729401	20140729	K-456
20141005456	20141005	H-567

회원정보

회원 번호	회원 이름	회원 주소	전화번호
L-123	Lee	서울 강남구	010-3664-1234
P-234	Park	서울 성동구	010-5301-3456
N-345	Nam	경기도 용인	010-7341-2345
K-456	Kim	강원도 속초	010-9279-3456
H-567	Hong	서울 서초구	010-8653-4567

주문내역

주문번호	도서 SNO	수량	단가
20140213123	97889123	1	25000
20140509238	97889234	1	27000
20140509238	97889235	2	18000
20140721376	97889345	5	24000
20140729401	97889234	1	28000
20141005456	97889567	4	27000

주문도서

도서 SNO	도서명
97889123	NoSQL의 진수를 만나다
97889234	데이터베이스 개론과 실습
97889235	Java for Beginner
97889345	정보보안 개론과 실습
97889567	오라클 프로그래밍

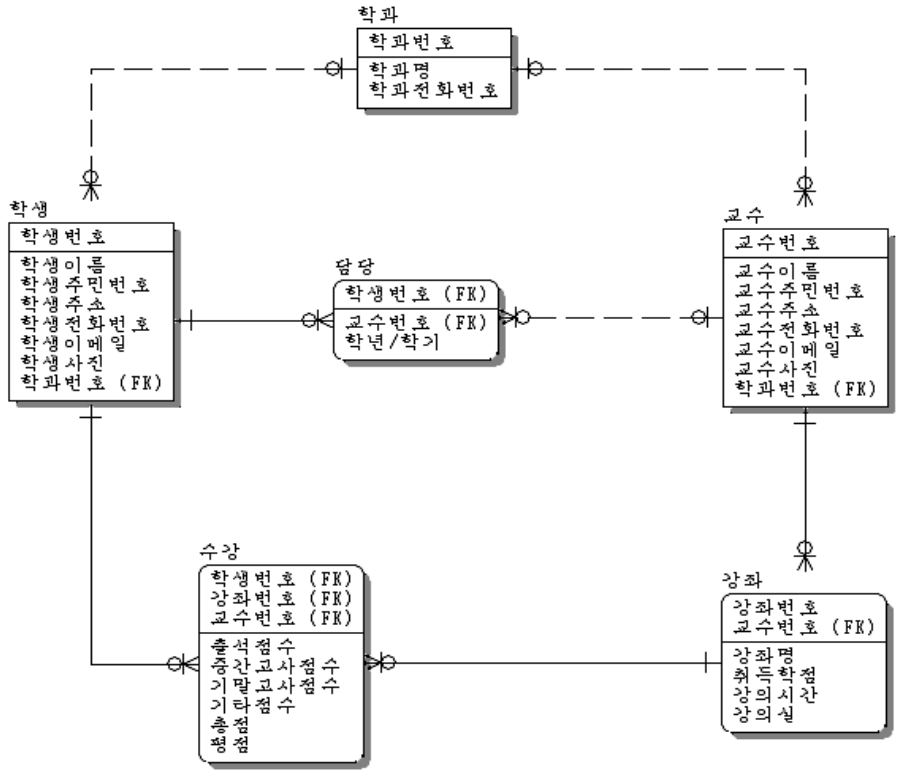
C. 데이터베이스 프로젝트

C.2 학사 관리 프로젝트(교수 전용 자료)

여기에서는 B.3절에서 만든 <학사관리> ER 다이어그램을 이용하여 학사 업무 관리 데이터베이스를 구축해본다. ER 다이어그램을 이용하여 데이터베이스의 테이블을 생성한 후 레코드를 삽입, 갱신, 삭제하는 작업을 하고 해당 데이터베이스와 계정(user)을 생성하여 관련 테이블의 Object를 생성하여 저장하는 작업을 한다.

과제 1 : ER 다이어그램 구축 및 계정(user) 생성

이 과제에서는 B.3절에서 만든 <학사관리> ERD로 테이블 설계를 한다.



계정(user)과 데이터베이스를 다음을 참고하여 생성한다.

```
create user U_HAKSA@localhost identified by 'password';
create database HAKSA;
grant all privileges on HAKSA.* to U_HAKSA@localhost with grant option;
```

과제 2 : 테이블 구축

과제 1에서 구성한 ERD를 근거로 각각의 테이블 구조, 관계, 제약조건 등을 설정하면 다음과 같다.

① 테이블명 : HS_STUDENTS(학생)

테이블 구조, 관계, 제약조건

개체	HS_STUDENTS(학생)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	학생번호	hs_stu_id	VARCHAR	10	Not Null / Unique
	학과번호	hs_class_id	VARCHAR	10	
	학생번호	hs_stu_id	VARCHAR	10	
외래키	교수번호	hs_prof_id	VARCHAR	10	
	학생이름	hs_stu_name	VARCHAR	15	Not Null
	학생주민번호	hs_stu_num	VARCHAR	14	Not Null
	학생주소	hs_stu_addr	VARCHAR	100	Not Null
속성	학생전화번호	hs_stu_phnum	VARCHAR	15	Not Null
	학생이메일	hs_stu_email	VARCHAR	100	Not Null
	학생사진	hs_stu_image	LONGBLOB	.	
	학년/학기	hs_stu_sub	VARCHAR	3	Not Null

테이블 생성 구문

```
CREATE TABLE HS_STUDENTS (
        hs_stu_id           VARCHAR(10),
         class_off_id               VARCHAR(10) NOT NULL,
         hs_prof_id                 VARCHAR(10) NOT NULL,
         hs_stu_name                VARCHAR(15) NOT NULL,
         hs_stu_jumin               VARCHAR(14) NOT NULL,
         hs_stu_addr                VARCHAR(100) NOT NULL,
         hs_stu_phnum               VARCHAR(15) NOT NULL,
        hs_stu_email                VARCHAR(100) NOT NULL,
        hs_stu_image                LONGBLOB,
        hs_stu_sub                  VARCHAR(3) NOT NULL
);
ALTER TABLE HS_STUDENTS ADD CONSTRAINT hs_stu_id_pk PRIMARY KEY (hs_stu_id);

ALTER TABLE HS_STUDENTS ADD CONSTRAINT hs_stu_id_fk1 FOREIGN KEY (hs_stu_id) REFERENCES hs_professor (prof_id);

ALTER TABLE HS_STUDENTS ADD CONSTRAINT class_off_id_fk2 FOREIGN KEY (class_off_id) REFERENCES HS_CLASS_OFFICE (class_off_id);

ALTER TABLE HS_STUDENTS ADD CONSTRAINT hs_prof_id_fk3 FOREIGN KEY (hs_prof_id) REFERENCES HS_PROFESSOR (prof_id);
```

② 테이블명 : HS_CLASS_OFFICE (학과)

테이블 구조, 관계, 제약조건

개체	HS_CLASS_OFFICE (학과)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	학과번호	class_off_id	VARCHAR	10	Not Null / Unique
외래키					
속성	학과명	class_off_name	VARCHAR	20	Not Null
	학과전화번호	class_off_phnum	VARCHAR	15	Not Null

테이블 생성 구문

```
CREATE TABLE HS_CLASS_OFFICE (
        class_off_id            VARCHAR(10) ,
        class_off_name          VARCHAR(20) NOT NULL,
        class_off_phnum         VARCHAR(15) NOT NULL
);

ALTER TABLE HS_CLASS_OFFICE ADD CONSTRAINT class_off_id_pk PRIMARY KEY
(class_off_id);
```

③ 테이블명 : HS_PROFESSOR (교수)

테이블 구조, 관계, 제약조건

개체	HS_PROFESSOR (교수)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	교수번호	prof_id	VARCHAR	10	Not Null / Unique
외래키	학과번호	class_off_id	VARCHAR	10	Not Null
	학생번호	hs_stu_id	VARCHAR	10	Not Null
속성	교수이름	prof_name	VARCHAR	15	Not Null
	교수주민번호	prof_jumin	VARCHAR	14	Not Null
	교수주소	prof_addr	VARCHAR	100	Not Null
	교수전화번호	prof_phnum	VARCHAR	15	Not Null
	교수이메일	prof_email	VARCHAR	100	Not Null
	교수사진	prof_image	LONGBLOB	.	

테이블 생성 구문

```
CREATE TABLE HS_PROFESSOR (
        prof_id                 VARCHAR(10),
        prof_name               VARCHAR(15) NOT NULL,
        prof_jumin              VARCHAR(14) NOT NULL UNIQUE,
        prof_addr               VARCHAR(100) NOT NULL,
        prof_phnum              VARCHAR(15) NOT NULL,
        prof_email              VARCHAR(100) NOT NULL,
        prof_image              LONGBLOB,
        class_off_id            VARCHAR(10) NOT NULL,
        hs_stu_id       VARCHAR(10) NOT NULL
);

ALTER TABLE HS_PROFESSOR ADD CONSTRAINT hs_prof_id_pk PRIMARY KEY (prof_id);

ALTER TABLE HS_PROFESSOR ADD CONSTRAINT hs_prof_stu_id_fk1 FOREIGN KEY
(hs_stu_id) REFERENCES hs_students (hs_stu_id);

ALTER TABLE HS_PROFESSOR ADD CONSTRAINT hs_prof_class_off_id_fk2 FOREIGN
KEY (class_off_id) REFERENCES HS_CLASS_OFFICE (class_off_id);
```

④ 테이블명 : HS_LECTURE (강좌)

테이블 구조, 관계, 제약조건

개체	HS_LECTURE (강좌)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	강좌번호	lec_id	VARCHAR	10	Not Null / Unique
외래키	교수번호	prof_id	VARCHAR	10	Not Null
속성	강좌명	lec_name	VARCHAR	15	Not Null
	취득학점	lec_arch_grade	char	1	Not Null
	강의시간	lec_time	INTEGER	1	Not Null
	강의실	lec_room	VARCHAR	10	Not Null

테이블 생성 구문

```
CREATE TABLE HS_LECTURE (
        lec_id                  VARCHAR(10),
        lec_name                VARCHAR(15) NOT NULL,
        lec_arch_grade          char(1) NOT NULL,
        lec_time        INTEGER NOT NULL,
        lec_room        VARCHAR(10) NOT NULL,
  prof_id               VARCHAR(10)
);

ALTER TABLE HS_LECTURE ADD CONSTRAINT hr_lec_id_pk PRIMARY KEY (lec_id);

ALTER TABLE HS_LECTURE ADD CONSTRAINT hs_lec_prof_id_fk FOREIGN KEY
(prof_id) REFERENCES hs_professor (prof_id);
```

⑤ 테이블명 : REGIS_COURSE (수강)

테이블 구조, 관계, 제약조건

개체	REGIS_COURSE (수강)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	학생번호 강좌번호 교수번호	hs_stu_id lec_id prof_id	VARCHAR VARCHAR VARCHAR	10 10 10	Not Null / Unique
외래키	학생번호	hs_stu_id	VARCHAR	10	
	강좌번호	lec_id	VARCHAR	10	
속성	출석점수	rc_atten_grade	INTEGER	2	Not Null
	중간고사점수	rc_midterm_grade	INTEGER	2	Not Null
	기말고사점수	rc_finalterm_grade	INTEGER	2	Not Null
	기타점수	rc_else_grade	INTEGER	2	Not Null
	총점	rc_total	INTEGER	3	Not Null
	평점	rc_avg_grade	char	1	

테이블 생성 구문

```
CREATE TABLE REGIS_COURSE (
        rc_atten_grade          INTEGER,
        rc_midterm_grade INTEGER,
        rc_finalterm_grade      INTEGER,
        rc_else_grade           INTEGER,
        rc_total        INTEGER,
        rc_avg_grade            char(1),
        hs_stu_id       VARCHAR(10),
        lec_id                  VARCHAR(10),
        prof_id                 VARCHAR(10)
        );

ALTER TABLE REGIS_COURSE ADD CONSTRAINT rc_stu_lec_prof_id PRIMARY KEY (hs_stu_id, lec_id, prof_id);

ALTER TABLE REGIS_COURSE ADD CONSTRAINT rc_lec_id_fk1 FOREIGN KEY (lec_id) REFERENCES HS_LECTURE (lec_id);

ALTER TABLE REGIS_COURSE ADD CONSTRAINT rc_stu_id_fk2 FOREIGN KEY (hs_stu_id) REFERENCES HS_STUDENTS (hs_stu_id);
```

과제 3 : 데이터 입력 (INSERT)

생성된 다섯 개의 테이블에 다음과 같은 정보를 입력한다. 각 테이블 컬럼의 데이터 타입 및 사이즈를 확인한 후 정확히 입력하도록 한다.

예) INSERT INTO HS_STUDENTS
VALUES ('04154123', '145', '1234', '서창호', '750813-1234567', '서울시 성동구', '010-4322-9876', 'sch@ab.com', null, '2-1');

① 테이블명 : HS_STUDENTS(학생)

② 테이블명 : HS_CLASS_OFFICE (학과)

③ 테이블명 : HS_PROFESSOR (교수)

④ 테이블명 : HS_LECTURE (강좌)

⑤ 테이블명 : REGIS_COURSE (수강)

과제 4 : 데이터 검색 (Query)

(1) 교수번호 145에게 수업을 받는 학생에 대한 정보를 출력하시오.

(2) 컴퓨터공학과에 재학중인 학생과 강의하는 교수에 대한 모든 정보를 출력하시오.

(3) 현재 2학년 2학기에 재학 중인 학생에 대한 정보를 오름차순 정렬하여 출력하시오.

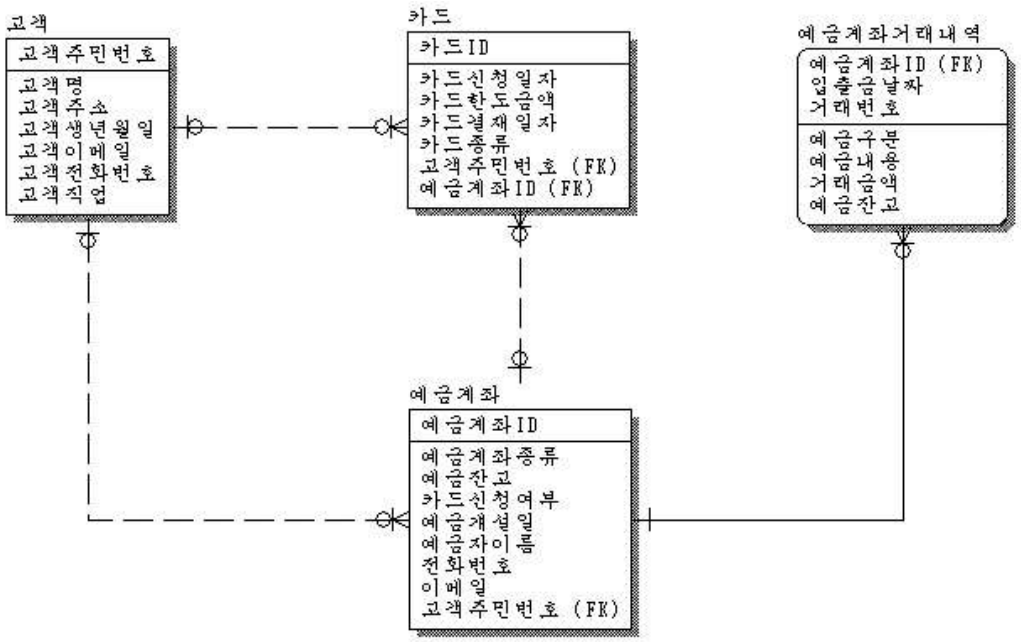
(4) 학생ID가 04로 시작하는 모든 학생 정보를 출력하시오.

C.3 은행 업무 관리 프로젝트(교수 전용 자료)

여기에서는 B.4절에서 만든 <은행업무 관리> ER 다이어그램을 이용하여 은행 업무 관리 데이터베이스를 구축해본다. ER 다이어그램을 이용하여 데이터베이스의 테이블을 생성한 후 레코드를 삽입, 갱신, 삭제하는 작업을 하고 해당 데이터베이스와 계정(user)을 생성하여 관련 테이블의 Object를 생성하여 저장하는 작업을 한다.

과제 1 : ER 다이어그램 구축 및 계정(Schema) 생성

이 과제에서는 B.4절에서 만든 <은행업무 관리> ERD로 테이블 설계를 한다.



계정(user)과 데이터베이스를 다음을 참고하여 생성한다.

```
create user U_BANK@localhost identified by 'password';
create database BANK;
grant all privileges on BANK.* to U_BANK@localhost with grant option;
```

과제 2 : 테이블 구축

과제 1에서 구성한 ERD를 근거로 각각의 테이블 구조 및 관계, 제약조건 등을 설정하면 다음과 같다.

① 테이블명 : B_CUSTOMERS (고객)

테이블 구조, 관계, 제약조건

개체	B_CUSTOMERS (고객)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	고객주민번호	cust_jumin	VARCHAR	20	Not Null / Unique
외래키					
속성	고객명	cust_name	VARCHAR	20	Not Null
	고객주소	cust_addr	VARCHAR	100	Not Null
	고객생년월일	cust_birth	date		Not Null
	고객이메일	cust_email	VARCHAR	50	Not Null
	고객전화번호	cust_phnum	VARCHAR	25	Not Null
	고객직업	cust_job	VARCHAR	30	

테이블 생성 구문

```
CREATE TABLE B_CUSTOMERS (
   cust_jumin VARCHAR(20),
   cust_name VARCHAR(20) NOT NULL,
   cust_addr VARCHAR(100) NOT NULL,
   cust_birth date,
   cust_email VARCHAR(50) NOT NULL UNIQUE,
   cust_phnum VARCHAR(25) NOT NULL,
   cust_job VARCHAR(30)
   );

ALTER TABLE B_CUSTOMERS ADD CONSTRAINT cust_jumin_pk PRIMARY KEY (cust_jumin);
```

② 테이블명 : B_CARDS (카드)

테이블 구조, 관계, 제약조건

개체	B_CARDS (카드)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	카드ID	card_id	VARCHAR	15	Not Null / Unique
외래키	고객주민번호	cust_jumin	VARCHAR	20	
	예금계좌ID	acc_id	VARCHAR	10	
속성	카드신청일자	card_register_date	date		
	카드한도금액	card_limit_money	BIGINT		
	카드결재일자	card_approve_date	date		
	카드종류	card_type	VARCHAR	10	Not Null

테이블 생성 구문

```
CREATE TABLE B_CARDS (
  card_id VARCHAR(15),
  card_register_date date,
  card_limit_money BIGINT,
  card_approve_date date,
  card_type VARCHAR(10) NOT NULL,
  cust_jumin VARCHAR(20),
  acc_id VARCHAR(10)
  );

ALTER TABLE B_CARDS ADD CONSTRAINT card_id_pk PRIMARY KEY (card_id);

ALTER TABLE B_CARDS ADD CONSTRAINT card_cust_jumin_fk1 FOREIGN KEY (cust_jumin) REFERENCES B_CUSTOMERS (cust_jumin);

ALTER TABLE B_CARDS ADD CONSTRAINT card_acc_id_fk2 FOREIGN KEY (acc_id) REFERENCES B_ACCOUNTS (acc_id);
```

③ 테이블명 : B_ACCOUNTS (예금계좌)

테이블 구조, 관계, 제약조건

개체	B_ACCOUNTS (예금계좌)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	예금계좌ID	acc_id	VARCHAR	10	Not Null / Unique
외래키	고객주민번호	cust_jumin	VARCHAR	20	
속성	예금계좌종류	acc_type	VARCHAR	20	Not Null
	예금잔고	acc_balance	BIGINT		Not Null
	카드신청여부	card_ask	VARCHAR	5	Not Null
	예금개설일	acc_register_date	date		
	예금자이름	acc_cust_name	VARCHAR	20	Not Null
	전화번호	acc_phnum	VARCHAR	20	
	이메일	acc_email	VARCHAR	50	Not Null

테이블 생성 구문

```
CREATE TABLE B_ACCOUNTS (
  acc_id VARCHAR(10),
  acc_type VARCHAR(20) NOT NULL,
  acc_balance BIGINT NOT NULL,
  card_ask VARCHAR(5) NOT NULL,
  acc_register_date date,
  acc_cust_name VARCHAR(20) NOT NULL,
  acc_phnum VARCHAR(20),
  acc_email VARCHAR(50) NOT NULL UNIQUE,
  cust_jumin VARCHAR(20)
  );

ALTER TABLE B_ACCOUNTS ADD CONSTRAINT acc_id_pk PRIMARY KEY (acc_id);

ALTER TABLE B_ACCOUNTS ADD CONSTRAINT acc_cust_jumin_fk FOREIGN KEY (cust_jumin) REFERENCES B_CUSTOMERS (cust_jumin);
```

④ 테이블명 : B_ACC_TRADES (예금계좌거래내역)

테이블 구조, 관계, 제약조건

개체	B_ACC_TRADES (예금계좌거래내역)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	예금계좌ID 입출금날짜 거래번호	acc_id imp_exp_date trade_id	VARCHAR date number	10	Not Null / Unique
외래키	예금계좌ID	acc_id	VARCHAR	10	
속성	예금구분	acc_class	VARCHAR	20	
	예금내용	acc_contents	VARCHAR	50	
	거래금액	trade_money	BIGINT		
	예금잔고	acc_balance	BIGINT		Not Null

테이블 생성 구문

```
CREATE TABLE B_ACC_TRADES (
  acc_id VARCHAR(10),
  imp_exp_date date,
  trade_id INTEGER,
  acc_class VARCHAR(20),
  acc_contents VARCHAR(50),
  trade_money BIGINT,
  acc_balance BIGINT NOT NULL
  );

ALTER TABLE B_ACC_TRADES ADD CONSTRAINT acc_trade_id_pk PRIMARY KEY (acc_id,imp_exp_date,trade_id);

ALTER TABLE B_ACC_TRADES ADD CONSTRAINT acc_trade_id_fk FOREIGN KEY (acc_id) REFERENCES B_ACCOUNTS (acc_id);
```

과제 3 : 데이터 입력 (INSERT)

생성된 네 개의 테이블에 다음과 같은 정보를 입력한다. 각 테이블 컬럼의 데이터 타입 및 사이즈를 확인한 후 정확히 입력하도록 한다.

예) INSERT INTO B_CUSTOMERS
VALUES ('850813-1234567', '서창호', '서울시 성동구', '85-08-13', 'sch@ab.com', '010-4322-9876', '주방장');

① 테이블명 : B_CUSTOMERS (고객)

② 테이블명 : B_CARDS (카드)

③ 테이블명 : B_ACCOUNTS (예금계좌)

④ 테이블명 : B_ACC_TRADES (예금계좌거래내역)

과제 4 : 데이터 검색 (Query)

(1) 카드를 신청한 모든 고객에 대한 정보를 출력하시오.

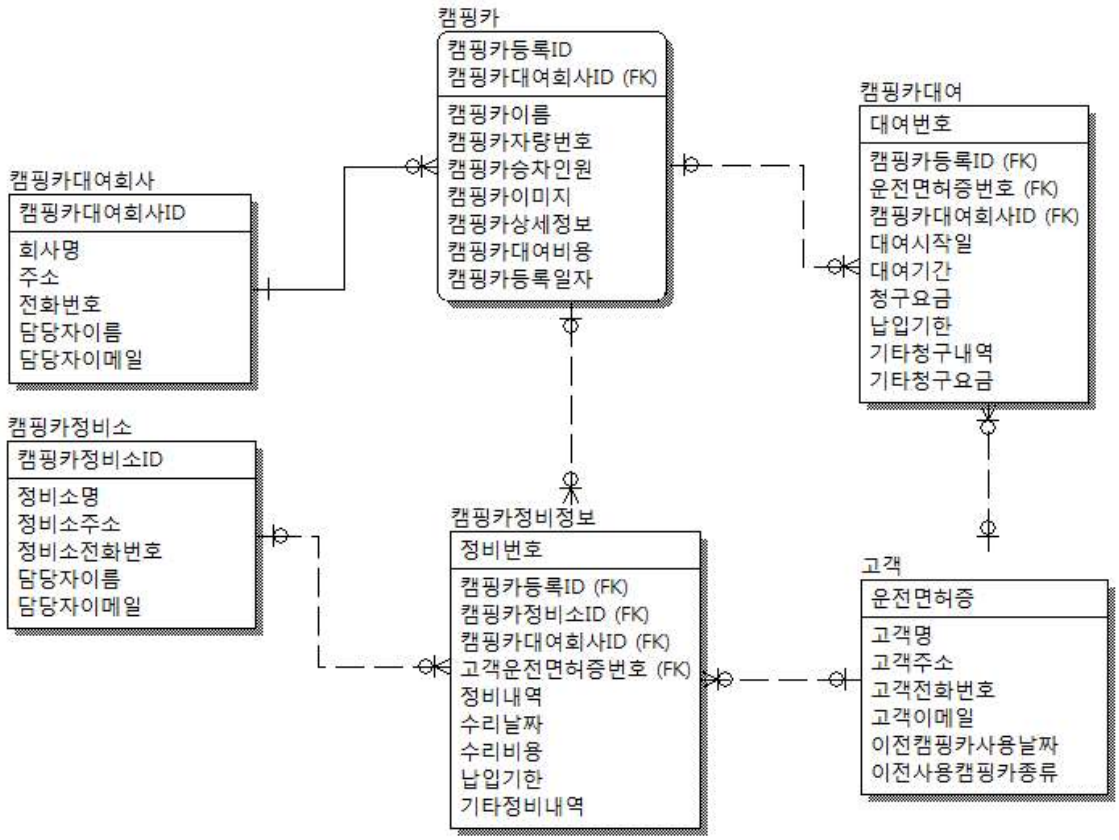
(2) 예금 잔고가 ₩1,000,000 ~ ₩5,000,000 인 고객에 대한 정보를 출력하시오.

(3) 카드종류가 VISA인 고객의 예금 잔고를 출력하시오.

C.4 캠핑카 대여 관리 프로젝트(교수 전용 자료)

과제 1 : ER 다이어그램 구축 및 계정(schema) 생성

본 과제에서는 B.5절에서 만든 <캠핑카 대여 관리> ERD를 이용하여 테이블 설계를 한다.



계정(user)과 데이터베이스를 다음을 참고하여 생성한다.

```
create user U_CRENT@localhost identified by 'password';
create database C_CAR_RENT;
grant all privileges on C_CAR_RENT.* to U_CRENT@localhost with grant option;
```

과제 2 : 테이블 구축

과제 1에서 구성한 ERD를 근거로 각각의 테이블 구조 및 관계, 제약조건 등을 설정하면 다음과 같다.

① 테이블명 : 캠핑카대여회사(C_CAR_RENT_COMP)

테이블 구조, 관계, 제약조건

개체	C_CAR_RENT_COMP(캠핑카 대여회사)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	캠핑카대여회사ID	c_car_rent_comp_id	VARCHAR	10	Not Null / Unique
외래키					
속성	회사명	c_comp_name	VARCHAR	25	Not Null
	주소	c_comp_addr	VARCHAR	100	Not Null
	전화번호	c_comp_phone	VARCHAR	15	Not Null
	담당자이름	c_comp_admin_name	VARCHAR	25	Not Null
	담당자이메일	c_comp_admin_email	VARCHAR	50	Not Null

테이블 생성 구문

```
CREATE TABLE C_CAR_RENT_COMP (''
        c_car_rent_comp_id    VARCHAR(10) NOT NULL,
        c_comp_name            VARCHAR(25) NOT NULL,
        c_comp_addr            VARCHAR(100) NOT NULL,
        c_comp_phone           VARCHAR(15) NOT NULL,
        c_comp_admin_name      VARCHAR(25) NOT NULL,
        c_comp_admin_email     VARCHAR(50) NOT NULL
);

ALTER TABLE C_CAR_RENT_COMP ADD CONSTRAINT c_rent_comp_id_pk PRIMARY KEY
(c_car_rent_comp_id);
```

② 테이블명 : 캠핑카(C_CARS)

테이블 구조, 관계, 제약조건

개체	C_CARS(캠핑카)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	캠핑카등록ID	c_car_reg_id	VARCHAR	10	Not Null / Unique
	캠핑카대여회사ID	c_car_rent_comp_id	VARCHAR	10	Not Null / Unique
외래키	캠핑카대여회사ID	c_car_rent_comp_id	VARCHAR	10	
속성	캠핑카이름	c_car_name	VARCHAR	25	Not Null
	캠핑카차량번호	c_car_number	VARCHAR	10	Not Null
	캠핑카승차인원	c_car_capa	INTEGER		Not Null
	캠핑카상세정보	c_car_detail	VARCHAR	1000	Not Null
	캠핑카대여비용	c_car_rent_price	INTEGER		Not Null
	캠핑카등록일자	c_car_reg_date	date		Not Null

테이블 생성 구문

```
CREATE TABLE C_CARS (
        c_car_reg_id            VARCHAR(10) NOT NULL,
        c_car_rent_comp_id     VARCHAR(10) NOT NULL,
        c_car_name              VARCHAR(25) NOT NULL,
        c_car_number            VARCHAR(10) NOT NULL,
        c_car_capa              INTEGER NOT NULL,
        c_car_detail            VARCHAR(1000) NOT NULL,
        c_car_rent_price        INTEGER NOT NULL,
        c_car_reg_date          date NOT NULL
);


ALTER TABLE C_CARS ADD CONSTRAINT c_car_rent_comp_id PRIMARY KEY
(c_car_reg_id, c_car_rent_comp_id);

ALTER TABLE C_CARS
        ADD  CONSTRAINT c_car_rent_comp_id_fk FOREIGN KEY (c_car_rent_comp_id)
                        REFERENCES C_CAR_RENT_COMP (c_car_rent_comp_id) ;
```

③ 테이블명 : 캠핑카대여(C_CAR_RENT)

테이블 구조, 관계, 제약조건

개체	C_CAR_RENT(캠핑카대여)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	대여번호	c_car_rent_id	INTEGER		Not Null / Unique
외래키	캠핑카등록ID	c_car_reg_id	VARCHAR	10	
	운전면허증번호	cust_driver_license_id	varcahr2	25	
	캠핑카대여회사ID	c_car_rent_comp_id	VARCHAR	10	
속성	대여시작일	c_car_rent_start_date	date		Not Null
	대여기간	c_car_rent_period	INTEGER		Not Null
	청구요금	charge_price	INTEGER		Not Null
	납입기한	price_deadline	date		Not Null
	기타청구내역	add_amount_contents	VARCHAR	300	
	기타청구요금	add_amount_price	BIGINT	10	

테이블 생성 구문

```
CREATE TABLE C_CAR_RENT (
        c_car_rent_id          INTEGER NOT NULL,
        c_car_reg_id           VARCHAR(10) ,
        cust_driver_license_id VARCHAR(25) ,
        c_car_rent_comp_id     VARCHAR(10) ,
        c_car_rent_start_date  date NOT NULL,
        c_car_rent_period      INTEGER NOT NULL,
        charge_price           INTEGER NOT NULL,
        price_deadline         date NOT NULL,
        add_amount_contents    VARCHAR(300) ,
        add_amount_price       BIGINT
);

ALTER TABLE C_CAR_RENT ADD CONSTRAINT c_rent_id_pk PRIMARY KEY
(c_car_rent_id);

ALTER TABLE C_CAR_RENT
        ADD  CONSTRAINT c_car_dlicense_fk FOREIGN KEY (cust_driver_license_id)
                        REFERENCES CUSTOMERS(cust_driver_license_id);

ALTER TABLE C_CAR_RENT
        ADD     CONSTRAINT c_car_rent_reg_comp_id_fk FOREIGN KEY (c_car_reg_id,
c_car_rent_comp_id)
                        REFERENCES C_CARS (c_car_reg_id, c_car_rent_comp_id) ;
```

④ 테이블 : 고객(CUSTOMERS)

테이블 구조, 관계, 제약조건

개체	CUSTOMERS(고객)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	운전면허번호	cust_driver_license_id	VARCHAR	25	Not Null / Unique
외래키					
속성	고객명	cust_name	VARCHAR	20	Not Null
	고객주소	cust_addr	VARCHAR	100	Not Null
	고객전화번호	cust_phone	VARCHAR	15	Not Null
	고객이메일	cust_email	VARCHAR	50	Not Null
	이전캠핑카이용날짜	cust_past_used_date	date		
	이전사용캠핑카종류	cust_past_used_type	VARCHAR	20	

테이블 생성 구문

```
CREATE TABLE CUSTOMERS (
        cust_driver_license_id VARCHAR(25) NOT NULL,
        cust_name               VARCHAR(20) NOT NULL,
        cust_addr              VARCHAR(100) NOT NULL,
        cust_phone              VARCHAR(15) NOT NULL,
        cust_email             VARCHAR(50) NOT NULL,
        cust_past_used_date   date ,
        cust_past_used_type   VARCHAR(20)
);

ALTER TABLE CUSTOMERS ADD CONSTRAINT cust_dlicense_id_pk PRIMARY KEY
(cust_driver_license_id);
```

⑤ 테이블 : 캠핑카정비정보(C_CAR_REPAIR_INFORM)

테이블 구조, 관계, 제약조건

개체	C_CAR_REPAIR_INFORM(캠핑카정비정보)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	정비번호	repair_number	VARCHAR	25	Not Null / Unique
외래키	캠핑카등록ID	c_car_reg_id	VARCHAR	10	
	캠핑카정비소ID	c_car_garage_id	VARCHAR	10	
	캠핑카대여회사ID	c_car_rent_comp_id	VARCHAR	10	
	고객운전면허번호	cust_driver_license_id	VARCHAR	25	
속성	정비내역	repair_contents	VARCHAR	15	Not Null
	수리날짜	repair_date	date		Not Null
	수리비용	repair_price	BIGINT		Not Null
	납입기한	price_deadline	date		Not Null
	기타정비내역	add_repair_contents	VARCHAR	300	

테이블 생성 구문

```
CREATE TABLE C_CAR_REPAIR_INFORM (
        repair_number          VARCHAR(25) NOT NULL,
        c_car_reg_id           VARCHAR(10) ,
        c_car_garage_id        VARCHAR(10) ,
        c_car_rent_comp_id     VARCHAR(10) ,
        cust_driver_license_id VARCHAR(25) ,
        repair_contents        VARCHAR(15) NOT NULL,
        repair_date            date NOT NULL,
        repair_price           BIGINT NOT NULL,
        price_deadline         date NOT NULL,
        add_repair_contents    VARCHAR(300) NULL
);

ALTER TABLE C_CAR_REPAIR_INFORM ADD CONSTRAINT repair_number_pk PRIMARY
KEY (repair_number);

ALTER TABLE C_CAR_REPAIR_INFORM
        ADD          CONSTRAINT      c_car_repair_inform_cust_fk      FOREIGN      KEY
(cust_driver_license_id)
                              REFERENCES CUSTOMERS (cust_driver_license_id) ;
```

```
ALTER TABLE C_CAR_REPAIR_INFORM
        ADD   CONSTRAINT c_car_inform_garage_id_fk FOREIGN KEY (c_car_garage_id)
                                REFERENCES C_CAR_GARAGE (c_car_garage_id) ;

ALTER TABLE C_CAR_REPAIR_INFORM
        ADD   CONSTRAINT c_repair_inform_car_comp_id_fk FOREIGN KEY (c_car_reg_id,
c_car_rent_comp_id)
                                REFERENCES C_CARS (c_car_reg_id, c_car_rent_comp_id) ;
```

⑥ 테이블명 : 캠핑카정비소(C_CAR_GARAGE)

테이블 구조, 관계, 제약조건

개체	C_CAR_GARAGE(캠핑카정비소)				
구분	Logical	Physical			
		Column	Data type	Size	Null / Unique
기본키	캠핑카정비소ID	c_car_garage_id	VARCHAR	10	Not Null / Unique
외래키					
속성	정비소명	garage_name	VARCHAR	15	Not Null
	정비소주소	garage_addr	VARCHAR	100	Not Null
	정비소전화번호	garage_phone	VARCHAR	15	Not Null
	담당자이름	garage_admin_name	VARCHAR	25	Not Null
	담당자이메일	garage_admin_email	VARCHAR	50	Not Null

테이블 생성 구문

```
CREATE TABLE C_CAR_GARAGE (
        c_car_garage_id        VARCHAR(10) NOT NULL,
        garage_name             VARCHAR(15) NOT NULL,
        garage_addr             VARCHAR(100) NOT NULL,
        garage_phone            VARCHAR(15) NOT NULL,
        garage_admin_name      VARCHAR(25) NOT NULL,
        garage_admin_email     VARCHAR(50) NOT NULL
);

ALTER TABLE C_CAR_GARAGE  ADD CONSTRAINT c_garage_id_pk PRIMARY
KEY (c_car_garage_id);
```

과제 3 : 데이터 입력 (INSERT)

생성된 여섯 개의 테이블에 다음과 같은 정보를 입력한다. 각 테이블 컬럼의 데이터 타입 및 사이즈를 확인한 후 정확히 입력하도록 한다.

예) INSERT INTO C_CAR_RENT_COMP
VALUES ('33345', '마당캠핑카', '서울시 마포구', '010-5344-6231', '이태정', 'ltj@cc.com');

① 테이블명 : C_CAR_RENT_COMP(캠핑카 대여회사)

② 테이블명 : C_CARS(캠핑카)

③ 테이블명 : C_CAR_RENT(캠핑카대여)

④ 테이블명 : CUSTOMERS(고객)

⑤ 테이블명 : C_CAR_REPAIR_INFORM(캠핑카정비정보)

⑥ 테이블명 : C_CAR_GARAGE(캠핑카정비소)

과제 4 : 데이터 검색 (Query)

(1) 2013년7월~8월까지 캠핑카를 대여한 모든 고객에 대한 정보를 출력하시오.

(2) 서울에 있는 캠핑카 정비소에 대한 모든 정보를 출력하시오.

(3) 캠핑카 승차 인원이 5인 이상인 캠핑카의 모든 정보를 출력하시오.