2024 년 1 학기 데이터베이스 ISQL 과제

숙명여자대학교 공과대학 인공지능공학부

Issue: 2024.10.25.

교수 박 영 호

1. Subject

   관계형 데이터베이스 관리 시스템에서 Interactive SQL (10 점 만점)

2. Due Date (2 주간)

   2024 년 11 월 8 일(금) 22 시 정시 마감, snowboard

3. Spec

   1) 사용 시스템 : **DBMS 선택에 제한 없이 어떠한 제품이든 자신의 PC 에서 설치 (MySQL, MS SQL, Oracle 등 상용 데이터베이스 자유 선택, 단 SQLite 는 기능 부족)**

   2) 실습 중에 다룰 내용 : isql (interactive SQL)의 사용

4. 과제 제출 방법

   질의를 작성하여 "그 질의 번호(그룹-번호)와 수행 결과 화면를 캡춰"하여 **Word** 나 **HWP** 에 순차적으로 담아 제출 문서를 작성한다. 그리고, **SnowBoard 에 제출**한다.

5. 제출 형식과 방법

   별도로 제시한 .doc 파일 참조

6. 채점 기준

   (1) 본인의 PC 에 선택한 DBMS 를 설치한 후, 주어진 지문에 대한 24 개의 질의를 수행합니다. 질의 결과 화면을 캡처하여 hwp 파일 또는 doc 파일로 제출 기한 내에 제출한 학생에게 점수를 부여한다.

   (2) 제출 기한이 지나면, 담당 교수는 시스템 상황을 포함한 어떠한 경우에도 과제를 수신하지 않는다. 개인적인 메일, 카톡, 문자 등으로도 과제 수신이 불가하며, 스노우보드 시스템에 제한 시간 내에 제출되지 않은 과제는 0 점을 부여한다.

   (3) PDF 파일 제출 불가 – 학생이 제출한 SQL 질의는 담당 교수가 copy & paste 하여 담당 교수의 PC 환경에서 실행할 수 있도록 hwp 또는 doc 파일로 제출해야 한다. PDF 로 제출된 경우 copy & paste 가 불가능하여 검사할 수 없으므로, 점수를

부여하지 않을 수 있다.

(4) 학생이 스스로 지문의 영문을 해석하고, 이해한 대로 질의 작성 문제를 해결했다고 인정한 경우 점수를 부여한다. 즉, 질의 결과는 사용한 DBMS 나 문제 해석에 따라 다를 수 있으므로, 해석이 혼란스러운 경우:

A. 각 문제별로 이해한 내용을 붉은색으로 부연 설명하고,
B. 그 설명대로 질의어를 작성하여 결과를 제출하며,
C. 본인의 생각대로 답이 나왔다면, 예상된 답이 아니더라도 담당 교수는 답으로 인정할 수 있다.
D. 단, 문제를 풀기 전에 본인의 가정 사항을 붉은색 글자로 명확히 기술한 경우에 한한다.

(5) 24 문제 중 본인이 직접 푼 문제가 만약 15 개면, 7.0 점, 개수 별 점수 체계 상이함.

< 배 점 표 >

정답 문항#	01	02	03	04	05	06	07	08
점수	1.0	1.5	2.0	2.5	3.0	3.4	3.8	4.2
정답 문항#	09	10	11	12	13	14	15	16
점수	4.6	5.0	5.4	5.8	6.2	6.6	7.0	7.4
정답 문항#	17	18	19	20	21	22	23	24
점수	7.7	8.0	8.3	8.6	8.9	9.2	9.5	10

본인 스스로 풀기

하루를 비워두고 풀기..

문제가 연관되어있기에 순차적으로 풀이하기

1~24번 문제까지 쭉 읽어보고 상의하기(하루를 비워두고)

<ISQL 과제 내용 설명>

본 실습 과정 동안 "suppliers-parts-projects"라는 데이타베이스를 사용한다. 이 데이타베이스는 예제와 숙제에 사용되는 sample database 이므로 실습 전에 데이타베이스의 스키마를 숙지하여야 한다. 이후, 문제로 주어진 6 개의 질의 그룹과 그에 속한 **24 문항을 반드시 순서대로 처리해야 한다**. 순서대로 지문의 지시에 따라 수행해야 한다.

다음은 sample database 에 대한 설명이다.

㈜ 어떤 상용데이터베이스는 특수 문자인 '#'을 속성명에사용할 수 없는 경우가 있다. 이때는 S# 대신 S_shap 과 같은 방법으로 속성명을 변경하여 스키마를 정의하여야 한다.

Sample Database 이름: suppliers-parts-projects (공급자-부품-프로젝트)

* **Table "Supplier"** : Supplier 에 대한 테이블이다. Supplier 에 대한 튜플은 supplier number (S#), supplier name (SNAME), rating or status value (STATUS), location (CITY)들로 이루어져 있다. 여기서 supplier number 는 supplier 마다 서로 다른 값을 갖는다. Supplier name 은 서로 다른 supplier 가 같은 name 을 가질 수도 있다. 또한 예제를 간단히 하기 위하여, 각 supplier 는 단지 한 도시에만 위치할 수 있다고 가정한다. STATUS 는 NULL 값을 가질 수 있다.

* **Table "Part"** : Part(정확히 말하면 part 의 종류)에 대한 테이블이다. Part 에 대한 튜플은 part number (P#), part name (PNAME), color (COLOR), weight (WEIGHT), 해당 part 가 저장되어 있는 location (CITY)들로 이루어져 있다. 여기서 각 part 는 part number 를 가지고 유일하게 식별이 가능하다. 또한 예제를 간단히 하기 위하여, 각 part 는 오직 한가지 color 만 가지며, 단 한군데의 city 에만 저장되어 있다고 가정한다.

* **Table "Project"** : Project 에 대한 테이블이다. Project 에 대한 튜플은 project number (J#), project name (JNAME), location (CITY)들로 이루어져 있다. 여기서 각 project 는 project number 를 가지고 유일하게 식별이 가능하다.

* **Table "SPJ"** : Supplier, part, project 사이의 관계를 나타내는 테이블이다. 즉, 각 튜플은 특정 supplier 가 어떤 project 에 무슨 part 를 얼마만큼 공급하고 있는지를 나타내기 위한, supplier number (S#), part number (P#), project number (J#), supplied quantity (Q#)로 구성되어 있다. 여기서 supplier number, part number, project number 의 쌍은 각 튜플을 유일하게 식별할 수 있다. 또한 QTY 는 NULL 값을 가질 수 있다.

Supplier	**(S#,**	**SNAME,**	**STATUS,**	**CITY)**	
Part	**(P#,**	**PNAME,**	**COLOR,**	**WEIGHT,**	**CITY)**
Project	**(J#,**	**JNAME,**	**CITY)**		
SPJ	**(S#,**	**P#,**	**J#,**	**QTY)**	

직접 불러오기 해야됨!

#이 불러들여지지않으면 이름을 바꾸어서 불러옴

create database -> create scema

[기준] Sample data values for suppliers-parts-projects database

Supplier	S#	SNAME	STATUS	CITY
	S1	Smith	20	London
	S2	Jones	10	Paris
	S3	Blake	30	Paris
	S4	Clark	20	London
	S5	Adams	30	Athens

Part	P#	PNAME	COLOR	WEIGHT	CITY
	P1	Nut	Red	12	London
	P2	Bolt	Green	17	Paris
	P3	Screw	Blue	17	Rome
	P4	Screw	Red	14	London
	P5	Cam	Blue	12	Paris
	P6	Cog	Red	19	London

Project	J#	JNAME	CITY
	J1	Sorter	Paris
	J2	Punch	Rome
	J3	Reader	Athens
	J4	Console	Athens
	J5	Collator	London
	J6	Terminal	Oslo
	J7	Tape	London

SPJ	S#	P#	J#	QTY
	S1	P1	J1	200
	S1	P1	J4	700
	S2	P3	J1	400
	S2	P3	J2	200
	S2	P3	J3	200
	S2	P3	J4	500
	S2	P3	J5	600
	S2	P3	J6	400
	S2	P3	J7	800
	S2	P5	J2	100
	S3	P3	J1	200
	S3	P4	J2	500
	S4	P6	J3	300
	S4	P6	J7	300
	S5	P2	J2	200
	S5	P2	J4	100
	S5	P5	J5	500
	S5	P5	J7	100
	S5	P6	J2	200
	S5	P1	J4	100
	S5	P3	J4	200
	S5	P4	J4	800
	S5	P5	J4	400
	S5	P6	J4	500

질의 그룹 # 1: 4 questions

<단순 질의 문제>

1. Get supplier numbers for suppliers who supply project J1, in supplier number order.
   (프로젝트 J1 에 자재를 공급하는 공급업체의 공급업체 번호를, 공급업체 번호 순서대로 가져오세요)

2. Get all shipments where the quantity is in the range 300 to 750 inclusive.
   (수량이 300 에서 750 사이(포함)인 모든 배송을 가져오세요.)

<Joins 을 사용하는 질의 문제>

3. Get part numbers for parts supplied by a supplier in London to a project in London.
   (런던에 있는 공급업체가 런던에 있는 프로젝트에 공급한 부품의 부품 번호를 가져오세요.)

4. Get all pairs of city names such that a supplier in the first city supplies a project in the second city.
   (첫 번째 도시의 공급업체가 두 번째 도시의 프로젝트에 자재를 공급하는 모든 도시 이름 쌍을 가져오세요.)

질의 그룹 # 2: 5 questions

<Aggregate Functions>

5. For each part being supplied to a project, get the part number, the project number, and the corresponding total quantity.
(각 프로젝트에 공급되는 부품에 대해 부품 번호, 프로젝트 번호 및 해당 총 수량을 가져오세요.)

6. Get part numbers of parts supplied to some project in an average quantity of more than 320.
(어떤 프로젝트에 공급된 부품의 부품 번호를 평균 수량이 320 을 초과하는 것으로 가져오세요.)

<Miscellaneous>

7. Get project numbers and cities where the city has an "o" as the second letter of its name.
(도시 이름의 두 번째 글자가 'o'인 프로젝트 번호와 도시를 가져오세요.)

<Subqueries>

8. Get project numbers for projects whose city is first in the alphabetic list of such cities.
(도시 목록에서 알파벳 순서로 첫 번째에 있는 도시의 프로젝트 번호를 가져오세요.)

9. Get project numbers for projects supplied with part P1 in an average quantity greater than the greatest quantity in which any part is supplied to project J1.
(어떤 부품이든 프로젝트 J1 에 공급된 최대 수량보다 평균 수량이 더 많은 부품 P1 을 공급받는 프로젝트 번호를 가져오세요.)

질의 그룹 # 3: 4 questions

<EXISTS>

10. Get part numbers for parts supplied to any projects in London. (Use EXISTS in your solution.)
(런던의 모든 프로젝트에 공급된 부품의 부품 번호를 가져오세요. (EXISTS 를 해결방법으로 사용함))

11. Get project numbers for projects not supplied with any red part by any London suppliers.
(런던의 공급업체가 공급하지 않은 빨간 부품이 포함되지 않은 프로젝트의 프로젝트 번호를 가져오세요.)

12. Get project numbers for projects supplied with at least all parts available from supplier S1.
(공급업체 S1 로부터 공급되는 모든 부품을 적어도 모두 공급받는 프로젝트의 프로젝트 번호를 가져오세요.)

<Union>

13. Construct an ordered list of all cities in which at least one supplier, part, or project is located.
(적어도 하나의 공급업체, 부품 또는 프로젝트가 위치한 모든 도시의 순서가 있는 목록을 작성하세요.)

질의 그룹 #4: 4 questions

<Update operations>

14. Delete all projects for which there are no shipments.
    (배송이 없는 모든 프로젝트를 삭제하세요.)

15. Insert a new supplier (S10) into table “Supplier”. The name and city are Smith and New York, respectively; the status is not yet known.
    ("새 공급업체(S10)를 'Supplier' 테이블에 삽입하세요. 이름과 도시는 각각 Smith 와 New York 이며, 상태는 아직 알려지지 않았습니다.)

16. Construct a table containing a list of part numbers for parts that are supplied either by a London supplier or to a London project.
    (런던의 공급업체가 공급하거나 런던의 프로젝트에 공급되는 부품 번호 목록을 포함하는 테이블을 작성하세요.)

17. Construct a table containing a list of project numbers for projects that are either located in London or are supplied by a London supplier.
    (런던에 위치하거나 런던의 공급업체가 공급하는 프로젝트의 프로젝트 번호 목록을 포함하는 테이블을 작성하세요.)

질의 그룹 #5: 3 questions

<Data Definition Language>

18. Write a suitable set of CREATE TABLE statements for the sample database suppliers-parts-projects.
(샘플 데이터베이스 suppliers-parts-projects 를 위한 적절한 CREATE TABLE 문을 작성하세요.)

19. Write a set of CREATE INDEX statements for the sample database suppliers-parts-projects to enforce the required primary key constraints.
("샘플 데이터베이스 suppliers-parts-projects 에서 필요한 기본 키 제약 조건을 강제하기 위해 CREATE INDEX 문 집합을 작성하세요.)

20. Create the sample database with sample data values.
(샘플 데이터 값을 사용하여 샘플 데이터베이스를 생성하세요.)

질의 그룹 #6: 4 questions

<Views>

21. Create a view consisting of supplier numbers and part numbers for suppliers and parts that are not "colocated."
(같은 위치에 있지 않은 공급업체와 부품에 대한 공급업체 번호와 부품 번호로 구성된 뷰를 생성하세요.)

22. Create a view consisting of supplier records for suppliers that are located in London.
(런던에 위치한 공급업체의 공급업체 기록으로 구성된 뷰를 생성하세요.)

23. Create a view from the suppliers-parts-projects database consisting of all projects (project number and city fields only) that are supplied by supplier S1 and use part P1.
("suppliers-parts-projects 데이터베이스에서 공급업체 S1 이 공급하고 부품 P1 을 사용하는 모든 프로젝트(프로젝트 번호와 도시 필드만 포함)를 구성하는 뷰를 생성하세요.)

24. (배점 다름에 유의) Given the view definition:

```
CREATE VIEW HEAVYWEIGHTS (P#, WT, COL)
        AS SELECT P#, WEIGHT, COLOR
            FROM Part
            WHERE WEIGHT > 14;
```

show the result for each of the following SQL statement:
(다음의 SQL 문 (a)~(d) 각각에 대한 결과를 표시하세요.):

a)
```
SELECT *
FROM HEAVYWEIGHTS
WHERE COL = 'Green';
```

b)
```
UPDATE HEAVYWEIGHTS
SET COL = 'White'
WHERE WT = 18;
```

c)
```
DELETE
FROM HEAVYWEIGHTS
WHERE WT < 10;
```

d)
```
INSERT
INTO HEAVYWEIGHTS(P#, WT, COL)
VALUES ('P99', 12, 'Purle');
```

(각 그룹의 문제를 풀기 위한)
과제 수행을 위한 참고 예제 질의

공과대학 인공지능공학부, 교수 박 영 호

질의 그룹 #1

1) Simple retrieval. Get part numbers for all parts supplied.

```
SELECT  P#
FROM            SPJ;

cf.     SELECT  DISTINCT P#
        FROM            SPJ;
```

2) Retrieval of computed values. For all parts, get the part number and the weight of that part in grams (part weights are given in table "Part" in pounds).

```
SELECT  Part.P#, 'Weight in grams = ', Part.WEIGHT*454
FROM            Part;
```

3) Simple retrieval ("SELECT *"). Get full details of all suppliers.

```
SELECT  *
FROM            Supplier;
```

4) Qualified retrieval. Get supplier numbers for suppliers in Paris with status > 20.

```
SELECT  S#
FROM            Supplier
WHERE  CITY = 'Paris'
AND             STATUS > 20;
```

5) Retrieval with ordering. Get supplier numbers and status for suppliers in Paris, in descending order of status.

```
SELECT  S#, STATUS
FROM            Supplier
WHERE  CITY = 'Paris'
ORDER BY        STATUS DESC;
```

6) Simple equijoin. Get all combinations of supplier and part information such that the supplier and part in question are located in the same city (i.e., are "colocated," to coin an ugly but convenient term).

```
SELECT  Supplier.*, Part.*
FROM            Supplier, Part
WHERE  Supplier.CITY = Part.CITY;
```

7) Greater-than join. Get all combinations of supplier and part information such that the supplier city follows the part city in alphabetical order.

```
SELECT  Supplier.*, Part.*
FROM            Supplier, Part
WHERE  Supplier.CITY > Part.CITY;
```

8) Join query with an additional condition. Get all combinations of supplier information and part information where the supplier and part concerned are colocated, but omitting suppliers with status 20.

```
SELECT  Supplier.*, Part.*
FROM            Supplier, Part
WHERE  Supplier.CITY = Part.CITY
AND             Supplier.STATUS <> 20;
```

9) Retrieving specified fields from a join. Get all supplier-number/part-number combinations such that the supplier and part in question are colocated.

```
SELECT  Supplier.S#, Part.P#
FROM            Supplier, Part
WHERE  Supplier.CITY = Part.CITY;
```

10) Join of three tables. Get all supplier-number/part-number/project-number triples such that the indicated supplier, part, and project are colocated.

```
SELECT  S#, P#, J#
FROM            Supplier, Part, Project
WHERE  Supplier.CITY = Part.CITY
AND             Part.CITY = Project.CITY;
```

11) Joining a table with itself. Get all pairs of supplier numbers such that the two suppliers concerned are colocated.

```
SELECT  FIRST.S#, ECOND.S#
FROM            Supplier FIRST,     Supplier SECOND
WHERE  FIRST.CITY = SECOND.CITY
AND             FIRST.S# < SECOND.S#;
```

질의 그룹 #2

1) Aggregate function in the SELECT clause. Get the total number of suppliers.

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

2) Aggregate function in the SELECT clause, with DISTINCT. Get the total number of suppliers currently supplying parts.

```
SELECT  COUNT(DISTINCT S#)
FROM            SPJ;
```

3) Aggregate function in the SELECT clause, with a condition. Get the number of shipments for part P2.

```
SELECT  COUNT(*)
FROM            SPJ
WHERE  P# = 'P2';
```

4) Aggregate function in the SELECT clause, with a condition. Get the total quantity of part P2 supplied.

```
SELECT  SUM(QTY)
FROM            SPJ
WHERE  P# = 'P2';
```

5) Use of GROUP BY. For each part supplied, get the part number and the total shipment quantity for that part.

```
SELECT  P#, SUM(QTY)
FROM            SPJ
GROUP BY        P#;
```

6) Use of HAVING. Get part numbers for all parts supplied by more than one supplier.

```
SELECT  P#
FROM            SPJ
GROUP BY        P#
HAVING          COUNT(*) > 1;
```

7) Retrieval using LIKE. Get all parts whose names begin with the letter C.

```
SELECT  Part.*
FROM            Part
WHERE  Part.PNAME LIKE 'C%';
```

8) Retrieval involving NULL. Get supplier numbers for suppliers with status greater than 25.

```
SELECT  S#
FROM            Supplier
WHERE  STATUS > 25;
```

9) Retrieval involving a subquery. Get supplier names for suppliers who supply part P2.

```
SELECT  SNAME
FROM            Supplier
WHERE  S#       IN
        ( SELECT        S#
```

```
                FROM        SPJ
                WHERE       P# = 'P2');
```

10) Subquery with multiple levels of nesting. Get supplier names for suppliers who supply at least one red part.

```
SELECT  SNAME
FROM            Supplier
WHERE S#        IN
          ( SELECT          S#
            FROM            SPJ
            WHERE           P# IN (   SELECT  P#
                                      FROM              Part
                                      WHERE     COLOR = 'Red'));
```

11) Subquery with comparison operator other than IN. Get supplier numbers for suppliers who are located in the same city as supplier S1.

```
SELECT  S#
FROM            Supplier
WHERE CITY =    ( SELECT          CITY
                  FROM            Supplier
                 WHERE  S# = 'S1');
```

12) Aggregate function in a subquery. Get supplier numbers for suppliers with status value less than the current maximum status value in the Supplier table.

```
SELECT  S#
FROM            Supplier
WHERE STATUS <    (       SELECT  MAX(STATUS)
                          FROM            Supplier);
```

질의 그룹 # 3

1) Query using EXISTS. Get supplier names for suppliers who supply part P2.

```
SELECT  SNAME
FROM            Supplier
WHERE EXISTS    (       SELECT *
                        FROM    SPJ
                        WHERE S#= Supplier.S#
                        AND     P#='P2'
                );
```

2) Query using NOT EXISTS. Get supplier names for suppliers who do not supply part P2.

```
SELECT SNAME
FROM Supplier
WHERE NOT EXISTS (      SELECT *
                        FROM    SPJ
                        WHERE S#= Supplier.S#
                        AND     P#='P2');
```

3) Query using NOT EXISTS. Get supplier names for suppliers who supply all parts.

```
SELECT SNAME
FROM Supplier
WHERE NOT EXIST
        ( SELECT *
          FROM Part
          WHERE NOT EXISTS (    SELECT *
                                FROM SPJ
                                WHERE S#= Supplier.S#
                                AND P#=Part.P#));
```

4) Query using NOT EXISTS. Get supplier numbers for suppliers who supply at least all those parts supplied by supplier S2.

```
SELECT DISTINCT         S#
FROM    SPJ SPJ_X
WHERE NOT EXISTS (  SELECT *
                    FROM    SPJ SPJ_Y
                    WHERE S# = 'S2'
                    AND     NOT EXISTS  (   SELECT *
                                            FROM    SPJ SPJ_Z
                                            WHERE SPJ_Z.S# = SPJ_X.S#
                                            AND SPJ_Z.P# = SPJ_Y.P#));
```

6) Query involving UNION. Get part numbers for parts that either weigh more than 16 pounds or are supplied by supplier S2 (or both).

```
SELECT          P#
FROM            Part
WHERE  WEIGHT > 16
UNION
SELECT  P#
FROM            SPJ
WHERE  S# = 'S2';
```

질의 그룹 #4

1) Single-record UPDATE. Change the color of part P2 to yellow, increase its weight by 5, and set its city to "unknown" (NULL).

```
UPDATE      Part
SET         COLOR='Yellow', WEIGHT=WEIGHT+5, CITY=NULL
WHERE  P# = 'P2';
```

2) Multiple-record UPDATE. Double the status of all suppliers in London.

```
UPDATE      Supplier
SET         STATUS = 2*STATUS
WHERE  CITY = 'London';
```

3) UPDATE with a subquery. Set the shipment quantity to zero for all suppliers in London.

```
UPDATE      SPJ
SET         QTY = 0
WHERE  'London' =   (  SELECT  CITY
                       FROM          Supplier
                       WHERE    Supplier.S# = SPJ.S#);
```

4) Multiple-table UPDATE. Change the supplier number for supplier S2 to S9.

```
UPDATE      S
SET         S# = 'S9'
WHERE  S# = 'S2';

UPDATE      SPJ
SET         S# = 'S9'
WHERE  S# = 'S2';
```

6) Single-record DELETE. Delete supplier S5.

```
DELETE
FROM        Supplier
WHERE  S# = 'S5';
```

6) Multiple-record DELETE. Delete all shipments with quantity greater than 300.

```
DELETE
FROM        SPJ
WHERE QTY > 300;
```

7) DELETE with a subquery. Delete all shipments.

```
DELETE
FROM        SPJ;
```

8) DELETE with a subquery. Delete all shipments for suppliers in London.

```
DELETE
FROM        SPJ
WHERE 'London' =    (   SELECT CITY
                        FROM Supplier
                        WHERE Supplier.S# = SP.S#);
```

9) Single-record INSERT. Add part P7 (city Athens, weight 24, name and color at present unknown) to table P.

```
INSERT
INTO P          (P#, CITY, WEIGHT)
VALUES          ('P7', 'Athens', 24);
```

10) Single-record INSERT, with field names omitted. Add part P8 (name Sprocket, color Pink, weight 14, city Nice) to table Part.

```
INSERT
INTO            Part
VALUES          ('P8', 'Sprocket', 'Pink', 14, 'Nice');
```

11) Multiple-record INSERT. For each part supplied, get the part number and the total quantity supplied of that part, and save the result in the database.

```
CREATE TABLE TEMP
        (P#             CHAR(6) NOT NULL,
         TOTQTY         INTEGER         NOT NULL,
         PRIMARY KEY (P#));

CREATE UNIQUE INDEX XT ON TEMP (P#);

INSERT
INTO TEMP (P#, TOTQTY)
        SELECT P#, SUM(QTY)
        FROM SPJ
        GROUP BY P#;
```

질의 그룹 5.

1) CREATE TABLE statement for table S:

```
CREATE TABLE Supplier
        ( S#            CHAR(5)         NOT NULL,
          SNAME         CHAR(20)        NOT NULL,
          STATUS        SMALLINT        NULL,
          CITY          CHAR(15)        NOT NULL,
        PRIMARY KEY (S#));
```

SYBASE 에서는 NULL 또는 NOT NULL 이라고 명시하지 않는 경우, default 는 NOT NULL 이 된다(ANSI 에서는 default 가 NULL 이다).

2) DROP TABLE statement for table Supplier:

```
DROP TABLE Supplier;
```

3) CREATE INDEX statement for index XS on S# of tables Supplier:

```
CREATE UNIQUE INDEX XS ON Supplier (S#) CLUSTER;
```

SYBASE 라는 DBMS 에서는 다음과 같이 사용한다. (다른 DBMS 에서도 유사하다.)

```
CREATE UNIQUE CLUSTERED INDEX XS ON Supplier (S#);
```

4) DROP INDEX statement for index XS:

```
DROP INDEX XS;
```

질의 그룹 6.

1) Create a view called REDPARTS, consisting of part numbers, part names, weights, and citys for parts that are red.

```
CREATE VIEW REDPARTS          (P#, PNAME, WT, CITY)
        AS      SELECT  P#, PNAME, WEIGHT, CITY
                FROM            Part
                WHERE   COLOR = 'Red';
```

2) Create a view consisting of part numbers and their total supplied quantity.

```
CREATE VIEW PQ (P#, TOTQTY)
        AS      SELECT  P#, SUM(QTY)
                FROM            SPJ
                GROUP BY        P#;
```

3) Create a view consisting of pairs of city names (x,y), where a supplier located in city x supplies a part stored in city y.

```
CREATE VIEW CITY_PAIRS (SCITY, PCITY)
        AS      SELECT  DISTINCT Supplier.CITY, Part.CITY
                FROM            Supplier, SPJ, Part
                WHERE   Supplier.S# = SPJ.S#
                AND             SPJ.P# = Part.P#;
```

4) Create a view consisting of part numbers and weights for parts that are red and located in London.

```
CREATE VIEW LONDON_REDPARTS
        AS      SELECT  P#, WT
                FROM            REDPARTS
                WHERE   CITY = 'London';
```

5) Create a view consisting of supplier numbers, status, and citys for suppliers that have a status value greater than 15.

```
CREATE VIEW GOOD_SUPPLIERS
        AS      SELECT  S#, STATUS, CITY
                FROM            Supplier
                WHERE   STATUS > 15
                                WITH CHECK OPTION;
```