SQL: 질의, 프로그래밍, 트리거

Chapter 5

예제 인스턴스

R1

뱃사람 번호	배번호	일자
22	101	10/10/96
58	103	11/12/96

- 옆에 있는 예약, 뱃사람 인스턴스들을 가지고 이야기를 진행하기로 한다.
- 예약 릴레이션의 키가 뱃사람번호와 배번호로 구성된다면, 의미가 어떻게 달라질까?

S1

뱃사람 번호	뱃사람 이름	등급	나이
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

뱃사람 번호	뱃사람 이름	등급	나이
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

기본적인 질의

```
SELECT    [DISTINCT] 목표-리스트
FROM      릴레이션-리스트
WHERE     조건식
```

- *릴레이션-리스트* : 릴레이션 이름들의 리스트 (각 이름 뒤에 *범위 변수* 표기 가능).
- *목표-리스트* : *릴레이션-리스트*에 있는 릴레이션들에서 애트리뷰트들을 뽑은 리스트
- *조건식* : 비교조건 (Attr *op* 상수 또는 Attr1 *op* Attr2, 이때 *op*는 <, >, =, ≤, ≥, ≠ 중 하나임) 들을 AND, OR, NOT으로 조합한 식.
- DISTINCT 는 선택적인 키워드로서, 결과에 중복이 없어야 함을 지시. 묵시적으로는 중복값들이 제거되지 *않음*!

개념상의 수행 전략

- SQL 질의의 의미는 다음과 같은 개념상의 수행 전략에 따른 결과와 같다.
  - *릴레이션-리스트*의 테이블들을 카티션 프로덕트한다.
  - 조건식에 어긋난 투플들을 제거한다.
  - 목표-리스트에 없는 필드들을 제거한다.
  - DISTINCT가 있으면, 중복 투플을 제거한다.
- 실제 이대로 하면 비효율적일 경우가 많다! 최적화기는 *동일한 결과*를 낼 수 있는, 더 효율적인 수행 전략을 찾는다.

개념상의 수행 과정(예)

SELECT S.뱃사람이름
FROM 뱃사람 S, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호 AND R.배번호=103

(뱃사람번호)	뱃사람 이름	등급	나이	(뱃사람번호)	배번호	일자
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

범위 변수(*Range Variables*)

- FROM 절에 같은 테이블이 두 번 나올 때 특히 필요함. 앞의 질의는 다음처럼 작성해도 된다.

SELECT S.뱃사람이름
FROM 뱃사람 S, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호 AND 배번호=103

그러나, 항상 범위 변수를 사용하는 것이 좋은 스타일임!

또는 SELECT 뱃사람이름
FROM 뱃사람, 예약
WHERE 뱃사람.뱃사람번호=예약.뱃사람번호
AND 배번호=103

최소한 한 척의 배를 예약한 뱃사람들을 구하라

```
SELECT S.뱃사람번호
FROM 뱃사람 S, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호
```

- DISTINCT를 붙이면 질의가 달라 질까?
- SELECT 절의 *S.뱃사람번호* 를 *S.뱃사람이름* 으로 바꾸면? 여기에 DISTINCT 를 붙이면 달라 질까?

수식과 문자열

```
SELECT S.나이, 나이1=S.나이-5, 2*S.나이 AS 나이2
FROM 뱃사람 S
WHERE S.뱃사람이름 LIKE 'B_%B'
```

- 수식과 문자열 패턴 매칭 용례: *이름이 B로 시작하고 끝나며 최소 3자로 구성되는 뱃사람에 대한 3쌍(나이 필드와 수식결과 2 필드로 구성)을 구하라.*
- AS 와 = 로 결과의 필드에 이름을 줄 수 있다.
- LIKE 를 문자열 매칭에 사용한다. '_'는 임의의 한 문자를 나타내며, '%'는 임의의 문자가 0개 이상 있음을 나타낸다.

적색 배나 녹색 배를 예약한 뱃사람들의 번호를 구하라.

- UNION: 합병가능(*union-compatible*)한 두 투플 집합(그 자체로 SQL 질의의 결과)을 합병할 때 사용.
- 첫번째 형태에서 OR 를 AND 로 바꾸면 결과는?
- EXCEPT 도 있음 : (UNION 을 EXCEPT 로 바꾸면?)

```
SELECT S.뱃사람번호
FROM 뱃사람 S, 배 B, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호
  AND R.배번호=B.배번호
  AND (B.색상='red' OR B.색상='green')
```

```
SELECT S.뱃사람번호
FROM 뱃사람 S, 배 B, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호
  AND R.배번호=B.배번호
  AND B.색상='red'
UNION
SELECT S.뱃사람번호
FROM 뱃사람 S, 배 B, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호
 AND R.배번호=B.배번호
 AND B.색상='green'
```

적색 배와 녹색 배를 예약한 뱃사람들의 번호를 구하라

- INTERSECT: 합병가능한 두 투플 집합간에 교집합할 때 사용.
- SQL/92 표준에는 들어 있지만, 지원 못하는 시스템도 있음.
- UNION 과 INTERSECT 의 대칭성을 잘 비교하기 바람.

```
SELECT S.뱃사람번호
FROM 뱃사람 S, 배 B1, 예약 R1, 배 B2, 예약 R2
WHERE S.뱃사람번호=R1.뱃사람번호
  AND R1.배번호=B1.배번호
  AND S.뱃사람번호=R2.뱃사람번호
  AND R2.배번호=B2.배번호
  AND (B1.색상='red' AND B2.색상='green')
```

```
SELECT S.뱃사람번호  <-------- 키 필드임!
FROM 뱃사람 S, 배 B, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호
  AND R.배번호=B.배번호 AND B.색상='red'
```

```
INTERSECT
SELECT S.뱃사람번호
FROM 뱃사람 S, 배 B, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호
  AND R.배번호=B.배번호 AND B.색상='green'
```


내포된 질의

배번호 103을 예약한 뱃사람들의 이름을 구하라:

```
SELECT S.뱃사람이름
FROM 뱃사람 S
WHERE S.뱃사람번호 IN (SELECT R.뱃사람번호
                       FROM 예약 R
                       WHERE R.배번호=103)
```

- SQL의 강력한 기능의 예: WHERE 절 자체 내에 또 SQL 질의가 들어갈 수 있음! (FROM 절과 HAVING 절에도 가능.)
- 103을 예약하지 않은 뱃사람들을 구하려면, NOT IN 을 사용.
- 내포된 질의의 의미를 이해하려면, *이중 루프* 를 생각할 것: *각 뱃사람 투플에 대하여, 부속질의에 따른 조건이 만족되는지를 검사.*

상호 관련된 내포된 질의

배번호 103을 예약한 뱃사람들의 이름을 구하라:

```
SELECT S.뱃사람이름
FROM 뱃사람 S
WHERE EXISTS (SELECT *
              FROM 예약 R
              WHERE R.배번호=103 AND S.배번호=R.배번호)
```

- EXISTS 는 IN처럼 집합 비교연산자의 일종.
- UNIQUE 를 넣고 *를 *R.배번호* 로 바꾸면, 배번호 103을 최대 한번 예약한 뱃사람들을 구하는 것이 된다.
  (UNIQUE 는 중복 투플 검사용. * 는 모든 필드를 의미함.
  왜 *를 *R.배번호*로 바꾸어야 할까?)
- (일반적으로) 각 뱃사람 투플에 대하여 부속질의를 계산하여야 하는 이유를 잘 보여줌.

기타 집합 비교 연산자

- 앞에서 IN, EXISTS, UNIQUE를 살펴보았음. NOT IN, NOT EXISTS, NOT UNIQUE도 있음.
- 기타 : *op* ANY, *op* ALL, *op* 는 >, <, =, ≤, ≥, ≠
- Horatio라는 이름의 뱃사람 보다 등급이 높은 뱃사람을 구하라:

```
SELECT *
FROM 뱃사람 S
WHERE S.등급 > ANY (SELECT S2.등급
                    FROM 뱃사람 S2
                    WHERE S2.뱃사람이름='Horatio')
```


INTERSECT 질의를 IN을 써서 재작성

적색 배와 녹색 배를 모두 예약한 뱃사람들을 구하라:

```
SELECT S.뱃사람번호
FROM 뱃사람 S, 배 B, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호 AND R.배번호=B.배번호
      AND B.색상='red' AND S.뱃사람번호 IN
      (SELECT S2.뱃사람번호 FROM 뱃사람 S2, 배 B2, 예약 R2
       WHERE S2.뱃사람번호=R2.뱃사람번호 AND R2.배번호=B2.배번호
       AND B2.색상='green')
```

- 마찬가지로, EXCEPT 질의는 NOT IN 을 써서 재작성 가능.
- 여기에서 *번호*가 아니라 *이름*을 구하려면, S.뱃사람번호를 S.뱃사람이름으로 바꾸기만 하면 됨. (INTERSECT 질의에서는 어떨까?)

SQL의 디비전

무든 배를 예약한 뱃사람들을 구하라.

- EXCEPT 없이 어렵게 해보자:

(1)
```
SELECT S.뱃사람이름
FROM 뱃사람 S
WHERE NOT EXISTS
  ((SELECT B.배번호
    FROM 배 B)
    EXCEPT
   (SELECT R.배번호
    FROM 예약 R
    WHERE R.뱃사람번호=S.뱃사람번호))
```

(2)
```
SELECT S.뱃사람이름
FROM 뱃사람 S
WHERE NOT EXISTS (SELECT B.배번호
                  FROM 배 B
                  WHERE NOT EXISTS (SELECT R.배번호
                                    FROM 예약 R
                                    WHERE R.배번호=B.배번호
                                          AND R.뱃사람번호=S.뱃사람번호))
```

…는 뱃사람 S

…하지 않는 배 B는 없다

S가 B를 예약했음을 보여주는 예약투플이 존재

집단(Aggregate) 연산자

- 관계 대수를 확장한 것이 됨.

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

단일 필드

```
SELECT  COUNT (*)
FROM  뱃사람 S

SELECT  AVG (S.나이)
FROM  뱃사람 S
WHERE  S.등급=10
```

```
SELECT  S.뱃사람이름
FROM  뱃사람 S
WHERE  S.등급= (SELECT  MAX(S2.등급)
                FROM  뱃사람 S2)
```

```
SELECT  COUNT (DISTINCT S.등급)
FROM  뱃사람 S
WHERE S.뱃사람이름='Bob'
```

```
SELECT  AVG (DISTINCT S.나이)
FROM  뱃사람 S
WHERE  S.등급=10
```


나이가 가장 많은 뱃사람(들)의 나이와 이름을 구하라

- 첫 질의 형태는 불법임! (GROUP BY를 다룰 때 이유를 설명함)
- 세번째 질의는 두번째 질의와 동등하고 SQL/92 표준에서도 허용하지만, 지원하지 못하는 시스템이 있음.

```
SELECT  S.뱃사람이름, MAX (S.나이)
FROM  뱃사람 S
```

```
SELECT  S.뱃사람이름, S.나이
FROM  뱃사람 S
WHERE  S.나이 =
        (SELECT  MAX (S2.나이)
         FROM  뱃사람 S2)
```

```
SELECT  S.뱃사람이름, S.나이
FROM  뱃사람 S
WHERE  (SELECT  MAX (S2.나이)
        FROM  뱃사람 S2) = S.나이
```


GROUP BY와 HAVING

- 지금까지는 모든(조건에 맞는) 투플들에 대하여 집단 연산자를 적용하였지만, *그룹*별로 적용하고 싶을 때도 있음.
- 예 : *각 등급별로 가장 나이어린 뱃사람을 구하라.*
  - 등급이 몇 가지나 있고 그 값은 어떻게 매기는지를 알 수 없는 경우가 일반적!
  - 등급 값이 1에서 10까지라는 것을 안다고 하더라도 다음과 같은 질의 10개를 만들 수야 없다 :

For i = 1, 2, ... , 10:

```
SELECT  MIN (S.나이)
FROM  뱃사람 S
WHERE  S.등급 = i
```

GROUP BY와 HAVING 을 넣은 질의

```
SELECT      [DISTINCT] 목표-리스트
FROM        릴레이션-리스트
WHERE       조건식
GROUP BY    그룹짓기-리스트
HAVING      그룹-조건식
```

- *목표-리스트* 에는 (i) 애트리뷰트 이름들 과 (ii) 집단연산자 항들(e.g., MIN (*S.나이*))이 들어감.
  - 애트리뷰트-리스트 (i)는 *그룹 짓기-리스트* 의 부분 집합이어야 함. *그룹*별로 결과 투플이 하나씩 생기므로, 이 애트리뷰트들은 그룹별로 단 하나의 값이 되어야 함. (*그룹*이란 *그룹 짓기-리스트*에 있는 모든 애트리뷰트들의 값이 동일한 투플들의 집합)

개념상의 계산법

- *릴레이션-리스트*들을 카티션 프로덕트하고, *조건식*에 맞지 않는 투플들은 버리며, 필요없는 필드들을 삭제하고, 남은 투플들은 *그룹짓기-리스트*의 애트리뷰트 값별로 그룹을 짓는다.
- 그 후에 *그룹-조건식*을 적용해서 일부 그룹들을 없앤다. *그룹-조건식*의 식들은 ***그룹당 하나의 값***이 되어야 한다!
  - 사실상, *그룹-조건식*에는 있는데 집단연산자의 매개변수로는 쓰이지 않은 필드들은 *그룹짓기-리스트*에 다시 나온다.
    (SQL은 이 상황에서는 기본 키 개념을 적용하지 않음!)
- 조건에 맞는 그룹별로 결과 투플이 하나씩 생긴다.

나이≥18 인 사람이 두 명 이상 있는 등급별로, 가장 젊은 뱃사람의 나이를 구하라

```
SELECT  S.등급, MIN (S.나이)
FROM    뱃사람 S
WHERE   S.나이 >= 18
GROUP BY S.등급
HAVING  COUNT (*) > 1
```

뱃사람 번호	뱃사람 이름	등급	나이
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

- S.등급과 S.나이만 SELECT, GROUP BY, HAVING절에 언급되며, 나머지들은 '*불필요*'.
- 결과의 두 번째 필드는 이름이 없음(AS로 이름을 줄 수 있음)

등급	나이
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

등급	
7	35.0

결과 테이블

적색 배 각각에 대해서, 체결된 예약 건수를 구하라

```
SELECT B.배번호, COUNT (*) AS scount
FROM 뱃사람 S, 배 B, 예약 R
WHERE S.뱃사람번호=R.뱃사람번호
      AND R.배번호 =B.배번호 AND B.색상= 'red'
GROUP BY B.배번호
```

- 세 릴레이션을 죠인한 후 그룹을 짓는다.
- *B.색상='red'*를 WHERE 절에서 빼고, 대신 HAVING 절에 넣으면, 결과는?
- '뱃사람'테이블과, S.뱃사람번호에 해당하는 조건을 없앤다면?

(아무 나이든) 두 사람 이상이 있는 등급별로, 나이≥18 인 가장 젊은 뱃사람의 나이를 구하라.

```
SELECT S.등급, MIN (S.나이)
FROM 뱃사람 S
WHERE S.나이 > =18
GROUP BY S.등급
HAVING 1 < (SELECT COUNT (*)
            FROM 뱃사람 S2
            WHERE S.등급=S2.등급)
```

- Having 절에도 부속질의가 나올 수 있음.
- 18세 이상인 사람이 두명 이상인 등급에 대한 질의와 비교!
- Having 절을 다음처럼 바꾼다면?
  - HAVING COUNT(*) >1

뱃사람들의 평균 나이가 가장 낮은 등급을 구하라

- 집단 연산자는 내포될 수 없다! WRONG:

```
SELECT S.등급
FROM 뱃사람 S
WHERE AVG (S.나이) =
  (SELECT MIN (AVG (S2.나이)) FROM 뱃사람 S2 GROUP BY S2.등급)
```

- 맞는 답 (SQL/92):

```
SELECT Temp.등급, Temp.평균나이
FROM (SELECT S.등급, AVG (S.나이) AS 평균나이
      FROM 뱃사람 S
      GROUP BY S.등급) AS Temp
WHERE Temp.평균나이 = (SELECT MIN (Temp.평균나이)
                       FROM Temp)
```

Null(널) 값

- 투플의 필드값 중에는 모르거나(*unknow)* (등급을 아직 매기지 않은 경우) 해당사항 없는 (*inapplicable)* (배우자가 없음) 경우가 있다.
  - SQL 은 이때 *null*이라는 특수한 값을 사용하도록 한다.
- *null* 때문에 여러가지로 복잡해짐.:
  - *Null*값 유무를 확인할 연산자 필요. (IS, IS NOT)
  - *등급*이 *null*일때 *등급>8*이 참/거짓인지? AND, OR ,NOT 이 붙으면어떻게 되는지?
  - 3가 논리(3-valued logic) (참, 거짓, *미상*)필요.
  - 각 구문의 의미 재설정 필요. (e.g.,WHERE 절은 참이 아닌 투플들을 제거함.)
  - 새로운 연산자 (특히, 외부조인(*outer join*))등장 가능/필요.
    - ※ SQL92 : R1 NATURAL LEFT OUTER JOIN R2

삽입 (Embedded)SQL

- SQL 명령을 호스트언어(host language) (e.g., C or COBOL) 프로그램내에서 호출할 수 있다.
  - SQL 문 내에서 호스트 변수 인용 가능 (상태 반환용 특수 변수 ₹함).
  - 알맞은 DB와 연결(접속)하는 문장 필요.
- SQL 릴레이션은 레코드의 (다중) 집합으로서, 구성 레코드의 수를 미리 한정하지 않는다. 이러한 자료구조가 C에는 없다.
  - 이 불일치를 해소하기 위하여 SQL에서는 커서(cursor)라는 메커니즘을 제공함.
    - ※ 투플이 하나 이하임이 확실한 경우에는 CURSOR가 없어도 된다! ex) SELECT INTO 문

커서

- 릴레이션이나, 릴레이션을 결과로 내는 질의에 대하여 커서 하나를 선언할 수 있음.
- 커서를 개방(*open)*하고나면, 투플을 반입(*fetch)*하거나 커서를 이동(*move*)하여 투플을 모두 읽어온다.
  - 투플 반입 순서를 ORDER BY 절로 제어 가능.
    - ORDER BY 절의 필드들은 SELECT 절에도 반드시 출현하여야 함.
  - ORDER BY 절은 커서를 사용할 때에만 허용됨.
- 커서가 현재 가리키고 있는 투플은 수정/삭제 가능.

적색 배를 예약한 뱃사람들의 이름을 알파벳 순서대로 얻어오는 커서

```
EXEC SQL DECLARE sinfo CURSOR FOR
  SELECT S.뱃사람이름
  FROM 뱃사람 S, 배 B, 예약 R
  WHERE S.뱃사람번호=R.뱃사람번호 AND R.배번호=B.배번호
        AND B.색상='red'
  ORDER BY S.뱃사람이름
```

- 위의 ORDER BY 절에서 S.뱃사람 이름을 S.뱃사람번호로 바꾸면 불법! (이유는?)
- SELECT 절에 S.뱃사람번호를 추가하면, 위의 ORDER BY 절의 S.뱃사람 이름을 S.뱃사람번호로 바꾸어도 될까?

C에 SQL을 삽입한 예

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
  SELECT S.sname, S.age FROM Sailors S
  WHERE S.rating > :c_minrating
  ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
  EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
  printf("%s is %d years old\n" c_sname, c_age);
} while (SQLSTATE != "02000");
EXEC SQL CLOSE sinfo;
```


데이타베이스 API: 삽입의 대안

컴파일러를 고치지 않고, 데이타베이스 호출용 라이브러리(API)를 추가함.

- 특수한 표준 인터페이스 : 프로시저(절차)/객체
- PL로부터 SQL문자열을 받고, 결과집합을 PL이 알 수 있는 형태로 반환.
- Microsoft의 *ODBC* 가 Windows환경의 C/C++ 표준이 되고 있음.
- Sun의 *JDBC*는 Java용.
- DBMS에 대해서는 독립적임.
  - 호출을 받고, 이를 특정 DBMS용 코드로 변환해주는 "드라이버(driver)" 존재.
  - DB자체는 네트워크를 통해 분산 가능.

SQL API in Java (JDBC)

```
Connection con = // connect
   DriverManager.getConnection(url, "login", "pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
    // loop through result tuples
    while (rs.next()) {
        String s = rs.getString("name");
        Int n = rs.getFloat("rating");
        System.out.println(s + "     " + n);
    }
} catch(SQLException ex) {
    System.out.println(ex.getMessage ()
        + ex.getSQLState () + ex.getErrorCode ());
}
```


무결성 제약조건 (Review)

- IC란 어떤 릴레이션의 인스턴스가 적법하려면 지켜야 하는 조건의 명세.
  - IC를 위배하는 삽입/삭제/갱신은 불허.
  - 응용의 의미성(e.g., 뱃사람번호는 키이다)을 보장하거나 불일치(e.g., 뱃사람이름은 문자열, 나이< 200 이어야 함)를 방지할 때 사용.
- *IC의 유형*: 도메인 제약조건, 기본 키 제약조건, 외래 키 제약조건, 일반 제약조건.
  - *도메인 제약조건*: 필드 값들은 적절한 타입이 되어야 함. 필수.

일반 제약조건

- 키와 관련없는 일반적인 형태의 IC에 유용.
- 질의를 사용하여 제약조건을 표현할 수 있음.
- 제약조건에 이름을 달 수 있음.

```
CREATE TABLE 뱃사람
    ( 뱃사람번호 INTEGER,
      뱃사람이름 CHAR(10),
      등급 INTEGER,
      나이 REAL,
      PRIMARY KEY (뱃사람번호),
      CHECK ( 등급 >= 1
              AND 등급 <= 10 ))

CREATE TABLE 예약
    ( 뱃사람이름 CHAR(10),
      배번호 INTEGER,
      일자 DATE,
      PRIMARY KEY (배번호,일자),
      CONSTRAINT noInterlakeRes
      CHECK (`Interlake?<>
                ( SELECT B.배이름
                  FROM 배 B
                  WHERE B.배번호=배번호)))
```

여러 테이블에 걸친 제약조건

- 어색하며 틀렸음!
- 뱃사람 투플이 0이면, 배 투플의 수는 어떻게 되어도 무방!

배의 수와 뱃사람의 수의 합은 < 100

```
CREATE TABLE 뱃사람
      ( 뱃사람번호 INTEGER,
        뱃사람이름 CHAR(10),
        등급 INTEGER,
        나이 REAL,
        PRIMARY KEY (뱃사람번호),
        CHECK
        ( (SELECT COUNT (S.뱃사람번호) FROM 뱃사람 S)
        + (SELECT COUNT (B.배번호) FROM 배 B) < 100 )
```

- 어떤 테이블에도 직접 연관되지 않는 ASSERTION이 해법임.

```
CREATE ASSERTION smallClub
CHECK
( (SELECT COUNT (S.뱃사람번호) FROM 뱃사람 S)
+ (SELECT COUNT (B.배번호) FROM 배 B) < 100 )
```


트리거

- 트리거 : DB에 특정한 변경이 가해졌을 때 DBMS가 이에 대응해서 자동적으로 호출하는 일종의 프로시저.
- 세 부분으로 구성:
  - 사건(event) : 트리거를 가동
  - 조건(condition) : 트리거 수행 여부 검사
  - 동작 (action) : 트리거가 수행될 때 일어나는 일

트리거의 예 (SQL:1999)

```
CREATE TRIGGER youngSailorUpdate
   AFTER INSERT ON 뱃사람
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
   INSERT
        INTO YoungSailors(sid, name, age, rating)
        SELECT 뱃사람번호, 뱃사람이름, 나이, 등급
        FROM NewSailors N
        WHERE N.나이 <= 18
```

요약

- SQL은 이전보다 더 자연스러운, 비절차형 질의어
  - 관계 모델 확산에 크게 기여.
- 관계적으로 완전하며, 관계 대수보다 표현력 우수.
- 관계 대수로 표현 가능한 질의도 SQL로는 더 자연스럽게 표현 가능.
- 최적화기가 최적의 수행 계획을 찾도록 한 질의를 여러가지로 표현 가능.
  - 질의의 최적화 방식 및 수행 방식을 이해하고 있는 것이 사용자에게 좋음.

요약(계속)

- 미상인 필드에 NULL값을 적용하기 때문에 여러가지로 복잡해짐.
- 삽입 SQL은 호스트 언어 내에서 수행됨. 커서 메커니즘을 이용하여 한 레코드씩 차례로 반입가능.
- ODBC나 JDBC같은 API로 응용 프로그램과 DBMS간의 독립성 유지 가능.
- SQL 로 풍부한 무결성 제약조건 명세 가능.
- 트리거로 DB의 변경에 대처 가능.