7. SQL

SQL 소개

SQL

- 사용자가 처리를 원하는 데이터가 무엇인지 제시하고 어떻게 처리해야 하는지 언급할 필요가 없어 비 절차적 데이터 언어의 특징을 가짐

역사

- SEQUEL: 1974년 IBM 연구소에서 개발한 연구용 관계 DBMS인 SYSTEM R을 위한 언어
- 많은 회사가 본인들의 RDBMS 제작
- ANSI, ISO 에서 RBDMS 최초로 표준 SQL 을 제정
- SQL1 -> SQL2(SQL-92) -> SQL3(SQL:1999)

DDL

- Data Definition Language (데이터 정의 언어)
- 테이블을 생성, 변경 및 삭제하는 기능을 제공

DML

- Data Manipulation Language (데이터 조작 언어)
- 테이블에 데이터를 삽입, 수정, 삭제 및 검색 하는 기능

DCL

- Data Control Language (데이터 제어 언어)
- 데이터의 무결성, 보안, 권한 제어및 회복하는 기능

SQL 을 이용한 데이터 정의

SQL의 데이터 정의 기능

테이블 정의 기능

- CREATE TABLE (테이블 생성)
- ALTER TABLE (테이블 변경)
- DROP TABLE (테이블 삭제)

테이블의 생성(CREATE TABLE)

- SQL에선 대소문자 구분 X
- 생성시 기본적으로 NULL 허용 -> 허용되지 않을땐 NOT NULL

```
CREATE TABLE 테이블_이름(
        속성_이름 데이터_타입 [NOT NULL] [DEFAULT 기본_값]
        [PRIMARY KEY (속성_리스트)]
        [UNIQUE (속성_리스트)] //여러개 존재 가능
        [FOREIGN KEY (속성_리스트) REFERENCES 테이블_이름(속성_리스트)]
         [ON DELETE 옵션] [ON UPDATE 옵션]
        [CONSTRAINT 이름] [CHECK(조건)]
)
```

표준 SQL 의 속성의 대표적 데이터 타입

데이터 타입	의미
INT 또는 INTEGER	정수
SMALLINT	INT보다 작은 정수
CHAR(n) 또는 CHARACTER(n)	길이가 n인 고정 길이의 문자열
VARCHAR(n) 또는 CHARACTER VARYING(n)	최대 길이가 n인 가변 길이의 문자열
NUMERIC(p, s) 또는 DECIMAL(p, s)	고정 소수점 실수 p는 소수점을 제외한 전체 숫자의 길이, s는 소수점 이하 숫자의 길이
FLOAT(n)	길이가 n인 부동 소수점 실수
REAL	부동 소수점 실수
DATE	연, 월, 일로 표현되는 날짜
TIME	시, 분, 초로 표현되는 시간
DATETIME	날짜와 시간

예제)

```
CREATE TABLE 테이블_이름(
        고객_아이디 VARCHAR(20) NOT NULL,
        적립금 INT DEFAULT 0,
        학점 CHAR(2) DEFAULT 'P',
        학점 CHAR(2) DEFAULT 'p',
        PRIMARY KEY(고객_아이디),
        PRIMARY KEY(주문고객, 주문제품),
```

```
        UNIQUE(고객이름)// 대체키(NULL 가능, 여러개 가능) 외래키는 UNIQUE 제약조건으로 지정된 대체키
    참조 가능
)
```

CREATE TABLE 외래키 삭제 옵션

- **ON DELETE NO ACTION**: 부서 테이블의 튜플을 삭제 못함
- **ON DELETE CASCADE**: 관련 튜플을 함께 삭제
- **ON DELETE SET NULL**: 관련 튜플의 외래키 값을 NULL로 변경
- **ON DELETE SET DEFAULT**: 관련 튜플의 외래키 값을 미리 지정한 기본 값으로 변경

그림 7-4 외래키를 통해 관계를 맺고 있는 2개의 테이블

- **ON UPDATE NO ACTION**: 부서 테이블의 튜플을 변경하지 못함
- **ON UPDATE CASCADE**: 관련 튜플을 함께 변경함
- **ON UPDATE SET NULL**: 관련 튜플의 외래키 값을 NULL로 변경함
- **ON UPDATE SET DEFAULT**: 관련 튜플의 외래키 값을 미리 지정한 기본 값으로 변경함
- **FOREIGN KEY(소속부서) REFERENCES 부서(부서번호) ON DELETE CASCADE ON UPDATE CASCADE**: 관련 튜플을 함께 삭제/수정함
- **FOREIGN KEY(소속부서) REFERENCES 부서(부서번호)**: 부서 테이블의 ON DELETE NO ACTION과 ON UPDATE NO ACTION이 자동으로 선택됨

그림 7-4 외래키를 통해 관계를 맺고 있는 2개의 테이블

CREATE TABLE 데이터 무결성 제약조건의 정의

- **CHECK(재고량 >= 0 AND 재고량 <= 10000)**: 재고량은 0개 이상, 10000개 이하로 유지되어야 함
- **CONSTRAINT CHK_CPY CHECK(제조업체 = '한빛제과')**: 데이터 무결성 제약조건에 CHK_CPY라는 이름 부여 (다른 테이블에는 없어야 하며, 유일하게 부여되야함)

CREATE TABLE 테이블 생성의 예

CREATE TABLE 데이터 무결성 제약조건의 정의

예제 1)

- 고객 테이블은 고객 아이디, 고객이름, 나이, 등급, 직업, 적립금 속성으로 구성되야 하고, 고객 아이디가 기본키다. 고객이름과 등급 속성은 값을 반드시 입력해야하고, 적립금 속성은 입력하지 않으면 0이 기본으로 설정되도록 고객 테이블을 작성하자

```
CREATE TABLE 고객(
        고객_아이디 VARCHAR(20) NOT NULL,
        고객_이름 VARCHAR(10) NOT NULL,
        나이 INT,
        등급 VARCHAR(10) NOT NULL,
        직업 VARCHAR(20),
        적립금 INT DEFUALT 0,
        PRIMARY KEY(고객_아이디)
)
```

예제 2)

- 제품 테이블은 제품번호, 제품명, 재고량, 단가, 제조업체 속성으로 구성되고, 제품번호 속성이 기본키이다. 재고량이 항상 0개 이상, 10000개 이하를 유지하도록 제품 테이블을 작성하자

```
CREATE TABLE 제품(
        제품_번호 CHAR(3) NOT NULL,
        제품명 VARCHAR(20),
        재고량 INT,
        단가 INT,
        제조업체 VARCHAR(20),
        PRIMARY KEY(제품_번호),
        CHECK (재고량 >= 0 AND 재고량 <= 10000)
)
```

예제 3)

- 주문 테이블은 주문번호, 주문고객, 주문제품, 수량, 배송지, 주문일자 속성으로 구분되고, 주문번호 속성이 기본키다. 주문고객 속성이 고객 테이블의 고객아이디 속성을 참조하는 외래키이고, 주문제품 속성이 제품 테이블의 제품번호 속성을 참조하는 외래키가 되도록 주문 테이블을 작성해보자

```
CREATE TABLE 주문(
        주문_번호 CHAR(3) NOT NULL,
        주문_고객 VARCHAR(20),
        주문_제품 VARCHAR(3),
        수량 INT,
```

```
        배송지 VARCHAR(30),
        주문_일자 DATE,   //MS SQL에선 DATETIME
        PRIMARY KEY(주문_번호)
        FOREGIN KEY(주문_고객) REFERENCE 고객(고객아이디),
        FOREGIN KEY(주문_제품) REFERENCE 제품(제품_번호)
)
```

예제 4)

- 배송업체 테이블은 업체번호, 업체명, 주소, 전화번호 속성으로 구성되고 업체번호 속성이 기본키다. 배송업체 테이블을 작성해보자

```
CREATE TABLE 배송업체(
        업체_번호 CHAR(3) NOT NULL,
        업체명 VARCHAR(20),
        주소 VARCHAR(100),
        전화번호 VARCHAR(20),
        PRIMARY KEY(업체_번호)
)
```

테이블의 변경(ALTER TABLE)

속성 추가 기본 형식

```
ALTER TABLE 테이블_이름
        ADD 속성_이름 데이터_타입 [NOT NULL] [DEFAULT 기본값];
```

예제)

- 고객 테이블에 가입날짜 속성 추가

```
ALTER TABLE 고객
        ADD 가입_날짜 DATE;
```

속성 삭제 기본 형식

```
ALTER TABLE 테이블_이름
        DROP COLUMN 속성_이름;
```

예제)

- 위에서 추가한 고객 테이블의 가입날짜 속성 삭제

```
ALTER TABLE 고객
        DROP COLUMN 가입_날짜;
```

새로운 제약조건 추가 기본 형식

```
ALTER TABLE 테이블_이름
        ADD CONSTRAINT 제약조건_이름 제약조건_내용;
```

예제)

- 고객 테이블에 20세 이상만 가입이 가능하다는 데이터 무결성 조건 추가

```
ALTER TABLE 고객
        ADD CONSTRAINT CHK_AGE CHECK(나이 >= 20);
```

제약조건 삭제 기본 형식

```
ALTER TABLE 테이블_이름
        DROP CONSTRAINT 제약조건_이름;
```

예제)

- 위에서 추가한 고객 테이블에 20세 이상의 고객만 가입할 수 있다는 데이터 무결성 제약조건을 삭제

```
ALTER TABLE 고객
        DROP CONSTRAINT CHK_AGE;
```

테이블 삭제 (DROP TABLE)

테이블 삭제 기본 형식

```
DROP TABLE 테이블_이름;
```

예제)

- 배송업체 테이블 삭제

```
DROP TABLE 배송업체
```

SQL 을 이용한 데이터 조작

데이터 검색

기본

```
SELECT [ALL | DISTINCT] 속성_리스트 FROM 테이블_리스트;
```

예제 1)

- 고객 테이블에서 고객아이디, 고객이름, 등급 속성을 검색

```
SELECT 고객아이디, 고객이름, 등급 FROM 고객;
```

예제 2)

- 고객 테이블에 존재하는 모든 속성 검색

```
SELECT * FROM 고객;
```

예제 3)

- 제품 테이블에서 제조업체 검색

```
SELECT 제조업체 FROM 제품;
```

예제 4)

- 제품 테이블에서 제조업체 속성을 중복없이 검색

```
SELECT DISTINCT 제조업체 FROM 제품;
```

AS

- 컬럼 Alies -> 출력시 출력이름 다르게 가능
  예제)
- 제품 테이블에서 제품명과 단가를 검색하되, 단가는 가격이라는 이름으로 출력

```
SELECT 제품명, 단가 AS 가격 FROM 제품;
```

산술식

- 속성값에 대해 사칙연산을 수행가능
  예제)

- 제품 테이블에서 제품명과 단가를 검색하되, 단가에 500원을 더해 '조정단가' 라는 이름으로 출력

```
SELECT 제품명, 단가 + 500 AS "조정 단가" FROM 제품;
```

조건식

표 7-2 비교 연산자

연산자	의미
=	같다.
< >	다르다.
<	작다.
>	크다.
<=	작거나 같다.
>=	크거나 같다.

표 7-3 논리 연산자

연산자	의미
AND	모든 조건을 만족해야 검색한다.
OR	여러 조건 중 한 가지만 만족해도 검색한다.
NOT	조건을 만족하지 않는 것만 검색한다.

```
SELECT [ALL|DISTINCT] 속성_리스트 FROM 테이블_리스트 [WHERE 조건];
```

예제 1)

- 제품 테이블에서 한빛제과가 제조한 제품의 제품명, 재고량, 단가를 검색

```
SELECT 제품명, 재고량, 단가 FROM 제품 WHERE 제조업체 = 한빛제과;
```

예제 2)

- 주문 테이블에서 apple 고객이 15개 이상 주문한 주문제품, 수량, 주문일자를 검색

```
SELECT 주문제품, 수량, 주문일자 FROM 주문 WHERE 주문고객 = 'apple' AND 수량 >= 15;
```

예제 3)

- 주문 테이블에서 apple 고객이 주문했거나 15개 이상 주문된 제품의 주문제품, 수량, 주문일자, 주문고객 검색

```
SELECT 주문제품, 수량, 주문일자, 주문고객 FROM 주문 WHERE 주문고객 = 'apple' OR 수량
>= 15;
```

예제 4)

- 제품 테이블에서 단가가 2000원 이상이면서, 3000원 이하인 제품의 제품명, 단가, 제조업체를 검색

```
SELECT 제품명, 단가, 제조업체 FROM 제품 WHERE 단가 >= 2000 AND 단가 <= 3000;
```

LIKE 이용

표 7-4 LIKE 키워드와 함께 사용할 수 있는 기호

기호	설명
%	0개 이상의 문자 (문자의 내용과 개수는 상관 없음)
_	1개의 문자 (문자의 내용은 상관 없음)

표 7-5 LIKE 키워드의 사용 예

사용 예	설명
LIKE '데이터%'	데이터로 시작하는 문자열 (데이터로 시작하기만 하면 길이는 상관 없음)
LIKE '%데이터'	데이터로 끝나는 문자열 (데이터로 끝나기만 하면 길이는 상관 없음)
LIKE '%데이터%'	데이터가 포함된 문자열
LIKE '데이터 _ _ _'	데이터로 시작하는 6자 길이의 문자열
LIKE '_ _ 한%'	세 번째 글자가 '한'인 문자열

예제 1)

- 고객 테이블에서 성이 김씨인 고객의 고객이름, 나이, 등급, 적립금을 검색

```
SELECT 나이, 고객이름, 등급, 적립금 FROM 고객 WHERE 고객이름 LIKE '김%';
```

예제 2)

- 고객 테이블에서 고객아이디가 5자인 고객의 고객아이디, 고객이름, 등급 검색

```
SELECT 고객아이디, 고객이름, 등급 FROM 고객 WHERE 고객아이디 LIKE '_____';
```

NULL 검색

유의할 점

- 제품 테이블의 재고량이 NULL 속성을 가지면 아래 연산 결과는 모두 FALSE
  - 재고량 > 10
  - 재고량 = 10

- 재고량 <> 10

예제 1)

- 고객 테이블에서 나이가 아직 입력되지 않은 고객의 고객이름 검색

```
SELECT 고객이름 FROM 고객 WHERE 나이 IS NULL;
```

예제 2)

- 고객 테이블에서 나이가 입력된 고객의 고객이름 검색

```
SELECT 고객이름 FROM 고객 WHERE 나이 IS NOT NULL;
```

정렬 검색

기본

```
SELECT [ALL|DISTINCT] 속성_리스트 FROM 테이블_리스트 [WHERE 조건] [ORDER BY 속성_리스트 [ASC|DESC]];
```

- ASC = 오름차순, DESC = 내림차순

예제 1)

- 고객 테이블에서 고객이름, 등급, 나이를 검색하되, 나이를 기준으로 내림차순 정렬

```
SELECT 고객이름, 등급, 나이 FROM 고객 ORDER BY 나이 DESC;
```

예제 2)

- 주문 테이블에서 수량이 10개 이상인 주문의 주문고객, 주문제품, 수량, 주문일자를 검색해보자. 단 주문제품을 기준으로 오름차순 정렬, 동일제품은 수량을 기준으로 내림차순

```
SELECT 주문고객, 주문제품, 수량, 주문일자 FROM 주문 WHERE 수량 >= 10 ORDER BY 주문제품 ASC, 수량 DESC;
```

집계 함수

표 7-6 집계 함수

함수	의미	사용 가능한 속성의 타입
COUNT	속성 값의 개수	모든 데이터
MAX	속성 값의 최댓값	
MIN	속성 값의 최솟값	
SUM	속성 값의 합계	숫자 데이터
AVG	속성 값의 평균	

유의할 점

- 집계함수는 NULL 속성 값은 제외하고 계산
- 집계함수는 WHERE 절에는 사용할 수 없음
- 집계함수는 SELECT 절이나 HAVING 절에서 사용할 수 없음

예제 1)

- 제품 테이블에서 모든 제품의 단가 평균 검색

```
SELECT AVG(단가) FROM 제품
```

예제 2)

- 한빛제과에서 제조한 제품의 재고량 합계를 제품 테이블에서 검색
  - 쿼리시 속성 별칭을 두는 것이 일반적

```
SELECT SUM(재고량) AS "제고량 합계" FROM 제품 WHERE 제조업체 = '한빛제과';
```

예제 3)

- 고객 테이블에 고객이 몇 명 등록되어 있는지 검색

```
// 고객아이디
SELECT COUNT(고객아이디) AS 고객수 FROM 고객;

// 나이
SELECT COUNT(나이) AS 고객수 FROM 고객;

// *
SELECT COUNT(*) AS 고객수 FROM 고객;
```

DISTINCT

- **예제)**
  - 제품 테이블에서 제조업체의 수를 검색

```
SELECT COUNT(DISTINCT 제조업체) AS "제조업체 수" FROM 제품;
```

그룹별 검색

- SELECT 절, GROUP BY 절 속성 개수를 동일하게 해야함 (집계함수는 제외)
- **그룹 특징: 문제에 ~별 같은 단어 들어감**

GROUP BY

```
SELECT [ALL|DISTINCT] 속성_리스트 FROM 테이블_리스트 [WHERE 조건] [GROUP BY 속성_리스트 [HAVING 조건]] [ORDER BY 속성_리스트 [ASC|DESC]];
```

- GROUP BY 속성을 제외하고 **집계 함수** 사용 가능
  - 예제) 제품테이블 에서 제조업체별로 제조한 제품의 개수와 제품중 가장 비싼 단가를 검색하되, 제품의 개수는 제품수 라는 이름으로 출력하고 가장 비싼 단가는 최고가라는 이름으로 출력

```
SELECT 제조업체, COUNT(*) AS 제품수, MAX(단가) AS 최고가 FROM 제품 GROUP BY 제조업체;
```

- GROUP BY 속성 제외하고 쿼리도 가능
  - 예제) 주문테이블 에서 주문제품별 수량의 합계

```
SELECT SUM(수량) AS 총주문수량 FROM 주문 GROUP BY 주문제품;
```

HAVING

- 제품 테이블에서 제품을 3개 이상 제조한 제조업체별로 제품의 개수와 제품중 가장 비싼 단가를 검색

```
SELECT 제조업체, MAX(단가) AS 최고가 FROM 제품 GROUP BY 제조업체 HAVING COUNT(*) >= 3;
```

- 고객 테이블에서 적립금 평균이 1000원 이상인 등급에 대해 등급별 고객수와 적립금 평균을 검색

```
SELECT 등급, COUNT(*) AS 고객수, AVG(적립금) AS 평균적립급
FROM 고객
GROUP BY 등급 HAVING AVG(적립금) >= 1000;
```

뷰

개념

원본 테이블을 기반으로 만들어진 가상 테이블

- 뷰의 생성, 삭제도 DDL (데이터 정의 언어)

생성

**SELECT 문과 함께 CREATE VIEW 로서 생성

WITH CHECK OPTION // 제약조건

- 일반적으로 ORDER BY 사용 불가능

```
CREATE VIEW 뷰_이름[(속성_리스트)]
AS SELECT 문
[WITH CHECK OPTION];
```

예제 1)

- 고객 테이블에서 등급이 VIP인 고객의 고객아이디, 고객이름, 나이, 등급으로 구성된 뷰를 우수고객이라는 이름으로 생성해보자. 그런 다음 우수고객 뷰의 모든 내용을 검색해보자

```
CREATE VIEW 우수고객(고객아이디, 고객이름, 나이, 등급)
AS SELECT 고객아이디, 고객이름, 나이, 등급
        FROM 고객
        WHERE 등급='VIP'
WITH CHECK OPTION;

SELECT * FROM 우수고객;
```

- 우수고객의 속성 == 고객 테이블의 속성 -> 속성 테이블 생략 가능
- VIP가 아닌 등급을 삽입 or 정의 조건 위반하는 수정 및 삭제는 거부

```
CREATE VIEW 우수고객
AS SELECT 고객아이디, 고객이름, 나이, 등급
        FROM 고객
        WHERE 등급='VIP'
WITH CHECK OPTION;
```

예제 2)

집계함수를 사용해 반드시 속성의 이름(제품수)를 제시해야함

- 제품 테이블에서 제조업체별 제품수로 구성된 뷰를 업체별제품수 라는 이름으로 생성해보자. 그런 다음 업체별제품수 뷰의 모든 내용을 검색

```
CREATE VIEW 업체별제품수
AS SELECT 제조업체, COUNT(*)
        FROM 제품
        GROUP BY 제조업체
WITH CHECK OPTION;

SELECT * FROM 업체별제품수;
```

예제 3)

뷰 테이블에서 일반 테이블처럼 쿼리가 가능
뷰를 통해 삽입·수정·삭제가 허용될 수 있음

- 우수고객 뷰에서 나이가 20세 이상인 고객에 대한 모든 내용 검색

```
SELECT FROM 우수고객 WHERE 나이>=20;
```

활용

변경이 가능한 뷰와 불가능한 뷰

```
CREATE VIEW 제품1
AS SELECT 제품번호, 재고량, 제조업체
    FROM 제품
WITH CHECK OPTION;

SELECT * FROM 제품1;
```

	제품번호	재고량	제조업체
1	p01	5000	대한식품
2	p02	2500	민국푸드
3	p03	3600	한빛제과
4	p04	1250	한빛제과
5	p05	2200	대한식품
6	p06	1000	민국푸드
7	p07	1650	한빛제과

```
CREATE VIEW 제품2
AS SELECT 제품명, 재고량, 제조업체
    FROM 제품
WITH CHECK OPTION;

SELECT * FROM 제품2;
```

	제품명	재고량	제조업체
1	그냥만두	5000	대한식품
2	매운쫄면	2500	민국푸드
3	쿵떡파이	3600	한빛제과
4	맛난초콜릿	1250	한빛제과
5	얼큰라면	2200	대한식품
6	통통우동	1000	민국푸드
7	달콤비스킷	1650	한빛제과

- 뷰 제품1에 신규 데이터 삽입
  - 제품번호가 p08, 재고량이 1000, 제조업체가 신선식품인 새로운 제품의 정보를 제품1 뷰에 삽입후 제품1 뷰에 있는 모든 내용 검색

```
INSERT INTO 제품1 VALUES ('p08','1000','신선식품');
SELECT * FROM 제품1;
```

- 원본 테이블을 검색해도 신규 데이터는 삽입되어 있음 (제품명, 단가는 NULL)
- 뷰 제품2에 신규 데이터 삽입 <- 에러

```
INSERT INTO 제품2 VALUES ('시원냉면', '1000', '신선식품')
```

- 뷰 제품2는 기본키(PRIMARY KEY)인 제품번호가 없기 때문에 기본키가 NULL로 제품 데이터 삽입이 안됨
- 뷰 업체별제품수에도 삽입시 동일한 에러 <- 업체별제품수에는 기본키가 없음

뷰 테이블로의 데이터 조작은 기본 테이블의 내용을 바꾸기에 주의해야함

변경이 불가능한 뷰의 특징

- 기본키를 구성하지 않는 뷰는 원본 테이블 변경 불가
- 원본 테이블에 없는 집계함수로 계산된 내용을 포함하는 뷰는 원본 테이블 변경 불가
- DISTINCT/GROUP BY 키워드를 포함하여 정의한 뷰는 변경 불가
- 여러개의 테이블을 조인하여 정의한 뷰는 대부분 불가능

뷰 테이블의 대표적 장점

- 질의문 쉽게 작성 가능
- 데이터 보안 유지 가능 // 뷰를 통해 데이터 접근이 가능하도록 권한 설정
- 데이터를 좀더 편리하게 관리 가능 // 기본 키를 갖고 있지 않은 뷰는 원본 테이블에 영향 없음

삭제

DROP VIEW 테이블

- CASCADE CONSTRAINTS 옵션을 쓰면 제약조건과 함께 삭제가 가능

```
DROP VIEW 뷰_이름;
```

- 우수고객 뷰 삭제법

```
DROP VIEW 우수고객;
```

삽입 SQL

개념과 특징

개념

- 응용 프로그래밍 안에 삽입하여 사용하는 SQL문

특징

- 일반 명령문이 위치할 수 있는 곳이라면 어디든지 삽입 가능
- 삽입 SQL문 앞에 EXEC SQL을 붙임
- 프로그램에 선언된 일반 변수 (앞에 : 붙임)를 삽입 SQL문에서 사용 가능
- 삽입 SQL문의 수행 결과로 여러개의 행을 반환하려면 커서가 필요

커서가 불필요한 경우

CREATE TABLE, INSERT, DELETE, UPDATE문

- 결과로 하나의 행을 반환하기에 커서 필요 X

```
int main() {
        EXEC SQL BEGIN DECLARE SECTION;
                CHAR p_no[4], p_name[21];
                int price;
        EXEC SQL END DECLARE SECTION;

        printf("제품번호 입력: ");
        scanf("%s", p_no);

        EXEC SQL SELECT 제품명, 단가 INTO :p_name, :price
                FROM 제품
                WHERE 제품번호 = :p_no;

        printf("\n 제품명 = %s", p_name);
        printf("\n 단가 = %d", price);

        return 0;
}
```

커서가 필요한 경우

커서의 이름의 선언 및 오픈

- 결과로 여러 행을 반환하기에 커서 필요함
- EXEC SQL DECLARE 커서_이름 CURSOR FOR SELECT 문;
- 예)

```
EXEC SQL DECLARE product_cursor CURSOR FOR SELECT 제품명, 단가 FROM 제품;
//제품 테이블에서 제품명과 단가를 모두 검색하는 SELECT 문을 위한 커서를 product_cursor 라는 이름으로 선언
```

- SELECT 문을 실행하기 위한 별도의 명령어
- EXEC SQL OPEN 커서_이름;
- 예)

```
EXEC SQL OPEN product_cursor
        //product_cursor라는 이름의 커서와 연결된 SELECT문을 실행
```

커서의 이동 및 종료

- 커서가 가리키는 행으로부터 속성값을 가지고 와서 변수에 저장
- 일반적으로는 반복문과 사용
- EXEC SQL FETCH 커서*이름 INTO 변수*리스트;
- 예)

```
EXEC SQL FETCH product_cursor INTO :p_name, :price;
//product_cursor를 이용해 결과 테이블의 다음 행에 접근하여 제품명 속성의 값을 p_name 변수
에 저장하고 단가 속성의 값을 price변수에 저장
```

- 커서 미사용시 종료
- EXEC SQL CLOSE 커서_이름;
- 예)

```
EXEC SQL CLOSE product_name;
```