## 1. SQL의 소개

\* SQL(Structered Query Language)

: 관계 DB를 위한 표준 질의어와 같은 역할(데이터 조작 기능+정의 및 제어 기능 제공)

- cf) 표준 질의어

: 데이터 언어 중 검색 위주의 기능을 하는 비절차적 데이터 조작어

\* 비절차적 데이터 언어

: 사용자가 처리를 원하는 데이터가 무엇인지만 제시하고 데이터를 어떻게 처리해야 하는지를 언급할 필요 없음

## 2. SQL을 이용한 데이터 정의

\* 이후의 코드 블록 코드는 모두 책에 명시된 고객, 제품, 주문 테이블을 사용한 예제이다

### 2.2 테이블의 생성

: 'CREATE TABLE' 키워드

```

CREATE TABLE CUSTOMER

```

\* 기본 형식(FUNCTION)

```

CREAT TABLE 테이블\_이름 (

속성\_이름 데이터\_타입 [NOT NULL] [DEFAULT 기본\_ 값] ## 테이블을 구성하는 속성들의 이름과 데이터 타입 및 제약 사항에 대한 정의

[PRIMARY KEY(속성\_리스트)] ## 기본키: 테이블에 하나만 존재

[UNIQUE (속성 리스트)] ## 대체키: 여러 개 존재 가능

[FOREIGN KET (속성\_리스트) REFERENCES 테이블\_이름(속성\_리스트)] ## 외래키: 여러 개 존재 가능

[ON DELETE 옵션] [ON UPDATE 옵션]

[CONSTRAINT 이름] [CHECK(조건)] ## 데이터 무결성을 위한 제약조건

); ## 모든 SQL문은 ';'으로 문장의 끝을 표시

```

\* 키워드는 대소문자를 구분하지 않음

```

CREATE TABLE; = create table;

```

#### 2.2.1 속성의 정의

(데이터 타입 정리하기\_구글 자료)

\* NOT NULL 키워드

- 용도: 속성에서 널 값을 허용하지 않는 경우에 사용

- 기본: (기본 키 속성) NOT NULL

- 기본키를 구성하는 속성은 반드시 NOT NULL을 만족해야 함(무결성 제약 조건)

```

ID VARCHAR(20) NOT NULL; ## 길이가 최대 20, 가변 길이, 문자열, 널 값 금지

```

\* DEFAULT 키워드

- 용도: 속성의 기본 값을 지정

- 기본: NULL

- 기본 값이 문자열/날짜 데이터인 경우 ‘’로 묶기

```

POINT INT DEFAULT 0;

GRADE VARCHAR(10) DEFAULT ‘SILVER’;

```

#### 2.2.2 키의 정의

\* 기본키(PRIMARY KEY)

\* 대체키(UNIQUE KEY)

: 기본키 + NULL 값을 가질 수 있음

- ex) UNIQUE [CUSTOMER\_NAME]

\* 외래키(FOREIGN KEY)

: 어떤 테이블의 무슨 속성을 참조하는지를 REFERENCES 키워드에서 제시해야 함

- 참조되는 테이블에서 튜플 변경/삭제에 제약을 줌

+ ON DELETE NO ACTION(튜플 삭제 금지)

+ 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 기본 값으로 변경)

+ cf) 변경 제약 시에는 DELETE 대신 UPDATE를 사용

#### 2.2.3 데이터 무결성 제약조건의 정의

\* ‘CHECK’ 키워드

: 특정 속성에 대한 제약조건을 지정

```

CHECK(STOCK >= 0 AND STOCK<=10000); -- 재고량이 0개 이상, 10000개 이하로 유지해야 함

CONSTRAINT CHK\_CPY CHECK(COMPANY == ‘HANBIT’); -- CHK\_CPY라는 무결성 제약 조건을 생성하여 제조업체는 ‘한빛제과’만 허용함

```

### 2.3 테이블의 변경

\* ‘ALTER TABLE’ 키워드

- 속성 추가/삭제, 제약조건 추가/삭제

- 새로운 속성의 추가

```

ALTER TABLE [테이블\_이름]

ADD [속성\_이름] [데이터\_타입] [NOT NULL] [DEFAULT 기본값]; -- 기본 형식

ALTER TABLE CUSTOMER ADD SIGN\_IN\_DATE; – 고객 테이블에서 가입날짜 속성을 추가

```

- 기존 속성의 삭제

```

ALTER TABLE [테이블\_이름] DROP [속성]이름] CASCADE[/RESTRICT]; -- 기본 형식

/\*

CASCADE: 관련된 제약조건이나 참조하는 다른 속성을 함께 삭제

RESTRICT: 관련된 제약조건이나 참조하는 다른 속성이 존재하면 삭제를 수행하지 않음

\*/

ALTER TABLE CUSTOMER GRADE CASCADE; -- 고객 테이블에서 등급 속성과 이와 관련한 제약조건과 다른 속성을 모두 삭제함

```

- 새로운 제약조건의 추가

```

ALTER TABLE [테이블\_이름] DROP CONSTRAINT [제약조건\_이름]; -- 기본 형식

ALTER TABLE CUSTOMER ADD CONSTRAINT CHK\_AGE CHECK(AGE >= 20); -- 고객 테이블에서 20세 이상의 고객만 가입할 수 있는 데이터 무결성 제약조건을 추가

```

- 기존 제약조건의 삭제

```

ALTER TABLE [테이블\_이름] DROP CONSTRAINT [제약조건\_이름];

ALTER TABLE CUSTOMER DROP CONSTRAINT CHK\_AGE; -- 고객 테이블에서 연령 제한 제약조건을 삭제

### 2.4 테이블의 제거

```

DROP TABLE [테이블\_이름] CASCADE[/RESTRICT]; -- 기본 형식

DROP TABLE CUSTOMER RESTRICT; -- 고객 테이블을 삭제하고 이를 참조하는 다른 테이블은 삭제를 수행하지 않음

## 3. SQL을 이용한 데이터 조작

#### 3.2.1 기본 검색: ‘SELECT’ 키워드

```

SELECT [속성\_리스트]

FROM [테이블\_리스트]; -- 기본 형식

SELECT \* FROM CUSTOMER – 고객 테이블의 모든 속성을 검색

SELECT ID, AGE, GRADE FROM CUSTOMER – 고객 테이블에서 ID, 나이, 등급 속성을 검색

\* ‘SELECT DISTINCT’ 키워드

- 튜플의 중복을 제거하고 한 번씩만 출력

```

SELECT DISTINCT COMPANY

FROM PRODUCT; -- 제품 테이블에서 제조업체의 속성을 중복 없이 검색

```

\* ‘SELECT AS’ 키워드

- 속성을 다른 이름으로 바꾸어 출력

- 원래 테이블의 속성 이름은 변경되지 않음

```

SELECT PRODUCT\_NAME, PRICE AS PRODUCT\_PRICE

FROM PRODUCT; -- 제품 테이블에서 제품명과 단가를 검색하되, 단가를 PRODUCT\_PRICE라는 이름으로 출력

#### 3.2.2 산술식을 이용한 검색

```

SELECT NUM, PRICE + 500 AS NEW PRICE

FROM PRODUCT; -- 제품 테이블에서 제품번호와 단가 속성을 검색하되, 단가에 500원을 더해 새 가격을 출력

#### 3.2.3 조건 검색

\* ‘WHERE’ 키워드

- 숫자, 문자, 날짜 값 모두 비교 가능

- 숫자 값은 그대로 값을 입력, 문자/날짜 값은 ‘’로 묶어야 함

```

SELECT [속성\_리스트]

FROM [테이블\_리스트]

WHERE [조건]; -- 기본형식

SELECT PRODUCT, CNT, ORDER\_DATE

FROM ORDER\_LIST

WHERE CUSTOMER = 'APPLE' OR CNT>= 15; -- 주문 테이블에서 주문 고객이 ‘APPLE’이거나 수량이 15개 이상인 경우에 주문제품, 수량, 주문일자를 검색

```

#### 3.2.4 ‘LIKE’를 이용한 검색

\* 부분적으로 일치하는 데이터를 검색

- 단, 문자열을 이용하는 조건에만 사용 가능

```

SELECT CUSTOMER\_NAME, AGE, GRADE

FROM CUSTOMER

WHERE CUSTOMER\_NAME LIKE 'KIM$’; -- 고객 테이블에서 성이 김씨인 고객의 이름, 나이, 등급을 검색

SELECT ID, CUSTOMER\_NAME, GRADE

FROM CUSTOMER

WHERE ID LIKE '\_\_\_\_\_'; -- 고객 테이블에서 아이디가 5자인 고객의 아이디, 이름, 등급을 검색

```

#### 3.2.5 NULL을 이용한 검색

\* ‘IS NULL’/’IS NOT NULL’ 키워드

: 특정 속성의 값이 NULL인지 비교

- NULL 값은 다른 값과 크기를 비교할 수 없다(결과: FALSE)

```

SELECT CUSTOMER\_NAME

FROM CUSTOMER

WHERE AGE IS NOT NULL; -- 고객 테이블에서 나이가 입력된 고객의 이름을 검색

```

#### 3.2.6 정렬 검색

\* 출력 순서 규칙

- 기준이 되는 속성을 지정

- ASC: 오름차순

- DESC: 내림차순

```

SELECT [속성\_리스트]

FROM [테이블\_리스트]

WHERE [조건]

ORDER BY [속성\_리스트] [ASC/DESC];

```

```

SELECT CUSTOMER\_NAME, GRADE, AGE

FROM CUSTOMER

ORDER BY AGE DESC; -- 고객 테이블에서 이름, 등급, 나이를 검색하되, 나이를 기준으로 내림차순 정렬

SELECT CUSTOMER, PRODUCT, CNT, ORDER\_DATE

FROM ORDER\_LIST

ORDER BY PRODUCT ASC, CNT DESC; -- 주문 테이블에서 고객, 제품, 수량, 주문일자를 검색하되, 제품명을 오름차순으로 정렬하고, 제품명이 같은 경우에는 추가로 수량을 내림차순 정렬

```

#### 3.2.7 집계 함수를 이용한 검색

\* 집계 함수(aggregate function)(= 열 함수= column function)

- NULL인 속성 값은 제외하고 계산

- SELECT 절, HAVIING 절에서만 사용 가능(WHERE 절에서 사용 불가)

(집계 함수 5개 목록)

```

SELECT SUM(PRICE) AS SUM

, AVG(PRICE) AS AVG

, MIN(PRICE) AS MIN

, MAX(PRICE) AS MAX

, COUNT(PRICE) AS CNT

FROM PRODUCT; -- 제품 테이블에서 단가의 합계, 평균, 최소값, 최대값, 개수를 검색(단, 검색한속성의 이름이 지정되지 않았으므로 SUM, AVG, MIN, MAX, CNT로 지정)

SELECT COUNT(ID), COUNT(AGE)

FROM CUSTOMER; -- 나이 속성에 NULL 값이 한 개 있으므로 ID 속성의 개수는 7, 나이의 개수는 6으로 출력됨

```

#### 3.2.8 그룹별 검색

\* ‘GROUP BY’ 키워드

```

SELECT [속성\_리스트]

FROM [테이블\_리스트]

WHERE [조건]

GROUP BY [속성\_리스트] [HAVING 조건] – HAVING: 그룹에 대한 조건 설정

ORDER BY [속성 리스트/ASC/DESC];

```

```

SELECT SUM(CNT) AS COUNT

FROM ORDER\_LIST

GROUP BY PRODUCT; -- 주문 테이블에서 제품별 수량의 합계를 검색

SELECT COMPANY, COUNT(\*) AS PRODUCT\_CNT, MAX(PRICE) AS MAX\_PRICE

FROM PRODUCT

GROUP BY COMPANY; -- 제품 테이블에서 제조사별로 제품의 개수(PRODUCT\_CNT로 출력)와 최고가(MAX\_PRICE)를 출력

SELECT COMPANY, COUNT(\*) AS PRODUCT\_CNT, MAX(PRICE) AS MAX\_PRICE

FROM PRODUCT

GROUP BY COMPANY HAVING COUNT(\*) >= 3; -- 제품을 3개 이상 제조한 제조업체만을 대상으로 제조사별 제품의 개수와 최고가를 검색

SELECT PRODUCT, CUSTOMER, CNT

FROM ORDER\_LIST

GROUP BY PRODUCT, CUSTOMER; -- 주문 테이블에서 각 주문고객이 주문한 제품의 총 주문수량을 주문제품별로 검색

-- 주문고객((각 주문고객))과 주문제품(주문제품별) 두가지의 GROUP이 필요

-- 주문제품을 먼저 GROUPING 해야 함(각 주문고객이 주문한 제품들에 관한 정보이므로)

```

#### 3.2.9 여러 테이블에 대한 조인 검색

\* 조인 속성

: 여러 개의 테이블을 연결해주는 속성

- 도메인이 같아야 조인 가능

```

SELECT PRODUCT, PRODUCT\_NAME

FROM PRODUCT, CUSTOMER

WHERE ORDER\_LIST.CUSTOMER = ‘BANANA’ AND PRODUCT.NUM = ORDER\_LIST.PRODUCT; -- 판매 DB에서 ‘BANANA’ 고객이 주문한 제품의 이름을 검색

SELECT ORDER\_PRODUCT, ORDER\_DATE

FROM CUSTOMER AS C, ORDER\_LIST AS O – C, O: TABLE의 이름을 대신함

WHERE C.AGE >= 30 – 판매 DB에서 나이가 30세 이상인 고객이 주문한 제품과 일자를 검색

```

#### 3.2.10 부속 질의문을 이용한 검색

: SELECT 문 안에 다른 SELECT 문을 포함

\* 주 질의문(main query)(= 상위 질의문)

: 다른 SELECT 문을 포함하는 SELECT 문

\* 서브 질의문(sub query)(= 부속 질의문)

: 다른 SELECT 문에 내포된 SELECT 문

- 괄호로 묶어 작성

- ORDER BY 절의 사용 불가

- 주 질의문보다 먼저 수행됨

- 서브 질의문을 다중 행 질의문으로 작성 시, 연산자 사용 불가

```

SELECT PRODUCT\_NAME, PRICE

FROM PRODUCT

WHERE COMPANY = (SELECT COMPANY

FROM PRODUCT

WHERE PRODUCT\_NAME = 'BISCUIT'); -- 서브 질의문(다중): 비스켓의 제조업체를 검색

-- 판매 DB에서 비스켓과 같은 제조업체가 제조한 제품의 제품명과 단가를 검색

SELECT CUSTOMER\_NAME, POINT

FROM CUSTOMER

WHERE POINT = (SELECT MAX(POINT) FROM CUSTOMER); -- 서브 질의문(단일): 적립금이 가장 많은 고객을 검색

-- 고객 DB에서 적립금이 가장 많은 고객의 이름과 적립금을 검색

(다중 행 부속 질의문에 사용 가능한 연산자)

SELECT PRODUCT\_NAME, COMPANY

FROM PRODUCT

WHERE NUM NOT IN('p01', 'p04', 'p06'); -- 제품 DB에서 제품번호가 'p01', 'p04', 'p06'가 아닌 제품들의 제품명과 제조업체를 검색

```

```

SELECT PRODUCT\_NAME, PRICE, COMPANY

FROM PRODUCT

WHERE PRICE > ALL (SELECT PRICE

FROM PRODUCT

WHERE COMPANY = 'DAEHAN');-- 제조업체가 DAEHAN인 모든 제품의 단가보다 비싼 제품의 제품명, 단가, 제조업체를 검색

-- ALL -> ANY/SOME: 제조업체가 DAEHAN인 어떤 제품보다 단가보다 비싼 제품의 제품명, 단가, 제조업체를 검색(제조업체가 DAEHAN인 제품 중에서 가장 단가가 낮은 제품과 비교 허용)

SELECT \*

FROM PRODUCT

WHERE EXISTS(SELECT \*

FROM ORDER\_LIST

WHERE ADDRESS LIKE 'KUNGGIDO%'

AND ORDER\_LIST.ORDER\_PRODUCT = PRODUCT.NUM); -- 주문 DB에서 주소가 경기도이고 주문 리스트 기록이 있는 제품을 검색

```

### 3.3 데이터의 삽입

#### 3.3.1 데이터 직접 삽입

```

INSERT

INTO [테이블\_이름(속성\_리스트)]

VALUES (속성값\_리스트);

```

```

INSERT

INTO CUSTOMER(ID, CUSTOMER\_NAME, AGE, GRADE, JOB, POINT)

VALUES ('MANGGO', 'LIM', 19, 'VIP', NULL, 7000); -- 고객 테이블에서 ID가 MANGGO이고, 이름이 LIM, … 적립금이 7000인 고객 데이터를 삽입

```

#### 3.3.2 부속 질의문을 이용한 데이터 삽입

```

INSERT INTO [테이블\_이름(속성\_리스트)]

SELECT 문;

```

```

INSERT

INTO HANBIT(PRODUCT, STOCK, PRICE)

SELECT PRODUCT\_NAME, STOCK, PRICE

FROM PRODUCT

WHERE COMPANY = ‘HANBIT’ – 제조업체가 한빛인 데이터들의 제품, 재고, 단가 속성을 검색하고, 이를 HANBIT이라는 테이블에 삽입

### 3.4 데이터의 수정

\* ‘UPDATE’ 키워드

```

UPDATE [테이블\_이름]

SET 속성\_이름1 = 값1, 속성\_이름2 = 값2, …

(WHERE 조건);

```

```

UPDATE ORDER\_LIST

SET CNT = CNT + 10

WHERE ORDER\_CUSTOMER IN (SELECT ID

FROM CUSTOMER

WHERE GRADE = ‘GOLD’) –GOLD 등급인 고객의 ID들에 해당하는 주문 테이블의 고객들에 한해, 수량을 10개 증가함

```

### 3.5 데이터의 삭제

\* ‘DELETE’ 키워드

```

DELETE

FROM [테이블\_이름]

(WHERE 조건);

```

```

DELETE

FROM ORDER\_LIST – 주문 테이블의 튜플이 모두 삭제됨(테이블은 남아 있음)

## 4. 뷰(view)

: 다른 테이블을 기반으로 만들어진 가상 테이블(virtual table)로

\* 데이터를 실제로 저장하고 있지 않음

- 논리적으로만 존재하면서 일반 테이블과 동일하게 사용 가능

\* 기본 테이블(base table)

: 뷰를 만드는 데 기반이 되는 물리적인 테이블

- CREATE TABLE문으로 정의한 테이블

### 4.2 뷰의 생성

```

CREATE VIEW [뷰\_이름](속성\_리스트)

AS SELECT 문 – ORDER BY는 사용 불가

(WITH CHECK OPTION); -- SELECT 문에서 제시한 뷰의 조건을 위반하면 수행되지 않음

```

```

CREATE VIEW VIP(ID, CUSTOMER\_NAME, AGE)

AS SELECT ID, CUSTOMER\_NAME, AGE

FROM CUSTOMER

WHERE GRADE = ‘VIP’

WITH CHECK OPTION; -- 등급이 VIP가 아닌 고객 데이터를 삽입/수정/삭제 시도는 모두 거부

-- 등급이 VIP인 고객들의 ID, 이름, 나이만 뷰로 생성

CREATE VIEW COMP\_CNT(COMPANY, CNT)

AS SELECT COMPANY, COUNT(\*)

FROM PRODUCT

GROUP BY COMPANY; -- 제조업체별 제품 수로 구성된 뷰를 생성

```

### 4.3 뷰의 활용

\* SELECT 문을 이용해 데이터 검색 가능

- 뷰에 대한 SELECT 문이 내부적으로는 기본 테이블에 대한 SELECT 문으로 변환되어 수행

\* INSERT, UPDATE, DELETE 문 수행 가능

- 단, 기본키로 지정된 속성을 포함한 뷰만 연산이 가능

```

INSERT INTO COMP\_CNT VALUES (‘BI’, 0)

-- INSERT 실패: 1) 기본 키인 제품번호 속성이 포함되지 않음

2) 집계 함수로 계산된 값은 새로 계산되었기 때문에 연산이 명확하지 않음

```

\* 변경이 불가능한 뷰

- 기본 테이블의 기본키를 구성하는 속성이 포함되어 있지 않은 뷰

- 기본 테이블에 있지 않은 내용을 포함하고 있는 뷰(ex. 집계함수)

- DISTINCT 키워드를 포함하여 정의한 뷰

- GROUP BY 절을 포함하여 정의한 뷰

- 여러 개의 테이블을 조인하여 정의한 뷰는 변경할 수 없는 경우가 많음

\* 뷰의 장점

1. 질의문을 좀 더 쉽게 작성 가능

- 특정 조건을 만족하는 튜플로 뷰를 만들면 이후에 WHERE, GROUP BY 등을 작성하지 않아도 SELECT, FROM 절만으로도 원하는 데이터 검색 가능

2. 데이터의 보안 유지에 도움

- 여러 사용자의 요구에 맞는 다양한 뷰를 미리 정의하고 권한 설정을 제한하면, 뷰에 포함되지 않은 데이터를 사용자로부터 보호 가능

3. 데이터를 보다 편리하게 관리 가능

- 필요한 속성만 포함하여 관리 가능, 제공된 뷰와 관련 없는 다른 테이블의 변화에도 영향을 받지 않음

### 4.4 뷰의 삭제

\* 기본 테이블은 영향 받지 않음

```

DROP VIEW 뷰\_이름 CASCADE[/RESTRICT];

```

## 5. 삽입 SQL(ESQL; Embedded SQL)

: JAVA 등과 같은 프로그래밍 언어로 작성된 응용 프로그램 안에 삽입하여 사용하는 SQL문

\* 특징

1. 프로그램 안에서 일반 명령문이 위치할 수 있는 곳이면 어디든 삽입 가능

2. 프로그램 안의 일반 명령문과 구별하기 위해 삽입 SQL문 앞에 EXEC SQL을 작성

3. 프로그램에 선언된 일반 변수를 삽입 SQL 문에서 사용 가능

- 단, SQL문에서 일반 변수를 사용하는 경우에는, 앞에 : 을 붙여 테이블이나 속성의 이름과 구분

\* 커서(cursor)

: 수행 결과로 반환된 여러 행을 한 번에 하나씩 가리키는 포인터 역할

- 프로그램에서는 SELECT 문의 수행 결과로 반환되는 여러 행을 한번에 처리할 수 없음

### 5.2 커서가 필요 없는 삽입 SQL

\* SQL 문을 실행했을 때 결과 테이블을 반환하지 않는 경우

- CREATE TABLE, INSERT, DELETE, UPDATE

\* 결과로 행 하나만 반환하는 SELECT 문

### 5.3 커서가 필요한 삽입 SQL

\* SELECT 문의 실행 결과로 여러 행이 검색되는 경우

- ‘DECLARE’: 커서를 선언

```

EXEC SQL DECLARE [커서\_이름] CURSOR FOR [SELECT 문];

```

```

EXEC SQL DECLARE PRODUCT\_CURSOR CURSOR FOR

SELECT PRODUCT\_NAME, PRICE FROM PRODUCT; -- 제품 테이블에서 제품명과 단가를 검색하는SELECT 문을 위한 PRODUCT\_CURSOR 커서를 선언

```

- 커서에 연결된 SELECT 문을 실행

```

EXEC SQL OPEN [커서\_이름] – PRODUCT\_CURSOR 커서에 연결된 SELECT 문을 실행

```

- ‘OPEN’: SELECT 문이 실행/반환되어 커서가 검색된 행들 중의 첫 번째 행의 바로 앞에 위치

- ‘FETCH’: 커서를 이동(검색된 행들을 차례로 처리하기 위함)

```

EXEC SQL FETCH [커서\_이름] INTO [변수\_리스트];

-- 일반적으로, 반복 수행을 위해 FOR, WHILE 문 등을 함께 사용

```

- ‘CLOSE’: 커서를 더 이상 사용하지 않는 경우

```

EXEC SQL CLOSE [커서\_이름];

```