SQL 활용 프로그래밍

SQL Application Programming

Contents

◦ CHAPTER 04 SQL 기본

- SECTION 01 PostgreSQL

  1.1 SQL 분류

  1.2 Schema와 Table

  1.3 Database 지정

  1.4 Schema 지정

  1.5 주석

- SECTION 02 SELECT문

  2.1 원하는 데이터를 가져와 주는 기본적인 <SELECT… FROM>

  2.2 DB, Table, 열의 이름이 확실하지 않을 때 조회하는 방법

  2.3 특정한 조건의 데이터만 조회하는 <SELECT… FROM… WHERE>

  2.4 ANY/ALL/SOME, Subquery

Contents

◦ CHAPTER 04 SQL 기본

- SECTION 02 SELECT문

  2.5 원하는 순서대로 정렬하여 출력 : ORDER BY

  2.6 특정한 조건의 데이터만 조회하는 <SELECT… FROM… WHERE>

- SECTION 03 데이터의 변경을 위한 SQL문

  3.1 데이터의 삽입 : INSERT

  3.2 데이터의 수정 : UPDATE

  3.3 데이터의 삭제 : DELETE

  3.4 조건부 데이터 입력, 변경

CHAPTER 04 기본 SQL

데이터베이스를 운영하기 위한 기본적인 SQL문에 대하여 알아본다.

SQL의 분류

- DDL (Data Definition Language, 데이터 정의 언어)
  - 데이터베이스, 테이블, 뷰, 인덱스 등의 데이터베이스 개체를 생성/삭제/변경하는 역할
  - CREATE, DROP, ALTER 자주 사용
  - **DDL은 트랜잭션 발생시키지 않음**
  - 되돌림(ROLLBACK)이나 완전적용(COMMIT) 사용 불가

- DCL (Data Control Language, 데이터 제어 언어)
  - 사용자에게 DENY 구문
  - 어떤 권한을 부여하거나 빼앗을 때 주로 사용하는 구문
  - GRANT/REVOKE

SQL의 분류

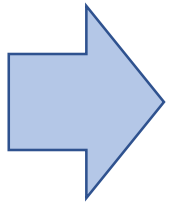
- DML (Data Manipulation Language, 데이터 조작 언어)
  - 데이터를 조작(선택, 삽입, 수정, 삭제)하는 데 사용되는 언어
  - DML 구문이 사용되는 대상은 **테이블의 행**
  - DML 사용하기 위해서는 **테이블이 정의되어 있어야 함**
  - SQL문 중 **SELECT, INSERT, UPDATE, DELETE**가 이 구문에 해당
  - 트랜잭션(Transaction)이 발생하는 SQL도 DML에 속함
    - 테이블의 데이터를 변경(입력/수정/삭제)할 때 실제 테이블에 완전히 적용하지 않고, **임시로 적용시키는 것**
    - **취소 가능**

SECTION 01 PostgreSQL

DATABASE 지정

- MySQL에서는 "USE database_name"과 같은 SQL문으로 사용하고자 하는 데이터베이스에 접속함
- PostgreSQL에서는 USE 예약어를 사용할 수 없음. 따라서 PgAdmin에서는 접속하고자 하는 데이터베이스를 직접 클릭하여 접속해야 함
- Psql을 사용하고 있을 때, 만약 특정 데이터베이스에 접속해 있는데 다른 데이터베이스에 접속하길 원하면 다음과 같이 ₩c 명령어를 통해서 다른 데이터베이스에 바로 접속할 수 있음

```
postgres=# select current_database();
 current_database
------------------
 postgres
(1개 행)
```



```
postgres=# \c employees;
접속정보: 데이터베이스="employees", 사용자="postgres".
employees=# select current_database();
 current_database
------------------
 employees
(1개 행)
```

SECTION 01 PostgreSQL

SCHEMA

- 데이터베이스에 작성되는 테이블이나 함수 등의 개체를 그룹화하는 것
- 스키마가 다르면 동일한 데이터베이스에도 동일한 테이블 이름으로 테이블을 만들 수 있음
- 데이터베이스를 작성하면 자동으로 public라는 특별한 스키마가 작성됨
- Public 스키마와는 별도로 스키마를 데이터베이스에 만들 수 있음

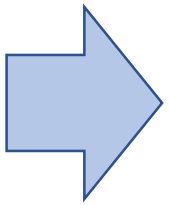
TABLE

- 테이블은 스키마에 작성함
- 스키마가 다르면 같은 테이블 이름의 테이블도 만들 수 있음. 또한 스키마마다 테이블 등의 오브젝트를 작성할 수 있는 권한을 설정할 수 있음
- CREATE TABLE 명령으로 테이블을 만들 경우 테이블 이름에 스키마를 생략하면 기본적으로 public 스키마에 테이블이 만들어짐

SET SEARCH_PATH TO SCHEMA_NAME

- 특정 데이터 베이스에 접속한 후 schema를 지정해야 schema 내의 테이블을 사용할 수 있음
- Schema를 지정하지 않고 create table을 하면 public schema에 테이블이 생성됨
- search_path 기능을 이용하여 어떤 schema에서 테이블을 우선적으로 사용할 것인지 명시함
- SET SCHEMA 'SCHEMA_NAME'을 입력해도 동일하게 작동함

```
SET search_path to schema_name;
SET SCHEMA 'schema_name';
```



```
SET search_path to shop;
SET SCHEMA 'shop';
```

주석(Remark)

- SQL문을 작성할 때, 주석을 작성하고자 하는 문장 앞에 하이픈을 두개 연속해서 입력한다.(--)

```
--한 줄 주석입니다.
```

- 여러 줄의 코멘트를 작성하려면 주석으로 작성하고자 하는 코드블록을 /*와 */로 감싸준다.

```
/*
  여러줄
  주석
  입니다.
*/
```

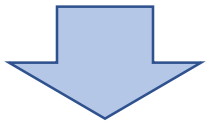
SECTION 02 SELECT문

<SELECT ··· FROM>

- 원하는 데이터를 가져와 주는 기본적인 구문
- 가장 많이 사용되는 구문
- 데이터베이스 내 테이블에서 원하는 정보 추출하는 명령

SQL은 일반적으로 **대소문자를 구별하지 않는다.**
PostgreSQL에서는 소문자가 기본이며, 대소문자를 구분하고 싶다면
명시적으로 " "(쌍따옴표)로 감싸줘야 한다.

```
SELECT select_expr
    [FROM table_references]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}]
```



```
SELECT 열 이름
FROM 테이블이름
WHERE 조건
```

SECTION 02 SELECT문

<SELECT ··· FROM>

- SELECT *
  - 다음 두 쿼리는 동일한 결과를 보여준다.

```
set schema 'employees';
select * from employees.employees;
select * from employees;
```

- SELECT 열 이름
  - 테이블에서 필요로 하는 열만 가져오기 가능

```
SELECT first_name FROM employees;
```

	first_name character varying (14)
1	Georgi
2	Bezalel
3	Parto
4	Chirstian
5	Kyoichi
6	Anneke
7	Tzvetan
8	Saniya
9	Sumant
10	Duangkaew

  - 여러 개의 열을 가져오고 싶을 때는 콤마로 구분

```
SELECT first_name, last_name, gender FROM employees;
```

	first_name character varying (14)	last_name character varying (16)	gender gen
1	Georgi	Facello	M
2	Bezalel	Simmel	F
3	Parto	Bamford	M
4	Chirstian	Koblick	M
5	Kyoichi	Maliniak	M
6	Anneke	Preusig	F
7	Tzvetan	Zielinski	F
8	Saniya	Kalloufi	M
9	Sumant	Peac	F
10	Duangkaew	Piveteau	F

  - 열 이름의 순서는 출력하고 싶은 순서대로 배열 가능

SECTION 02 SELECT문

DB, TABLE, 열의 이름이 확실하지 않을 때 조회하는 방법

- 현재 서버에 어떤 DB가 있는지 보기

```
select * from pg_database;
```

- 현재 서버에 어떤 TABLE이 있는지 보기
  - 데이터베이스에 있는 테이블 정보 조회

```
select * from pg_tables;
```

  - 테이블 이름만 간단히 보기

```
select tablename from pg_tables;
```

- employees 테이블의 열이 무엇이 있는지 확인

```
select column_name
from information_schema.columns
where table_name='employees'
order by column_name desc
```

  - Column 정보를 알고 싶은 테이블명을 작은따옴표 사이에 입력해준다

[실습 1] 데이터베이스 이름, 테이블 이름, 필드 이름이 정확히 기억나지 않거나 또는 각 이름의 철자가 확실하지 않을 때 찾아서 조회하는 방법 실습

1. 현재 서버에 어떤 데이터베이스가 있는지 조회한다.

```
select datname from PG_DATABASE;
```

	datname name
1	postgres
2	template1
3	template0
4	employees
5	sqlDB

2. 원하는 데이터베이스에 접속 후 테이블 조회

```
select * from pg_tables;
```

	schemaname name	tablename name	tableowner name	tablespace name	hasindexes boolean	hasrules boolean	hastriggers boolean	rowsecurity boolean
1	public	departments	postgres	[null]	true	false	true	false
2	public	dept_emp	postgres	[null]	true	false	true	false
3	public	employees	postgres	[null]	true	false	true	false
4	public	dept_manager	postgres	[null]	true	false	true	false
5	public	salaries	postgres	[null]	true	false	true	false
6	public	titles	postgres	[null]	true	false	true	false
7	pg_catalog	pg_statistic	postgres	[null]	true	false	false	false

.
.
.

3. 테이블 이름만 간단히 조회

```
select tablename from pg_tables;
```

	tablename name
1	departments
2	dept_emp
3	employees
4	dept_manager
5	salaries
6	titles
7	pg_statistic
8	pg_type

[실습 1] 데이터베이스 이름, 테이블 이름, 필드 이름이 정확히 기억나지 않거나 또는 각 이름의 철자가 확실하지 않을 때 찾아서 조회하는 방법 실습

4. employees 테이블의 열 확인

```
select column_name
from information_schema.columns
where table_name='employees'
```

	column_name name
1	emp_no
2	birth_date
3	gender
4	hire_date
5	first_name
6	last_name

```
select column_name
from information_schema.columns
where table_name='employees'
order by column_name desc
```

	column_name name
1	last_name
2	hire_date
3	gender
4	first_name
5	emp_no
6	birth_date

[실습 1] 데이터베이스 이름, 테이블 이름, 필드 이름이 정확히 기억나지 않거나 또는 각 이름의 철자가 확실하지 않을 때 찾아서 조회하는 방법 실습

5. 최종적으로 데이터 조회

```
select first_name, gender, hire_date
from employees;
```

	first_name character varying (14)	gender gen	hire_date date
1	Georgi	M	1986-06-26
2	Bezalel	F	1985-11-21
3	Parto	M	1986-08-28
4	Chirstian	M	1986-12-01
5	Kyoichi	M	1989-09-12
6	Anneke	F	1989-06-02
7	Tzvetan	F	1989-02-10
8	Saniya	M	1994-09-15
9	Sumant	F	1985-02-18
10	Duangkaew	F	1989-08-24
11	Mary	F	1990-01-22
12	Patricio	M	1992-12-18
13	Eberhardt	M	1985-10-20
14	Berni	M	1987-03-11
15	Guoxiang	M	1987-07-02

+ 열 이름 별칭 설정

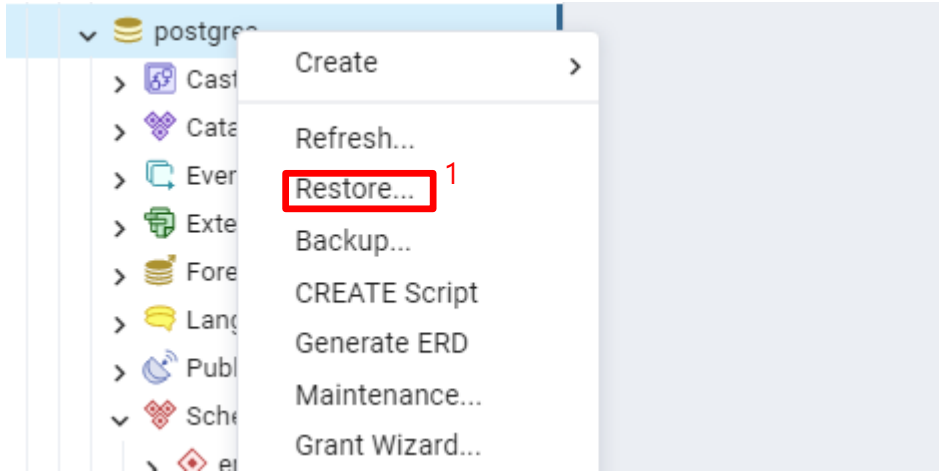
```
select first_name as 이름, gender as 성별, hire_date as "회사 입사일"
from employees;
```

띄어쓰기 있을 경우
큰 따옴표 " " 로 감싸주기

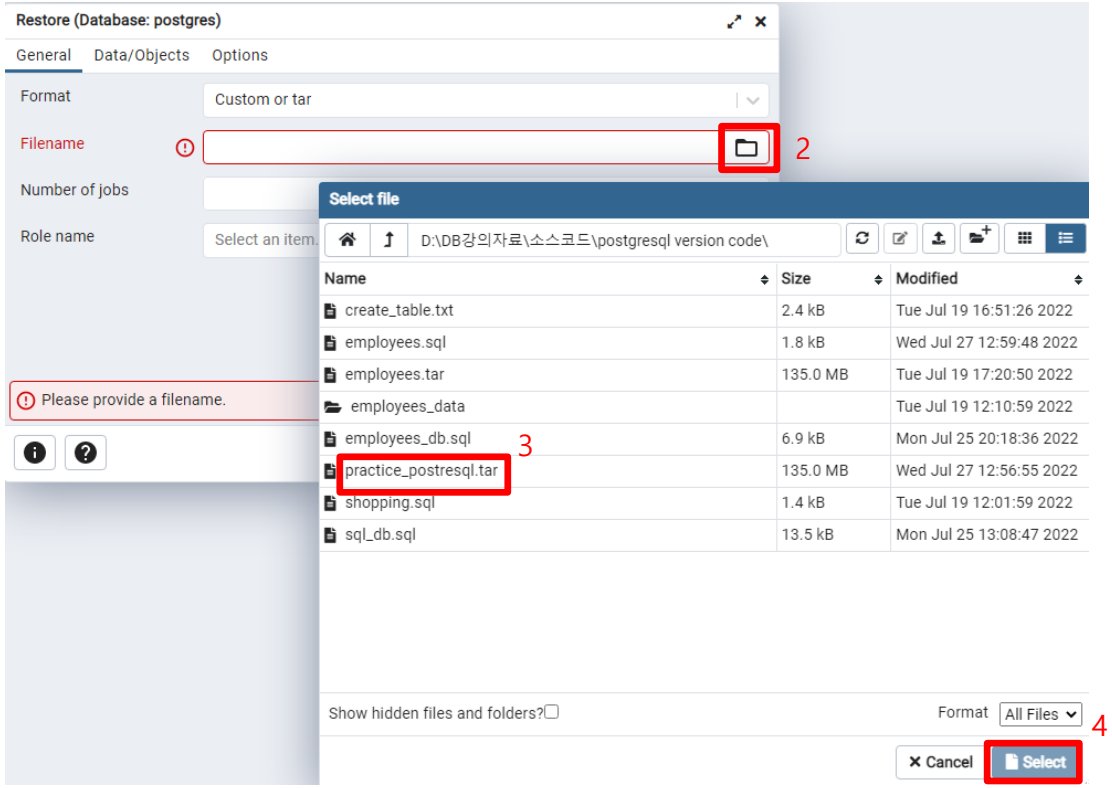
	이름 character varying (14)	성별 gen	회사 입사일 date
1	Georgi	M	1986-06-26
2	Bezalel	F	1985-11-21
3	Parto	M	1986-08-28
4	Chirstian	M	1986-12-01
5	Kyoichi	M	1989-09-12
6	Anneke	F	1989-06-02
7	Tzvetan	F	1989-02-10
8	Saniya	M	1994-09-15
9	Sumant	F	1985-02-18
10	Duangkaew	F	1989-08-24
11	Mary	F	1990-01-22
12	Patricio	M	1992-12-18
13	Eberhardt	M	1985-10-20
14	Berni	M	1987-03-11
15	Guoxiang	M	1987-07-02

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(PgAdmin4 사용)

1. DB RESTORE



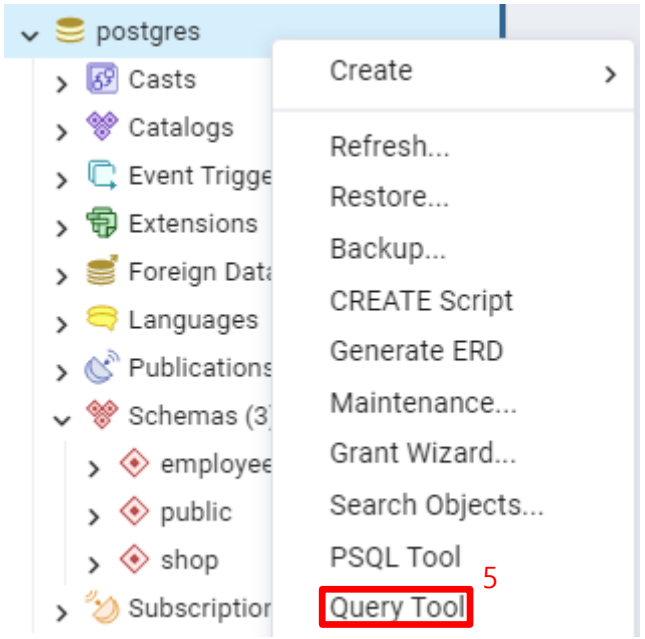
'postgres' 우클릭 -> Restore Click



폴더 모양 클릭 -> 'practice_postgresql.tar' 클릭
-> 'select' 클릭

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(PgAdmin4 사용)

2. ‘shop’ SCHEMA 안에 TABLE 생성



‘postgres’ 커서 지정 후 마우스 오른쪽 클릭하고 ‘Query Tool’ 클릭

```sql
set schema 'shop';

CREATE TABLE usertbl -- 회원 테이블
( userID    CHAR(8) NOT NULL PRIMARY KEY, -- 사용자 아이디(PK)
  name      VARCHAR(10) NOT NULL, -- 이름
  birthYear   INT NOT NULL,  -- 출생년도
  addr      CHAR(2) NOT NULL, -- 지역(경기,서울,경남 식으로 2글자만입력)
  mobile1   CHAR(3), -- 휴대폰의 국번(011, 016, 017, 018, 019, 010 등)
  mobile2   CHAR(8), -- 휴대폰의 나머지 전화번호(하이픈제외)
  height        SMALLINT,  -- 키
  mDate     DATE  -- 회원 가입일
);

CREATE TABLE buytbl -- 회원 구매 테이블(Buy Table의 약자)
(  num      SERIAL NOT NULL PRIMARY KEY, -- 순번(PK)
   userID   CHAR(8) NOT NULL, -- 아이디(FK)
   prodName     CHAR(6) NOT NULL, --  물품명
   groupName    CHAR(4)  , -- 분류
   price        INT  NOT NULL, -- 단가
   amount       SMALLINT  NOT NULL, -- 수량
   FOREIGN KEY (userID) REFERENCES usertbl(userID)
);
```

위와 같은 SQL문을 이용하여 ‘usertbl’, ‘buytbl’ 테이블 생성(코드 주어질 예정)

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(PgAdmin4 사용)

3. INSERT를 이용하여 데이터 입력

```
INSERT INTO usertbl VALUES('LSG', '이승기', 1987, '서울', '011', '1111111', 182, '2008-8-8');
INSERT INTO usertbl VALUES('KBS', '김범수', 1979, '경남', '011', '2222222', 173, '2012-4-4');
INSERT INTO usertbl VALUES('KKH', '김경호', 1971, '전남', '019', '3333333', 177, '2007-7-7');
INSERT INTO usertbl VALUES('JYP', '조용필', 1950, '경기', '011', '4444444', 166, '2009-4-4');
INSERT INTO usertbl VALUES('SSK', '성시경', 1979, '서울', NULL  , NULL      , 186, '2013-12-12');
INSERT INTO usertbl VALUES('LJB', '임재범', 1963, '서울', '016', '6666666', 182, '2009-9-9');
INSERT INTO usertbl VALUES('YJS', '윤종신', 1969, '경남', NULL  , NULL      , 170, '2005-5-5');
INSERT INTO usertbl VALUES('EJW', '은지원', 1972, '경북', '011', '8888888', 174, '2014-3-3');
INSERT INTO usertbl VALUES('JKW', '조관우', 1965, '경기', '018', '9999999', 172, '2010-10-10');
INSERT INTO usertbl VALUES('BBK', '바비킴', 1973, '서울', '010', '0000000', 176, '2013-5-5');
INSERT INTO buytbl VALUES(DEFAULT, 'KBS', '운동화', NULL   , 30,   2);
INSERT INTO buytbl VALUES(DEFAULT, 'KBS', '노트북', '전자', 1000, 1);
INSERT INTO buytbl VALUES(DEFAULT, 'JYP', '모니터', '전자', 200,  1);
INSERT INTO buytbl VALUES(DEFAULT, 'BBK', '모니터', '전자', 200,  5);
INSERT INTO buytbl VALUES(DEFAULT, 'KBS', '청바지', '의류', 50,   3);
INSERT INTO buytbl VALUES(DEFAULT, 'BBK', '메모리', '전자', 80,  10);
INSERT INTO buytbl VALUES(DEFAULT, 'SSK', '책'    , '서적', 15,   5);
INSERT INTO buytbl VALUES(DEFAULT, 'EJW', '책'    , '서적', 15,   2);
INSERT INTO buytbl VALUES(DEFAULT, 'EJW', '청바지', '의류', 50,   1);
INSERT INTO buytbl VALUES(DEFAULT, 'BBK', '운동화', NULL   , 30,   2);
INSERT INTO buytbl VALUES(DEFAULT, 'EJW', '책'    , '서적', 15,   1);
INSERT INTO buytbl VALUES(DEFAULT, 'BBK', '운동화', NULL   , 30,   2);
```

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(PgAdmin4 사용)

4. 데이터 확인

```
SELECT * FROM usertbl;
SELECT * FROM buytbl;
```

	userid [PK] character (8)	name character varying (10)	birthyear integer	addr character (2)	mobile1 character (3)	mobile2 character (8)	height smallint	mdate date
1	LSG	이승기	1987	서울	011	1111111	182	2008-08-08
2	KBS	김범수	1979	경남	011	2222222	173	2012-04-04
3	KKH	김경호	1971	전남	019	3333333	177	2007-07-07
4	JYP	조용필	1950	경기	011	4444444	166	2009-04-04
5	SSK	성시경	1979	서울	[null]	[null]	186	2013-12-12
6	LJB	임재범	1963	서울	016	6666666	182	2009-09-09
7	YJS	윤종신	1969	경남	[null]	[null]	170	2005-05-05
8	EJW	은지원	1972	경북	011	8888888	174	2014-03-03
9	JKW	조관우	1965	경기	018	9999999	172	2010-10-10
10	BBK	바비킴	1973	서울	010	0000000	176	2013-05-05

	num [PK] integer	userid character (8)	prodname character (6)	groupname character (4)	price integer	amount smallint
1	1	KBS	운동화	[null]	45	2
2	2	KBS	노트북	전자	1500	1
3	3	JYP	모니터	전자	300	1
4	4	BBK	모니터	전자	300	5
5	5	KBS	청바지	의류	75	3
6	6	BBK	메모리	전자	120	10
7	7	SSK	책	서적	23	5
8	8	EJW	책	서적	23	2
9	9	EJW	청바지	의류	75	1
10	10	BBK	운동화	[null]	45	2
11	11	EJW	책	서적	23	1
12	12	BBK	운동화	[null]	45	2

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(psql 사용)

1. DB RESTORE

```
D:\>pg_restore -h localhost -d postgres -U postgres -F t -v D:/practice_postgresql.tar
pg_restore: 복원 작업을 위해 데이터베이스에 접속 중
암호:
pg_restore: SCHEMA "employees" 만드는 중
pg_restore: SCHEMA "shop" 만드는 중
pg_restore: EXTENSION "adminpack" 만드는 중
pg_restore: COMMENT "EXTENSION adminpack" 만드는 중
pg_restore: TYPE "employees.gen" 만드는 중
pg_restore: TABLE "employees.departments" 만드는 중
pg_restore: TABLE "employees.dept_emp" 만드는 중
```

- **호스트 이름** : localhost(-h *host*,--host=*host*)
- **데이터 베이스 명** : postgres(-d *dbname*,--dbname=*dbname*)
- **데이터 베이스 사용자 이름** : postgres(-U *username*,--username=*username*)
- **파일 타입** : t = tar type(-F *format,* --format*=format*),
- **실행 과정 출력** : -v(--verbose)
- **파일 경로 지정** : D:/practice_postgresql.tar(상대 경로 지정 가능)

위와 같은 옵션으로 'practice_postgresql.tar' DB 파일 resotre 진행

PostgreSQL: Documentation: 9.6: pg_restore

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(psql 사용)

2. DB 접속

```
D:\>psql -h localhost -U postgres -p 5432 -d postgres
postgres 사용자의 암호:
psql (14.4)
도움말을 보려면 "help"를 입력하십시오.

postgres=#
```

or

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
postgres 사용자의 암호:
psql (14.4)
도움말을 보려면 "help"를 입력하십시오.

postgres=#
```

'명령 프롬프트' 혹은 'SQL shell(psql)'을 실행하여 다음과 같은 옵션으로 'postgres' 데이터베이스에 접속

- **호스트 이름** : localhost(-h *host*,--host=*host*)
- **데이터 베이스 사용자 이름**: postgres(-U *username*,--username=*username*)
- **포트 번호** : 5432(-p *port*, --port=*port*)
- **데이터 베이스 명** : postgres(-d *dbname*,--dbname=*dbname*)

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(psql 사용)

3. ‘shop’ SCHEMA 안에 buytbl, usertbl TABLE 생성 및 데이터 삽입

```
postgres=# \i D:/create_table.sql
SET
SET
CREATE TABLE
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
SET
```

- ₩i 명령어를 이용하여 주어진 파일에서 sql 명령어를 실행하도록 함
- 해당 파일에는 ‘shop’ schema 지정, ‘buytbl’과 ‘usertbl’ table 생성, 데이터 삽입 sql문이 저장되어 있음
- 단, 한글을 포함한 데이터 삽입 시 encoding 문제가 발생할 우려가 있음. (자세한 내용은 다음 슬라이드를 참조)

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(psql 사용)

3. ‘shop’ SCHEMA 안에 buytbl, usertbl TABLE 생성 및 데이터 삽입

※ 주의 사항

- cmd에서 한글을 포함한 데이터의 삽입을 진행하면 다음과 같은 encoding 문제가 발생할 수 있음

```
psql:D:/DB강의자료_최신/source/create_table.sql:25: 오류:  0xec 0x9d 바이트로 조합된
문자(인코딩: "UHC")와 대응되는 문자 코드가 "UTF8" 인코딩에는 없습니다
```

- 따라서 다음과 같이 server와 client의 encoding을 확인한 후 encoding이 ‘UHC’라면 ‘UTF-8’(server encoding과 맞춰 줘야함)로 바꾸어주고, Insert를 진행해야 함.

```
postgres=# show server_encoding;
 server_encoding
-----------------
 UTF8
(1개 행)
```

```
postgres=# show client_encoding;
 client_encoding
-----------------
 UHC
(1개 행)


postgres=# set client_encoding='UTF-8';
SET
```

- Insert가 끝난 후 다시 ‘UHC’로 변경해주어야함. 변경해주지 않으면 다음과 같이 한글이 깨져서 보임

```
          릴레이션(relation) 목록
 \xBD\xBA키\xB8\xB6 | \u000C\xB8\xA7 |  종\xB7\xF9  | \xBC弩\xAF주
-------------------+----------------+-------------+------------
 shop              | buytbl         | 테\u000C\xBA | postgres
 shop              | usertbl        | 테\u000C\xBA | postgres
(2개 행)
```

- ‘create_table.sql’파일에는 위와 같은 encoding 처리과정이 모두 포함되어 있음

[실습 2] 앞으로 책의 전 과정에서 사용할 데이터베이스와 테이블 생성(psql 사용)

4. 데이터 확인

```
postgres=# SELECT * FROM buytbl;
 num |  userid  | prodname | groupname | price | amount
-----+----------+----------+-----------+-------+--------
   1 | KBS      | 운동화   |           |    30 |      2
   2 | KBS      | 노트북   | 전자      |  1000 |      1
   3 | JYP      | 모니터   | 전자      |   200 |      1
   4 | BBK      | 모니터   | 전자      |   200 |      5
   5 | KBS      | 청바지   | 의류      |    50 |      3
   6 | BBK      | 메모리   | 전자      |    80 |     10
   7 | SSK      | 책       | 서적      |    15 |      5
   8 | EJW      | 책       | 서적      |    15 |      2
   9 | EJW      | 청바지   | 의류      |    50 |      1
  10 | BBK      | 운동화   |           |    30 |      2
  11 | EJW      | 책       | 서적      |    15 |      1
  12 | BBK      | 운동화   |           |    30 |      2
(12개 행)
```

```
postgres=# SELECT * FROM usertbl;
  userid  |  name  | birthyear | addr | mobile1 | mobile2  | height |    mdate
----------+--------+-----------+------+---------+----------+--------+------------
 LSG      | 이승기 |      1987 | 서울 | 011     | 1111111  |    182 | 2008-08-08
 KBS      | 김범수 |      1979 | 경남 | 011     | 2222222  |    173 | 2012-04-04
 KKH      | 김경호 |      1971 | 전남 | 019     | 3333333  |    177 | 2007-07-07
 JYP      | 조용필 |      1950 | 경기 | 011     | 4444444  |    166 | 2009-04-04
 SSK      | 성시경 |      1979 | 서울 |         |          |    186 | 2013-12-12
 LJB      | 임재범 |      1963 | 서울 | 016     | 6666666  |    182 | 2009-09-09
 YJS      | 윤종신 |      1969 | 경남 |         |          |    170 | 2005-05-05
 EJW      | 은지원 |      1972 | 경북 | 011     | 8888888  |    174 | 2014-03-03
 JKW      | 조관우 |      1965 | 경기 | 018     | 9999999  |    172 | 2010-10-10
 BBK      | 바비킴 |      1973 | 서울 | 010     | 0000000  |    176 | 2013-05-05
(10개 행)
```

SECTION 02 SELECT문

특정 조건의 데이터만 조회 - <SELECT … FROM … WHERE>

- 기본적인 WHERE절
  - 조회하는 결과에 특정한 조건을 줘서 원하는 데이터만 보고 싶을 때 사용
  - SELECT 필드이름 FROM 테이블이름 WHERE 조건식;
    - ex)

```
SELECT * FROM usertbl WHERE name = '김경호';
```

	userid [PK] character (8)	name character varying (10)	birthyear integer	addr character (2)	mobile1 character (3)	mobile2 character (8)	height smallint	mdate date
1	KKH	김경호	1971	전남	019	3333333	177	2007-07-07

- 관계 연산자의 사용
  - OR 연산자 : '…했거나', '… 또는'
  - AND 연산자 : '…하고', '…면서', '… 그리고'
  - 조건 연산자(=, <, >, <=, >=, < >, != 등)와 관계 연산자(NOT, AND, OR 등)를 조합하여 데이터를 효율적으로 추출 가능
    - ex)

```
SELECT userID, Name FROM usertbl WHERE birthYear >= 1970 AND height >= 182;
```

	userid [PK] character (8)	name character varying (10)
1	LSG	이승기
2	SSK	성시경

SECTION 02 SELECT문

특정 조건의 데이터만 조회 - <SELECT … FROM … WHERE>

- BETWEEN… AND와 IN( ) 그리고 LIKE
  - 데이터가 숫자로 구성되어 있으며 연속적인 값 : **BETWEEN … AND 사용**
    - ex)

```
SELECT name,  height FROM usertbl WHERE height BETWEEN 180 AND 183;
```

	name character varying (10)	height smallint
1	이승기	182
2	임재범	182

  - 이산적인(Discrete) 값의 조건 : **IN() 사용**
    - ex)

```
SELECT name,  addr FROM usertbl WHERE addr IN ('경남','전남','경북');
```

	name character varying (10)	addr character (2)
1	김범수	경남
2	김경호	전남
3	윤종신	경남
4	은지원	경북

  - 문자열의 내용 검색 : **LIKE 사용(문자 뒤에 % - 무엇이든 허용, 한 글자와 매치 '_' 사용)**
    - ex)

```
SELECT name,  height FROM usertbl WHERE name LIKE '김%';
```

	name character varying (10)	height smallint
1	김범수	173
2	김경호	177

SECTION 02 SELECT문

ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

- ANY
  - 서브쿼리의 여러 개의 결과 중 한 가지만 만족해도 가능
  - SOME은 ANY와 동일한 의미로 사용
  - '= ANY(서브쿼리)'는 'IN(서브쿼리)'와 동일한 의미

- ALL
  - 서브쿼리의 결과 중 여러 개의 결과를 모두 만족해야 함

ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

- 서브쿼리
  - 쿼리문 안에 또 쿼리문이 들어 있는 것
  - 서브쿼리 사용하는 쿼리로 변환 예제
    - ex) 김경호보다 키가 크거나 같은 사람의 이름과 키 출력
      - WHERE 조건에 김경호의 키를 직접 써주는 것을 쿼리로 해결

```
SELECT name,  height FROM usertbl WHERE height  > 177;
```

	name character varying (10)	height smallint
1	이승기	182
2	성시경	186
3	임재범	182

```
SELECT name,  height FROM usertbl
   WHERE height > (SELECT height FROM usertbl WHERE Name = '김경호');
```

	name character varying (10)	height smallint
1	이승기	182
2	성시경	186
3	임재범	182

SECTION 02 SELECT문

ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

- 서브쿼리의 결과가 둘 이상이 되면 에러 발생

(SELECT height FROM usertbl WHERE addr = '경남'); 이 173과 170이라는 두 개의 값을 반환하기 때문에 발생하는 오류

```
SELECT name, height FROM usertbl
   WHERE height >= (SELECT height FROM usertbl WHERE addr = '경남');
```

```
ERROR: 오류:  표현식에 사용된 서브쿼리 결과가 하나 이상의 행을 리턴했습니다
```

```
SELECT name, height FROM usertbl
   WHERE height >= ANY (SELECT height FROM usertbl WHERE addr = '경남');
```

	name character varying (10)	height smallint
1	이승기	182
2	김범수	173
3	김경호	177
4	성시경	186
5	임재범	182
6	윤종신	170
7	은지원	174
8	조관우	172
9	바비킴	176

SECTION 02 SELECT문

ANY/ALL/SOME ,서브쿼리(SubQuery, 하위쿼리)

```
SELECT name, height FROM usertbl
   WHERE height >= ALL (SELECT height FROM usertbl WHERE addr = '경남');
```

	name character varying (10)	height smallint
1	이승기	182
2	김범수	173
3	김경호	177
4	성시경	186
5	임재범	182
6	은지원	174
7	바비킴	176

```
SELECT name, height FROM usertbl
   WHERE height = ANY (SELECT height FROM usertbl WHERE addr = '경남');
```

	name character varying (10)	height smallint
1	김범수	173
2	윤종신	170

```
SELECT name, height FROM usertbl
  WHERE height IN (SELECT height FROM usertbl WHERE addr = '경남');
```

	name character varying (10)	height smallint
1	김범수	173
2	윤종신	170

원하는 순서대로 정렬하여 출력 : ORDER BY

- ORDER BY절
  - 결과물에 대해 영향을 미치지는 않고 출력되는 순서를 조절하는 구문
  - **기본적으로 오름차순** (ASCENDING) 정렬
  - **내림차순**(DESCENDING)으로 정렬하려면 **열 이름 뒤에 DESC**
  - ORDER BY 구문을 혼합해 사용하는 구문도 가능
    - 키가 큰 순서로 정렬하되 만약 키가 같을 경우 이름 순으로 정렬

```
SELECT name,  height FROM usertbl ORDER BY height DESC, name ASC;
```

	name character varying (10)	height smallint
1	성시경	186
2	이승기	182
3	임재범	182
4	김경호	177
5	바비킴	176
6	은지원	174
7	김범수	173
8	조관우	172
9	윤종신	170
10	조용필	166

    - ASC(오름차순)는 디폴트 값이므로 생략 가능

SECTION 02 SELECT문

- 중복된 것은 하나만 남기는 DISTINCT
  - 중복된 것을 골라서 세기 어려울 때 사용하는 구문
  - 테이블의 크기가 클수록 효율적
  - 중복된 것은 1개씩만 보여주면서 출력

```
SELECT addr FROM usertbl;
```

	addr character (2)
1	서울
2	경남
3	전남
4	경기
5	서울
6	서울
7	경남
8	경북
9	경기
10	서울

```
SELECT addr FROM usertbl ORDER BY addr;
```

	addr character (2)
1	경기
2	경기
3	경남
4	경남
5	경북
6	서울
7	서울
8	서울
9	서울
10	전남

```
SELECT DISTINCT addr FROM usertbl;
```

	addr character (2)
1	서울
2	경북
3	전남
4	경기
5	경남

SECTION 02 SELECT문

- 출력하는 개수를 제한하는 LIMIT
  - 일부를 보기 위해 여러 건의 데이터를 출력하는 부담 줄임
  - 상위의 N개만 출력하는 'LIMIT N' 구문 사용

```
set schema 'employees';
select emp_no, hire_date
from employees
order by hire_date asc;
```

	emp_no [PK] integer	hire_date date
1	110085	1985-01-01
2	111692	1985-01-01
3	111035	1985-01-01
4	111400	1985-01-01
5	110183	1985-01-01
6	110511	1985-01-01
7	110725	1985-01-01
8	110022	1985-01-01
9	110303	1985-01-01
10	110114	1985-01-14
11	87761	1985-02-01
12	98636	1985-02-01
13	200241	1985-02-01
14	102004	1985-02-01
15	98018	1985-02-01
16	20539	1985-02-01

```
set schema 'employees';
select emp_no, hire_date
from employees
order by hire_date asc
limit 5;
```

	emp_no [PK] integer	hire_date date
1	110303	1985-01-01
2	110183	1985-01-01
3	110022	1985-01-01
4	110085	1985-01-01
5	110511	1985-01-01

```
LIMIT 5 OFFSET 0; -- LIMIT 5와 동일
```

SECTION 02 SELECT문

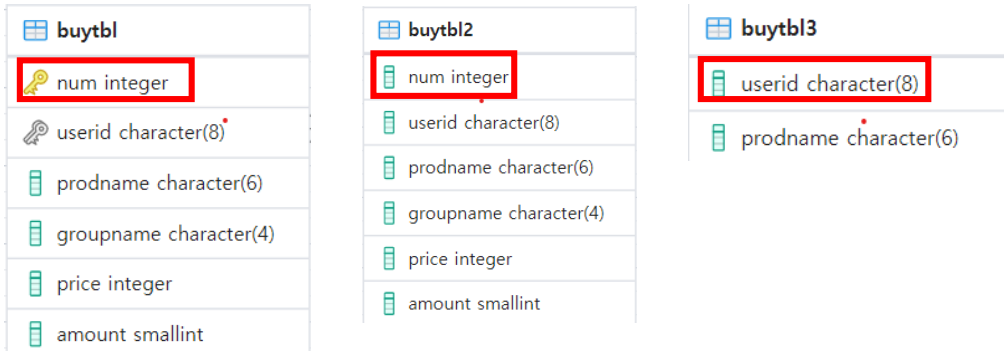
- 테이블을 복사하는 CREATE TABLE … SELECT
  - 테이블을 복사해서 사용할 경우 주로 사용
  - CREATE TABLE 새로운 테이블 (SELECT 복사할 열 FROM 기존테이블)
  - 지정한 일부 열만 복사하는 것도 가능
  - PK나 FK 같은 제약 조건은 복사되지 않음

```
set schema 'shop';
create table buytbl2 as (select * from buytbl);
select * from buytbl2;
```

	num integer	userid character (8)	prodname character (6)	groupname character (4)	price integer	amount smallint
1	1	KBS	운동화	[null]	30	2
2	2	KBS	노트북	전자	1000	1
3	3	JYP	모니터	전자	200	1
4	4	BBK	모니터	전자	200	5
5	5	KBS	청바지	의류	50	3
6	6	BBK	메모리	전자	80	10
7	7	SSK	책	서적	15	5
8	8	EJW	책	서적	15	2
9	9	EJW	청바지	의류	50	1
10	10	BBK	운동화	[null]	30	2
11	11	EJW	책	서적	15	1
12	12	BBK	운동화	[null]	30	2

```
set schema 'shop';
create table buytbl3 as (select userID,prodName from buytbl);
select * from buytbl3;
```

	userid character (8)	prodname character (6)
1	KBS	운동화
2	KBS	노트북
3	JYP	모니터
4	BBK	모니터
5	KBS	청바지
6	BBK	메모리
7	SSK	책
8	EJW	책
9	EJW	청바지
10	BBK	운동화
11	EJW	책
12	BBK	운동화

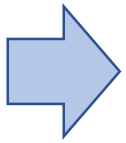


PK나 FK 등의 제약조건은 복사되지 않음

GROUP BY 및 HAVING 그리고 집계 함수

- GROUP BY절
  - 그룹으로 묶어주는 역할
  - 집계 함수(Aggregate Function)와 함께 사용
    - 효율적인 데이터 그룹화 (Grouping)
    - ex) 각 사용자 별로 구매한 개수를 합쳐 출력

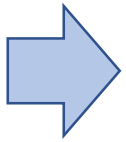
```
select userID, sum(amount) from buytbl group by userID
```



	userid character (8)	sum bigint
1	KBS	6
2	BBK	19
3	EJW	4
4	JYP	1
5	SSK	5

  - 읽기 좋게 하기 위해 별칭(Alias) AS 사용

```
select userID as "사용자 아이디", sum(amount) as "총 구매 개수"
from buytbl group by userID
```



	사용자 아이디 character (8)	총 구매 개수 bigint
1	KBS	6
2	BBK	19
3	EJW	4
4	JYP	1
5	SSK	5

GROUP BY 및 HAVING 그리고 집계 함수

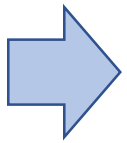
◦ GROUP BY와 함께 자주 사용되는 집계 함수

함수명	설명
AVG( )	평균을 구한다.
MIN( )	최소값을 구한다.
MAX( )	최대값을 구한다.
COUNT( )	행의 개수를 센다.
COUNT(DISTINCT)	행의 개수를 센다(중복은 1개만 인정).
STDEV( )	표준편차를 구한다.
VAR_SAMP( )	분산을 구한다.

[표 6-1] GROUP BY와 함께 사용되는 집계 함수

- ex) 전체 구매자가 구매한 물품의 개수 평균

```
select avg(amount) as "평균 구매 개수"
from buytbl;
```



	평균 구매 개수 numeric
1	2.9166666666666667

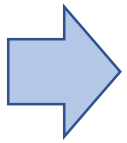
GROUP BY 및 HAVING 그리고 집계 함수

```
select userID as "사용자", sum(price*amount) as "총구매액"
from buytbl
group by userID
having sum(price*amount) > 1000
order by sum(price*amount);
```

	사용자 character (8)	총구매액 bigint
1	KBS	1210
2	BBK	1920

- Having절
  - WHERE와 비슷한 개념으로 조건 제한하는 것이지만, 집계 함수에 대해서 조건을 제한하는 것
  - HAVING절은 꼭 GROUP BY절 다음에 나와야 함(순서 바뀌면 안됨)
- ROLLUP
  - 총합 또는 중간 합계가 필요할 경우 사용
  - GROUP BY절과 함께 WITH ROLLUP문 사용
    - ex) 분류(groupName) 별로 합계 및 그 총합 구하기

```
select num, groupName, sum(price*amount) as "비용"
from buytbl
group by
rollup(groupName,num)
order by groupName;
```



	num [PK] integer	groupname character (4)	비용 bigint	
1	7	서적	75	
2	8	서적	30	
3	11	서적	15	
4	[null]	서적	120	소합계
5	5	의류	150	
6	9	의류	50	
7	[null]	의류	200	소합계
8	2	전자	1000	
9	3	전자	200	
10	4	전자	1000	
11	6	전자	800	
12	[null]	전자	3000	소합계
13	1	[null]	60	
14	10	[null]	60	
15	12	[null]	60	
16	[null]	[null]	180	소합계
17	[null]	[null]	3500	총합계

SECTION 03 데이터의 변경을 위한 SQL문

데이터의 삽입 : INSERT

- INSERT문의 기본

```
INSERT [INTO] 테이블[(열1, 열2, …)] VALUES (값1, 값2 …)
```

  - 테이블 이름 다음에 나오는 열 생략 가능
    - 생략할 경우에 VALUES 다음에 나오는 값들의 순서 및 개수가 테이블이 정의된 열 순서 및 개수와 동일해야 함

- 자동으로 증가하는 Sequence 타입을 지원

```
CREATE TABLE table_name(
    col_name SERIAL
)
```

  - 1부터 증가하는 값 자동 입력
  - SERIAL 키워드를 이용하여 Sequence 타입을 사용할 수 있음
  - INTEGER TYPE의 시퀀스를 생성
  - NOT NULL 제약 조건이 적용되어 null 값을 삽입할 수 없음
  - 대부분의 경우 실수로 중복 값이 삽입되는 것을 방지하기 위해 UNIQUE 또는 PRIMARY KEY 제약 조건을 첨부하려고 하지만 자동으로 수행되지는 않음

SECTION 03 데이터의 변경을 위한 SQL문

데이터의 삽입 : INSERT

◦ 대량의 샘플 데이터 생성

- INSERT INTO … SELECT 구문 사용

```
형식:
INSERT INTO 테이블이름 (열 이름1, 열 이름2, …)
   SELECT문  ;
```

- 다른 테이블의 데이터를 가져와 대량으로 입력하는 효과
- SELECT문의 열의 개수 = INSERT 할 테이블의 열의 개수
- 테이블 정의 까지 생략 하려면 CREATE TABLE … SELECT 구문을 사용

[실습 3] INSERT문 사용해보기

1. 열 이름 생략 가능

```
set schema 'public';
create table testTbl1(id int, userName char(3),age int);
insert into testTbl1 values(1,'홍길동',25)
```

	id integer	username character (3)	age integer
1	1	홍길동	25

2. 원하는 데이터 입력

```
insert into testTbl1(id,userName) values(2,'설현');
```

VALUES 다음에 나오는 값들의 순서 및 개수가 테이블이 정의된 열 순서 및 개수와 동일해야 함

	id integer	username character (3)	age integer
1	1	홍길동	25
2	2	설현	[null]

3. 열의 순서 바꿔서 입력 가능

```
insert into testTbl1(userName,age,id) values('하니',26,3);
```

	id integer	username character (3)	age integer
1	1	홍길동	25
2	2	설현	[null]
3	3	하니	26

[실습 3] INSERT문 사용해보기

4. 자동으로 증가하는 SERIAL TYPE

```
create table testTbl2(
    id serial primary key,
    userName char(3),
    age integer
);
insert into testTbl2 values (DEFAULT,'지민',25);
insert into testTbl2 values (DEFAULT,'유나',22);
insert into testTbl2 values (DEFAULT,'유경',21);
select * from testTbl2;
```

```
alter sequence testTbl2_id_seq restart with 100;
insert into testTbl2 values(default,'찬미',23);
select * from testTbl2;
```

```
create table testTbl3(
    id serial primary key,
    userName char(3),
    age integer
);
alter sequence testTbl3_id_seq restart with 1000;
alter sequence testTbl3_id_seq increment by 3;
insert into testTbl3 values (default,'나연',20);
insert into testTbl3 values (default,'정연',18);
insert into testTbl3 values (default,'모모',19);
select * from testTbl3;
```

테이블 생성시 SERIAL로 지정하여 자동으로 INDEX가 증가하며 삽입되는 것을 확인할 수 있음

	id [PK] integer	username character (3)	age integer
1	1	지민	25
2	2	유나	22
3	3	유경	21

	currval bigint
1	3

어느 숫자까지 증가되었는지 확인할 때 사용

	id [PK] integer	username character (3)	age integer
1	1	지민	25
2	2	유나	22
3	3	유경	21
4	100	찬미	23

	id [PK] integer	username character (3)	age integer
1	1000	나연	20
2	1003	정연	18
3	1006	모모	19

3씩 증가하는 것을 확인할 수 있음

[실습 3] INSERT문 사용해보기

5. 대량의 데이터 생성

```
set schema 'employees';
create table testTbl4(
    id integer,
    Fname varchar(50),
    Lname varchar(50)
);
insert into testTbl4
    select emp_no,first_name,last_name
    from employees;
```

	id integer	fname character varying (50)	lname character varying (50)
1	10001	Georgi	Facello
2	10002	Bezalel	Simmel
3	10003	Parto	Bamford
4	10004	Chirstian	Koblick
5	10005	Kyoichi	Maliniak
6	10006	Anneke	Preusig
7	10007	Tzvetan	Zielinski
8	10008	Saniya	Kalloufi
9	10009	Sumant	Peac
10	10010	Duangkaew	Piveteau
11	10011	Mary	Sluis
12	10012	Patricio	Bridgland

```
create table testTbl5 as (
    select emp_no,first_name,last_name
    from employees
);
```

	emp_no integer	first_name character varying (14)	last_name character varying (16)
1	10001	Georgi	Facello
2	10002	Bezalel	Simmel
3	10003	Parto	Bamford
4	10004	Chirstian	Koblick
5	10005	Kyoichi	Maliniak
6	10006	Anneke	Preusig
7	10007	Tzvetan	Zielinski
8	10008	Saniya	Kalloufi
9	10009	Sumant	Peac
10	10010	Duangkaew	Piveteau
11	10011	Mary	Sluis
12	10012	Patricio	Bridgland

SECTION 03 데이터의 변경을 위한 SQL문

데이터의 수정 : UPDATE

- 기존에 입력되어 있는 값 변경하는 구문

```
UPDATE 테이블이름
    SET 열1=값1, 열2=값2 …
    WHERE 조건 ;
```

- WHERE절 생략 가능하나 WHERE절 생략하면 테이블의 전체 행의 내용 변경됨
  - 실무에서 실수가 종종 일어남, **주의 필요**
  - 원상태로 복구하기 복잡하며, 다시 되돌릴 수 없는 경우도 있음
  - Ex) WHERE절 없이 UPDATE testTbl4 SET Lname = '없음' 실행 시 전체 행 Lname 모두 '없음' 으로 변경됨

[실습 4] UPDATE문 사용해보기

1. Kyoichi의 Lname을 '없음 ' 으로 변경

```
UPDATE testTbl4
    SET Lname = '없음'
    WHERE Fname = 'Kyoichi';
```

	id integer	fname character varying (50)	lname character varying (50)
1	10001	Georgi	Facello
2	10002	Bezalel	Simmel
3	10003	Parto	Bamford
4	10004	Chirstian	Koblick
5	10005	Kyoichi	없음

- 만약, 실수로 WHERE절을 빼먹고 **UPDATE testTbl4 SET Lname = '없음'**을 실행했다면 전체 행의 Lname이 모두 '없음' 으로 변경된다.
- 예) 구매 테이블에서 현재 단가가 모두 1.5배 인상되었다면 다음과 같이 사용할 수 있다.

```
set schema 'shop';
UPDATE buytbl SET price = price * 1.5 ;
```

	num [PK] integer	userid character (8)	prodname character (6)	groupname character (4)	price integer	amount smallint
1	1	KBS	운동화	[null]	45	2
2	2	KBS	노트북	전자	1500	1
3	3	JYP	모니터	전자	300	1
4	4	BBK	모니터	전자	300	5
5	5	KBS	청바지	의류	75	3
6	6	BBK	메모리	전자	120	10

SECTION 03 데이터의 변경을 위한 SQL문

데이터의 삭제 : DELETE

- 행 단위로 데이터 삭제하는 구문

```
DELETE FROM 테이블이름 WHERE 조건;
```

- WHERE절 생략되면 전체 데이터를 삭제함
- 테이블을 삭제하는 경우의 속도 비교
  - DML문인 DELETE는 트랜잭션 로그 기록 작업 때문에 삭제 느림
  - DDL문인 DROP과 TRUNCATE문은 트랜잭션 없어 빠름
    - 테이블 자체가 필요 없을 경우에는 DROP 으로 삭제
    - 테이블의 구조는 남겨놓고 싶다면 TRUNCATE로 삭제하는 것이 효율적

```
DELETE FROM testTbl4 where Fname='Aamer';
```

```
DELETE FROM testTbl4
where id in (
    select id
    from testTbl4
    where Fname='Aamer'
    limit 5
);
```

'Aamer' 중 상위 5건만 삭제

[실습 5] 대용량 테이블을 삭제하자

STEP 1

- 대용량 테이블을 세 개 생성하자. empolyees에서 약 30만 건이 있는 테이블을 복사해서 사용해보자

```
set schema 'employees';
create table bigTbl1 as (select * from employees);
create table bigTbl2 as (select * from employees);
create table bigTbl3 as (select * from employees);
```

STEP 2

pgadmin 쿼리 창에서, 먼저 DELETE, DROP, TRUNCATE문으로 세 테이블을 모두 삭제하고, 세 문 모두 테이블의 행을 삭제함

```
DELETE FROM bigTbl1;
DROP TABLE bigTbl2;
TRUNCATE TABLE bigTbl3;
```

STEP 3

출력값을 비교하면, DELETE만 시간이 오래걸리고 나머지는 짧은 시간이 걸린 것을 확인할 수 있다.

SECTION 03 데이터의 변경을 위한 SQL문

조건부 데이터 입력, 변경

◦ 기본 키가 중복된 데이터를 입력한 경우

- 오류로 입력불가

◦ 대용량 데이터 처리의 경우 에러 발생하지 않은 구문 실행

- ON CONFLICT (col_name) DO NOTHING
  - 에러 발생해도 다음 구문으로 넘어가게 처리
- ON CONFLICT (col_name) DO UPDATE SET
  - 지정한 열에서 데이터의 중복이 발생하면 데이터를 수정되도록 하는 구문도 활용 가능

[실습 6] 조건부 데이터 입력 및 변경

STEP 1

멤버테이블(memberTBL)을 정의하고 데이터를 입력하자.(CREATE TABLE..SELECT이용)

```
set schama 'shop';
drop table if exists memberTBL;
CREATE TABLE memberTBL as (SELECT userID, name, addr FROM usertbl LIMIT 3); --3건만 가져옴
ALTER TABLE memberTBL
    ADD constraint pk_memberTBL PRIMARY KEY(userID); --PK를 지장함
SELECT * FROM memberTBL;
```

STEP 2

여러건 입력 시 오류가 발생해도 나머지는 계속 입력

2-1 데이터를 추가로 2건 입력해 보자. 그런데 첫 번째 데이터에서 PK를 중복하였다.

```
INSERT INTO memberTBL VALUES('KKH', '강경호', '미국');
INSERT INTO memberTBL VALUES('SJH', '서장훈', '서울');
SELECT * FROM memberTBL;
```

조회해보면 데이터가 그대로 3건뿐이고, 첫 번째 오류 때문에 나머지 건도 입력이 되지 않았다.

2-2 ON CONFLICT (col_name) DO NOTHING 구문을 추가

```
insert into memberTBL values ('KKH','강경호','미국') ON CONFLICT (userID) DO NOTHING;
insert into memberTBL values ('SJH','서장훈','서울') ON CONFLICT (userID) DO NOTHING;
select * from memberTBL; --2건이 추가로 들어간것을 볼 수 있다. Pk가 중복이더라도 오류를 발생시키지 않으며, 무시하고 넘어간다
```

[실습 6] 조건부 데이터 입력 및 변경

STEP 3

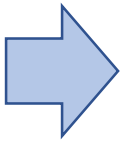
기본키가 중복되면 수정해보도록 해보자

```
INSERT INTO memberTBL VALUES('SJH', '서줌훈', '평양')
    ON CONFLICT(userID) DO UPDATE SET userid='SJH',name='성지현',addr='영국';
INSERT INTO memberTBL VALUES('DJM', '동짜몽', '일본')
    ON CONFLICT (userID) DO UPDATE SET name='동짜몽', addr='일본';
SELECT * FROM memberTBL;
```

첫 번째 행에서 SJH는 중복이 되었으므로 update문이 수행되었다. 그리고 두 번째 입력한 DJM은 없으므로 일반적인 insert문처럼 데이터가 입력되었다.

결국 ON DUPLICATE UPDATE는 PK가 중복되지 않으면 일반 INSERT처럼 되는 것이고, PK가 중복되면 그 뒤의 UPDATE문이 수행된다.

	userid [PK] character (8)	name character varying (10)	addr character (2)
1	LSG	이승기	서울
2	KBS	김범수	경남
3	KKH	김경호	전남
4	SJH	서장훈	서울



	userid [PK] character (8)	name character varying (10)	addr character (2)
1	LSG	이승기	서울
2	KBS	김범수	경남
3	KKH	김경호	전남
4	SJH	성지현	영국
5	DJM	동짜몽	일본