

MS SQL

목차

1. MS SQL 작업 환경 만들기 . . . 4p
2. SQL(Structured Query Language)이란? . . . 10p
3. 기본 용어 . . . 11p
4. 데이터 타입의 차이 . . . 13p
5. 제약 조건 . . . 14p
6. SQL의 종류 . . . 15p
7. 데이터베이스 만들기 . . . 16p

목차

8. 데이터 조회 . . . 25p
9. 함수 활용 . . . 34p
10. 테이블 조인(Join) . . . 37p
11. 서브 쿼리 . . . 41p
12. 데이터 조작과 트랜잭션 . . . 44p
13. 데이터베이스 다루기 . . . 47p
14. 데이터 무결성과 제약 조건 . . . 53p

MS SQL 작업 환경 만들기

- ▶ MS-SQL을 사용하기 위해서는 기본적으로 SQL Server를 설치하여야 한다
- ▶ SQL Server을 관리하기 위한 툴인 SSMS

MS SQL 작업 환경 만들기

▶ SQL Server 다운로드

- <https://www.microsoft.com/ko-kr/sql-server/sql-server-downloads>



Developer

SQL Server 2019 Developer는 비 프로덕션 환경에서 개발 및 테스트 데이터베이스로 사용하도록 라이선스가 제공되며 모든 기능을 갖춘 무료 버전입니다.

지금 다운로드 >

MS SQL 작업 환경 만들기

SQL Server 2019



-



Developer Edition

설치 유형 선택:

기본(B)

SQL Server 데이터베이스 엔진 기능을 기본 구성으로 설치하려면 기본 설치 유형을 선택하세요.

사용자 지정(C)

사용자 지정 설치 유형을 선택하여 SQL Server 설치 마법사의 단계를 진행하고 설치하려는 항목을 선택하세요. 이 설치 유형은 프로세스가 세부적이므로 기본 설치를 실행하는 것보다 오랜 시간이 소요됩니다.

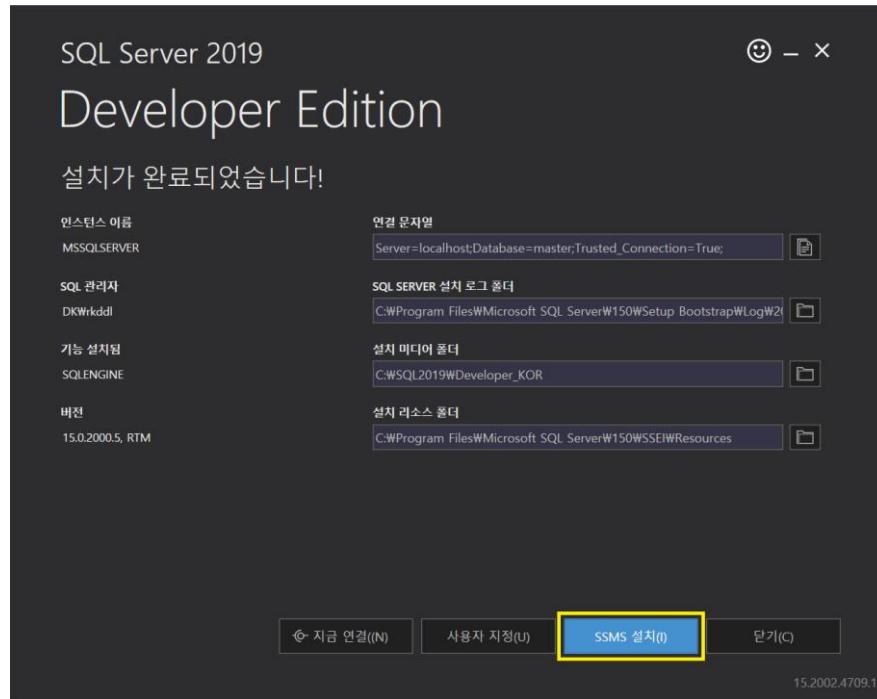
미디어 다운로드(D)

SQL Server 설치 프로그램 파일을 지금 다운로드하고 원하는 경우 컴퓨터에 나중에 설치하세요.

SQL Server에서는 제품 개선에 도움이 되도록 설치 환경에 대한 정보는 물론, 기타 사용량 현황 및 성능 데이터도 Microsoft에 전송합니다. 데이터 처리 및 개인 정보 관리 방법에 대해 자세히 알아보고 설치 후 이러한 정보를 수집하는 기능의 설정을 해제하려면 다음을 참조하세요.
[설명서](#)

15.2002.4709.1

MS SQL 작업 환경 만들기



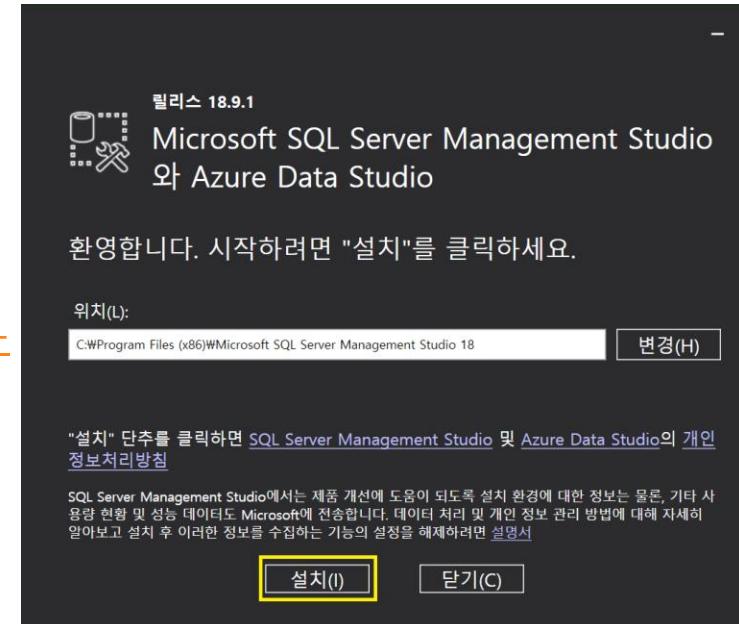
- ▶ SSMS 다운로드 주소
 - <https://docs.microsoft.com/ko-kr/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>

SSMS 다운로드



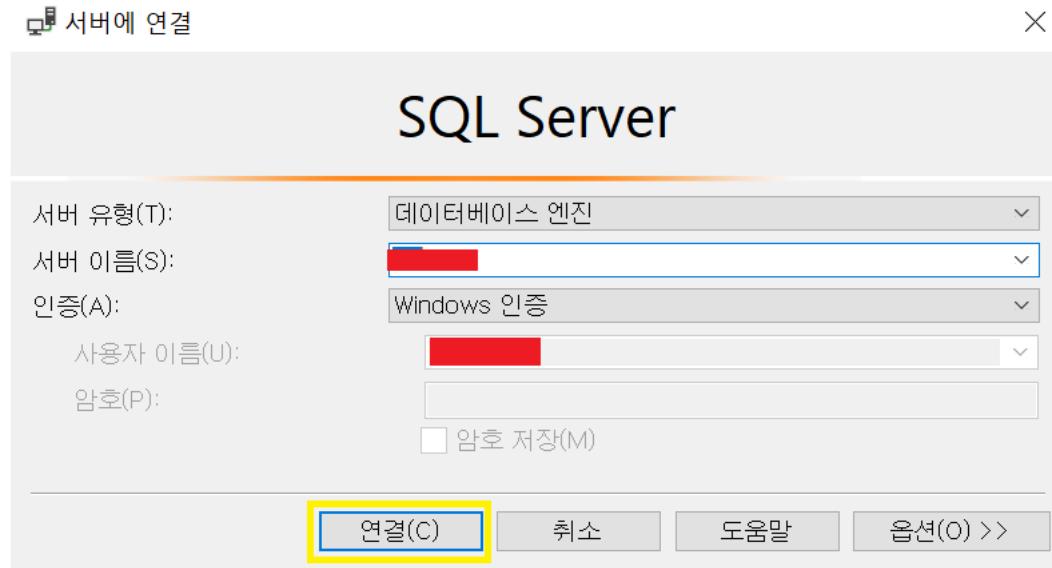
[SSMS\(SQL Server Management Studio\) 18.9.1 다운로드 ↗](#)

설치가 완료되면
바로 닫기를 눌러 나가시지 말고
[SSMS 설치]를 누르면 **이동**하는
이동되는 **사이트**에서 **SSMS**를 받아
설치해야 한다

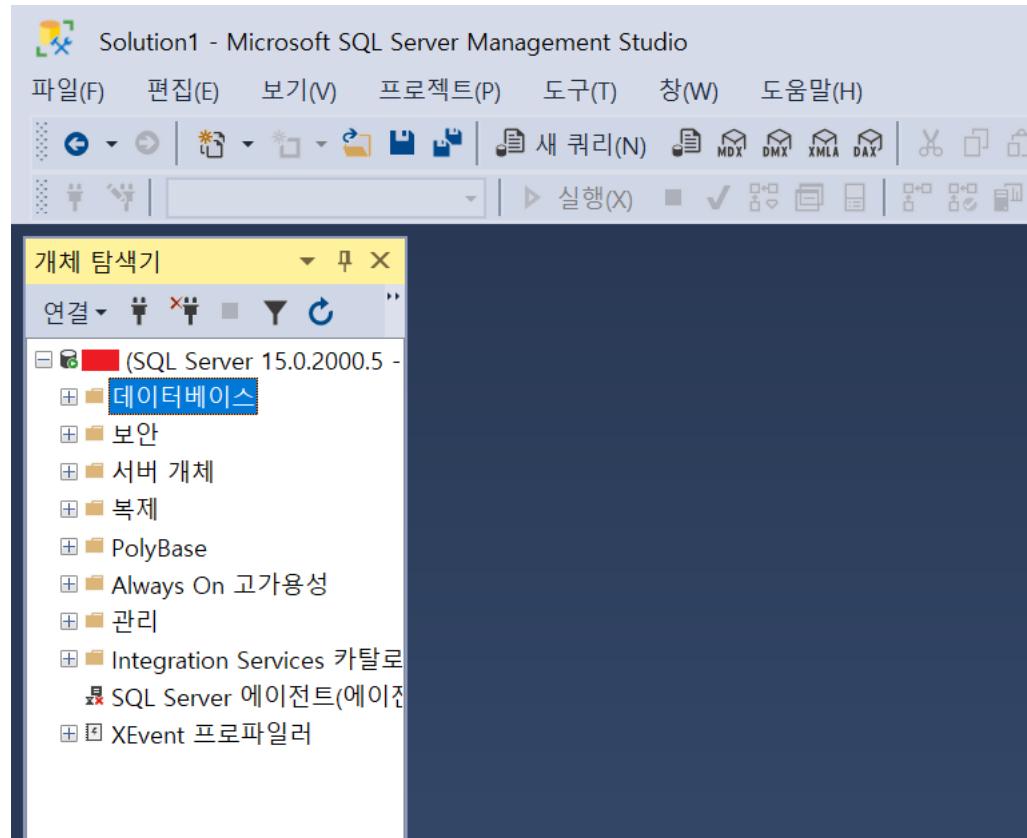


MS SQL 작업 환경 만들기

- ▶ Microsoft **SSMS**(SQL Server Management Studio)를 실행
 - 실행을 하게되면 기본적으로 서버 연결 창이 뜬다
 - 기본 인증은 [Windows 인증]으로 되어 있으며 아무나 접근이 가능하다
 - [SQL Server 인증]으로 만들게 되면 비밀번호를 지정하여 한정된 유저만 사용할 수 있게 한다



MS SQL 작업 환경 만들기



SQL(Structured Query Language)이란?

- ▶ 관계형 데이터베이스 관리 시스템(RDBMS)에서 자료를 관리 및 처리하기 위해 설계된 언어
 - 대소문자를 구별하지 않는다
 - 고유 값은 따옴표(' ')로 감싼다
 - 단일 주석은 "--"를 쓴다
- ▶ 데이터베이스
 - 유용한 데이터의 집합
 - 저장된 데이터의 정보 검색, 수정, 삭제 등이 용이하다
- ▶ 데이터베이스 관리 시스템(DBMS)
 - 방대한 양의 데이터를 편하게 정리하고 효율적으로 관리, 검색할 수 있는 환경을 제공해주는 시스템
 - 데이터를 공유하여 정보의 체계적인 활용을 가능하게 한다
 - Oracle, MS-SQL, MySQL, Informix, DB2...
- ▶ 관계형 데이터베이스(RDBMS)
 - 정보 저장을 위해 2차원의 테이블을 이용한다
 - 데이터 간의 상호 관계에서 개체간의 관계를 표현한 것

기본 용어

▶ 테이블(Table)

- 데이터를 저장하기 위한 **표 형태의 틀**

▶ 식별자

- 데이터베이스, 테이블, 칼럼 등과 같은 **개체들을 구분할 수 있는 이름**

▶ 칼럼(Column)

- 각 테이블의 열

▶ 로우(Row)

- 각 테이블의 행
- 테이블에서 **단일 구조의 데이터 항목**을 가리킨다

▶ 쿼리(Query)

- 데이터베이스의 정보 처리를 요청하는 질의문

▶ 서브 쿼리(Sub Query)

- **쿼리문 안에 포함된 또 다른 쿼리문**
- 내부 쿼리(Inner Query)라고도 부른다

기본 용어

- ▶ 시스템 데이터베이스
 - SQL Server가 **시스템을 운영, 관리하기 위해 필요한 데이터**와 추가적으로 **사용자 데이터베이스를 관리하기 위한 모든 데이터**를 담아 두는 곳
 - **master** : 시스템을 유지 및 관리하기 위해 요구되는 기본적인 내용을 저장
로그인 사용자 계정, 변경 가능한 각종 설정 값, 시스템 오류 메시지,
사용자 데이터베이스 관리
 - **model** : 새로운 **사용자 데이터베이스의 원본** 역할
사용자 데이터베이스를 새로 만들면 **model 데이터베이스를 복사하여 만듬**
사용자 데이터베이스는 model 데이터베이스보다 용량이 더 커야만 한다
 - **msdb** : 경고 및 작업을 예약하고 **운영자를 기록**하기 위해 SQL Server 에이전트에서 사용된다
 - **tempdb** : SQL Server에서 **임시적으로 사용하는 메모리** 역할
데이터베이스 작업 중, 정렬이나 조인 등등 **임시적으로 만들어 사용하는 것**
- ▶ **사용자 데이터베이스**
 - 개발자들이 만들어 사용하는 **일반적인 의미의 데이터베이스**

데이터 타입의 차이

▶ 정수

- SQL Server에서는 **나타내야 할 최대 자리수**의 크기와 **음수값의 사용 여부**에 따라 세가지 타입을 제공
- **tinyint** : 0 ~ 255
- **smallint** : -2^{15} (-32,768) ~ $2^{15} - 1$ (32,767)
- **int** : -2^{31} (-2,147,483,648) ~ $2^{31} - 1$ (2,147,483,647)
- **bigint** : -2^{63} (-9,223,372,036,854,775,808) ~ $2^{63} - 1$ (9,223,372,036,854,775,807)

▶ 날짜 및 시간

- “년/월/일”, “시/분/초”를 나타내기 위하여 사용
- **datetime** : 1753년 1월 1일 ~ **9999년 12월 31일까지**를 **3.33ms 간격**으로 표현
- **smalldatetime** : 1900년 1월 1일 ~ **2079년 6월 6일까지**를 **1분 간격**으로 표현

▶ 문자열

- 고정 문자열(char, nchar)과 가변 문자열(varchar, nvarchar)을 제공
- n이 붙어 있으면 **유니 코드** 문자, 없다면 **아스키 코드** 문자

제약 조건

- ▶ **제약 조건** : 칼럼(column)에 들어가는 값을 제한하여 **무결성(Integrity)** 조건을 만족하기 위한 표준 방법 중의 하나
- **NOT NULL** : NULL값을 저장하지 못하도록 한다
- **PRIMARY KEY(기본 키)** : 칼럼 이름 앞에 으로 확인 가능하며 **테이블의 식별자** 역할로 **테이블** 당 **하나만 설정 가능, 유일무이한 값**만을 저장하게 한다
- **UNIQUE KEY** : 한 테이블에 여러 개 설정이 가능, **유일무이한 값**만을 저장하게 한다
- **FOREIGN KEY(외래 키)** :
 - **두 테이블을 연결**하는데 사용
 - **외래 키가 포함된 테이블이 자식 테이블**이고 외래 키 값을 제공하는 테이블이 부모 테이블
 - 외래 키 값은 **NULL or 부모 테이블의 기본 키**

SQL의 종류

- ▶ SQL은 크게 세 종류로 분류 한다
 - 예제 이외의 명령어가 더 존재 한다

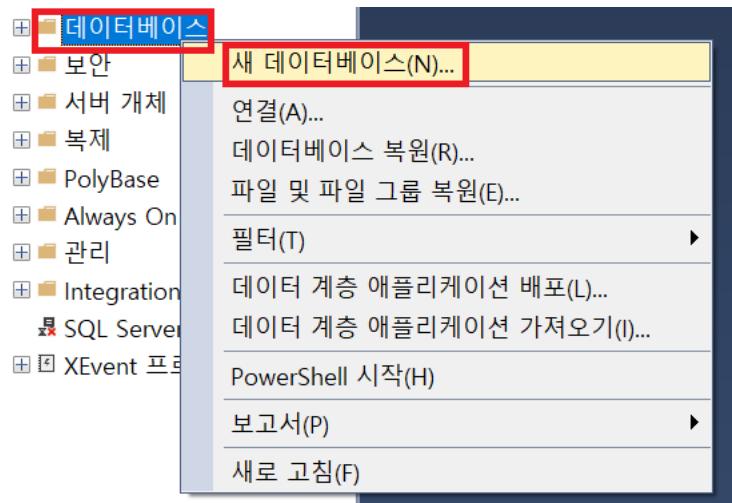
데이터 조작 언어 (DML, Data Manipulation Language)	명령어	설명
	SELECT	데이터베이스에 들어있는 데이터를 조회하거나 검색하기 위한 명령어
	INSERT	
	UPDATE	데이터베이스 테이블에 들어있는 데이터의 변형(추가, 수정, 삭제)을 하기위한 명령어
	DELETE	

데이터 정의 언어 (DDL, Data Definition Language)	명령어	설명
	CREATE	
	ALTER	
	DROP	테이블 같은 데이터 구조를 정의하는데 사용되는 명령어 (생성, 변경, 삭제, 이름변경, 모든 행 삭제)
	RENAME	
	TRUNCATE	

데이터 제어 언어 (DCL, Data Control Language)	명령어	설명
	GRANT	데이터베이스에 접근하고 객체들을 사용 하도록 권한을 주고 회수하는 명령어
	REVOKE	

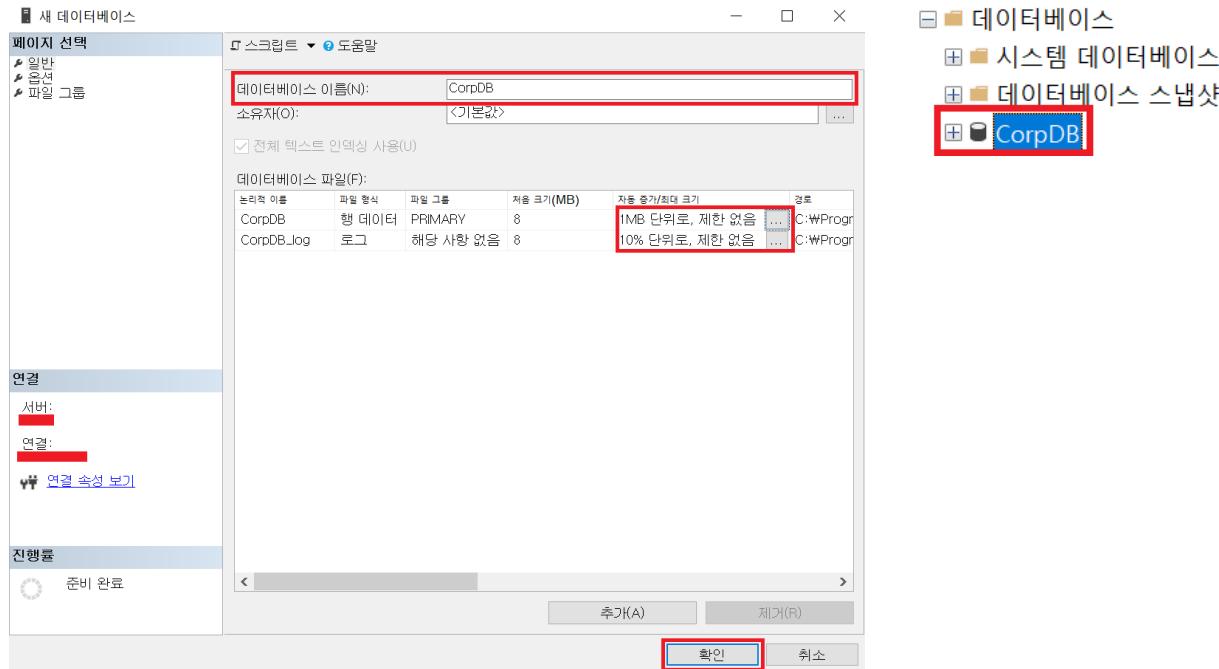
데이터베이스 만들기

- ▶ 데이터베이스 만드는 방법은 2 가지
 - 1. 화면 왼쪽의 [개체 탐색기]를 이용하여 DB생성 방법
 - 2. 쿼리문을 이용한 DB생성 방법
- ▶ [개체 탐색기]를 이용하여 DB 만들기
 - [개체 탐색기]의 [데이터베이스]를 마우스 오른쪽 클릭하여 [새 데이터베이스]를 선택



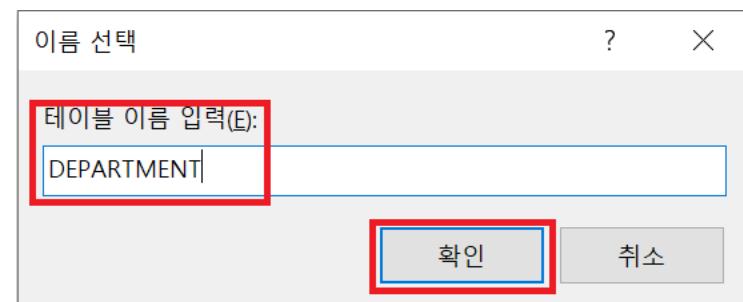
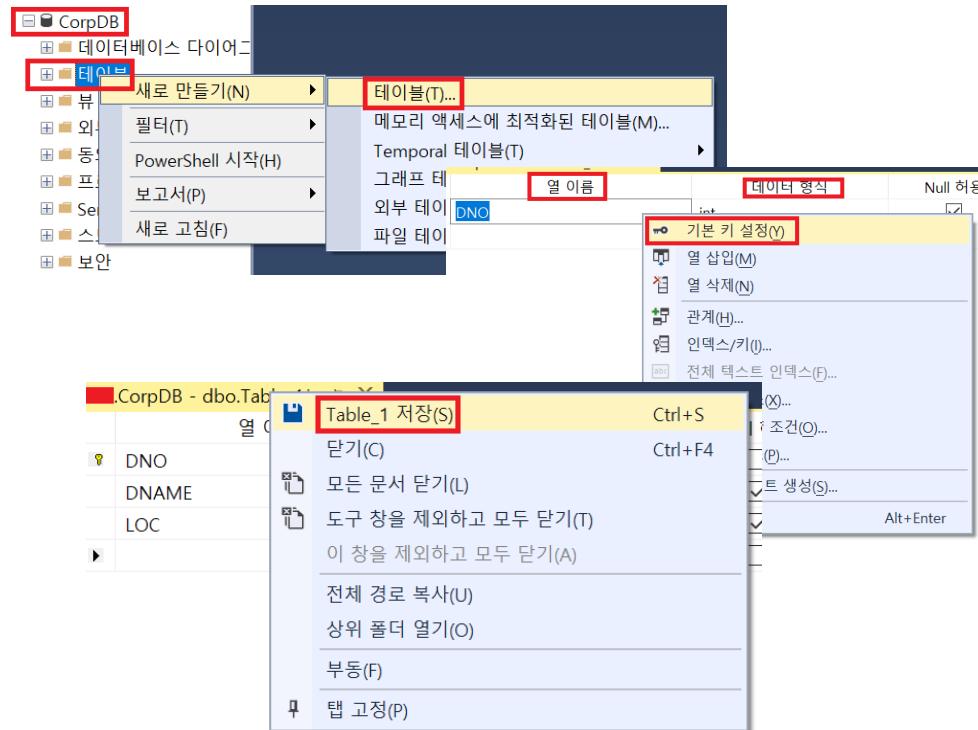
데이터베이스 만들기

- 원하는 DB 이름을 입력하고, 데이터와 로그의 저장 단위를 원하는 값으로 변경
- [확인] 버튼을 누르면 [개체 탐색기]에 새로 만든 DB가 추가 된다
- 보이지 않을 경우 [새로고침]을 하여 갱신



데이터베이스 만들기

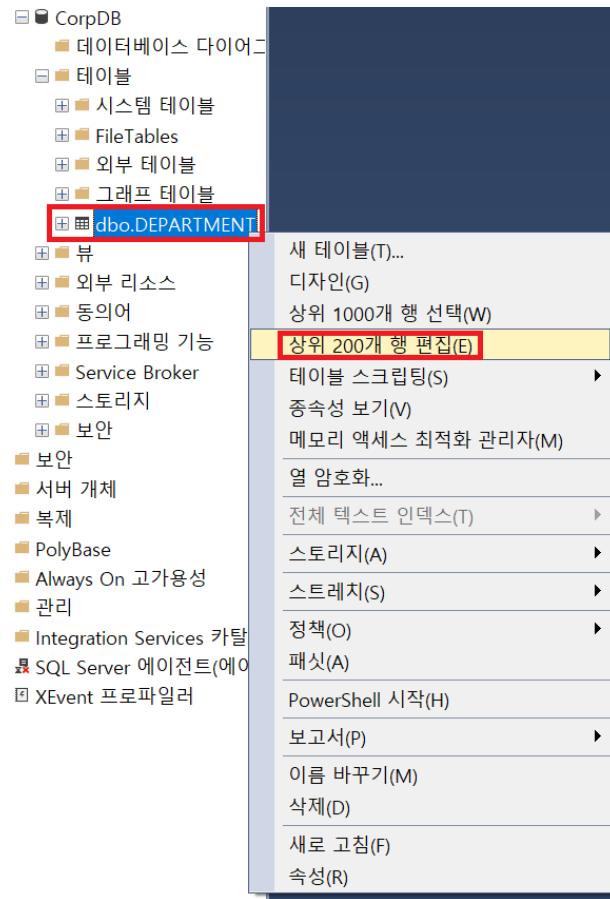
- 새로 만든 DB의 [테이블]에서 오른쪽 마우스 클릭을 하여 **새로운 테이블 추가**가 가능
- **테이블을 정의**하고 저장, **테이블 명**을 지정하고 **[확인]**을 누르면 테이블이 생성된다



데이터베이스 만들기

- 생성된 테이블이 보이지 않을 경우 [새로고침]을 한다
- 생성한 테이블을 마우스 오른쪽 클릭
- 상위 200개 행 편집을 선택
- 테이블 수정을 위한 표를 확인 가능

	DNO	DNAME	LOC
▶*	NULL	NULL	NULL



데이터베이스 만들기

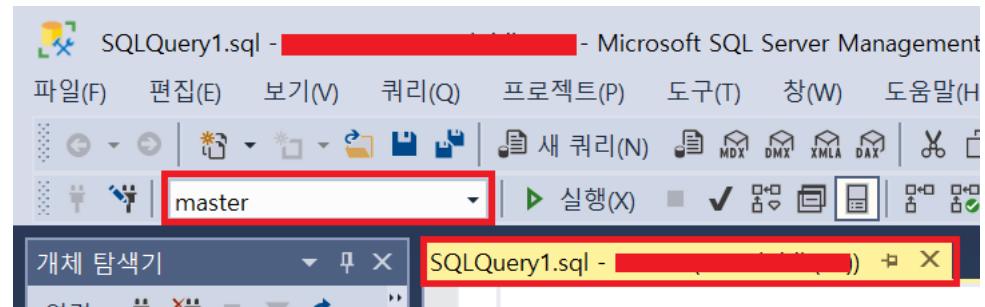
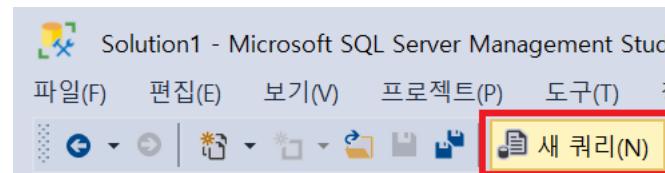
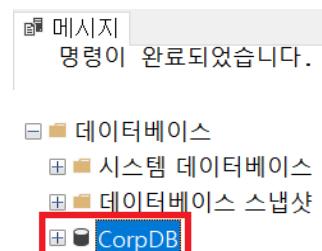
▶ 커리문을 이용하여 DB 만들기

- 쿼리문 작성 중, [개체 이름이 유효하지 않습니다]라는 문구가 나오면 **[Ctrl]+[Shift]+[R]**으로 **[로컬 캐시 새로고침]**을 하면 해결 된다
- [도구 메뉴]에서 **[새 쿼리]**를 선택
- 데이터베이스는 **[master]**로 설정되며 **쿼리문 작성을 위한 창이 생성**된다
- 데이터베이스가 **[master]**가 아닌 경우, **직접 설정** 또는 **USE master** 쿼리문 추가
- 형식 : **CREATE DATABASE** 데이터베이스명

```
USE master
```

```
CREATE DATABASE CorpDB
```

- 해당 쿼리를 사용하여 실행(F5)



데이터베이스 만들기

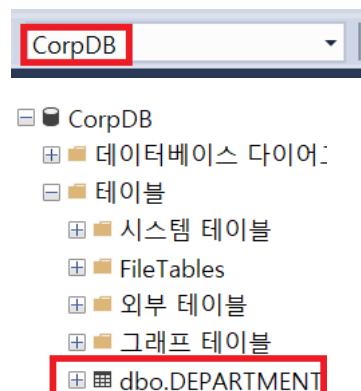
- 적용한 커리문은 **삭제 또는 주석 처리**를 한다
- 테이블을 만들 데이터베이스인 **[CorpDB]**로 변경 또는 **USE CorpDB** 쿼리문을 추가
- 형식 : **CREATE TABLE** 테이블명(칼럼명 데이터형 제약조건, ...)

USE CorpDB

```
CREATE TABLE DEPARTMENT(  
DNO int PRIMARY KEY, -- 부서번호 / 기본 키 설정.  
DNAME nvarchar(20), -- 부서명.  
LOC nchar(20) -- 지역명.  
)
```

- 해당 쿼리를 적용하여 실행(F5)

메시지
명령이 완료되었습니다.



데이터베이스 만들기

- 해당 테이블에 데이터를 추가할 수 있다
- 값을 넣을 칼럼과 순서 등을 변경하여 추가 가능
- 반드시 기술한 칼럼의 수와 순서에 맞게 칼럼에 알맞은 데이터를 넣어야 한다
- 모든 칼럼에 데이터를 입력하면 굳이 칼럼명을 기술하지 않아도 된다
- NULL을 허용하지 않는 제약 조건을 추가한 칼럼에는 NULL값을 넣을 수 없다
- 칼럼명과 데이터를 기술하지 않으면 암시적으로 NULL 값이 들어가게 된다
- 형식 : `INSERT INTO 테이블명(칼럼명1, 칼럼명2, ...) VALUES(칼럼값1, ...)`

```
INSERT INTO department(dno, dname, loc) VALUES(10, '경리부', '서울')
```

```
INSERT INTO department(dname, loc, dno) VALUES('인사부', '인천', 20)
```

```
INSERT INTO department(loc, dno, dname) VALUES('대전', 30, '영업부')
```

```
INSERT INTO department VALUES(40, '전산부', '부천')
```

데이터베이스 만들기

CorpDB

DEPARTMENT(부서) 테이블

칼럼 명	데이터 타입	크기	설명
DNO	int		부서 번호
DNAME	nvarchar	20	부서명
LOC	nchar	20	지역명

EMPLOYEE(사원) 테이블

칼럼 명	데이터 타입	크기	설명
ENO	int		사원 번호
ENAME	nvarchar	20	사원명
JOB	nvarchar	20	업무명
MANAGER	int		해당 사원의 상사 번호 (EMPLOYEE 테이블의 ENO와 연결됨)
HIREDATE	datetime		입사일
SALARY	int		급여
COMMISION	int		커미션
DNO	int		부서 번호 (DEPARTMENT 테이블의 DNO와 연결됨)

SALGRADE(급여) 테이블

칼럼 명	데이터 타입	크기	설명
GRADE	int		급여 등급
LOWSAL	int		급여 하한값
HIGHSAL	int		급여 상한값

데이터베이스 만들기

- ▶ [예제소스]에서 [CorpDB 만들기] 쿼리 파일을 이용하여 데이터베이스와 테이블을 만들고 데이터를 입력
 - 미리 데이터베이스와 테이블을 만들었다면
CorpDB 데이터베이스 생성, DEPARTMENT 테이블 생성 제외
 - 쿼리문을 따라 하셨다면
DEPARTMENT 테이블 데이터 추가는 제외

데이터 조회

- ▶ *(에스터리스크)
 - 테이블 내의 모든(all) 칼럼의 데이터를 선정할 때 사용
- ▶ USE 문
 - 형식 : USE 데이터베이스명
 - 사용할 데이터베이스를 지정
- ▶ SELECT 문
 - 형식 : SELECT 칼럼명1, 칼럼명2, ... FROM 테이블명
 - 형식 : SELECT * FROM 테이블명
 - 데이터를 조회하기 위하여 사용
 - 반드시 SELECT와 FROM이라는 2개의 키워드로 구성되어야 한다

데이터 조회

▶ 산술 연산자

- 다른 프로그래밍 언어와 같이 산술 연산자 사용이 가능하다

```
SELECT salary + commision FROM employee
```

```
SELECT salary - commision FROM employee
```

```
SELECT salary * 12 FROM employee
```

```
SELECT salary / 12 FROM employee
```

▶ NULL

- 0(zero)가 아니다
- 빈 공간이 아니다
- ? 또는 ∞ , 어떤 값인지는 몰라도 값이 존재하고 있다
- **연산, 비교가 불가능**
- 0 또는 다른 값으로 **변환**하기 위해 **isnull()** 함수를 사용

```
SELECT isnull(commision, 0) FROM employee
```

데이터 조회

▶ 칼럼 별칭 지정

- 산술 연산 등을 하게 되면 칼럼명이 **(열 이름 없음)**으로 표기되는 경우 또는 **칼럼명을 알아보기 쉬운 다른 이름으로 출력하고 싶을 경우, 원하는 이름으로 지정 가능하다**
- **AS 연산자**, **[]**, **' '**, **" "**로 별칭을 부여할 수 있다
- **AS 연산자는 생략이 가능하다**
- **[], ' ', " "**를 사용하면 **공백, 특수문자 등을 포함할 수 있다**

```
SELECT salary * 12 + isnull(commision, 0) AS 연봉 FROM employee
```

```
SELECT salary * 12 + isnull(commision, 0) 연봉 FROM employee
```

```
SELECT salary * 12 + isnull(commision, 0) [연 봉] FROM employee
```

▶ 중복 데이터 한 번씩만 출력하기

- **DISTINCT 키워드**를 사용하여 중복 데이터가 한 번씩만 출력되게 한다
- 형식 : **SELECT DISTINCT 칼럼명 FROM 테이블명**

```
SELECT DISTINCT dno FROM employee
```

데이터 조회

▶ 비교 연산자

- 다른 프로그래밍 언어와 같이 비교 연산자 사용이 가능하다
- 다른 프로그래밍 언어와 다르게, 같은 값 확인은 **= (equal)**을 사용한다
- 다르다(같지 않다)는 != 외에도 ◆도 사용한다

▶ WHERE 조건문

- 특정 조건에 맞는 데이터만을 출력하게 해준다
- 형식 : FROM 테이블명 WHERE 조건식

```
SELECT * FROM employee WHERE dno = 10
```

```
SELECT * FROM employee WHERE salary <> 300
```

```
SELECT * FROM employee WHERE salary > 300
```

```
SELECT * FROM employee WHERE salary < 300
```

```
SELECT * FROM employee WHERE salary >= 300
```

```
SELECT * FROM employee WHERE salary <= 300
```

데이터 조회

▶ IS NULL, IS NOT NULL 연산자

- 비교 연산자 사용이 불가능한 NULL 값을 위한 연산자
- 형식 : WHERE 칼럼명 IS NULL

```
SELECT * FROM employee WHERE commision IS NULL
```

```
SELECT * FROM employee WHERE commision IS NOT NULL
```

▶ 문자 데이터 조회

- 찾으려고 하는 문자열을 '' 사이에 작성한다
- 형식 : WHERE 칼럼명 연산자 '문자열'

```
SELECT * FROM employee WHERE ename = '조인성'
```

▶ 날짜 데이터 조회

- 문자 데이터 조회와 같이 시간 정보를 '' 사이에 작성한다

```
SELECT * FROM employee where HIREDATE > '2002'
```

```
SELECT * FROM employee where HIREDATE > '2002/01/01'
```

```
SELECT * FROM employee where HIREDATE > '2003-01-01'
```

데이터 조회

▶ 논리 연산자

- 두 가지 이상의 조건을 제시할 수 있다
- AND, OR, NOT 키워드를 사용한다
- 형식 : WHERE 조건식A AND 조건식B
- 형식 : WHERE NOT 조건식

```
SELECT * FROM employee where SALARY > 200 AND dno > 10
```

```
SELECT * FROM employee where SALARY > 200 OR dno > 10
```

```
SELECT * FROM employee where NOT SALARY > 200 AND NOT dno < 10
```

▶ BETWEEN AND 연산자

- 특정 범위 내의 값을 지닌 데이터를 출력할 수 있다
- 형식 : where 칼럼명 BETWEEN A AND B

```
SELECT * FROM employee where SALARY BETWEEN 200 AND 400
```

```
SELECT * FROM employee where SALARY NOT BETWEEN 200 AND 400
```

데이터 조회

▶ IN 연산자

- 임의의 값, 10 또는 30을 지닌 특정한 데이터를 찾기위해 **OR 연산자 대신 간단하게 사용**이 가능
- 형식 : **where 칼럼명 IN(A, B, ...)**

```
SELECT * FROM employee where dno IN(10, 30)
```

▶ LIKE 연산자와 와일드카드(_, %)

- 데이터의 일부만이 일치하여 조회할 경우 **LIKE 연산자와 와일드카드 패턴 조합**으로 사용
- **%** : 문자가 없거나, 하나 이상의 문자로 어떤 값이 와도 상관없다
- **_** : 하나의 문자로 어떤 값이든 상관없다
- 형식 : **WHERE 칼럼명 LIKE 패턴**

```
SELECT * FROM employee WHERE ename LIKE '이%'
```

```
SELECT * FROM employee WHERE ename LIKE '%성%'
```

```
SELECT * FROM employee WHERE ename LIKE '%성'
```

```
SELECT * FROM employee WHERE ename LIKE '_성%'
```

```
SELECT * FROM employee WHERE ename LIKE '__성%'
```

데이터 조회

▶ ORDER BY 정렬문

- 데이터 출력 순서를 입력된 순서가 아닌, **컬럼 값을 이용**하여 오름차순(ASC) 또는 내림차순(DESC)으로 정렬되어 출력할 수 있다
- 정렬 방식을 지정하지 않으면 **디폴트로 오름차순(ASC)** 정렬이 된다
- 형식 : 테이블명 ORDER BY 칼럼명1 정렬 방식, ...

```
SELECT * FROM employee ORDER BY salary
```

```
SELECT * FROM employee ORDER BY salary ASC
```

```
SELECT * FROM employee ORDER BY salary DESC
```

```
SELECT * FROM employee ORDER BY salary DESC, ename ASC
```

	ASC(오름차순)	DESC(내림차순)
숫자	작은 값부터 정렬	큰 값부터 정렬
문자	사전 순서로 정렬	사전 반대 순서로 정렬
날짜	빠른 날짜 순서로 정렬	늦은 날짜 순서로 정렬
NULL	가장 마지막에 나온다	가장 먼저 나온다

데이터 조회

▶ TOP 연산자

- 데이터를 원하는 개수(n)의 만큼 출력 할 수 있다
- **WITH TIES** 문을 붙이면 같은 값을 가진 데이터를 같이 출력한다
- **PERCENT** 문을 붙이면 수가 아닌 퍼센트(%)만큼 출력된다
- 형식 : **SELECT TOP(n) WITH TIES * FROM 테이블명 ...**

```
SELECT TOP(5) * FROM employee ORDER BY salary DESC
```

```
SELECT TOP(5) WITH TIES * FROM employee ORDER BY salary DESC
```

```
SELECT TOP(10) PERCENT * FROM employee ORDER BY salary DESC
```

함수 활용

▶ 집계 함수

- 전체 데이터를 **그룹별로 구분**하여 **통계적인 결과**를 구할 수 있다
- **COUNT** 함수는 **NULL** 값을 세지 않는다
- 형식 : **SELECT 함수(칼럼명) FROM 테이블명**

SELECT SUM(salary) [급여 총액] FROM employee

SELECT AVG(salary) [평균 급여] FROM employee

SELECT MAX(salary) [최대 급여], MIN(salary) [최소 급여] FROM employee

SELECT COUNT(commision) [커미션 받는 사원 수] FROM employee

SELECT COUNT(DISTINCT job) [업무 수] FROM employee

구분	설명
SUM	그룹의 누적 합계를 반환.
AVG	그룹의 평균을 반환.
COUNT	그룹의 총 개수를 반환.
COUNT_BIG	그룹의 총 개수를 반환. 단, 결과값이 bigint 형이다.
MAX	그룹의 최대값을 반환.
MIN	그룹의 최소값을 반환.
STDDEV	그룹의 표준편차를 반환.

함수 활용

▶ GROUP BY 문

- 칼럼 값이 같은 데이터를 그룹화한다
- 형식 : SELECT 칼럼명1, ... FROM 테이블명 WHERE 조건 GROUP BY 칼럼명

SELECT dno, AVG(salary) [평균 급여] FROM employee GROUP BY dno

SELECT dno, MAX(salary) [최대 급여], MIN(salary) [최소 급여] FROM employee
GROUP BY dno

SELECT dno, COUNT(*) [부서별 사원 수], COUNT(commision) [커미션 받는 사원 수]
FROM employee GROUP BY dno

▶ HAVING 문

- 그룹화 된 데이터에 조건을 건다
- 형식 : GROUP BY 칼럼명 HAVING 조건식

SELECT dno, AVG(salary) [평균 급여] FROM employee GROUP BY dno HAVING
AVG(salary) >= 500

SELECT dno, MAX(salary) [최대 급여], MIN(salary) [최소 급여] FROM employee
GROUP BY dno HAVING MAX(salary) > 500

함수 활용

▶ ROLLUP 문

- 그룹별 중간 합산 데이터와 전체 합산 데이터를 출력
- 형식 : **SELECT 칼럼명1, …, 함수 FROM 테이블명 GROUP BY 칼럼명1, … WITH ROLLUP**

SELECT dno, job, SUM(salary) [급여 총액] FROM employee GROUP BY dno, job WITH ROLLUP

▶ CASE 문

- 여러 경우 중 하나를 선택하는 함수, 프로그래밍 언어의 **if else**와 유사한 구조
- 형식 : **SELECT 칼럼1, …, CASE WHEN 조건1 THEN 결과1 … ELSE 결과n END FROM 테이블명**

SELECT ename, dno,

CASE

WHEN dno = 10 THEN '경리부'

WHEN dno = 20 THEN '인사부'

WHEN dno = 30 THEN '영업부'

WHEN dno = 40 THEN '전산부'

END [부서명]

FROM employee

테이블 조인(Join)

- ▶ 테이블 조인이란?
 - 한 개 이상의 테이블을 결합하여 데이터를 조회하기 위한 방법
 - 관계형 데이터베이스에서는 테이블 간의 관계가 중요하기 때문에 하나 이상의 테이블이 빈번히 결합되어 사용된다

▶ 조인의 종류

구분	설명
Equi Join	동일 칼럼을 기준으로 조인한다
Non-Equi Join	동일 칼럼이 없이 다른 조건을 사용하여 조인한다
Self Join	한 테이블 내에서 조인한다
Outer Join	조인 조건에 만족하지 않는 행(Row)도 나타난다

▶ 칼럼명의 모호성

- 두 개 이상의 테이블에 동일한 이름의 칼럼을 사용하면 어느 테이블 소속인지 불분명하기 때문에 칼럼명 앞에 테이블명을 명시하여 사용한다
- 테이블명과 칼럼명은 점(.)으로 구분
- 형식 : 테이블명.칼럼명

SELECT employee.dno FROM employee

테이블 조인(Join)

▶ 테이블에 별칭 부여

- 두 개 이상의 테이블을 조인할 경우 **칼럼 앞에 테이블 이름을 기술하는 일이 잦다.** 그런데 **테이블명이 길면 테이블명을 매번 붙이기가 번거로워지기 때문에 테이블에 간단한 별칭을 부여하여 사용한다**
- 형식 : **FROM 테이블명1 별칭1, ...**

`SELECT e.dno FROM employee e`

▶ Equi Join

▶ 가장 많이 사용되는 조인 방법

- 조인 대상이 되는 테이블에서 **공통적으로 존재하는 칼럼의 값이 일치되는 행을 연결하여 결과를 생성**
- 테이블 간의 조인에 **콤마(,) 대신 INNER JOIN으로 사용 가능**하며 **INNER는 생략이 가능, 조건식은 WHERE 대신에 ON을 사용**한다
- 형식 : **SELECT 칼럼명1, ... FROM 테이블명1, ... WHERE 조건식**
- 형식 : **SELECT 칼럼명1, ... FROM 테이블명1 INNER JOIN ... ON 조건식**

`SELECT eno, ename FROM employee, department WHERE employee.dno = department.dno`

`SELECT e.eno, e.ename, e.dno FROM employee e, department d WHERE e.dno = d.dno AND e.dno = 10`

`SELECT e.eno, e.ename, d.dno, d.dname FROM employee e JOIN department d ON e.dno = d.dno AND e.dno = 10`

테이블 조인(Join)

▶ Non-Equi Join

- 조인 대상이 되는 테이블에서 **공통적으로 존재하는 칼럼의 값이 없이** 조건을 이용하여 결과를 생성
- 테이블 간의 조인에 **콤마(,) 대신 INNER JOIN으로 사용 가능**하며 **INNER는 생략이 가능, 조건식은 WHERE 대신에 ON 을 사용한다**
- 형식 : **SELECT 칼럼명1, … FROM 테이블명1, … WHERE 조건식**
- 형식 : **SELECT 칼럼명1, … FROM 테이블명1 INNER JOIN … ON 조건식**

```
SELECT e.ename, e.salary, s.grade FROM employee e, salgrade s WHERE e.salary BETWEEN s.lowsal AND s.highsal
```

```
SELECT e.ename, e.salary, s.grade FROM employee e JOIN salgrade s ON e.salary BETWEEN s.lowsal AND s.highsal
```

▶ Self Join

- **하나의 테이블 내에서 칼럼을 연결**하여 결과를 생성
- 테이블 간의 조인에 **콤마(,) 대신 INNER JOIN으로 사용 가능**하며 **INNER는 생략이 가능, 조건식은 WHERE 대신에 ON 을 사용한다**
- 형식 : **SELECT 별칭1.칼럼, 별칭2.칼럼, … FROM 테이블A 별칭1, 테이블A 별칭2, … WHERE 조건식**
- 형식 : **SELECT 별칭1.칼럼, 별칭2.칼럼, … FROM 테이블A 별칭1 INNER JOIN 테이블A 별칭2, … ON 조건식**

```
SELECT member.ename [사원이름], manager.ename [직속상관이름] FROM employee member, employee manager  
WHERE member.manager = manager.eno
```

```
SELECT member.ename [사원이름], manager.ename [직속상관이름] FROM employee member JOIN employee manager  
ON member.manager = manager.eno
```

테이블 조인(Join)

▶ Outer Join

- 조인 조건에 맞지 않는 데이터도 출력되게 하여 준다
- Self Join의 예제에서 처럼 조인 조건에 맞지 않는 [직속상관이름]의 데이터도 출력되도록 하기 위해 사용
- OUTER JOIN 키워드 앞에 LEFT, RIGHT, FULL 키워드를 붙여 사용한다
- LEFT : 왼쪽에 표기된 테이블 데이터를 기준으로 조인을 수행
- RIGHT : 오른쪽에 표기된 테이블 데이터를 기준으로 조인을 수행
- FULL : 양쪽 테이블 데이터의 모든 값을 읽어 조인을 수행
- 형식 : **SELECT 칼럼명1, … FROM 테이블명1 LEFT 또는 RIGHT 또는 FULL INNER JOIN 테이블명2 ON 조건식**

```
SELECT member.ename [사원이름], manager.ename [직속상관이름] FROM employee  
member LEFT OUTER JOIN employee manager ON member.manager = manager.eno
```

서브 쿼리

- ▶ 서브 쿼리의 규칙
 - 서브 쿼리 안에 서브쿼리가 들어갈 수 있다. 이것을 **네스팅(nesting)**이라고 하며 **메모리가 허용하는 한 무제한으로 중첩이 가능**하다
 - 다중 행(Row) 연산자를 사용하지 않은 **일반적인** 서브 쿼리에서는 **하나의 레코드 값만 리턴**해야 하기 때문에 대부분의 경우 서브 쿼리에 **GROUP BY, HAVING** 문을 **사용할 수 없다**
 - **ORDER BY** 문은 **TOP** 연산자와 함께 있을 때만 사용 가능
 - **SELECT 하지 않은 칼럼은 메인 쿼리에서 사용 할 수 없다**
 - 형식 : **SELECT 칼럼 리스트 FROM 테이블 리스트 WHERE 칼럼 = /*메인 쿼리*/ (SELECT 결과값 FROM ... WHERE ...) /*서브 쿼리*/**
- ▶ 단일 행(Row) 서브 쿼리
 - 서브 쿼리에서 **하나의 결과 값만**을 받아 처리한다

```
SELECT dname FROM department WHERE dno = ( SELECT dno FROM employee WHERE ename = '조인성' )
```

```
SELECT ename, salary FROM employee WHERE salary > ( SELECT AVG(salary) FROM employee )
```

서브 쿼리

▶ 다중 행(Row) 서브 쿼리

- 서브 쿼리에서 결과 값이 하나 이상일 경우 **다중 행(Row) 연산자**를 사용하여 서브 쿼리를 만든다

▶ 다중 행(Row) 연산자

종류	의미
IN	메인 쿼리의 비교 조건('=' 연산자로 비교할 경우)이 서브 쿼리의 결과 중에서 하나라도 일치하면 true
EXISTS	메인 쿼리의 비교 조건이 서브 쿼리의 결과 중에서 만족하는 값이 하나라도 존재하면 true
ANY, SOME	메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 하나 이상이 일치하면 true
ALL	메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 모든 값이 일치하면 true

▶ IN 연산자

- 형식 : **SELECT ... FROM ... WHERE 칼럼명 IN (서브 쿼리)**

```
SELECT ename, salary, dno FROM employee WHERE dno IN (SELECT DISTINCT dno FROM employee WHERE salary > 500)
```

서브 쿼리

▶ EXISTS 연산자

- 서브 쿼리 사용 시 IN보다 EXISTS가 훨씬 빠르다

- 형식 : SELECT ... FROM ... WHERE EXISTS (서브 쿼리)

```
SELECT * FROM department WHERE EXISTS ( SELECT * FROM employee WHERE dno = 10 )
```

```
SELECT * FROM department d WHERE EXISTS ( SELECT * FROM employee e WHERE e.dno = d.dno )
```

▶ ANY, SOME 연산자

- 형식 : SELECT ... FROM ... WHERE 칼럼명 비교연산자 ANY 또는 SOME (서브 쿼리)

```
SELECT ename, salary FROM employee WHERE salary > ANY ( SELECT salary FROM employee WHERE dno = 30 )
```

▶ ALL 연산자

- 형식 : SELECT ... FROM ... WHERE 칼럼명 비교연산자 ALL (서브 쿼리)

```
SELECT ename, salary FROM employee WHERE salary > ALL( SELECT salary FROM employee WHERE dno = 30 )
```

데이터 조작과 트랜잭션

▶ SELECT INTO 문

- 어느 테이블의 **데이터를 새로운 테이블에 복사**한다
- 형식 : `SELECT 복사할컬럼명, ... INTO 새테이블명 FROM 대상테이블명`

```
SELECT * INTO employee_cpy FROM employee
```

```
SELECT * INTO department_cpy FROM department
```

▶ UPDATE 문

- 테이블에 **저장된 데이터를 수정**한다
- 형식 : `UPDATE 테이블명 SET 칼럼명 = 값, ... WHERE 조건식`

```
UPDATE employee_cpy SET salary *= 1.1, hiredate = GETDATE() WHERE dno = 10
```

```
UPDATE department_cpy SET loc = ( SELECT loc FROM department_cpy WHERE dno = 40 ) WHERE dno = 20
```

데이터 조작과 트랜잭션

▶ DELETE 문

- 테이블에 저장된 데이터를 삭제한다
- 형식 : `DELETE FROM 테이블명 WHERE 조건`

`DELETE FROM employee_cpy WHERE dno = 30`

`DELETE FROM employee_cpy WHERE dno = (SELECT dno FROM department WHERE dname = '전산부')`

▶ 트랜잭션(Transaction)

- 데이터 처리의 한 단위

- SQL Server에서 발생하는 여러 개의 SQL 명령문들을 하나의 논리적인 작업단위로 처리
- **하나의 트랜잭션은 ALL OR Nothing 방식**, 여러 개의 명령어의 집합이 정상적으로 처리되면 정상 종료하도록 하고 **하나의 명령어라도 잘못되면 전체를 취소**한다
- 데이터를 조작하는 명령어인 DML(Data Manipulation Language)은 실행과 동시에 트랜잭션이 진행된다

데이터 조작과 트랜잭션

▶ ROLLBACK과 COMMIT 명령어

- 데이터의 무결성 보장
- 영구적인 변경 전에 데이터의 변경 사항을 확인 가능
- 논리적으로 연관된 작업을 그룹화할 수 있다
- ROLLBACK : 작업 중 문제가 발생하여 트랜잭션 처리 과정 중 발생한 변경 사항을 취소하는 명령어, 작업 내용을 취소하고 BEGIN TRAN 직전의 상태로 복구
- COMMIT : 모든 작업을 정상적으로 처리하겠다고 확정하는 명령어, 작업 내용을 실제 데이터베이스에 저장, ROLLBACK을 하여도 이전 상태로 복구 할 수 없다
- 형식 : BEGIN TRAN DML쿼리 [COMMIT 또는 ROLLBACK TRAN]

BEGIN TRAN

DELETE FROM department_cpy

SELECT * FROM department_cpy

ROLLBACK TRAN

SELECT * FROM department_cpy

데이터베이스 다루기

▶ 데이터베이스 생성

- 새 데이터베이스를 만든다
- [해당 데이터베이스->속성->파일]로 정보를 확인할 수 있다
- 형식 : **CREATE DATABASE** 데이터베이스명
- 형식 :

CREATE DATABASE 데이터베이스명

ON

PRIMARY (NAME = 파일명 ,

FILENAME = '파일경로\파일명.mdf' ,

SIZE = 기본데이터크기MB ,

MAXSIZE = 최대데이터크기 ,

FILEGROWTH = 자동증가데이터크기)

LOG ON (NAME = 파일명_LOG ,

FILENAME = '파일 경로\파일명.ldf' ,

SIZE = 기본데이터크기MB ,

MAXSIZE = 최대데이터크기 ,

FILEGROWTH = 자동증가데이터크기)



데이터베이스 다루기

```
CREATE DATABASE Test01
ON
PRIMARY-- PRIMARY 그룹에서 데이터 파일 생성.
-----
-- 데이터 파일 설정 --
-----
( NAME = Test01,-- 논리적 이름.
FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\test01.mdf', -- 경로 및 파일명.
SIZE = 100MB,-- 처음 크기.
MAXSIZE = 200,-- 최대 크기.
FILEGROWTH = 20 )-- 자동 증가 크기.
-----
-- 트랜잭션 로그 파일 설정 --
-----
LOG ON
( NAME = Test01_LOG,
FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\test.ldf', -- 경로 및 파일명.
SIZE = 20MB,-- 처음 크기.
MAXSIZE = 50,-- 최대 크기.
FILEGROWTH = 10% )-- 자동 증가 %
```

데이터베이스 다루기

▶ 데이터베이스 수정

- **ALTER** 문을 사용하여 데이터베이스를 수정할 수 있다
- 데이터베이스 이름 변경
- 데이터, 로그 파일 수정(MODIFY), 추가(ADD) 및 삭제(REMOVE)
- 형식 :

ALTER DATABASE 데이터베이스명

MODIFY FILE

(**NAME** = 파일명,
SIZE = 기본데이터크기,
MAXSIZE = 최대데이터크기,
FILEGROWTH = 자동증가데이터크기)

ALTER DATABASE 데이터베이스명

MODIFY NAME = 변경할데이터베이스명

ALTER DATABASE 데이터베이스명

ADD FILE

(**NAME** = 파일명,
FILENAME = ‘파일경로\파일명.mdf’,
SIZE = 기본데이터크기,
MAXSIZE = 최대데이터크기,
FILEGROWTH = 자동증가데이터크기)

ALTER DATABASE 데이터베이스명

REMOVE FILE 삭제할파일명

데이터베이스 다루기

▶ 데이터베이스 삭제

- 데이터베이스의 **데이터, 로그 파일이 전부 삭제** 된다
- 형식 : **DROP DATABASE** 데이터베이스명

▶ 테이블 생성

- **IDENTITY(초기값, 증가값)**를 이용하여 해당 칼럼에 **자동으로** 연속적인 숫자, 즉 **일련 번호**가 들어가게 하여 항상 **유일한 값**을 만들기에 주로 **PRIMARY KEY(기본 키)**로 많이 사용된다
- 형식 : **CREATE TABLE** 테이블명(칼럼명 데이터형 제약조건, ...)

```
CREATE TABLE memberTest
(
    id int PRIMARY KEY IDENTITY(1, 1),
    name varchar(20)
)
```

GO

```
INSERT memberTest VALUES('김나리')
INSERT memberTest VALUES('이백합')
INSERT memberTest VALUES('김장미')
```

데이터베이스 다루기

▶ 테이블 구조 변경

➤ ADD : 새로운 칼럼 추가

- 새로운 칼럼은 **기존 테이블의 맨 마지막에 추가**되며, 원하는 위치에 넣을 수 없다
- **DEFAULT 제약 조건이 없다면** 이미 추가된 row(행)이 있을 경우 **추가한 칼럼이 적용되고 NULL값이 입력된다**
- 형식 : **ALTER TABLE 테이블명 ADD 칼럼명 데이터형 제약조건**

ALTER TABLE memberTest ADD [date] DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP

➤ ALTER COLUMN : 기존 칼럼 수정

- **기존 칼럼의 데이터 형, 크기, 기본 값 변경이 가능하다**
- 이미 데이터가 있을 경우 자유롭게 변경이 힘들며, CHAR와 VARCHAR사이의 타입 변경이 유일하게 가능하다
- 이미 있는 데이터 크기보다 작게는 변경할 수 없다
- 형식 : **ALTER TABLE 테이블명 ALTER COLUMN 칼럼명 데이터형 및 크기**

ALTER TABLE memberTest ALTER COLUMN name char(10)

➤ DROP COLUMN : 기존 칼럼 삭제

- 제약 조건이 있을 경우 먼저 제약조건을 삭제 해야 한다
- 형식 : **ALTER TABLE 테이블명 DROP COLUMN 칼럼명**

해당 칼럼의 제약 조건 삭제 (제약 조건 삭제를 위한 값이 달라, 예제 확인 불가)

ALTER TABLE memberTest

DROP COLUMN date

데이터베이스 다루기

▶ 테이블의 모든 Row(행) 제거

- 해당 테이블의 모든 데이터를 제거한다
- 형식 : **TRUNCATE TABLE 테이블명**
- **TRUNCATE TABLE memberTest**

▶ 테이블 삭제

- 테이블을 제거하며 저장되어 있던 데이터도 같이 제거된다
- 형식 : **DROP TABLE 테이블명**

DROP TABLE memberTest

데이터 무결성과 제약 조건

- ▶ 데이터 무결성 제약 조건(Data Integrity Constraint Rule)
 - 테이블에 부적절한 자료가 입력되는 것을 방지하기 위하여 테이블을 생성할 때 각 칼럼에 대해서 정의하는 여러 가지 규칙

제약 조건	역할
NOT NULL	NULL을 허용하지 않는다
UNIQUE	중복된 값을 허용하지 않는다 항상 유일한 값을 갖도록 한다
PRIMARY KEY	NULL을 허용하지 않고 중복된 값을 허용하지 않는다 NOT NULL 조건과 UNIQUE 조건을 결합한 형태
FOREIGN KEY	참조되는 테이블의 칼럼 값이 존재하면 허용한다
CHECK	저장 가능한 데이터 값의 범위나 조건을 지정하여 설정한 값만을 허용한다

▶ CONSTRAINT 키워드

- 제약 조건에 이름을 설정할 수 있다
- 형식 : ...CONSTRAINT 제약조건명 제약조건, ...

▶ DEFAULT 제약 조건

- 데이터 추가할 경우 해당 칼럼에 아무런 값을 입력하지 않았을 때 디폴트 값이 입력된다
- 형식 : ... 제약조건 DEFAULT 초기값

데이터 무결성과 제약 조건

- ▶ 테이블 레벨 제약 조건 지정
 - 칼럼을 모두 정의하고 테이블 정의를 마무리 짓기 전에 따로, 생성된 **칼럼들의 제약 조건을 한꺼번에 지정**하는 방식
 - **DEFAULT, (NOT) NULL** 제약 조건은 **테이블 레벨로 지정할 수 없다**
 - **칼럼 레벨 제약 조건** : **칼럼 정의에 제약 조건을 포함**하여 지정, 지금까지의 제약 조건 지정 방식
 - 형식 : **CREATE TABLE 테이블명(…, CONSTRAINT 제약조건명 제약조건(칼럼명))**
- ▶ 테이블 레벨 지정을 하는 이유
 - 복합 키를 지정할 경우
 - **2개 이상의 칼럼을 기본 키로 설정**하는 것을 복합 키라고 한다
 - 복합 키 형태로 제약 조건을 지정할 경우 **칼럼 레벨로는 불가능**하며 **반드시 테이블 레벨 방식을 사용**해야 한다
 - **ALTER TABLE 제약 조건을 추가할 때**
 - 이미 생성하여 사용하고 있는 테이블에 제약 조건을 추가하려면 테이블 레벨 방식으로 제약 조건으로 지정해야 한다

데이터 무결성과 제약 조건

▶ 제약 조건 변경

- 제약 조건 추가 및 삭제가 가능하다
- 형식 : **ALTER TABLE** 테이블명 **ADD CONSTRAINT** 제약조건명 **제약조건(칼럼명)**
- 형식 : **ALTER TABLE** 테이블명 **DROP CONSTRAINT** 제약조건명

▶ FOREIGN KEY(외래 키) 제약 조건

- 참조 하려는 테이블(**부모 테이블**)의 칼럼은 **PRIMARY KEY** 또는 **UNIQUE** 이어야만 한다
- **FOREIGN KEY**는 참조하려는 테이블(**자식 테이블**)에 정의한다
- 형식 : **CREATE TABLE** 테이블명(... 칼럼명 데이터형 **CONSTRAINT** 제약조건명
REFERENCES 부모테이블명(부모테이블의칼럼명), ...)
- 형식 : **CREATE TABLE** 테이블명(..., **CONSTRAINT** 제약조건명 **FOREIGN KEY(칼럼명)**
REFERENCES 부모테이블명(부모테이블의칼럼명))

▶ CHECK 제약 조건

- 설정된 이외의 값이 들어오면 오류 메시지와 함께 명령이 수행되지 못하게 한다
- 형식 : **CHECK(칼럼명 조건식)**

연습 문제

▶ 아래 표를 참고하여
새로운 데이터베이스와 테이블을 만들어 봅시다

customer

속성명	칼럼명	데이터 타입	크기	NULL 허용	키	디폴트 값	비고
아이디	id	varchar	20	X	PK		
비밀번호	pwd	varchar	20	X			
이름	name	varchar	20	X			
전화 번호	phone	varchar	11	X			
등록 일자	register	datetime		O		CURRENT_TIMESTAMP	
주소	address	varchar	100	X			

products

속성명	칼럼명	데이터 타입	크기	NULL 허용	키	디폴트 값	비고
상품 코드	pcode	varchar	20	X	PK		LIKE('P%')
상품명	pname	varchar	20	O			
상품 가격	price	int		O		0	

orders

속성명	칼럼명	데이터 타입	크기	NULL 허용	키	디폴트 값	비고
주문 번호	oseq	int		X	PK		IDENTITY
수량	quantity	int		O		1	
주문 일자	indate	datetime		O		CURRENT_TIMESTAMP	
주문자 아이디	id	varchar	20	X	FK		customer(id)
상품 코드	pcode	varchar	20	X	FK		products(pcode)