<SQL 2012(코드명 Denali) 시리즈 강좌 리스트>

목차

[SQL2011 강좌] 1. 코드명 Denali 설치 방법 3
[SQL2011 강좌] 2. SSMS 접속 및 간단한 쿼리 실행 21
[SQL2011 강좌] 3. SQL Server 의 MDF, LDF 그리고 데이터베이스 생성하기 27
[SQL2011 강좌] 4. DDL 1 탄: 테이블 생성 34
[SQL2011 강좌] 5. DDL 2 탄: 쿼리를 이용한 테이블 컬럼 추가 및 삭제 41
[SQL2011 강좌] 6. Primary key 와 Unique 제약조건 49
[SQL2011 강좌] 7. DML 1 탄 INSERT 56
[SQL2011 강좌] 8. DML 1 탄 INSERT(identity 속성) 62
[SQL2011 강좌] 9. Sequence 66
[SQL2011 강좌] 10. pubs, northwind Database 예제 설치 68
[SQL2011 강좌] 11. SELECT(단일 테이블) 70
[SQL2011 강좌] 12. SELECT(JOIN) 72
[SQL2011 강좌] 13. UPDATE, DELETE 절 80
[SQL2011 강좌] 14. DISTINCT, UNION, UNION ALL 키워드 84
[SQL2011 강좌] 15. ORDER BY, GROUP BY, HAVING 89
[SQL2011 강좌] 16. SubQuery(서브쿼리) 96
[SQL2011 강좌] 17. 뷰에 대한 이해 및 생성, 수정, 삭제 105
[SQL2011 강좌] 18. 저장 프로시저에 대한 이해, 생성, 수정, 삭제 115
[SQL2011 강좌] 19. 인덱스에 대한 이해 122
[SQL2011 강좌] 20. 클러스터드 인덱스와 넌 클러스터드 인덱스 126
[SQL2011 강좌] 21. 트랜잭션에 대한이해 147
[SQL2011 강좌] 22. 트랜잭션의 격리수준 4 가지 151
[SQL2011 강좌] 23. 잠금에 대한 이해 162
[SQL2011 강좌] 24. 백업과 복구에 대한 이해(풀백업, 차등백업, 트랜잭션 로그 백업) 172
[SQL2011 강좌] 25. 백업과 복구 전략 178
[SQL2011 강좌] 26. 커서란? 211

[SQL 2012(코드명 Denali) 시리즈 강좌 및 동영상 리스트]

[SQL2011 동영상 강좌] 1. 코드명 Denali 설치 방법

[SQL2011 동영상 강좌] 2. SSMS 접속 및 간단한 쿼리 실행

[SQL2011 동영상 강좌] 3. SQL Server 의 MDF, LDF 그리고 데이터베이스 생성하기

[SQL2011 동영상 강좌] 4. DDL 1 탄: 테이블 생성

[SQL2011 동영상 강좌] 5. DDL 2 탄: 쿼리를 이용한 테이블 컬럼 추가 및 삭제

[SQL2011 동영상 강좌] 6. Primary key 와 Unique 제약조건

[SQL2011 동영상 강좌] 7. DML 1 탄 INSERT

[SQL2011 동영상 강좌] 8. DML 1 탄 INSERT(identity 속성)

[SQL2011 동영상 강좌] 9. Sequence

[SQL2011 동영상 강좌] 10. pubs, northwind Database 예제 설치

[SQL2011 동영상 강좌] 11. SELECT(단일 테이블)

[SQL2011 동영상 강좌] 12. SELECT(JOIN)

[SQL2011 동영상 강좌] 13. UPDATE, DELETE 절

[SQL2011 동영상 강좌] 14. DISTINCT, UNION, UNION ALL 키워드

[SQL2011 동영상 강좌] 15. ORDER BY, GROUP BY, HAVING

[SQL2011 동영상 강좌] 16. SubQuery(서브쿼리)

[SQL2011 동영상 강좌] 17. 뷰에 대한 이해 및 생성, 수정, 삭제

[SQL2011 동영상 강좌] 18. 저장 프로시저에 대한 이해, 생성, 수정, 삭제

[SQL2011 동영상 강좌] 19. 인덱스에 대한 이해

[SQL2011 동영상 강좌] 20. 클러스터드 인덱스와 넌 클러스터드 인덱스

[SQL2011 동영상 강좌] 21. 트랜잭션에 대한이해

[SQL2011 동영상 강좌] 22. 트랜잭션의 격리수준 4 가지

[SQL2011 동영상 강좌] 23. 잠금에 대한 이해

[SQL2011 동영상 강좌] 24. 백업과 복구에 대한 이해(풀백업, 차등백업, 트랜잭션 로그 백업)

[SQL2011 동영상 강좌] 25. 백업과 복구 전략

[SQL2011 동영상 강좌] 26. 커서란?

[SQL2011 강좌] 1. 코드명 Denali 설치 방법

이 내용은 2011 년 6 월 현재 SQL2011(코드명 Denali) Beta 를 기준으로 작성 되었으며 SQL2011(코드명 Denali) 공식버전(RTM) 발표까지 꾸준히 업데이트 예정이다.

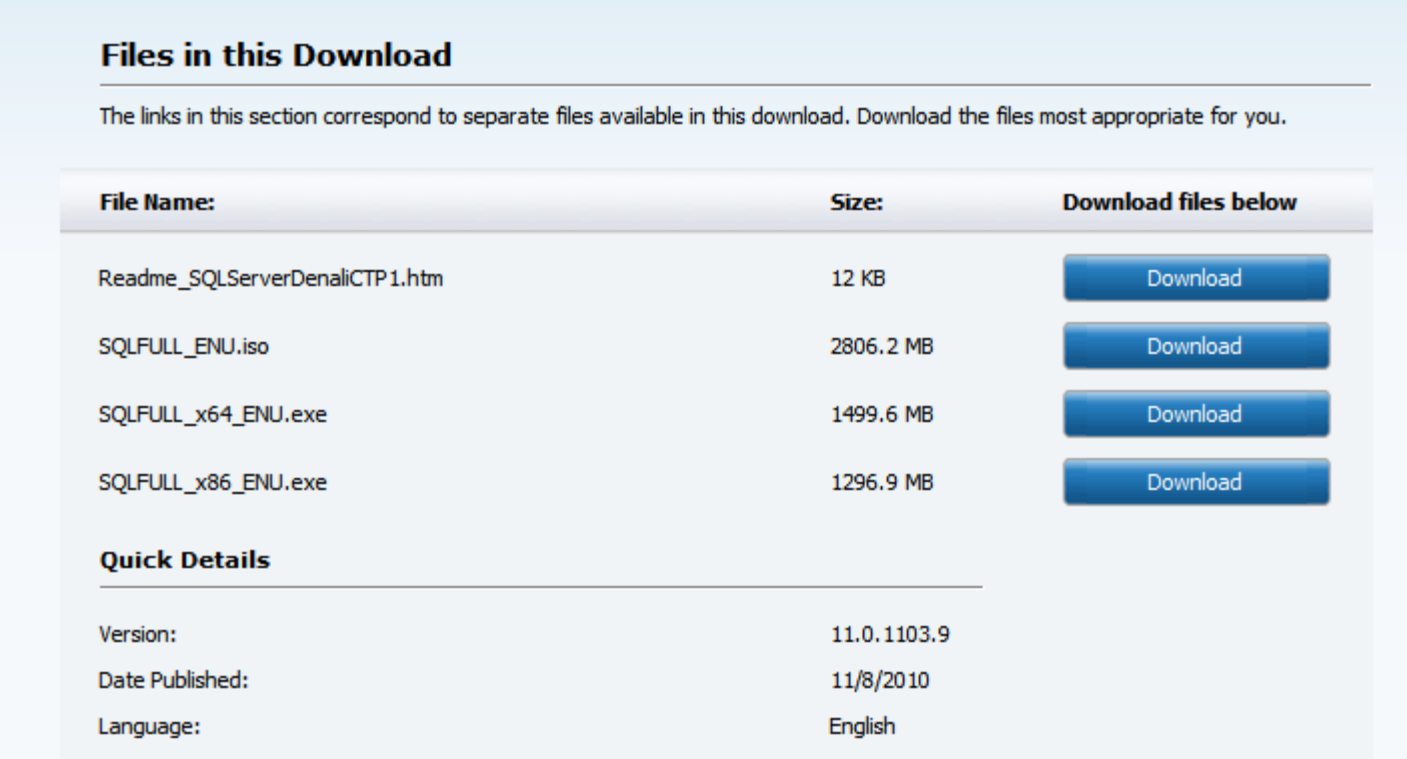
==========

이번 강좌는 SQL Server 2011 Denali 의 설치에 대해서 알아보도록 하겠다.

현재 영문판만 제공되기 때문에.. 아래 링크를 통해서 다운 받으시면 된다

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=6a04f16f-f6be-4f92-9c92-f7e5677d91f9

위 링크 중에서 exe 파일을 내려 받으시면 된다.



Files in this Download

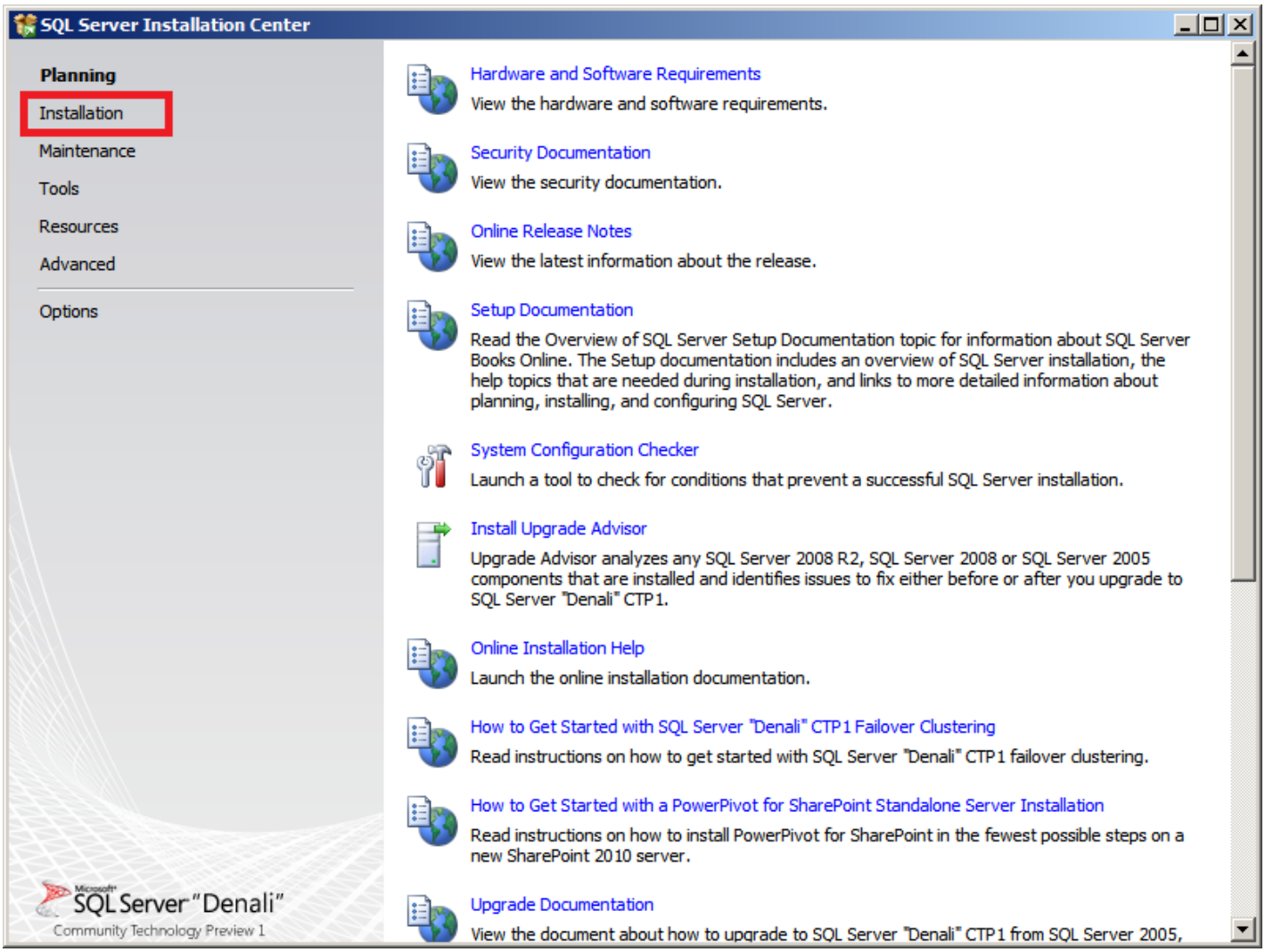
The links in this section correspond to separate files available in this download. Download the files most appropriate for you.

File Name:	Size:	Download files below
Readme_SQLServerDenaliCTP1.htm	12 KB	Download
SQLFULL_ENU.iso	2806.2 MB	Download
SQLFULL_x64_ENU.exe	1499.6 MB	Download
SQLFULL_x86_ENU.exe	1296.9 MB	Download

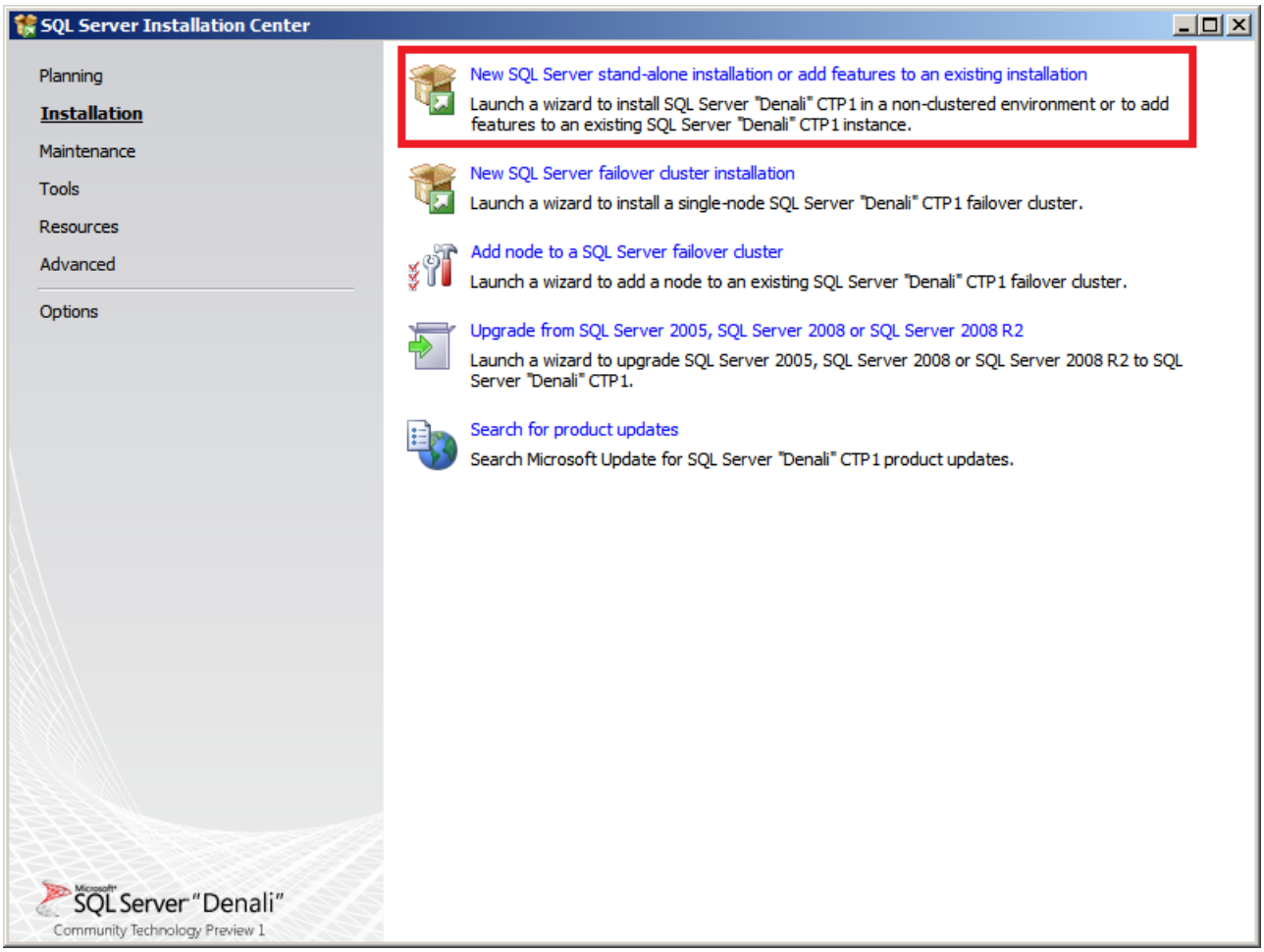
Quick Details

Version:	11.0.1103.9
Date Published:	11/8/2010
Language:	English

다운 받은 파일을 실행을 하시면 아래와 같은 화면이 나오게 된다. 일단 설치를 위해 Installation 을 클릭한다.

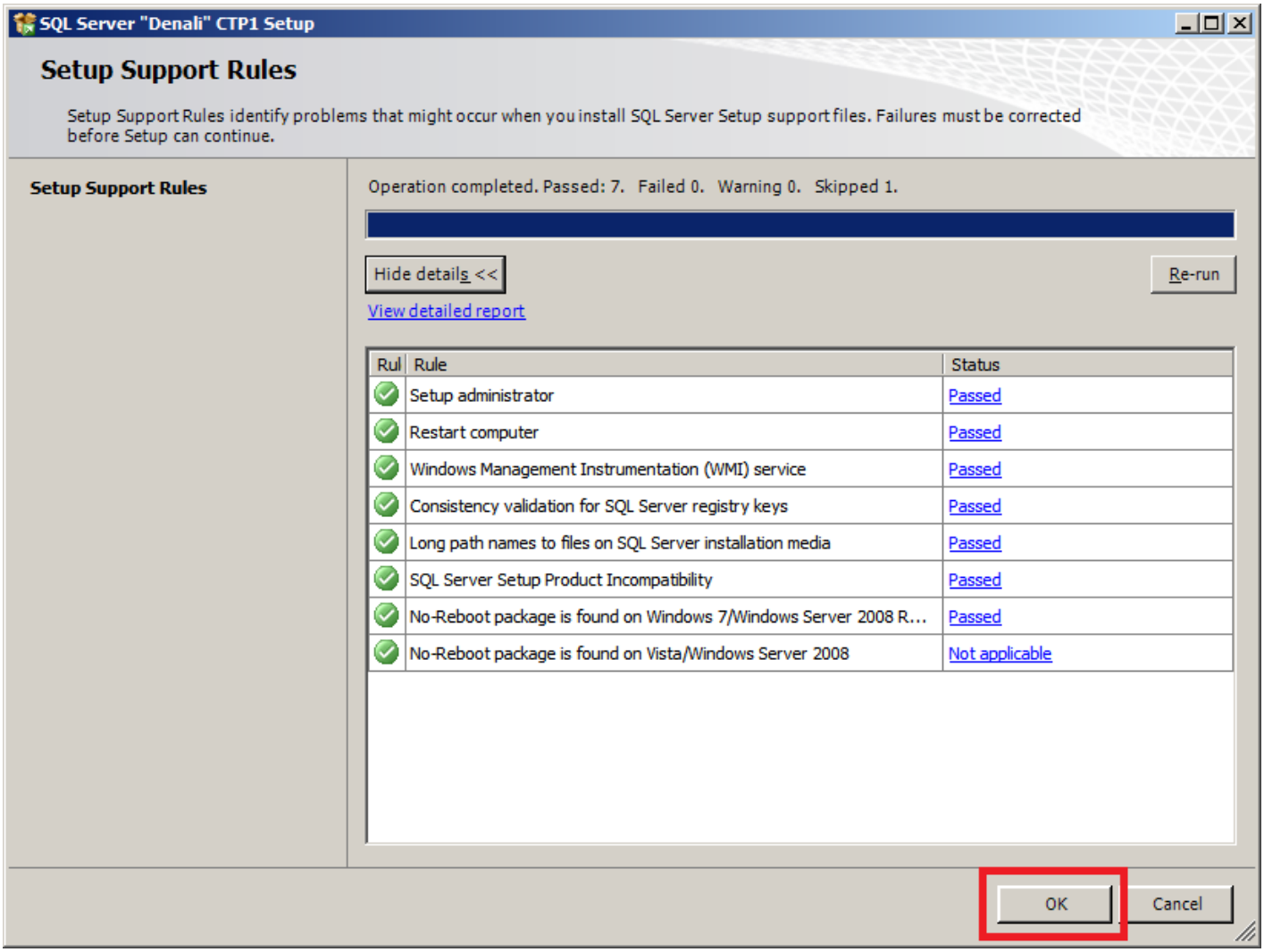


Installation 을 클릭 후에는 아래와 같은 화면이 나오게 된다. 이 중에서 저희는 SQL Server 를 설치하는 것이 목적이기 때문에 첫 번째 항목을 선택하도록 한다.

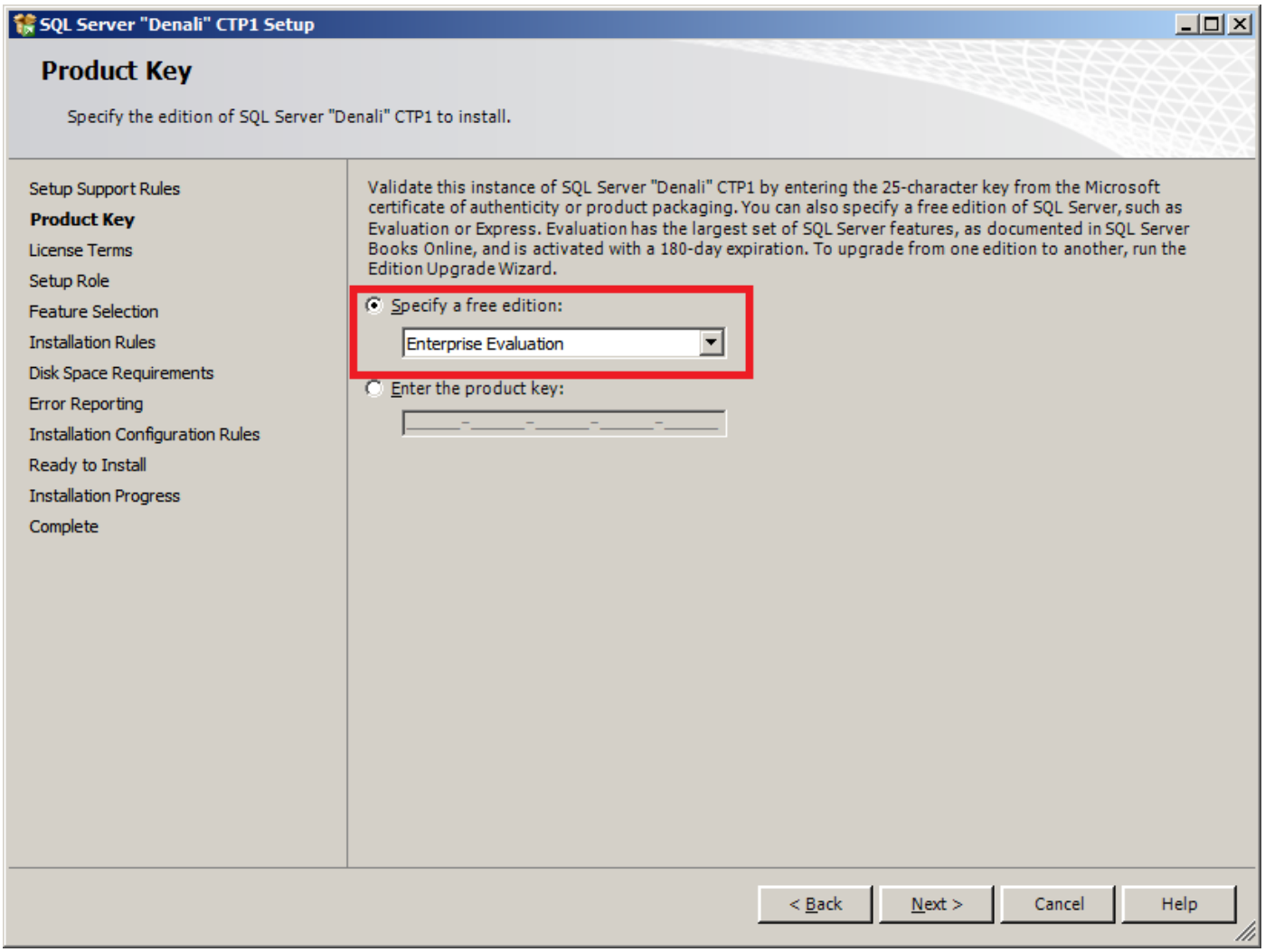


새 창이 뜨면서 설치가 진행이 된다.

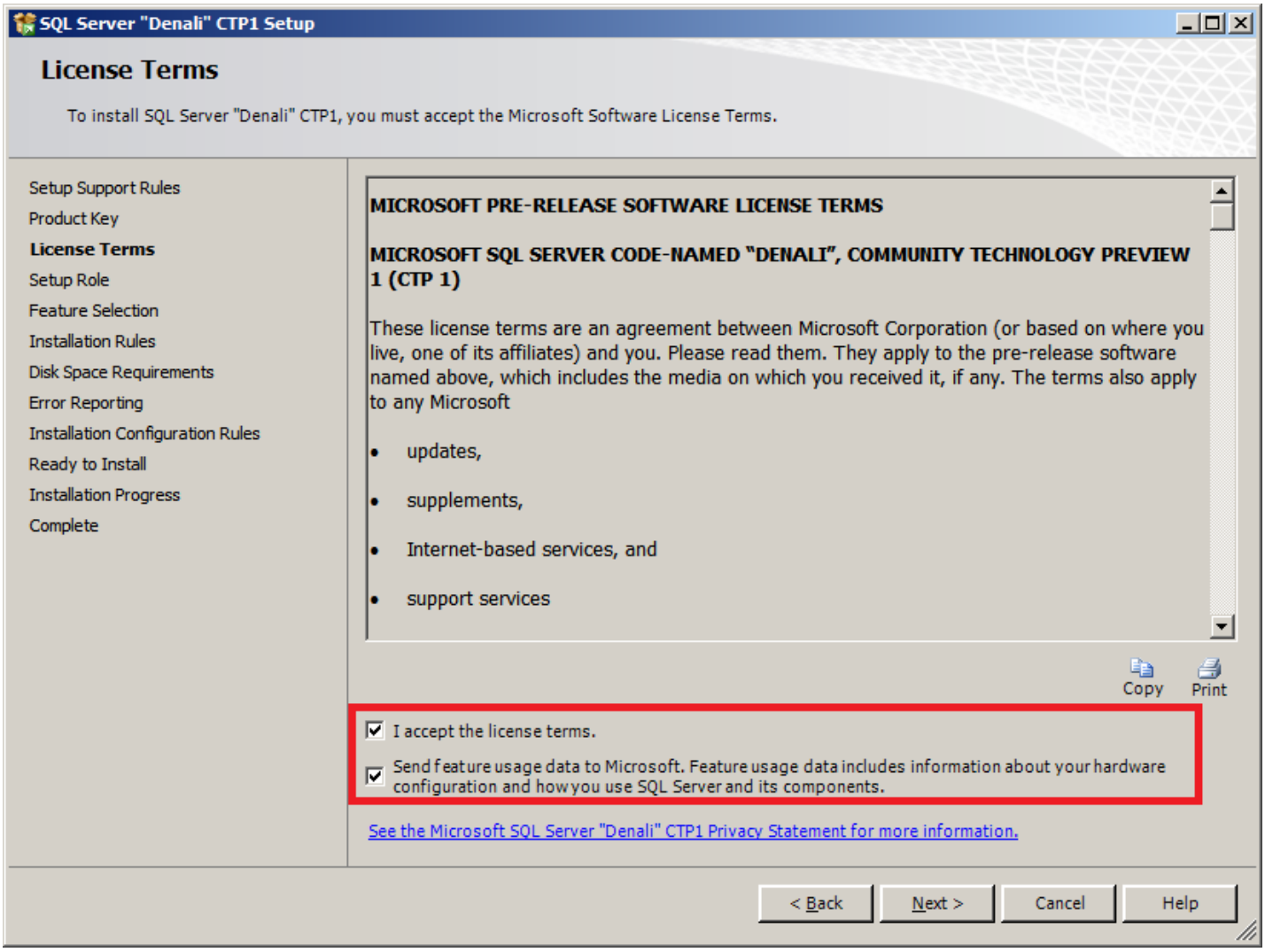
아래 항목들은 현재 서버에 SQL Server 2011 이 설치가 가능한지에 대해서 알아보는 화면이다. SQL Server 2011 을 설치하기 위해서는.. 최소 .NET Framework 3.5 Package 이상이 설치되어야 한다. 또한 기타 여러 가지 제약조건이 통과가 되면 아래 Next 버튼이 활성화가 된다. 만약 Next 버튼이 비활성화 되어있다면.. Status 항목 중에 실패한 것이 없는지 체크해보시기 바란다.



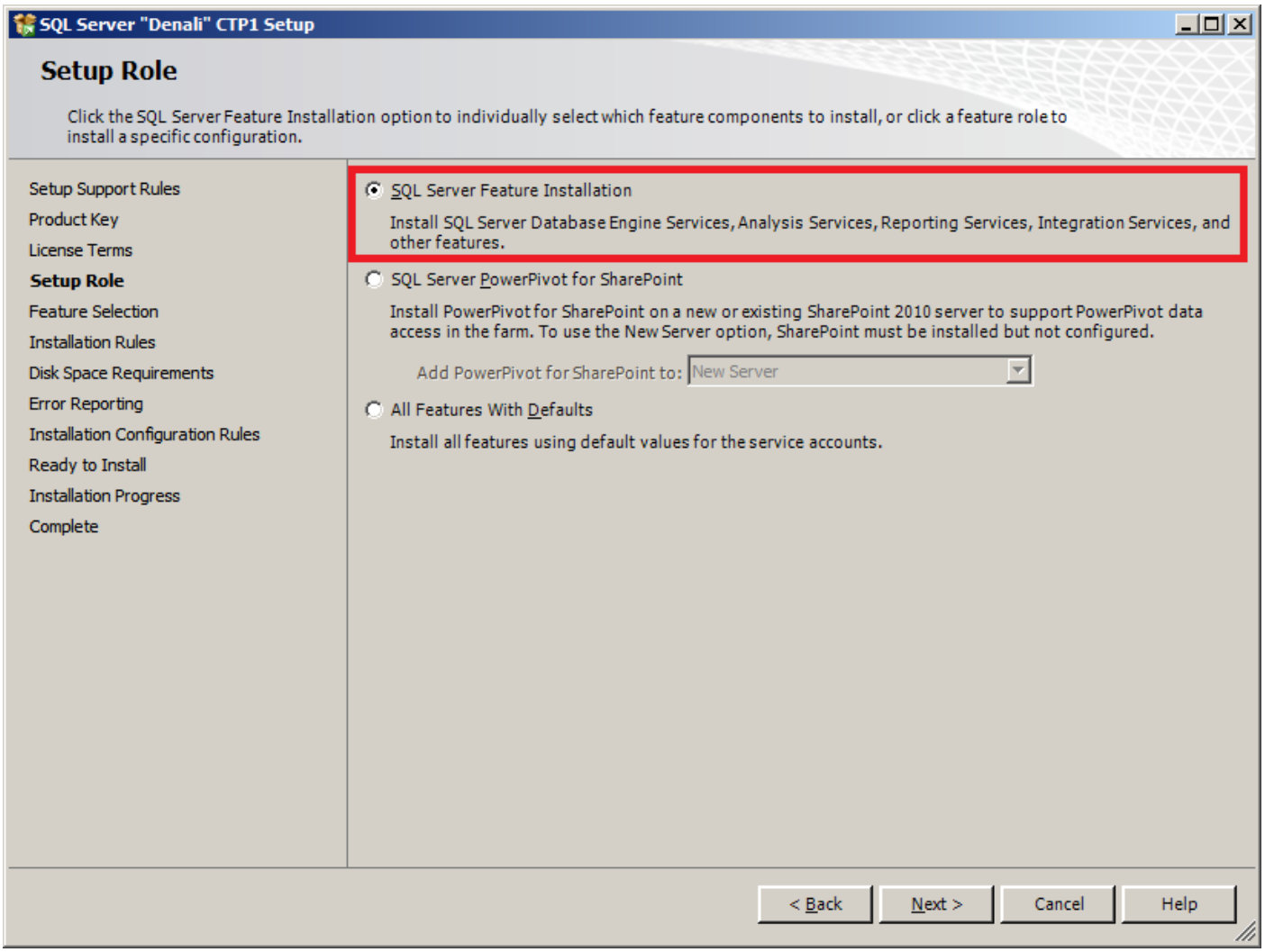
여기서는 제품 종류를 선택하는 화면이 나옵니다. 일단 저희는 Enterprise Evaluation edition 을 설치해보도록 하겠다. 제품을 선택하고, Next 를 선택한다.



이번 화면은.. 설치 동의서 라고 보면 된다. 체크 박스에 모두 체크하고 Next 를 선택한다.

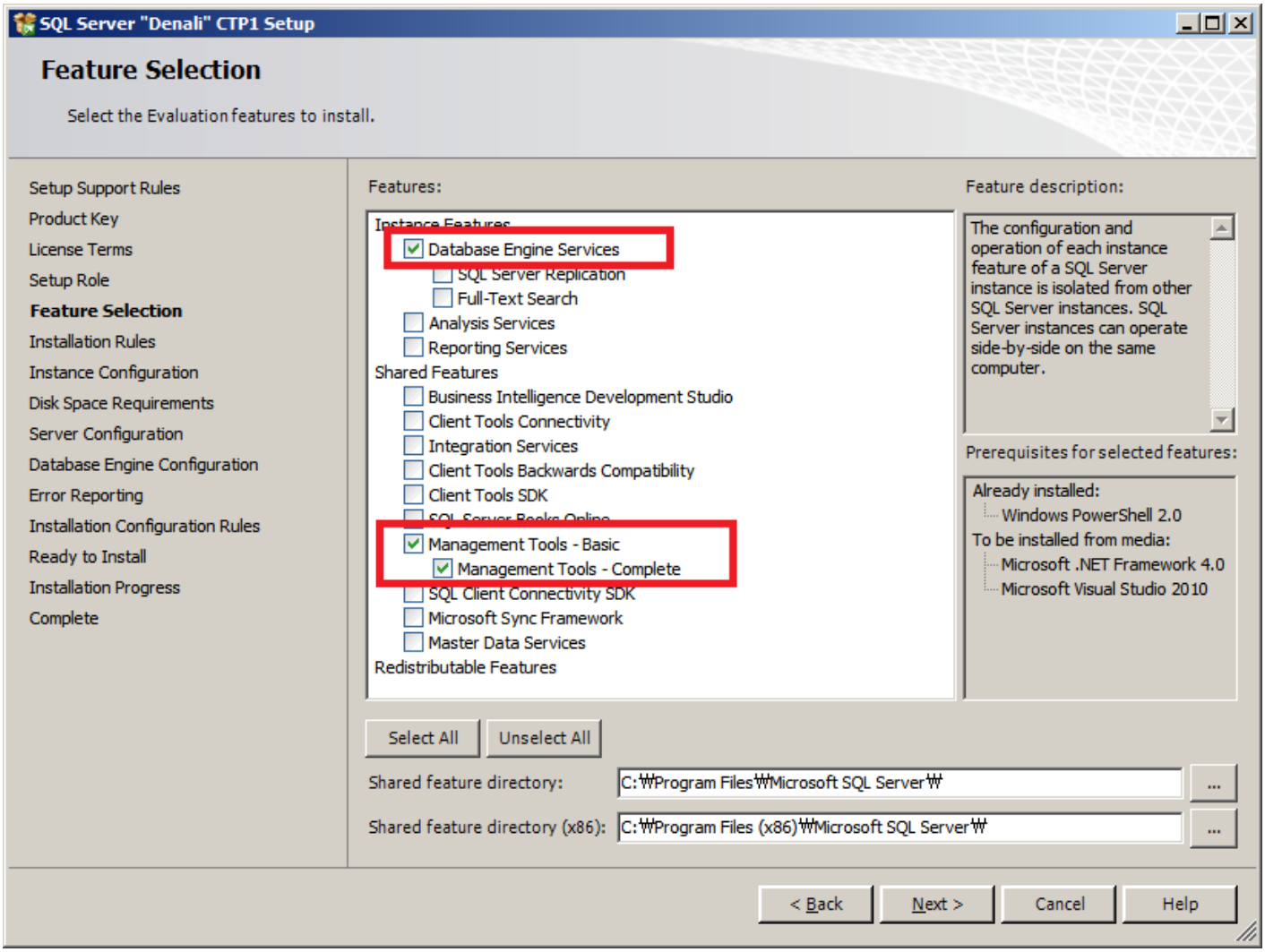


이번 화면은 SQL Server Denali 을 어떤 방식으로 설치할 지에 대해서 선택하는 화면이다. 첫 번째 항목을 선택한 후 Next 를 선택한다.

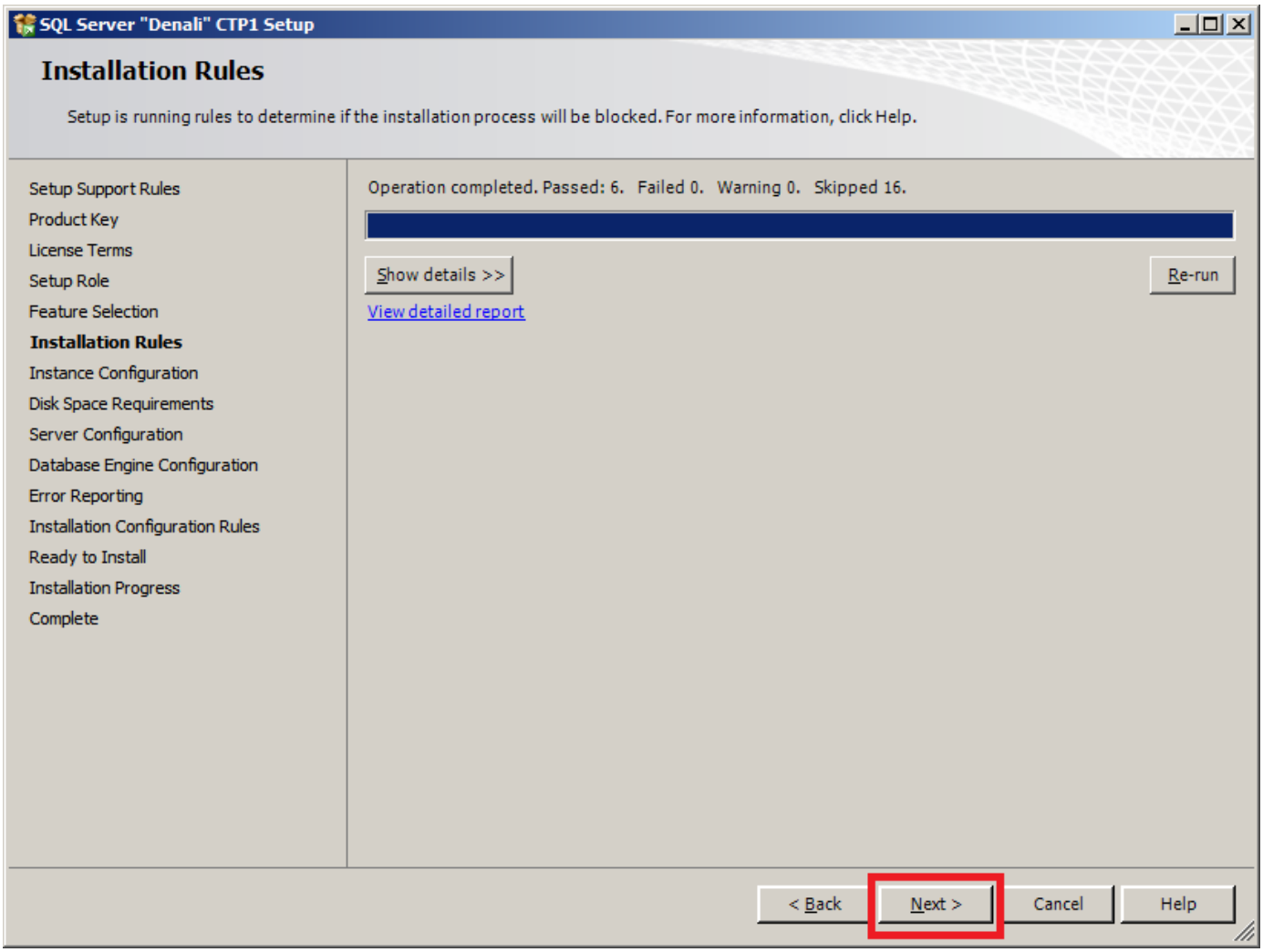


이번 화면은.. Denali 중에서 어떤 항목을 설치할 것인지를 선택하는 화면이다. 기본 적인 구동을 위해서는 Database Engine Service 와 Management Tools - Basic 과 complete 항목만 선택하도록 한다.

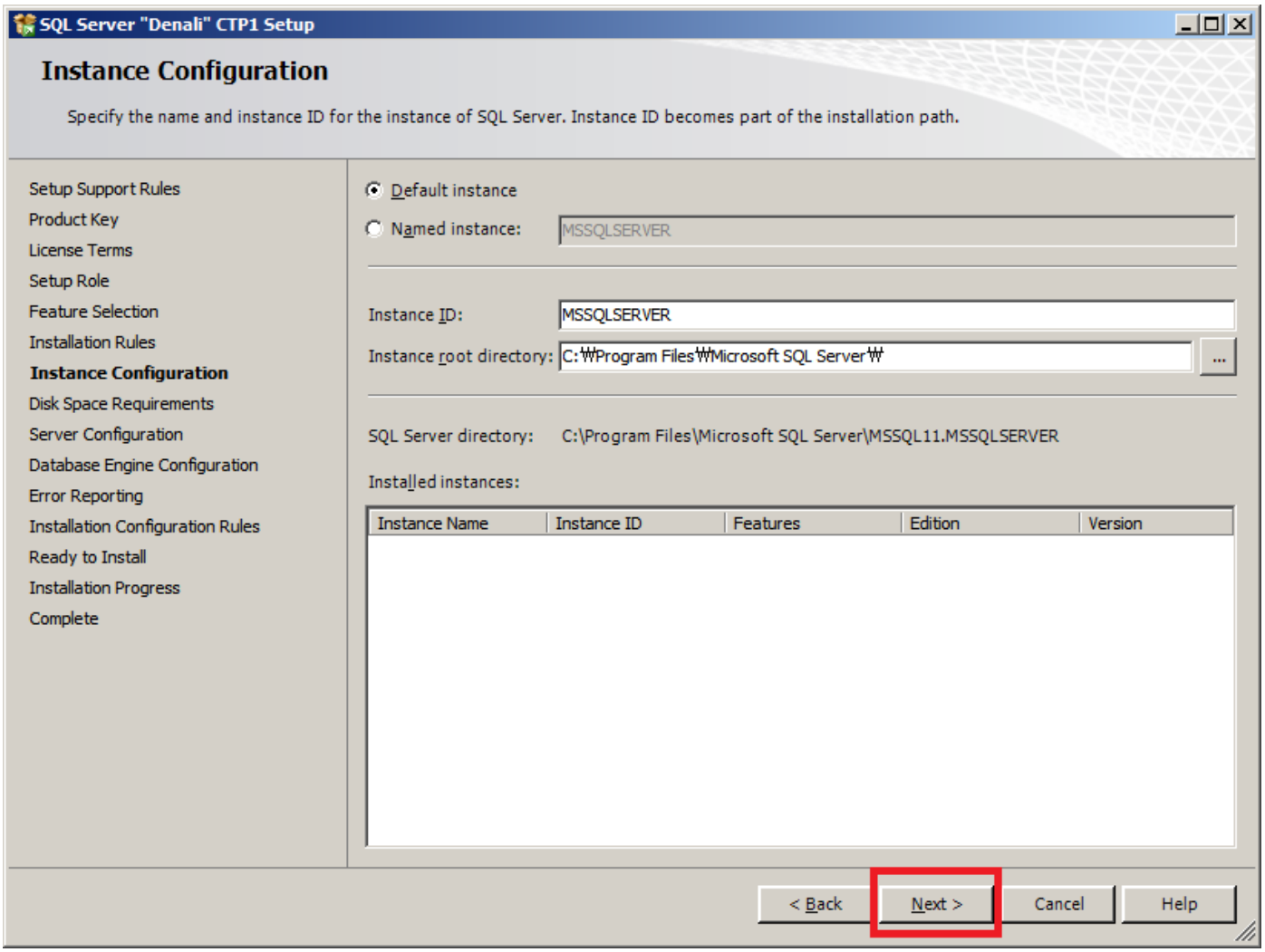
그리고 Next 를 선택한다.



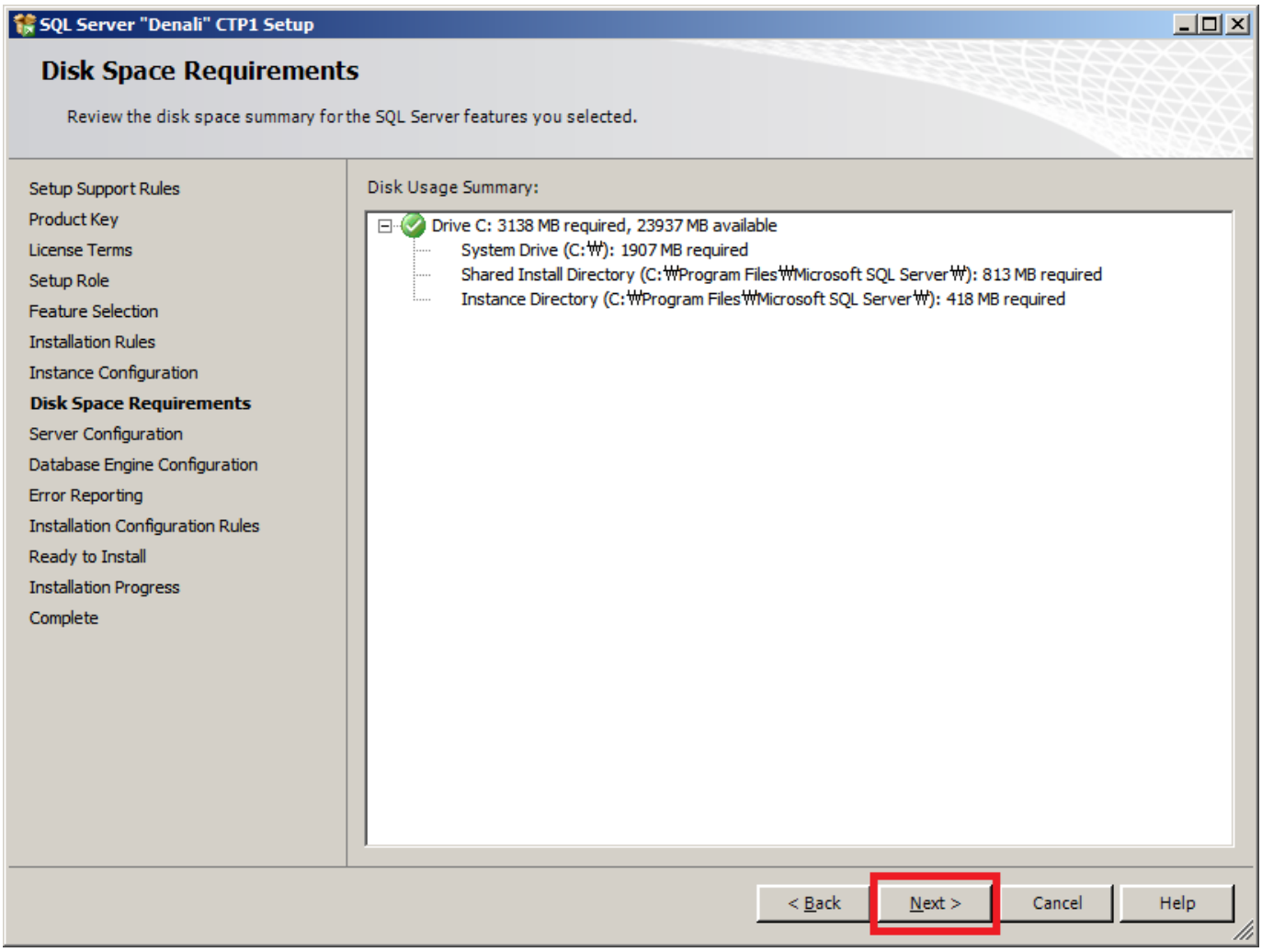
이번 화면은.. Denali 설치에 위배되는 제약조건이 없는지 검사하는 부분이다. 만약 문제가 된다면 Show details 를 클릭해서 문제가 되는 조건을 확인해보시기 바란다. 일단 문제가 없으므로 Next 를 선택한다.



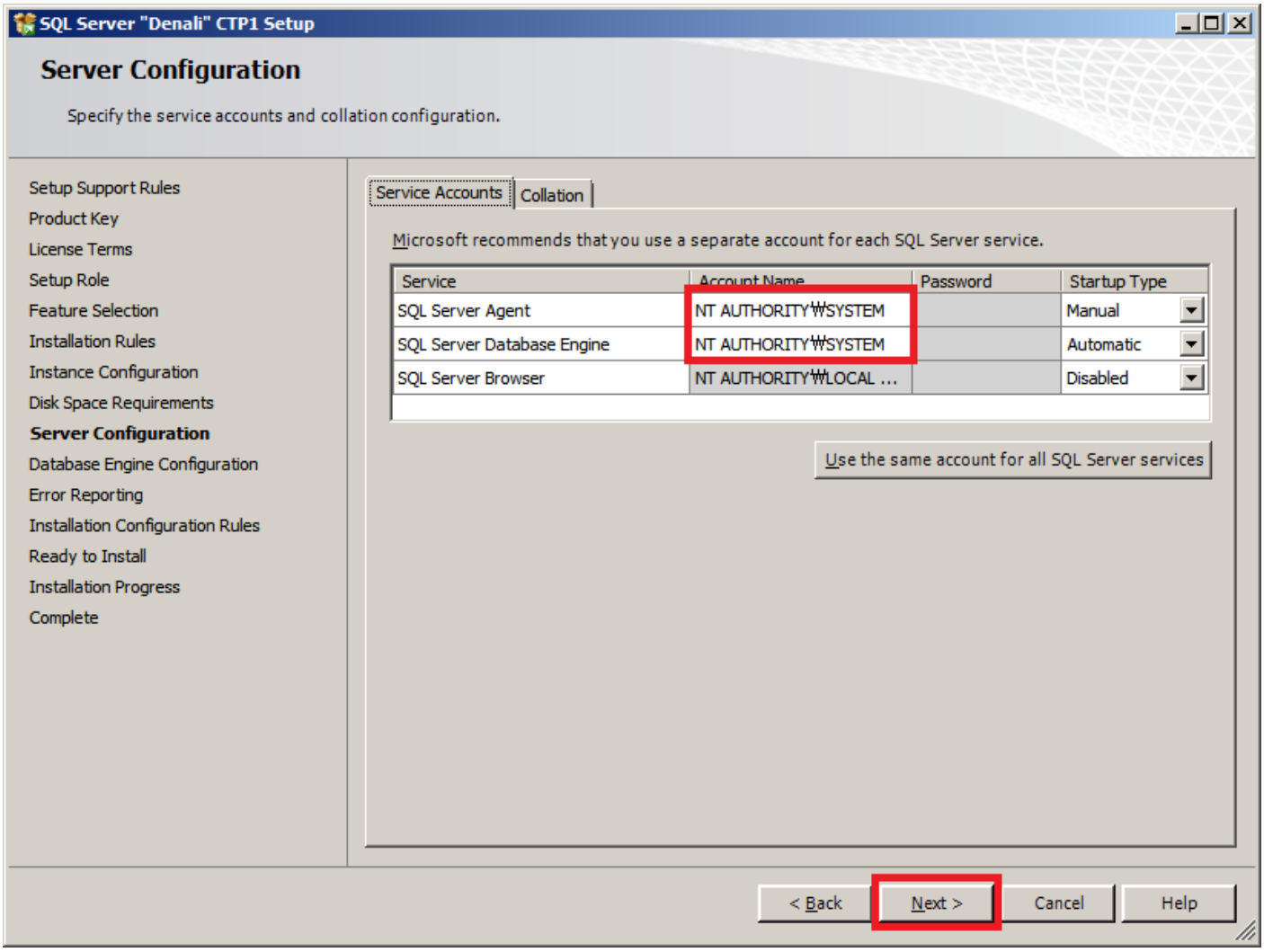
이번 화면은.. 인스턴스 명을 지정하는 것이다. 하나의 윈도우 서버에는 여러 개의 SQL Server 를 설치할 수 있다. 이때 인스턴스 명으로 여러 개의 SQL Server 가 구분이 될 수 있다. 저희는 하나만 설치하는 것이기 때문에.. Default instance 로 설치하도록 하겠다.



이번 화면은.. 설치 디스크의 사용 가능 공간과 설치에 필요한 용량을 수치로 표시한 화면이다.크게 의미는 없으므로 Next 를 선택한다.



이번 화면은.. SQL Server 구성 정보를 등록하는 화면이다. SQL Server 는.. Windows 서버에서 하나의 서비스 형태로 작동을 하게 된다. 이때 이 서비스를 실행하기 위한 윈도우 계정이 필요하게 된다. SQL Server Agent 또한 SQL Server 와는 다르게 또 다른 서비스의 형태로 작동하게 된다. SQL Server Agent 는 특정 시간에 원하는 구문을 처리한다거나.. 백업 등을 담당하는 아주 중요한 서비스 이다. 깊이 있는 부분은 추후 강좌에 설명 드리도록 하겠다. 보안상 이번 스텝은 중요한 부분이다만... 추후 강좌에 설명드릴 수 있다면 깊이있게 설명드리겠다. 우선은 NT AUTHORITY₩SYSTEM 계정으로 선택한 뒤에 Next 를 선택한다.

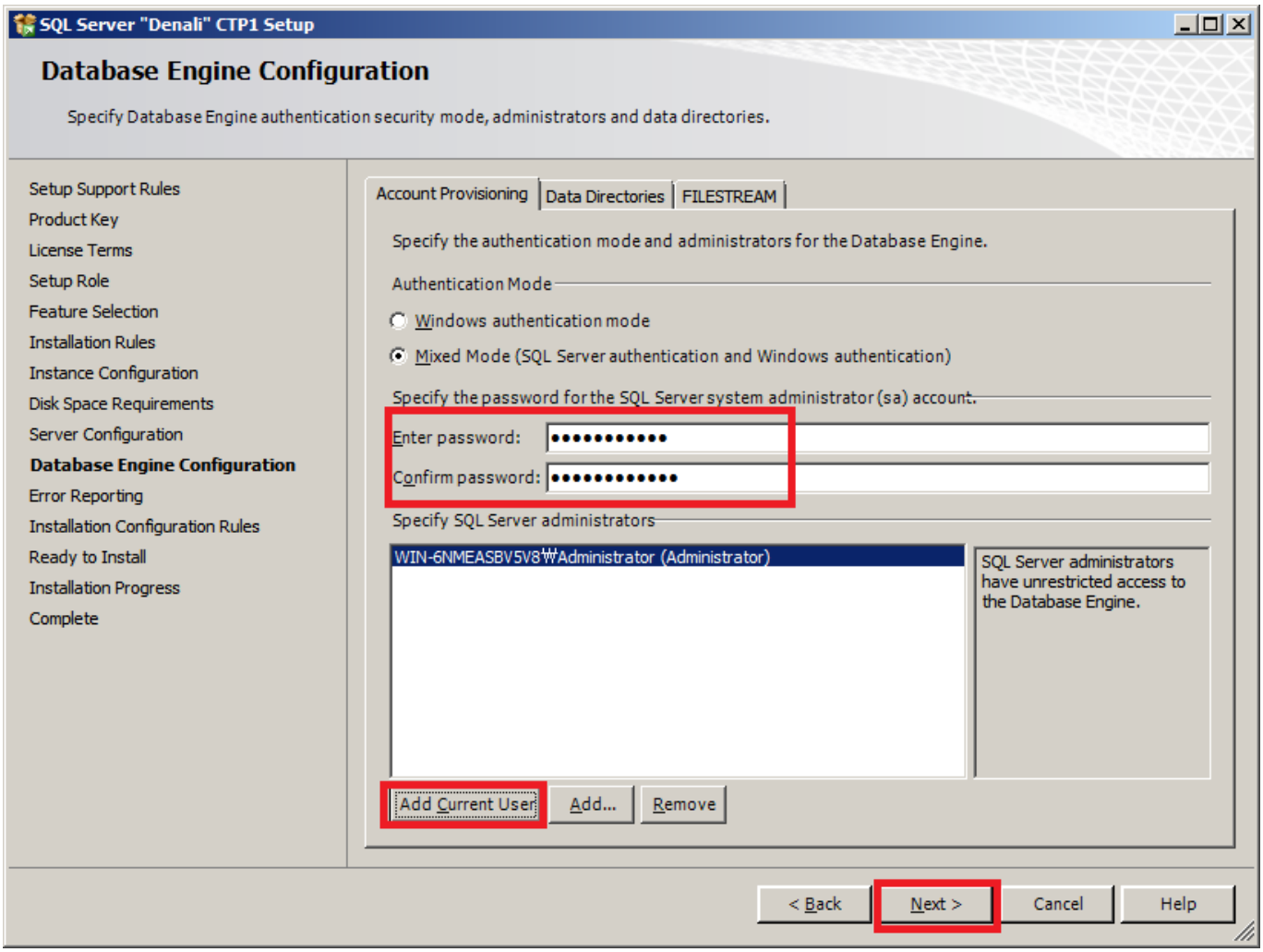


이번 화면은.. SQL Server 의 계정을 어떤 식으로 이용할지에 대한 화면 이다.

일반 적으로 SQL Server 는 윈도우 계정 모드와 윈도우 계정 + SQL 계정 모드를 혼합할 수 있는 두 가지 형태로 제공되고 있다.

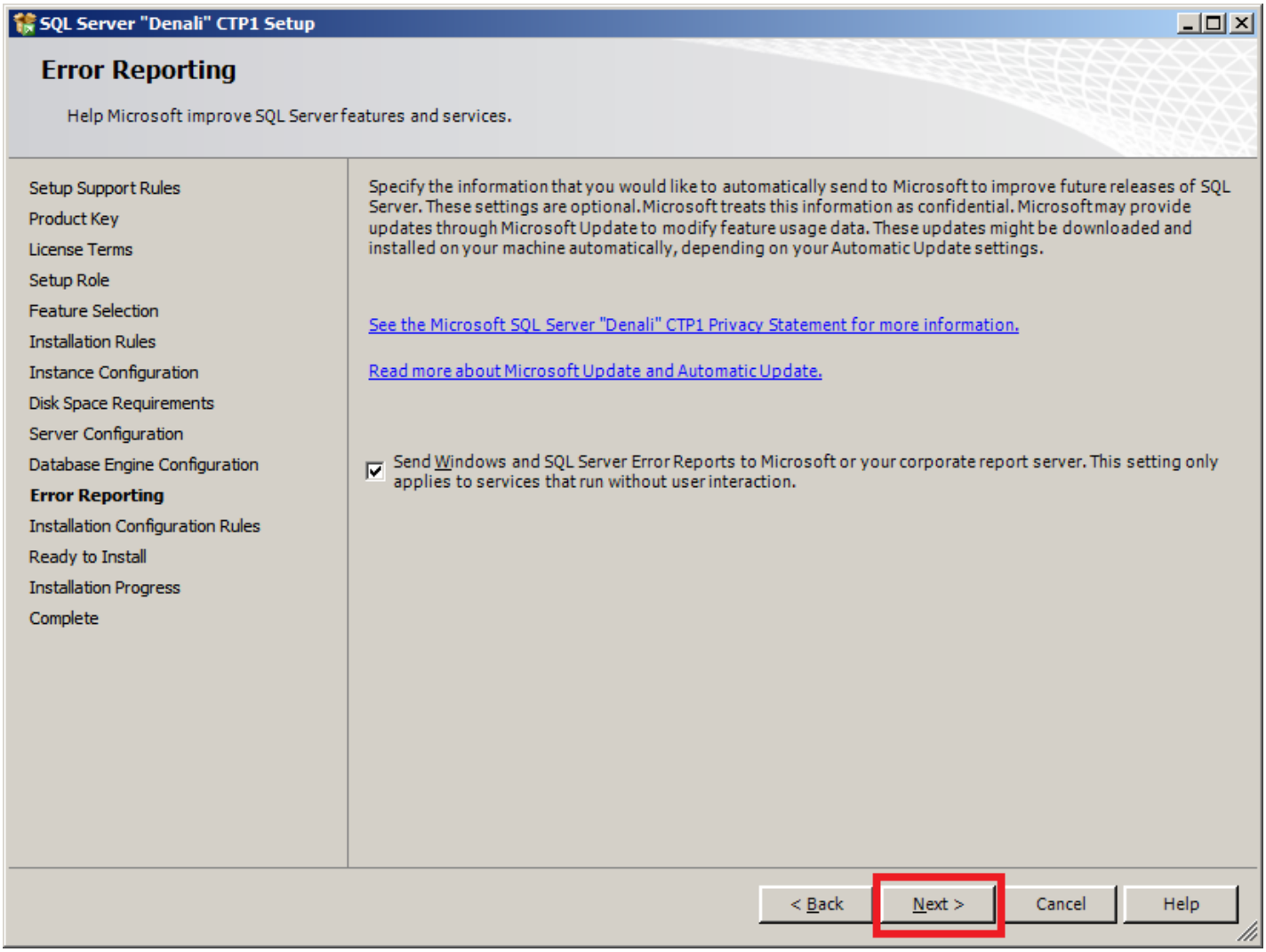
윈도우 계정이라고 하는 것은... 우리가 현재 설치를 하고 있는 윈도우 계정을 말하고

SQL 계정은.. SQL Server 내에 계정을 생성해서 로그인을 할 수 있는 방법을 말한다. 우리는 여기서 2 번째 혼합 모드로 선택을 하고 SQL Server 의 기본 admin 계정인 sa 의 비밀번호를 넣은 다음 현재 윈도우 계정을 Server admin 으로 등록하기 위해 add Current User 를 클릭 후 Next 를 선택한다.

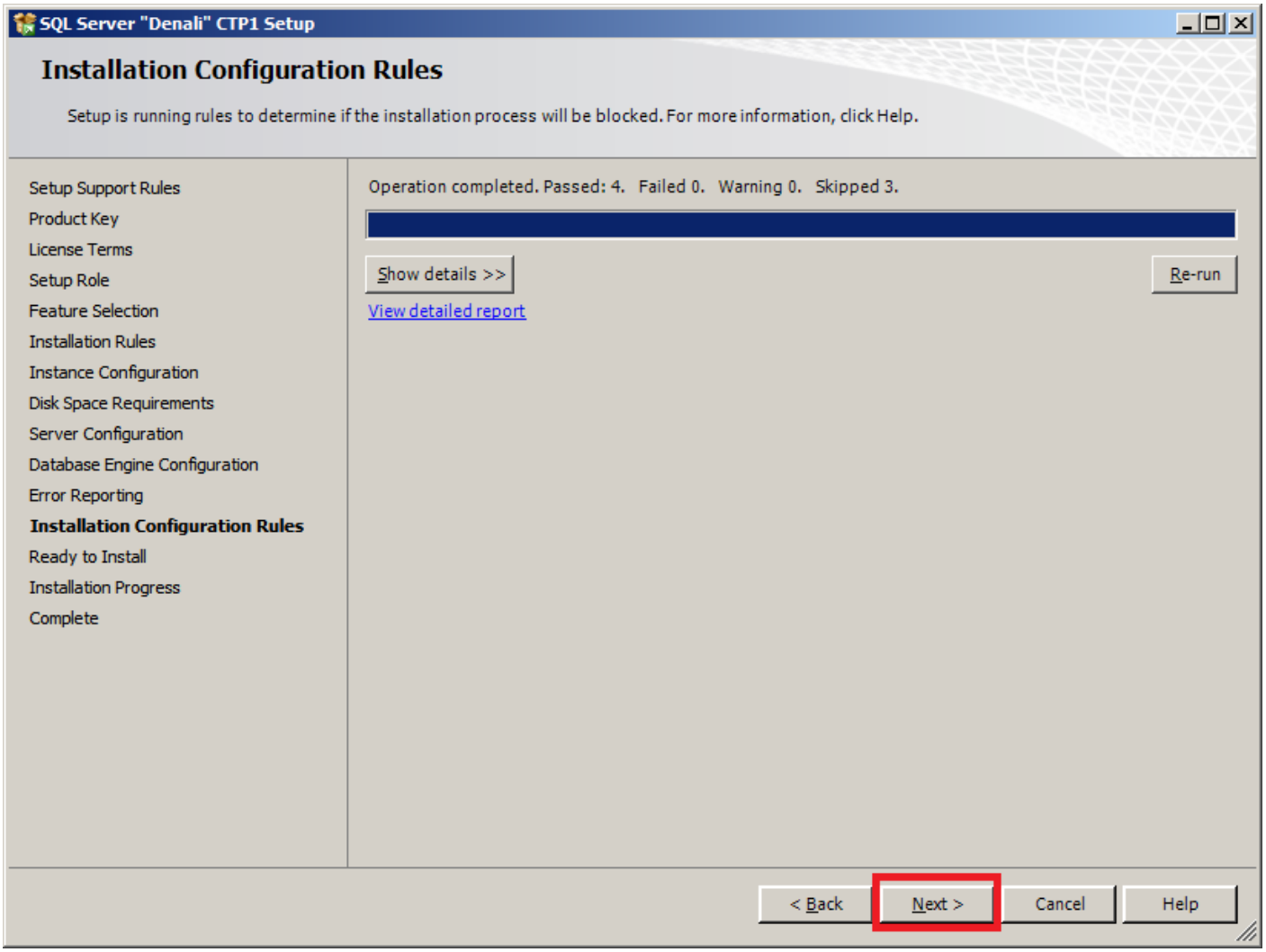


이번 화면은.. 설치 한 정보를 MS 에다가 제공할지 여부를 결정하는 정보이다. 이런 설치 정보를 MS 에서 수집함으로 써.. 좀 더 설치 오류에 대해서 개선을 하게 되고, 제품의 질을 향상되게 할 수 있다.

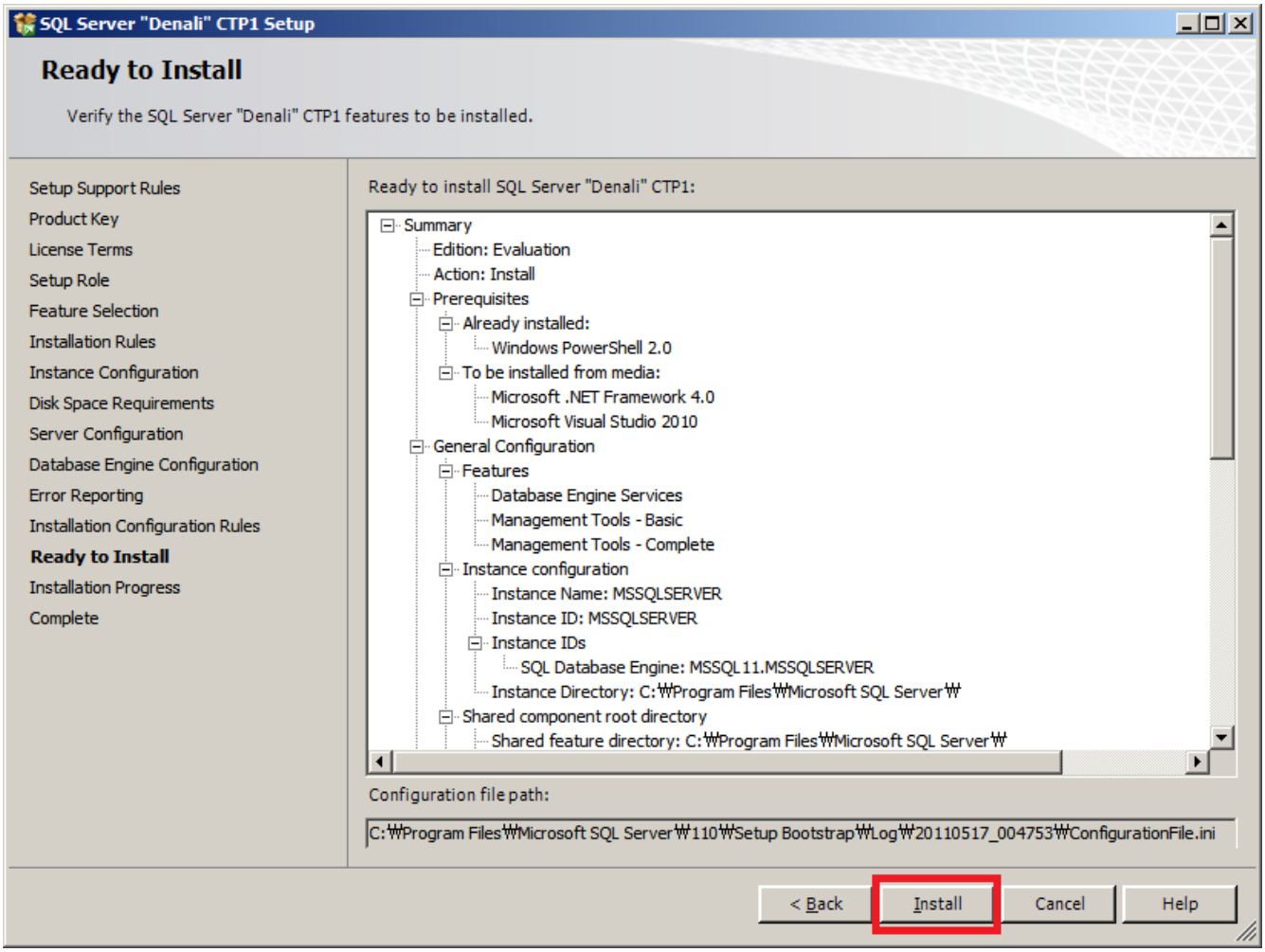
체크 후 Next 를 선택한다.



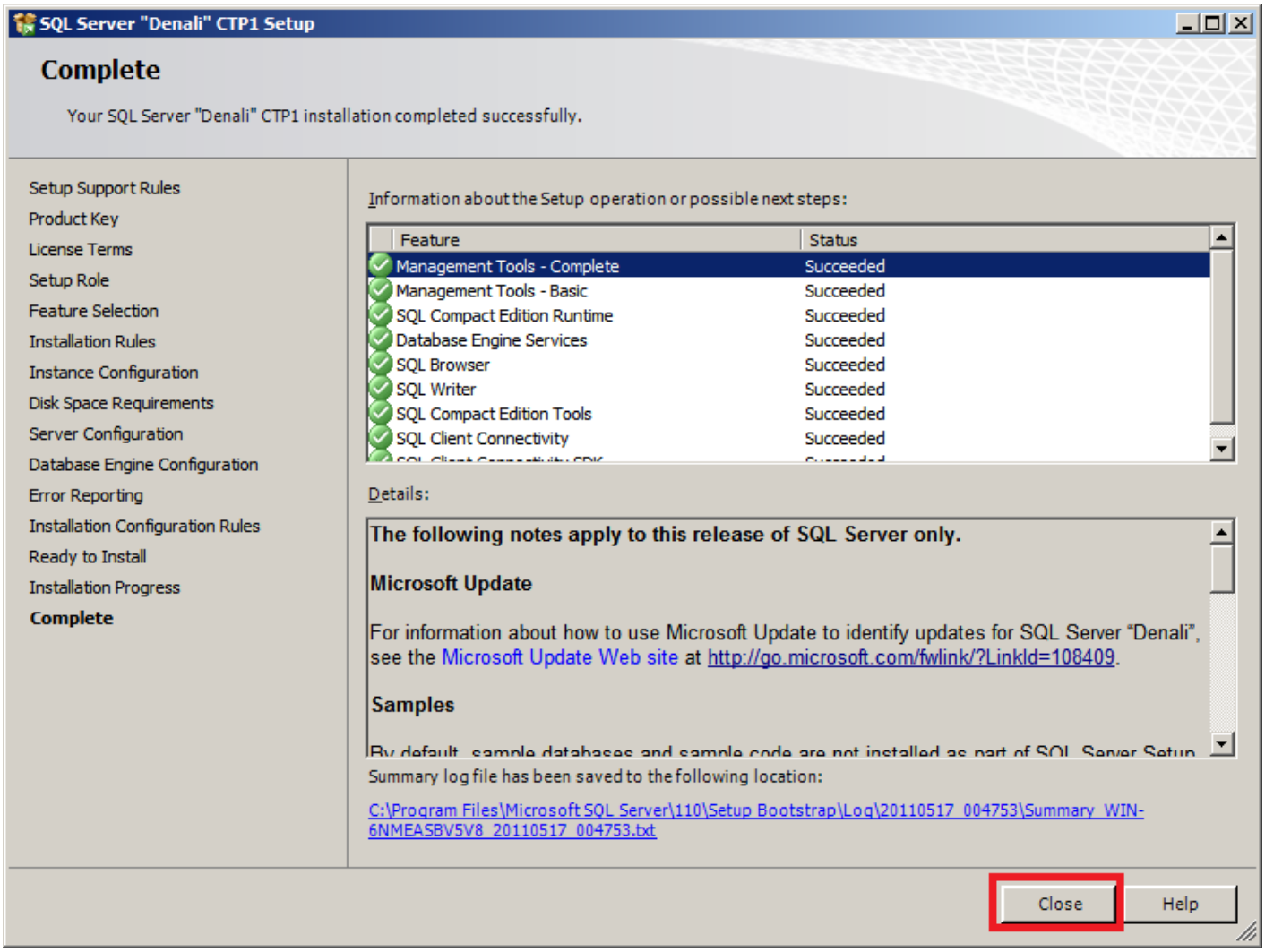
이제 우리가 설정했던 인스톨의 내용이 제대로 설정을 한 건지 최종적으로 체크한 뒤 Next 를 선택한다.



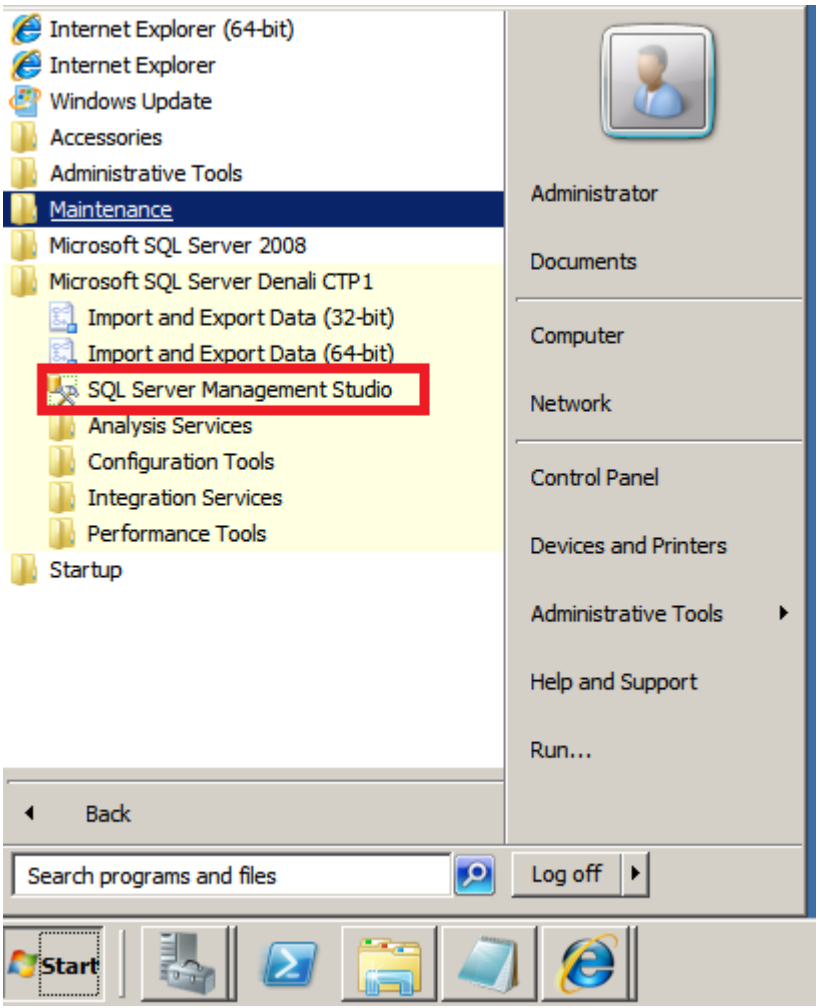
이번 화면은... 최종적으로.. 이렇게 설치될 것이라고 보여주는 화면이다. 이제 Install 을 선택한다.



이제 인스톨이 완료되었다. 모든 항목이 제대로 설치 되었는지 체크해보시고, Close 를 누른다.



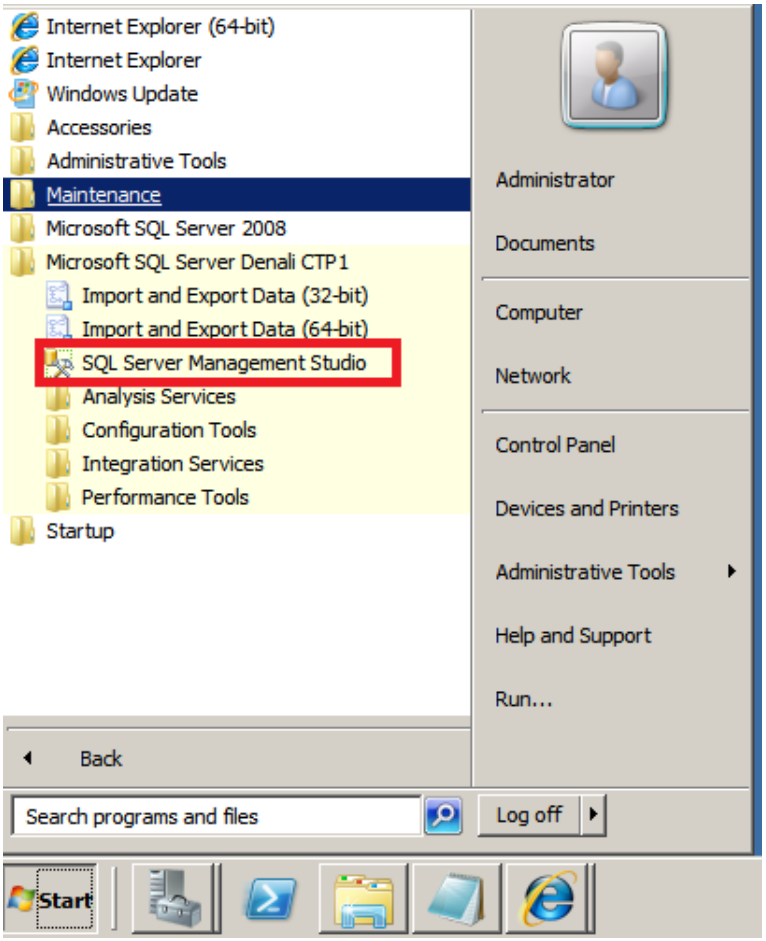
이제 마지막으로 시작 버튼을 클릭해서.. SQL Server Denali 이 제대로 설치가 되었는지 확인해 볼 차례이다.



정상적으로 설치가 되었다.

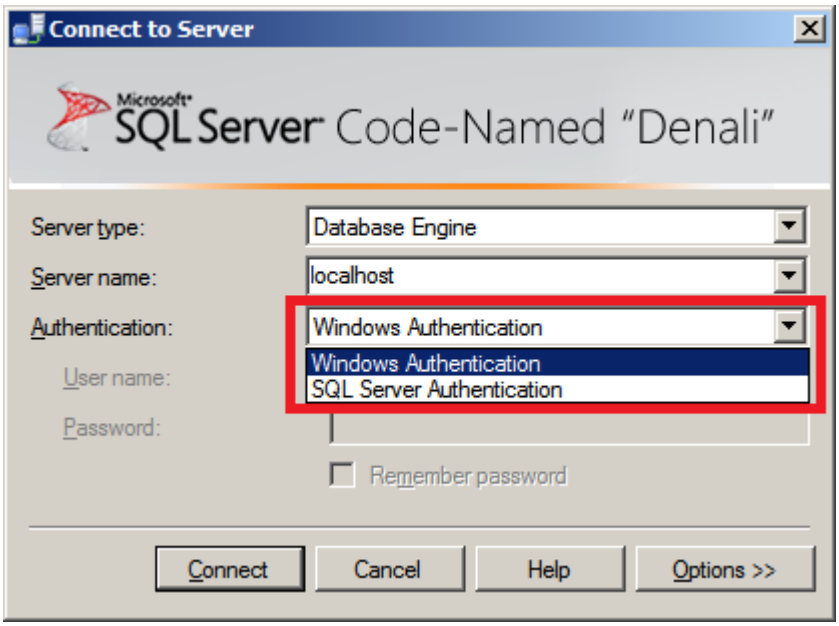
[SQL2011 강좌] 2. SSMS 접속 및 간단한 쿼리 실행

이번 강좌에서는.. 설치된 SQL Server 를 가지고 접속에 대해서 알아볼 차례이다. 접속을 하기 위해서는.. SSMS 라는 도구가 필요하다.(물론 설치시에 설치가 되어있다.) SSMS 는.. SQL server Management Studio 의 약자이며, 아래 위치에 설치가 되어있다.

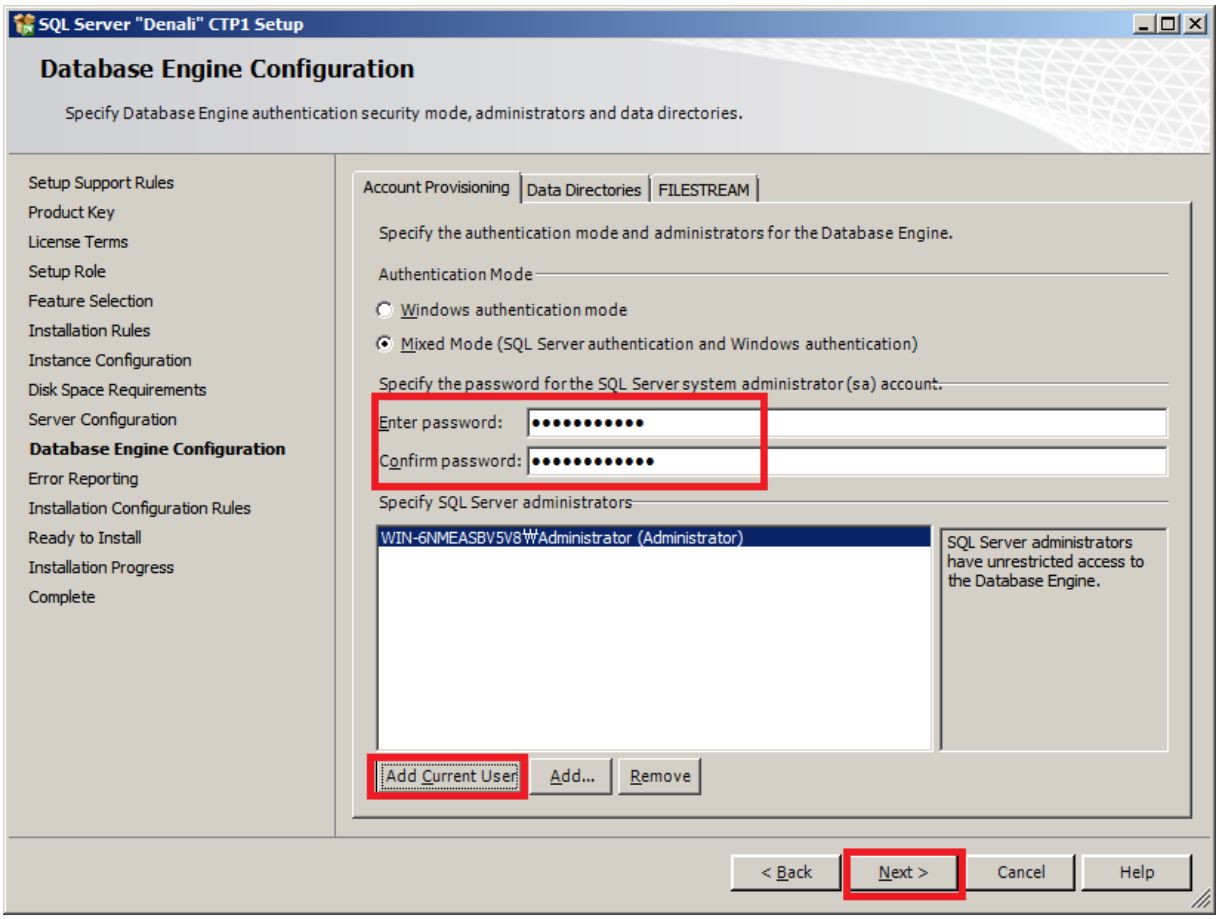


이걸 클릭해서 SQL Server 에 접속을 해보도록 한다. 처음 접속을 하게 되면 아래와 같은 화면이 나오게 된다. 일단 Server Name 에는... localhost 혹은 . 혹은 (local)이라고 적으시면 된다.

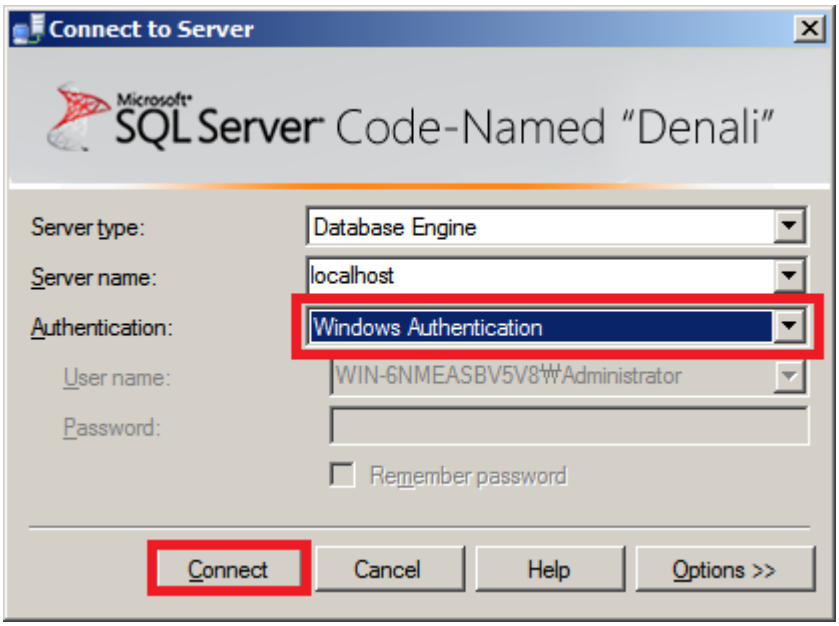
로그인 할 수 있는 계정에는 두가지가 있다. 바로 윈도우 계정과 SQL 계정이다.



혹시 설치 중에 아래화면이 있었다.



이전 단계에서 혼합모드로 설치를 했었기 때문에 지금은 윈도우 인증 모드와 SQL 인증 모두 모두가 가능하다. 윈도우 인증 모드로 하신다면, 아래 화면처럼 선택 후 Connect!!

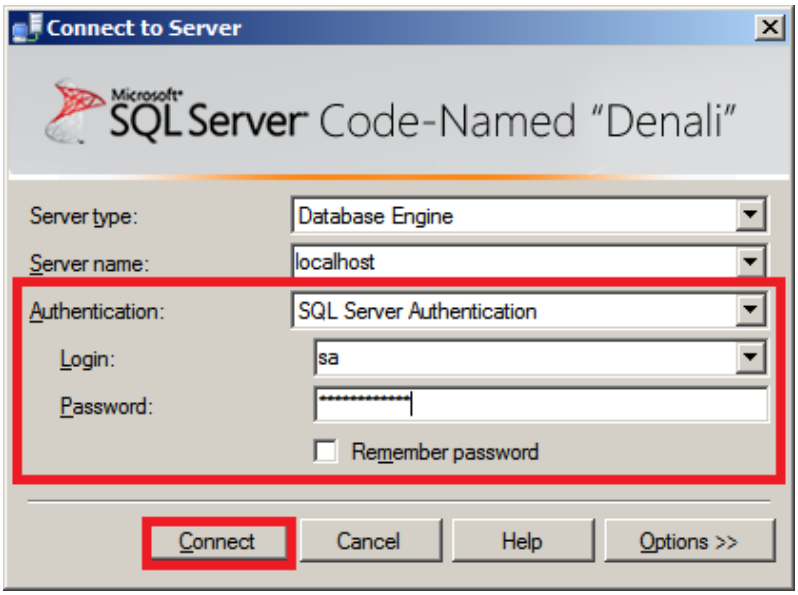


SQL 인증 모드로 하신다면...

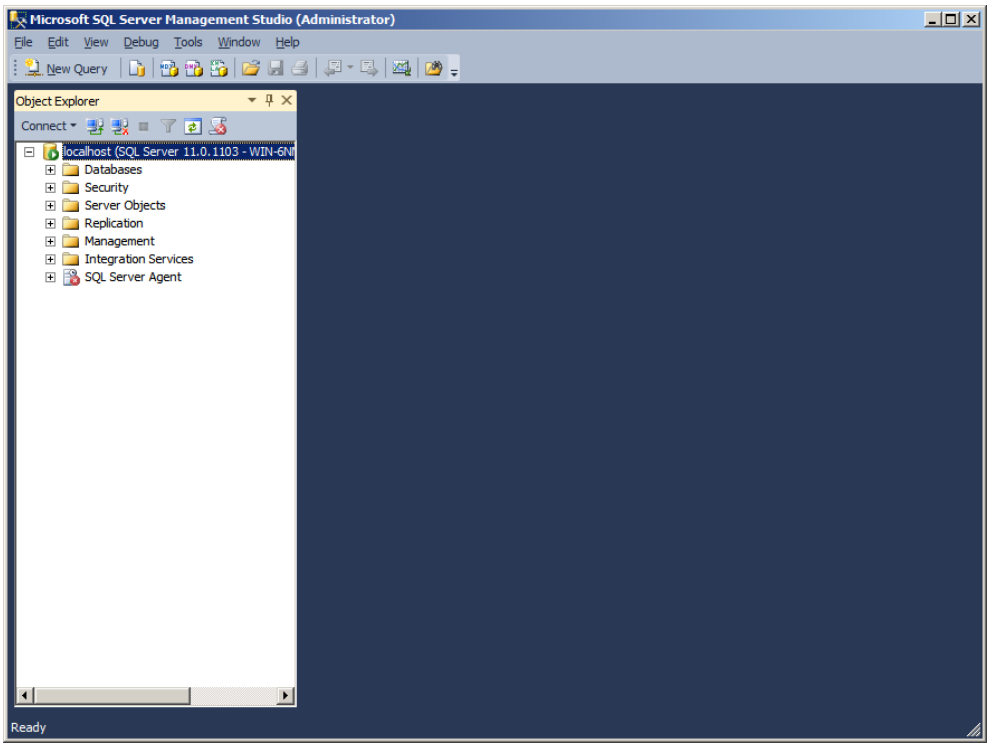
계정명: sa

비밀번호: 인스톨시 입력했던 비밀번호

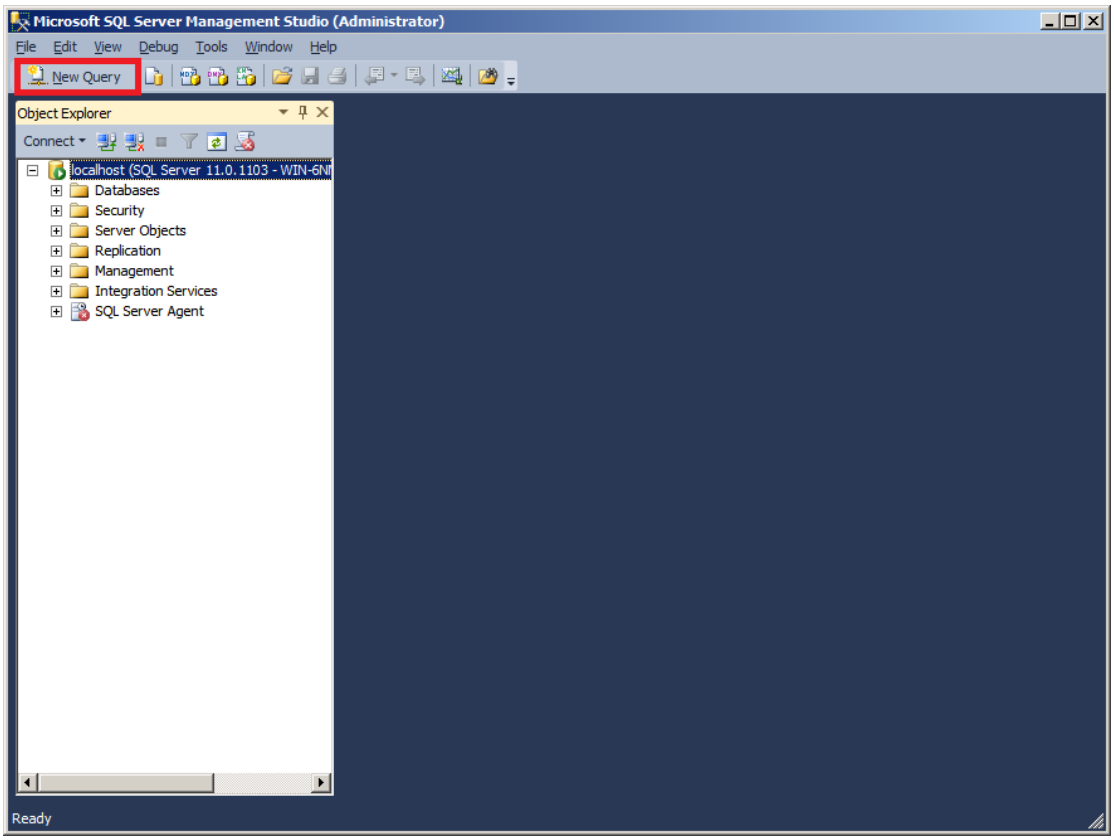
를 넣고 Connect 를 누르시면 된다.



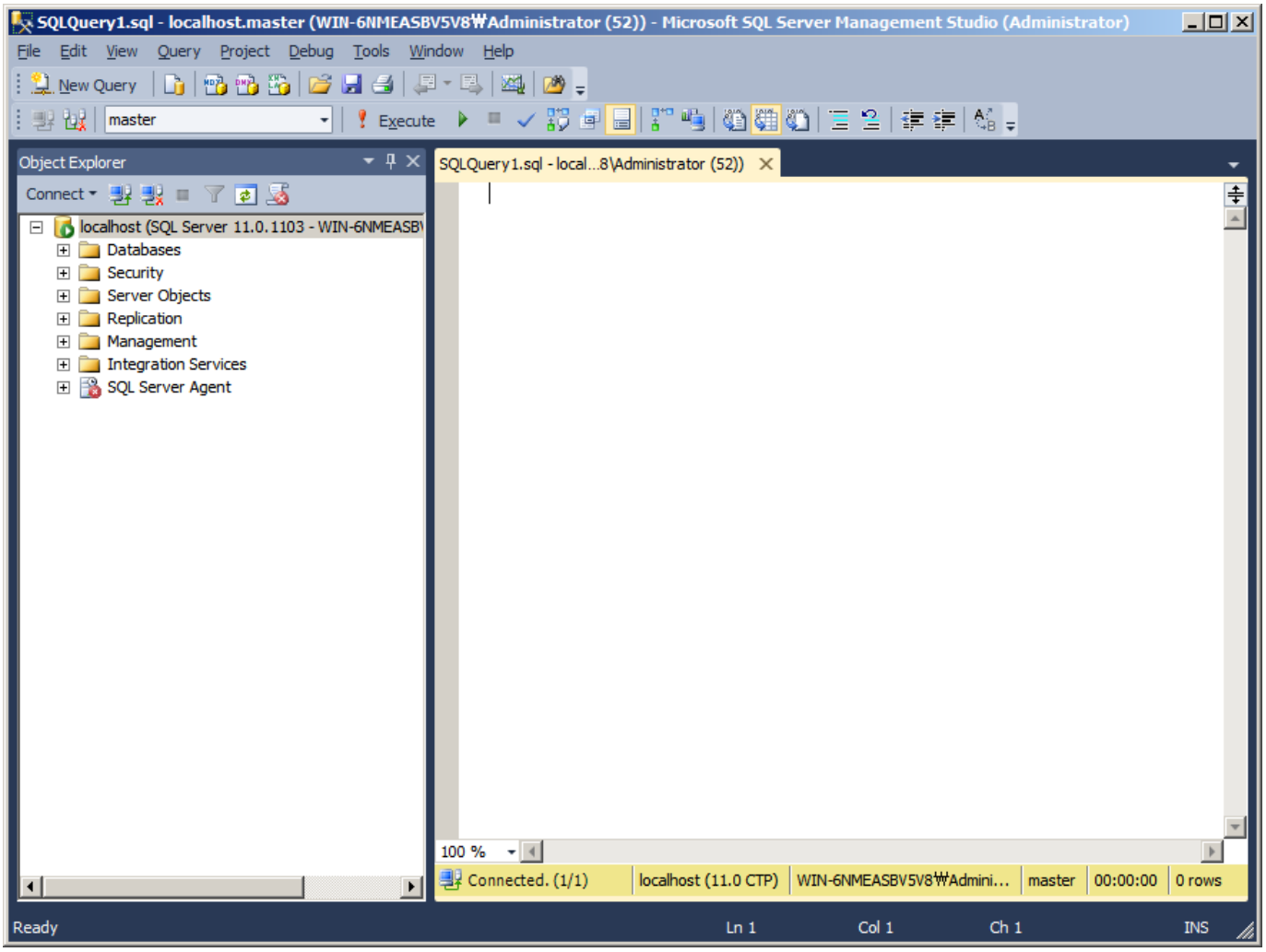
여기까지 했으면 아래와 같은 그림을 볼 수 있다. 바로 이게 SQL Server 에 정상적으로 접속이 된 상태이다. (참고로 저는 윈도우 계정 모드를 선택해서 로그인을 했다.)



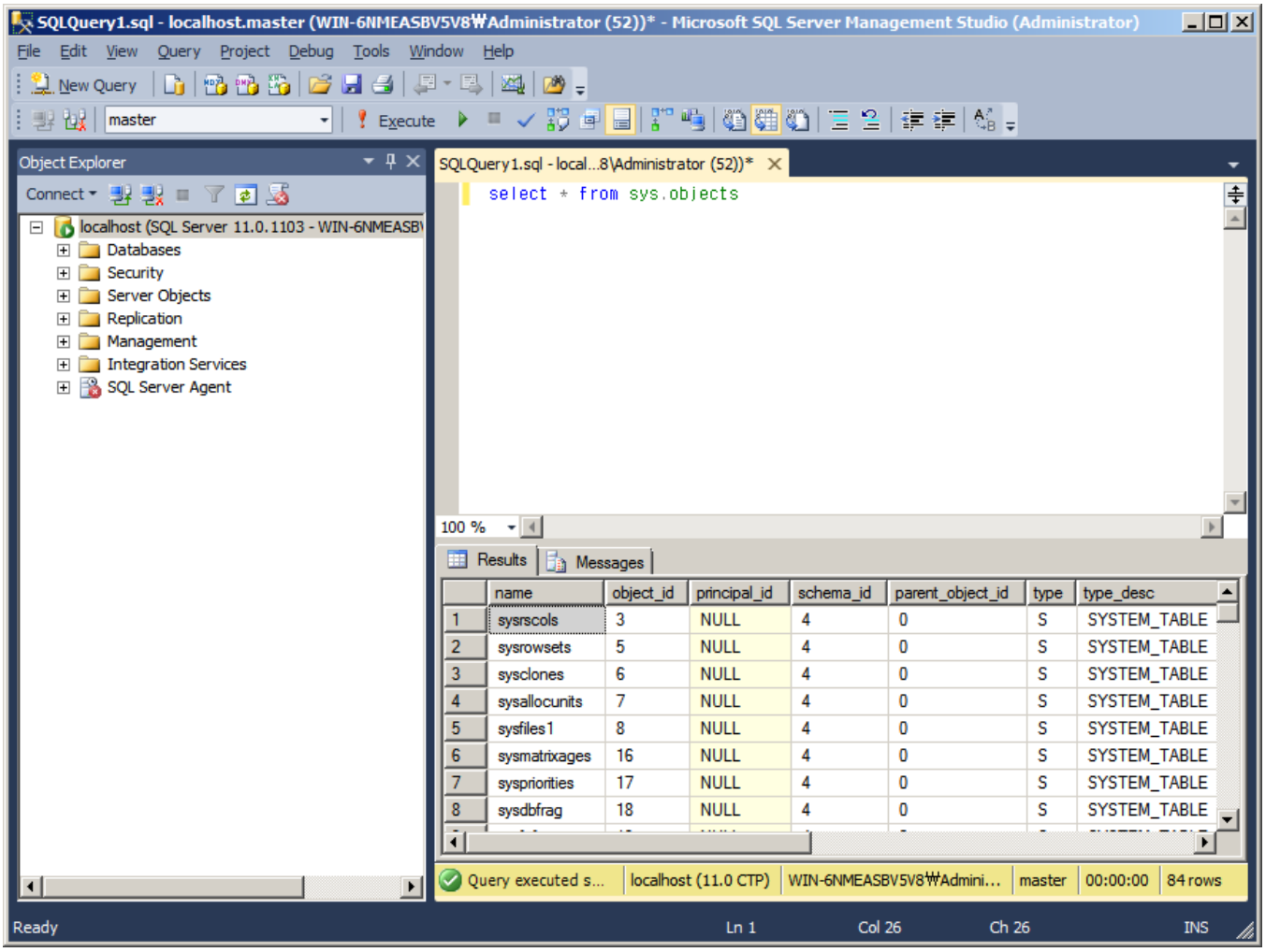
이제 New Query 를 선택해 본다.



오른쪽은 Query 를 이용해서 데이터를 가져오거나 입력, 수정, 삭제 할 수 있는 창이다.



select * from sys.obects 라는 쿼리를 입력해보도록 하겠다. 아래처럼 결과가 나왔다.



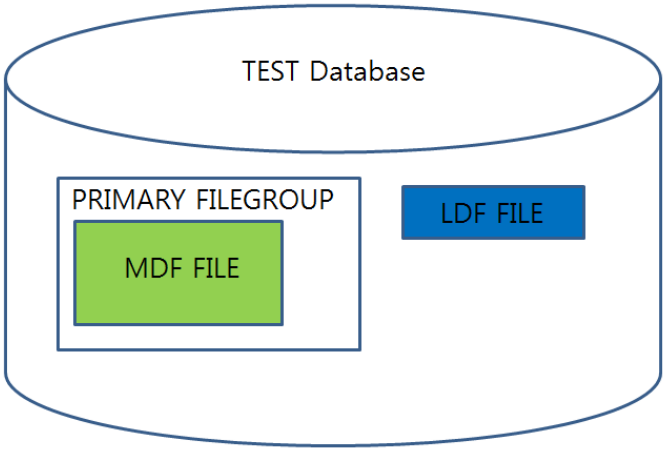
[SQL2011 강좌] 3. SQL Server 의 MDF, LDF 그리고 데이터베이스 생성하기

데이터 베이스에 데이터를 저장하기 위해서는 어떻게 해야 할까? SQL Server 는 기본적으로 MDF 확장자를 가진 파일에 데이터를 저장하게 된다. 또한 데이터베이스에 있어서 가장 중요한 것은 로그이다.

데이터가 변경되거나 수정되거나.. 삭제되었을때는.. 반드시..로그성 데이터를 기록하게 된다.
왜냐하면.. 정전이나 기타 물리적인 이슈로 데이터 손실을 방지하기 위해 데이터베이스는..
트랜잭션이라는 개념으로 데이터를 보존하게 된다. 트랜잭션에 대한 깊이 있는 설명은 나중에 하기로 하고 여기서는 간단히.. **데이터 손실을 막기 위한 수단**이라고 보면 된다.

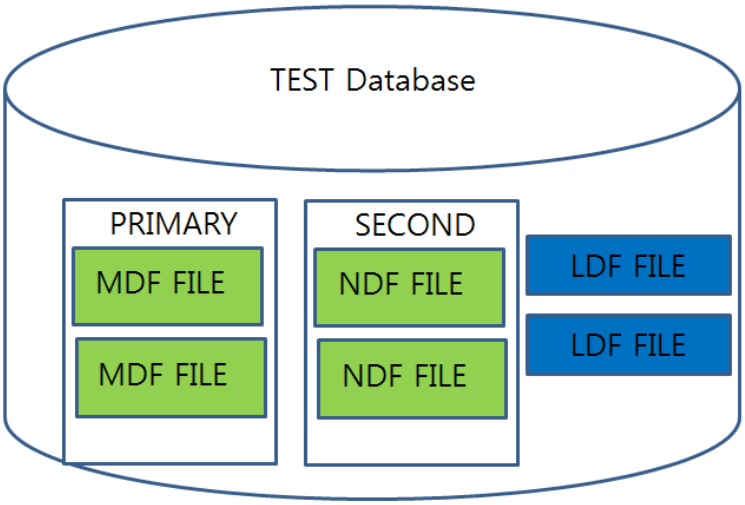
책이나 기타 강좌에서 가장 많이 드는.. 바로.. 은행에서 돈 이체시 정전이 된다면 어떻게 해야 할까??.. 물론 정전으로 인해서.. 빠져나간 돈은 다시 내 통장으로 들어오는 게 정상일 것 이다. 이 강좌 후반부에 트랜잭션에 대해서 자세히 다룰 예정이기 때문에 여기까지만 설명하고 어가도록 하겠다.

SQL Server 는.. 이런 데이터 손실을 방지하기 위해.. LDF 라는 확장자를 가진 곳에.. 데이터가 변경된 내역을 저장하게 된다. 따라서.. SQL Server 에서 데이터베이스를 생성하게 되면.. 기본적으로 MDF 와 LDF 하나씩은 만들어야 한다. 그리고.. MDF 는 파일 그룹에 속하게 된다. SQL Server 는 기본적으로 PRIMARY 라는 파일 그룹이 있다. 물론 MDF 가 한 개 이상의 파일그룹을 가질 순 없고 무조건 1:1 로만 가능하다. 이걸 그림으로 그리면 아래와 같은 그림이 된다.

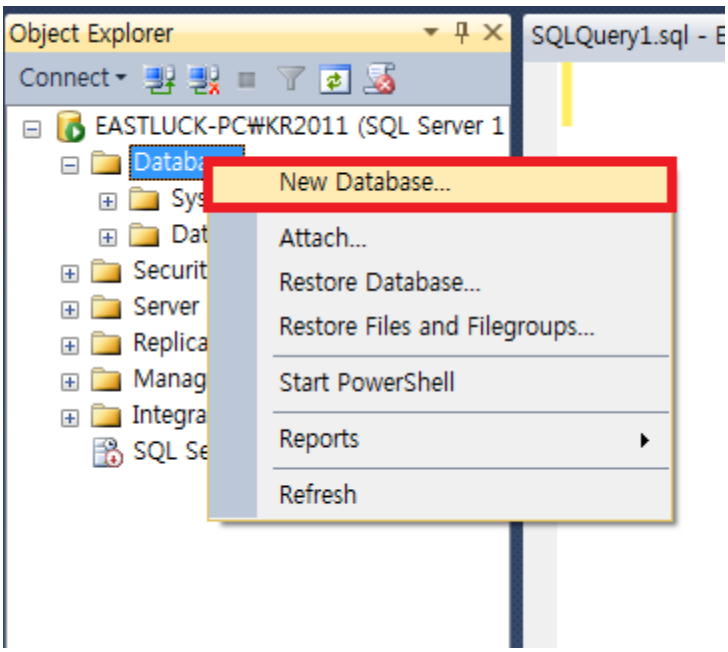


파일 그룹이 두 개가 이상 될 수도 있고... MDF 파일이 여러 개가 될 수 있다.

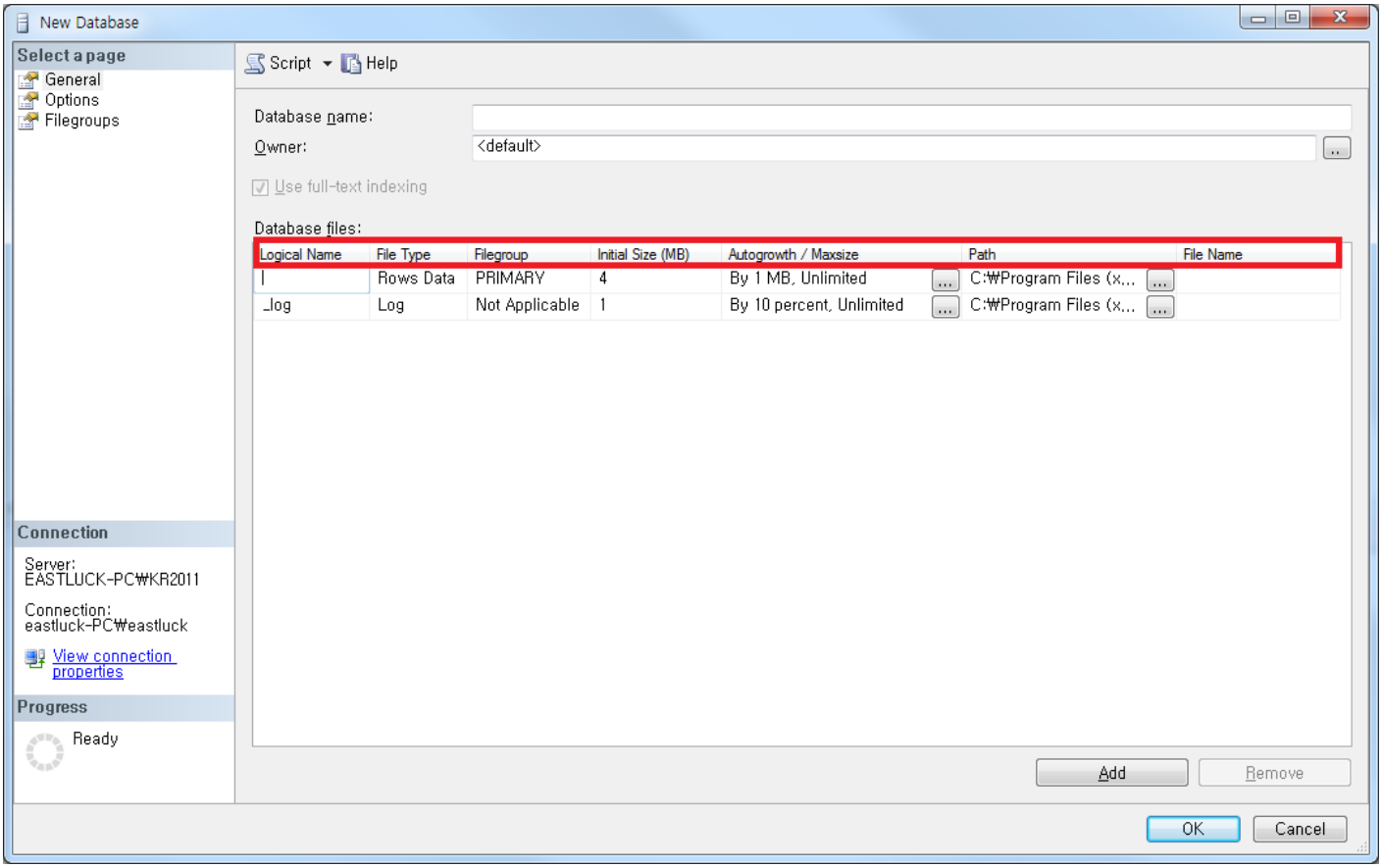
MDF 가 두 개 이상이 되면.. MDF 말고 NDF 로 만들어 집니다. 아래와 같은 그림도 될 수 있다.



그럼 데이터 베이스를 한번 만들어볼까?? 우선 저번 강좌에 설명 드린 데이터 베이스에 접속을 하신 뒤에 Database 항목에 우클릭 => New Database 클릭!



위와 같이 하시면 아래와 같은 창이 나타나게 된다.



Logical Name: 논리적인 이름

File Type: Rows Data(MDF 파일), Log(LDF 파일)

Filegroup: 파일 그룹(기본값 PRIMARY)

Initial Size (MB): 기본적인 파일 사이즈(단위: MB)

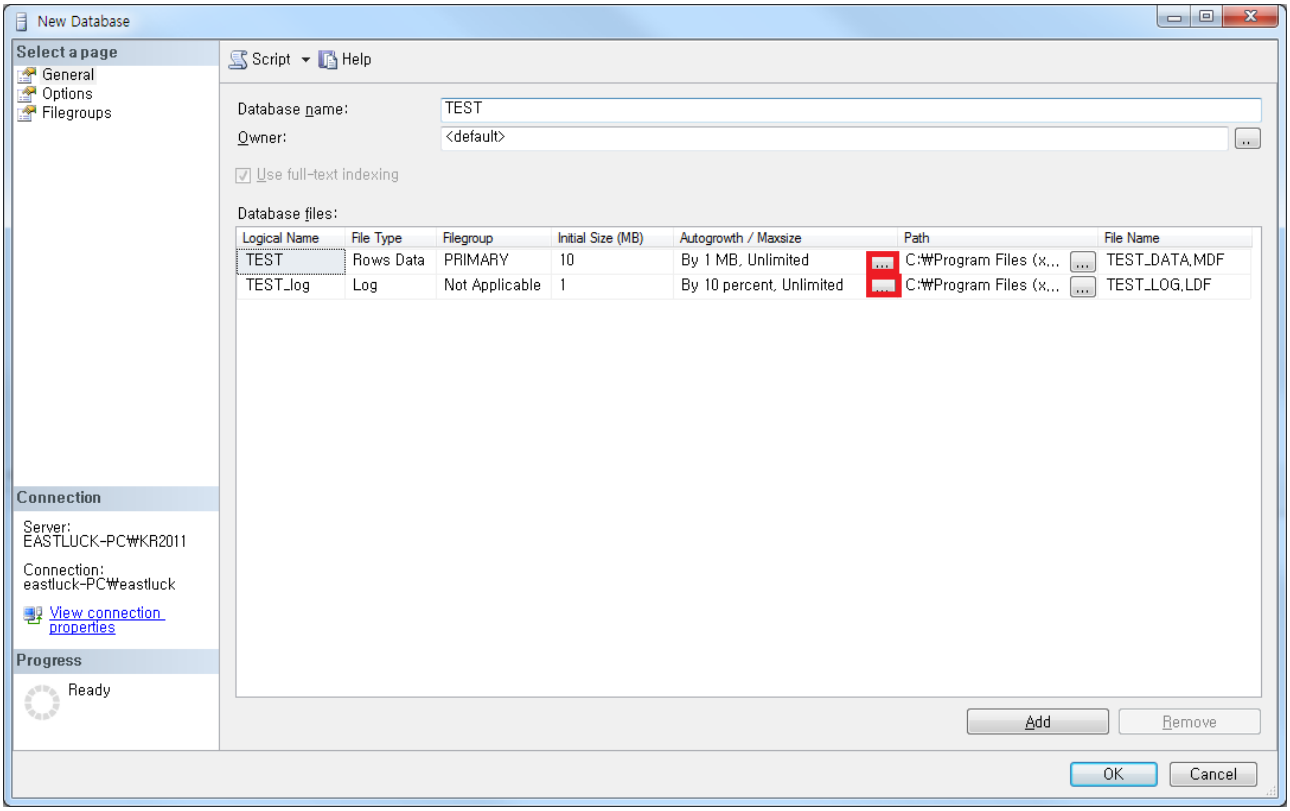
Autogrowth / Maxsize: 자동 증가 설정 및 최대 사이즈 설정 값

Path: 실제 파일이 위치할 경로

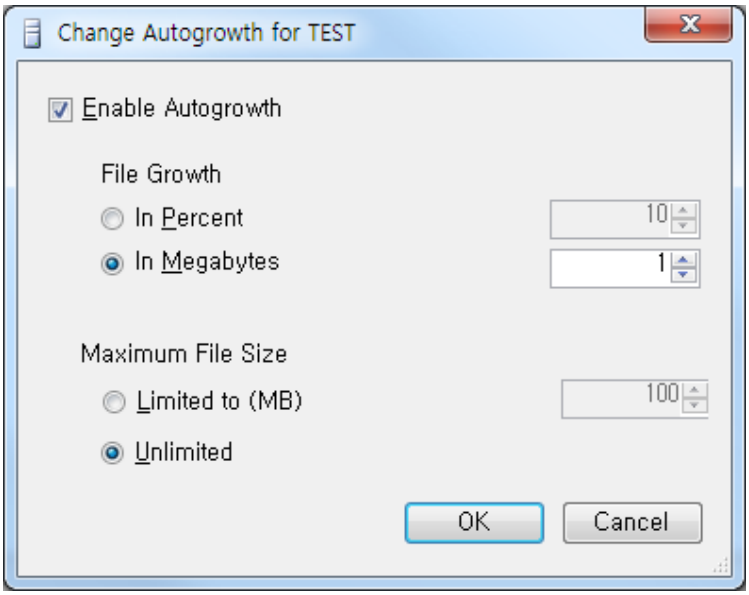
File Name: 실제 파일 이름 이다.

아래 그림처럼

Database name 과 File Name 을 입력하시고.. 빨간색으로 처리된 부분을 선택해 본다.



그러면 아래와 같은 박스가 나오게 된다.



Enable Autogrowth: 자동증가 활성 여부

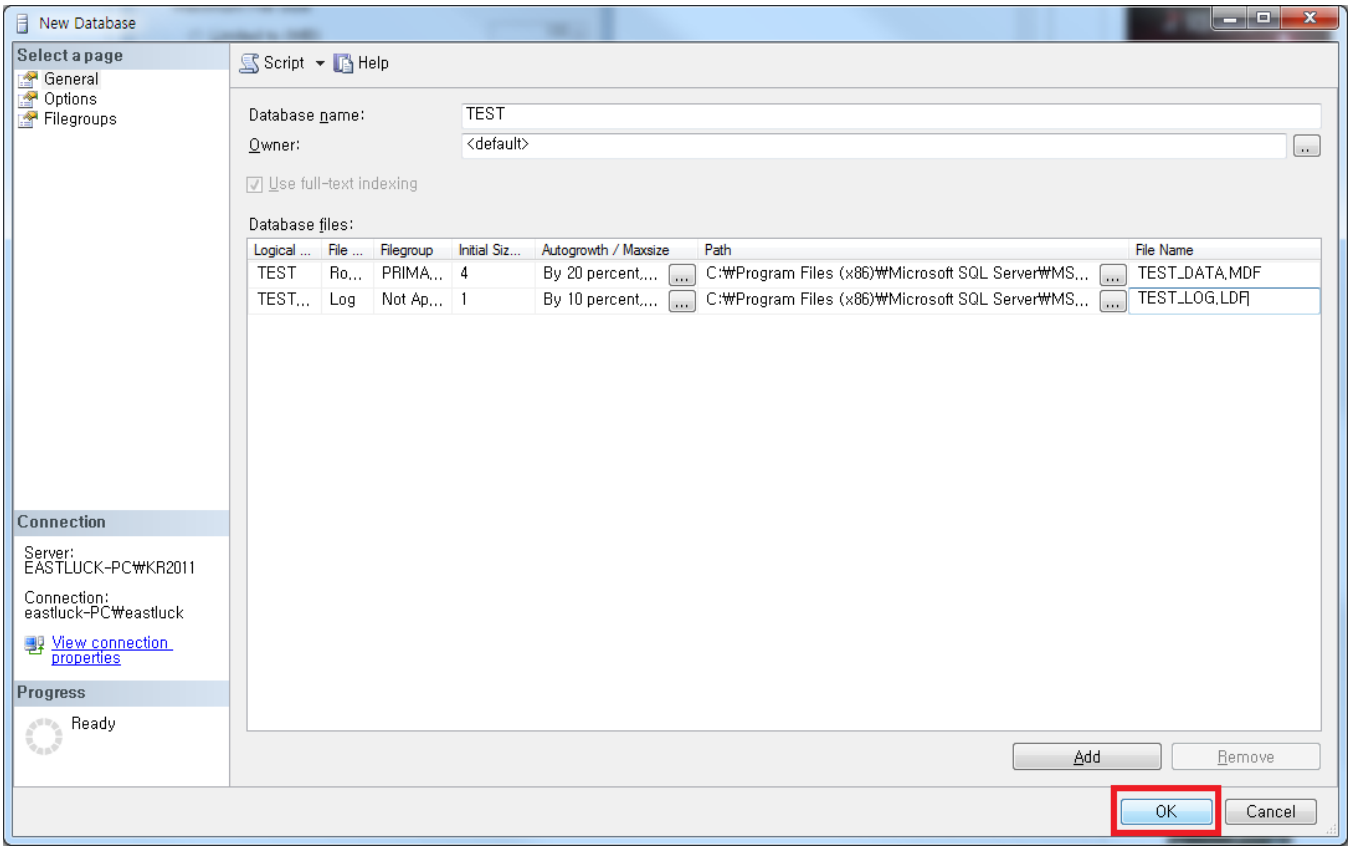
File Growth: 파일 증가 방법

1. In Percent: 퍼센트
2. In Megabytes: MB 량으로 늘리기

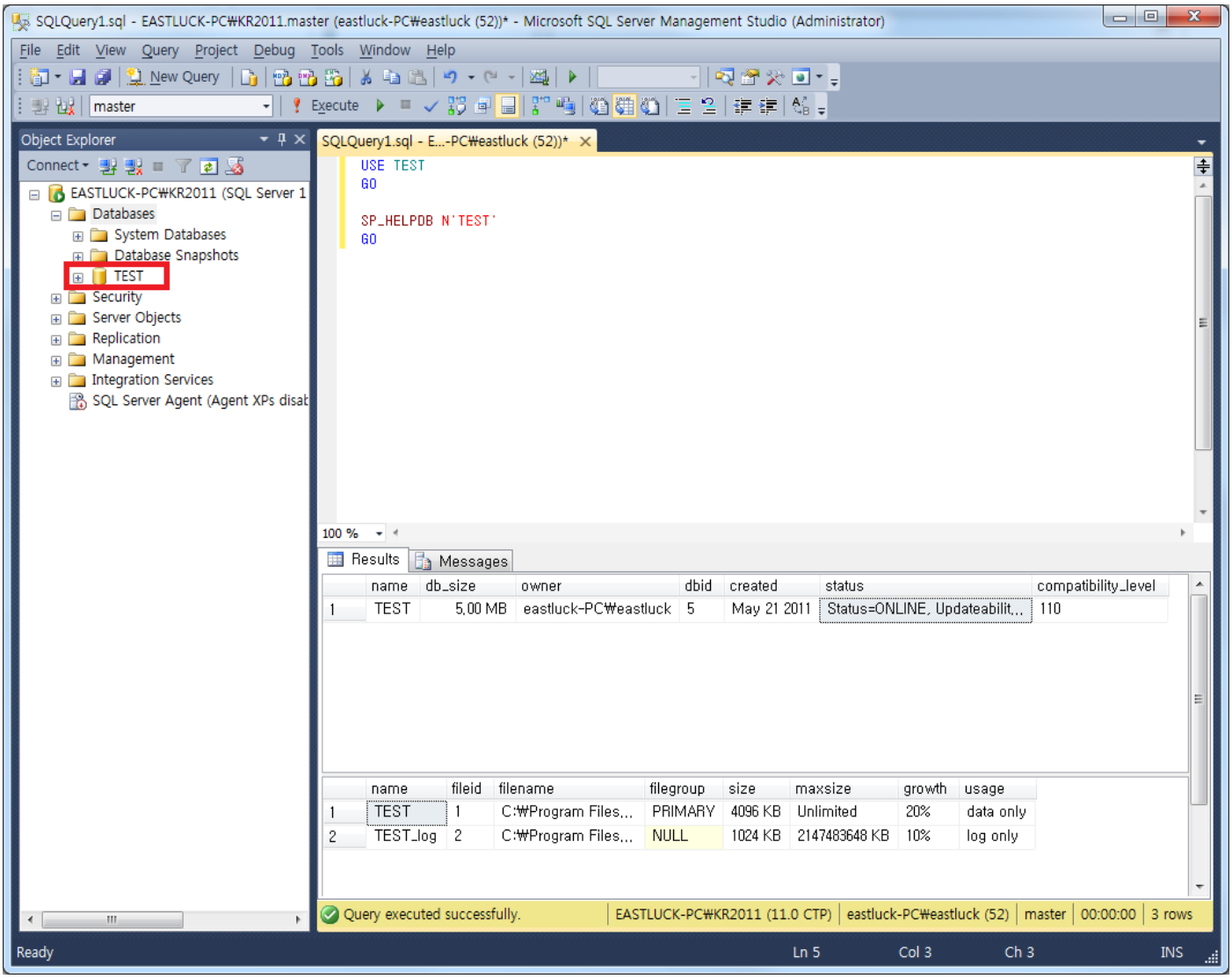
Maximun File Size: 최대 파일 사이즈

1. Limited to (MB): 지정된 파일 사이즈
2. Unlimited: 제한 없음

MDF 에 대해서 20% 씩 파일 자동 증가와 제한 없음 LDF 에 대해서는 10% 씩 파일 자동 증가와 제한 없음을 선택하도록 하겠다. 자 이제 모든 준비가 끝났으니.. OK 를 눌러서.. DATABASE 를 생성해보도록 하겠다!



데이터 베이스가 정상적으로 만들어졌다. New Query 를 이용해서.. 아래 쿼리로.. 만들어진 데이터 베이스 정보를 볼 수 있다.



물론 Query 로도 데이터베이스를 만들 수 있다. 따로 옵션 부분들은 위에 모두 설명 드렸기 때문에 설명 드리지 않겠다. 살짝만 눈여겨 보시기 바란다.

```
CREATE DATABASE TestQuery ON
(
        NAME = N'TestQuery'
,       FILENAME = 'C:₩TestQuery_DATA.MDF'
,       SIZE = 4MB
,       MAXSIZE = UNLIMITED
,       FILEGROWTH = 1MB
)
LOG ON
(
        NAME = N'TestQuery_log'
,       FILENAME = 'C:₩TestQuery_LOG.LDF'
,       SIZE = 1MB
,       MAXSIZE = 1024GB
,       FILEGROWTH = 10%
);
```

한가지 추가로 설명하자면 DATABASE 를 가장 쉽게 만들 수 있는 방법이 있다. 바로.. 아래의 쿼리로 만드실 수 있다. 이 방법은.. system db 중에서 model 디비가 붕어빵이 되서 만들어진다.

system db 가 무엇인지는 나중에 기회되면 설명 드리겠다.

```
CREATE DATABASE 데이터베이스명
GO
```

[SQL2011 강좌] 4. DDL 1 탄: 테이블 생성

이번 강좌는 바로 DDL 1 탄 테이블 생성이다.

DDL 은 Data Definition Language 의 약자로서..

1. 데이터베이스 객체 생성(CREATE)

2. 데이터베이스 객체 수정(ALTER)

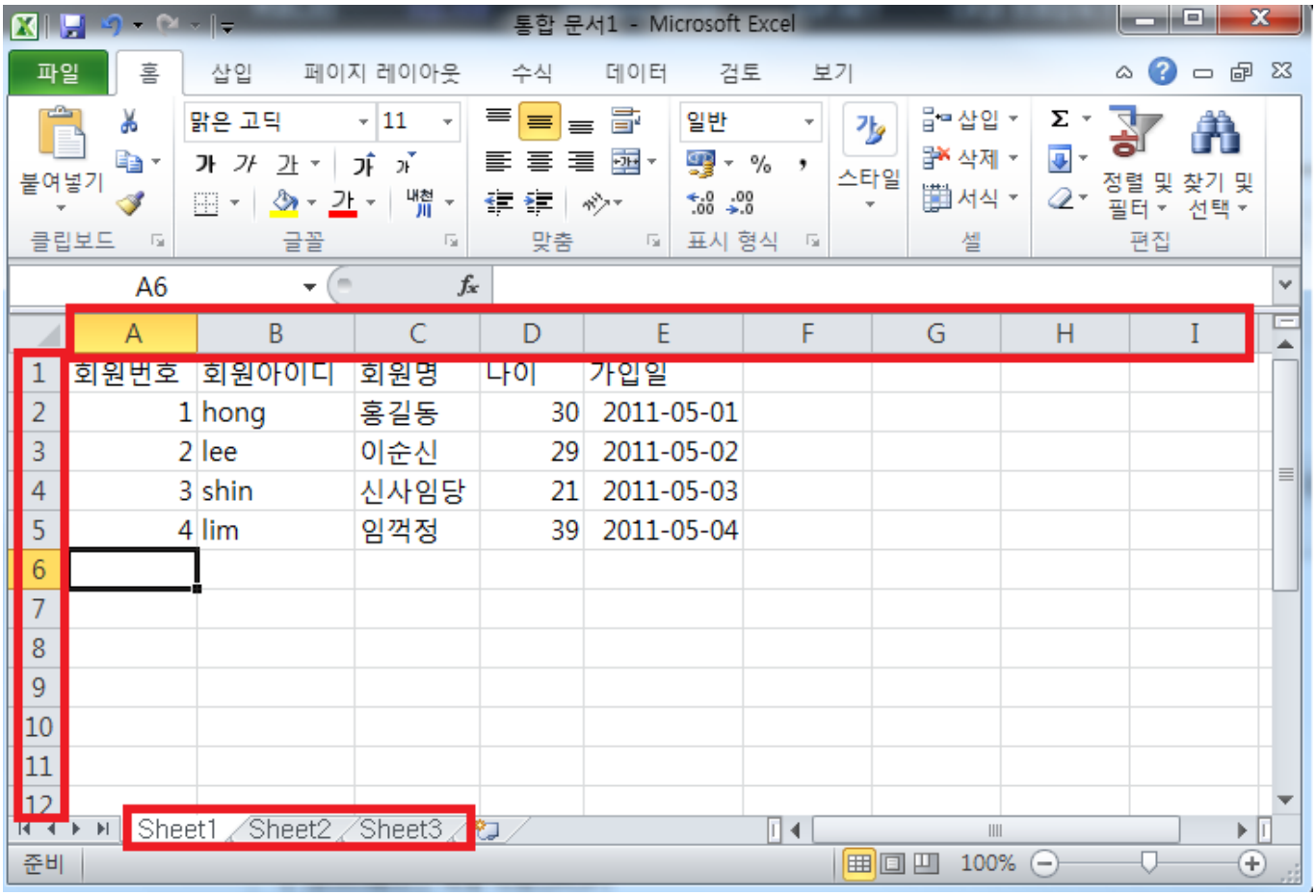
3. 데이터베이스 객체 삭제(DROP)

4. 데이터베이스 이름 변경(RENAME)

5. 테이블 초기화(TRUNCATE)

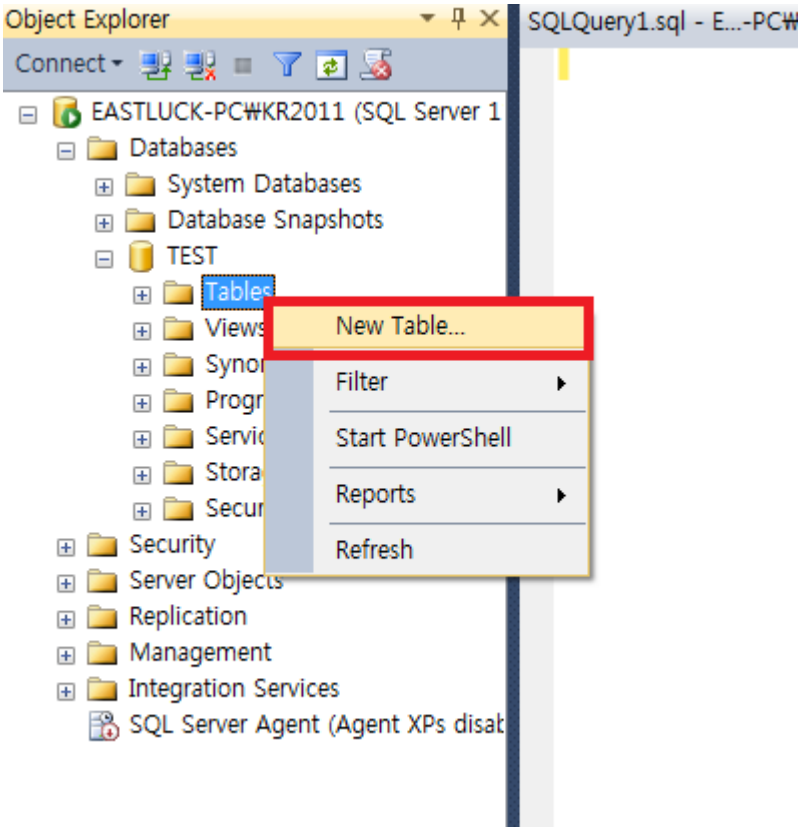
이렇게 5 가지로 구분이 된다. 이 중에서 우리가 이번에 살펴봐야 할 부분은.. 생성 부분 중에서 **테이블**이다. SQL Server 에서 테이블은 뭘까?? 쉽게 생각하시면 엑셀로 생각하시면 된다.

이번에 새로 출시된 Microsoft Office 2010 이다. 2007 과 거의 흡사하게 디자인이 되었으며.. 많은 기능들이 추가되었다.



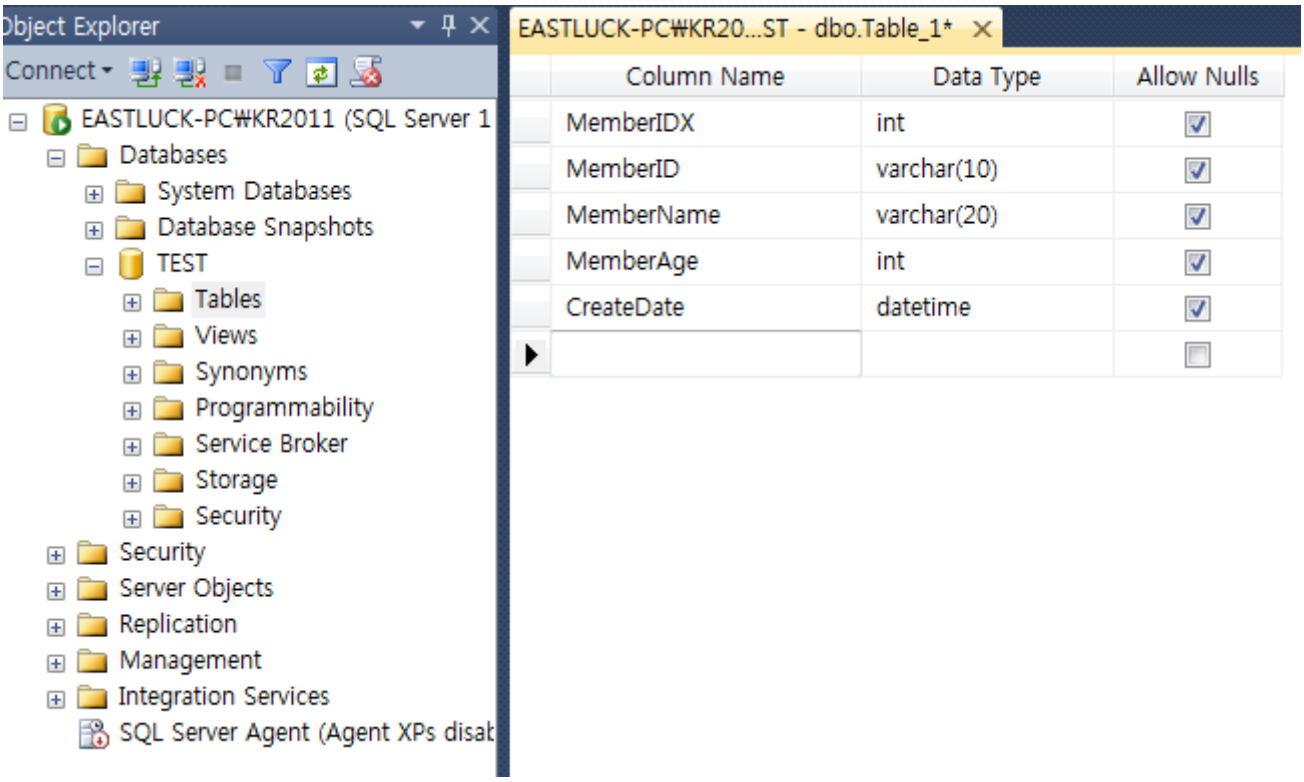
열이라는 속성을 가지고 있으며 행이라는 데이터를 가지고 있다.물론 각 시트들은 하나하나의 테이블이 된다.

SSMS 접속 => 데이터베이스 => Table 우클릭 => New Table



New Table 을 클릭하시면 아래와 같은 화면이 나옵다.

제가 입력한 항목에 맞게 순서대로 입력해 본다.



- Column Name: 컬럼 명을 기술한다. 한글도 가능하지만 되도록 영문을 쓰며, 특수문자 사용을 하지 않는 것이 좋다.
- Data Type: 데이터 타입은 정말 여러가지가 있다. 일단 저희가 이번에 다루는 데이터 타입은 int, varchar, datetime 이 3 가지 이다.
- int: 약 -21 억 ~ 21 억 까지의 숫자를 저장할 수 있는 4 바이트 정수이다.
  (추가로 알아두면 좋은 데이터 타입: smallint: 약 -3 만 2 천 ~ 3 만 2 천, tinyint: 0~255)
- varchar: 가변 길이의 문자열을 저장 괄호안의 숫자로 넣을 수 있는 숫자를 조절할 수 있다.

숫자는 바이트이지만.. 한글은 2 바이트이기 때문에.. varchar(10)으로 지정하게 되면, 영문/숫자 10 자 또는 한글은 최대 5 자가 들어갈 수 있다.

(추가로 알아두면 좋은 데이터 타입: nvarchar: 가변 길이 유니코드형 데이터 저장, char: 고정 길이 문자열 저장, nchar: 고정 길이 유니코드형 데이터 저장)

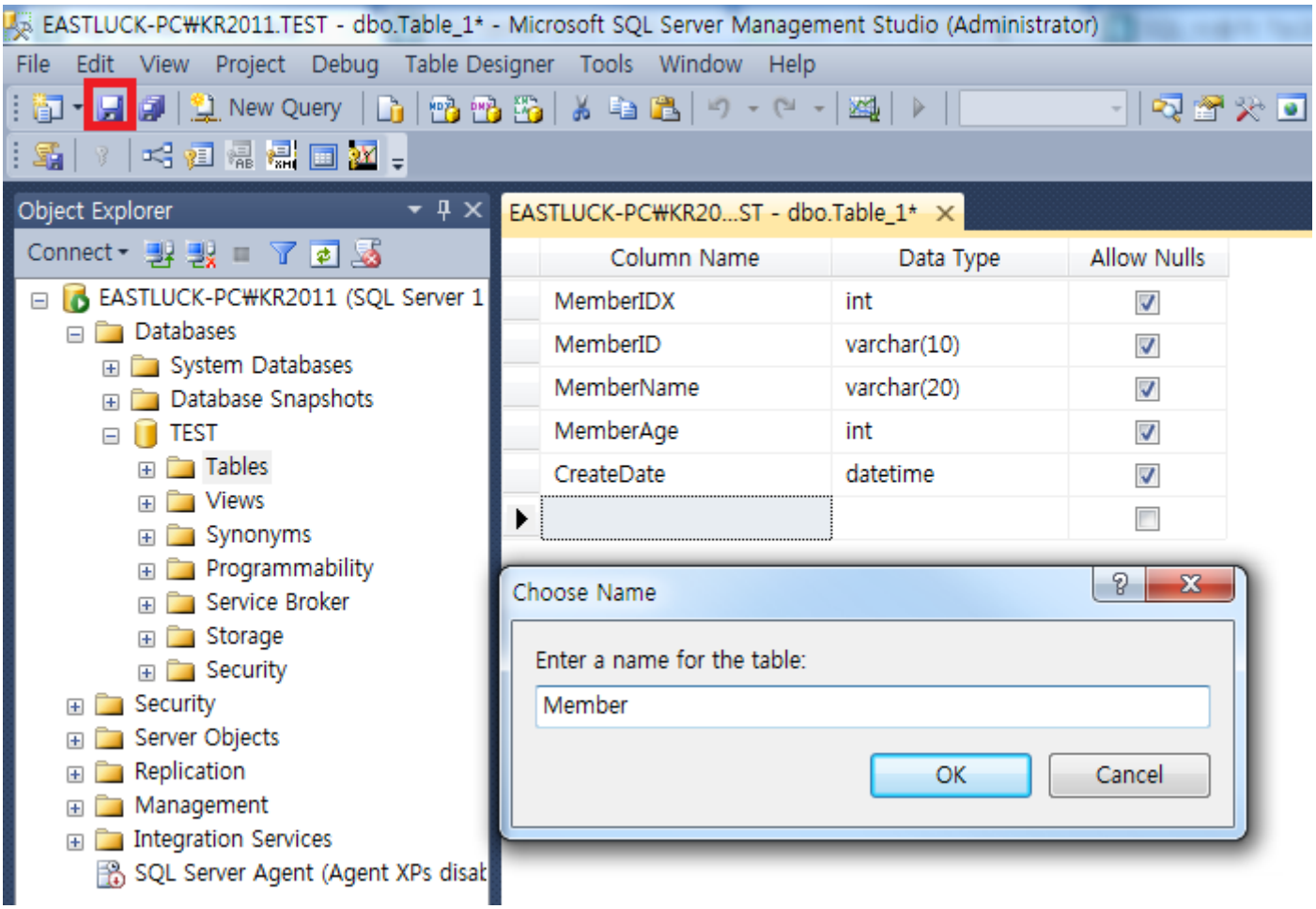
- datetime: 2011-05-11 05:11:11.997 처럼... 밀리세컨드까지 저장할 수 있는 데이터 타입이다.

데이터 형식에 대해서 자세한 내용은 아래 링크 참고하시기 바란다.

데이터 형식(Transact-SQL)

http://msdn.microsoft.com/ko-kr/library/ms187752.aspx

입력이 다 끝나셨으면.. 오른쪽 상단에 있는 저장 또는 (Ctrl+S)를 클릭해서 테이블명 Member 라고 입력하고 확인을 선택한다



아직 INSERT 를 배우진 않았지만.. 쿼리를 이용해서 데이터를 입력하고.. 내용을 확인해보도록 하겠다.

좌측 상단의 New Query 를 클릭하고! 아래의 쿼리를 입력한다.

use TEST

go

insert into Member values(1,'hong', '홍길동', 30, '2011-05-01')

insert into Member values(2, 'lee', '이순신', 29 , '2011-05-02')

insert into Member values(3, 'shin', '신사임당', 21, '2011-05-03')

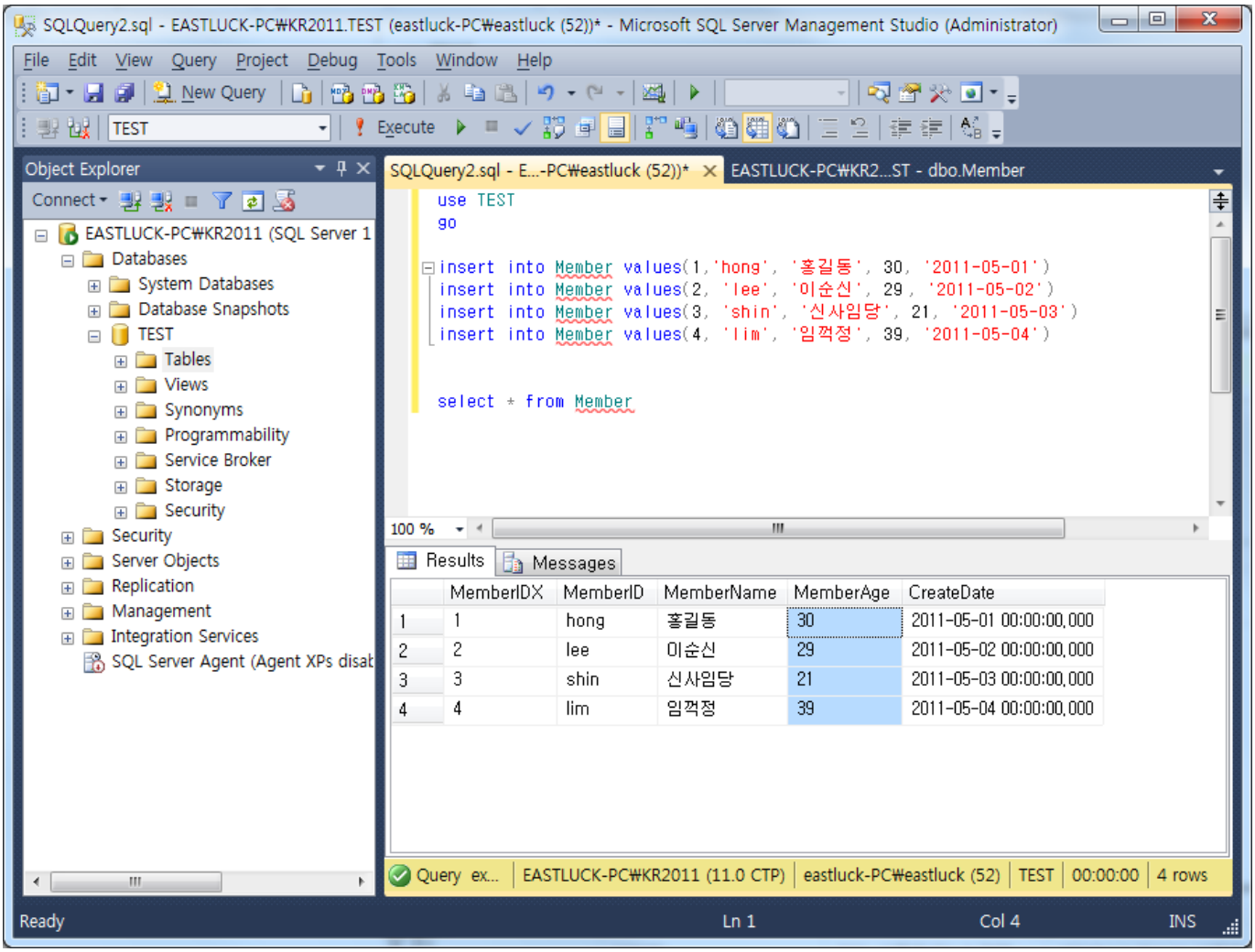
insert into Member values(4, 'lim', '임꺽정', 39, '2011-05-04')

go

select * from Member

go

아래처럼 원하는 결과를 볼 수 있다.



아까 엑셀에서 봤던 모습과 흡사하다. 이렇게 테이블을 생성하고 테스트 데이터를 넣어보고, 조회를 해봤다. 테이블 생성은 Query 로 가능하다.

아주 간단하게 아래처럼 작성할 수 있다.

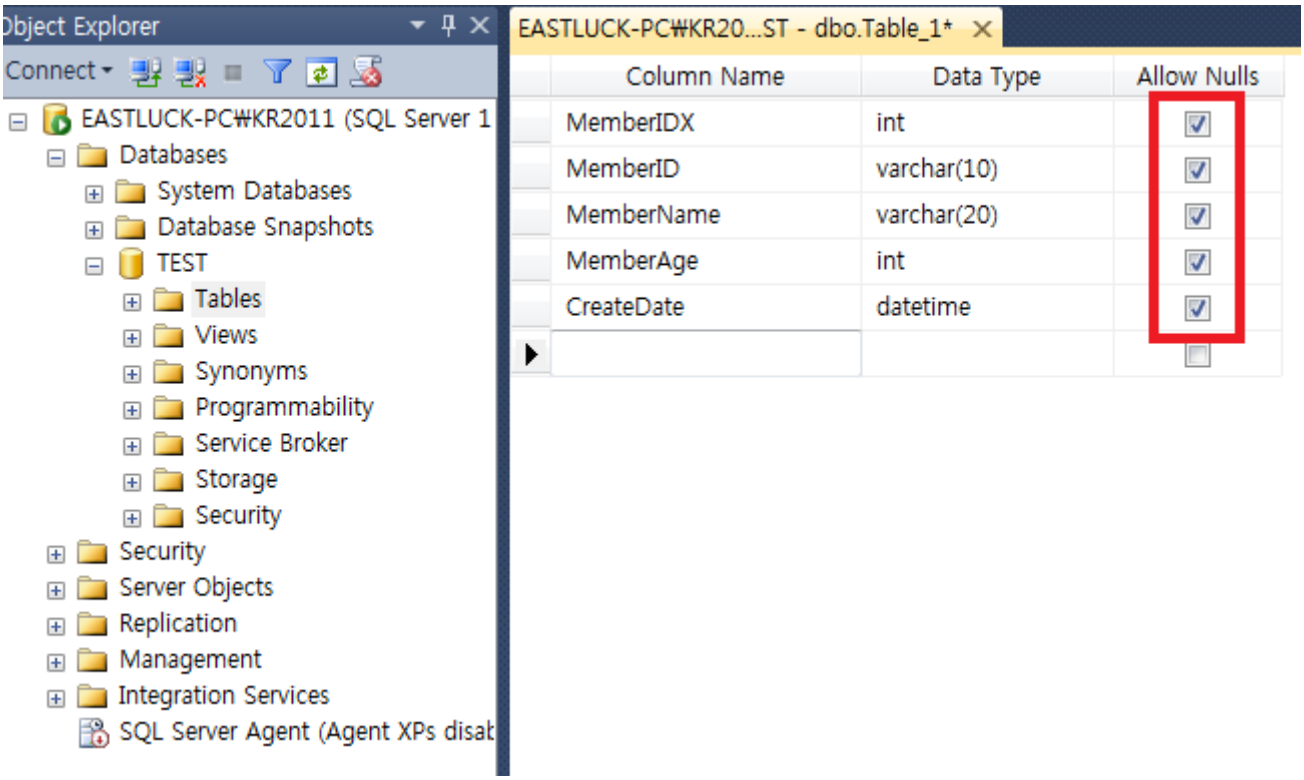
```
CREATE TABLE MemberQuery
(
        MemberIDX     INT
,       MemberID              VARCHAR(10)
,       MemberName VARCHAR(20)
,       MemberAge     INT
,       CreateDate            DATETIME
)
```

[SQL2011 강좌] 5. DDL 2 탄: 쿼리를 이용한 테이블 컬럼 추가 및 삭제

이번 강좌에서는 저번 강좌에서 추가한 Member 테이블을 가지고 컬럼을 추가, 삭제, 컬럼명을 변경하는 법에 대해서 알아보도록 하겠다. 물론 추가로 테이블명도 변경을 해보도록 하겠다. 전에 말씀 드렸다시피.. 데이터베이스 객체를 수정(ALTER) 하거나 삭제(DROP)하는 작업이기 때문에 DDL 에 속하게 된다. 물론 이름변경(RENAME)도 DDL 에 속한다. 일단 강좌에 들어가기 앞서 NULL 에 대해서 잠깐 얘기해볼까 한다. 테이블의 컬럼에는 NULL 이 가능/불가능 할 수 있다.

그럼 데이터베이스에서 말하는 이 NULL 이라는 값은 뭘까? NULL 의 정의를 정리해봤다.

1. 알 수 없는 값
2. 사용할 수 없는 값
3. 적용할 수 없는 값
4. 0 과 공백('') 과는 다른 값
5. 산술식과 비교하거나 연산을 하면 무조건 NULL



위의 체크박스에 **체크를 하게 되면 NULL 허용, 체크를 하지 않으면 NULL 불가한** 된다. MemberIDX 컬럼을 NULL 불가로 바꾼다면 체크를 없애고 저장을 하시면 된다만 쿼리를 이용하게 되면 아래처럼 입력해야 한다.

```
ALTER TABLE Member
ALTER COLUMN MemberIDX INT NOT NULL
GO
```

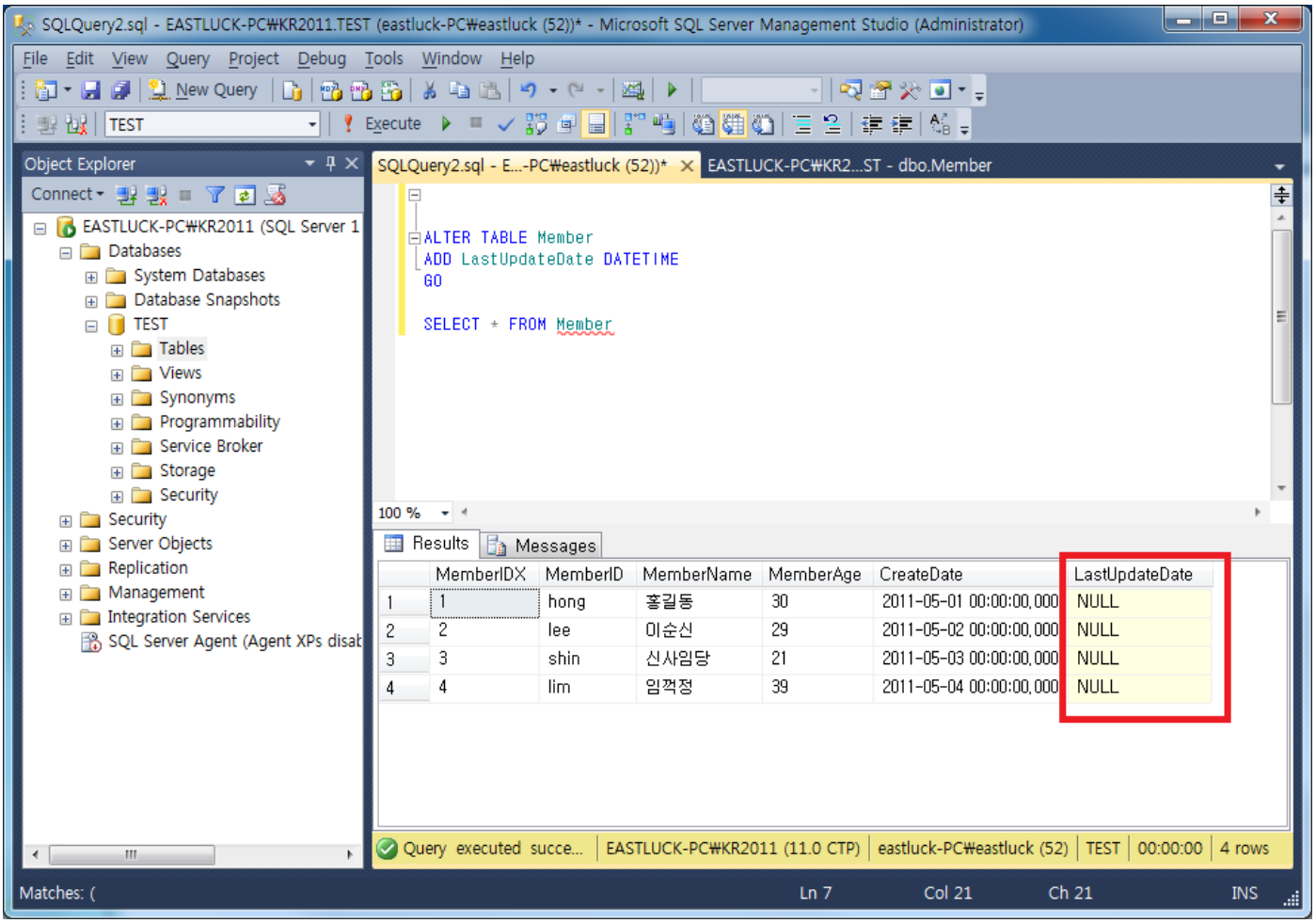
주의:이 작업은 모든 데이터 행을 수정하기 때문에 데이터가 많은 경우 무척 위험한 행동이 될 수 있다!

하지만 저희는 테스트로 해보는 것이기 때문에 이런 문제는 걱정 안하셔도 된다.^^

이제 컬럼을 한번 추가해볼까??

최종 수정일이라는 LastUpdateDate 라는 컬럼을.. DATETIME 형으로 추가해보도록 하겠다.

```
ALTER TABLE Member
ADD LastUpdateDate DATETIME
GO
SELECT * FROM Member
GO
```

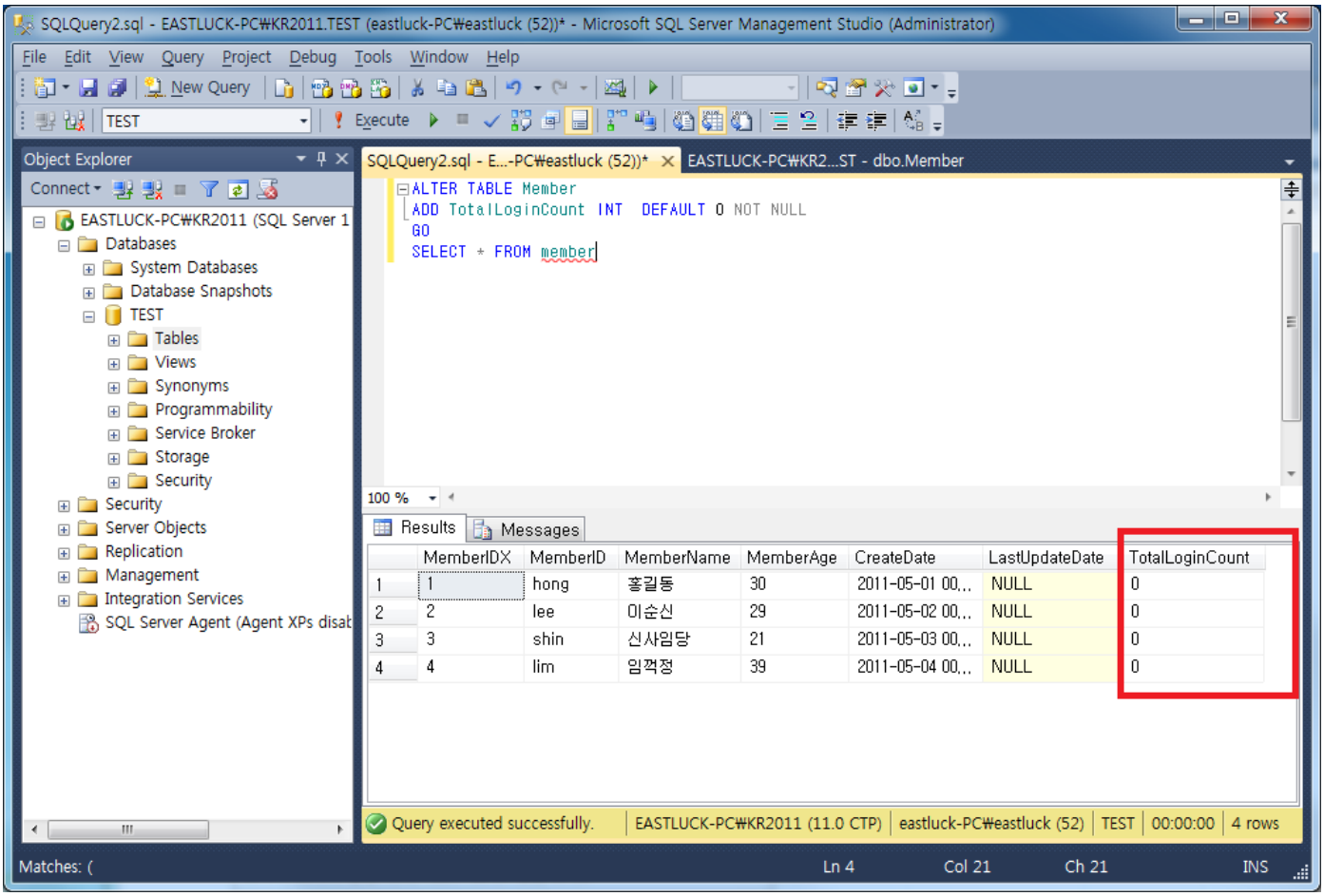


빨간 부분이 추가가 된 것을 확인할 수 있다. 혹시 기본 값을 지정할 순 없을까? 바로 디폴트 값을 이용하면 된다. TotalLoginCount 라는 int 타입을 추가하고 기본 값으로 0 을 준다.

ALTER TABLE Member

ADD TotalLoginCount INT DEFAULT 0 NOT NULL

GO



TotalLoginCount 가 0 으로 초기화 되면서 추가된 것을 볼 수 있다.

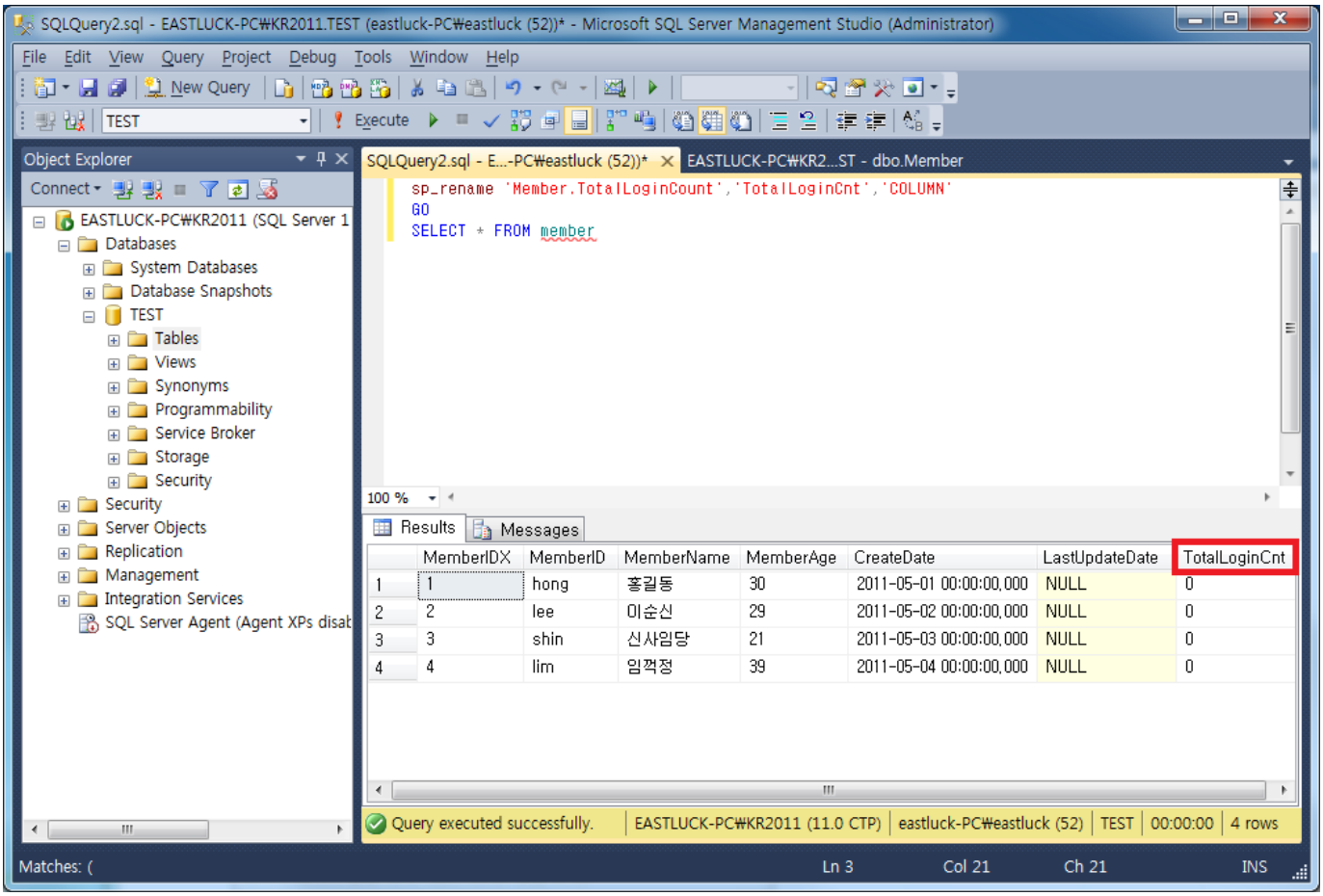
이제 컬럼명을 바꿔 볼까?? TotalLoginCount 컬럼명을.. TotalLoginCnt 로 변경해 본다.

sp_rename 'Member.TotalLoginCount','TotalLoginCnt','COLUMN'

--// sp_rename '테이블명.컬럼명','컬럼명','COLUMN'

GO

SELECT * FROM member



컬럼명이 성공적으로 변경 된 것을 볼 수 있다. sp_rename 에 관련해서는.. 아래 링크를 참고해 보기를 바란다.

sp_rename(Transact-SQL): http://msdn.microsoft.com/ko-kr/library/ms188351.aspx

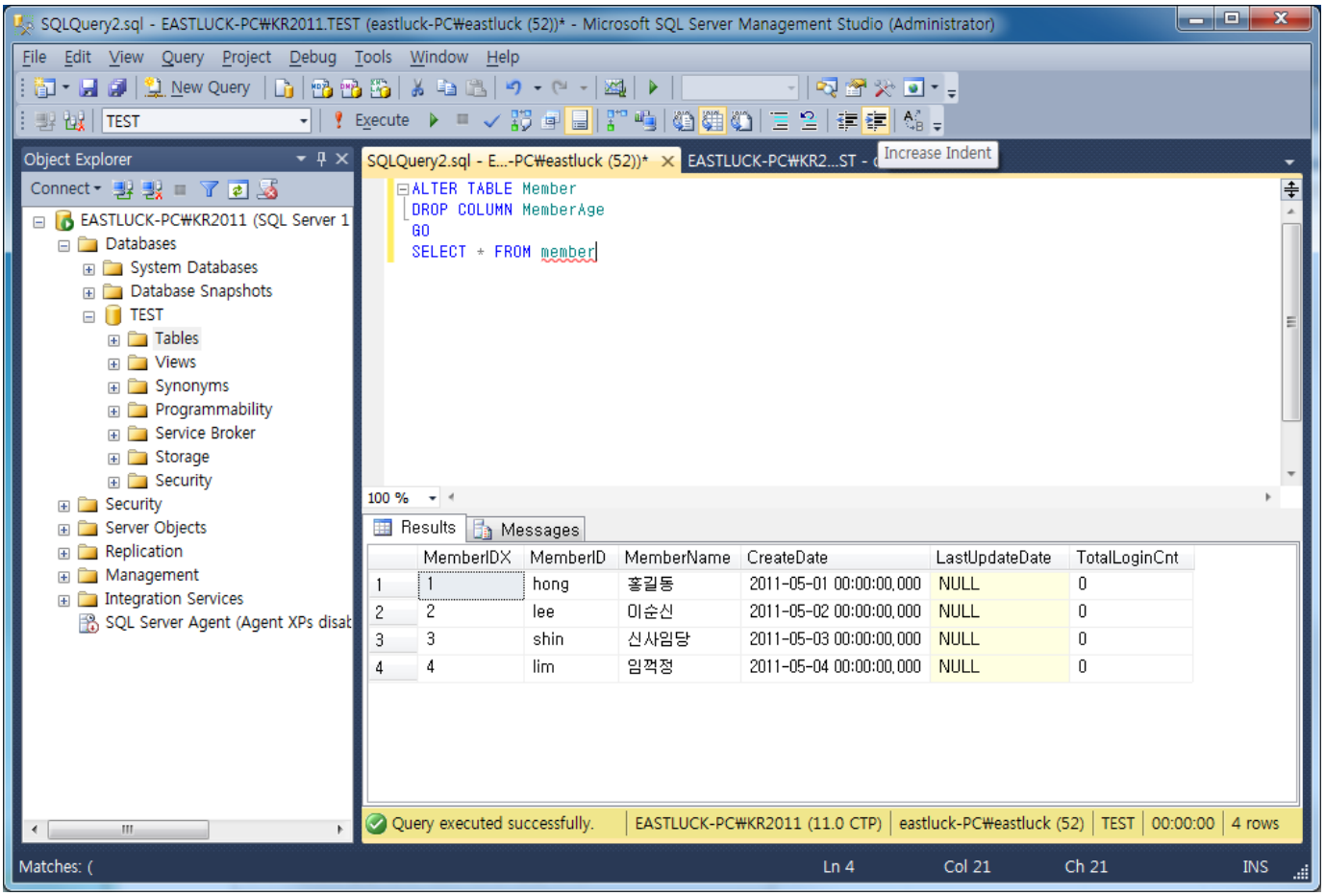
MemberAge 라는 컬럼을 삭제해보도록 하겠다.

ALTER TABLE member

DROP COLUMN MemberAge

GO

SELECT * FROM member



이렇게 컬럼이 삭제된 것을 볼 수 있다.

이제 TotalLoginCnt 라는 컬럼을 삭제해보도록 하겠다.

일단 에러 내용을 한국어로 보기 위해서..

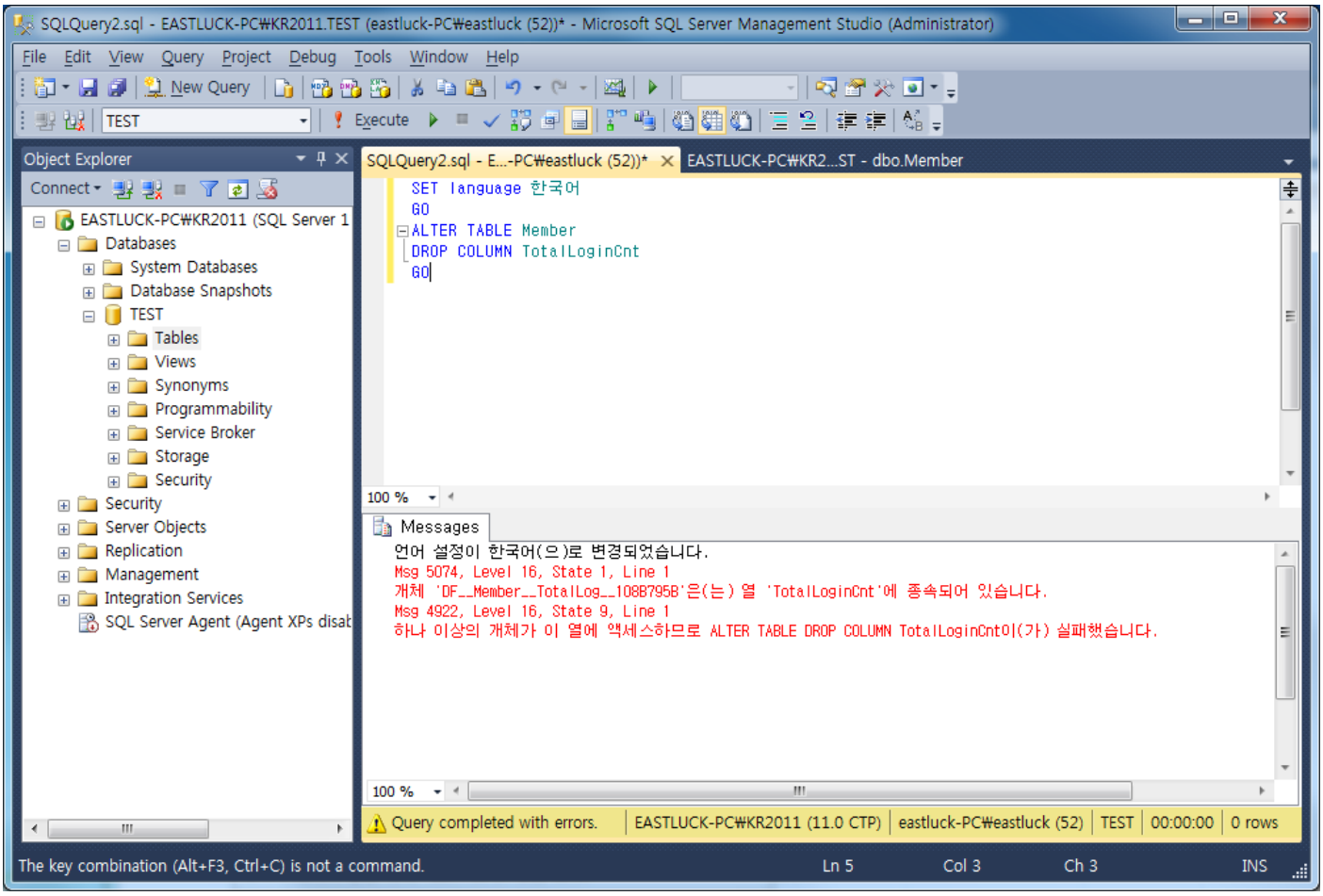
SET language 한국어

GO

ALTER TABLE Member

DROP COLUMN TotalLoginCnt

GO



개체 'DF__Member__TotalLog__108B795B'은(는) 열 'TotalLoginCnt'에 종속되어 있다.

라는 에러와 함께 삭제가 되지 않다. 이것은 무엇일까?

아까 컬럼을 추가했을 당시.. DEFAULT 값을 주었다. 이것이 default 제약 조건에 포함이 되어있기 때문에 먼저 삭제를 해야 삭제가 가능하다.

물론 DEFAULT 제약조건도 자식이기 때문에 부모가 없어지려면 자식객체 부터 지워야 한다.

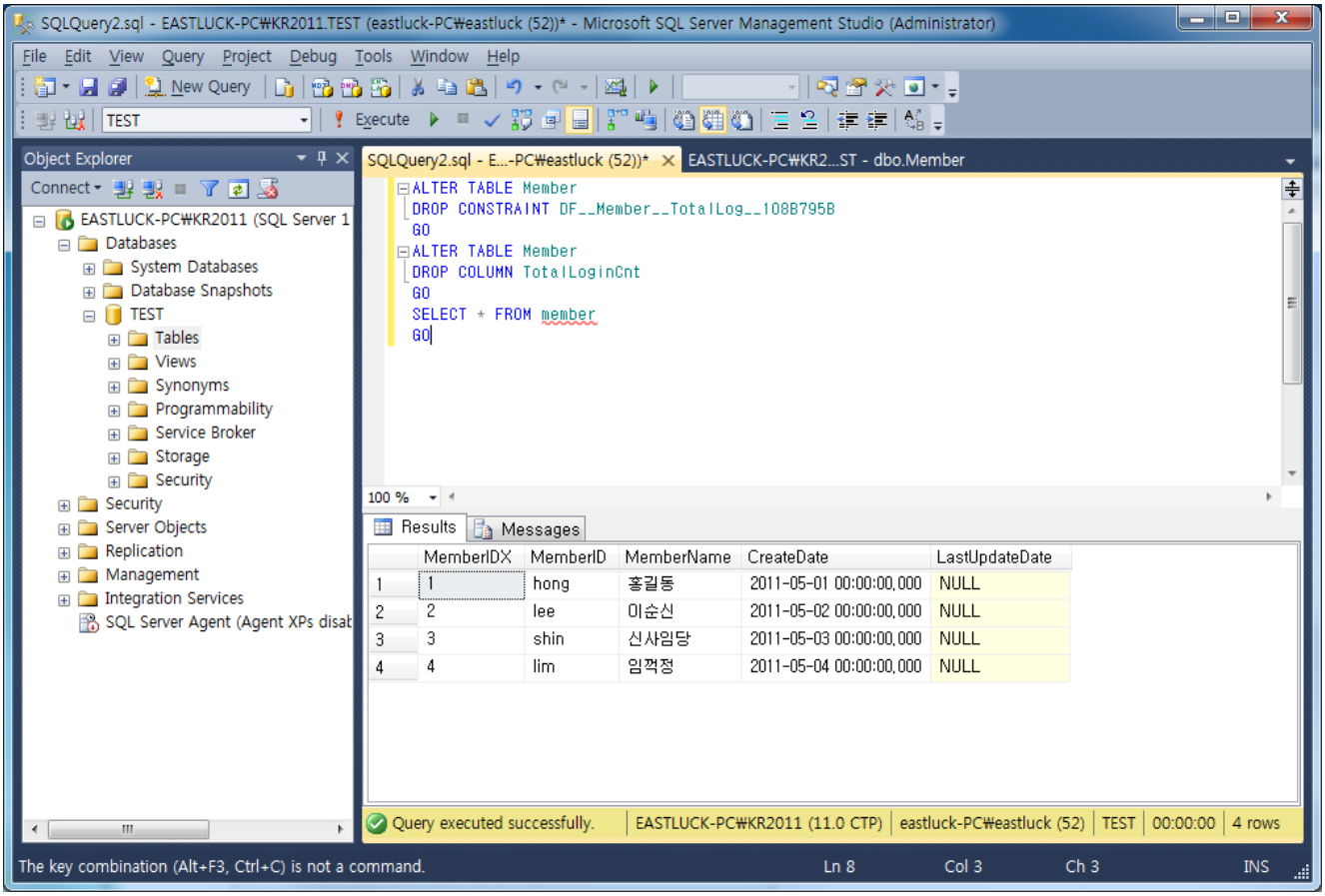
ALTER TABLE Member

DROP CONSTRAINT DF__Member__TotalLog__108B795B

GO

주의: DF__Member__TotalLog__108B795B 값은 컴퓨터 마다 달라질 수 있다.

물론 default 제약조건에 이름을 지정할 수는 있지만.. 우리는 지정하지 않았기 때문에 .. SQL Server 내부적으로 알아서 지정하게 된다. 이제 다시 컬럼을 삭제해보면 ..정상적으로 삭제되는 것을 볼 수 있다.



[SQL2011 강좌] 6. Primary key 와 Unique 제약조건

개체 무결성에 대해 알아볼까 한다. 물론 그 중심에는.. Primary Key(기본키) 그리고 유니크 제약조건이 있다.

- 개체 무결성의 정의: **기본키를 구성하는 속성은 널(NULL)값이나 중복값을 가질 수 없음**

  [출처] 개체 무결성, 참조 무결성|**작성자** Ian

Primary Key 의 조건은 아래와 같다.

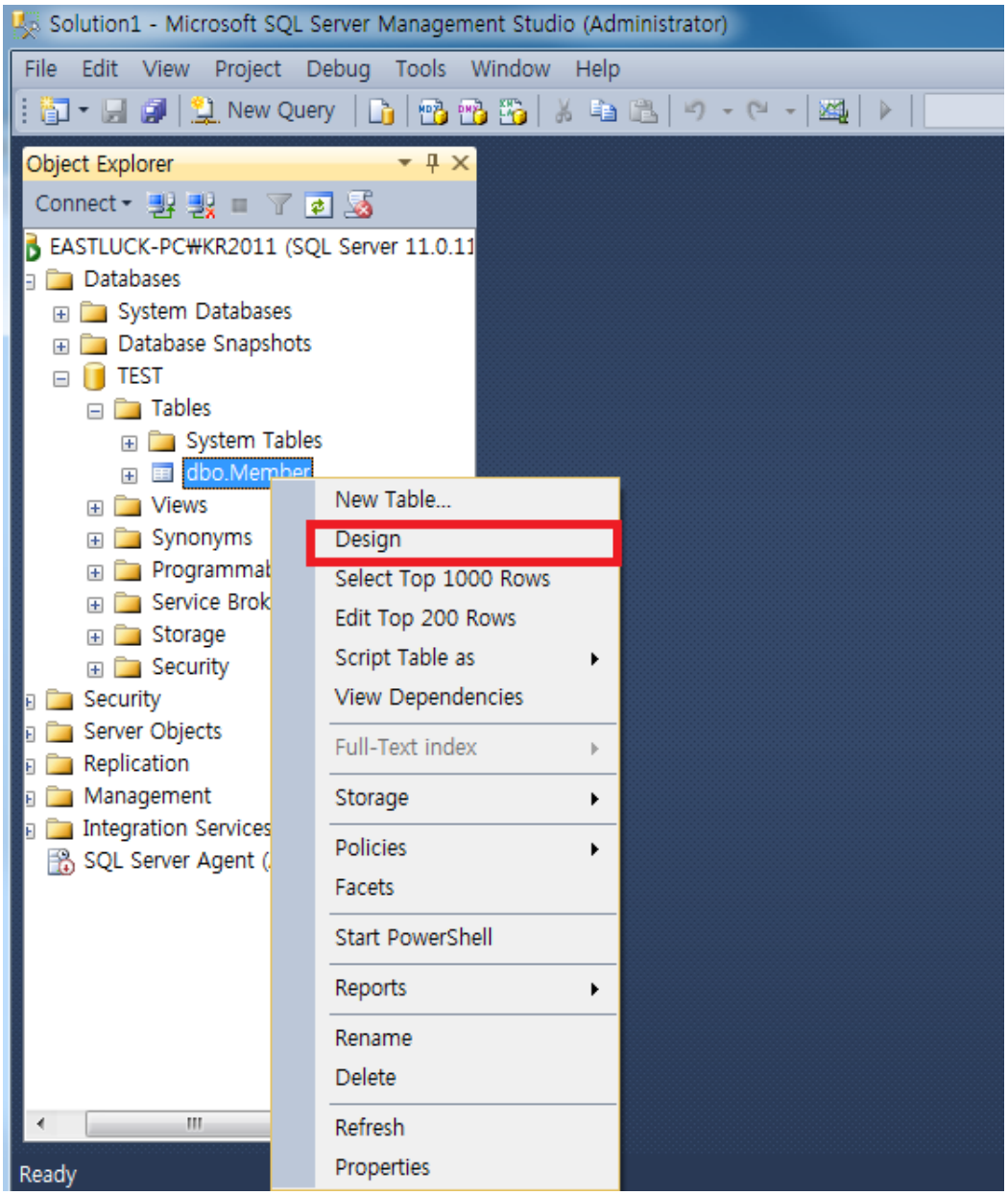
1. 유니크 해야함.
2. NULL 을 허용할 수 없음.

개체 무결성과도 같다. 회원 테이블을 살펴보면.. 아래 두 가지 경우가 대상이 될 수 있다.

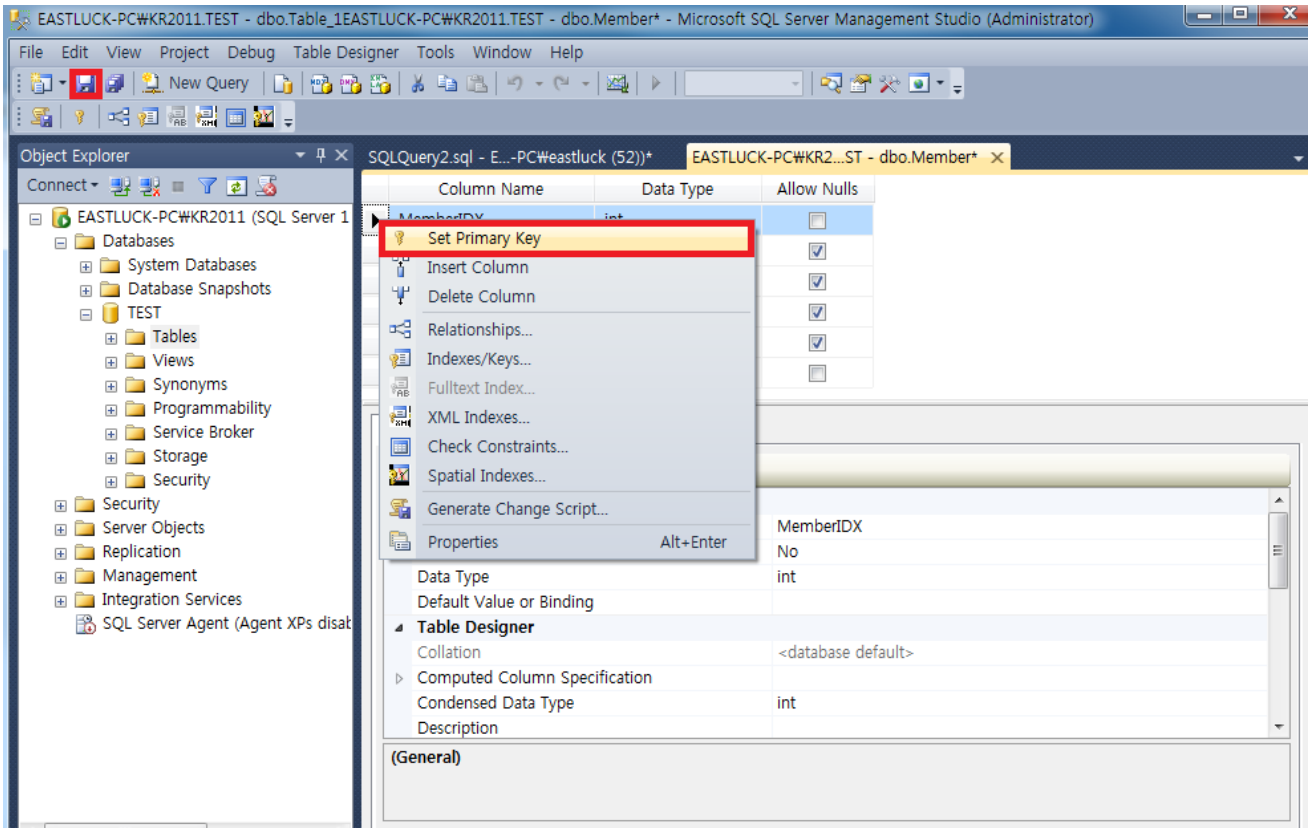
1. 순차적으로 증가하는 회원번호(INT 4 bytes)
2. 회원 아이디(VARCHAR 10~15 bytes)
3. 주민등록번호(CHAR 형 13 bytes)

하지만.. 순차적으로 증가하는 회원번호의 사이즈가 작기 때문에 Primary Key 에 적당하다. 또한 주민등록번호 혹은 아이디는 약관에 따라 탈퇴를 하는 경우에는 지워질 수 있기 때문에 대상에서 살짝 밀리게 된다. 이처럼 Primary Key 외에 아이디 혹은 주민등록번호처럼 대체가 될 수 있는 키를 대체키라고 한다. 영어로는 Alternate Key 라고 한다.

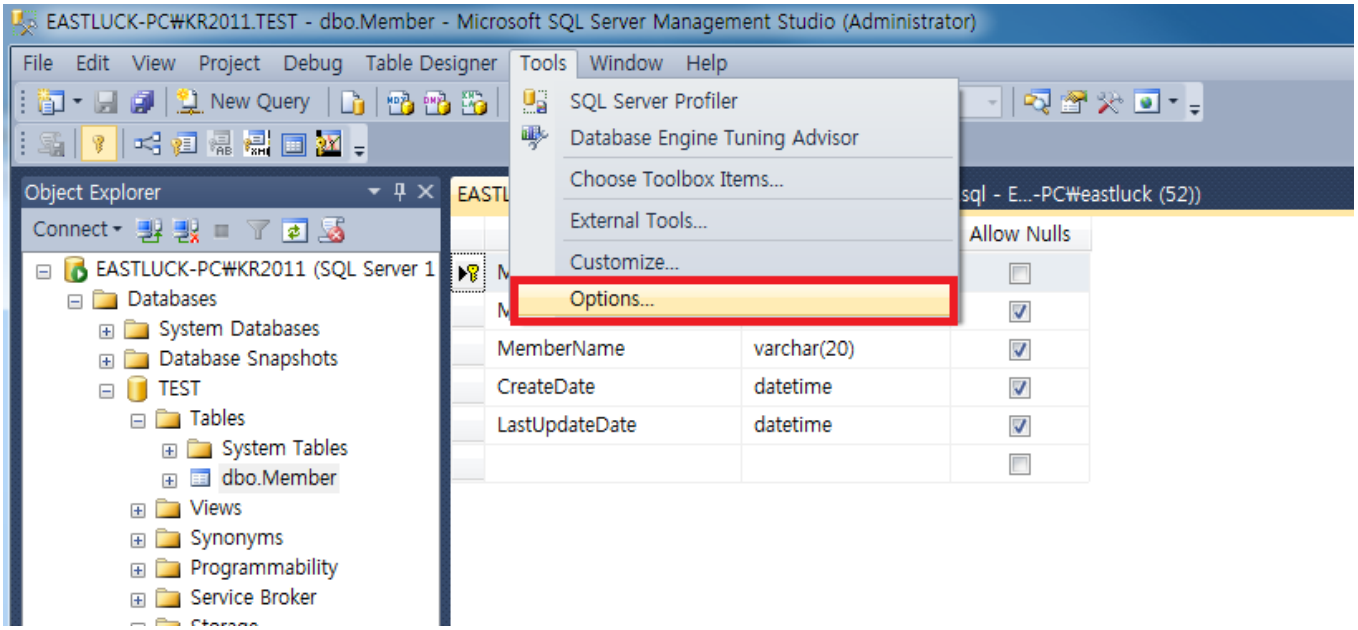
그럼 한번 SSMS 를 통해서 Primary Key 를 생성해보도록 하겠다. SSMS 접속 => Database => Test Databse => Member Tables 우클릭 => Desgin



디자인을 클릭하면 아래와 같은 화면이 나온다. 여기서 전에 만들었던 MemberIDX 에 Primary Key 를 걸어보도록 하겠다. MemberIDX 에 마우스 우클릭 => Set Primary Key => 그리고 저장을 선택한다.

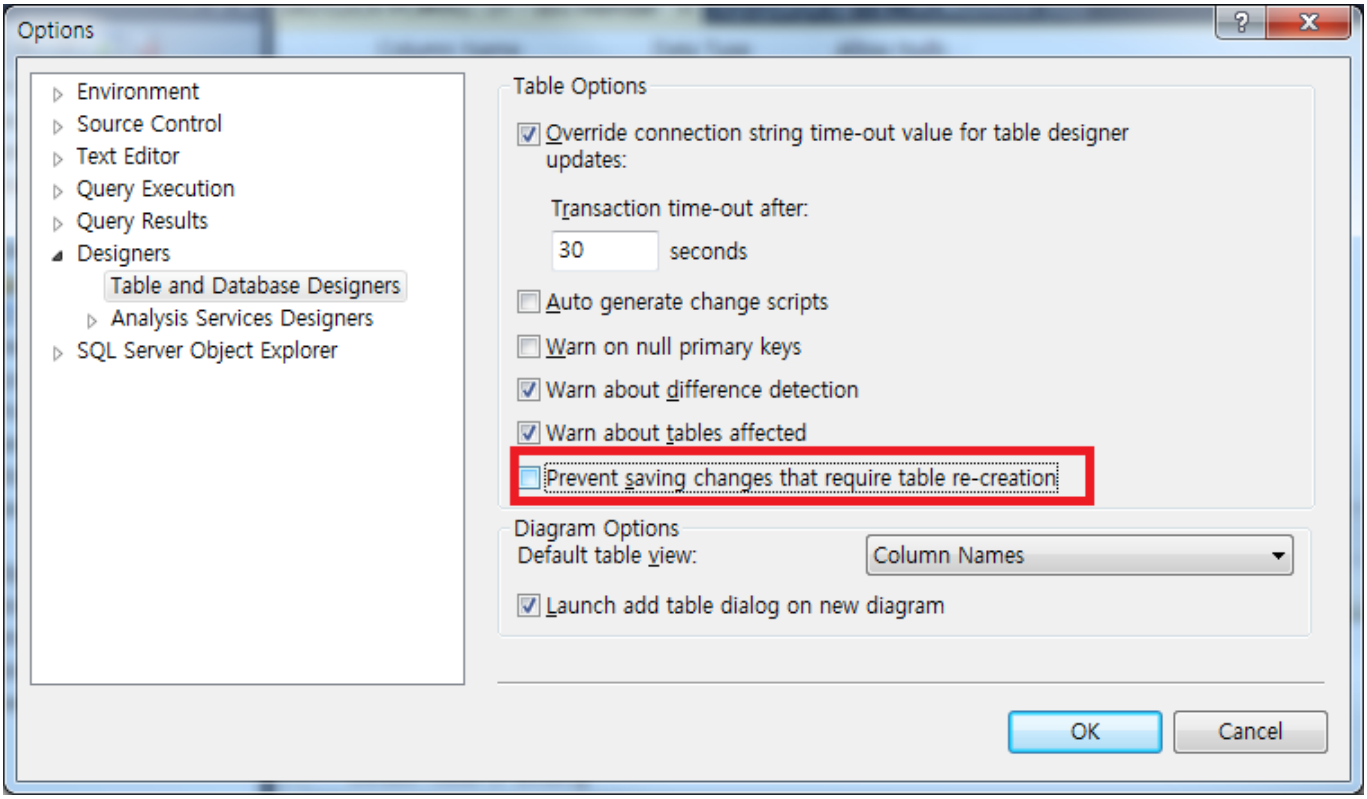


(혹시 위 설정에서 저장이 안되시는 분들은.. Tools => Option => Designers => Table and Database Designers => Prevent saving changes that require table re-creation 을 체크하시기 바란다.)



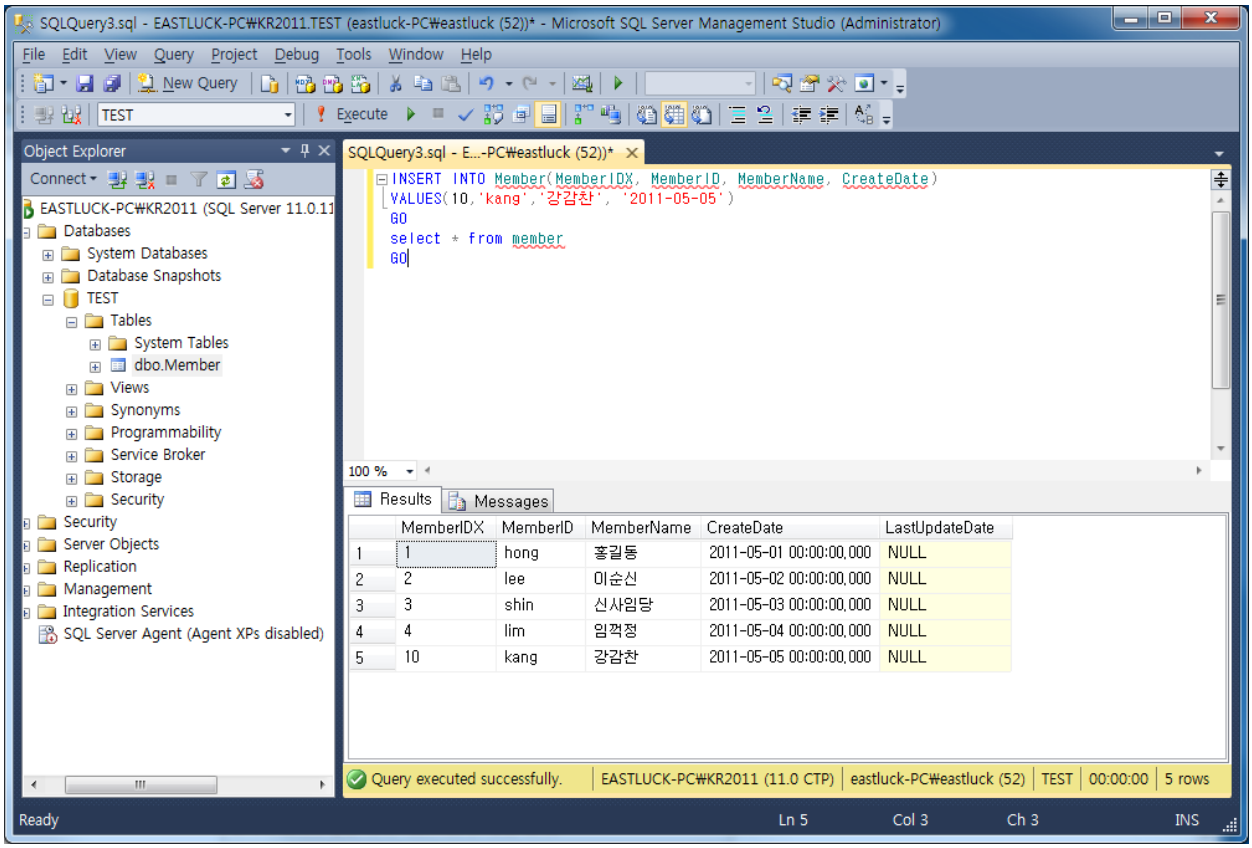
이 옵션은.. Design 에서 테이블을 수정 시에 테이블이 다시 만들어질 수 있다. 이 부분을 사전에 방지하고자 기본 값으로 체크가 되어있는데 체크를 빼주시면 될 것 같다.

주의: 데이터가 아주 많은 경우 테이블이 다시 만들어지면 곤란해 질 수 있으므로 주의한다.

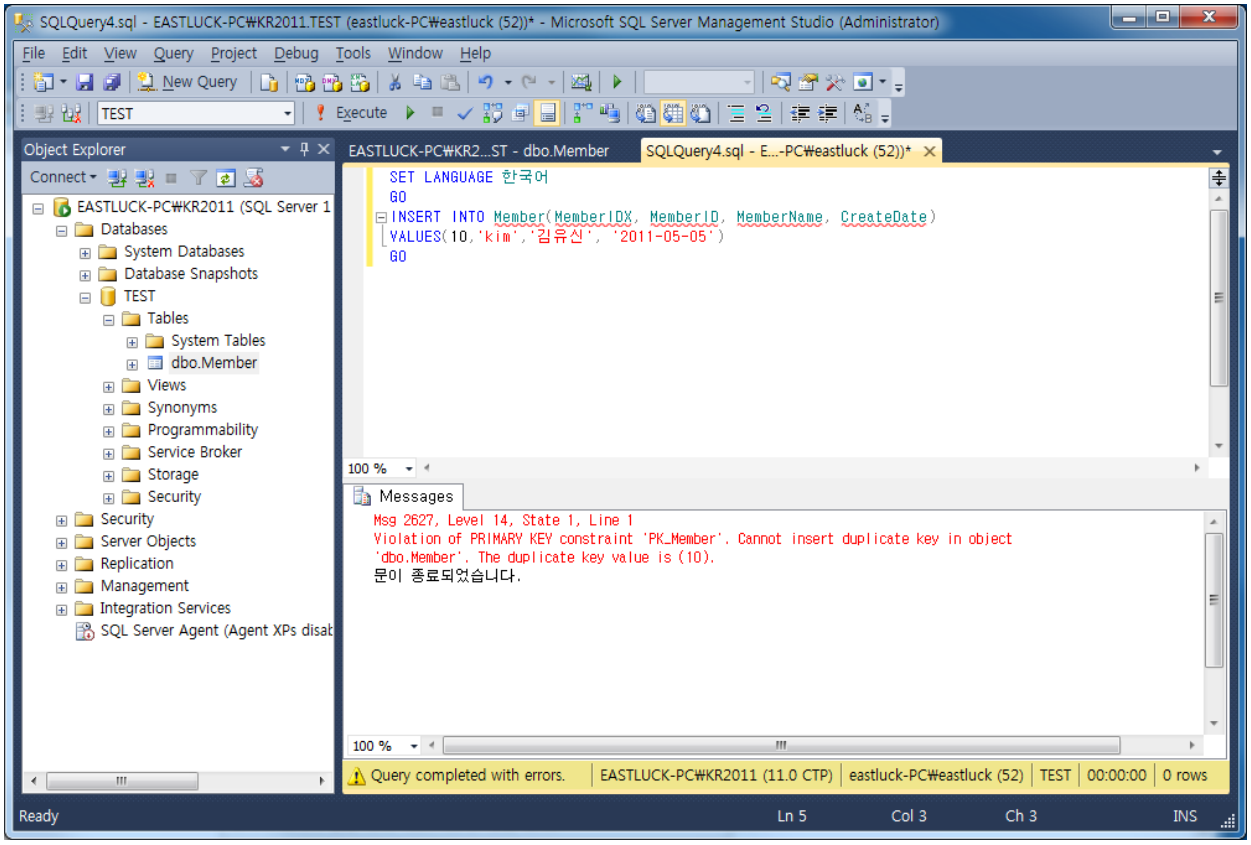


INSERT 문을 배운 건 아니지만 한번 NULL 값이 들어가는지 중복된 값이 들어가는지 테스트를 해보도록 하겠다. 우선 10 번으로 강감찬을 넣어보도록 하겠다.

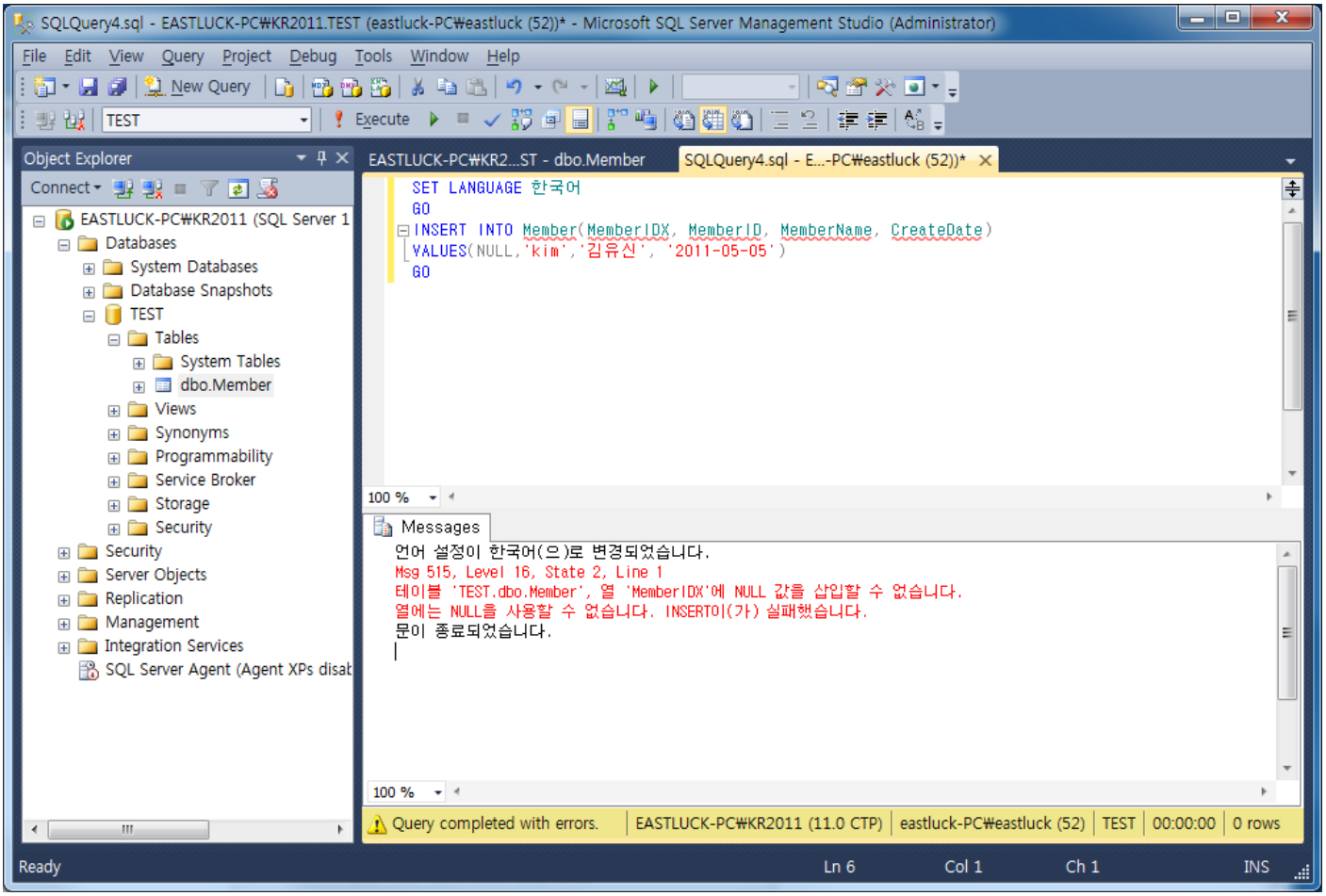
```
INSERT INTO Member(MemberIDX, MemberID, MemberName, CreateDate)
VALUES(10,'kang','강감찬', '2011-05-05')
GO
select * from member
GO
```



이제 10 번을 다시 한번 넣어보도록 하겠다. 이때 오류가 발생하는데 중복된 키 값을 넣을 수 없다는 에러 이다.



이제 NULL 값을 넣어보도록 하겠다. NULL 값 역시 허용하지 않았기 때문에 데이터가 입력되지 않는다.



그럼 중복되지 않지만 NULL 을 넣을 순 없을까?? 바로 UNIQUE 제약조건이 있다. MemberID 에 유니크 제약조건을 걸어보겠다.

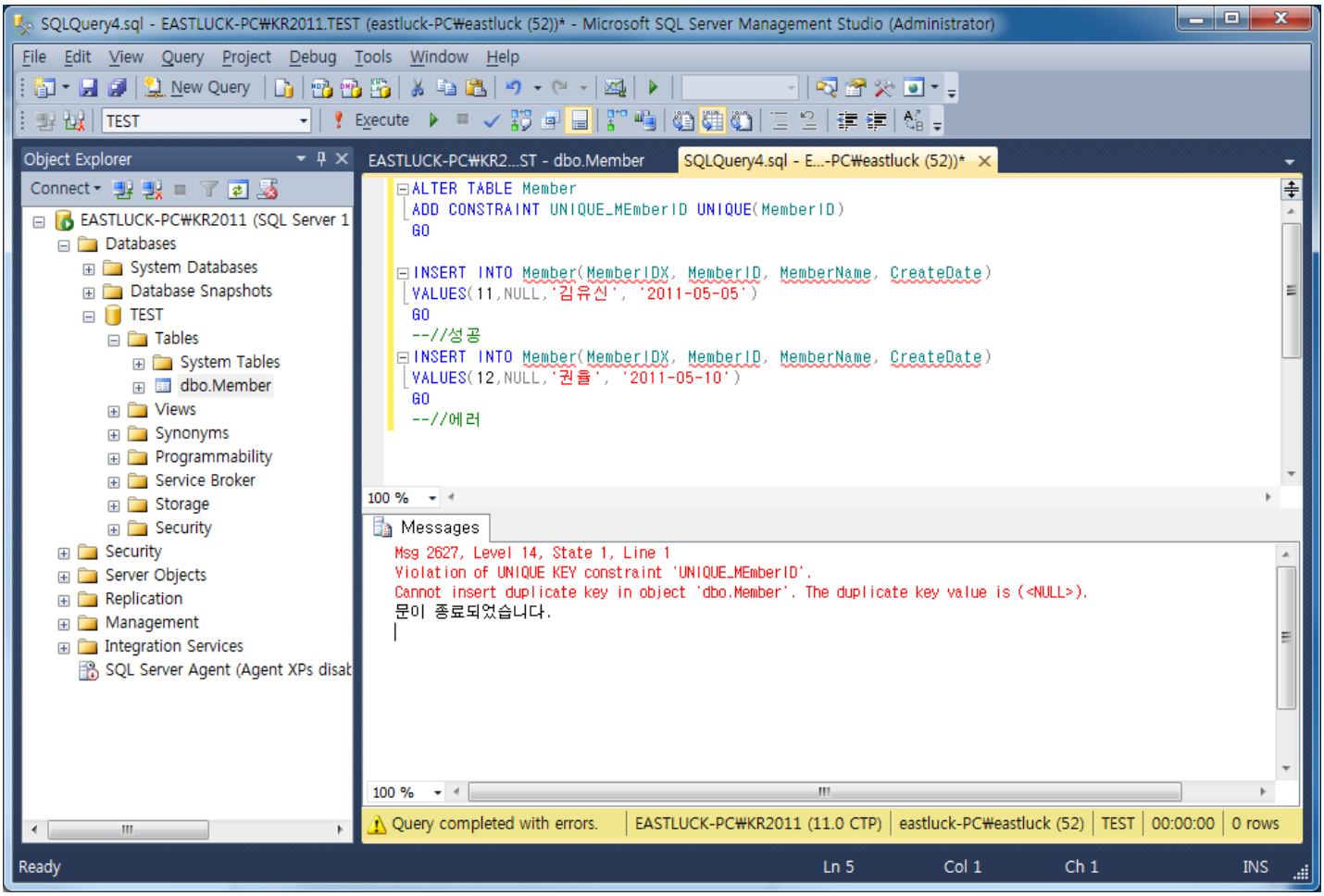
ALTER TABLE Member

ADD CONSTRAINT UNIQUE_MEmberID UNIQUE(MemberID)

GO

--// ALTER TABLE 테이블명

--// ADD CONSTRAINT 제약조건명칭 UNIQUE(컬럼명)



역시 NULL 은 한 개 입력되지만 중복이 허락되진 않는다. 이처럼 Primary Key 와 Unique 의 차이점은 바로 NULL 이 한 개 들어갈 수 없냐의 문제였다.

[SQL2011 강좌] 7. DML 1 탄 INSERT

DML 은 Data Manipulation Language 의 약자로써 데이터 조작어에 해당된다. 이에 관련되서는 SELECT, INSERT, UPDATE, DELETE 가 해당이 된다. 이 중에서 우리는 오늘 INSERT 에 대해서 알아볼 예정이다.

이와 관련해서 임시 테이블을 생성해보도록 하겠다.

```
--//회원(회원번호, 회원아이디, 회원이름, 회원나이, 회원지역)
CREATE TABLE MemberSelect
(
MemberIDX INT
, MemberID VARCHAR(10)
, MemberName VARCHAR(20)
, MemberAge INT
, MemberArea VARCHAR(4)
, CreateDate DATETIME
)
GO
```

데이터를 INSERT 하는 방법에는 여러 가지가 있다.

1. 첫 번째 모든 컬럼명을 기술하기.

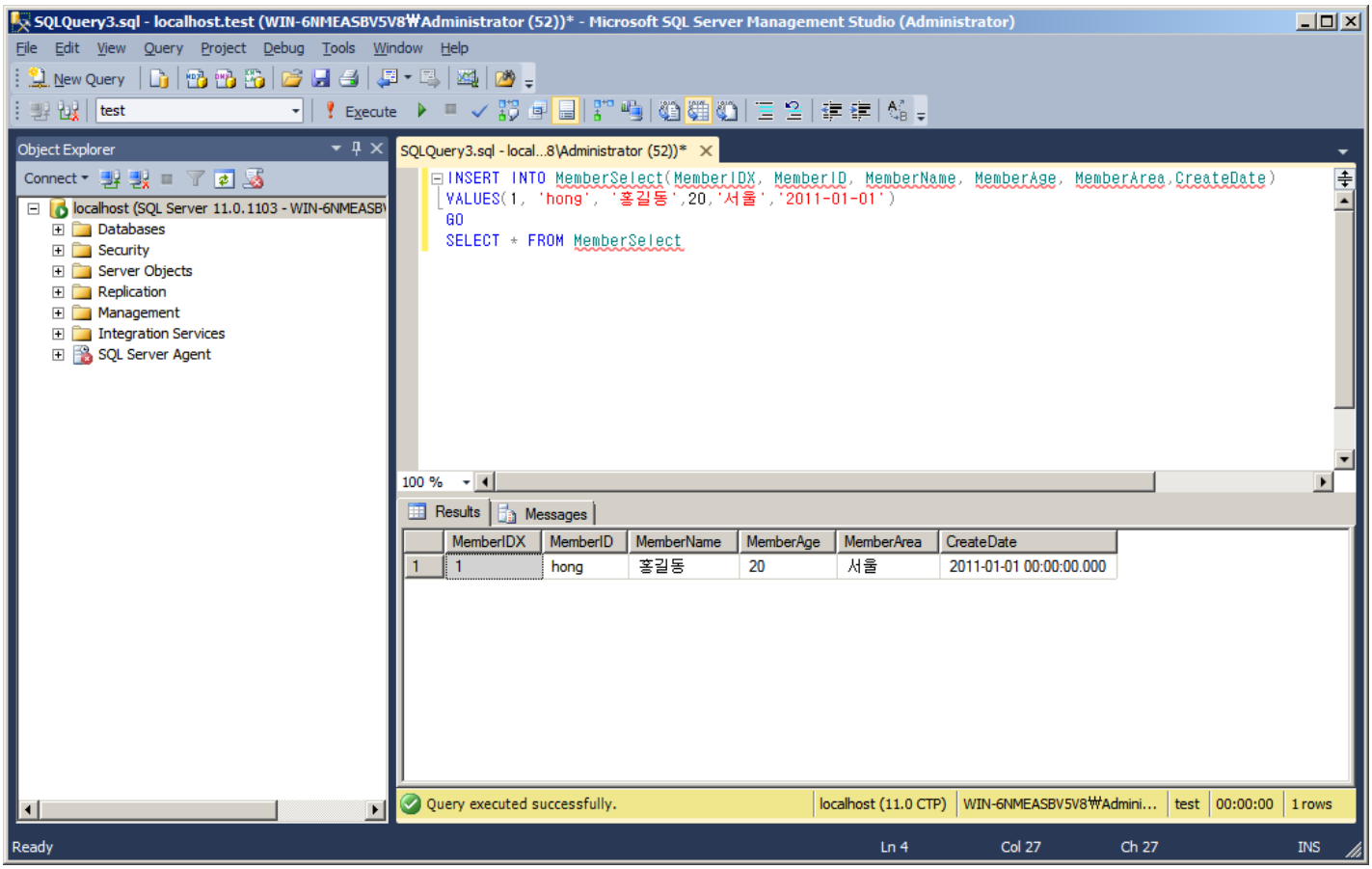
INSERT INTO 테이블명(컬럼 1, 컬럼 2, 컬럼 3 ....) VALUES(컬럼 1 의 값, 컬럼 2 의 값, 컬럼 3 의 값 ...)

GO

INSERT INTO MemberSelect(MemberIDX, MemberID, MemberName, MemberAge, MemberAre

VALUES(1, 'hong', '홍길동',20,'서울','2011-01-01')

GO



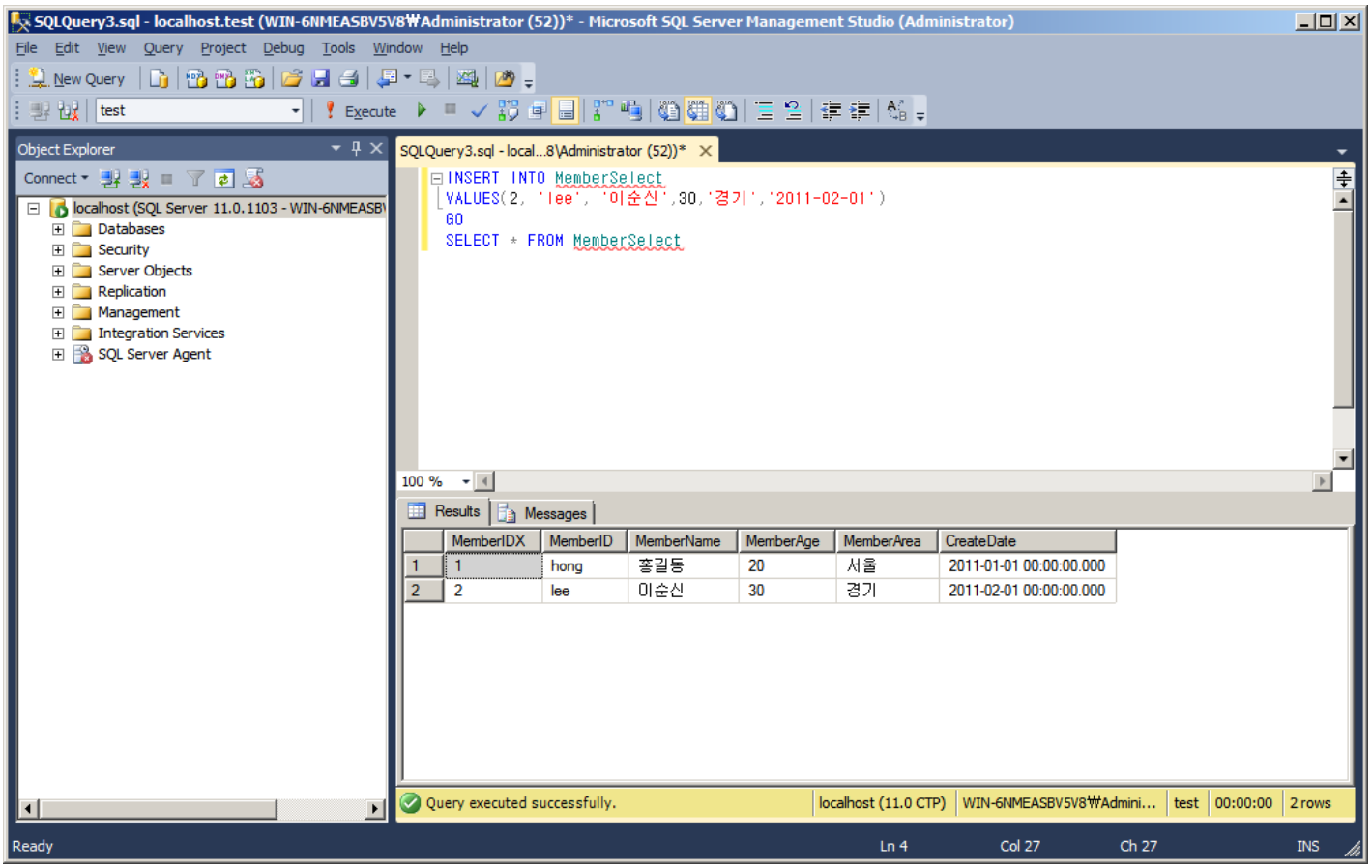
2. --//컬럼명을 기술하지 않고 입력하는 두 번째 방법

주의: 이 방법은 VALUES 에 순서데로 기술된 컬럼의 순서가 실제로 생성한 테이블의 컬럼 수 와 순서가 반드시 맞아야 한다.

INSERT INTO MemberSelect

VALUES(2, 'lee', '이순신',30,'경기','2011-02-01')

GO



3. 3 번째! SELECT 를 활용한 INSERT 문법

INSERT INTO MemberSelect

(

MemberIDX

, MemberID

, MemberName

, MemberAge

, MemberArea

, CreateDate

)

SELECT

3

, 'kim'

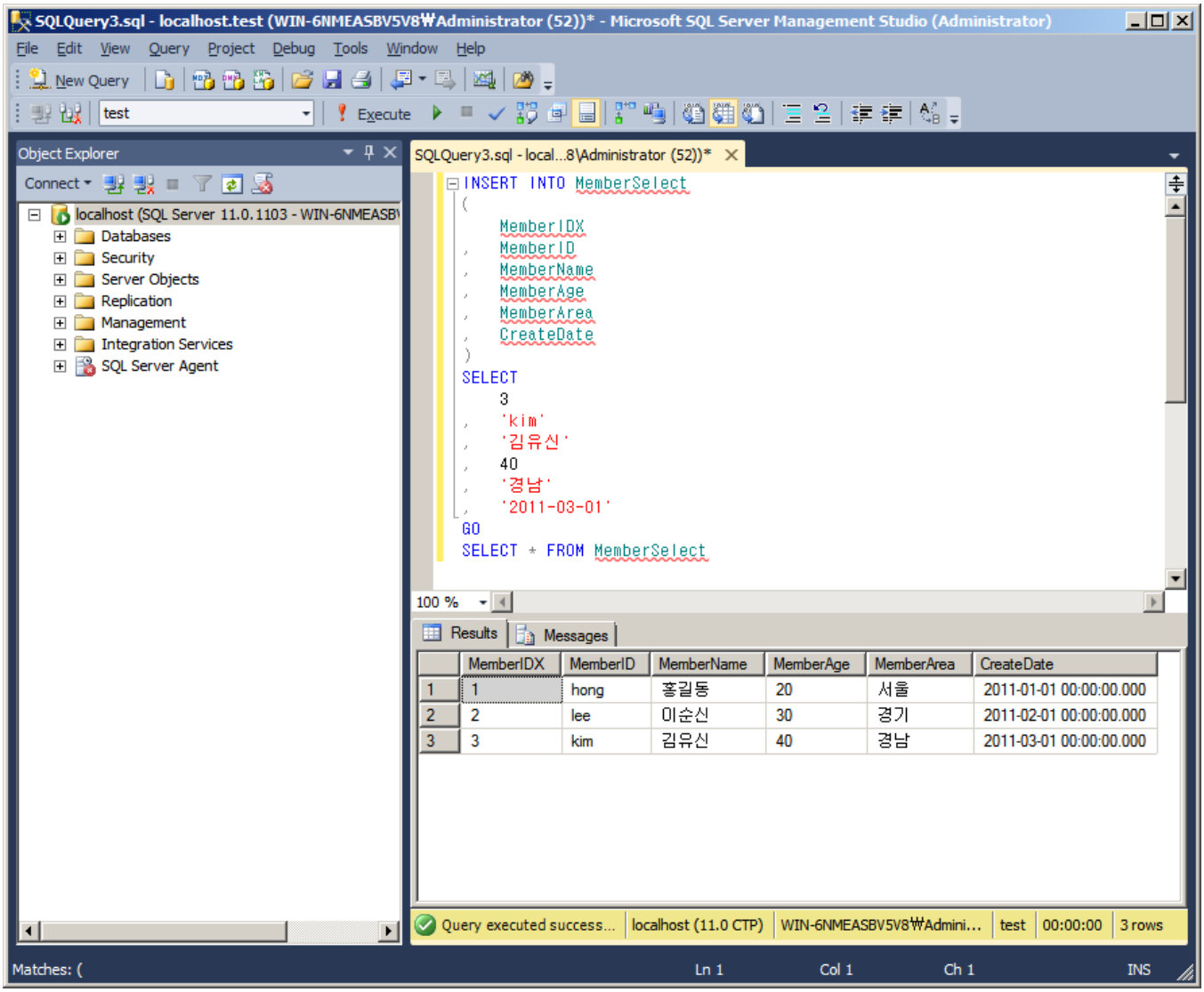
, '김유신'

, 40

, '경남'

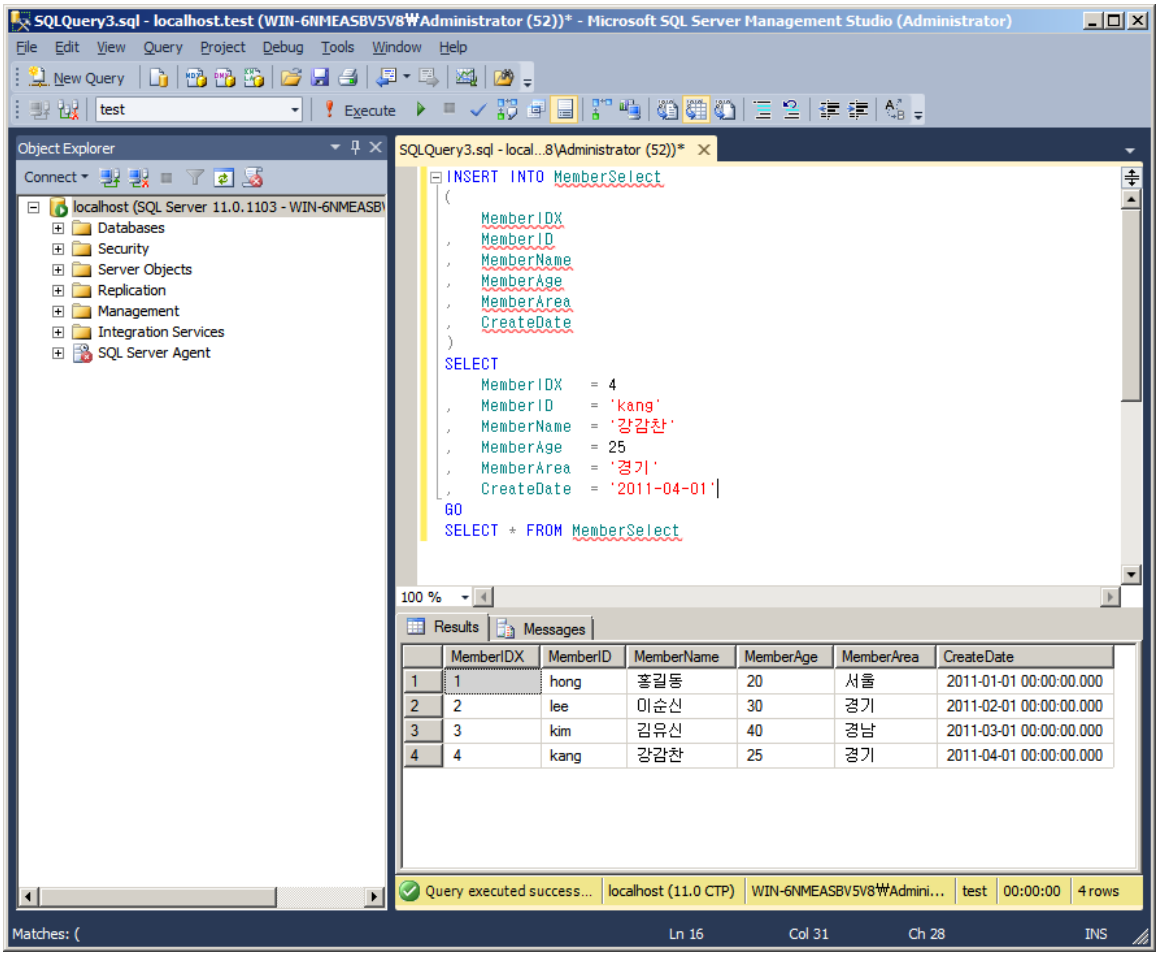
, '2011-03-01'

GO

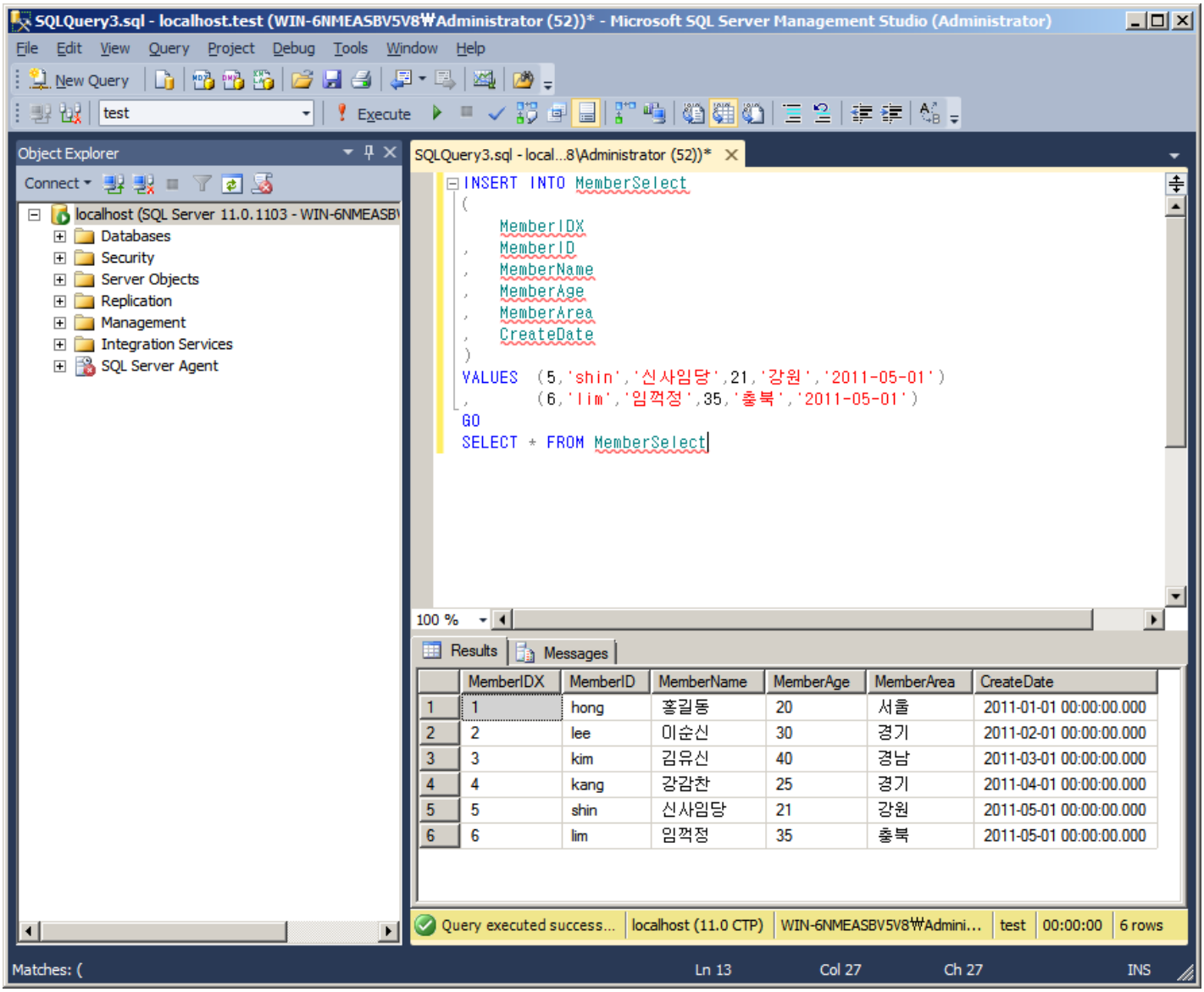


4. 4 번째 SELECT 절에 ROW 명을 기술하는 방법.

```
INSERT INTO MemberSelect
(
MemberIDX
, MemberID
, MemberName
, MemberAge
, MemberArea
, CreateDate
)
SELECT
MemberIDX = 4
, MemberID = 'kang'
, MemberName = '강감찬'
, MemberAge = 25
, MemberArea = '경기'
, CreateDate = '2011-04-01'
GO
```



SQL 2008 부터 지원하는 다중 행 입력.

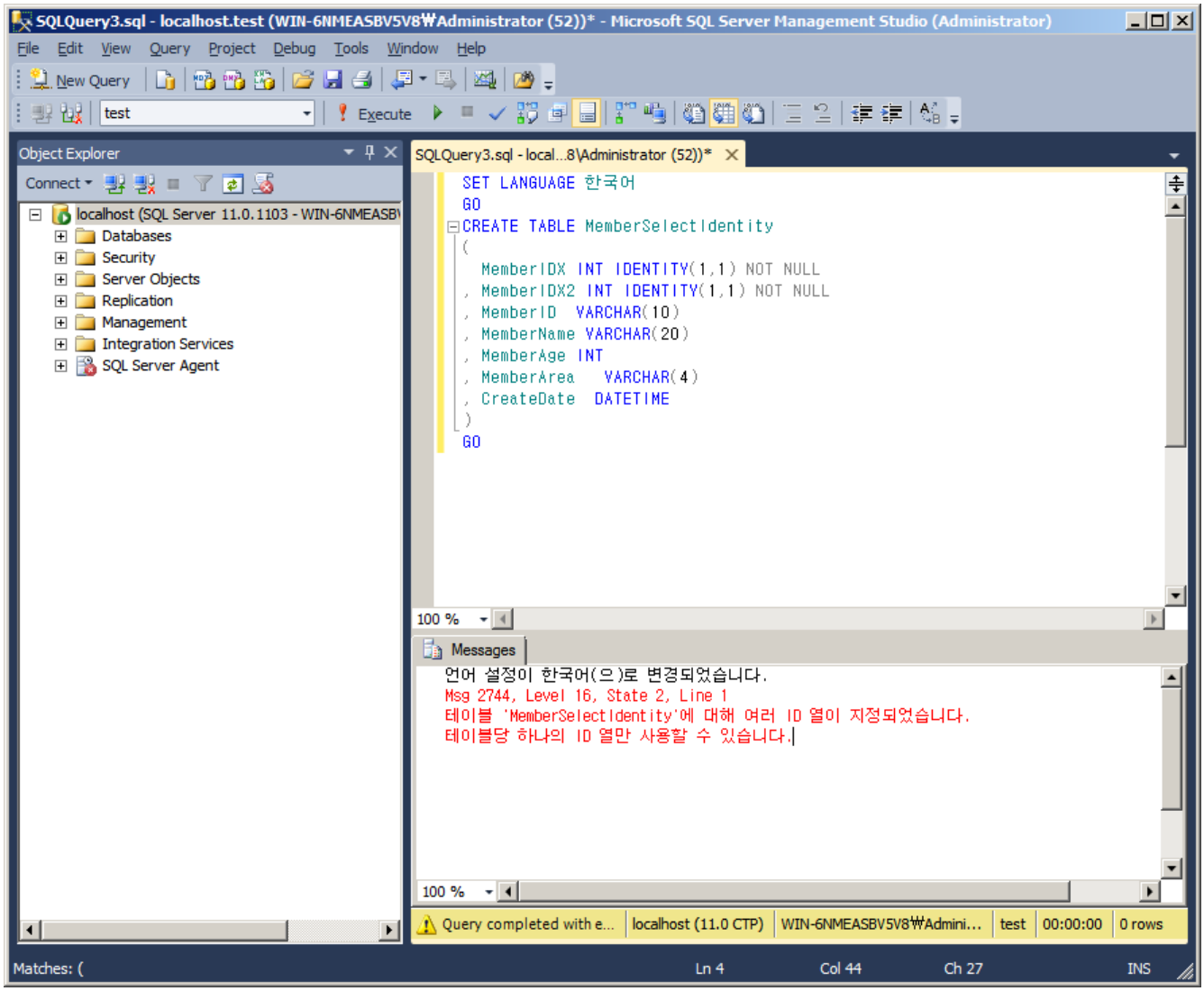


지금까지 가장 기본적으로 INSERT 하는 방법을 알아봤다. 사실 데이터 입력방법은 이것보다 훨씬 다양하고 많다. 저는 개인적으로 테이블에 컬럼이 많아지면 어떤 컬럼이 어떤 데이터인지 구분하기 힘들기 때문에 4 번 방법을 주로 이용한다.

다중행 입력관련된 좋은 글: http://sqlsql.tistory.com/entry/Array-Insert

[SQL2011 강좌] 8. DML 1 탄 INSERT(identity 속성)

여기서는 DML 중에서 INSERT 항목 그 중 identity 에 대해서 알아보도록 하겠다. 이와 관련해서 임시

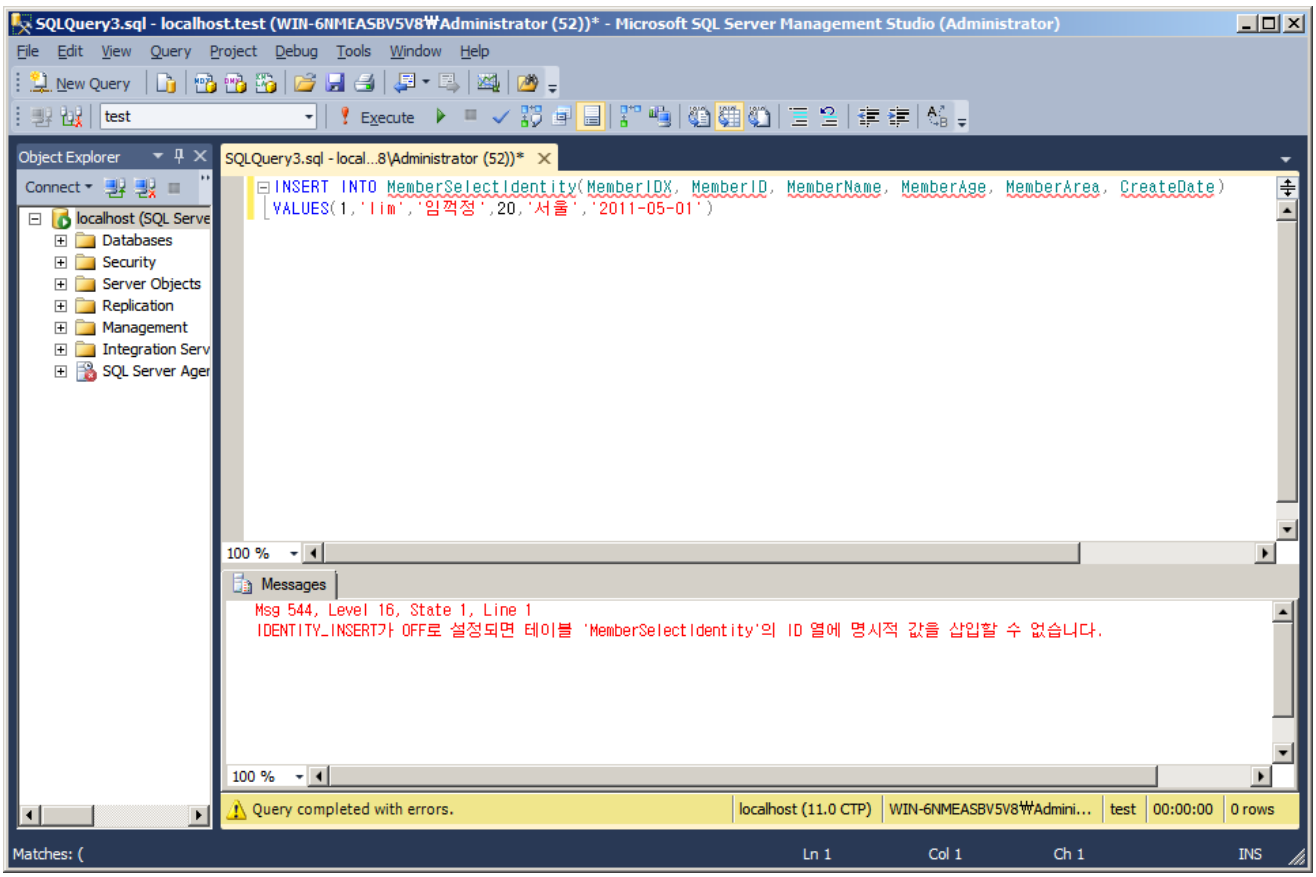


테이블을 생성해보도록 하겠다. 저번과 다른 부분은 MemberIDX 에 IDENTITY(시작값, 증가값) 라는 속성을 부여했다. 테이블에 IDENTITY 속성이 있는 컬럼을 두 개 이상 만드는 경우에는 아래와 같은 에러가 발생한다. 따라서 테이블에 IDENTITY 속성은 반드시 하나만 존재할 수 있다. 아래와 같이 테이블을 만들어본다.

```
IF OBJECT_ID('MemberSelectIdentity') IS NOT NULL
DROP TABLE MemberSelectIdentity
```

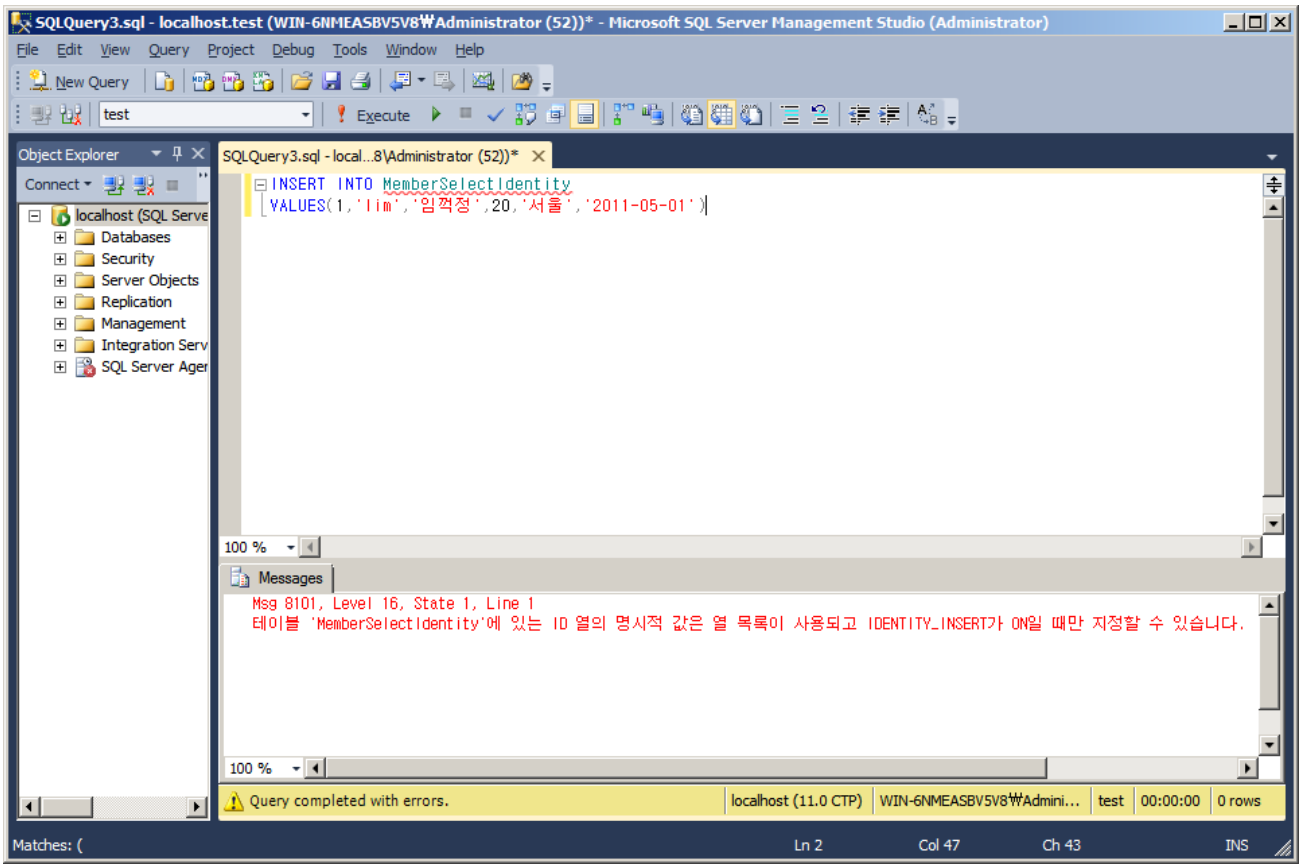
```
GO
CREATE TABLE MemberSelectIdentity
(
MemberIDX INT IDENTITY(1,1) NOT NULL
, MemberID VARCHAR(10)
, MemberName VARCHAR(20)
, MemberAge INT
, MemberArea VARCHAR(4)
, CreateDate DATETIME
)
GO
```

테이블에 IDENTITY 속성이 있는 경우에는 기본적으로 DATA INSERT 시에 IDENTITY 속성을 가진

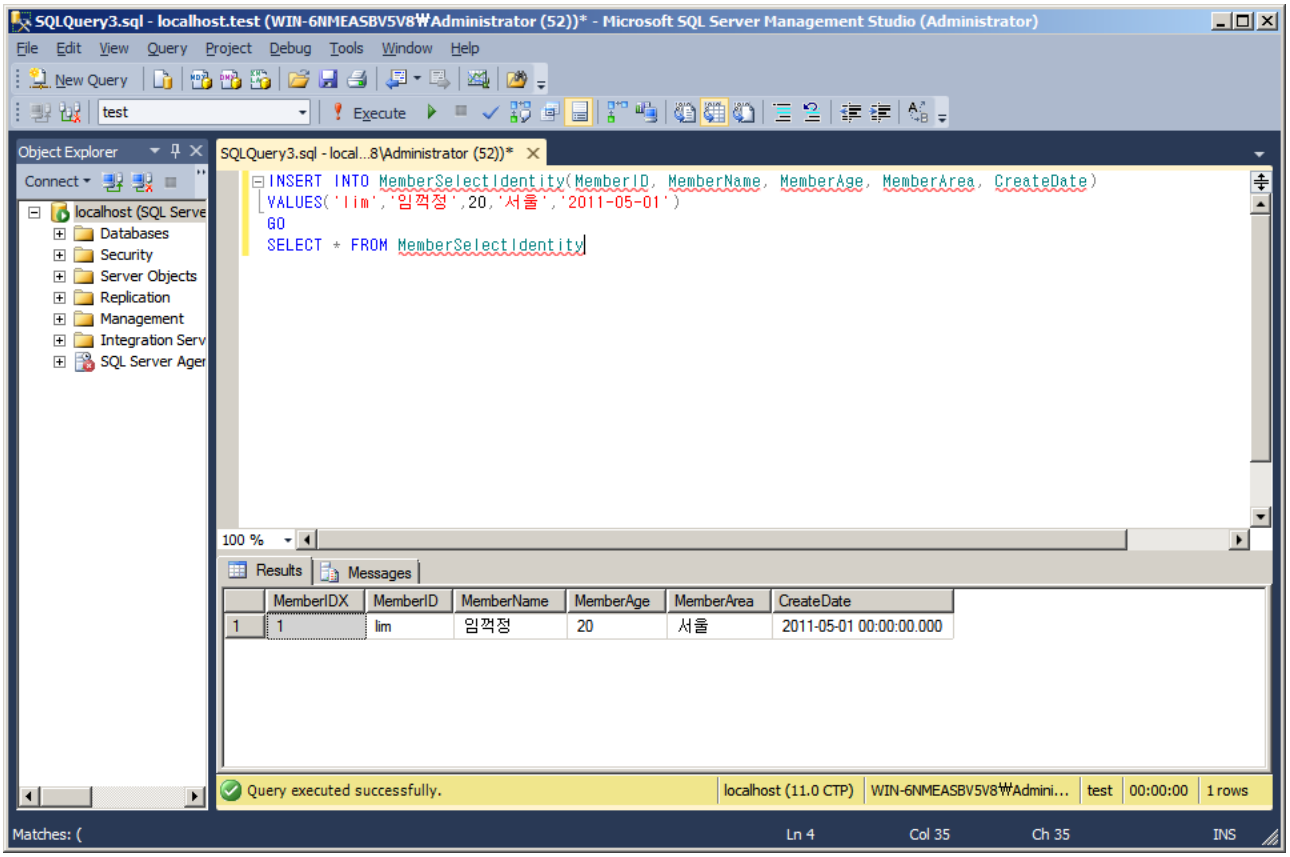


컬럼명을 제외해야 한다. 만약 함께 기술한다면 아래와 같은 에러가 발생하게 된다.

또한. 컬럼명을 생략한 경우에도 마찬가지겠다.

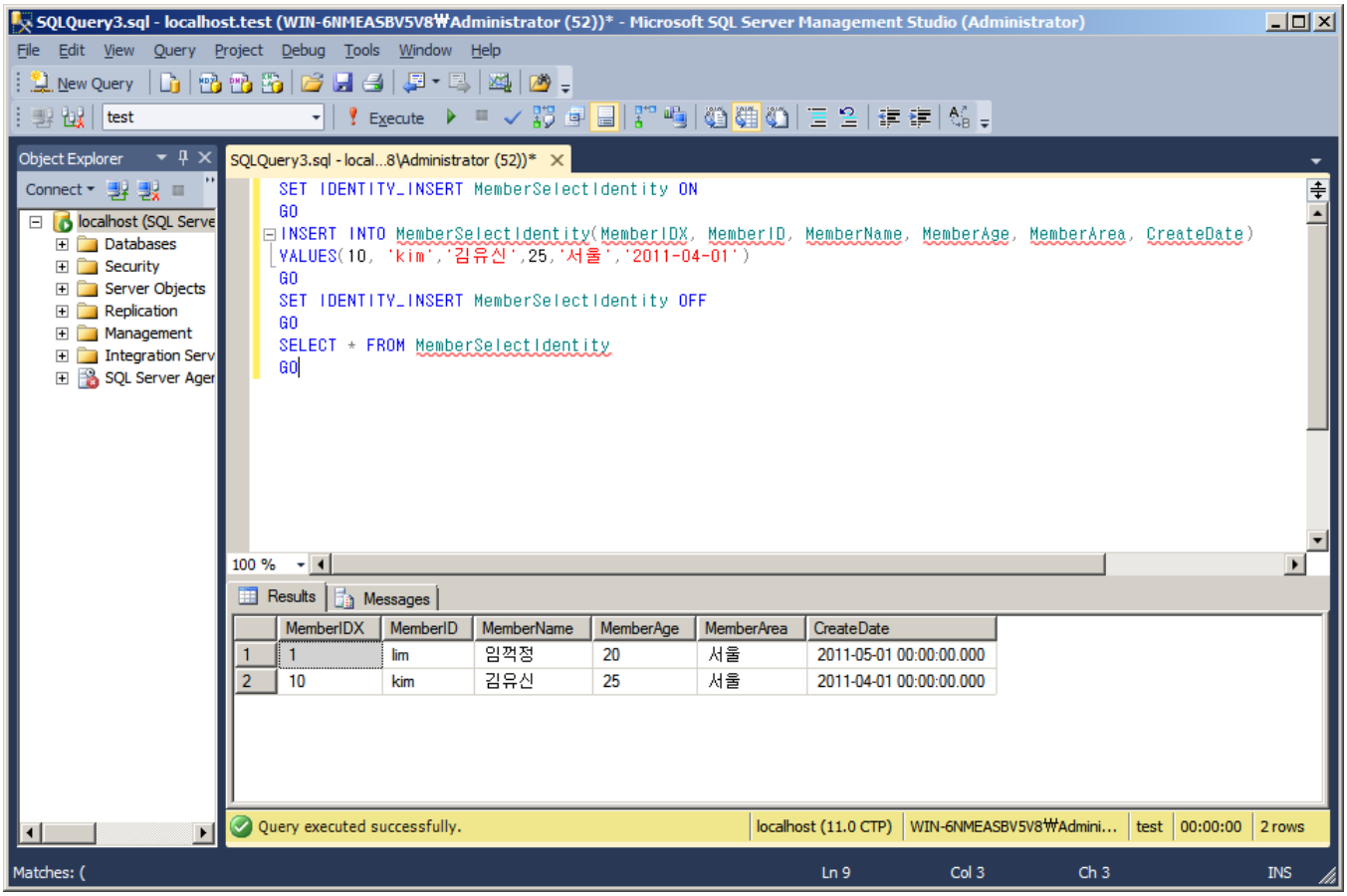


이제 컬럼명을 생략해서 넣어보도록 하겠다.



정상적으로 들어갔음을 확인할 수 있다. 하지만. 에러 내용들 처럼 SET IDENTITY_INSERT 를 사용해서 MemberIDX 에 원하는 값을 넣을 수 있다.

Link: [SET IDENTITY_INSERT(Transact-SQL)]



그렇다면 현재 IDENTITY 값은 알 수 없을까?? 또한 IDENTITY 값을 변경할 순 없을까?? 라는 의문을 가지게 된다. 이것은 DBCC CHECKIDENT 를 통해 가능하다.

Link: [DBCC CHECKIDENT(Transact-SQL)]

[SQL2011 강좌] 9. Sequence

이번에는 SQL 2011 에서 새로 추가된 SEQUENCE 에 대해서 알아보도록 하겠다. SEQUENCE 는 ORACLE 에 있고 SQL Server 에는 없어서 많은 SQL Server DBA 들이 추가되기를 기대했던 기능이다. 이번 버전에서는 SQL Server 에서도 시퀀스가 추가되어서 사용할 수 있게 되었다.

SEQUENCE 에 대해서 깊이 있게는 알아보시기 위해서는 아래 링크 참고하시기 바란다.

Link: CREATE SEQUENCE(Transact-SQL)

Link: NEXT VALUES FOR(Transact-SQL)

오늘은 위의 예제 중에서 생성 해서 어떻게 가져오는 지에 대한 간단한 예제를 보여드리겠다.

```
IF OBJECT_ID('SequenceStep5','SO') IS NOT NULL
 DROP SEQUENCE SequenceStep5
GO
IF OBJECT_ID('SequenceStep1','SO') IS NOT NULL
 DROP SEQUENCE SequenceStep1
GO
CREATE SEQUENCE dbo.SequenceStep1--//옵션 없이 생성하는 방법
GO
CREATE SEQUENCE dbo.SequenceStep5 --//시작 값, 증가 값 옵션을 설정하는 방법
   START WITH 5
   INCREMENT BY 5
GO
```

SQLQuery3.sql - localhost.test (WIN-6NMEASBV5V8₩Administrator (52))* - Microsoft SQL Server Management Studio (Administrator)

```
IF OBJECT_ID('SequenceStep5','SO') IS NOT NULL
    DROP SEQUENCE SequenceStep5
GO

IF OBJECT_ID('SequenceStep1','SO') IS NOT NULL
    DROP SEQUENCE SequenceStep1
GO
CREATE SEQUENCE dbo.SequenceStep5
    START WITH 5
    INCREMENT BY 5
GO
CREATE SEQUENCE dbo.SequenceStep1
GO

select NEXT VALUE FOR DBO.SequenceStep1, NEXT VALUE FOR DBO.SequenceStep5
```

	(No column name)	(No column name)
1	-2147483648	5

Query executed successfully. localhost (11.0 CTP) WIN-6NMEASBV5V8₩Admini... test 00:00:00 1 rows

같은 쿼리 한번 더 수행.

SQLQuery3.sql - localhost.test (WIN-6NMEASBV5V8₩Administrator (52))* - Microsoft SQL Server Management Studio (Administrator)

```
select NEXT VALUE FOR DBO.SequenceStep1, NEXT VALUE FOR DBO.SequenceStep5
```

	(No column name)	(No column name)
1	-2147483647	10

Query executed successfully. localhost (11.0 CTP) WIN-6NMEASBV5V8₩Admini... test 00:00:00 1 rows

[SQL2011 강좌] 10. pubs, northwind Database 예제 설치

이번에는 SQL Server 2000 에서 예제로 사용했던 pubs 와 northwind database 를 설치하는 방법에 대해서 알려드리겠다. 각각의 예제 때마다 테이블을 생성하는 것이 번거롭기 때문에 이 예제를 통해 활용해보도록 하겠다. 일단 SQL 로 접속 후에 New Query 를 통해 창을 생성 하고, 두 개의 sql 파일을 가지고, SQL 창에서 실행하시면 된다.

instnwnd.sql

instpubs.sql

아래 그림은 northwind database 를 넣었을 때 예제 이다.

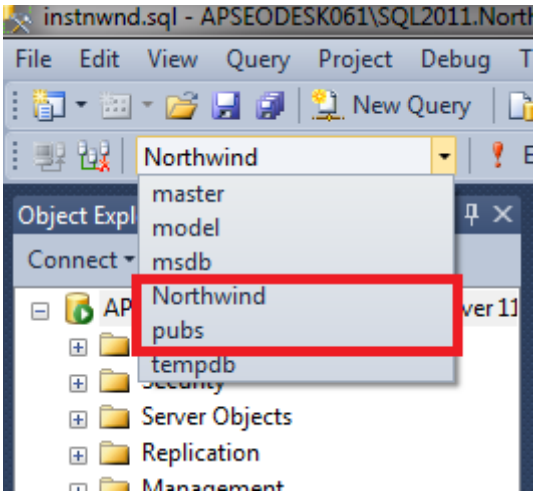
```
/*
** Copyright Microsoft, Inc. 1994 - 2000
** All Rights Reserved.
*/

SET NOCOUNT ON
GO

USE master
GO
if exists (select * from sysdatabases where name='Northwind')
        drop database Northwind
go

DECLARE @device_directory NVARCHAR(520)
SELECT @device_directory = SUBSTRING(filename, 1, CHARINDEX(N'master.mdf', LOWER(filename)) - 1)
FROM master.dbo.sysaltfiles WHERE dbid = 1 AND fileid = 1

EXECUTE (N'CREATE DATABASE Northwind
  ON PRIMARY (NAME = N''Northwind'', FILENAME = N''' + @device_directory + N'northwnd.mdf'')
  LOG ON (NAME = N''Northwind_log'',  FILENAME = N''' + @device_directory + N'northwnd.ldf'')')
go

exec sp_dboption 'Northwind','trunc. log on chkpt.','true'
exec sp_dboption 'Northwind','select into/bulkcopy','true'
GO
```

100 %

Messages

```
Command(s) completed successfully.
```

모두 완료가 되면 두 개의 데이터베이스가 생겼음을 볼 수 있다.


instnwnd.sql - APSEODESK061\SQL2011.North
File
Edit
View
Query
Project
Debug
New Query
Northwind
master
model
msdb
Northwind
pubs
tempdb
Object Expl
Connect
Server Objects
Replication

[SQL2011 강좌] 11. SELECT(단일 테이블)

이번에는 DDL 에서 INSERT 다음으로 SELECT 절 그 중에서 단일 테이블의 데이터를 가져오는 방법을 알아보도록 하겠다. SELECT 절에 대한 구조는 아래와 같다.

```
SELECT 컬럼 1, 컬럼 2, SUM(컬럼 3)

FROM 테이블명

WHERE 컬럼 5 = 20

GROUP BY 컬럼 1,컬럼 2, 컬럼 3

HAVING 컬럼 1 = '서울'
```

GROUP BY 와 HAVING 은 추후 강좌에 다룰 예정이다. 아래 예제와 함께 살펴보시기 바란다. SELECT 절에 대해서 **아주 간단히 설명 드리면** 아래와 같은 형태로 구성되어 있다.

```
방법 1: 컬럼: SELECT * FROM 테이블명

방법 2: SELECT 컬럼 1, 컬럼 2, 컬럼 3... FROM 테이블명
```

방법 1 로 조회를 하시면 테이블안에 있는 모든 컬럼을 가지고 올 수 있다. 방법 2 는 테이블 내에 추출하고 싶은 컬럼만 가져올 경우 사용한다. 위와 같은 기본 조회 방법에 조건을 줘서 필터를 줄 수 있다. 예를 들면 사는 지역이 '서울', 성별이 남자만 나오게 하거나 등등..

조건 절이 추가되면 아래처럼 뒤에 WHERE 절이 추가 된다. WHERE 절이 추가된 예제를 살펴보도록 하겠다.

예 1) Member 라는 테이블에서 MemberAge 값이 10 일 때 모든 컬럼을 가져오는 경우

```
SELECT * FROM Member WHERE MemberAge = 10
```

예 2) Member 라는 테이블에서 MemberAge 값이 20 보다 크거나 같고 MemberArea 가서울 인 경우

```
SELECT * FROM Member WHERE MemberAge >= 10 AND MemberArea = '서울'
```

(참고: 문자열을 조회하기 위해서는 '문자' 형태로 묶어야 한다.)

예 3) Member 라는 테이블에서 MemberAge 값이 10 보다 크거나 같고 20 보다 작거나 같은 경우

```
SELECT * FROM Member WHERE MemberAge >= 10 AND MemberAge <= 20
```

아래와 같은 방법도 가능 하다

```
SELECT * FROM Member WHERE MemberAge BETWEEN 10 AND 20
```

예 4) Member 라는 테이블에서 MemberName 이 '김'씨에 해당하는 모든 컬럼을 가져오려고 하는 경우

```
SELECT * FROM Member WHERE MemberName LIKE '김%'
```

(참고: %는 와일드 카드로서 모든 문자가 가능한 것을 말한다. 예를 들어 '김%' 으로 검색을 하게 되면 '김유신', '김유', '김' 모든 문자가 검색이 가능하다.

예 4) Member 라는 테이블에서 MemberName 이 특수문자 '%' 로 시작하는 모든 컬럼을 가져오는 경우

```
SELECT * FROM Member WHERE MemberName LIKE '[%]%'
```

아래와 같은 방법으로도 가능하다.

```
SELECT * FROM Member WHERE MemberName LIKE '₩%%' ESCAPE '₩'
```

예 5) Member 테이블에서 MemberAge 가 10 살 또는 20 살 인 데이터만 가져오는 경우

```
SELECT * FROM Member WHERE MemberAge = 10 OR MemberAge = 20
```

이렇게도 가능 하지만 아래와 같은 방법도 가능하다.

```
SELECT * FROM Member WHERE MemberAge IN (10,20)
```

이 외에도 CASE 문을 활용한 아래와 같은 형태로도 가능하다.

```
SELECT * FROM Member WHERE (CASE MemberAge WHEN 20 THEN 10 ELSE MemberAge END) = 10
```

Case 문(Transact-SQL): http://msdn.microsoft.com/ko-kr/library/ms181765.aspx

SELECT 문(Transact-SQL) Denali: http://msdn.microsoft.com/ko-kr/library/ms189499(v=SQL.110).aspx

SELECT 문(Transact-SQL) SQL Server 2008 R2:

http://msdn.microsoft.com/ko-kr/library/ms189499(v=SQL.100).aspx

[SQL2011 강좌] 12. SELECT(JOIN)

이번에는 DDL 에서 SELECT 절을 이용해 두 개 이상의 테이블에서 데이터를 가져올 경우 JOIN 을 활용해서 가지고 올 수 있다. 여기서 JOIN 의 방식들은 어떤 것이 있는지에 대해서 설명을 드리겠다.

JOIN 이란? A 테이블과 B 의 테이블의 결합으로 즉 두 개 이상의 테이블에 있는 데이터를 가지고 오는 수단이라고 보면 되겠다. JOIN 해서 얻을 수 있는 집합은 크게 3 가지로 구성되어있으며, OUTER JOIN 은 3 가지로 나뉜다.

1. CROSS JOIN
2. INNER JOIN
3. OUTER JOIN(LEFT, RIGHT, FULL OUTER)

예제를 위해 테이블을 만들어보도록 하겠다.

```
USE tempdb

GO

IF OBJECT_ID('JoinA','U') IS NOT NULL

        DROP TABLE JoinA

GO

IF OBJECT_ID('JoinB','U') IS NOT NULL

        DROP TABLE JoinB

GO

CREATE TABLE JoinA(idx INT NOT NULL)

GO

CREATE TABLE JoinB(idx INT NOT NULL)

GO
```

```
INSERT INTO JoinA VALUES(1),(2),(3)

INSERT INTO JoinB VALUES(1),(5)

GO
```

JoinA 테이블에는 1,2,3 의 row 를 가지고 있고 JoinB 테이블에는 1,5 의 row 를 가지고 있다.

1. 첫 번째, CROSS JOIN

CROSS JOIN 은 A 테이블에 n 개가 있고, B 테이블에 m 개가 있으면, 얻을 수 있는 row 는총 m*n 개에 해당이 된다. 즉 A 테이블에 있는 모든 원소와 B 테이블에 있는 모든 원소가 결합되는 테이블을 말한다.

예제를 살펴보도록 하겠다.

```
--//ANSI SQL-92 문법

SELECT

                A.IDX AS AIDX

,               B.IDX AS BIDX

FROM JoinA AS A --// AS A 는 JoinA 라는 테이블의 별칭이다.

CROSS JOIN JoinB        AS B --// AS B 는 JoinB 라는 테이블의 별칭이다.
```

Results | Messages

	AIDX	BIDX
1	1	1
2	2	1
3	3	1
4	1	5
5	2	5
6	3	5

A 에 있는 1,2,3 과 B 에 있는 1,5 와 모두 결합이 되어 나타나게 된다. 이게 바로 CROSS JOIN 이며, 보통 원본 테이블을 두 번 이상 읽어야 하는 경우에 한번만 읽어서 뻥튀기 하는데 주로 이용한다.

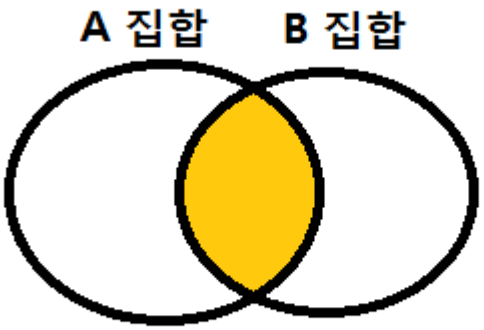
나중에 접하시겠지만, 열 => 행으로 바꿀 때 많이 사용하는 방법 중 하나 이다.

```
--//ANSI SQL-89 문법

SELECT

                A.IDX AS AIDX

,               B.IDX AS BIDX

FROM JoinA AS A , JoinB AS B
```

ANSI SQL-92 문법에 비해 오래된 ANSI SQL-89 문법은 중간에 CROSS JOIN 이란 단어가 없이 컴마로 구분된다. SQL Server Denali 에서는 ANSI SQL-92 와 89 문법을 모두 제공하지만, 되도록이면 최신에 계정된 ANSI SQL-92 문법을 사용하는 것이 좋다. 오래된 방식이다 보니 차기버전 혹은 그 후 버전에서는 사용하지 못할 수도 있기 때문이다.

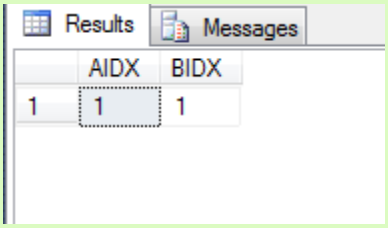
2. 두 번째, INNER JOIN(내부 조인).

INNER JOIN 은 A 테이블과 B 테이블간의 **교집합**을 의미한다.



예제를 바로 보도록 하겠다.

```
--//ANSI SQL-92

SELECT

                    A.IDX AS AIDX

,                   B.IDX AS BIDX

FROM                    JoinA           AS A

INNER JOIN JoinB                AS B ON A.idx = B.IDX
```



CROSS JOIN 는 두개의 테이블의 행이 조인이 되기 때문에 따로 조인 조건이 필요하지 않지만, INNER JOIN 은 서로 공통이 되는 키 값을 가지고 올 수 있도록 하기 위해 ON 절 뒤에연결고리가 될 수 있는 Key 를 지정해야 하는 것이 특징이다.

또한 중간에 INNER 라는 단어를 넣지 않아도 DEFAULT 값이기 때문에 똑같이 작동한다.

```
--//ANSI SQL-92

SELECT

                    A.IDX AS AIDX

,                   B.IDX AS BIDX

FROM                    JoinA       AS A

JOIN JoinB          AS B ON A.idx = B.IDX
```

INNER JOIN 의 ANSI SQL-89 버전은 아래와 같다.

```
--//ANSI SQL-89

SELECT

                    A.IDX AS AIDX

,                    B.IDX AS BIDX

FROM JoinA          AS A, JoinB AS B

WHERE A.IDX = B.IDX
```

또한 꼭 조건이 = 이 아니라도 가능하다.

```
--//ANSI SQL-92

SELECT

                    A.IDX AS AIDX

,                   B.IDX AS BIDX

FROM JoinA          AS A

INNER JOIN JoinB AS B ON A.IDX < B.IDX
```

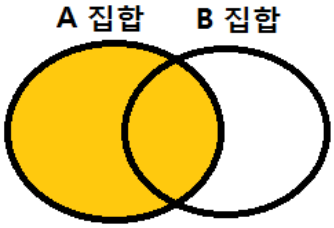
Results | Messages

	AIDX	BIDX
1	1	5
2	2	5
3	3	5

3. 세 번째, OUTER JOIN(외부 조인)

OUTER JOIN 의 종류는 3 가지(LEFT, RIGHT, FULL)가 있다.

1) LEFT OUTER JOIN



LEFT OUTER JOIN 예제를 살펴보도록 하겠다.

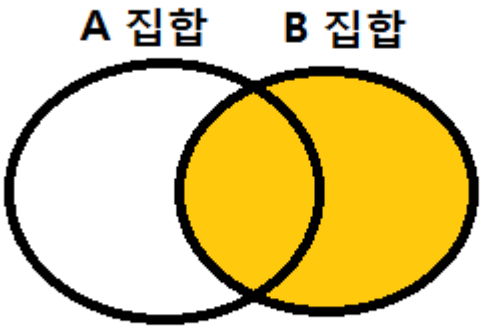
```
--//ANSI SQL-92

SELECT

                    A.IDX AS AIDX

,                   B.IDX AS BIDX

FROM            JoinA       AS A

LEFT OUTER JOIN JoinB AS B ON A.IDX = B.IDX
```

Results | Messages

	AIDX	BIDX
1	1	1
2	2	NULL
3	3	NULL

A 집합에 해당하는 TableA 에 있는 1,2,3 과 대칭이 되는 JoinB 와 매칭되어 나오게 된다. 매칭이 되지 않은 값(없는 값)에 대해서는 NULL 의 데이터를 가지고 나오게 된다. 위의 예제에서는 JoinB 에는 IDX 값이 2, 3 이 없기 때문에 NULL 을 반환하게 된다.

2) RIGHT OUTER JOIN



RIGHT OUTER JOIN 은 단지 기준이 되는 테이블의 위치가 변경 된 것을 말한다. 예제를 살펴보도록 하겠다.

```
--//ANSI SQL-92

SELECT

                    A.IDX AS AIDX

,                   B.IDX AS BIDX

FROM                JoinA AS A

RIGHT OUTER JOIN JoinB AS B ON A.IDX = B.IDX
```

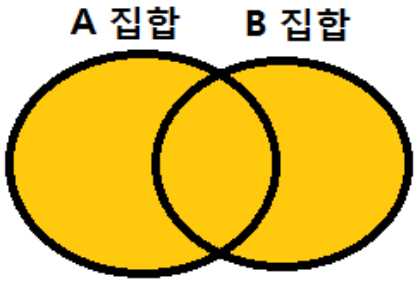
Results | Messages

	AIDX	BIDX
1	1	1
2	NULL	5

이제 좀 이해가 가시나요? 위의 결과는 아래와도 같겠죠~!

```
--//ANSI SQL-92

SELECT

                    A.IDX AS AIDX

,                   B.IDX AS BIDX

FROM                JoinB AS B

LEFT OUTER JOIN JoinA AS A ON A.IDX = B.IDX
```

3) FULL OUTER JOIN



FULL OUTER JOIN 은 두 개의 테이블에 조합이 가능한 모든 키를 매칭해서 보여준다. 예제를 살펴보도록 하겠다.

```
--//ANSI SQL-92

SELECT

                    A.IDX AS AIDX

,                   B.IDX AS BIDX

FROM                JoinA AS A

RIGHT OUTER JOIN JoinB AS B ON A.IDX = B.IDX
```

Results | Messages

	AIDX	BIDX
1	1	1
2	2	NULL
3	3	NULL
4	NULL	5

JoinA 테이블과 JoinB 테이블간의 조인 key 가 되는 IDX 가 조합이 가능한 모든 조합이 추출이 된다.

각 키값에 매칭이 되지 않는 데이터는 NULL 을 반환하게 된다.

[SQL2011 강좌] 13. UPDATE, DELETE 절

이번에는 DDL 에서 테이블의 데이터를 수정하는 UPDATE 절, 그리고 데이터를 삭제하는 DELETE 절에 대해서 알아보도록 하겠다. UPDATE 절에 대한 기본 구조는 아래와 같다.

```
UPDATE 테이블명

SET 컬럼 1 = 변경할 컬럼 1 의 값

, 컬럼 2 = 변경할 컬럼 2 의 값

.....

WHERE 조건
```

간단한 예제로 한번 살펴보도록 하겠다.

예 1) Member 테이블에서 MemberAge 값을 모두 10 으로 변경하려는 경우.

(모두 변경하기 때문에 WHERE 에 조건을 주지 않다.)

```
UPDATE Member

SET MemberAge = 10
```

예 2) Member 테이블에서 MemberArea 가 '서울' 인 사람들만 MemberAge 를 20 으로 변경

```
UPDATE Member

SET MemberAge = 20

WHERE MemberArea = '서울'
```

(참고: 문자열을 조회하기 위해서는 '문자' 형태로 묶어야 한다.)

예 3) Member 테이블에서 MemberArea 가 '서울' 인 사람들은 MemberAge 를 15 로 변경하고, MemberArea 가 '부산'인 사람들은 MemberAge 를 25 로 변경하자.

(조건이 두개라 쿼리를 두개로 나눠서 처리하게 된다.)

```
UPDATE Member

SET MemberAge = 15

WHERE MemberArea = '서울'

UPDATE Member

SET MemberAge = 25

WHERE MemberArea = '부산'
```

하지만, CASE 문을 활용해서 한번에 처리할 수도 있다.

```
UPDATE Member

SET MemberAge = (CASE MemberArea

WHEN '서울' THEN 15

WHEN '부산' THEN 25

END)

WHERE MemberArea IN ('서울','부산')
```

Case 문(Transact-SQL): http://msdn.microsoft.com/ko-kr/library/ms181765.aspx

물론 업데이트 역시 JOIN 문을 활용할 수 있다.

예를 들어 Login 이라는 테이블이 있고, 여기서 한번 이상 로그인 한 유저에 대해서 MemberAge 를 10 으로 바꾸는 예제 이다.

```
UPDATE Member

SET MemberAge = 10

FROM Member

INNER JOIN Login ON Member.MemberIDX = Login.MemberIDX
```

지금까지 UPDATE 절을 아주 간단하게 활용하는 방법에 대해서 알아봤다.

사실 이건 정말 기초적인 부분이고, UPDATE 절 또한 엄청 복잡하게 구성되어 있다.

UPDATE(Transact-SQL) Denali: http://msdn.microsoft.com/en-us/library/ms177523(v=SQL.110)

UPDATE(Transact-SQL) SQL Server 2008 R2: http://msdn.microsoft.com/ko-kr/library/ms177523(v=SQL.100)

DELETE 절은 매우 위험하기 때문에 조심스럽게 사용하는 것이 좋다.

실수로 DELETE 절로 데이터를 날렸을 때에 대해서는 강좌 후반부에 설명드릴 데이터베이스 복원 때 보다 자세히 설명을 드리도록 하겠다.

DELETE 절에 대한 기본 구조는 아래와 같다.

```
DELETE FROM 테이블명

WHERE 조건 ...
```

간단한 예제로 한번 살펴보도록 하겠다.

예 1) Member 테이블의 모든 데이터를 삭제할 경우

```
DELETE FROM Member
```

예 2) Member 테이블에서 MemberAge 값 중 10 인 데이터를 삭제할 경우

(모두 변경하기 때문에 WHERE 에 조건을 주지 않는다.)

```
DELETE FROM Member

WHERE MemberAge = 10
```

[SQL2011 강좌] 14. DISTINCT, UNION, UNION ALL 키워드

이번에는 DISTINCT, UNION, UNION ALL 에 대해서 살펴보도록 하겠다.

1. DISTINCT

DISTINCT 키워드는 중복을 제거하는 명령어 이다. 중복을 제거하기 때문에 GROUP BY 와 비슷하게 볼 수 있지만 다른 점이 분명히 있다.

이제 예제를 통해 살펴보도록 하겠다.

예제를 사용하기 위해 데이터를 넣도록 하겠다.

```
USE tempdb

GO

IF OBJECT_ID('test','U') IS NOT NULL

                        DROP TABLE test

GO

CREATE TABLE test(number INT NOT NULL, name VARCHAR(10))

GO

INSERT INTO test(number, name) VALUES(1, '가')

INSERT INTO test(number, name) VALUES(1, '나')

INSERT INTO test(number, name) VALUES(1, '다')

INSERT INTO test(number, name) VALUES(2, '라')

INSERT INTO test(number, name) VALUES(2, '마')

INSERT INTO test(number, name) VALUES(2, '바')
```

```
INSERT INTO test(number, name) VALUES(3, '사')

INSERT INTO test(number, name) VALUES(4, '사')

INSERT INTO test(number, name) VALUES(5, '가')

INSERT INTO test(number, name) VALUES(6, '나')
```

기존에 배웠던 number 를 GROUP BY 로 묶는 방법

```
SELECT
        number
FROM test
GROUP BY number
```

하지만 이 경우는 DISTINCT 로도 가능하다.

```
SELECT
        DISTINCT
        number
FROM test
```

하지만 DISTINCT 는 중복을 제거만 할 뿐, 중복된 행 수에 대해서는 알 수가 없기 때문에 아래 쿼리는 에러가 발생한다.

```
SELECT
        DISTINCT
```

```
                    number

,                   COUNT(*)

FROM test

에러!!
```

바로 이게 DISTINCT 와 GROUP BY 의 차이라고 보면 되겠다.

2. UNION 과 UNION ALL

UNION 은 두 개 이상의 테이블을 결합할 때 주로 사용한다. 이는 조인이랑 다소 똑같다고 생각을 할 수 있겠지만~! 조인은 반드시(CROSS JOIN 제외) 두 개의 테이블에 공통된 KEY 가 있어야 하지만, UNION 은 다른 테이블이라도 나열된 컬럼의 속성이 같다면 합칠 수 있다.

예제를 사용하기 위해 데이터를 넣도록 하겠다.

```
USE tempdb

GO

IF OBJECT_ID('testA','U') IS NOT NULL

                    DROP TABLE testA

GO

IF OBJECT_ID('testB','U') IS NOT NULL

                    DROP TABLE testB

GO

CREATE TABLE testA(number INT NOT NULL, name VARCHAR(10))

GO

CREATE TABLE testB(idx INT NOT NULL, name VARCHAR(10))
```

```
GO

INSERT INTO testA(number, name) VALUES(1, '가')

INSERT INTO testA(number, name) VALUES(2, '나')

INSERT INTO testA(number, name) VALUES(3, '다')

INSERT INTO testB(idx, name) VALUES(2, '라')

INSERT INTO testB(idx, name) VALUES(3, '마')

INSERT INTO testB(idx, name) VALUES(4, '바')

INSERT INTO testB(idx, name) VALUES(5, '사')
```

TestA 테이블과 TestB 테이블을 만들고 각 데이터를 넣었다.

나열되는 컬럼이 다르다면? 당연히 에러가 발생한다.

```
SELECT number FROM testA

UNION

SELECT idx, name FROM testB

에러!!
```

컬럼명은 다르지만, 타입이 맞고 순서가 맞다면

```
SELECT number FROM testA

UNION

SELECT idx FROM testB
```

	number
1	1
2	2
3	3
4	4
5	5

testA 테이블에는 1~3, testB 테이블에는 2~5 의 데이터가 들어가 있었지만, 중복이 제거된 1,2,3,4,5 만 추가 되어있는 것을 볼 수 있다.

UNION 으로 두 개의 테이블을 결합하게 되면, 중복을 제거하는 연산을 한번 더 수행하기 때문에 그렇다.

만약 중복 제거 없이 모두 출력하기 위해서는 어떻게 해야 할까?

이때 UNION ALL 을 사용하면 된다.

```
SELECT number FROM testA

UNION ALL

SELECT idx FROM testB
```

	number
1	1
2	2
3	3
4	2
5	3
6	4
7	5

대용량의 테이블끼리 서로 UNION 을 시킨다면, 테이블을 결합하고 중복된 행을 제거하기 위한 작업을 내부적으로 또 수행하게 된다. (내부적으로는 데이터를 정렬을 시킨 뒤에 중복을 제거하는 형태로 진행을 한다.)

결국 결합하는 테이블이 중복되지 않는다고 가정을 하면 불필요한 중복 제거작업을 시키지 않기 위해 UNION ALL 을 사용하시는 것이 좋다.

[SQL2011 강좌] 15. ORDER BY, GROUP BY, HAVING

이번에는 ORDER BY, GROUP BY 그리고 HAVING 에 대해서 살펴보도록 하겠다.

1. ORDER BY

ORDER BY 는 데이터를 정렬해서 출력하기 위해 사용된다. 정렬을 하기 때문에 내림차순과 오름차순으로 정렬할 수 있다. 오름차순과 내림차순을 헷갈려 하시는 분이 있을 것 같아서 설명 드리도록 하겠다. 오름차순은 낮은 값부터 높은 값으로 나열하는 것을 말한다. 예를 들면 1,2,3,4,5 이런 방식이 오름차순으로 보면 된다. 내림차순은 오름차순의 반대라고 보면 된다. 높은 값부터 낮은 값으로 나열하는 것을 말한다. 예를 들면 하,사,나,가 이런 방식이 내림차순으로 보면 된다. SQL Server 에서는 기본적으로 오름 차순이 기본 값으로 되어있다. ASC 라는 단어로 사용한다. 내림차순은 DESC 라는 단어를 사용한다.

예 1) MemberAge 를 오름차순으로 정렬하는 방법

```
SELECT * FROM Member ORDER BY MemberAge

SELECT * FROM Member ORDER BY MemberAge ASC
```

기본 값이기 때문에 명시하지 않아도 오름차순으로 정렬하도록 되어있다.

예 2) MemberName 으로 내림차순으로 정렬하는 방법

```
SELECT * FROM Member ORDER BY MemberName DESC
```

이제 예제를 통해 살펴보도록 하겠다.

예제를 사용하기 위해 데이터를 넣도록 하겠다.

```
USE tempdb

GO

IF OBJECT_ID('test','U') IS NOT NULL
```

```
DROP TABLE test

GO

CREATE TABLE test(number INT NOT NULL, name VARCHAR(10))

GO

INSERT INTO test(number, name) VALUES(1, '가')

INSERT INTO test(number, name) VALUES(1, '나')

INSERT INTO test(number, name) VALUES(1, '다')

INSERT INTO test(number, name) VALUES(2, '라')

INSERT INTO test(number, name) VALUES(2, '마')

INSERT INTO test(number, name) VALUES(2, '바')

INSERT INTO test(number, name) VALUES(3, '사')

INSERT INTO test(number, name) VALUES(4, '사')

INSERT INTO test(number, name) VALUES(5, '가')

INSERT INTO test(number, name) VALUES(6, '나')
```

간단한 예제로 한번 살펴보도록 하겠다.

예 1) number 로 내림 차순 하는 방법

```
SELECT * FROM test ORDER BY number DESC
```

Results | Messages

	number	name
1	6	나
2	5	가
3	4	사
4	3	사
5	2	라
6	2	마
7	2	바
8	1	가
9	1	나
10	1	다

ORDER BY 는 두 개 이상의 컬럼도 가능하다.

예 2) number 로 내림차순, name 내림차순

Number 가 같은 값이면 name 은 내림 차순이 될 것 이다.

```
SELECT * FROM test ORDER BY number DESC, name DESC
```

Results | Messag

	number	name
1	6	나
2	5	가
3	4	사
4	3	사
5	2	바
6	2	마
7	2	라
8	1	다
9	1	나
10	1	가

(참고: 문자열을 조회하기 위해서는 '문자' 형태로 묶어야 한다.)

조금 더 응용해보도록 하겠다.

예 3) number 로 오름차순으로 하되, 1 인 값을 가장 아래로 보내려면? Case 문을 활용하면 가능하다.

```
SELECT * FROM test

ORDER BY (CASE number WHEN 1 THEN 99 ELSE number END) DESC
```

Results | Messages

	number	name
1	2	라
2	2	마
3	2	바
4	3	사
5	4	사
6	5	가
7	6	나
8	1	가
9	1	나
10	1	다

2. GROUP BY 그리고 HAVING...

GROUP BY 는 데이터를 그룹화 시킬 때 주로 사용한다. 저도 SQL 을 처음 다뤘을 때 GROUP BY 부분이 엄청 헷갈려 고생한적이 있다. 저 말고도 분명 그런 분들이 있을 것 같다. 많은 예제로 쉽게 이해할 수 있도록 설명 드리겠다. HAVING 은 GROUP BY 로 그룹화에 대한 WHERE 절이라고 보면 된다. 말이 너무 어렵죠? 예제를 바로 보면서 설명 드리겠다.

예 1) 각 number 가 있는 행 수를 가져오는 방법

```
SELECT

  Number

, COUNT(*) AS TotalCount

FROM test

GROUP BY number
```

Results | Messages

	Number	TotalCount
1	1	3
2	2	3
3	3	1
4	4	1
5	5	1
6	6	1

중복되지 않게 NUMBER 로 그룹화가 되었고, TotalCount 로 행 수를 출력할 수 있었다.

COUNT 함수에 대해서 따로 설명을 드리진 않았지만, 행의 총 개수를 가져옵니다.

COUNT(Transact-SQL): http://msdn.microsoft.com/ko-kr/library/ms175997(v=SQL.100).aspx

name 으로도 그룹화를 시켜보도록 하겠다.

```
SELECT

  name

, COUNT(*) AS TotalCount
```

```
FROM test

GROUP BY name
```

	name	TotalCount
1	가	2
2	나	2
3	다	1
4	라	1
5	마	1
6	바	1
7	사	2

Number 와 name 을 그룹으로 묶는다면?

```
SELECT

  number

, name

, COUNT(*) AS TotalCount

FROM test

GROUP BY number, name
```

	number	name	TotalCount
1	1	가	1
2	5	가	1
3	1	나	1
4	6	나	1
5	1	다	1
6	2	라	1
7	2	마	1
8	2	바	1
9	3	사	1
10	4	사	1

중복된 number, name 조합은 없기 때문에 모든 열이 출력된다.

이제 HAVING 을 이용해서 조건을 주도록 해본다.

예) number 로 그룹화를 시킨 상태에서 number 가 2~4 인 데이터를 출력하고자 한다면?

```
SELECT
  number
, COUNT(*) AS TotalCount
FROM test
GROUP BY number
HAVING number BETWEEN 2 AND 4
```

	number	TotalCount
1	2	3
2	3	1
3	4	1

HAVING 은 반드시 GROUP BY 에 명시된 열에 대해서만 조건을 줄 수 있다. 물론 당연한 이야기 이다. 이제 GROUP BY, HAVING 그리고 WHERE 까지 결합해 보면

예) 그룹으로 묶인 number 와 row count 를 출력하되, number 는 2~4 이며, name 은 '바'인 것만 출력하시오.

```
SELECT
  number
, COUNT(*) AS TotalCount
FROM test
WHERE name ='바'
GROUP BY number
HAVING number BETWEEN 2 AND 4
```

아래 같은 쿼리는 에러가 발생된다.

```
SELECT
number
, name
```

```
, COUNT(*) AS TotalCount

FROM test

WHERE name ='바'

GROUP BY number

HAVING number BETWEEN 2 AND 4

에러!!
```

Name 이 추가되었는데 이 구문이 에러를 발생시킨다. Name 은 group by 에 지정이 되어있지 않았기 때문에 에러가 나게 되어있다. 마지막으로 GROUP BY 는 대부분의 경우 정렬 순서를 보장하지 않는다. 따라서 그룹으로 묶인 열에 대해서 정렬이 필요하다면, 반드시 ORDER BY 로 명시해주는 것이 좋다.

만약 ORDER BY 에 명시한 컬럼이 GROUP BY 에 없다면 이 또한 에러가 된다.

```
SELECT

                number

FROM test

GROUP BY number

ORDER BY number, name desc

에러!!
```

지금까지 ORDER BY, GROUP BY 그리고 HAVING 에 대해서 간단한 예제로 살펴보았다.

ORDER BY(Transact-SQL): http://msdn.microsoft.com/ko-kr/library/ms188385.aspx

GROUP BY(Transact-SQL): http://msdn.microsoft.com/ko-kr/library/ms177673.aspx

[SQL2011 강좌] 16. SubQuery(서브쿼리)

이번에는 서브 쿼리에 대해서 살펴보도록 하겠다.

서브 쿼리는 크게 총 4 가지 형태로 나뉠 수 있다.

1. 단일 행 서브쿼리
2. 다중 행 서브쿼리
3. 인라인 뷰
4. 상관관계 서브쿼리

예제 Database 설치: http://www.sqler.com/390101

일단 단일 행을 살펴보기 전에 단일 행을 반환하는 함수 몇 가지에 대해서 소개해볼까 한다.

```
USE pubs

GO

SELECT

  COUNT(*) AS Count1 --// 총 행 수

, COUNT(price) AS Count2 --// 총 행 수를 구하되,price 가 null 이 아닌 것만

, SUM(Price) AS SumPrice --//price 의 총 합(null 은 무시됨)

, AVG(Price) AS AvgPrice --//price 의 평균 값(null 은 무시됨)

, MIN(Price) AS MinPrice --//price 의 최소 값(null 은 무시됨)

, MAX(Price) AS MaxPrice --//price 의 최대 값(null 은 무시됨)

from TITLES
```

	Count1	Count2	SumPrice	AvgPrice	MinPrice	MaxPrice
1	18	16	236.26	14.7662	2.99	22.95

Price 의 값 중 null 인 데이터가 두개가 존재하기 때문에, COUNT(price)는 16 을 반환하게 된다.

TOP Keyword

```
--//하나만 가져오기 위해서

SELECT TOP (1) title_id, title FROM titles
```

	title_id	title
1	PC1035	But Is It User Friendly?

```
--//10 %의 데이터를 가져오는 방법.

SELECT TOP 10 PERCENT title_id, title FROM titles
```

	title_id	title
1	PC1035	But Is It User Friendly?
2	PS1372	Computer Phobic AND Non-Phobic Individuals: Beha...

```
--//이런식으로 괄호안에 단일행 반환으로 이용할 수도 있음.

SELECT TOP (SELECT 10) title_id, title FROM titles
```

	title_id	title
1	PC1035	But Is It User Friendly?
2	PS1372	Computer Phobic AND Non-Phobic Individuals: Beha...
3	BU1111	Cooking with Computers: Surreptitious Balance Sheets
4	PS7777	Emotional Security: A New Algorithm
5	TC4203	Fifty Years in Buckingham Palace Kitchens
6	PS2091	Is Anger the Enemy?
7	PS2106	Life Without Fear
8	PC9999	Net Etiquette
9	TC3218	Onions, Leeks, and Garlic: Cooking Secrets of the M...
10	PS3333	Prolonged Data Deprivation: Four Case Studies

1. 단일 행 서브쿼리

단일 행 서브쿼리는 위의 TOP 예제처럼 괄호 안의 쿼리에서 한 개의 행을 반환하는 쿼리를 이용해 서브쿼리를 작성하는 방법이다. 예제를 통해서 살펴보도록 하겠다.

예) price 가 가장 싼 titles 의 판매정보(sales) 데이터 추출 하기

```
select * from sales

where title_id = (select top 1 title_id from titles where price > 0 order by price)
```

	stor_id	ord_num	ord_date	qty	payterms	title_id
1	7896	X999	1993-02-21 00:00:00.000	35	ON invoice	BU2075

예) 첫 번째인 title_id 의 판매정보(sales) 데이터 추출하기

```
select * from sales

where title_id = (select MIN(title_id) from titles)
```

	stor_id	ord_num	ord_date	qty	payterms	title_id
1	6380	6871	1994-09-14 00:00:00.000	5	Net 60	BU1032
2	8042	423LL930	1994-09-14 00:00:00.000	10	ON invoice	BU1032

그렇다면 한 개 말고 두 개 이상의 행일 때는 어떻게 될까?

```
set language '한국어'

select * from sales

where title_id = (select title_id from titles) --//서브쿼리에 여러행 반환하도록 수정

Msg 512, Level 16, State 1, Line 1

하위 쿼리에서 값을 둘 이상 반환했다. 하위 쿼리 앞에 =, !=, <, <=, >, >= 등이
오거나 하위 쿼리가 하나의 식으로 사용된 경우에는 여러 값을 반환할 수 없다.
```

그렇다면, 반환되는 행이 두 개 이상일 경우에는 어떻게 해야 할까? 이럴 때 사용하는 것이 다중 행 서브쿼리가 되겠다.

2. 다중 행 서브쿼리

반환되는 행이 여러 행인 경우에는 위의 에러처럼 = 를 사용할 수 없다. 바로 in 연산자를 사용해야 한다.

```
select * from sales

where title_id in (select title_id from titles)
```

	stor_id	ord_num	ord_date	qty	payterms	title_id
1	6380	6871	1994-09-14 00:00:00.000	5	Net 60	BU1032
2	6380	722a	1994-09-13 00:00:00.000	3	Net 60	PS2091
3	7066	A2976	1993-05-24 00:00:00.000	50	Net 30	PC8888
4	7066	QA7442.3	1994-09-13 00:00:00.000	75	ON invoice	PS2091
5	7067	D4482	1994-09-14 00:00:00.000	10	Net 60	PS2091
6	7067	P2121	1992-06-15 00:00:00.000	40	Net 30	TC3218
7	7067	P2121	1992-06-15 00:00:00.000	20	Net 30	TC4203
8	7067	P2121	1992-06-15 00:00:00.000	20	Net 30	TC7777
9	7131	N914008	1994-09-14 00:00:00.000	20	Net 30	PS2091
10	7131	N914014	1994-09-14 00:00:00.000	25	Net 30	MC3021
11	7131	P3087a	1993-05-29 00:00:00.000	20	Net 60	PS1372
12	7131	P3087a	1993-05-29 00:00:00.000	25	Net 60	PS2106
13	7131	P3087a	1993-05-29 00:00:00.000	15	Net 60	PS3333
14	7131	P3087a	1993-05-29 00:00:00.000	25	Net 60	PS7777
15	7896	QQ2299	1993-10-28 00:00:00.000	15	Net 60	BU7832
16	7896	TQ456	1993-12-12 00:00:00.000	10	Net 60	MC2222
17	7896	X999	1993-02-21 00:00:00.000	35	ON invoice	BU2075
18	8042	423LL922	1994-09-14 00:00:00.000	15	ON invoice	MC3021
19	8042	423LL930	1994-09-14 00:00:00.000	10	ON invoice	BU1032
20	8042	P723	1993-03-11 00:00:00.000	25	Net 30	BU1111

In 말고 not in 도 가능하다.

Price 가 가장 높은 상위 10 percent 의 책을 제외한 판매 정보이다.

```
select * from sales

where title_id not in (select top 10 percent title_id from titles order by price desc)
```

	stor_id	ord_num	ord_date	qty	payterms	title_id
1	6380	6871	1994-09-14 00:00:00.000	5	Net 60	BU1032
2	6380	722a	1994-09-13 00:00:00.000	3	Net 60	PS2091
3	7066	A2976	1993-05-24 00:00:00.000	50	Net 30	PC8888
4	7066	QA7442.3	1994-09-13 00:00:00.000	75	ON invoice	PS2091
5	7067	D4482	1994-09-14 00:00:00.000	10	Net 60	PS2091
6	7067	P2121	1992-06-15 00:00:00.000	40	Net 30	TC3218
7	7067	P2121	1992-06-15 00:00:00.000	20	Net 30	TC4203
8	7067	P2121	1992-06-15 00:00:00.000	20	Net 30	TC7777
9	7131	N914008	1994-09-14 00:00:00.000	20	Net 30	PS2091
10	7131	N914014	1994-09-14 00:00:00.000	25	Net 30	MC3021
11	7131	P3087a	1993-05-29 00:00:00.000	25	Net 60	PS2106
12	7131	P3087a	1993-05-29 00:00:00.000	15	Net 60	PS3333
13	7131	P3087a	1993-05-29 00:00:00.000	25	Net 60	PS7777
14	7896	QQ2299	1993-10-28 00:00:00.000	15	Net 60	BU7832
15	7896	TQ456	1993-12-12 00:00:00.000	10	Net 60	MC2222
16	7896	X999	1993-02-21 00:00:00.000	35	ON invoice	BU2075
17	8042	423LL922	1994-09-14 00:00:00.000	15	ON invoice	MC3021
18	8042	423LL930	1994-09-14 00:00:00.000	10	ON invoice	BU1032
19	8042	P723	1993-03-11 00:00:00.000	25	Net 30	BU1111

3. 인라인 뷰

인라인 뷰는 FROM 절 뒤에 테이블 명이 나와야 하지만 서브쿼리가 하나의 가상의 테이블을 반환함으로써 사용되는 경우를 말한다. 바로 예제를 통해서 알아보도록 하겠다.

예 1) 한번 이상 판매된 책들의 titles 정보의 데이터 반환

```
select *
from
(
        select
                title_id
,                 COUNT(*) as TotalCount
        from sales
        group by title_id
) as a
inner join titles as b on a.title_id = b.title_id
```

	title_id	TotalCount	title_id	title	type	pub_id	price	advanc
1	BU1032	2	BU1032	The Busy Executive's Database Guide	business	1389	19.99	5000.0
2	MC3021	2	MC3021	The Gourmet Microwave	mod_cook	0877	2.99	15000.
3	PS2091	4	PS2091	Is Anger the Enemy?	psychology	0736	10.95	2275.0

예 2) 3 번 이상 판매된 책들의 titles 정보의 데이터 반환

```
select *
from
(
        select
```

```
                                    title_id

,                                     COUNT(*) as TotalCount

                  from sales

                  group by title_id

                  having COUNT(*) >=3

) as a

inner join titles as b on a.title_id = b.title_id
```

	title_id	TotalCount	title_id	title	type	pub_id	price	advance	royalty	ytd_
1	PS2091	4	PS2091	Is Anger the Enemy?	psychology	0736	10.95	2275.00	12	204

이처럼 from 절 뒤에 나오는 괄호의 데이터가 다중 행이 되면서 하나의 가상의 테이블이 되는 경우를 인라인 뷰라고 한다. 정말 많은 곳에 사용되고, 필수로 익혀야 할 기술 이다.

4. 상관관계 서브쿼리

상관관계 서브쿼리는 말이 참 어려운데 아마도 번역 때문에 그렇게 된 것 같다. 참으로 안타깝다. 예제를 보면 좀 더 빠르게 이해가 될 수 있을 것 이다. 판매 정보 테이블에서 최근에 주문된 10 건의 데이터를 title 과 함께 출력하라

```
SELECT

                    TOP 10

                    stor_id

,                   ord_num

,                   ord_date

,                   title_id

,                   (select top

1 title from titles where title_id = a.title_id) as titles

FROM sales a

order by ord_date desc
```

	stor_id	ord_num	ord_date	title_id	titles
1	7067	D4482	1994-09-14 00:00:00.000	PS2091	Is Anger the Enemy?
2	7131	N914008	1994-09-14 00:00:00.000	PS2091	Is Anger the Enemy?
3	7131	N914014	1994-09-14 00:00:00.000	MC3021	The Gourmet Microwave
4	6380	6871	1994-09-14 00:00:00.000	BU1032	The Busy Executive's Database Guide
5	8042	423LL922	1994-09-14 00:00:00.000	MC3021	The Gourmet Microwave
6	8042	423LL930	1994-09-14 00:00:00.000	BU1032	The Busy Executive's Database Guide
7	6380	722a	1994-09-13 00:00:00.000	PS2091	Is Anger the Enemy?
8	7066	QA7442.3	1994-09-13 00:00:00.000	PS2091	Is Anger the Enemy?
9	7896	TQ456	1993-12-12 00:00:00.000	MC2222	Silicon Valley Gastronomic Treats
10	7896	QQ2299	1993-10-28 00:00:00.000	BU7832	Straight Talk About Computers

판매정보 테이블에는 title_id 만 있고 title 는 없다. 따라서 이 정보를 가져오기 위해서는 titles 에 조인을 해서 데이터를 가져와야 한다. 위의 예제의 상관관계 서브쿼리는 바깥쪽의 sales 테이블에 대해서 10 개의 행을 가지고 오고, 각 title 를 가져오기 위해 괄호 안에 있는 쿼리를 10 번을 수행하게 된다. 이때 두 개의 쿼리가 상관이 있다고 하여 상관관계 서브쿼리라 부릅니다. 사실 이는 조인으로도 풀 수 있다.

```
SELECT
                    TOP 10
                    a.stor_id
,                   a.ord_num
,                   a.ord_date
,                   a.title_id
,                   b.title
FROM sales a
inner join titles b on a.title_id = b.title_id
order by ord_date desc
```

결과는 위의 쿼리와 똑같다.

서브 쿼리는 SQL Server 에선 없어서 안될 아주 중요한 요소 이다. 다양하게 많이 쓰이고 있고, 제가 소개한 방법을 기본으로 삼아 응용과 활용한다면, 보다 다양한 형태로 쿼리를 작성할 수 있다.

[SQL2011 강좌] 17. 뷰에 대한 이해 및 생성, 수정, 삭제

이번에 소개해 드릴 내용은 뷰 이다. 중요한 DB 의 개체 이다. 뷰는 가상의 테이블이다.

지난 서브쿼리 강좌에서 인라인 뷰라는 예제를 통해 저희는 뷰라는 단어를 이미 한 번쯤은 들어봤다.

뷰는 - 테이블의 거울이다.

뷰는 테이블의 거울이지만 거울은 원래 모습을 다양하게 비출 수도 있다.

뷰는 테이블의 거울이다라는 샘플을 보여 드리지요.

```
USE pubs

GO

--konan_view_test1 이라는 뷰를 생성한다.

CREATE VIEW konan_view_test1

AS

SELECT * FROM titles

GO

--뷰를 조회 한다.

SELECT * FROM konan_view_test1

GO
```

	title_id	title	type	pub_id	price	advance	royalty	ytd_sales	notes
1	BU1032	The Busy Executive's Database Guide	business	1389	19,99	5000,00	10	4095	An overview of ava
2	BU1111	Cooking with Computers: Surreptitious Balance Sheets	business	1389	11,95	5000,00	10	3876	Helpful hints on ho
3	BU2075	You Can Combat Computer Stress!	business	0736	2,99	10125,00	24	18722	The latest medical
4	BU7832	Straight Talk About Computers	business	1389	19,99	5000,00	10	4095	Annotated analysis
5	MC2222	Silicon Valley Gastronomic Treats	mod_cook	0877	19,99	0,00	12	2032	Favorite recipes fo
6	MC3021	The Gourmet Microwave	mod_cook	0877	2,99	15000,00	24	22246	Traditional French
7	MC3026	The Psychology of Computer Cooking	UNDECIDED	0877	NULL	NULL	NULL	NULL	NULL
8	PC1035	But Is It User Friendly?	popular_comp	1389	22,95	7000,00	16	8780	A survey of softwa
9	PC8888	Secrets of Silicon Valley	popular_comp	1389	20,00	8000,00	10	4095	Muckraking reporti
10	PC9999	Net Etiquette	popular_comp	1389	NULL	NULL	NULL	NULL	A must-read for co
11	PS1372	Computer Phobic AND Non-Phobic Individuals: Beh...	psychology	0877	21,59	7000,00	10	375	A must for the spe
12	PS2091	Is Anger the Enemy?	psychology	0736	10,95	2275,00	12	2045	Carefully research

자 데이터를 조회해 보면? 테이블과 같은 처리가 된다. 뷰를 다르게 생성해 쓸 수 있다.

```
USE pubs

GO

--konan_view_test2 이라는 뷰를 생성한다.

CREATE VIEW konan_view_test2

AS

SELECT title, type, price FROM titles

GO

--뷰를 조회 한다.

SELECT * FROM konan_view_test2

GO
```

	title	type	price
1	The Busy Executive's Database Guide	business	19,99
2	Cooking with Computers: Surreptitious Balance Sheets	business	11,95
3	You Can Combat Computer Stress!	business	2,99
4	Straight Talk About Computers	business	19,99
5	Silicon Valley Gastronomic Treats	mod_cook	19,99
6	The Gourmet Microwave	mod_cook	2,99
7	The Psychology of Computer Cooking	UNDECIDED	NULL
8	But Is It User Friendly?	popular_comp	22,95
9	Secrets of Silicon Valley	popular_comp	20,00
10	Net Etiquette	popular_comp	NULL
11	Computer Phobic AND Non-Phobic Individuals: Beh...	psychology	21,59
12	Is Anger the Enemy?	psychology	10,95
13	Life Without Fear	psychology	7,00
14	Prolonged Data Deprivation: Four Case Studies	psychology	19,99
15	Emotional Security: A New Algorithm	psychology	7,99
16	Onions, Leeks, and Garlic: Cooking Secrets of the M...	trad_cook	20,95

이렇게 쓸 수도 있다. 사용자가 볼 경우를 생각해 보면 일반 사용자들이 볼 경우

title_id 라는 이상한 값들이 필요하다. pubs_id 라는 출판사 ID 의 고유 값이 의미가 없다.

뷰를 이용하면 사용자가 원하는 데이터에만 초점을 맞추게 할 수 있다.

또 다른 뷰의 샘플을 보면

```
USE pubs

GO

--konan_view_test3 이라는 뷰를 생성한다.

CREATE VIEW konan_view_test3

AS

SELECT

        titles.title

,       authors.au_lname

,       publishers.pub_name

FROM        dbo.titles

INNER JOIN dbo.titleauthor ON titles.title_id = titleauthor.title_id

INNER JOIN dbo.authors      ON titleauthor.au_id = authors.au_id

INNER JOIN dbo.publishers  ON titles.pub_id = publishers.pub_id

GO

--뷰를 조회한다

SELECT * FROM konan_view_test3

GO
```

	title	au_lname	pub_name
1	The Busy Executive's Database Guide	Green	Algodata Infosystems
2	The Busy Executive's Database Guide	Bennet	Algodata Infosystems
3	Cooking with Computers: Surreptitious Balance Sheets	O'Leary	Algodata Infosystems
4	Cooking with Computers: Surreptitious Balance Sheets	MacFeather	Algodata Infosystems
5	You Can Combat Computer Stress!	Green	New Moon Books
6	Straight Talk About Computers	Straight	Algodata Infosystems
7	Silicon Valley Gastronomic Treats	del Castillo	Binnet & Hardley
8	The Gourmet Microwave	DeFrance	Binnet & Hardley
9	The Gourmet Microwave	Ringer	Binnet & Hardley
10	But Is It User Friendly?	Carson	Algodata Infosystems
11	Secrets of Silicon Valley	Dull	Algodata Infosystems
12	Secrets of Silicon Valley	Hunter	Algodata Infosystems
13	Net Etiquette	Locksley	Algodata Infosystems
14	Computer Phobic AND Non-Phobic Individuals: Beh...	MacFeather	Binnet & Hardley
15	Computer Phobic AND Non-Phobic Individuals: Beh...	Karsen	Binnet & Hardley
16	Is Anger the Enemy?	Ringer	New Moon Books
17	Is Anger the Enemy?	Ringer	New Moon Books

대강 봐도 조인을 3 개의 테이블과 하는데 이럴 경우 저렇게 조인구문을 길게 쓰는 것 보다는 뷰를 생성해 놓고 사용하는 것이 장점이 많다. 이제 뷰를 생성하는 방법에 대해서 알아보도록 하겠다.

뷰의 생성 구문은 아래와 같다.

CREATE VIEW [< database_name > .] [< owner > .] view_name [(column [,...n])]

[WITH < view_attribute > [,...n]]

AS

select_statement

[WITH CHECK OPTION]

< view_attribute > ::=

{ ENCRYPTION | SCHEMABINDING | VIEW_METADATA }

CREATE VIEW(Transact-SQL) Denali:

http://msdn.microsoft.com/en-us/library/ms187956(v=SQL.110)

```
USE pubs

GO

CREATE VIEW titles_view

AS
```

```
SELECT title, type, price, pubdate

FROM titles

GO

--뷰를 조회한다.

SELECT * FROM titles_view

GO
```

	title	type	price	pubdate
1	The Busy Executive's Database Guide	business	19,99	1991-06-12 00:00:00,000
2	Cooking with Computers: Surreptitious Balance Sheets	business	11,95	1991-06-09 00:00:00,000
3	You Can Combat Computer Stress!	business	2,99	1991-06-30 00:00:00,000
4	Straight Talk About Computers	business	19,99	1991-06-22 00:00:00,000
5	Silicon Valley Gastronomic Treats	mod_cook	19,99	1991-06-09 00:00:00,000
6	The Gourmet Microwave	mod_cook	2,99	1991-06-18 00:00:00,000
7	The Psychology of Computer Cooking	UNDECIDED	NULL	2011-06-16 00:02:05,793
8	But Is It User Friendly?	popular_comp	22,95	1991-06-30 00:00:00,000
9	Secrets of Silicon Valley	popular_comp	20,00	1994-06-12 00:00:00,000
10	Net Etiquette	popular_comp	NULL	2011-06-16 00:02:05,797
11	Computer Phobic AND Non-Phobic Individuals: Beh...	psychology	21,59	1991-10-21 00:00:00,000
12	Is Anger the Enemy?	psychology	10,95	1991-06-15 00:00:00,000
13	Life Without Fear	psychology	7,00	1991-10-05 00:00:00,000
14	Prolonged Data Deprivation: Four Case Studies	psychology	19,99	1991-06-12 00:00:00,000
15	Emotional Security: A New Algorithm	psychology	7,99	1991-06-12 00:00:00,000
16	Onions, Leeks, and Garlic: Cooking Secrets of the M...	trad_cook	20,95	1991-10-21 00:00:00,000
17	Fifty Years in Buckingham Palace Kitchens	trad_cook	11,95	1991-06-12 00:00:00,000

간단한 뷰의 생성 샘플 이다.

아래의 쿼리를 이용해서 뷰의 생성 구문을 보실 수 있다.

```
sp_helptext titles_view
```

위의 쿼리를 실행해 보면 뷰 생성 구문이 보일 것이다.

보안상의 문제가 있을 뷰를 생성 하실 경우는 **뷰에 대한 암호화 옵션을 주셔서 뷰 생성 구문을 숨기실 수 있다.**

```
USE pubs

GO

--속성으로 price * royalty * ytd_sales 값을 amt_due

--로 줄 수 있다.

CREATE VIEW accounts (title, advance, amt_due)

WITH ENCRYPTION

AS

SELECT title, advance, price * royalty * ytd_sales

FROM titles

WHERE price > 5

GO

--뷰를 생성후 암호화된 뷰의 생성구문을 보려 하면?

SET language '한국어'

go

sp_helptext accounts

개체 'accounts'의 텍스트가 암호화되었다.
```

개체 'accounts'의 텍스트가 암호화되었다. 라는 메세지와 함께 뷰 생성 구문을 보실 수 없으실 것이다.

단 이때는 반드시 생성하신 분이 뷰 생성 스크립트를 백업을 안전한 곳에 해 두어야 한다.

다음은 Check 옵션 이다.

```
USE pubs

GO

CREATE VIEW CAonly

AS

SELECT au_lname, au_fname, city, state

FROM authors

WHERE state = 'CA'

GO
```

이런 경우를 생각해 본다.

위의 뷰는 뷰의 정의대로 state 가 'CA'인 사람의 정보만 나타날 듯 하다.

만약 state 가 'CA'인 사람의 데이터를 수정해서 'OR'이라는 곳에 거주하게 한다면?

```
SELECT * FROM CAonly

--잠시후 말할 뷰의 수정이다.

--뷰에대한 수정은 분명 가능하다.

--여기선 이 WHERE state = 'CA' 로

--'CA'에서 사는 사람만 보는 것인데..

--이에 대해서 수정을 가해 OR 로 바꾸는 것이다.

--질의가 될지 안될지 먼저 생각해 보자.

UPDATE CAonly SET state = 'OR' WHERE au_lname = 'White'
```

이렇게 하면 실제로 수정이 되는 것을 볼 수 있다. 그럼 어떻게 이런 수정 작업이 불가하게 해야 할까?

이럴 때 사용하는 것이 바로 WITH CHECK OPTION 이다.

```
USE pubs
```

```
--인포메이션 정보 뷰에서 해당 뷰 이름이 존재하면?

--지워라 라는 구문 이다.

IF EXISTS (SELECT TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS

WHERE TABLE_NAME = 'CAonly')

DROP VIEW CAonly

GO

--뷰생성

CREATE VIEW CAonly

AS

SELECT au_lname, au_fname, city, state

FROM authors

WHERE state = 'CA'

--체크 옵션을 사용한다.

WITH CHECK OPTION

GO

--CA 를 OR 로 변화 시키려 하면?

UPDATE CAonly SET state = 'OR' WHERE au_lname = 'Carson'
```

Msg 550, Level 16, State 1, Line 2

대상 뷰가 WITH CHECK OPTION 을 지정하거나 WITH CHECK OPTION 을 지정하는 뷰에 걸쳐 있고 해당 연산의 하나 이상의 결과 행이 CHECK OPTION 제약 조건을 충족하지 않았으므로 삽입 또는 업데이트 시도가 실패했다.

문이 종료되었다.

WITH CHECK OPTION 이 걸려 있어서 수정이 불가하다는 메세지를 볼 것이다.

```
sp_help CAonly
```

이렇게 뷰에 대한 정보를 보실 수 있다.

다음은 뷰 수정에 대해서 보시지요.

```
CREATE VIEW All_authors (au_fname, au_lname, address, city, zip)

AS

SELECT au_fname, au_lname, address, city, zip

FROM pubs..authors

GO

--Public 역할(일종의 사용자 그룹이라고 생각 하면 된다.)에

--뷰를 SELECT 할 수 있는 권한 부여

--GRANT 는 SELECT, UPDATE 등의 권한을 부여하는 명령 이다.

--추후에 배우게 되실 것이다. ^_^GRANT SELECT ON All_authors TO public

GO

--이때 DROP 하고 뷰를 재생성 하면?
```

아직 배우진 않았지만, GRANT 권한할당 명령어가 있다.

만약 특정 계정에 대해서 GRANT 로 조회권한(SELECT)를 허용한 뒤에 뷰를 DROP 후 재 생성하면, 특정 계정에 부여했던 조회권한(SELECT)가 사라지게 된다.즉 다시 권한을 부여하는 문제가 생긴다.

따라서 이런 문제를 방지하기 위해 ALTER VIEW 명령어로 VIEW 를 수정하는 것이 좋다.

```
ALTER VIEW All_authors (au_fname, au_lname, address, city, zip)

AS
```

```
SELECT au_fname, au_lname, address, city, zip

FROM pubs..authors

WHERE state = 'UT'

GO
```

마지막으로 뷰 삭제에 대해서 알아보도록 하겠다.

```
DROP VIEW 뷰명
```

[SQL2011 강좌] 18. 저장 프로시저에 대한 이해, 생성, 수정, 삭제

이번에 소개해 드릴 내용은 저장 프로시져의 사용이다.

저장 프로시져의 장점

1. 매우 좋은 성능

2. 보안성을 높일 수 있음.

3. 다양한 처리가 가능

4. 넷웍의 부하를 줄일 수 있음.

저장 프로시져의 단점

1. 접하기가 어려움

2. VB 나 ASP 에서 호출 방식이 틀려 처음 저장 프로시져를 접할 경우 거부감이 생김

3. 새로운 언어를 배우는 정도의 부담이 있음.

저장프로시져는 보통 프로시져, 저장 프로시져, SP 라고도 불립니다. 영어로는 Stored Procedure 라고 한다. **뷰는 특정 SELECT 구문등을 단순화가 가능하다. - 하지만 뷰는 성능에 영향을 주지는 못한다. 프로시져는 성능에 영향을 많이 준다.**

```
USE pubs

GO

IF OBJECT_ID('up_konan1','P') IS NOT NULL

        DROP PROC up_konan1

GO

--프로시저 생성 구문
```

```
CREATE PROC dbo.up_konan1

AS

SELECT * FROM titles

GO

--프로시저 수행 구문(프로시저 실행 시에는 반드시 스키마 명을 붙여준다)

EXEC dbo.up_konan1
```

	title_id	title	type	pub_id	price	advance	royalty	ytd_sales	notes
1	BU1032	The Busy Executive's Database Guide	business	1389	19,99	5000,00	10	4095	An overview
2	BU1111	Cooking with Computers: Surreptitious Balance Sheets	business	1389	11,95	5000,00	10	3876	Helpful hints
3	BU2075	You Can Combat Computer Stress!	business	0736	2,99	10125,00	24	18722	The latest m
4	BU7832	Straight Talk About Computers	business	1389	19,99	5000,00	10	4095	Annotated a
5	MC2222	Silicon Valley Gastronomic Treats	mod_cook	0877	19,99	0,00	12	2032	Favorite rec
6	MC3021	The Gourmet Microwave	mod_cook	0877	2,99	15000,00	24	22246	Traditional F
7	MC3026	The Psychology of Computer Cooking	UNDECIDED	0877	NULL	NULL	NULL	NULL	NULL
8	PC1035	But Is It User Friendly?	popular_comp	1389	22,95	7000,00	16	8780	A survey of
9	PC8888	Secrets of Silicon Valley	popular_comp	1389	20,00	8000,00	10	4095	Muckraking
10	PC9999	Net Etiquette	popular_comp	1389	NULL	NULL	NULL	NULL	A must-read
11	PS1372	Computer Phobic AND Non-Phobic Individuals: Beh...	psychology	0877	21,59	7000,00	10	375	A must for th
12	PS2091	Is Anger the Enemy?	psychology	0736	10,95	2275,00	12	2045	Carefully res
13	PS2106	Life Without Fear	psychology	0736	7,00	6000,00	10	111	New exercis
14	PS3333	Prolonged Data Deprivation: Four Case Studies	psychology	0736	19,99	2000,00	10	4072	What happe

AS 라는 프로시져 생성 구문을 보니 뷰랑 비슷하다. PROC 라는 말만 틀리지 거의 같고 아울러 눈에 많이 익은 SELECT ~~ 로 시작 된다.

그런데 일반 SQL 구문인 SELECT 등등의 구문으로 이루어지는데 어떻게 더 빠를 수 있을까?

일반 SQL 구문을 수행할 경우

- 처음 수행 시

1. 구문 분석단계 - SQL 구문의 문법을 검사한다.

2. 표준화 - DB 의 개체들(테이블 등)에 대해서 검사한다.

3. 보안 점검 - 해당하는 DB 개체들에 대한 사용자의 사용 권한을 검사한다.

4. 최적화 - 최적의 성능을 내기 위한 사항을 적용한다.

5. 컴파일

- 반복 수행 시

1. 처음 수행한 일반 SQL 구문의 실행 계획이 캐싱되어 있는지 확인 후 이를 수행한다

2. 캐싱되어 있지 않을 경우 처음수행시의 5 단계를 다시 수행함.

저장 프로시져를 수행할 경우

- 생성시

1. 구문 분석 단계 - 프로시져 생성 구문의 SQL 구문들을 검사한다.

2. 표준화 - DB 의 개체들(테이블 등)에 대해서 검사한다.

3. 보안 점검 - 프로시져에 대해서 생성이 가능한지 검사한다.

4. 해당 프로시져의 구문과 생성 정보를 저장한다.

- 첫 실행 시

1. 보안 점검 - 해당하는 DB 개체들에 대한 사용자의 사용 권한을 검사한다.

2. 최적화 - 최적의 성능을 내기 위한 사항을 적용한다.

3. 컴파일 후 수행 계획을 생성 후 캐시에 저장 후 실행한다.

- 반복 수행 시

1. 실행 계획이 캐싱이 된지 확인 후 실행 한다.

2. 캐시에 실행 계획이 없을 경우 - 처음 실행 과정을 반복 한다.

SQL 수행시의 과정은 오로지 캐싱에만 의존해야 한다. 이는 실행 계획의 재사용으로 인해 파라미터 스니핑이라는 문제가 발생할 수 있다. 그리고 SQL Server 가 CBO(Cost Based Optimazation) 기반 설계라 이런 문제가 발생한다. 파라미터 스니핑은 조금 만 검색하셔도 많은 자료를 얻으실 수 있을 것이므로 여기서는 자세히 설명해드리지는 않겠다.

이제 저장 프로시저를 생성하는 방법에 대해서 알아보도록 하겠다.

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
    [ { @parameter data_type }
      [ VARYING ] [ = default ] [ OUTPUT ]
    ] [ ,...n ]
```

```
[ WITH
    { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]

[ FOR REPLICATION ]

AS sql_statement [ ...n ]
```

CREATE PROCEDURE(Transact-SQL): http://msdn.microsoft.com/en-us/library/ms187926(v=SQL.110).aspx

저장 프로시져는 위와 같은 어려워 보이는 방식으로 생성한다.

하지만 몇 가지 방식 사용 안한다. 관심있으신 분들은 온라인 도움말(북스 온라인)을 참고 할 수 있다..

<일반적인 프로시져의 생성>

```
USE pubs

GO

--프로시져 생성구문

CREATE PROC dbo.up_konan2

AS

SELECT pub_id 저자

, type 책종류, royalty 로열티 , ytd_sales 팔린수, AVG(price) 평균값

FROM dbo.titles

GROUP BY pub_id, type, royalty, ytd_sales

WITH CUBE

GO

--프로시져 수행 구문
```

```
EXEC dbo.up_konan2
```

	저자	책종류	로열티	팔린수	평균값
1	1389	popular_comp	NULL	NULL	NULL
2	NULL	popular_comp	NULL	NULL	NULL
3	0877	UNDECIDED	NULL	NULL	NULL
4	NULL	UNDECIDED	NULL	NULL	NULL
5	NULL	NULL	NULL	NULL	NULL
6	NULL	NULL	NULL	NULL	NULL
7	0736	psychology	10	111	7,00
8	NULL	psychology	10	111	7,00

<프로시져 수정>

대부분의 수정 구문처럼 ALTER 구문을 사용 한다. 간단히 약간 수정을 가해 보면

```
--프로시저 수정 구문

ALTER PROC dbo.up_konan2

AS

SELECT pub_id

FROM dbo.titles

GO

--프로시져 수행 구만

EXEC dbo.up_konan2
```

	pub_id
1	1389
2	1389
3	0736
4	1389
5	0877
6	0877
7	0877
8	1389

DROP 을 할 경우 여러 묶여 있는(sysobjects 테이블의 id 로 묶여 있는) 여러 개체들과 비호환의 문제, 또는 유저에 대한 권한 문제가 있을 수 있으므로 가능한 저렇게 ALTER 구문을 쓰시면 좋다.

<프로시져 삭제>

```
--프로시저 삭제

DROP Proc dbo.up_konan2
```

<프로시져 생성 구문 조회>

저장 프로시져를 생성 후 생성 구문을 추후에 조회할 수 있다.

```
if OBJECT_ID('up_konan3','P') IS NOT NULL

        DROP PROC up_konan3

GO

--프로시져 생성 구문

CREATE PROC dbo.up_konan3

AS

SELECT pub_id, title

FROM dbo.titles

GO

--프로시저 수행구문

EXEC dbo.up_konan3

go

--생성된 프로시저의 내용 조회
```

```
sp_helptext up_konan3

GO
```

컨트롤 + T 키를 누르신 후 결과를 Text 모드로 보면 더 잘 보이다.

<저장 프로시져의 종류>

저장 프로시져의 종류는 3 가지 라고 보면 된다.

1. 사용자 정의 저장 프로시져 – 지금까지 만들었던 형태

2. 시스템 저장 프로시져 - sp_help, sp_helptext 와 같이 sp_로 시작하는 프로시져.

3. 확장 시스템 저장 프로시져 - xp_ 로 시작하는 프로시져를 의미 한다.

예를 들면..

```
EXEC master..xp_cmdshell 'dir'
```

와 같은 시스템 저장 프로시져를 의미 한다.

이 정도면 어느 정도 생성 방법은 설명을 드린 듯 하다.

또한 프로시저를 생성 시에는 반드시 sp_로 만들지 마시길 바란다. SQL Server 내부적으로 프로시저 수행 시 sp 로 시작하는 단어면 우선 시스템 프로시저인지 검사를 하게 되는데.. 이는 불필요한 작업이 될 수 있다. 대부분의 많은 사람들은 user sp 라고 해서.. usp_ 라고 많이들 사용한다.

[SQL2011 강좌] 19. 인덱스에 대한 이해

오늘은 드디어 SQL Server 에서 가장 중요한 인덱스에 대해서 알아볼 차례 이다. 이번에 소개해드릴 내용은 색인 이다. 색인이라는 주제는 사실 매우 광범위하고 성능에 지대한 영향을 줄 수 있는 주제 이다. 아울러 SQL 서버의 상당한 내부적인 구조를 알아야 하므로 대단히 많은 분들이 꺼려하고 어려워하는 부분이기도 하다. 실제로 사이트에 나가 튜닝 작업을 할 경우 먼저 보는 부분이기도 하다.

<색인은 무엇인가?>

색인은 트리구조(계층형)로 이루어져 있는 실제 페이지의 데이터를 포인팅하는 데이터 이다.

색인에 대해서 오해하기 쉬운 몇 가지를 먼저 정리하면 다음과 같다.
1. 색인은 공간을 차지하지 않는다. - 아뇨. 색인의 종류에 따라 가변적으로 인덱스의 크기가 잡힙니다. 물론 실제 데이터에 비례적인 크기가 된다. - 종류별로 정확한 크기 예측은 힘들죠. 중요한 건 색인 역시 데이터이다.

2. 색인을 만들면 항상 빠르다. – 항상 그렇지는 않다. 잘 만들어야 빠르다.

3. 데이터가 적어도 인덱스가 유용하다. - 아닙니다. 데이터가 많을 수록 인덱스는 환상의 속도를 비례적으로 낼 수 있다. - 권고적으로 테이블의 크기가 대략 1M 정도보다 작으면 역시나 인덱스가 없는 게 유용할 수 있다.

우선 다음과 같은 회원 테이블 정보가 있다고 할 경우에

번호	이름	생긴정도
1	김대우	죽임
2	한기환	살림
3	이승용	죽임
4	박훈	살림
5	강해원	죽임
6	이문규	살림
7	이수선	죽임
8	차영인	살림
9	곽광철	죽임
10	유병수	살림
11	이재형	죽임
12	김태영	살림
13	은정범	죽임

이 회원 테이블은 보시는 바와 같이 번호, 이름, 생긴 정도를 포함하고 있다.
번호와 이름은 믿을만한 정보이나 모든 정보가 믿을만한 건 아니다. – 주의

여하간 이런 부분이 실제 데이터페이지에는 어떻게 기록이 되어 있을까?

데이터 페이지 1		
1	김대우	죽임
2	한기환	살림
3	이승용	죽임

데이터 페이지 2		
4	박훈	살림
5	강해원	죽임
6	이문규	살림

데이터 페이지 3		
7	이수선	죽임
8	차영인	살림
9	곽광철	죽임

데이터 페이지 4		
10	유병수	살림
11	이재형	죽임
12	김태영	살림

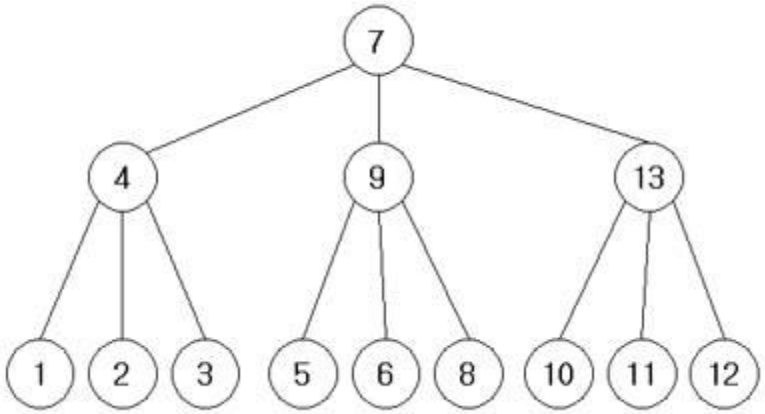
데이터 페이지 5		
13	은정범	죽임

내부 데이터 페이지에는 위와 같은 방식으로 저장되어 있을 것이다. 다음처럼 번호 컬럼의 값을 찾고자 한다고 생각해 보면

번호	이름	생긴정도
1	김대우	죽임
2	한기환	살림
3	이승용	죽임
4	박훈	살림
5	강해원	죽임
6	이문규	살림
7	이수선	죽임
8	차영인	살림
9	곽광철	죽임
10	유병수	살림
11	이재형	죽임
12	김태영	살림
13	은정범	죽임

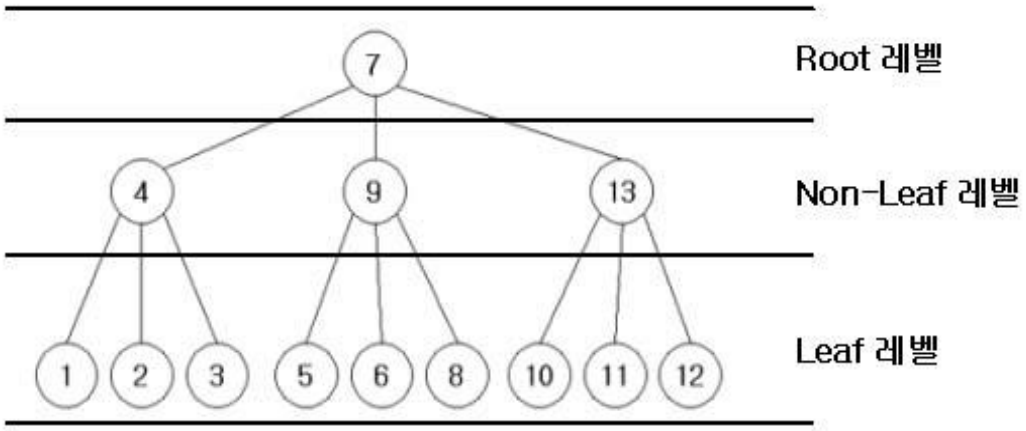
1 번 번호를 찾으려 한다면? 1 번에 찾을 수 있고 8K Byte 의 IO 로 찾을 수 있었다. 만약 **7 번 번호를 찾으려 한다면?** 3 번째 페이지에 있기 때문에 3 * 8K Byte 크기의 IO 로 찾았다. **만약 13 번을 찾으려**

한다면? 4 페이지에 있으니 32K Byte 가 소요 된다. 데이터가 늘어나 만약 10 만번째를 찾으려 한다면? 대단한 IO 가 발생하게 된다.



인덱스를 이해하기 위해서는 위의 그림을 보면서 밸런스드 트리를 이해해야 한다. 밸런스드 트리의 근간은 말 그대로 밸런스 이다. 균형이 중요하게 된다.

이 구조에서 **1 번을 찾으려 하면?** 맨 위의 7 을 보니 1 <> 7 그러므로 아래를 보니 4, 9 ,13 이 있다. 그 후. 1 < 4 그러니 4 로 가면 되고 4 로 가서 딱 보니 1 에 접근 할 수 있다. 각각의 단계를 페이지와 비교해 보면. 7 에서 1 개 페이지. 4 에서 1 개페이지를 보니 우리의 값인 1 이 보이고 3 개의 페이지를 읽게 된다. **다음 7 번을 찾으려 하면?** 맨 위에서 찾게 된다. **다음 13 을 보면** 역시 2 번에 찾을 수 있게 된다. 자 저 트리의 깊이가 한 단계 두 단계 늘어난다고 생각해 보면 지금은 비록 평균적으로 3 개 정도면 읽을 수 있겠지만. 노드가 늘어갈수록 오래 걸릴 수 밖에 없는 구조이다.



계속적으로 이야기가 나오겠지만. **위처럼 최상위위치를 Root 레벨이라고 부른다.** 중간 단계의 레벨, 위에서는 중간 레벨이 하나뿐이지만 B-Tree 의 데이터가 많아 질수록 중간 레벨은 계속 늘어날 수 있겠죠? 이것들을 **Non-Leaf 레벨이라고 부른다. 또는 중간레벨(Inertmediate Level) 라고 부르기도 한다.**

레벨대신 가끔 노드라고도 부른다. 끝으로 **최하위의 레벨을 부를 때 Leaf 레벨 또는 Leaf 노드라고도 한다.** 한글로는 각각 뿌리수준 / 잎이 아닌 수준, 또는 가지단계, 또는 중간단계 / 잎수준 이라고 부르기도 한다.

자 이제 기본적인 인덱스의 형태에 대한 이야기는 간단히 드렸다. 다음 장에서 실제 인덱스들을 가지고 이야기를 풀어 보도록 한다.

[SQL2011 강좌] 20. 클러스터드 인덱스와 넌 클러스터드 인덱스

이번에는 클러스터드 인덱스와 넌 클러스터드 인덱스에 대해서 소개하겠다.

색인의 종류 SQL 서버는 두가지 종류의 색인이 있다.

그 **두 가지는 Clustered 인덱스와 Non-Clustered 인덱스 두 가지 이다.**

각각 한글로는 클러스터된 인덱스 / 클러스터되지 않은 인덱스로 불린다.

인덱스는 분명 만들어야 하는 것이다. 이 말은

1. 만드는 비용이 있다.

다음 인덱스는 공간이 필요하다고 했다.

2. 디스크 비용이 필요하며 항상 공간 유지를 적절해야 한다.

관리가 필요하다.

3. 유지보수에 비용이 필요하다.

인덱스를 만들면 데이터를 조회하는 속도를 높일 순 있지만

만약 데이터가 삽입 / 수정 / 삭제가 이루어지면 앞에서 본 인덱스의 구조를 변경하면서 재생성 해야한다.

4. 데이터 변경이 있을 때 비용이 추가 된다.

앞에서 또한 말씀 드렸지만 인덱스를 사용하면 항상 빨라진다고 말씀 안드렸죠.

5. 인덱스는 항상 빠르지 않다.

인덱스를 사용하면 유용한 부분은?

5-1. WHERE 절에서 참조되는 컬럼

5-2. 참조키가 설정되어 있는 컬럼

5-3. 참조키는 아니지만 JOIN 에 사용되는 컬럼

5-4. 범위 검색이 일어나는 컬럼

5-5. ORDER BY 로 정렬 되는 컬럼, GROUP BY 로 그룹핑 되는 컬럼

5-6. TOP 구문과 함께 사용되는 SELECT 절

등에서 사용하면 좋다.

인덱스의 부적절한 사용은 다음과 같다.

5-7. WHERE 절에서 사용되지 않는 컬럼에는 물론 효과 없음.

5-9. WHERE 절에서 변환(함수등이 사용되는)되는 컬럼과 비교 시 효과 없음.

5-10. 선택도(찾을 데이터 / 전체 데이터)가 클 경우 효과 적음.

잠시 후 말씀 드리겠지만.. 예를 들어 성별 컬럼과 같은 남 / 여 비율적으로 대략 50 : 50 의 구성비가 있는 컬럼이라면? 인덱스의 효과가 떨어 진다.

이런 주의 사항이 필요하다.

자 이제 인덱스 생성 구문을 실제로 봐 보도록 한다.

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name

    ON <object> (column [ ASC | DESC ] [ ,...n ] )

    [ INCLUDE (column_name [ ,...n ] ) ]

    [ WHERE <filter_predicate> ]

    [ WITH ( <relational_index_option> [ ,...n ] ) ]

    [ ON { partition_scheme_name (column_name)

        | filegroup_name

        | default

        }

   ]

   [ FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name |
"NULL" } ]

[ ; ]

<object> ::=

{

   [ database_name. [ schema_name ] . | schema_name. ]

   table_or_view_name

}

<relational_index_option> ::=

{
```

```
  PAD_INDEX = { ON | OFF }
 | FILLFACTOR =fillfactor
 | SORT_IN_TEMPDB = { ON | OFF }
 | IGNORE_DUP_KEY = { ON | OFF }
 | STATISTICS_NORECOMPUTE = { ON | OFF }
 | DROP_EXISTING = { ON | OFF }
 | ONLINE = { ON | OFF }
 | ALLOW_ROW_LOCKS = { ON | OFF }
 | ALLOW_PAGE_LOCKS = { ON | OFF }
 | MAXDOP =max_degree_of_parallelism
 | DATA_COMPRESSION = { NONE | ROW | PAGE}
   [ ON PARTITIONS ( { <partition_number_expression> | <range> }
   [ , ...n ] ) ]
}
```

몇 가지 중요한 정보만 말씀드리도록 하겠다.

UNIQUE: 지정해 고유 인덱스 또는 고유하지 않은 인덱스 생성이 가능하다.

색인의 종류는 두가지로 **CLUSTERED | NONCLUSTERED** 로 할지 지정이 가능하다.

table 이나 view 에 생성이 가능하다.

컬럼을 ASC 또는 DESC 로 정렬해 생성 가능하다. 특히 클러스터드 인덱스를 생성시 유용하며 ORDER BY 구문과도 밀접하다.

다음 인덱스 옵션에서

PAD_INDEX 는 중간 레벨을 적절히 비워 데이터 삽입 등에 대비하기 위한 것이며

FILLFACTOR 는 리프 레벨을 적절히 비워 역시 삽입 등에 대비하는 것이다.

IGNORE_DUP_KEY 는 중복되는 값을 무시한다는 의미이다.

DROP_EXISTING 은 이미 존재하는 인덱스가 있으면 제거하고 재 생성하라는 의미 이다.

SORT_IN_TEMPDB TEMPDB 상에서 정렬하라는 옵션 이다. 예를들어 데이터와 인덱스가 같은 물리적인 디스크에 있고 데이터가 한 1000 만건 정도 된다면 인덱스 생성에 대단히 많은 시간이 소요 된다. 이때 TEMPDB 에서 인덱스 생성시 필요한 정렬작업을 시키고 사용자 데이터베이스의 물리적인 디스크와 TEMPDB 물리적인 디스크가 틀리다면 인덱스 생성시 부하를 줄일 순 있지만 TEMPDB 에 다른 불필요 공간이 생기니 주의 해야 한다.

끝으로 **ON FILEGROUP** 은 인덱스 역시 데이터라고 말씀 드렸다. 데이터베이스 강좌에서 filegroup 을 적절히 분산시켜 생성해 속도를 높일 수 있다고 말씀 드린 것처럼 인덱스 역시 적절한 filegroup 에 위치시켜 최적의 속도를 낼 수 있게 할 수 있다.

```
CREATE DATABASE konanTestDB

GO

USE konanTestDB

GO

--테이블생성

CREATE TABLE konan_test_table(

konan_id int IDENTITY (1, 1) NOT NULL

, konan_data char (50) NOT NULL

, konan_date char (50) NOT NULL

)

GO

--10000 건의샘플데이터삽입

set nocount on

GO

declare @i int

set @i = 0

while @i < 10000

begin

--WAITFOR DELAY '00:00:01'

insert into konan_test_table values

(
```

```
@i ,

convert(varchar, datepart(yy, getdate())) + '년'

+ convert(varchar, datepart(mm, getdate())) + '월'

+ convert(varchar, datepart(dd, getdate())) + '일'

+ convert(varchar, datepart(hh, getdate())) + '시'

+ convert(varchar, datepart(mi, getdate())) + '분'

+ convert(varchar, datepart(ss, getdate())) + '초'

+ convert(varchar, datepart(ms, getdate())) + '미리초'

)

set @i = @i + 1

end

GO

--10 초.

set nocount off

GO

--샘플데이터조회

SELECT TOP 100 * FROM konan_test_table

SELECT COUNT(konan_id) FROM konan_test_table
```

	konan_id	konan_data	konan_date
1	1	0	2011년 6월 16일 21시 36분 48초 160마리초
2	2	1	2011년 6월 16일 21시 36분 48초 223마리초
3	3	2	2011년 6월 16일 21시 36분 48초 223마리초
4	4	3	2011년 6월 16일 21시 36분 48초 243마리초
5	5	4	2011년 6월 16일 21시 36분 48초 243마리초
6	6	5	2011년 6월 16일 21시 36분 48초 243마리초
7	7	6	2011년 6월 16일 21시 36분 48초 243마리초
8	8	7	2011년 6월 16일 21시 36분 48초 247마리초
9	9	8	2011년 6월 16일 21시 36분 48초 247마리초
10	10	9	2011년 6월 16일 21시 36분 48초 247마리초
11	11	10	2011년 6월 16일 21시 36분 48초 247마리초

	(No column name)
1	10000

대략적인 샘플데이터 100 건과 10000 건이 잘 들어간게 보인다. 여기서 어느 정도의 IO 비용이 소요되는지 확인해 보면 아래와 같다.

```
--IO 통계보기

SET STATISTICS IO ON

SELECT * FROM konan_test_table WHERE konan_id = 5000

--IO 통계?

SELECT * FROM konan_test_table WHERE konan_id < 10

--IO 통계?

SELECT * FROM konan_test_table WHERE konan_id < 45

--IO 통계?

SELECT * FROM konan_test_table WHERE konan_id < 1000

--IO 통계?
```

```
--IO 통계끄기

SET STATISTICS IO OFF
```

Results | Messages

	konan_id	konan_data	konan_date
1	5000	4999	2011년 6월 16일 21시 36분 50초 213미리초

	konan_id	konan_data	konan_date
1	1	0	2011년 6월 16일 21시 36분 48초 160미리초
2	2	1	2011년 6월 16일 21시 36분 48초 223미리초
3	3	2	2011년 6월 16일 21시 36분 48초 223미리초
4	4	3	2011년 6월 16일 21시 36분 48초 243미리초
5	5	4	2011년 6월 16일 21시 36분 48초 243미리초
6	6	5	2011년 6월 16일 21시 36분 48초 243미리초
7	7	6	2011년 6월 16일 21시 36분 48초 243미리초
8	8	7	2011년 6월 16일 21시 36분 48초 247미리초

	konan_id	konan_data	konan_date
1	1	0	2011년 6월 16일 21시 36분 48초 160미리초
2	2	1	2011년 6월 16일 21시 36분 48초 223미리초
3	3	2	2011년 6월 16일 21시 36분 48초 223미리초
4	4	3	2011년 6월 16일 21시 36분 48초 243미리초
5	5	4	2011년 6월 16일 21시 36분 48초 243미리초
6	6	5	2011년 6월 16일 21시 36분 48초 243미리초

Results | Messages

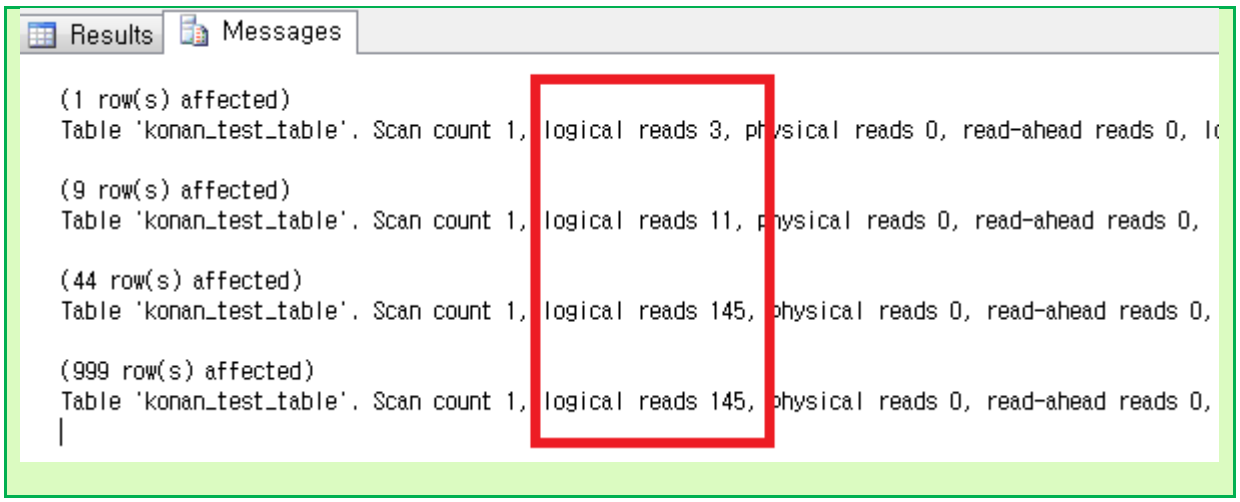
```
(1 row(s) affected)
Table 'konan_test_table'. Scan count 1, logical reads 145, physical reads 0, read-ahead reads 0,

(9 row(s) affected)
Table 'konan_test_table'. Scan count 1, logical reads 145, physical reads 0, read-ahead reads 0,

(29 row(s) affected)
Table 'konan_test_table'. Scan count 1, logical reads 145, physical reads 0, read-ahead reads 0,

(999 row(s) affected)
Table 'konan_test_table'. Scan count 1, logical reads 145, physical reads 0, read-ahead reads 0,
```

메시지 부분을 보면 IO 통계를 확인할 수 있을 것이다. 이제 인덱스를 생성해 보도록 한다.

```
--간단한인덱스생성

CREATE INDEX idx ON konan_test_table (konan_id)

GO
```

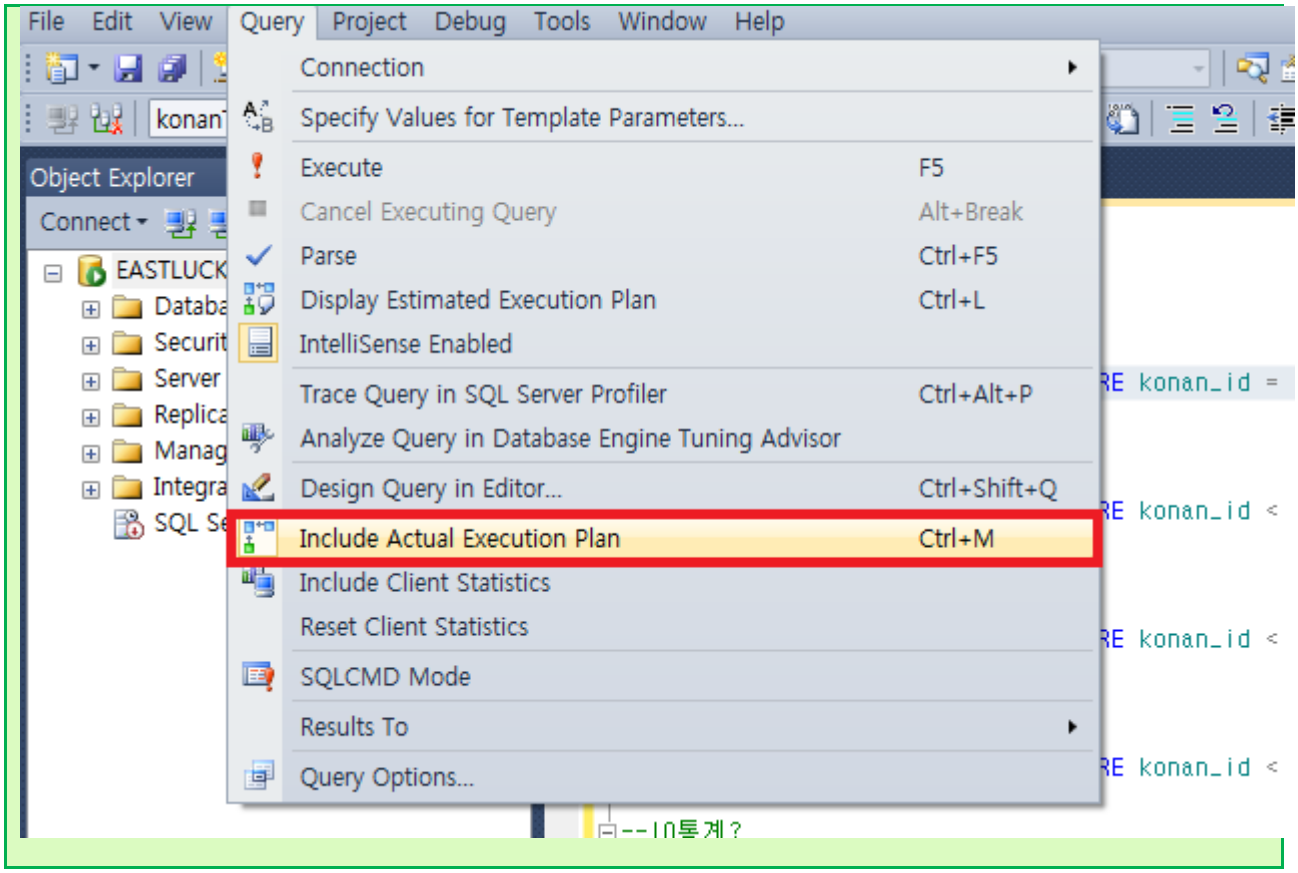
인덱스 생성은 잘 되셨을 거구요. 다시 데이터를 조회해 보면

```
--IO 통계보기

SET STATISTICS IO ON

SELECT * FROM konan_test_table WHERE konan_id = 5000

--IO 통계?

SELECT * FROM konan_test_table WHERE konan_id < 10

--IO 통계?

SELECT * FROM konan_test_table WHERE konan_id < 45

--IO 통계?

SELECT * FROM konan_test_table WHERE konan_id < 1000

--IO 통계?

--IO 통계끄기

SET STATISTICS IO OFF
```

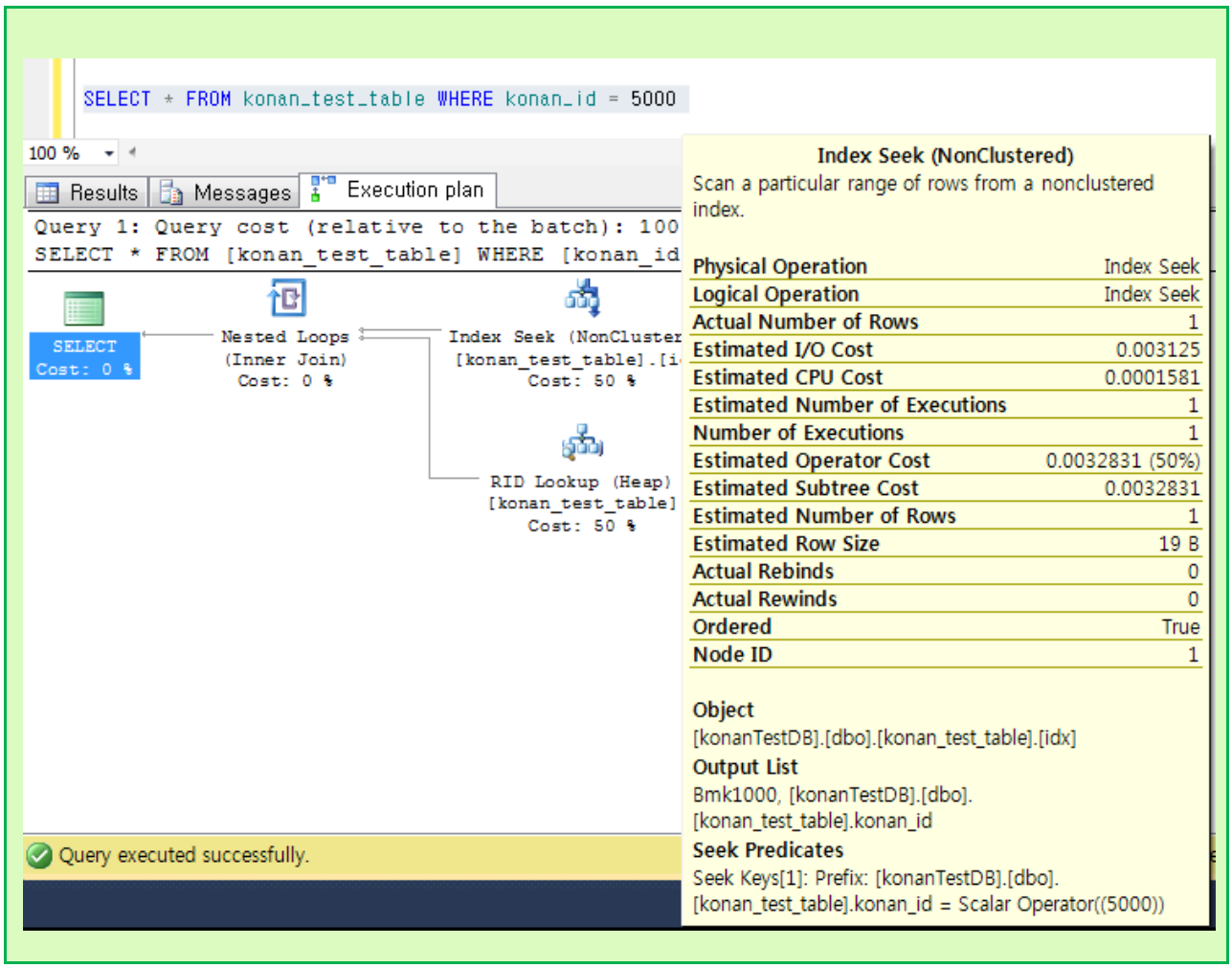


통계를 봐 보면 WHERE konan_id < 30 부터 인덱스가 없을 때와 비슷한 수치가 나올 것이다.

실행 계획을 보면 이런 결과가 나온 이유를 이해할 수 있다. 실행 계획 활성화를 위해서.. Query => Include Actual Execution Plan 을 선택한다.(단축키는 Ctrl+M) 이다.



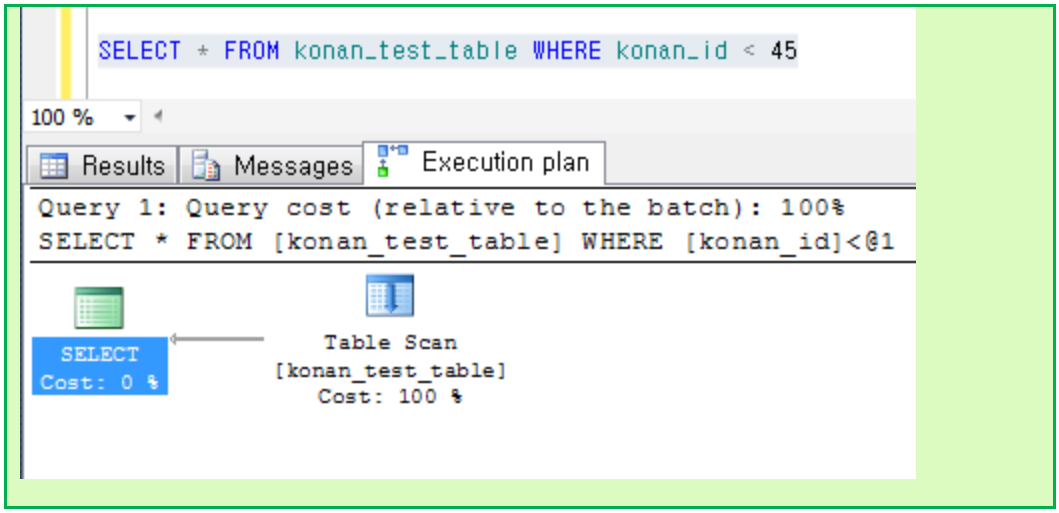
그리고 나서 쿼리를 수행해보면.. Results 와 Message 말고 Execution plan 탭이 생성된다.



위의 그림처럼 실행 계획을 보실 수 있다.

중요한 건 Index Seek 를 했다는 점이다.

다음 쿼리에서 수행 계획을 보면?



이렇게 konan_id < 45 을 봐 보면 Table Scan 을 하는 것을 알 수 있다.

분명 어떤 것은 인덱스를 타고 어떤 것은 인덱스를 타지 않는다. 앞에서 말씀 드린 선택도 라고 해서 (찾을 데이터 수 / 전체 데이터 수)로 판단 되게 된다.

아래와 같이 실행하면 인덱스의 정보를 확인할 수 있다.

```
--인덱스정보조회

sp_helpindex konan_test_table
```

	index_name	index_description	index_keys
1	idx	nonclustered located on PRIMARY	konan_id

흥미있게 보실 부분으로 인덱스의 종류를 지정하지 않으면? 뒤에서 보시겠지만 넌클러스터드 인덱스로 잡히게 된다.

인덱스의 변경 구문은 없으며 DROP INDEX 구문을 이용해 인덱스를 제거할 수 있다.

```
--인덱스제거

DROP INDEX konan_test_table.idx
```

물론 앞에서 말씀 드린 CREATE INDEX 구문의 DROP EXISTING 구문으로 존재하는 인덱스를

지우고 생성할 수 도 있다.

<클러스터드 인덱스>

클러스터드 인덱스는 간단히 인덱스의 리프레벨이 데이터 페이지와 같은 것이다.

뭔소리냐구요? 천천히 설명 드리지요.

인덱스 페이지 1	
1	1
4	2
7	3
10	4
13	5

데이터 페이지 1		
1	김대우	죽임
2	한기환	살림
3	이승용	죽임

데이터 페이지 2		
4	박훈	살림
5	강해원	죽임
6	이문규	살림

데이터 페이지 3		
7	이수선	죽임
8	차영인	살림
9	곽광철	죽임

데이터 페이지 4		
10	유병수	살림
11	이재형	죽임
12	김태영	살림

데이터 페이지 5		
13	은정범	죽임

이런 화면을 생각해 보세요.

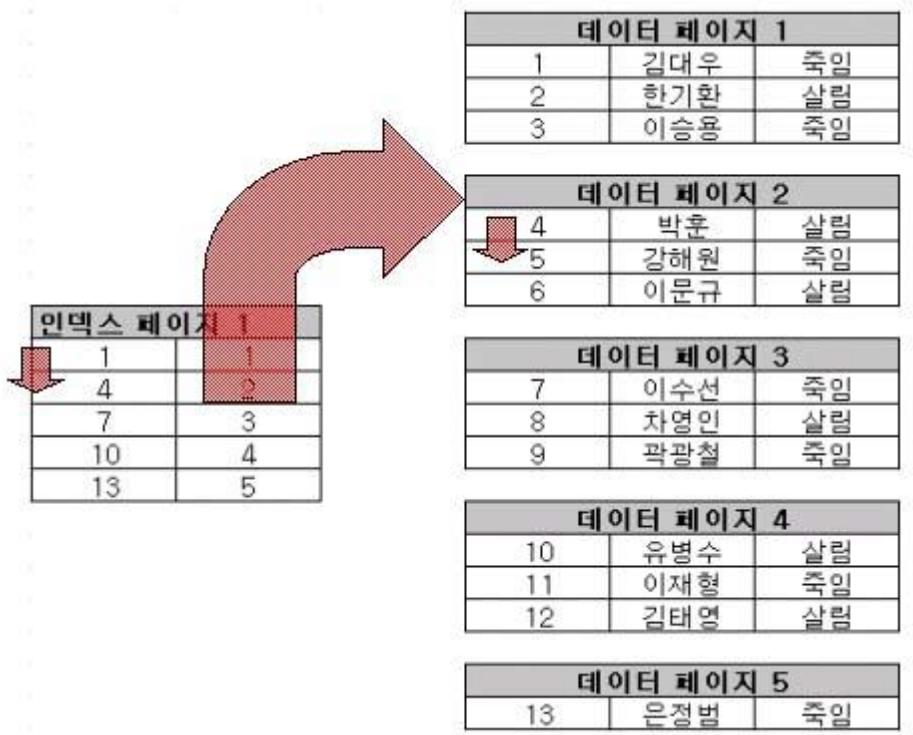
클러스터드 인덱스는 물리적으로 행을 재배열해 리프 레벨이 차곡차곡 오른쪽의 데이터 페이지처럼 쌓이게 한다. 그럼 인덱스를 봐 볼까?

인덱스가 설정된 컬럼의 값이 순차적으로 재배열 되게 되므로 왼쪽에 보시는 인덱스 페이지는 단지 키값과 페이지의 번호만이 지정되게 된다.

자 그럼 우리가 만약 5 번 강해원을 찾으려 한다고 생각해 보도록 한다.

이때는 먼저 클러스터드 인덱스의 값을 보니 4 < 5 < 7 사이가 되므로

4 - 2 인 2 번 페이지로 가게 된다. 2 번 페이지로 가서 바로 5 번 강해원을 찾게 된다.



대략 이러한 그림으로 이루어 지게 되는 것이다.

만약 12 번 김태영을 찾는다면? 이때는 역시 10 < 12 < 13 이므로 4 페이지로 가서 12 번 김태영을 찾을 수 있게 된다.

이것이 가능한 이유가 뭘까? 바로 물리적으로 행이 정렬되어 있기 때문에 가능한 것이다

만약 물리적으로 행이 재배열 되어 있지 않다면? 이런 작업은 불가해 지는 것이다.

아울러 물리적으로 행들이 재배열 되어 있으므로 범위 검색에 대단히 유용하다.

이 말은 선택도가 어느 정도 높아도 - 클러스터드 인덱스의 경우 30%정도도 가능 - 인덱스를 이용해 데이터를 조회할 수 있게 된다.

아울러 클러스터드 인덱스가 저렇게 테이블에 하나 생성되어 있다..

만약 클러스터드 인덱스를 하나 더 테이블에 만들고 싶다면 어떨까?

안타깝게도 다음 번 클러스터드 인덱스는 테이블을 다시 재구성 하고 싶겠지만..

이미 한번 테이블이 정렬된 상태로 재구성 되어 있으므로 불가해 집니다.

오로지 테이블에 단 1 개의 클러스터드 인덱스만 생성이 가능하니 주의 해야 한다.

그럼 앞의 샘플쿼리를 클러스터드 인덱스로 생성하고 장난을 조금 쳐 볼까?

```
--인덱스가존재하면지울것

DROP INDEX konan_test_table.idx

--클러스터드인덱스생성

CREATE CLUSTERED INDEX idx ON konan_test_table (konan_id)

GO
```

이렇게 클러스터드 인덱스를 생성할 수 있다. 그런데 특이하게도 SQL 서버는 클러스터드 인덱스에 대해서 대단히 높은 우선권을 부여 한다.

예를 들어 이럴 땐.. 풀스캔을 해도 좋을것 같은데... 클러스터드 인덱스를 써서 검사할 경우가 있다. 이는 말씀 드린대로 SQL 쿼리 최적화기가 판단하는데 테이블의 크기가 작거나(로우의 건수가 아닌 전체적인 크기) 또는 클러스터드 인덱스의 키값 컬럼이 작을 경우 종종 발생 한다.

실제 현업에서는 범위검색에 종종 이 클러스터드 인덱스를 두게 되므로 정상적으로 잘 동작하게 된다.

또한 클러스터드 인덱스는 비교적 넌클러스터드 인덱스보다 크기가 작다.

아울러 클러스터드 인덱스는 크기를 대략적으로 예측할 수 있는 인덱스이기도 한다.

다음 넌 클러스터드 인덱스를 봐 보도록 할까?

<넌 클러스터드 인덱스>

실제 테이블의 데이터가 항상 순차적으로 들어가 있는 것은 아닙니다.

관계형 데이터베이스에서 순차라는 것은 사실 의미가 없다. 관계형 데이터베이스의 순차 유지는 오로지 ORDER BY 에 적절히 이용되는 컬럼을 잘 구성해야만 하는 것이다.

예를 들어 실제 진짜 SQL 서버에 들어가 있는 테이블의 순서는 다음과 같을 것이다.

번호	이름	생긴정도
4	박훈	살림
7	이수선	죽임
9	곽광철	죽임
11	이재형	죽임
6	이문규	살림
5	강해원	죽임
1	김대우	죽임
8	차영인	살림
2	한기환	살림
10	유병수	살림
3	이승용	죽임
13	은정범	죽임
12	김태영	살림

이런 식의 데이터가 실제로 들어가 있게 된다. 그렇다면 페이지 내부는 어떤 식일까?

데이터 페이지 1		
4	박훈	살림
7	이수선	죽임
9	곽광철	죽임

데이터 페이지 2		
11	이재형	죽임
6	이문규	살림
5	강해원	죽임

데이터 페이지 3		
1	김대우	죽임
8	차영인	살림
2	한기환	살림

데이터 페이지 4		
10	유병수	살림
3	이승용	죽임
13	은정범	죽임

데이터 페이지 5		
12	김태영	살림

대략적으로 위의 그림과 같은 식의 데이터가 들어가 있게 된다.실질적인 데이터 페이지 이다. 이런 데이터페이지들의 번호에 클러스터드 인덱스를 만약 생성한다면 어떻게 될까? 이럴 경우는 RID 라는 녀석이 필요하게 된다. 간단히 RID 는 로우의 구별하는 특정 값이라고 생각 하시면 된다. 넌 클러스터드 인덱스는 바로 이 RID 를 가지고 데이터 페이지를 포인팅 하게 된다. RID 샘플을 보면 다음과 같은 식이다.

번호	RID
1	1-3-1
2	1-3-3
3	1-4-2
4	1-1-1
5	1-2-3
6	1-3-3
7	1-1-2
8	1-3-2
9	1-1-3
10	1-4-1
11	1-2-1
12	1-5-1
13	1-4-3

여기서 RID 의 첫 번째 1 은 화일 그룹을 의미하게 된다. 그 다음 숫자는 데이터 페이지 번호이며 마지막 세 번째 숫자는 페이지 옵셋으로 정확히 페이지의 한 로우를 포인팅(Pointing)하게 된다. 또한 이 인덱스 페이지는 다음과 같은 형식으로 분할 된다.

인덱스 페이지 1	
1	1-3-1
2	1-3-3
3	1-4-2
4	1-1-1

인덱스 페이지 2	
5	1-2-3
6	1-3-3
7	1-1-2
8	1-3-2

인덱스 페이지 3	
9	1-1-3
10	1-4-1
11	1-2-1
12	1-5-1

인덱스 페이지 4	
13	1-4-3

이러한 형식이 넌클러스터드 인덱스에서 과연 어떻게 사용 될까?

실제 구조를 그려 보도록 한다.

인덱스 페이지 7	
1	1
5	2
9	3
13	4

인덱스 페이지 1	
1	1-3-1
2	1-3-3
3	1-4-2
4	1-1-1

인덱스 페이지 2	
5	1-2-3
6	1-3-3
7	1-1-2
8	1-3-2

인덱스 페이지 3	
9	1-1-3
10	1-4-1
11	1-2-1
12	1-5-1

인덱스 페이지 4	
13	1-4-3

데이터 페이지 1		
4	박훈	살림
7	이수선	죽임
9	곽광철	죽임

데이터 페이지 2		
11	이재형	죽임
6	이문규	살림
5	강해원	죽임

데이터 페이지 3		
1	김대우	죽임
8	차영인	살림
2	한기환	살림

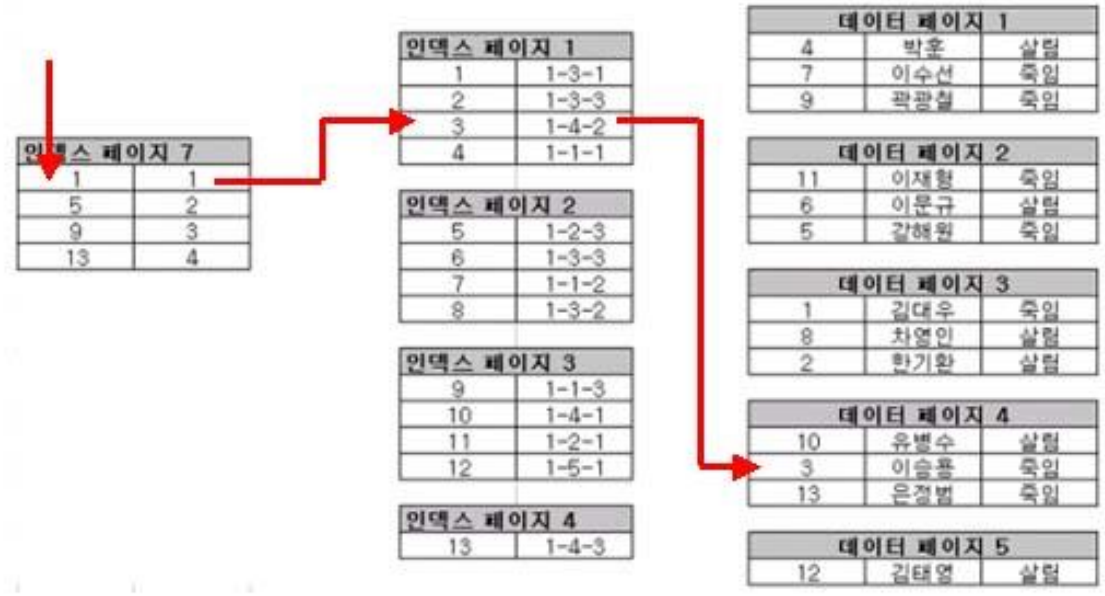
데이터 페이지 4		
10	유병수	살림
3	이승용	죽임
13	은정범	죽임

데이터 페이지 5		
12	김태영	살림

이러한 식으로 생성이 되게 된다. 루트 레벨은 인덱스 페이지 7 이며 중간 레벨은 엔덱스 페이지 1,2,3,4 이고 실제 데이터페이지는 1,2,3,4,5 데이터 페이지가 된다. 자 값을 네비게이션 해 보도록 한다.

만약 3 번 이승용을 찾으려 한다고 생각해 보지요 1 < 3 < 5 이므로 1 페이지로 가야 한다 같더니 3 은 4 페이지 2 번째 로우에 있다고 한다.

바로 4 페이지로 가서 두 번째 로우를 컨택한다.



또한 만약 8 차영인을 찾으려 한다면

5 < 8 < 9 이므로 인덱스페이지 2 로 가서 8 차영인을 보니 1-3-2 라고 되어 있다.

데이터 페이지 3 의 2 번째 로우를 포인팅 하게 된다.

이것이 넌클러스터드 인덱스 이다.

넌클러스터드 인덱스는 이렇게 포인팅 정보를 가지게 되므로 인덱스의 크기가 커지게 된다. 아울러 선택도가 높으면 바로 쿼리 최적화기는 이 넌클러스터드 인덱스를 사용하지 않게 된다. 일반적으로 3% 이내 정도면 이 넌클러스터드 인덱스를 사용하지 않게 된다.

넌클러스터드 인덱스는 데이터페이지를 물리적으로 재배열하지 않으므로 여러 개의 인덱스를 생성할 수 있다. 최대생성 가능 개수는 249 개 이다.

다음으로 클러스터드 인덱스가 있는 테이블의 넌클러스터드 인덱스를 알아 보도록 한다.

물론 당연히 클러스터드 인덱스와 넌 클러스터드 인덱스가 있을 때 넌클러스터드 인덱스를 조회할 경우 이다.

현재는 번호에 클러스터드 인덱스가 걸린 상태이며 이름에 넌클러스터드 인덱스가 설정된 상태 이다.

이런 식으로 구성이 되게 된다. 먼저 알아야 하는 것은 넌클러스터드 인덱스의 RID 는 더이상 RID 가 아닌 클러스터드 인덱스의 키값을 가지게 된다. 좀더 간단히 실제 조회를 해 보도록 한다.

넌클러스터드 인덱스가 걸린 이름 컬럼의 유병수라는 이름을 조회할 경우에 박훈 < 유병수 < 이수선 이므로 넌클러스터드 중간 레벨의 2 페이지로 가게 될것이다.

여기서 유병수를 찾으니 유병수는 클러스터드 인덱스 10 번 키값을 가지고 있다.

10 이라는 값을 가지고 클러스터드 인덱스 페이지로 가니 10 은 데이터페이지 4 에 있다고 한다.
데이터페이지 4 로 가니? 바로 10 유병수를 찾을 수 있게 된다.

바로 이런 그림이 되는 것이다. 만약 한기환을 찾으려 한다면 어떻게 될까? 한기환 < 한기환 이므로 넌클러스터드 중간레벨의 4 페이지로 가게 된다. 클러스터드 인덱스의 키값 2 이니. 1 < 2 < 4 이므로 1 데이터 페이지로 가게 되며 1 데이터 페이지에서 2 번 한기환을 발견하게 될 것이다.

[SQL2011 강좌] 21. 트랜잭션에 대한이해

트랜젝션에 대한 이야기는 많은 곳에서.. 이런 저런 이유로 들어 보았을 것이다. 프로그래밍을 하실 때도 이 트랜젝션은 분명히 대단히 속도에 영향을 주는 요소 이다. 특히나 데이터 수정이 있을 경우인 Insert, Update, Delete 를 잘못 수행했을 경우 대단히 많은 문제를 줄 수 있으며.. - CPU 등의 시스템은 펑펑 놀고 있는데..SELECT 의 조회 속도가 대단히 느려지는 사태가 발생하는 것이다.

이럴 때는 한번쯤 트랜젝션 처리 모듈의 문제를 어플리케이션에서 점검을 해 보셔야 한다.

트랜젝션은 작업의 단위를 지정해 주는 중요한 부분이기 때문이다.

또한 트랜젝션은 SQL 구문이나 쿼리를 실제로 처리할 경우의 가장 작은 단위 이다. 은행이야기를 조금 해 볼까? 코난이 계좌에 200 원이 있고 수선이 계좌에 300 원이 있다.내기에서 진 코난이가 100 원을 수선이에게 잃어 200 원중 100 원을 온라인으로 계좌이체 를 시켜줘야 한다.

은행에서 온라인으로 입금을 하기 위해

1. 코난이 통장에서 100 원을 뺌 - 코난이 통장 = 100 원 수선이 통장 = 200 원

2. 수선이 통장에 100 원을 더함 - 코난이 통장 = 100 원 수선이 통장 = 300 원

하려고 했다. 그런데 웬일.. 정전이 나서... 1 번 작업을 끝마치니 시스템이 꺼졌다.

"대우네 은행"에서 그 당시 온라인으로 계좌 이체를 시킨 사람이 1000 명이었다...

그 999 명과 코난이는(도합 1000 명) 이 문제를 어떻게 할 것인가?

저런 문제가 발생할 수 있다.

저 내용을 우리가 사용하는 구문으로 살짝 돌려 본다면 이런 이야기가 될지 모른다.

```
update 계좌 set 금액 = 금액 - 100 where id like '코난'

update 계좌 set 금액 = 금액 + 100 where id like '수선'
```

하지만 위의 쿼리가 아니다. SQL 서버에서 위와 같은 처리를 위해서는 위처럼 쓰면 안된다. 위와 같은 구문으로 쓰면? 위의 쿼리는 다음과 같이 변환 된다.

```
begin tran

update 계좌 set 금액 = 금액 - 100 where id like '코난'

commit tran



begin tran

update 계좌 set 금액 = 금액 + 100 where id like '수선'

commit tran
```

이렇게 변화하게 된다. - 내부적으로 SQL 서버의 엔진이 저렇게 바꾸는 것이다. 그렇다면 어떻게 해야 하나의 단위로 위 두 개의 작업을 하나로 묶어서 수행하게 할 수 있을까? 바로 다음과 같이 명시적으로 begin tran 과 commit tran 을 사용하면 된다.

```
begin tran

update 계좌 set 금액 = 금액 - 100 where id like '코난'

update 계좌 set 금액 = 금액 + 100 where id like '수선'

commit tran
```

이렇게 사용하시면 두개의 작업이 하나의 단위로 처리가 되게 된다.

<트랜젝션의 네가지 성격>

트랜젝션은 DBMS 차원에서.. 다음 네 가지 성격을 만족해야만 한다.

1. 원자성(atomicity) : 트랜젝션은 전부, 전무의 실행만이 있지.. 일부 실행으로 트랜젝션의 기능을 가질 수는 없다.

2. 일관성(consistency) : 트랜젝션이 그 실행을 성공적으로 완료하면 언제나 일관된 데이터베이스 상태로 된다라는 의미이다. 즉, 이 트랜젝션의 실행으로 일관성이 깨지지 않는다 라는 의미 이다.

3. 격리성(isolation) : 연산의 중간결과에 다른 트랜젝션이나 작업이 접근할 수 없다..라는 의미이다.

4. 영속성(durability) : 트랜젝션의 일단 그 실행을 성공적으로 끝내면 그 결과를 어떠한 경우에라도 보장받는다.. 라는 의미이다.

<장애(failure)와 회복(recovery)>

회복(recovery)이란 장애(failure)가 일어났을 때 데이터 베이스를 장애 이전의 상태로 다시 복구 시켜 일관된 상태로(consistent state)로 복구 시키는 작업이다. 장애를 조금 더 알아본다면? 관리자가 구두로 컴퓨터 파워 버튼을 실수로 차버리거나. 해서 문제가 생기는 상황?? 이렇게 대충은 생각이 가능할 것이다.

조금만 더 세분화를 시켜 본다면..

1. 트랜젝션 장애

: 트랜젝션 내의 논리적 오류나 내부조건, 입력 데이터의 불량, 데이터의 불명, 시스템 자원의

과다한 사용요구 등으로 정상적인 실행을 계속 할 수 없는 상태를 의미 한다.

2. 시스템 장애

: 이는 하드웨어의 오동작으로 메인 메모리에 있는 정보의 손실이나 교착 상태가 발생하여

더 이상 실행을 계속할 수 없는 상태를 의미 한다...

3. 미디어 장애

: 디스크 헤드 붕괴나 고장으로 인하야... 저장 장치의 데이터베이스 일부 또는 전부가

손상된 상태이다.

이렇게 세분화가 가능해 진다. 여기서 회복이란 말 그대로 데이터를 안정된 상태로 복구 시키는 것을 의미 한다.

[SQL2011 강좌] 22. 트랜잭션의 격리수준 4 가지

SQL Server 는 총 6 가지 격리 수준을 제공하고 있다.

하지만 오늘은 1~4 번에 해당하는 격리 수준에 대해서 알아보려고 한다.

[트랜잭션 격리 수준]

1. READ UNCOMMITTED
2. READ COMMITTED
3. REPEATABLE READ
4. SERIALIZABLE

SQL Server 2005부터 추가된 두가지
~~5. READ COMMITTED WITH SNAPSHOT~~
~~6. SNAPSHOT~~

[선행 필요 Lock에 대한 지식]

1. **S Lock: 공유 잠금(Shared Lock)**
2. **X Lock: 단독 잠금(Exclusive Lock)**
3. **Row, Page, Extent에 대한 기본 개념 및 잠금 범위**
4. **암시적과 명시적 트랜잭션에 대한 이해**

S Lock은.. 데이터를 읽어올때 주로 걸리는 Lock 입니다.
X Lock은.. 데이터를 입력/수정/삭제 할 때 걸리는 Lock 입니다.

X Lock은 혼자만 잠글 수 있기 때문에.. 다른 어떤 Lock이랑도 호환되지 않습니다. 하지만.. S Lock은... 다른 S Lock과 호환이 되기 때문에.. 서로 공유할 수 있습니다.

이 외에도 업데이트락, 인텐트 락등이 있지만.. 이번에 설명드릴 부분외라 따로 설명드리진 않겠습니다!

1. READ UNCOMMITTED

READ UNCOMMTED 를 해석을 하게되면 => COMMIT 되지 않은 데이터를 읽을 수 있다.

이 말은.. 다른 세션에서 트랜잭션을 걸어서 데이터를 INSERT 나 UPDATE 할 경우... COMMIT 이 되지 않았는데도 불구하고, 데이터를 읽을 수 있다는 말입니다.

마치 남자친구의 맘속을 훤히 모두 들여다보는..변심도 미리 알아채는 여친님의 마음이라 볼 수 있죠~!

예제를 통해 살펴보도록 하겠습니다.

#1번 세션

```
BEGIN TRAN
UPDATE tblClusteredIndex
SET DATA = '하'
```

#2번 세션

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT * FROM tblClusteredIndex
GO
--결과
IDX   DATA
1     하
2     하
3     하
4     하
5     하
6     하
7     하
```

데이터를 SELECT 할때 S Lock으로 잠근다고 미리 언급을 했습니다.

하지만.. **READ UNCOMMITTED 에서는 S Lock 조차 걸지 않기 때문에..**
다른 세션에서 커밋되지 않은 수정하는 (X Lock 데이터) 데이터를 볼 수 있습니다.

바로 변심한 남친의 마음을 즉각 알아채는 여친의 마음!

이걸 Dirty Read 라고 합니다.

```
INSERT INTO tblClusteredIndex SELECT 1, '가'
INSERT INTO tblClusteredIndex SELECT 2, '나'
INSERT INTO tblClusteredIndex SELECT 3, '다'
INSERT INTO tblClusteredIndex SELECT 4, '라'
INSERT INTO tblClusteredIndex SELECT 5, '마'
INSERT INTO tblClusteredIndex SELECT 6, '바'
INSERT INTO tblClusteredIndex SELECT 7, '사'
GO

--//tblClusteredIndex 에는 IDX Column에 PRIMARY KEY 생성
ALTER TABLE tblClusteredIndex
ADD CONSTRAINT tblClusteredIndex_PK PRIMARY KEY(IDX)
GO
```

2. READ COMMITTED

READ COMMITTED 를 해석을 하게되면 => COMMIT 된 데이터만 읽을 수 있다.
READ UNCOMMITTED와 다른 점은 실제 SELECT 할때 S Lock을 소유 합니다. 하지만!!! SELECT 가 끝나는 동시에 S Lock을 해제 합니다.

이말은..... 트랜잭션 내에서 SELECT가 끝나더라도(커밋되기전) 다른 세션에서 INSERT나 UPDATE가 가능하다는 말이 됩니다.

남친의 변한 마음만 확인할 수 있는 여친의 마음... 훗~~ 속마음까진 확인할 방법이.....없다!

예제를 통해 살펴보도록 하겠습니다.

1번 예제

#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
SELECT * FROM tblClusteredIndex
GO
--결과
IDX   DATA
1     가
2     나
3     다
4     라
5     마
6     바
7     사

--트랜잭션이 커밋되지 않았지만... SELECT로 이미 결과를 반환했습니다. (S Lock에 해제 된 상태)
```

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

BEGIN TRAN
UPDATE tblClusteredIndex
SET DATA = '가'
WHERE IDX = 7
COMMIT  --//수정한 데이터를 COMMIT(1번 세션 S Lock이 해제 되었기 때문에 UPDATE 가 가능)
```

#다시 1번 세션

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT * FROM tblClusteredIndex
GO
--결과
IDX  DATA
1    가
2    나
3    다
4    라
5    마
6    바
7    가
--//2번 세션에서 이미 커밋을 했기 때문에 변경된 내용을 불러옵니다.
```

만약 SELECT 하더라도 COMMIT 되기 전까지 S Lock을 소유 한다면...
다른 세션에선 업데이트가 불가능 하겠죠..
(사실 이게 REPEATABLE READ 입니다. 뒤에서 설명드리겠습니다.)

SELECT가 아닌 업데이트는 어떻게 다른지 보도록 하겠습니다.

2번 예제

#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
UPDATE tblClusteredIndex
SET DATA = '가'
WHERE IDX = 7
GO
```

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT * FROM tblClusteredIndex WHERE IDX = 7
--대기 상태.............
```

READ COMMITTED의 해석 기억하십니까? 커밋된 데이터만 읽기가 가능하다는 얘기죠!

1번 세션에서 아직 COMMIT을 하지 않았기 때문에.. 2번 세션에선 IDX = 7을 읽지 못하고 1번 세션의 COMMIT을 기다리게 됩니다.

그렇다면 2번 세션에서 아래와 같은 쿼리를 날리면 어떻게 될까요?

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT * FROM tblClusteredIndex WHERE IDX = 1
--결과
IDX   DATA
1      가
```

1번 세션에서 수정으로 인해 IDX = 7 인 row를 X Lock을 걸은 상태기 때문에.. 다른 컬럼들은 가지고 나오는데 문제는 없습니다.

그렇다면 .. 아래도 충분히 가능하겠죠~!

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
UPDATE tblClusteredIndex
SET DATA = '가'
WHERE IDX = 1
GO
ROLLBACK
```

다른 ROW에 대해서 X 락을 걸었으니까요~ 이 증거가 SQL Server가 ROW Lock이 가능한 이유입니다.

3. REPEATABLE READ

REPEATABLE READ 를 해석을 하게되면 => 반복 읽기

한 트랜잭션 안에서 내가 읽었던 녀석은 절대 변하면 안되~~!

마치.. 간절한 여자친구처럼 오빠 마음 절대 변하면 안되!! 이렇게 Lock을 걸어두는 것이죠...
만날때부터 헤어질때까지~~~ㅎㅎ

이미 위에서 한번 설명을 드렸죠!! 트랜잭션내에서 읽은 녀석은.. 다른 세션에서 변경조차 안됩니다.
물론 S Lock이기 때문에.. 읽을 순 있겠죠 ^^;

1번 예제

#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
SELECT * FROM tblClusteredIndex
GO
--결과
IDX   DATA
1     가
2     나
3     다
4     라
5     마
6     바
7     사

--트랜잭션이 커밋되지 않았지만...  S Lock은 계속 유지 됩니다.
```

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

BEGIN TRAN
UPDATE tblClusteredIndex
SET DATA = '가'
WHERE IDX = 7
---------- 대기 상태
```

이런 경우 1번세션에서 이미 S Lock을 걸고 있기 때문에..수정할 수가 없게 됩니다.
하지만 읽을 순 있겠죠~

#2번 세션 을 중지하고..

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
SELECT * FROM tblClusteredIndex
GO
--결과
IDX   DATA
1     가
2     나
3     다
4     라
5     마
6     바
7     사
--읽는건 정상적으로 됩니다. 당근 S Lock은 S Lock 끼리 공유할 수 있으니깐요 ^^
```

2번 예제를 위해서 데이터를 1,3,5만 남기고 지워봅시다.

2번 세션에 ROLLBACK을 하지 않았기 때문에.. 우선 ROLLBACK을 시킵니다.

#2번 세션(혹시나 모를 롤백 시키기)

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO
```

#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO
DELETE FROM tblClusteredIndex WHERE IDX NOT IN (1,3,5)
GO
```

2번 예제

REPEATABLE READ는 처음 읽은놈은 끝까지 유지한다고 했습니다.
그럼 중간에 낑겨 드는 놈들은 어떻게 될까요?

#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
SELECT * FROM tblClusteredIndex
--결과
IDX        DATA
1          가
3          다
5          마
```

#2번 세션

```
INSERT INTO tblClusteredIndex SELECT 2, '나'
--오~~ 입력이 됩니다.
```

#1번 세션(커밋되기전..)

```
SELECT * FROM tblClusteredIndex
--결과
IDX        DATA
1          가
2          나
3          다
5          마
```

REPEATABLE READ는 처음 읽었을 당시에 존재하는 데이터에 대해서만 Lock을 걸게 됩니다.
따라서 2번째 세션에서 입력된 IDX = 2의 값은.. 첫번째 세션에서 SELECT 할 당시엔 없었기 때문에.. LOCK이 걸리지 않았으므로..
INSERT가 되는것이죠

오빠 마음이 변하진 않지만.... 다른 미모의 여인이 나타나면.. 여친몰래 마음속에 INSERT 되듯이 말이죠!!
물론 여친은 알아채지 못하겠죠 ㅋㅋㅋ..

이 외도의 주인공!! 바로 팬텀이라고 부릅니다. ^^

그렇다면 미모의 여인도 접근 못하게 확실히 틀어 막는 방법은 없을까요???...

4. SERIALIZABLE

SERIALIZABLE => 직렬화.....

마치.. 일거수 일투족을 감시하는 여자친구처럼 옆에 찰싹 붙어서 다른 어떤 여자도 들어오지 못하게 막는게 바로! SERIALIZABLE 입니다.

예제를 바로 보시죠!

1번 예제

1번 예제를 위해서 데이터를 1,3,5만 남기고 다시 지워봅시다. 그리고 7을 넣습니다.
#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO
DELETE FROM tblClusteredIndex WHERE IDX NOT IN (1,3,5)
GO
INSERT INTO tblClusteredIndex SELECT 7,'사'
GO
```

#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
SELECT * FROM tblClusteredIndex WHERE IDX BETWEEN 3 AND 5
GO
--결과
IDX   DATA
3     다
5     마
-- RangeS-S Lock이라는 녀석으로 범위로 S Lock을 걸어버립니다.
```

이때 4가 들어가려 한다면??? REPATABLE READ와 다르게 분명 막힐 것입니다.

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

INSERT INTO tblClusteredIndex SELECT 4, '나'
---------- 대기 상태
```

하지만 여기서 3~5 범위에 상관없는 2가 들어가려고 하면 어떻게 될까요??

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

INSERT INTO tblClusteredIndex SELECT 2, '나'
---------- 대기 상태
```

똑같이 대기 상태입니다.

SERIALIZABLE은.. 3 AND 5 로 SELECT로 잠궜다고 하더라도... 3~5 까지 잠그는 것이 아닙니다.

3 이전행인 1 부터... 5 다음행인 7까지 잠그게 됩니다.

결국... 1 <= IDX <=7인 값은 다른 세션에서 INSERT가 되지 않습니다.

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

INSERT INTO tblClusteredIndex SELECT 100, '나'
--성공!
```

만약... 1 AND 3으로 잠궜다면??? 테이블에 1보다 작은 값은 존재하지 않고, 3보다 큰 값은 5기 때문에..
IDX >= 6인 값만 INSERT 가 가능하게 됩니다. 물론 0보다 작은 값은 절대 들어갈 수 없습니다.

물론 다른 세션에서 말이죠!

SERIALIZABLE에서 테이블에 인덱스가 없으면 어떻게 될까요?

#1번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
SELECT * FROM tblOnlyHeap WHERE IDX BETWEEN 3 AND 5
GO
--결과
IDX   DATA
3      다
5      마
```

#2번 세션

```
IF(@@TRANCOUNT > 0) ROLLBACK
GO

INSERT INTO tblClusteredIndex SELECT 0, '나'
---------- 대기 상태

INSERT INTO tblClusteredIndex SELECT 100000, '나'
---------- 대기 상태
```

테이블에 인덱스가 없기 때문에.. 범위를 알 수 없습니다. 그래서 모두 잠그게 됩니다.

[SQL2011 강좌] 23. 잠금에 대한 이해

트랜잭션은 잠금과 대단히 밀접한 관계가 있다.

예를 들어.. 공유 폴더에 워드 문서를 두고 워드 문서를 오픈 하고 다른 사용자가 한번 더 워드 파일을 오픈 할 경우에 **다른 프로세스나 사용자가 현재 해당 파일을 오픈 하고 있다면 읽기 전용으로 여시겠습니까? 라는 메세지와 함께 파일은 읽기 전용으로 열리거나 열 수 없을 것이다.**
이런 상황이 바로 잠금의 개념과 비슷하다.
두 사용자가 모두 쓰기를 해 버리면? 어느 것을 적용시켜야 할 지 알 수 없겠다. DB 의 경우라면 더더욱 큰 문제가 된다. 기본적으로 병행 DBMS 는 기본적으로 여러 명의 사용자프로세스가 붙어서 작업을 하게 되니 위와 같은 병행처리는 기본적으로 제공이 되어야 한다.
그 개념이 바로 잠금이며 SQL 서버는 이 잠금이 대단히 다양하고 세분화 되어 있다. 먼저 잠금 내역을 확인해 보면

```
--//1 번 세션

use pubs

go

--가격* 2

begin tran

update titles set price = price * 2
```

```
--//2 번 세션

--하나의창을더오픈한다.

--파일-> 연결을하고같은서버에접속한다.

use pubs

go
```

```
--데이터조회

select * from titles

--조회가불가하다.
```

```
--//3 번 세션

--하나의창을더오픈한다.

--파일-> 연결을하고같은서버에접속한다.

use pubs

go

--잠금정보를조회한다.

sp_lock
```

Results | Messages | Execution plan

	spid	dbid	ObjId	IndId	Type	Resource	Mode	Status
1	52	1	1483152329	0	TAB		IS	GRANT
2	52	6	0	0	DB		S	GRANT
3	53	6	0	0	DB		S	GRANT
4	53	6	389576426	1	KEY	(35d5699be98f)	S	WAIT
5	53	6	389576426	1	PAG	1:163	IS	GRANT
6	53	6	389576426	0	TAB		IS	GRANT
7	54	6	389576426	0	TAB		IX	GRANT
8	54	6	389576426	1	KEY	(9faa7ef924c8)	X	GRANT
9	54	6	389576426	1	KEY	(4ce4ddbadbad)	X	GRANT
10	54	6	389576426	1	KEY	(1762b92bf1ed)	X	GRANT
11	54	6	389576426	1	KEY	(35d5699be98f)	X	GRANT
12	54	6	389576426	1	PAG	1:163	IX	GRANT
13	54	6	389576426	1	KEY	(51af2bbf61e9)	X	GRANT
14	54	6	389576426	1	KEY	(abb59a50f741)	X	GRANT
15	54	6	389576426	1	KEY	(a905e86e826d)	X	GRANT
16	54	6	389576426	1	KEY	(1759aac600cc)	X	GRANT
17	54	6	389576426	1	KEY	(0917b96f2ad1)	X	GRANT
18	54	6	389576426	1	KEY	(8bd2832523a6)	X	GRANT
19	54	6	389576426	1	KEY	(cb5035369cd0)	X	GRANT
20	54	6	389576426	1	KEY	(28e28bed9a4f)	X	GRANT
21	54	6	389576426	1	KEY	(f1128f4da721)	X	GRANT
22	54	6	389576426	1	KEY	(f296e9925f52)	X	GRANT
23	54	6	389576426	1	KEY	(266a7de4408e)	X	GRANT
24	54	6	389576426	1	KEY	(8c9a725e59fd)	X	GRANT
25	54	6	389576426	1	KEY	(d6cf2fb2beda)	X	GRANT
26	54	6	389576426	1	KEY	(e6e19e753289)	X	GRANT
27	54	6	0	0	DB		S	GRANT

맨 처음 보시는 쿼리는 begin tran 으로 트랜잭션 범위 내에서 시작하고 commit 이나 rollback 등 트랜잭션 끝 이라는 키워드가 없다.

즉, 트랜잭션 중으로 하겠다는 것이다. 그리고 이때 다른 사용자는 계속 pubs 에서 대기하게 되는 것이다. - 나중에 배우시겠지만 블러킹(Blocking)이라고 한다.

바로 현재 spid 가 53 번인 부분에 Status 가 WAIT 인 것을 볼 수 있다. 먼저 type 으로 잠금의 타입인이다. 잠금이 걸리는 형식 이다. 보면 DB, PAG, KEY, TAB 등이 있다. 잠금이 걸리는 단위는 무엇일까? 이 단위를 상세하게 알아 보도록 한다.

리소스 유형	설명
RID	테이블에 있는 한 행을 잠그기 위한 행 식별자이다.
KEY	인덱스에 있는 행 잠금이다. 순차 가능한 트랜잭션에서 키 범위를 보호하기 위해 사용한다.
PAG	데이터 또는 인덱스 페이지이다.
EXT	인접한 여덟 개의 데이터 페이지 또는 인덱스 페이지 그룹이다.
TAB	모든 데이터와 인덱스가 포함된 전체 테이블이다.
DB	데이터베이스이다.

SQL 서버는 이렇게 잠금의 단위를 쿼리에 따라서 자동으로 설정한다. 물론 사용자가 개입해서 잠금을 걸 수도 있다. 하지만 이렇게 사용자가 잠금을 거는 것은 거의 권장되지 않으며 오로지 BCP, Bulk insert 등과 같은 대용량 벌크 작업 시 TABLE Lock 을 걸 경우를 걸 때를 제외하고는 권장하지 않는다.

벌크작업 시는 걸고 해야만 제 속도를 얻을 수 있다. 아래는 사용자가 직접 잠금을 거는 예이다.

```
use pubs

go

begin tran

select * from authors (ROWLOCK) where au_id = '172-32-1176'
```

잠금의 힌트로 사용 가능한 옵션들은 아래와 같다.

잠금 참고	설명
HOLDLOCK	필요한 테이블, 행 또는 데이터 페이지가 더 이상 필요 없게 되자마자 해제하지 않고 트랜잭션이 완료될 때까지 공유 잠금을 보유한다. HOLDLOCK 은 SERIALIZABLE 과 같은 의미이다.
NOLOCK	공유 잠금을 실행하거나 단독 잠금을 유지하지 않는다. 이 옵션을 적용하면 커밋되지 않은 트랜잭션이나 읽는 중 롤백된 페이지 집합을 읽을 수 있다. 커밋되지 않은 읽기가 가능하다. SELECT 명령문에만 적용된다.
PAGLOCK	주로 단일 테이블 잠금이 취해지는 곳에서 페이지 잠금을 사용한다.
READCOMMITTED	READ COMMITTED 격리 수준에서 실행되는 트랜잭션과 같은 잠금 방법을 사용하여 스캔을 수행한다. 기본적으로, SQL Server 2000 은 이 격리 수준에서 실행된다.
READPAST	잠겨 있는 행을 건너뜁니다. 이 옵션을 사용하면 다른 트랜잭션이 이러한 행에 대해 잠금을 해제할 때까지 기다리지 않고 다른 트랜잭션에 의해 잠겨 있는 행을 건너뜁니다. 그렇지 않으면 일반적으로 결과 집합에 나타납니다. READPAST 잠금 참고는 READ COMMITTED 격리 수준에서 작동하는 트랜잭션에만 적용되며 행 수준 잠금 뒤만 읽는다. SELECT 문에만 적용된다.
READUNCOMMITTED	NOLOCK 과 같다.

REPEATABLEREAD	REPEATABLE READ 격리 수준에서 실행되는 트랜잭션과 같은 잠금 방법으로 스캔을 수행한다.
ROWLOCK	성긴 페이지 잠금 및 테이블 수준의 잠금 대신 행 수준 잠금을 사용한다.
SERIALIZABLE	SERIALIZABLE 격리 수준에서 실행되는 트랜잭션과 같은 잠금 방법으로 스캔을 수행한다. HOLDLOCK 과 같다.
TABLOCK	세부적인 행 또는 페이지 수준 잠금 대신 테이블 잠금을 사용한다. SQL Server 는 명령문이 끝날 때까지 이 잠금을 보유한다. 그러나 HOLDLOCK 을 함께 지정했으면 트랜잭션이 끝날 때까지 잠금이 보유된다.
TABLOCKX	테이블에 대해 단독 잠금을 사용한다. 이 잠금을 사용하면 다른 트랜잭션이 테이블을 읽거나 업데이트할 수 없고 명령문이나 트랜잭션이 끝날 때까지 보유된다.
UPDLOCK	테이블을 읽는 중 공유 잠금 대신 업데이트 잠금을 사용하며 명령문이나 트랜잭션이 끝날 때까지 보유된다. UPDLOCK 을 사용하면 다른 트랜잭션이 읽는 것을 차단하지 않고 데이터를 읽을 수 있고 마지막으로 읽은 후 데이터가 변경되지 않으며 나중에 업데이트할 수 있다.
XLOCK	명령문에 의해 처리되는 모든 데이터에 대해 트랜잭션이 끝날 때까지 보유될 단독 잠금을 사용한다. 이 잠금은 PAGLOCK 또는 TABLOCK 으로 지정할 수 있으며 이 경우 단독 잠금이 해당 세부성 수준에 적용된다.

사용자가 잠금 힌트를 주는 것 보다는 SQL 서버가 자동으로 정하게 하는 것이 대부분의 상황에서 좋으며 잠금의 범위가 커질 경우 역시 SQL 서버가 자동으로 락을 에스켈레이션 하게 하는 것이 좋다. 아래는 리소스 컬럼의 정보 이다.

리소스	설명
RID	테이블 내의 잠겨진 행의 행 식별자이다. 행은 필드:페이지:rid 조합으로 식별된다. 여기서 rid 는 해당 페이지의 행 식별자이다.
KEY	SQL Server 에서 내부적으로 사용되는 16 진수이다.
PAG	페이지 번호이다. 페이지는 필드:페이지 조합으로 식별된다. 여기서 파일 번호는 **sysfiles** 테이블의 **fileid** 이며, 페이지는 해당 파일 내의 논리적인 페이지 번호이다.
EXT	익스텐트에서 첫 번째로 잠글 페이지 번호이다. 페이지는 필드:페이지 조합으로 식별된다.
TAB	**ObjId** 열에 테이블의 개체 ID 가 이미 있으므로 아무 정보도 제공되지 않는다.
DB	**dbid** 열에 데이터베이스의 데이터베이스 ID 가 이미 있으므로 아무 정보도 제공되지 않는다.

잠금 모드	설명
공유(S)	SELECT 문처럼 데이터를 변경하거나 업데이트하지 않는 작업(읽기 전용 작업)에 사용한다.

업데이트(U)	업데이트할 수 있는 리소스에 사용한다. 여러 개의 세션이 리소스를 읽고, 잠그고, 나중에 업데이트할 때 발생하는 일반적인 교착 상태를 방지한다.
단독(X)	INSERT, UPDATE, DELETE 와 같은 데이터 수정 작업에 사용한다. 여러 개의 업데이트 작업이 같은 리소스에 대해 동시에 이루어지지 못하게 한다.
내재	잠금 계층 구조를 만드는 데 사용한다. 내재된 잠금의 종류에는 내재된 공유(IS) 잠금, 내재된 단독(IX) 잠금, 공유 및 내재된 단독(SIX) 잠금이 있다.
스키마	테이블의 스키마에 종속되는 작업이 실행될 때 사용한다. 스키마 잠금의 종류에는 스키마 수정(Sch-M)과 스키마 안전성(Sch-S) 두 가지가 있다.
대량 업데이트(BU)	데이터를 테이블로 대량 복사하는 경우와 **TABLOCK** 참고가 지정된 경우 사용한다.

아래 표는 내재된 잠금에 대한 설명이다.

잠금 모드	설명
내재된 공유(IS) 잠금	트랜잭션이 각 리소스에 대해 S 잠금을 설정하여 계층의 아래쪽에 있는 일부 리소스를 읽으려 하는 것이다.
내재된 단독(IX) 잠금	트랜잭션이 각 리소스에 대해 X 잠금을 설정하여 계층의 아래쪽에 있는 일부 리소스를 수정하려 하는 것이다. IX 는 IS 의 상위 집합이다.
공유 및 내재된 단독(SIX) 잠금	트랜잭션이 각 리소스에 대해 IX 잠금을 설정하여 계층의 아래쪽에 있는 모든 리소스를 읽고 일부 리소스를 수정하려 하는 것이다. 최상위 수준 리소스에서는 동시 IS 잠금이 허용된다.예를 들어, 테이블에 대한 SIX 잠금은 테이블에 대해 SIX 잠금을 설정하여 동시 IS 잠금을 허용하고, 수정 중인 페이지에 IX 잠금을 설정하고 수정된 행에 대해 X 잠금을 설정한다. 리소스 당 한 번에 하나의 SIX 잠금을 설정할 수 있으므로 다른 트랜잭션이 테이블 수준에서 IS 잠금을 얻어 계층 아래쪽에 있는 리소스를 읽을 수 있어도 다른 트랜잭션이 리소스를 업데이트할 수는 없다.

내재 잠금은 계층으로 볼 때 동시에 여러 개의 로우에 잠금이 걸리면 페이지 잠금 등으로 잠금을 관리 할 수 있다. 그런 계층적인 잠금에서 상위 잠금에 대해서 내재된 잠금을 걸고 처리하게 되는 것이다. 보시는 바와 같이 하위에 대해서 X(단독잠금)이 걸릴 경우 상위 개체는 (IX)내재된 단독 잠금 모드가 걸리게 된다.

중요하게 또한 보셔야 하는 부분은 잠금 / 프로세스 이다. 예를 들어. 맨 위의 쿼리를 생각해 보면. 그 쿼리는 update titles set price = price * 2 라는 구문으로 다른 사용자의 select * from titles 쿼리를 막고 있다. 그렇다면.. 바로 문제가 되는 update 쿼리의 사용자가 잠금의 차단 주체(차단하는 녀석)이 될 것이고 select 하는 사용자는? 차단을 받고 있을 것이다. 이것을 볼 수 있게 해주는 것은 sp_who 라는 명령이 있다.

sp_who2

Results | Messages | Execution plan

	SPID	Status	Login	HostName	BlkBy	DBName	Command	CPUTime	DiskIO	LastBatch	ProgramName	SPID
7	7	BACKGROUND	sa	.	.	NULL	XE DISPATCHER	0	0	06/16 20:43:01		7
8	8	BACKGROUND	sa	.	.	master	BRKR TASK	0	0	06/16 20:43:01		8
9	9	BACKGROUND	sa	.	.	master	CHECKPOINT	31	124	06/16 20:43:01		9
10	10	BACKGROUND	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		10
11	11	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		11
12	12	BACKGROUND	sa	.	.	master	TRACE QUEUE TASK	0	0	06/16 20:43:01		12
13	13	BACKGROUND	sa	.	.	NULL	RECEIVE	374	0	06/16 20:43:01		13
14	14	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		14
15	15	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		15
16	16	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		16
17	17	BACKGROUND	sa	.	.	master	BRKR EVENT HNDLR	0	43	06/16 20:43:01		17
18	18	BACKGROUND	sa	.	.	master	BRKR TASK	0	0	06/16 20:43:01		18
19	19	BACKGROUND	sa	.	.	master	BRKR TASK	0	0	06/16 20:43:01		19
20	20	BACKGROUND	sa	.	.	master	BRKR TASK	0	0	06/16 20:43:01		20
21	21	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		21
22	22	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		22
23	23	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		23
24	24	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		24
25	25	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		25
26	26	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		26
27	27	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		27
28	28	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		28
29	29	sleeping	sa	.	.	master	TASK MANAGER	0	0	06/16 20:43:01		29
30	51	sleeping	ea...	EASTLU...	.	master	AWAITING COMMAND	94	52	06/16 23:33:59	Microsoft S...	51
31	52	RUNNABLE	ea...	EASTLU...	.	pubs	SELECT INTO	1480	487	06/16 23:43:39	Microsoft S...	52
32	53	SUSPENDED	ea...	EASTLU...	54	pubs	SELECT	15	0	06/16 23:39:59	Microsoft S...	53
33	54	sleeping	ea...	EASTLU...	.	pubs	AWAITING COMMAND	16	1	06/16 23:39:55	Microsoft S...	54

하게 되면 blk 라는 컬럼에 차단 주체를 볼 수 있게 된다. 이렇게 SPID 54 번 프로세스가 막고 있게된다. SPID 54 번 프로세스는 53 번 바로 아래 있다. 이렇게 차단이 일어날 경우를 블러킹(Blocking)이 어난다고 부르며 블러킹은 데이터베이스 리소스를 여러 명이 사용하기 때문에 당연히 일어나는 것이다.

- 특히 돈이나 재무와 관련되는 프로세스일 경우는 대부분 이 트랜잭션을 이용한 처리를 하기 때문에 종종 블러킹이 일어 난다. 하지만 대단히 응답속도가 늦거나 - CPU 는 놀고 있는데도 불구하고 그렇다면 여러 블러킹 처리를 확인해서 개선할 필요가 있다.

[SQL2011 강좌] 24. 백업과 복구에 대한 이해(풀백업, 차등백업, 트랜잭션 로그 백업)

이번에 소개해 드릴 내용은 백업과 복구 이다.

<백업이란 무엇인가?>

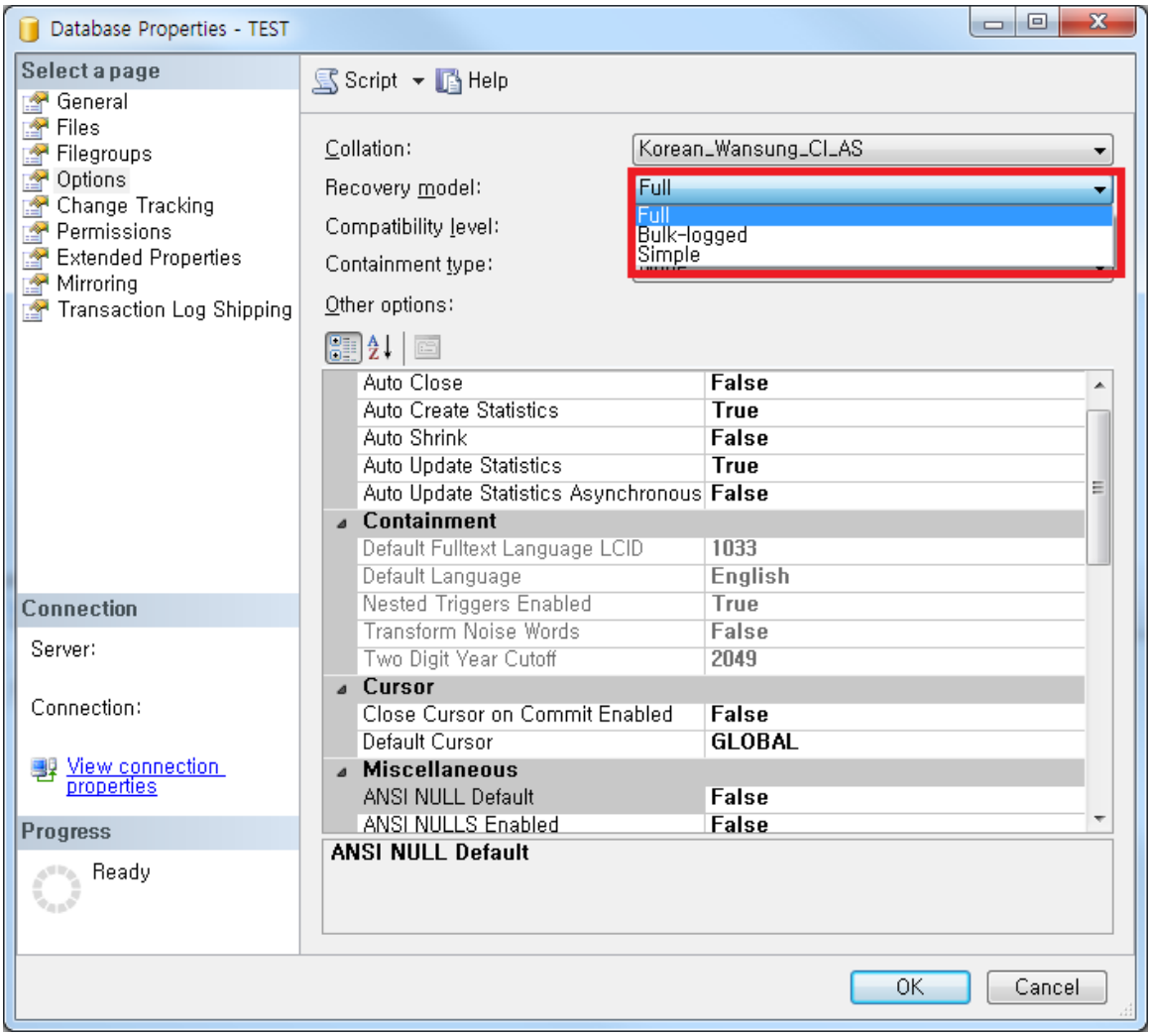
백업은 간단히 데이터를 복사해 둔다는 의미로 그냥 복사만 하는 것이 아니라 정확히, 가능한 많은 데이터를 문제가 발생시 복구하기 위해 복사본을 만들어 두는 것이다.

전체백업 = 풀백업을 의미한다.
차등백업 = 디퍼런셜 백업과 같은 말이다.

SQL Server 는 복구 모델이라고 해서 3 가지 방식을 제공한다.

1. 단순
2. 대량 로그
3. 최대

엔터프라이즈 관리자의 아무 데이터베이스에서 등록정보 -> 옵션 탭을 보면



Full: 전체

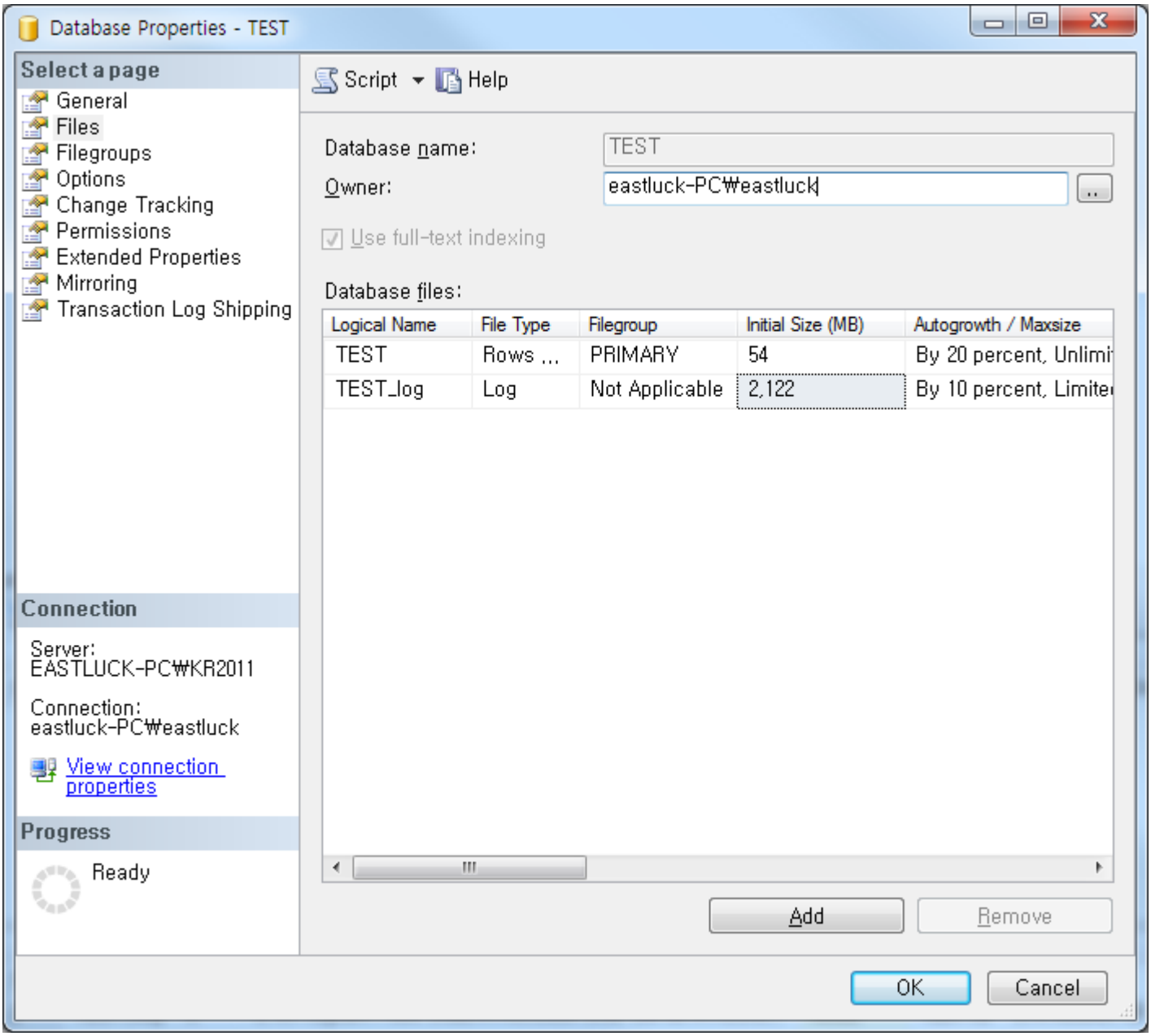
Bulk-logged: 대량 로그

Simple: 단순

이러한 화면을 볼 수 있다. 복구 모델은 데이터베이스를 특정 상태로 지정하는 것이다.

로그는 데이터의 변경이 저장되는 곳이다. 따라서 대부분의 경우 실제 데이터에 비해서 로그의 크기는 매우 작다. 그리고 **로그는 백업하면 비워집니다. - 지워진다기 보다는 비워진다는 게 정확하다.** 껍데기는 그대로 이며 오로지 로그 데이터만이 비워지기 때문이다. 만약 로그를 한번도 백업을 받지 않으셨다면?

다음 그림처럼.. 약간 이상한 상태를 보게 된다.



대단히 흥미 있게도..데이터는 54 메가 인데..

트랜잭션 로그는 한번도 백업을 받지 않았기 때문에 트랜잭션 로그 공간은 2.4 기가 가까이 된다. 로그는 매우 작은 분량임에도 불구하고 한번도 로그를 백업하지 않아 생기는 상황이다. 사용되는 공간도 문제이지만 로그 데이터로 인해 시스템이 대단히 느려지는 사태 역시 발생할 수 있게 된다.

1. 예를 들어 pus 데이터베이스라면

```
use pubs

go

DBCC SHRINKFILE (pubs_log, TRUNCATEONLY)
```

명령으로 로그를 비워낸 후 불필요하게 커져버린 로그 화일의 크기를 줄일 수 있다.

2. 두 번째로 확실한 방법으로 아래와 같이 할 수도 있다.

```
--작업을수행하기전만약을위해풀백업을받는다.

--해당하는 DB 를 detach 로 떼어낸다.

EXEC sp_detach_db 'pubs', 'true'



--잠시 ldf 화일의이름을바꿔두자. – 만약을위해지우지않는다.

--mdf 파일 만으로 다시 붙인다.

--이때 ldf(로그화일)화일은최소크기인 1M 로재생성된다.

EXEC sp_attach_single_file_db 'pubs'

, 'c:₩Program Files₩Microsoft SQL Server₩MSSQL₩Data₩pubs.mdf'
```

다음으로 복구의 모델 세 가지를 알아 보도록 합시다.

복구 가능 분량에 대해서 보는 게 좋겠지요.

- **최대 모델** : 가장 많은 데이터를 복구할 수 있다. 실제 문제가 발생한 시각까지의 모든 데이터를 복구가 가능하다.
- **대량로그** : 마지막 로그 백업 때까지만 복구가 가능하다.
- **단순** : 이 모델로 지정 시 로그 백업이 불가하므로 오로지 마지막 전체 백업 시까지만 복구가 가능하다.

또한 종종 접하는 문제 상황 발생시의 해결 방법을 알아 보도록 한다.

Q. 현재 웹사이트를 개발하는 중이다. 개발 중이라 한번도 백업을 받은 적이 없다. 개발자의 실수로 update 회원테이블 set 이름 = '아무개' 로 WHERE 절 없이 update 를 해 버렸다. 복구할 방법이 없을까?

A. 복구는 불가하다.

Q. 지속적으로 전체 백업만을 받아 왔다. 개발자의 실수로 update 회원테이블 set 이름 = '아무개' 로 WHERE 절 없이 update 를 해 버렸다. 복구할 방법이 없을까?

A. 제약 조건이 있다.

1. 복구 모델이 전체 모델일 것(DB 를 생성하면 기본이다.)
2. 문제 발생 후 풀백업이나 로그에 쓰지 않는 작업을 하지 않았을 것
3. 로그 백업이 가능할 것 이다.

만약 이 세 조건이 맞다면 바로 로그를 with no_truncate 옵션으로 백업하신 후.

Stop at 명령으로 복구할 수 있다.

Q. 풀백업과 로그백업을 혼용해 지속적으로 백업하고 있다. 개발자의 실수로 update 회원테이블 set 이름 = '아무개' 로 WHERE 절 없이 update 를 해 버렸다. 복구할 방법이 없을까?

A. 가능하다.

1. 복구 모델이 전체 모델일 것(DB 를 생성하면 기본이다.)
2. 문제 발생후 풀백업이나 로그에 쓰지 않는 작업을 하지 않았을 것 이 두 조건이 맞다면 즉시 로그를 with no_truncate 옵션으로 백업 후 Stop at 명령으로 복구할 수 있다.

Q. 이전 서버에서 사용하던 시스템의 문제로 해당하는 데이터베이스의 mdf 화일과 ldf 화일만 남았다. 새로 설치하는 시스템에 복구가 가능할까?

A. 이전 서버와 문자셋 정보(sp_helpsort 명령으로 보실 수 있다.)가 같고 이전에 사용하던 SQL 서버가 정상적인 상태로 종료 되었을 때(엔터프라이즈 관리자에서 SQL 서버를 스탑 시켰거나.. 정상적으로 윈도우 시스템을 종료시킨 경우 – 해당 mdf 화일과 ldf 화일이 정상적으로 닫혔을 경우) 복구가 가능하며 이때 사용하는 명령은

```
EXEC sp_attach_db 'pubs'

, 'c:₩Program Files₩Microsoft SQL Server₩MSSQL₩Data₩pubs.mdf'

, 'c:₩Program Files₩Microsoft SQL Server₩MSSQL₩Data₩pubs_log.ldf'
```

명령으로 복구할 수 있다.

Q. 이전 서버에서 사용하던 시스템의 문제로 해당하는 데이터베이스의 mdf 화일만 남았다. 새로 설치하는 시스템에 복구가 가능할까?

A. 가능하다.

이전 서버와 문자셋 정보(sp_helpsort 명령으로 보실수 있다.)가 같고 이전에 사용하던 SQL 서버가 정상적인 상태로 종료 되었을때(엔터프라이즈 관리자에서 SQL 서버를 스탑 시켰거나.. 정상적으로 윈도우 시스템을 종료시킨 경우 – 해당 mdf 화일이 정상적으로 닫혔을 경우) 복구가 가능하며 이때 사용하는 명령은

```
EXEC sp_attach_single_file_db 'pubs'

, 'c:₩Program Files₩Microsoft SQL Server₩MSSQL₩Data₩pubs.mdf'
```

명령으로 복구 할 수 있다.

Q. 이전시스템에서 사용하던 풀 백업본이 있다. 새로 시스템을 설치한 후 이 풀 백업본을 복구하려 하는데 방법이 없을까?

A. 가능하다. 약간 다른 방식으로 아래처럼 하심 된다.

1. 같은 이름의 db 를 만든다 (텅빈 db)
2. 빈 db 를 그냥 풀 백업 한다.
3. 원본서버에서 가져온 백업본으로 리스토어 한다.
4. 리스토어시 옵션의 강제로 덮어쓰기 하시면 된다.

의 순서로 진행하시면 무리없이 할 수 있다.

[SQL2011 강좌] 25. 백업과 복구 전략

이번에는 백업과 복구 전략에 대해서 설명 하겠다.

```
USE master

GO

--백업을테스트할 DB 생성

CREATE DATABASE bkupTest
```

먼저 테스트할 데이터베이스를 기본 옵션으로 생성한다. 간단히 데이터베이스의 정보를 보도록 한다.

```
sp_helpdb bkupTest

go
```

Results | Messages

	name	db_size	owner	dbid	created	status	compatibility_level
1	bkupTest	4,92 MB	eastluck-PC₩eastluck	8	Jun 16 2011	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	110

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	bkupTest	1	C:₩Program Files (x86)₩Microsoft SQL Server₩MSSQ...	PRIMARY	4032 KB	Unlimited	1024 KB	data only
2	bkupTest_log	2	C:₩Program Files (x86)₩Microsoft SQL Server₩MSSQ...	NULL	1008 KB	2147483648 KB	10%	log only

status 라는 부분을 보면? Recovery=FULL 이라는 게 있을 것이다. 기본 옵션으로 데이터베이스를 생성하면 복구 모델은 FULL 이라는 것이다. 백업의 타겟은 크게 두 가지로 나눌 수 있다.

디바이스와 파일

디바이스라는 가상의 장치 또는 테입장치 등을 연결해 백업을 하거나..파일단위로 백업을 할 수 있다. 최근의 상황을 볼 때 디스크가 매우 저렴해졌다. 디바이스 백업을 수행 시 디바이스라는 하나의 장치에 여러 백업을 두게 되므로 편할 수 있을지 모르나..독립적인 파일로 위치시키실 경우 조금 더 눈으로 쉽게 확인이 가능하므로 필자의 경우 유지관리계획에 등록시켜 파일로 생성해 두는 것을 좋아한다. 참고로 제가 말씀 드리는 디바이스는 영구 디바이스라고도 불리며 파일 단위 백업은 임시 디바이스라고도 한다. 먼저 파일로 백업하는 경우 이다.이렇게 테이블을 생성하고 데이터를 넣은 후 백업을 진행 했다.

```
--테이블을생성한다.

USE bkupTest

GO

CREATE TABLE tblBkupTest(

name varchar(10)

, age int

)

GO

INSERT INTO tblBkupTest(name, age) VALUES('코난', 20)

INSERT INTO tblBkupTest(name, age) VALUES('까까', 30)

GO

--파일로백업

SET LANGUAGE '한국어'

BACKUP DATABASE bkupTest TO DISK='C:₩bkupTest.bak' WITH INIT

GO
```

백업 후 결과는

파일 2 에서데이터베이스'bkupTest', 파일'bkupTest'에 대해 296 개의페이지를 처리했다.

파일 2 에서데이터베이스'bkupTest', 파일'bkupTest_log'에 대해 2 개의페이지를 처리했다.

BACKUP DATABASE 이(가) 298 개의 페이지를 0.437 초동안 처리했다(5.312MB/초).

식으로 결과를 보실 수 있을 것이다. – **결과 역시 중요하다. 특히 백업 수행 시각이 중요하다.**

복구는 아래와 같이 할 수 있다.

```
--테이블삭제

DROP TABLE tblBkupTest

GO

--데이터조회불가.

SELECT * FROM tblBkupTest

GO

--masterDB 사용- 해당 DB 가사용중이면복구가불가하기때문

USE master

GO

--복구

restore database bkupTest from disk ='C:₩bkupTest.bak'

WITH REPLACE

GO

USE bkupTest

GO

--데이터조회

SELECT * FROM tblBkupTest
```

복구 시 완료 메세지는 다음과 같을 것이다.

파일 1 에서데이터베이스'bkupTest', 파일'bkupTest'에 대해 296 개의페이지를 처리했다.

파일 1 에서데이터베이스'bkupTest', 파일'bkupTest_log'에 대해 6 개의페이지를 처리했다.

RESTORE DATABASE 이(가) 302 개의페이지를 0.557 초동안처리했다(4.224MB/초).

역시나 주의하실 점으로 항상 복구 완료 시각을 눈여겨 보시길 바란다.

위의 케이스는 단순히 데이터베이스를 풀 백업 했다. WITH INIT 은 해당 백업 타겟을 초기화 시키고 백업을 진행하라는 의미 이다. - 백업의 여러 옵션은 천천히 다룰 것이다.

아울러 풀 백업을 restore 구문으로 복구 했다 DROP 으로 날린 테이블이 잘 복구가 되었다.

복구하기 전에 USE master 구문으로 master DB 에서 복구를 진행한 이유는 해당하는 DB 가 사용 중이면 복구가 불가하기 때문이다. 만약 넷웍을 통해 사용자가 붙을 가능성이 있다면.. 잠시 사용자를 sp_who 명령으로 SPID 를 보신 후.. KILL <SPID 번호>로 죽이시고 수행하시면 되며 처음 SQL 서버를 시작할 때.. 시작 매개변수를 주어서

sqlservr.exe -m

형식으로 단일사용자 모드로 SQL 서버 시작을 하시고.. 복구를 하시는 것도 좋은 방법 이다.

별로 사용하지 않지만.. 영구 디바이스를 생성하고 백업하고 복구하는 것을 해 보도록 한다.

```
--영구백업디바이스생성

sp_addumpdevice 'disk', 'bkupTestDevice','c:₩bkupTestDevice_full_bkup'

--디바이스로백업할때는?

BACKUP DATABASE bkupTest TO bkupTestDevice
```

'Disk' 장치를 추가했다. 라는 결과와 함께 디바이스가 추가 된걸 아실 것이다. 아울러.. 디바이스에 백업 완료 시 메세지는..

파일 3 에서데이터베이스'bkupTest', 파일'bkupTest'에 대해 296 개의페이지를 처리했다.
파일 3 에서데이터베이스'bkupTest', 파일'bkupTest_log'에 대해 2 개의페이지를 처리했다.
BACKUP DATABASE 이(가) 298 개의페이지를 0.444 초동안 처리했다(5.227MB/초).

식일 것이다.

```
--테이블삭제

DROP TABLE tblBkupTest

GO

--데이터조회불가.

SELECT * FROM tblBkupTest

GO

--masterDB 사용- 해당 DB 가사용중이면복구가불가하기때문

USE master

GO

--복구

restore database bkupTest from bkupTestDevice

WITH REPLACE

GO

USE bkupTest

GO

--데이터조회

SELECT * FROM tblBkupTest
```

복구 시 메세지는

파일 1 에서데이터베이스'bkupTest', 파일'bkupTest'에 대해 296 개의페이지를 처리했다.

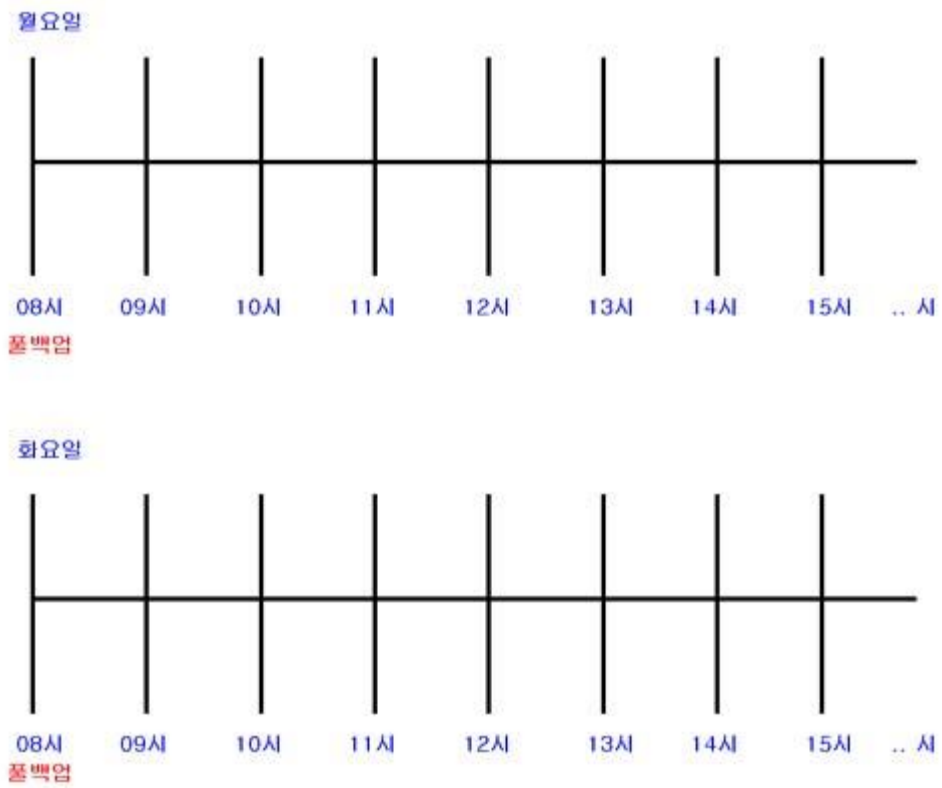
파일 1 에서데이터베이스'bkupTest', 파일'bkupTest_log'에 대해 6 개의페이지를 처리했다.

RESTORE DATABASE 이(가) 302 개의페이지를 0.491 초동안 처리했다(4.792MB/초).

식일 것이다. 아울러 데이터를 조회해 보면 잘 된 것을 알 것이다.

많은 분들이 여기까지는 쉽게 이해하시고 잘 하시는데 다음에 보시는 로그 백업에 대해서 많이 이해를 못하는 편이다. 전략적인 측면에서.. 전체 / 로그 / 차등 백업을 이해해 보도록 한다.

1. 풀백업만을 진행할 경우



먼저 이러한 타임라인이 있다고 생각해 본다. 주 단위이며.. 위는 월요일 / 화요일 / ... 식으로 나올 것이다.

두 번째로. 시각은 08 시부터 회사의 업무 시간인 6 시까지 있을 것이다. 이럴 때 풀백업 만을 진행할 경우 이다. 풀 백업은 매일 아침 08 시 업무가 시작되기 전에 이루어 집니다. 매일매일 하루에 한번씩 08 시에 풀 백업을 받는 것이다. 이렇게 운영하시는 경우가 많은데 문제 상황을 생각해 보면.

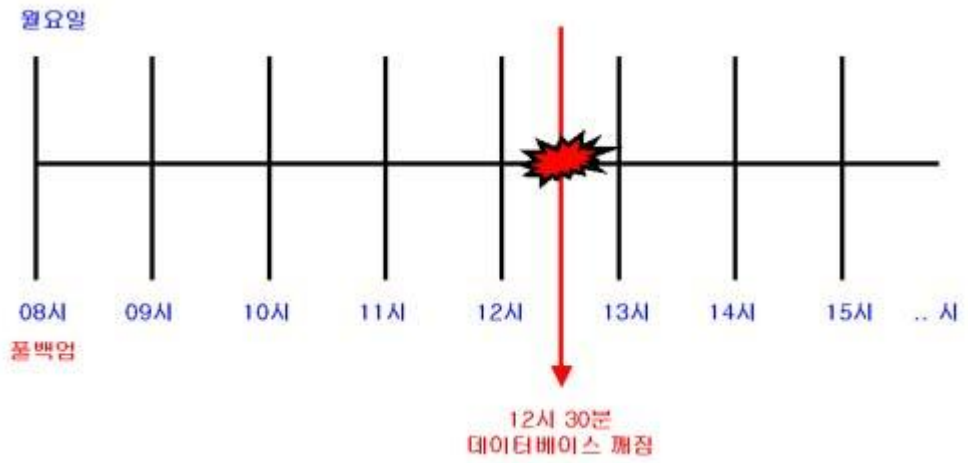
1. 로그를 백업하지 않으므로 로그가 계속 불어나는 상황이 발생할 수 있다.

로그는 데이터의 변경본 이다. 백업하면 지워지지만 지우지 않으므로.. 데이터는

50 메가인데.. 로그는 1 기가 이런 사태가 발생하게 된다.

2. 문제 발생시 복구 가능한 데이터가 가장 적다.

실제 문제 상황을 생각해 본다.

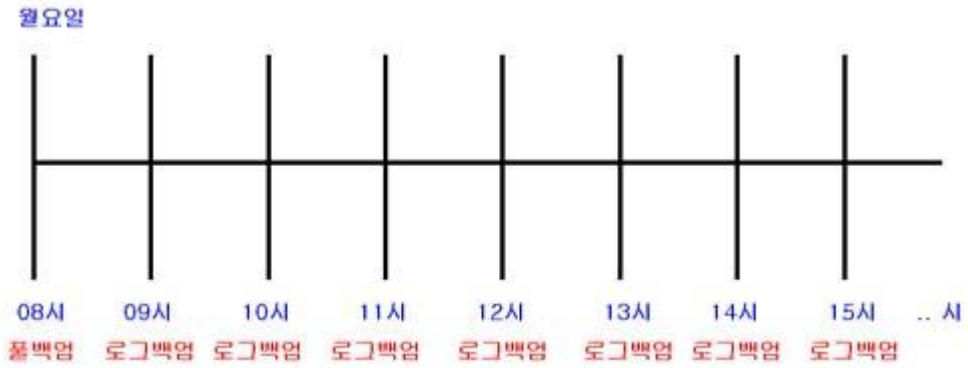


자 이렇게 12 시 30 분에 데이터베이스가 깨졌다고 가정해 본다.

언제까지의 데이터로 복구가 가능할까? 최근의 백업본이 월요일 08 시이기 때문에 월요일 08 시의 데이터로만 복구가 가능한 것이다. - 물론 한번도 백업한 적이 없다면? 복구는 불가능 하다.

08 시부터 4 시간 30 분간의 데이터는 복구가 불가해 지는 것이다. 아득하다.

2. 풀백업 + 로그 백업

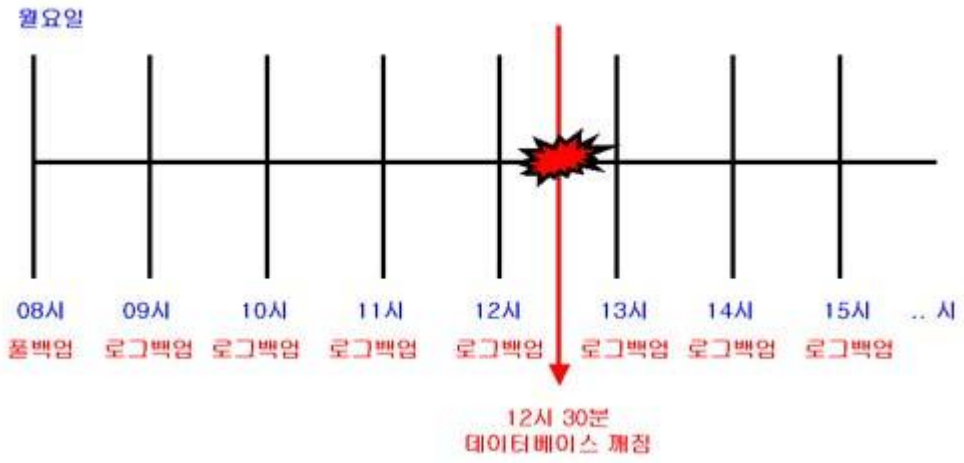


이러한 식으로.. 매일 아침 08 시 풀 백업.. 매시간마다 로그를 백업하는 것이다. 그럼 어떨까? - 이때의 제약 사항으로 SQL 서버의 해당 DB 는 반드시 단순 모델이 아니어야 한다. SQL 서버의 데이터베이스가 단순 모델이면? 로그 백업이 불가해 집니다. 이때는 전체 모델이나.. 또는 대량 로그(Bulk Load)모드 이어야 한다.

"저렇게 한 시간에 한번씩 로그를 백업하는 것도 부하가 대단히 많지 않나요?"

라고 생각하실지도 모르지만.. 로그는 대단히 그 크기가 작다.

예를 들어 게시판을 생각해 보면 INSERT UPDATE DELETE 작업은 대단히 작지만 실제 내용은 수천 건이 들어가 있을 수 있다. 실제 회사의 기간 데이터일 경우는 더더욱 그렇다. 이제 실제 문제 상황을 생각해 보도록 한다.



이럴 경우는 어디까지 복구가 가능할까?

08시 데이터

성명	나이
코난	16
까까	17
asdf	17
sgpoong	18

09시 데이터

성명	나이	
코난	16	
까까	17	
asdf	17	
sgpoong	99	<-데이터 수정

10시 데이터

성명	나이	
		<-데이터 삭제
까까	17	
asdf	17	
sgpoong	99	

11시 데이터

성명	나이	
까까	17	
asdf	17	
sgpoong	99	
crcap	18	<-데이터 삽입

12시 데이터

성명	나이	
까까	17	
asdf	17	
sgpoong	99	
crcap	18	
LeeHack	15	<-데이터 삽입

12시 20분 데이터

성명	나이	
		<-데이터 삭제
asdf	17	
sgpoong	99	
crcap	18	
LeeHack	15	

이렇게 보는 바와 같이.. 08 시부터의 데이터가 저렇다고 생각해 보도록 한다.

08 시에 풀 백업본이 있었지요? 풀 백업본을 리스토어 하면? 데이터는 어떻게 될까?

08시 데이터

성명	나이
코난	16
까까	17
asdf	17
sgpoong	18

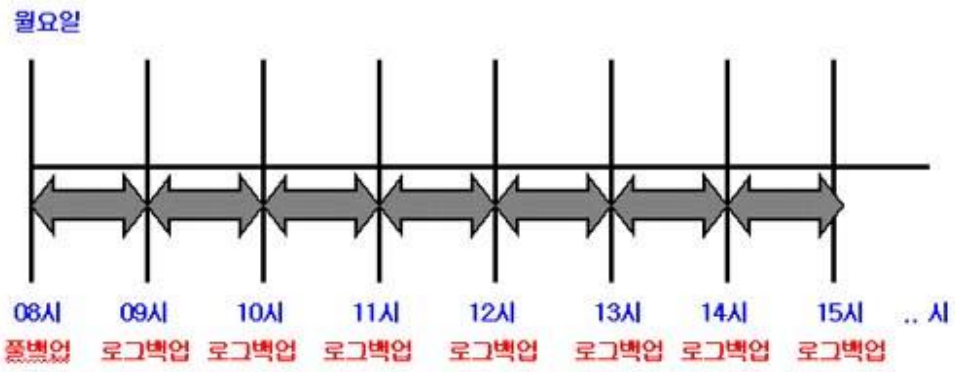
이렇게 하면 위와 같이 데이터가 나타나게 된다. 하지만 그렇다고 해도 복구 된 것은 아니다. 09 시의 로그 백업본을 리스토어 하면 그 때 완전히 복원되게 된다.

12시 데이터

성명	나이	
까까	17	
asdf	17	
sgpoong	99	
crcap	18	
LeeHack	15	<-데이터 삽입

즉 로그는 데이터의 변경이므로.. 데이터의 변경을 그대로 적용시켜 나가면 데이터가 나오게 된다. 이작업을 REDO 라고 말하기도 한다.

여기서 문제 이다. 9 시, 10 시, 11 시, 12 시까지의 로그를 잘 리스토어 했다.

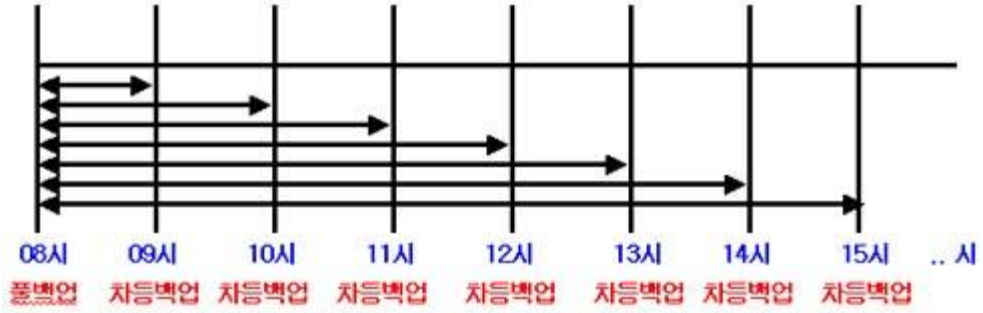


그렇지만.. 이후 12 시 20 분의 데이터는 로그를 백업하지 못했을 경우 복구가 불가해 진다.

단 SQL2000 의 전체 복구 모델일 경우는 12 시 20 분뿐 아니라... 12 시 29 분 까지 복구가 가능하다. 만약 데이터 파일만 깨지고 로그가 깨진 것이 아니라면? 로그를 데이터베이스가 깨진 후 백업할 수 있다.

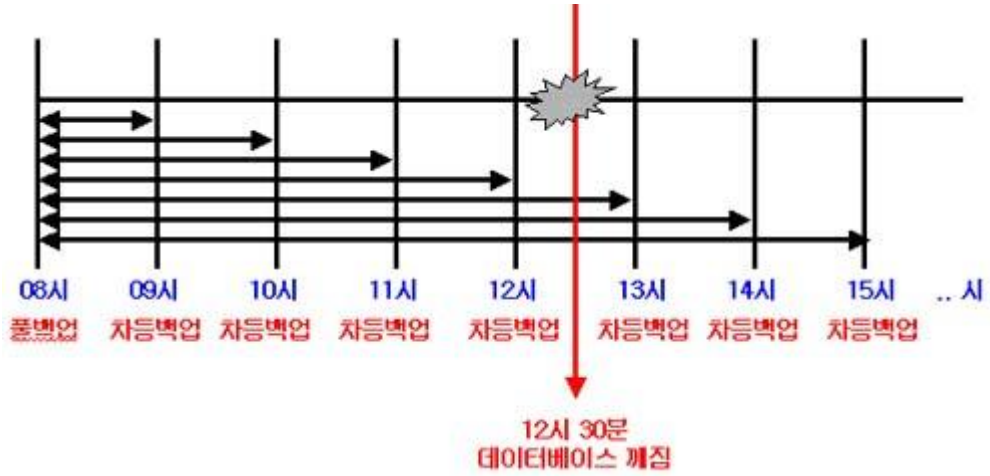
이 로그를 복구하면? 12 시 29 분까지의 데이터를 복구할 수 있게 되는 것이다.

3. 차등 백업



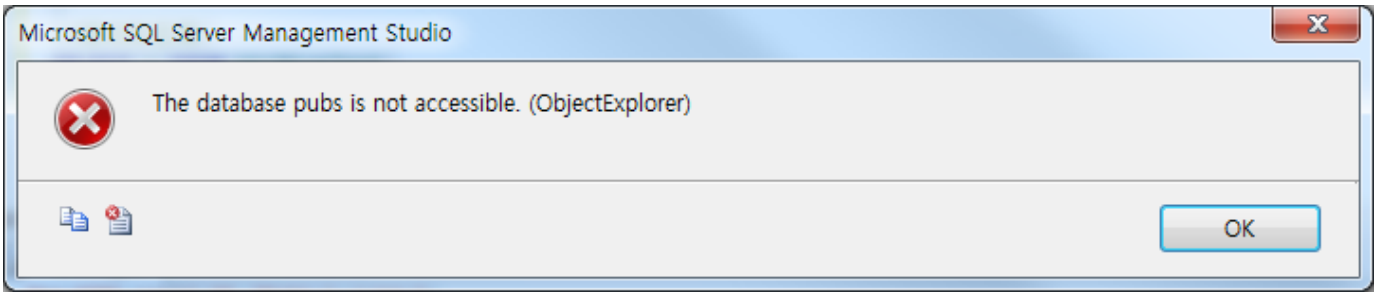
로그 백업의 범위는 이렇다. 정확히 마지막 로그 백업 이후의 로그 데이터를 백업하게 된다. 따라서 09 시의 로그 백업은? 08 시~09 시 까지의 데이터 변경만을 백업하게 되며.. 10 시의 로그 백업본은 09 시~10 시 사이의 데이터 변경만을.. 식으로 백업을 하게 된다.

하지만 차등 백업은?



이러한 방식으로.. 가장 최근의 풀백업본부터 전체 변경을 포함하게 된다.

즉! 09 시의 차등 백업본은? 08 시~09 시 까지의 데이터 변경을 가지게 되며 12 시의 차등 백업본은? 08 시~12 시 까지의 데이터 변경을 가지게 된다. 로그를 중복해서 가지게 된다는 의미 이다.
하지만 중복보다도.. 어느 정도의 복구 속도를 높일 수 있으며 SQL2000 부터는 비트맵 백업 방식이라고 해서 대단히 빠르게 백업이 가능하다는 장점이 있게 되었다. 하지만 역시 개인적으로는 풀백업 + 로그 백업을 선호하며 여러 장점이 있기 때문에 더 선호한다. 차등 백업일 경우 역시나 12 시 30 분에 데이터베이스가 깨졌다고 할 경우 어떻게 복구를 해야 할까?



08 시의 풀백업본을 리스토어 한다. 이어서 12 시의 차등 백업본 하나만 복구를 하면 가능한 데이터의 복구가 된 것이다. 세 가지의 복구 방법을 차근차근 보았다.

<복구 모델>

1. 단순 복구 모델

```
USE pubs

GO

--pubsDB 의복구모델을단순으로한다.

ALTER DATABASE pubs SET RECOVERY simple

--풀백업수행

BACKUP DATABASE pubs TO DISK = 'c:₩pubsFull' WITH INIT

--데이터조회

SELECT TOP 1 title_id, price FROM titles

--조회된값은? - 꼭적으세요.

--데이터수정

UPDATE titles SET price = price * 2

--데이터조회- 조회된값꼭적으세요.

SELECT TOP 1 title_id, price FROM titles

--복구모델이 단순일 경우 로그백업이 가능한가?

BACKUP LOG pubs TO DISK = 'c:₩pubsLog' WITH INIT

--불가능 하다.

Msg 4208, Level 16, State 1, Line 1

복구모델이단순이면문 BACKUP LOG 을(를) 사용할수없다. BACKUP
DATABASE 를사용하거나 ALTER DATABASE 를사용하여복구모델을변경하십시오.
```

```
Msg 3013, Level 16, State 1, Line 1

BACKUP LOG 이(가) 비정상적으로종료된다.

--SQL 서버시스템종료 SHUTDOWN

--탐색기등으로 pubs 의데이터파일인

--mdf 파일을지우자.

--C:₩Program Files₩Microsoft SQL Server₩MSSQL₩Data

--이기본경로이다.

--다시 SQL 서버를시작한다.
```

SQL 서버를 시작하면 당연히 문제가 있다. PubsDB 를 로드할 수 없다고 나온다. pubs - 주의대상 - 바로 공포의 서스펙트 모드라고 하는 혼수상태 이다. 이렇게 데이터베이스 이름이 나오는 이유는?
데이터베이스 이름과 같은 정보는 바로 MasterDB 에 있기 때문이다.

여하간 우리가 mdf 파일을 지웠기 때문이다. 이제 복구를 시도하면

```
--master DB 에서작업한다.

USE master

GO

--복구를진행하자.

RESTORE DATABASE pubs FROM DISK = 'c:₩pubsFull.bak'

GO

use pubs

go

--데이터조회
```

```
SELECT TOP 1 title_id, price FROM pubs..titles
```

이렇게 복구가 되었고 값을 볼 수 있게 된다.

BU1032 19.9900

의 값이 나왔을 것이다. 왜 그럴까? 위에서 분명히 UPDATE 를 했지만 풀 백업전에 진행했기 때문에 무효가 되었다. 이게 바로 단순 모델이다. - 물론 로그 백업 자체가 불가하다.

하지만 자동으로 로그를 비워주니 로그가 한없이 커지는 사태는 발생하지 않는다.

2. 대량 로그 모델

먼저 샘플텍스트화된 파일 을 다운로드 받는다. 압축을 푸시면 titles.txt 와 titles2.txt 파일이 있다.

아래 샘플에서 사용되니 이 두 파일을 c:₩ 에 두시길 바란다.

```
--pubs DB 를대량로그모델로바꾼다.

ALTER DATABASE pubs SET RECOVERY bulk_logged

--데이터베이스를풀백업한다.

BACKUP DATABASE pubs to DISK = 'c:₩pubsFull' WITH INIT

--데이터를조회한다.

SELECT TOP 1 title_id, price FROM pubs..titles

--조회된값은?

--가격을*2 한다.

UPDATE pubs..titles SET price = price * 2

--데이터를조회한다.

SELECT TOP 1 title_id, price FROM pubs..titles

--조회된값은?
```

```
--BULK 작업을한다.

--다운로드받은 titles.txt 파일을 c:₩에둔다. - SQL7 에서

--BULK 작업은로그에남지않았으나 SQL2000 에서는남는다.

--일반적인 INSERT, UPDATE, DELETE 는기본적으로로그에남는다.

--그리고작업한다.

BULK INSERT pubs..titles FROM 'c:₩titles.txt'

--데이터를조회한다.

SELECT top 2 title_id, price FROM pubs..titles

--Bulk INSERT 로삽입된값은 BU1001 에 39.98 이다.

--SQL7 과틀리게 sp_dboption 없이 BULK INSERT 가가능하다.

--로그를백업하자. - BULK INSERT 가로그에남을까남지않을까?

--SQL7 까지는 LOG 에남지않는작업이었다.

BACKUP LOG pubs TO DISK = 'c:₩pubsLog' WITH INIT

--다시하나의데이터를 BULK INSERT 로넣자.

--지금넣는값은 title_id 가 BU1002 이며가격은 23.9 이다.

BULK INSERT pubs..titles FROM 'c:₩titles2.txt'

--데이터를조회해보자.

SELECT top 3 title_id, price FROM pubs..titles

--조회된값은?
```

자 이 샘플에서 언급하는 조회되는 값을 잘 적어 두시길 바란다.

그래야 어디까지 복구가 가능한지 알 수 있다.

이제 복구 작업을 진행해 보도록 한다.

```
--다시 SQL 서버를시작시키자.

USE master

GO

--데이터가조회가능한가? - 불가하다.

SELECT TOP 3 title_id, price FROM pubs..titles

--데이터베이스를복구하자.

--현재풀백업본을복구하는것이다.

--이후 LOG 를복구할게더있으므로옵션으로

--WITH NORECOVERY 를붙인다.

RESTORE DATABASE pubs FROM DISK = 'c:₩pubsFull'

WITH NORECOVERY

--로그를복구하자.

--이때마지막복구본이므로

--WITH RECOVERY 옵션을준다.

RESTORE LOG pubs FROM DISK = 'c:₩pubsLog'

WITH RECOVERY

--데이터를조회해보자.

SELECT TOP 3 title_id, price FROM pubs..titles
```

결과는 BU1001, BU1002, BU1032 였을 것이다. 분명히 LOG 에 BULK INSERT 의 데이터가 남았기 때문에 BU1001 이 보이다? 그리고 두 번째 BULK INSERT 한 BU1002 값은 추후 로그를 백업하지 않았기 때문에 볼 수 없다.

따라서.. 대량로드 작업에서는 가장 최근의 로그 백업본 까지만!!! 복구가 가능할 것이다.

이제 전체 모델을 봐 보도록 한다.

3. 전체 모델

```
--사용된데이터베이스를최초상태로복구한다.

--맨처음단순모델할때사용했던백업본이다.

RESTORE DATABASE pubs FROM DISK = 'c:₩pubsFull'

--다음풀모델을위해풀백업을진행한다.

BACKUP DATABASE pubs TO DISK = 'c:₩pubs1' WITH INIT

--데이터를조회해보면?

SELECT TOP 1 title_id, price FROM pubs..title

--조회된값은?

--price 를*2 한다.

UPDATE pubs..titles SET price = price * 2

--BULK INSERT 로데이터를넣는다. 값은 BU1001 이다.

BULK INSERT pubs..titles FROM 'c:₩titles.txt'

--데이터를조회하면?

SELECT TOP 2 title_id, price FROM pubs..titles

--조회된값은?

--로그를백업한다.

BACKUP LOG pubs TO DISK = 'c:₩pubsLog' WITH INIT
```

```
--또다시 BULK INSERT 를한다.

BULK INSERT pubs..titles FROM 'c:₩titles2.txt'

--데이터를조회하면?

SELECT TOP 3 title_id, price FROM pubs..titles

--조회된값은?

--시스템셧다운

SHUTDOWN

--pubs 의 mdf 파일을삭제한다. 단!

--ldf 파일은건드리지마세요.
```

mdf 파일을 삭제 했다. 조회된 값은 BU1001, BU1002, BU1032 일 것이다.

```
--복구를진행한다.

USE master

GO

--데이터를조회가가능한가요? - 불가한다.

SELECT TOP 2 title_id, price FROM pubs..titles

--로그를백업한다.

--바로풀백업의중요한부분으로

--문제가발생해도로그가깨지지않았다면?

--로그가백업이된다.

--이때반드시 WITH NO_TRUNCATE 옵션이필요한다.

--INIT 는초기화시켜백업하라는옵션이다.
```

```
BACKUP LOG pubs TO DISK = 'c:₩pubsLogNoTrunc' WITH NO_TRUNCATE, INIT

--로그백업이성공한다.

--데이터베이스를복구해보도록한다.

--복구할게더있으니 NORECOVERY 로한다.

RESTORE DATABASE pubs FROM DISK = 'c:₩pubsFull'

WITH NORECOVERY

--로그를복구해보도록한다.

--복구할게더있으니 NORECOVERY 로한다.

RESTORE LOG pubs FROM DISK = 'c:₩pubsLog'

WITH NORECOVERY

--로그를복구해보도록한다.

--이로그는? pubs 가깨진후로그만백업한것이죠?

--위에서옵션인 WITH NO_TRUNCATE 옵션으로백업한로그이다.

--복구할게더이상없으니 RECOVERY 로한다.

RESTORE LOG pubs FROM DISK = 'c:₩pubsLogNoTrunc'

--데이터를조회해보면?

SELECT TOP 3 title_id, price FROM pubs..titles

--어디까지복구가되었나요?
```

4. STOPAT 을 이용한 복구

STOPAT 를 사용하려면 반드시 모델이 전체 모델 또는 대량 로그 이어야 한다.

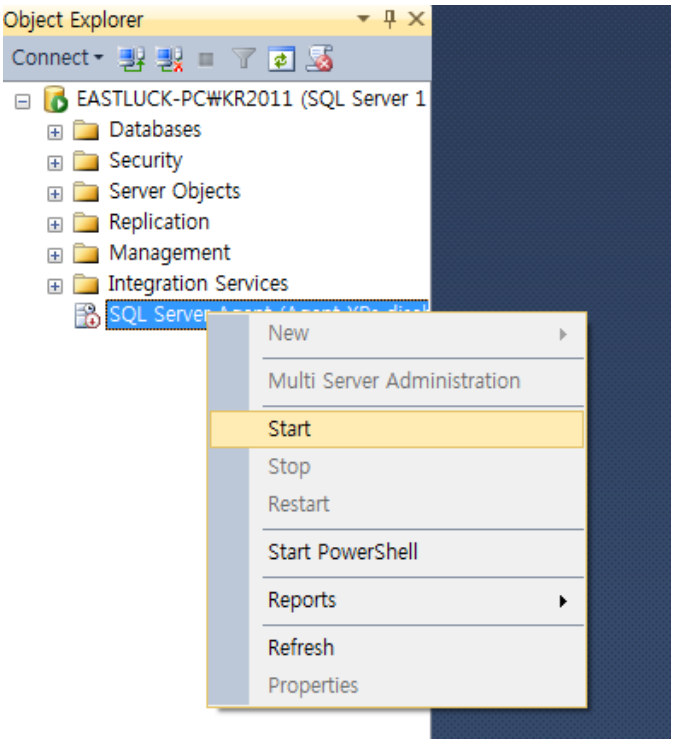
```
--데이터베이스를초기화시키기위해리스토어한다.

RESTORE DATABASE pubs FROM DISK = 'c:₩pubsFull'

--데이터베이스를풀백업한다.

BACKUP DATABASE pubs TO DISK = 'c:₩pubs1' WITH INIT

--데이터를조회한다.

SELECT TOP 1 title_id, price FROM pubs..titles

--초기화가잘되었는지확인하는것이다.

--데이터를하나삽입한다.

BULK INSERT pubs..titles FROM 'c:₩titles.txt'

--데이터를조회하면?

SELECT TOP 2 title_id, price FROM pubs..titles

--조회된값은?

--로그를백업해보도록한다.

BACKUP LOG pubs TO DISK = 'c:₩pubsLog' WITH INIT

--역시나데이터를하나더넣도록한다.

BULK INSERT pubs..titles FROM 'c:₩titles2.txt'

--데이터를조회하면?

SELECT TOP 3 title_id, price FROM pubs..titles

--조회된값은?

--!!자이제문제상황이다.

--개발자의실수로 UPDATE 를 WHERE 절없이쳐버렸다.
```

```
UPDATE pubs..titles SET price = 0

--컨트롤+ Z 는아무리눌러도데이터는복구안된다.

SELECT title_id, price FROM pubs..titles

--바로문제가발생한시각을알아야만한다.

SELECT GETDATE()

--결과시각은?

--코난이의경우아래와같은시각이다.

2011-06-17 00:00:13

--어떻게해야할까?

--1. 즉시로그를백업한다.

--2. STOP AT 구문으로복구한다.

--로그를백업한다.

BACKUP LOG pubs TO DISK = 'c:₩pubsLogNoTrunc' WITH NO_TRUNCATE, INIT

--데이터베이스를복구한다.

--이때첫번째로그백업과.. 문제가발생한후로그백업사이로

--복구를진행해야한다. 그렇지요?

--STOP AT 을적절히사용하도록한다.

--풀백업본을리스토어한다.

RESTORE DATABASE pubs FROM DISK = 'c:₩pubsFull'

WITH NORECOVERY

--첫번째로그백업본을복구한다.
```

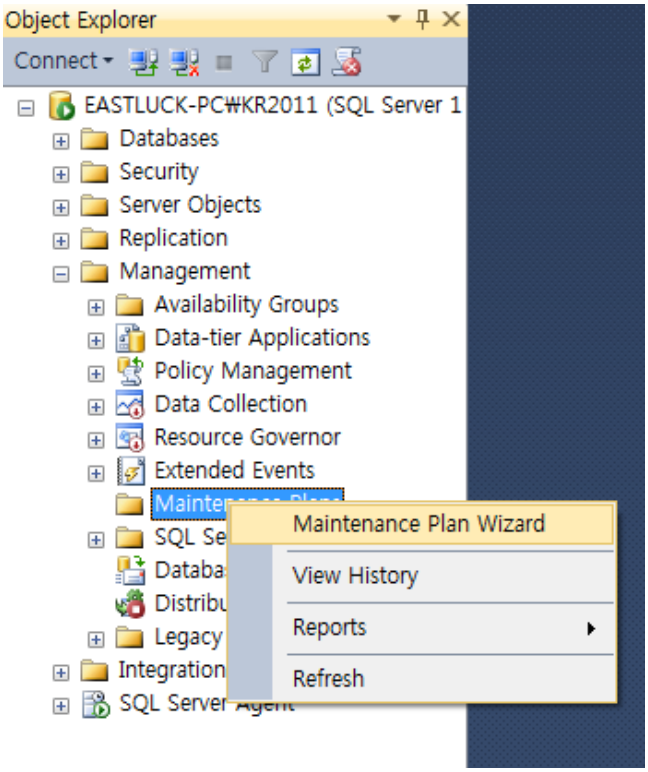
```
RESTORE LOG pubs FROM DISK = 'c:₩pubsLog'

WITH NORECOVERY

--문제가발생한후로그를백업한것이다.

--약간특이하게 STOP AT 이있다.

--시각지정은.. 문제가발생한시각- 1 분정도로하겠다.

--2001-08-23 16:06:14.320 빼기 1 분

--그래야만.. 문제가발생하기바로전시각으로되돌릴수있겠지요?

--당연히문제가발생한후즉시알아야만데이터손실을그만큼줄일수있겠죠

--한일주일정도지난후문제가발생한걸알고.. 또한

--중요한회원데이터였다면? 생각하기도끔찍해지겠지요.

RESTORE LOG pubs FROM DISK = 'c:₩pubsLogNoTrunc'

WITH STOPAT = '2011-06-17 00:00:13.320', RECOVERY

--데이터를조회해보면?

SELECT TOP 3 title_id, price FROM pubs..titles

--원하는데이터인 BU1001, BU1002, BU1032 가맞습니까?
```

데이터베이스 유지관리 계획은.. 총체적으로 데이터베이스를 유지 관리하기 위한 기능이다.
백업 / 인덱스 구축 등등의 관리자의 손이 개입되어야 하는 작업들을 자동적으로 수행할 수 있게 도움을 주는 매우 유용한 기능이다.

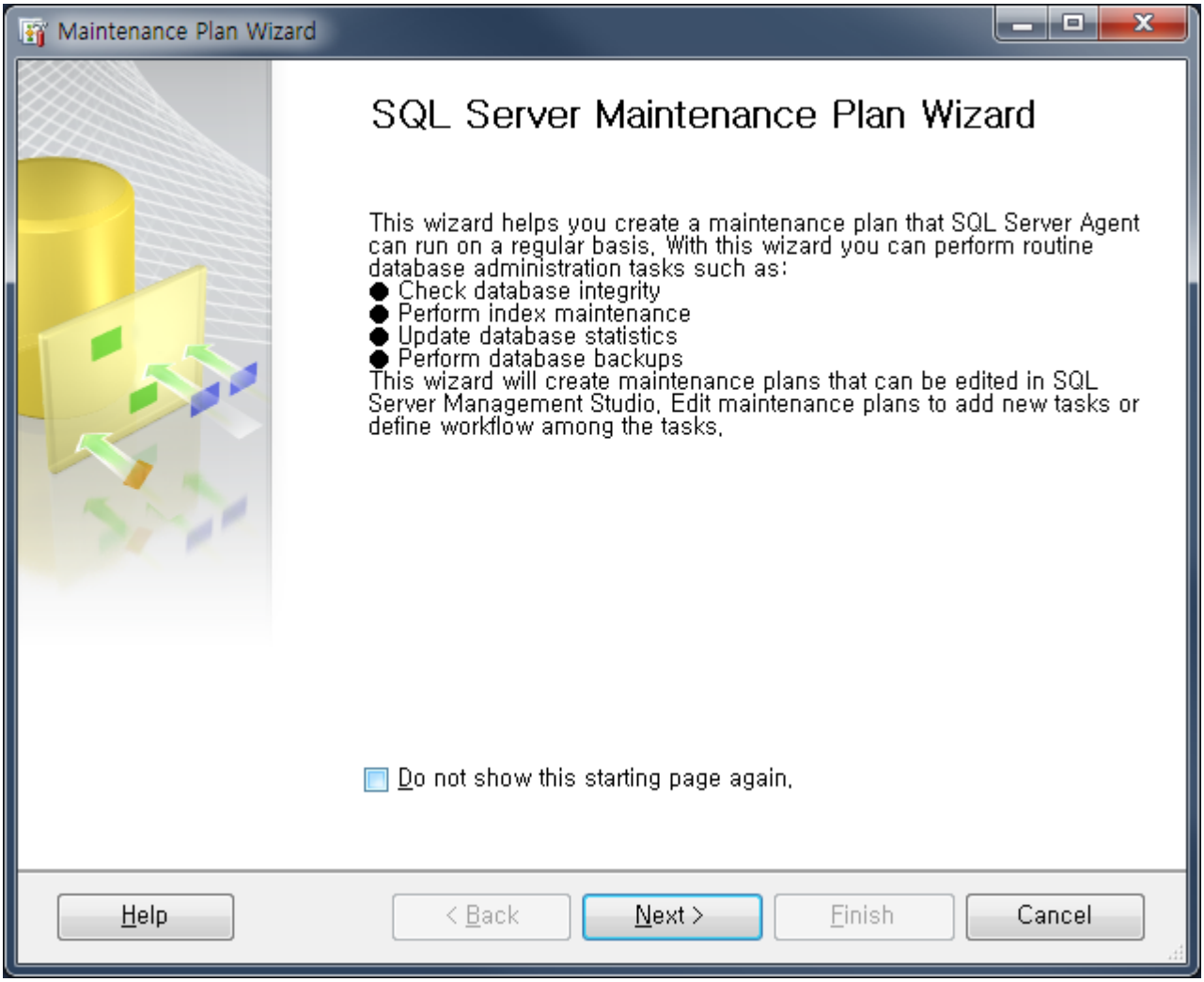
우선 SQL Server Agent 가 중지된 경우에는 아래와 같은 방법으로 실행한다.

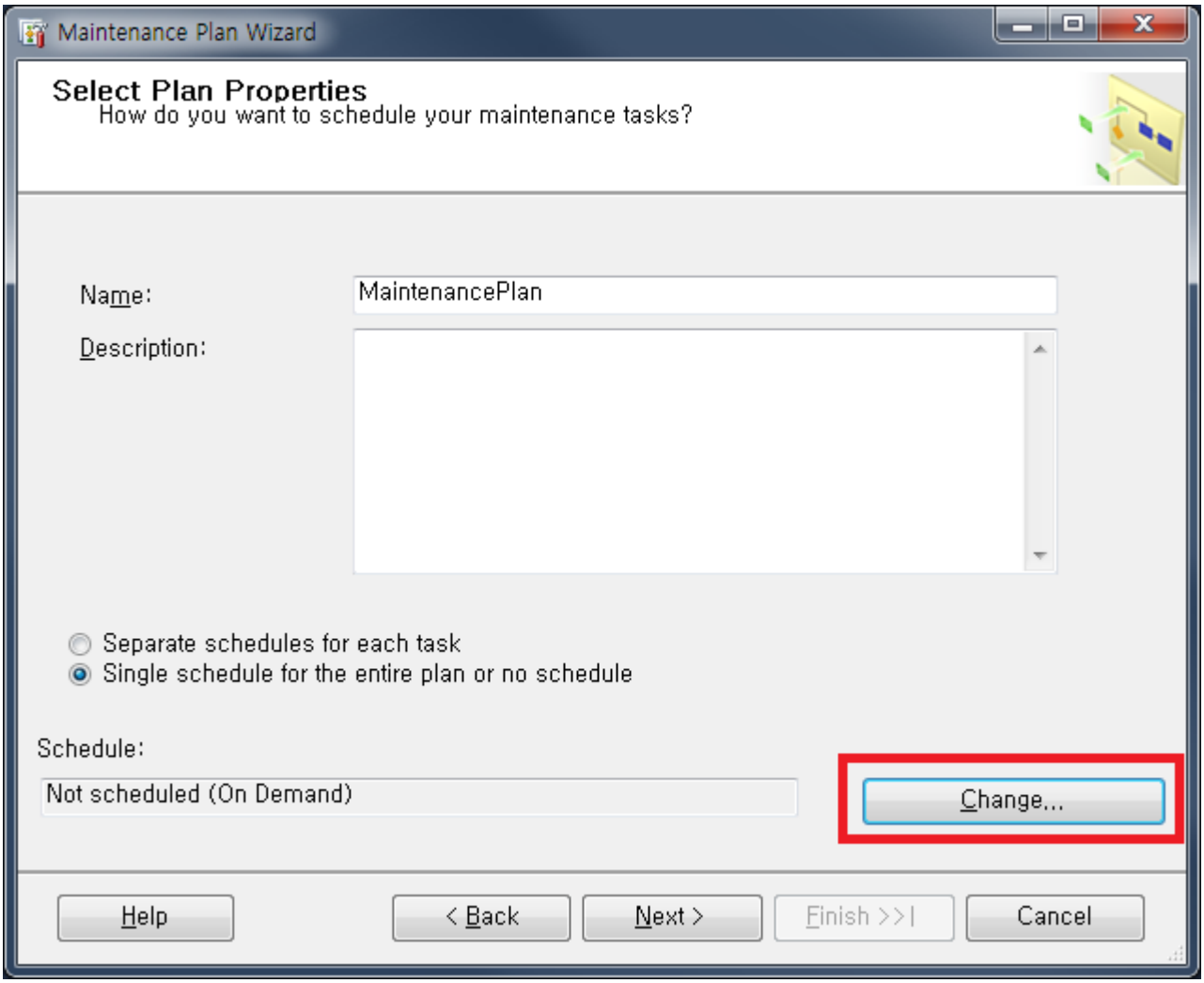


SQL Server Agent 가 실행 중이면, 이렇게 해당 DB 에서.. Maintenance Plan Wizard 를 선택한다.

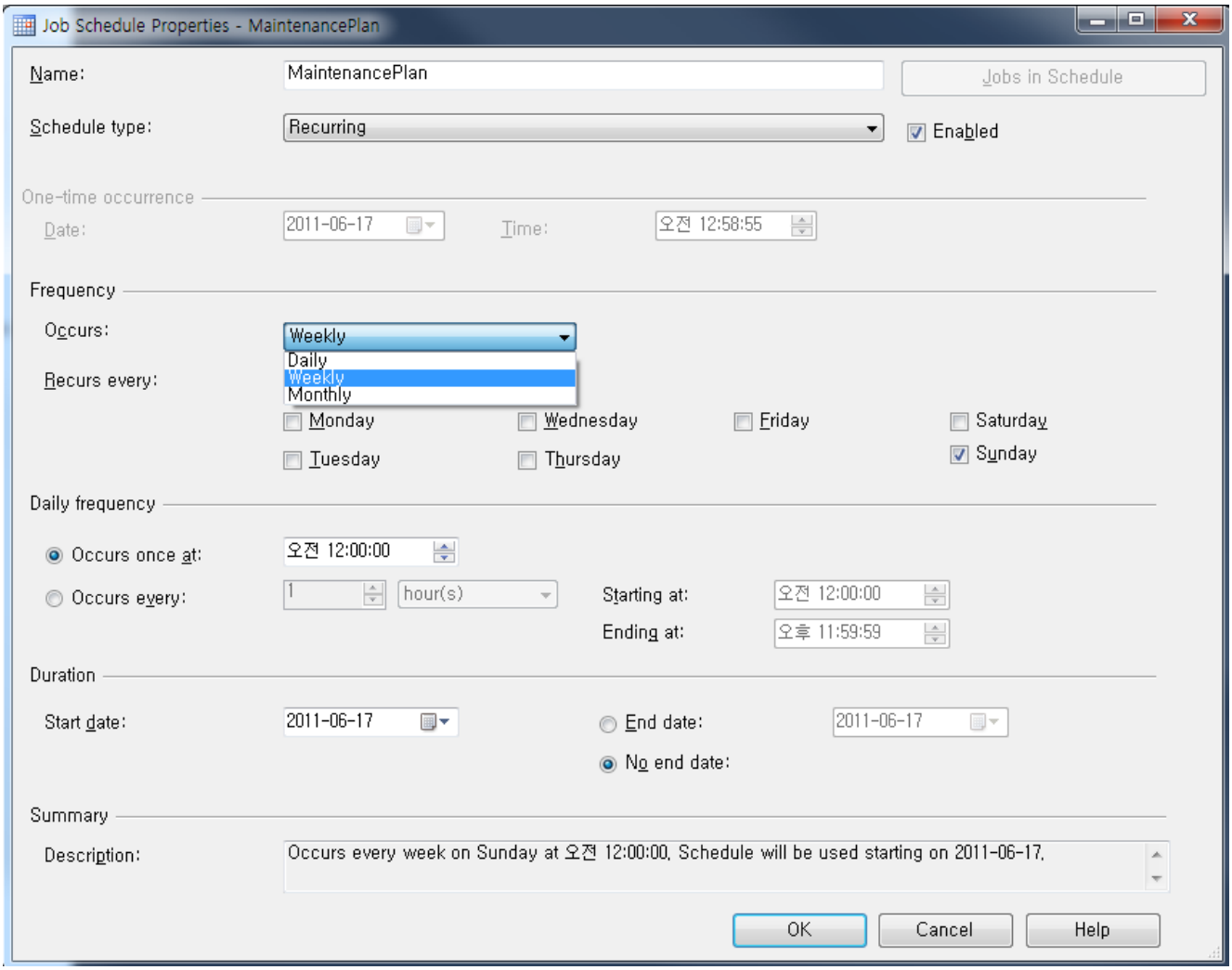


Maintenance Plan Wizard 를 실행하면 아래와 같은 그림이 나오는데 다음을 선택한다



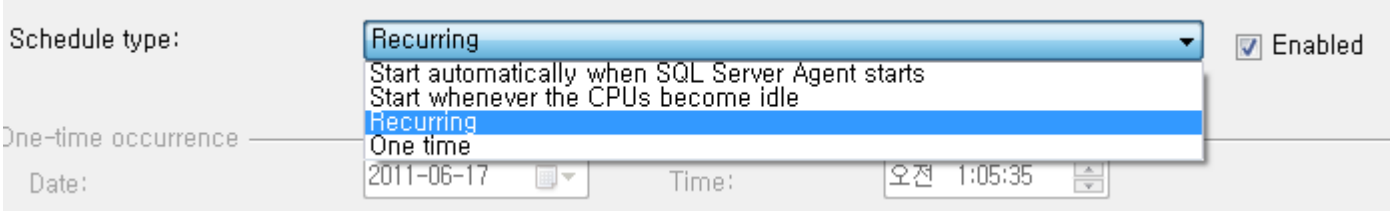


여기서는 각각의 작업마다 스케줄을 나눠서 셋팅 하는 방법과 하나의 스케줄로 작업을 만드는 방법이 있다. 일단 저희는 아래거인 하나의 스케줄로 만들고 Change 를 눌러서 일정을 선택 한다.



일정 선택 화면 이다.

1. 스케줄 타입에는 총 4 가지가 있다.



Start automatically when SQL Server Agent starts: SQL Server Agent 가 시작했을 때 자동으로 수행

Start whenever the CPUs become idle: CPU 가 유휴상태가 될 때마다 시작

Recurring: 반복 수행

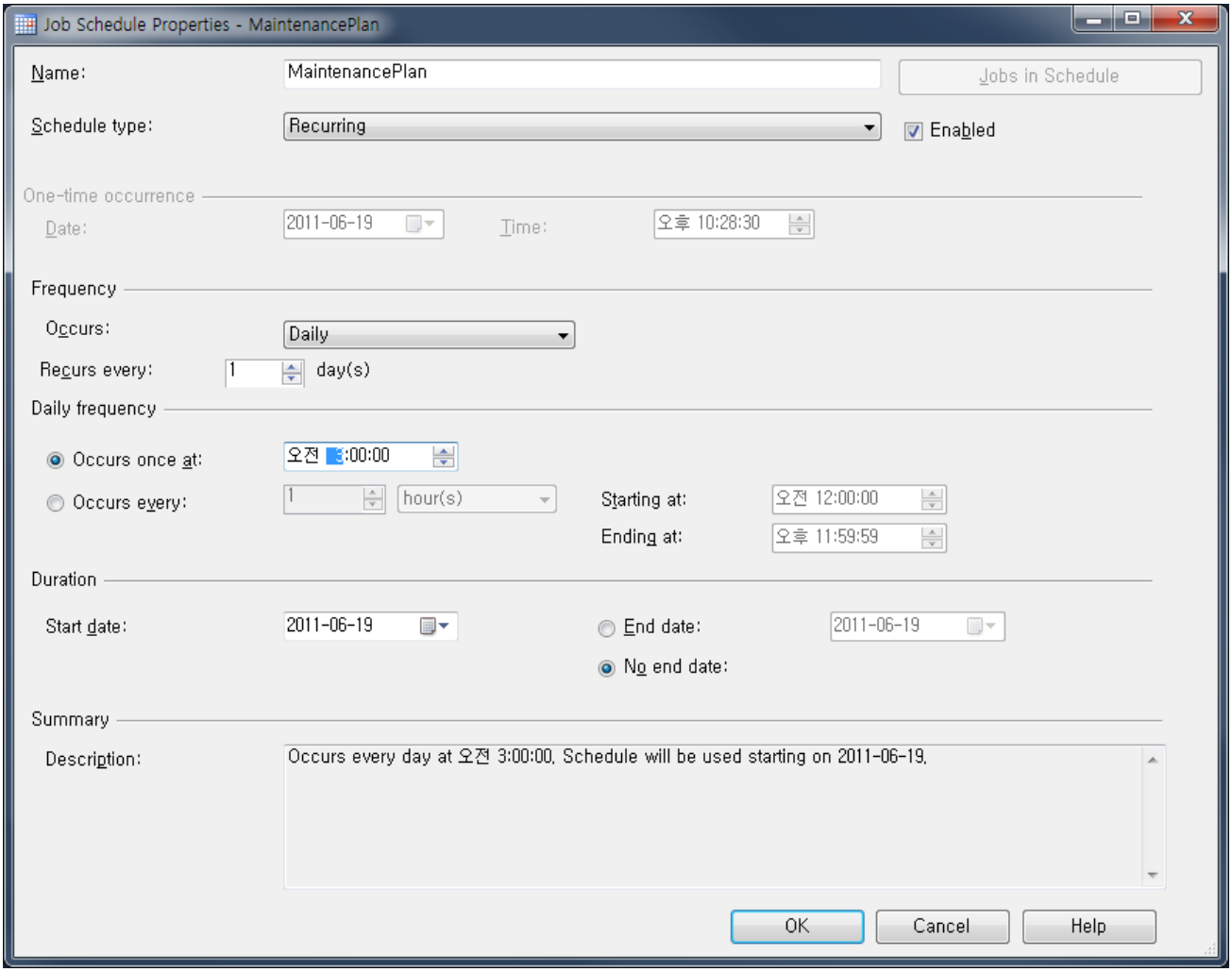
One time: 한번

2. Frequency(빈도) 는 총 3 가지가 있다. 일별, 주별, 월별이다. 빈도에 따라서 설정할 수 있는 방법이 다양하게 나타난다.

보통 풀 백업은 하루에 한번.. 트랜잭션 로그 백업은 30 분 또는 1 시간에 한번씩 백업 받는 것이 좋다. 예를 들어 새벽 3 시에 풀 백업을 받고, 1 시간 마다 트랜잭션 로그 백업을 하면서.. 오후 11 시 데이터로 복원을 해야 한다면, 복원하는데 상당히 귀찮을 수 있다. 만약 복원을 자주 해야 한다면, 차등 백업도 함께 이용하는 것도 좋은 선택이라고 생각이 된다.

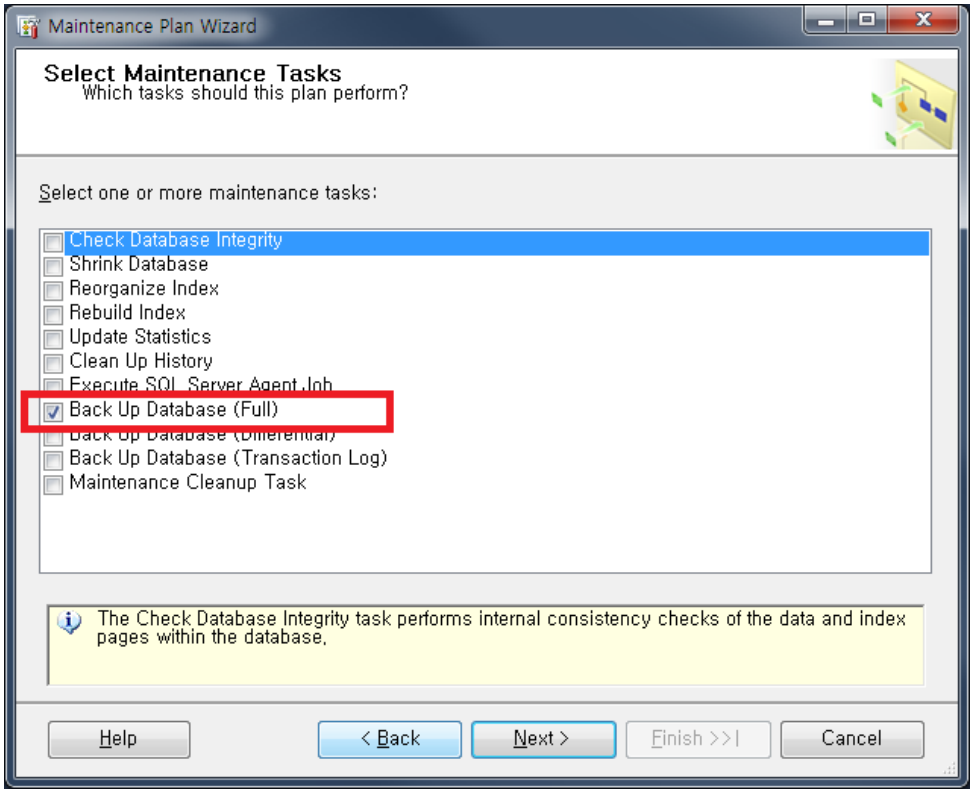
저희는 풀백업을 셋팅하는 것이기 때문에.. 스케쥴 타입은 Recurring(반복) 과 빈도는 일별을 선택하고 매일 오전 3 시에 수행되도록 셋팅 한다.

아래와 같이 설정이 된다.

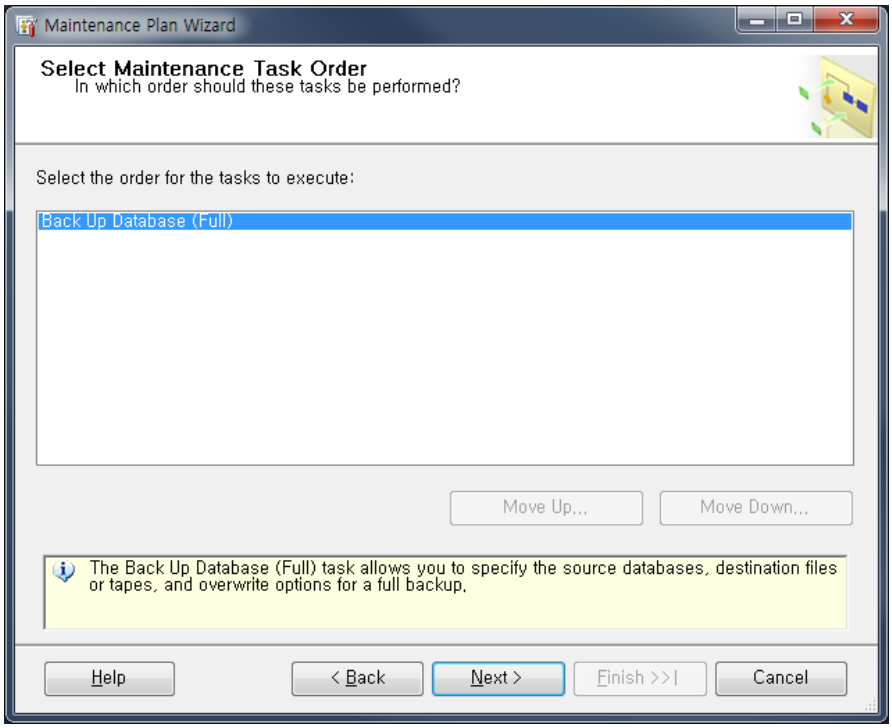


일정을 셋팅한 뒤에 OK 를 눌러서 창을 닫는다.

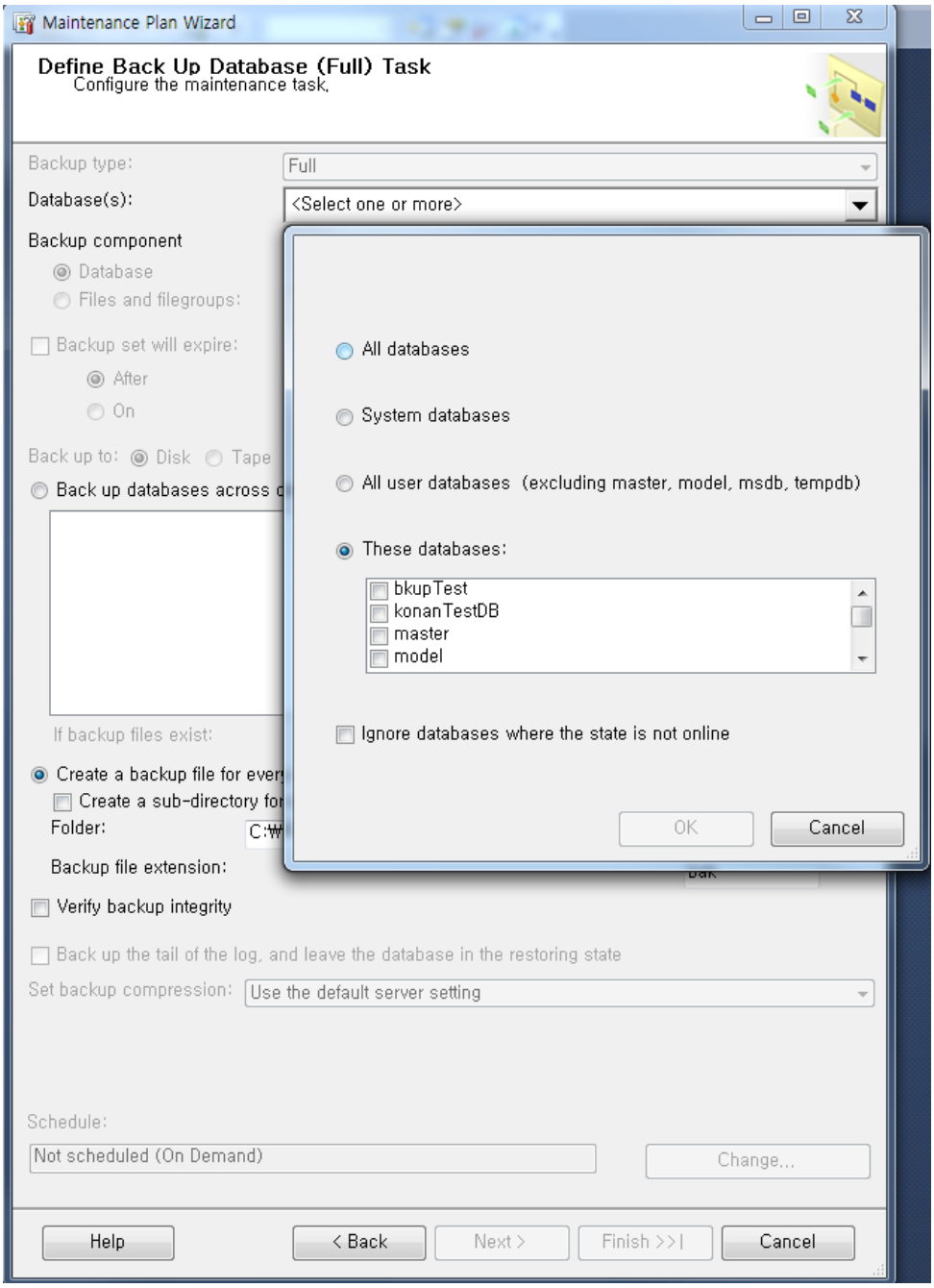
그리고 Next 를 누르면 아래와 같은 화면이 나타나게 된다.



하고자 하는 것은 풀 백업이기 때문에 Back up Database(Full)을 선택을 하고 Next 를 선택한다.



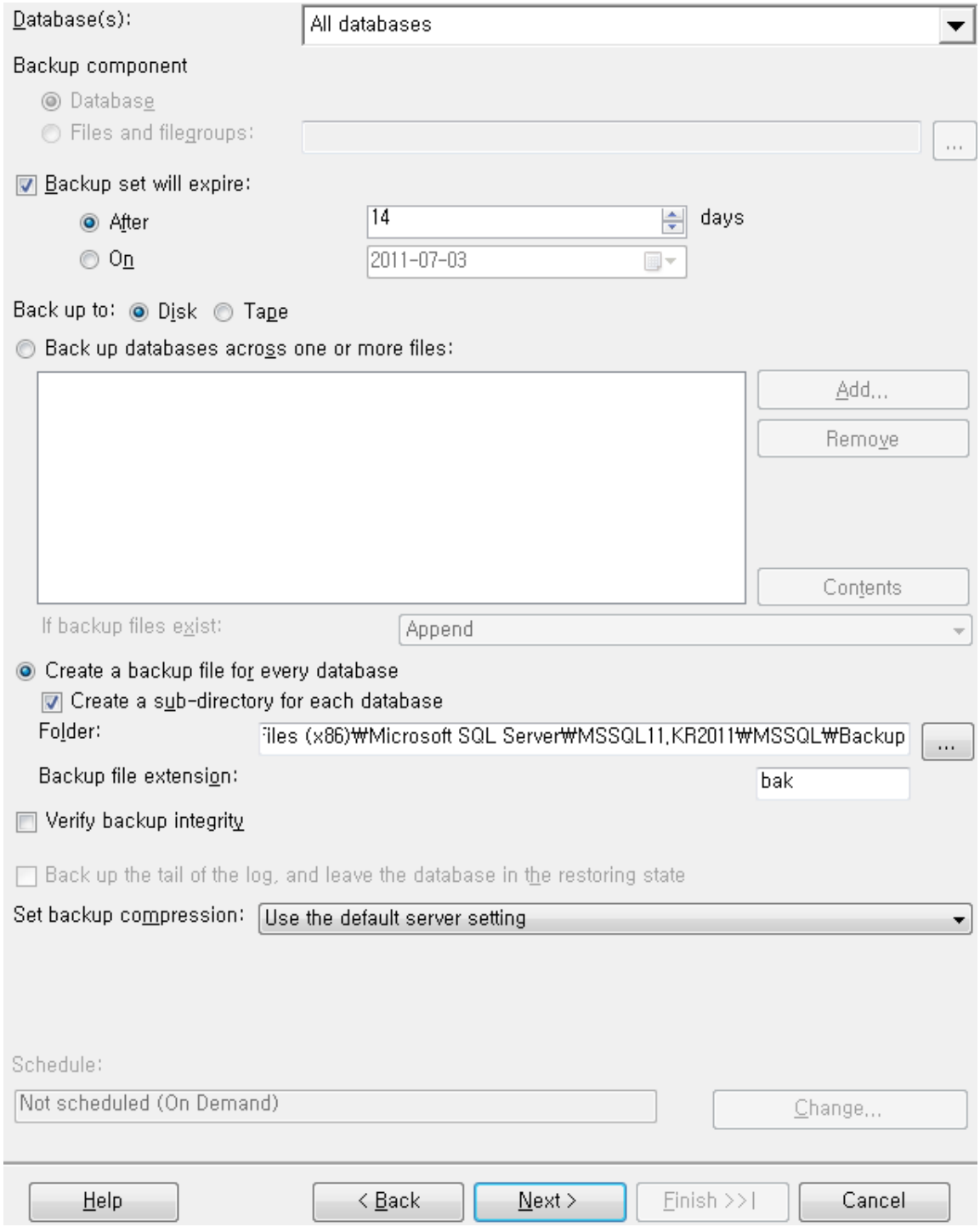
이번 화면은 실행 순서를 정하는 화면이다. 만약 이전 창에서 여러 개를 선택한 경우, 여기서 실행 순서를 결정할 수 있다. 하나 밖에 없기 때문에 바로 Next 를 선택한다.



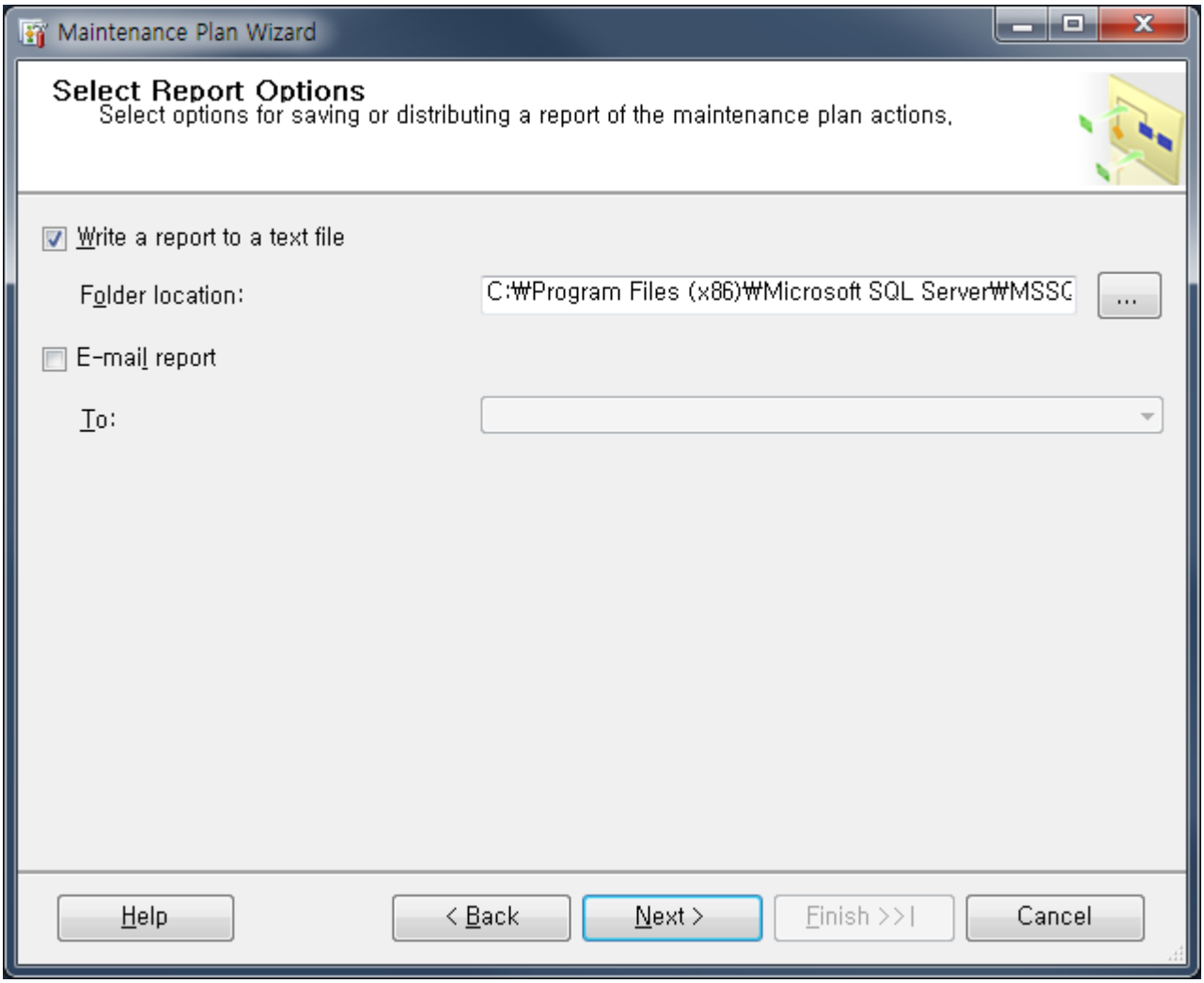
이번 화면에서 상단에 Database(s)를 누르면 내가 백업하고 싶은 Database 를 선택할 수 있다.

그리고 나서 Backup set will expire 를 통해 특정 일자 후에 자동으로 삭제할 수 있도록 설정할 수 있다.

그리고 Back up to 에 disk 그리고 Create a backup file for every database 를 선택하고, Create a sub-directory for each database(각각의 데이터베이스를 위한 서브 폴더 생성)을 체크하고 백업할 디렉토리를 선택한다.

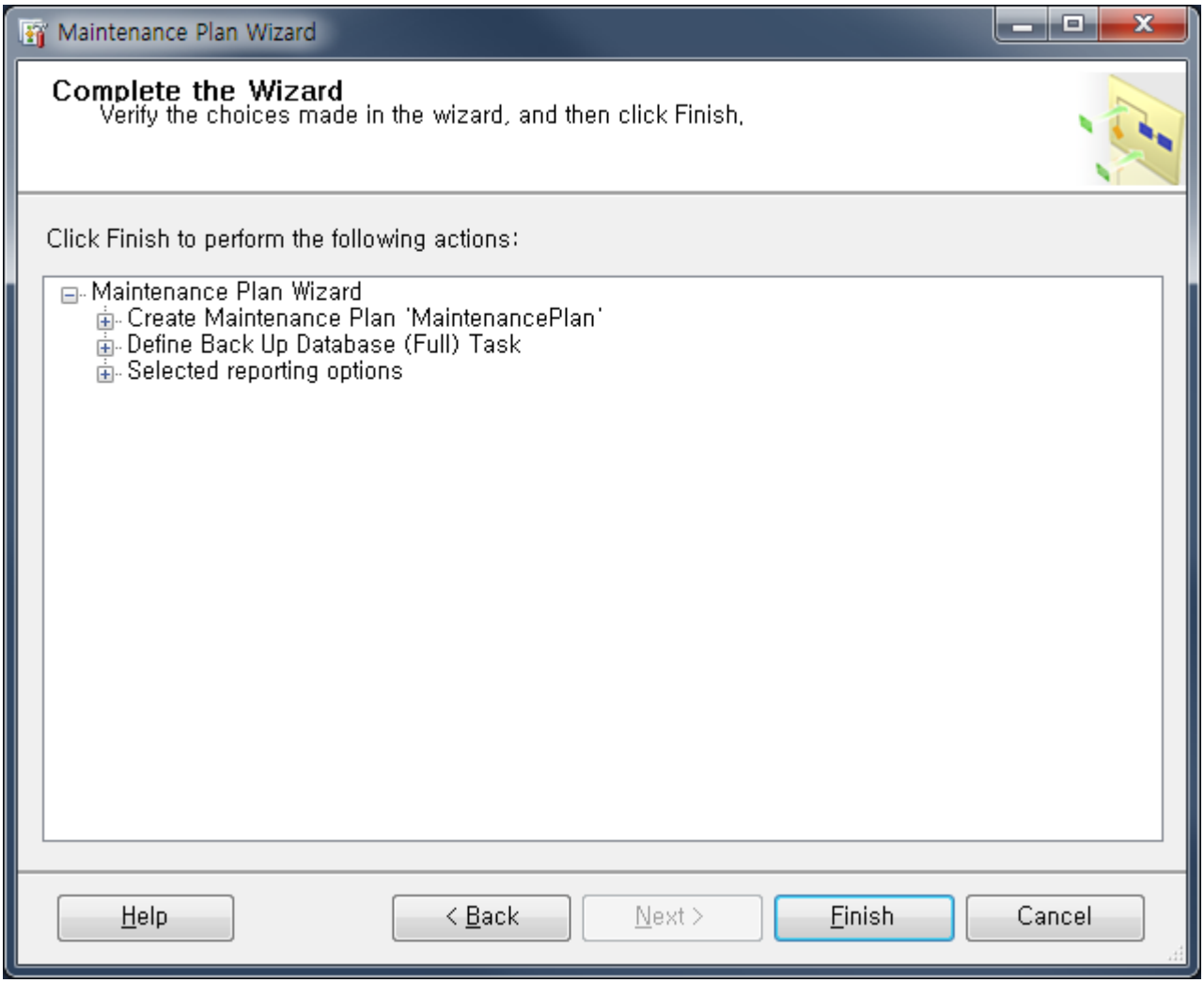


그리고 나서 Next 를 선택한다

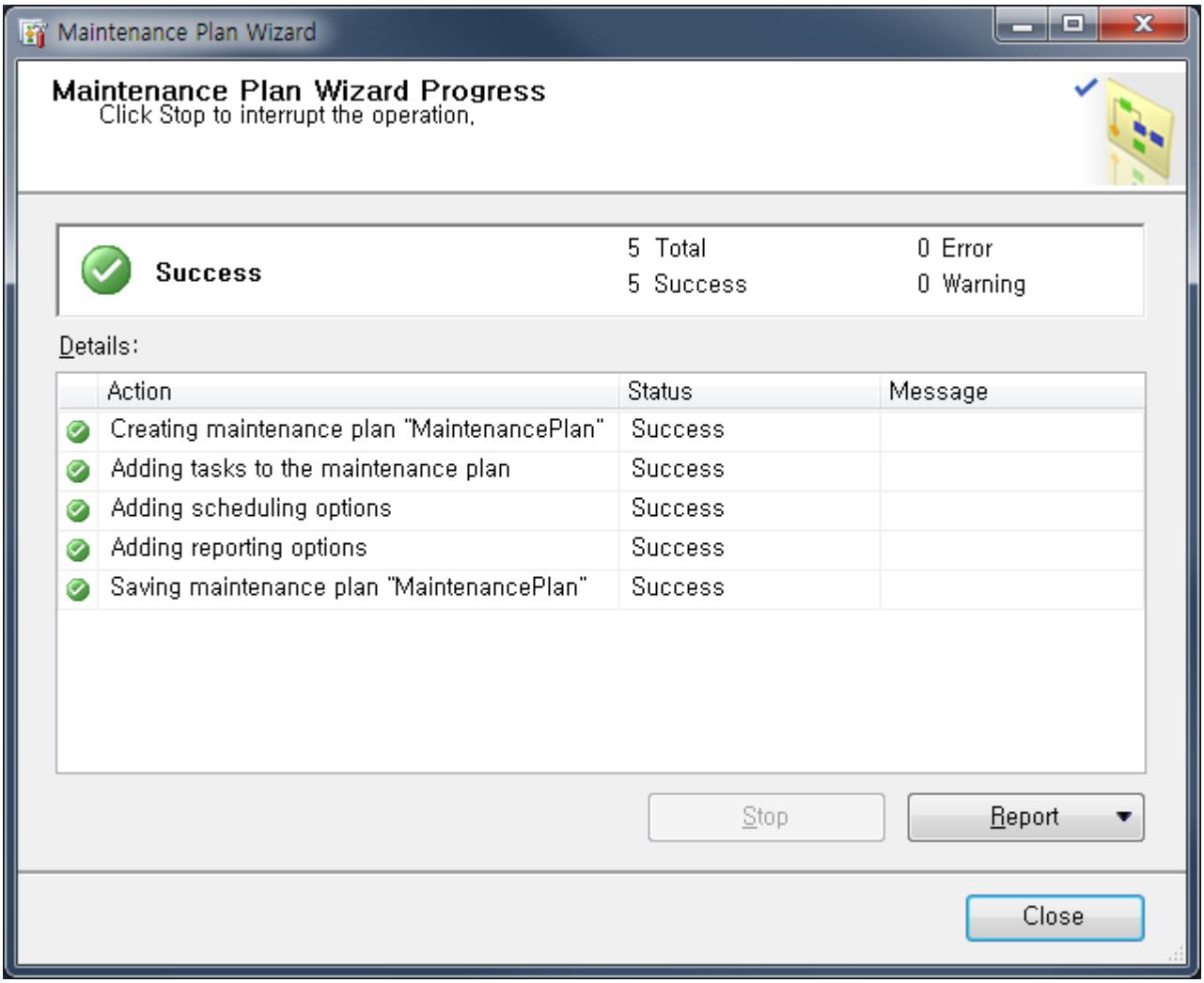


이번에는 백업된 결과를 Text 또는 이메일을 통해 받을 수 있는 부분을 셋팅 한다.

Next 를 선택한다



마지막으로 Finish 를 선택한다



위의 화면이 나오면 정상적으로 스케쥴이 생성된 것이다.

복잡하면서도 간단한 백업 셋팅이 완료되었다.

트랜잭션 로그 백업 설정도 비슷하게 할 수 있다.

[SQL2011 강좌] 26. 커서란?

커서는 SELECT 되는 특정 집합에 대해서 각 행에 대해 특정한 처리를 할 때 주로 사용된다.
예를 들어, 쌓여있는 우편물을 예로 들어보겠다.

1. 쌓여있는 우편물 중 맨 위에 것 하나를 집는다.
2. 우편번호와 주소를 확인한다.
    2-1. 우편번호가 있다면 체크 표시를 싸인펜으로(이용도 이다.) 한 후 해당 지역의 나무 박스에 넣는다.
    2-2. 우편 번호가 없다면 체크표시를 한 후 싸인펜으로(아주 유용!) 우편번호를 우편번호 책에서 찾아(대부분 머리 속에 있음) 쓴 후 나무 박스에 넣는다.
    2-3. 이 작업을 쌓여 있는 우편물이 없을 때까지 계속한다.

SQL 서버의 데이터 처리를 다시 생각해 본다..
SQL 서버는 모든 작업을 컬럼(열)기반으로 처리 한다. 작업의 처리 방식은 컬럼에 대해서 수행 된다는 의미 이다. WHERE 절은? 컬럼의 특정 값에 대해서 로우(행)을 선택할 뿐이다. 여기서 또 정신적으로 많이 불안하신 분들은 이런 생각을 하실 것이다. identity 로 컬럼을 잡은후
1. identity 값이 가장 낮은 녀석을 SELECT 한다.
2. identity 값을 + 1 한 녀석을 SELECT 한 후 가져온다. 없다면? 또 +1 한 녀석을 가져온다.
3. Max 값까지 한다.
이렇게 생각하시는 분도 계실 것이다. 천천히 몇 번을 전체 테이블을 뒤져서 처리해야 할지 생각해 보세요.
그러면
위의 우체국의 일중에서.. 쌓여있는 우편물 = 테이블의 로우(행) 이라고 생각해 보면 우편물(로우)을 하나 가져와서 우편번호 체크(어떠한 처리작업) 후 해당하는 나무박스(테이블이나 저장소)에 넣는 작업을 우편물(로우)가 없을 때까지 계속 한다.

즉 커서는 로우를 기반으로 하는 작업이 된다. 라는 것이다. 저렇게 로우를 하나 SELECT 해서 여러 가지 처리를 한 후 어떠 어떠한 작업을 한다. 이것을 가능하게 하는 것이 바로!!! 커서 이다. 물론 SQL 구문만을 가지고 억지로 어찌어찌 한다면 가능할 것이다. 하지만 커서를 이용하시면 훨씬 빠르고 간단히 저러한 처리를 가능하게 할 수 있다.

커서는

1. 커서 선언(Declare)

2. 커서 오픈(Open)

3. 데이터 행 가져오기(Fetch)

4. 커서 클로즈(Close)

5. 커서 선언 제거(Deallocate)

로 이루어 진다.

간단히 샘플을 이용해 하나의 작업을 수행해 보도록 한다. 샘플 데이터를 생성해 보도록 한다.

```
CREATE TABLE 우편물(

번호 int identity(1,1)

, 우편번호 varchar(3)

, 우편주소 varchar(6)

)

INSERT INTO 우편물(우편번호, 우편주소)

VALUES('001', '코난동')

INSERT INTO 우편물(우편번호, 우편주소)

VALUES('002', '악마동')

INSERT INTO 우편물(우편번호, 우편주소)

VALUES('003', '악어동')

INSERT INTO 우편물(우편번호, 우편주소)
```

```
VALUES('004', '태오동')

INSERT INTO 우편물(우편번호, 우편주소)

VALUES('', '코난동')

INSERT INTO 우편물(우편번호, 우편주소)

VALUES('', '악마동')

CREATE TABLE 우편번호(

우편번호 varchar(3)

, 우편주소 varchar(6)

)

INSERT INTO 우편번호 VALUES('001', '코난동')

INSERT INTO 우편번호 VALUES('002', '악마동')

INSERT INTO 우편번호 VALUES('003', '거북동')

INSERT INTO 우편번호 VALUES('004', '태오동')

INSERT INTO 우편번호 VALUES('005', '악어동')

SELECT * FROM 우편물

SELECT * FROM 우편번호
```

Results | Messages

	번호	우편번호	우편주소
1	1	001	코난동
2	2	002	악마동
3	3	003	악어동
4	4	004	태오동
5	5		코난동
6	6		악마동

	우편번호	우편주소
1	001	코난동
2	002	악마동
3	003	거북동
4	004	태오동
5	005	악어동

데이터를 보면? **대부분의 사람들이 우편번호를 잘못 넣거나 아예 넣지 않는다.**

먼저 악어동은 005 번인데.. 003 번으로 잘 못 넣었다.

다음으로 맨 마지막의 두 건은? 아예 우편 번호를 넣지 않았다.

이를 어떻게 처리하면 좋을까? 바로 커서를 사용해 완벽하게 마무리 지어 보면.

```
--커서선언

DECLARE cur_konan_Test CURSOR

FOR

SELECT 번호, 우편번호, 우편주소 FROM 우편물

--커서오픈

OPEN cur_konan_Test

--변수선언
```

```
DECLARE @v_번호 INT

DECLARE @v_우편번호 VARCHAR(3)

DECLARE @v_우편주소 VARCHAR(6)

--첫로우 FETCH

FETCH NEXT FROM cur_konan_Test INTO @v_번호, @v_우편번호, @v_우편주소

WHILE @@FETCH_STATUS = 0

BEGIN

--FETCH 된데이터를 tempdb 에삽입

UPDATE 우편물

SET 우편번호 = (SELECT 우편번호 FROM 우편번호 WHERE 우편주소 =
@v_우편주소)

WHERE 번호 = @v_번호

--다음로우 FEETCH - 루프

FETCH NEXT FROM cur_konan_Test INTO @v_번호, @v_우편번호, @v_우편주소

END

--커서 CLOSE

CLOSE cur_konan_Test

--커서 DEALLOCATE

DEALLOCATE cur_konan_Test

GO

SELECT * FROM 우편물
```

```
SELECT * FROM 우편번호
```

Results | Messages

	번호	우편번호	우편주소
1	1	001	코난동
2	2	002	악마동
3	3	005	악어동
4	4	004	태오동
5	5	001	코난동
6	6	002	악마동

	우편번호	우편주소
1	001	코난동
2	002	악마동
3	003	거북동
4	004	태오동
5	005	악어동

자 지금은 간단히 수작업으로 해도 되겠다고 생각하시겠지만, 만약 회사 회원이 10 만명이라면? 아울러 전국 수 만개 우편번호에서 찾아야 된다면? 시간이 많이 걸릴 것 같은 저런 작업을 순식간에 커서를 이용해 마무리 지을 수 있다.

실제 커서에 대해서 이야기할 때가 되었다.

```
CREATE TABLE 우편물(

번호 int identity(1,1)

, 우편번호 varchar(3)

, 우편주소 varchar(6)

)

--커서선언

DECLARE cur_konan_Test CURSOR

FOR

SELECT 번호, 우편번호, 우편주소 FROM 우편물
```

자 여기서 커서를 선언 한다? 그러면 해당 커서는 SELECT 구문을 가지고 선언 되는 것이다.

사실 커서에서 가장 중요하고 속도를 좌지우지 하며 여러 기능을 사용 가능하게 할지 말지를 결정하는 이곳이 가장 중요한 곳이다.

여기서 키 포인트 이다. **READ ONLY 와 FORWARD ONLY 가 빠르다.** 하지만 데이터 처리에 제약이 있다. 예를 들면.. 전진 커서만 가능하고 한 앞의 내용을 가져오기 등을 못하는 제약 이다. 아울러 정적 커서인 STATIC 과 KEYSET 커서 방식, DYNAMIC 커서 방식이 있다. 경우에 따라서 다릅니다만. STATIC 은 사용할 데이터를 tempdb 에 복사 한다. KEYSET 방식은 행과 순서를 고정하여 keyset 이라는 키 집합이란 것만, 즉 테이블의 일부만을 tempdb 에 저장한다. 만약 데이터가 삽입되거나 삭제할 경우를 고민할 수 밖에 없는 상황이다. DYNAMIC 커서는 모든 데이터 변경사항이 반영된다. 데이터 삽입 삭제 등이 있어도 반영이 된다는 의미 이다. 예를 들어.. STATIC 으로 커서를 선언하고.. OPEN 까지 한 후, 데이터를 INSERT 한다.

이 커서에서는 tempdb 에 데이터가 있으므로 데이터 반영이 안된다는 것이며 DYNAMIC 은 반영이 되고 이어서 KEYSET 은 어중간하다는 의미 이다.

```
--커서오픈

OPEN cur_konan_Test
```

이어서 커서를 오픈 한다. 아래의 내용은 변수를 선언한다.

```
--변수선언

DECLARE @v_번호 INT

DECLARE @v_우편번호 VARCHAR(3)

DECLARE @v_우편주소 VARCHAR(20)
```

DECLARE 로 변수를 선언하기 어렵지 않으실 것이다.

```
--첫로우 FETCH

FETCH NEXT FROM cur_konan_Test INTO @v_번호, @v_우편번호, @v_우편주소
```

커서에서는 첫 행의 값인 1, 001, 코난동이 각각의 변수에 들어가 있다. 이 내용이 제일 중요하며 이렇게 하는 이유는 **관계형 데이터 처리인 열단위-컬럼단위 처리가 아닌 행단위 처리 하기 때문이다.**

```
WHILE @@FETCH_STATUS = 0

BEGIN

        --FETCH 된데이터를 tempdb 에삽입

        UPDATE 우편물

        SET 우편번호 = (SELECT 우편번호 FROM 우편번호 WHERE 우편주소 = @v_우편주소)

        WHERE 번호 = @v_번호
```

이어서 나타난 **WHILE @@FETCH_STATUS = 0 이것은!!! @@FETCH_STATUS 바로 요건 행 반입 상태를 의미 한다.**

반환 값	**설명**
0	**FETCH 문은 성공적이었다.**
-1	FETCH 문은 실패했거나 행이 결과 집합의 범위를 벗어났다.
-2	반입된 행이 없다.

0 일 경우. 성공 했을 때만 아래 구문을 수행하고 성공이 아니면? 에러이거나 더 이상 가져올 행이 없을 때 테이블의 모든 데이터를 가져 왔을 때 WHILE 루프를 벗어나는 것이다.

UPDATE 우편물
SET 우편번호 = (SELECT 우편번호 FROM 우편번호 WHERE 우편주소 = @v_우편주소)
WHERE 번호 = @v_번호

구문에서 @v_우편주소 에는 무슨 값이 있었을까? 위의 변수 값을 봐 보기 바란다.

이어서 @v_번호 에는 무슨 값이 있었을까? 이 값이 바로 커서의 끝이다.

```
  --다음로우 FEETCH - 루프

FETCH NEXT FROM cur_konan_Test INTO @v_번호, @v_우편번호, @v_우편주소

END
```

루프를 돌면서.. 계속 행을 반입하게 한다.

```
--커서 CLOSE

CLOSE cur_konan_Test

--커서 DEALLOCATE

DEALLOCATE cur_konan_Test

GO
```

이것으로 커서를 닫고!!! 이어서 다 사용한 커서를 해제한다.

커서의 사용 방법은 다양할 수 있다. 배치작업 - (여기서 배치는 T-SQL 문법상의 SQL 구문 실행 단위가 아닌 업무에서 일반적으로 말하는 데이터의 이관작업이나 시간이 오래 걸리는 데이터 처리 작업을 말한다.) 간에 주로 많이 이용을 한다. 데이터를 정밀하게 여러 가지 방법으로 처리를 할 수 있어 유용하다. 위에 정리된 내용을 하나의 형태로 합치면 아래와 같은 결과를 볼 수 있다.

```
--커서선언

DECLARE cur_konan_Test CURSOR

FOR

SELECT 번호, 우편번호, 우편주소 FROM 우편물

--커서오픈

OPEN cur_konan_Test

--변수선언

DECLARE @v_번호 INT

DECLARE @v_우편번호 VARCHAR(3)

DECLARE @v_우편주소 VARCHAR(20)

--첫로우 FETCH
```

```
FETCH NEXT FROM cur_konan_Test INTO @v_번호, @v_우편번호, @v_우편주소

WHILE @@FETCH_STATUS = 0

BEGIN

        --FETCH 된데이터를 tempdb 에삽입

        UPDATE 우편물

        SET 우편번호 = (SELECT 우편번호 FROM 우편번호 WHERE 우편주소 = @v_우편주소)

        WHERE 번호 = @v_번호

--다음로우 FEETCH - 루프

FETCH NEXT FROM cur_konan_Test INTO @v_번호, @v_우편번호, @v_우편주소

END

--커서 CLOSE

CLOSE cur_konan_Test

--커서 DEALLOCATE

DEALLOCATE cur_konan_Test

GO
```

마지막으로 커서를 활용한 간단한 예제로 pubs db 의 모든 테이블의 조회하기 위해 사용하는 예제 이다.

```
use pubs

go

SET NOCOUNT ON

DECLARE cur_konan_Test CURSOR FAST_FORWARD

FOR
```

```
SELECT name FROM sysobjects WHERE xtype='U'

OPEN cur_konan_Test

DECLARE @v_name VARCHAR(100)

FETCH NEXT FROM cur_konan_Test INTO @v_name

WHILE @@FETCH_STATUS = 0

BEGIN

SELECT 'SELECT COUNT(*) FROM ' +@v_name

FETCH NEXT FROM cur_konan_Test INTO @v_name

END

--커서 CLOSE

CLOSE cur_konan_Test

--커서 DEALLOCATE

DEALLOCATE cur_konan_Test

GO
```

```
Results

------------------------------------------------------------
SELECT COUNT(*) FROM authors


------------------------------------------------------------
SELECT COUNT(*) FROM publishers


------------------------------------------------------------
SELECT COUNT(*) FROM titles


------------------------------------------------------------
SELECT COUNT(*) FROM titleauthor


------------------------------------------------------------
SELECT COUNT(*) FROM stores


------------------------------------------------------------
SELECT COUNT(*) FROM sales
```

이런 식으로 문자열 생성을 해낸 후 처리할 수 있겠지요?

하지만 커서는 단점을 가지고 있다. SQL Server 의 리소스를 많이 잡아먹을 뿐 아니라 프로그램의 성능을 떨어뜨리게 되므로 반드시 필요한 곳에서만 사용하도록 합시다.

SQL Server 2005 부터는 이런 커서의 단점을 보완하기 위해 어느 정도 CROSS APPLY 를 대체해서 사용할 수 있다.

DECLARE CURSOR(Transact-SQL): http://msdn.microsoft.com/ko-kr/library/ms180169(v=SQL.110).aspx

CROSS APPLY(Transact-SQL): http://msdn.microsoft.com/en-us/library/ms175156(v=SQL.110).aspx