



SQL 활용

데이터베이스



한국기술교육대학교
온라인평생교육원

학습내용

- 데이터베이스의 개념
- 데이터베이스 관리 시스템

학습목표

- 데이터베이스에 대해 설명할 수 있다.
- 데이터베이스 관리 시스템의 정의를 이해하고, 기능에 대해 설명할 수 있다.

● 데이터베이스의 개념

1. 데이터베이스의 정의

◆ 데이터베이스

- 어느 한 조직의 다양한 응용 프로그램들이 **공동**으로 사용하는 데이터들을 **통합**하여 **저장**한 **운영** 데이터의 집합
- 사람들이 필요로 하는 데이터를 모아둔 것

◆ 데이터베이스 정의에 함축된 개념

① 공용 데이터(Shared Data)

- 한 조직의 여러 응용 프로그램이 공동으로 사용하는 것
- 여러 사용자가 서로 다른 목적으로 공유함

② 통합된 데이터(Integrated Data)

- 여러 부서에서 사용하는 데이터를 한 곳에 모아서 공동 관리하는 것
- 최소한의 중복, 통제된 중복

③ 저장된 데이터(Stored Data)

- 컴퓨터가 접근할 수 있는 디스크와 같은 저장 매체에 저장된 것

④ 운영 데이터(Operational Data)

- 조직의 운영에 기본적으로 반드시 필요한 데이터를 저장하는 것
- 조직의 고유한 기능을 수행하는 데 필수적인 데이터를 저장하는 것

● 데이터베이스의 개념

1. 데이터베이스의 정의

◆ 통합된 데이터란?

- 한 조직 내에서 여러 부서가 유사한 데이터 집합을 사용함

예 학사 관리 시스템

- 교무과 : 학생을 포함한 학교의 전반적 행정사항 관리
 - 학적과 : 학생의 주요 정보(입학, 휴학, 복학, 자퇴 등) 관리
 - 학생과 : 학생의 장학금, 진학, 취업 등 지원
- ⇒ 학생 정보를 세 개의 부서가 사용함

- 중복성 문제 inconsistency

예 학사 관리 시스템

- 각 부서가 독립적으로 학생 정보를 관리함
- 학적과에서는 휴학중인 학생이 학생과에 의해서 근로 장학생으로 선발됨
⇒ 데이터의 일관성 문제 발생

- 한 여러 부서가 사용하는 데이터를 통합하여 중복성을 없애고 일관성을 유지함
- 각 부서가 데이터를 따로 가지고 있으면, 검색이나 운영 시에 편리함
 - 최소한의 중복
 - 통제된 중복

● 데이터베이스의 개념

2. 데이터베이스의 특징

① 실시간 접근성(Real-Time Accessibility)

- 질의에 대한 실시간 처리 및 응답

② 계속적인 변화(Continuous Evolution)

- 간신, 삽입, 삭제 : 동적 특성

③ 동시 공용(Concurrent Sharing)

- 여러 사용자가 동시에 사용함

④ 내용에 의한 참조(Content References)

- 위치나 주소가 아닌 값에 따라 참조함

◆ 내용에 의한 참조란?

프로그래밍 언어

- 주소에 의한 검색
 - 특정 메모리 위치에 있는 값을 알려줌
- 메모리 주소 : 0xFFFFA18FF
 - 외우기 어려움
- 변수
 - 메모리 주소 대신 특정이름으로 x, idx 등으로 표시 놓으면 기억하기 좋음
- 데이터베이스에서의 검색

예 학번이 100번인 학생의 이름을 검색하라.

```
SQL : SELECT NAME  
      FROM STUDENT  
      WHERE STUDENTNUM = 100
```

● 데이터베이스 관리 시스템

1. 데이터베이스 관리 시스템의 정의

◆ 데이터베이스

- 관련 있는 데이터들의 집합

◆ 데이터베이스 관리 시스템

Database Management System(DBMS)

- 데이터베이스를 생성 및 관리해주는 기능을 제공하는 소프트웨어 패키지 / 시스템
- 데이터와 응용 프로그램 사이의 중계자
- 모든 사용자와 응용 프로그램들이 데이터베이스를 공유할 수 있도록 지원해주는 범용 목적의 소프트웨어 시스템

DB+DBMS+응용프로그램

- 데이터베이스, DBMS, 데이터베이스 시스템을 혼용해서 사용함

● 데이터베이스 관리 시스템

2. 데이터베이스 관리 시스템의 기능

① 데이터 정의 기능

- 여러 사용자의 데이터를 통합하여 저장하고 공유할 수 있도록 데이터 모델에 따라서 정의하는 기능

② 데이터 조작 기능

- 사용자와 데이터베이스 간의 의사소통
- 데이터베이스의 접근 및 조작 기능 제공
 - 삽입
 - 삭제
 - 변경 및 검색

- 사용자가 사용하기 쉽고, 원하는 처리를 자연스럽게 표현할 수 있어야 함

③ 데이터 제어 기능

- 데이터 일관성(Consistency)과 무결성(Integrity), 보안(Security)을 유지하는 기능
 - 백업과 파손 회복(Recovery)
 - 인증(Authorization)과 보안(Security)
 - 병행제어(Concurrency Control)

● 데이터베이스 관리 시스템

3. 데이터베이스 관리 시스템의 역사

① 1세대 DBMS

- IDS (Integrated Data Store)
 - 최초의 범용 목적의 DBMS
 - 1960년대 초 GE의 Charles Bachman에 의하여 제시됨
 - Network Data Model 기반
- IMS(Information Management System) DBMS
 - 1960년 후반 IBM에서 제시함
 - 계층적 데이터 모델에 기반함
 - 1970년대 초 많은 회사들이 자신들만의 DBMS를 만들기 시작함

② 2세대 DBMS

- Relational Data Model, SQL
 - IBM의 E.F. Codd
- Commercial DBMS
 - Oracle, DB2, Ingress, Sybase, Informix

③ 3세대 DBMS

- 데이터의 복잡성 증가
 - Image, Video 등
- 새로운 데이터 모델의 대두
 - 객체지향 데이터베이스

④ 현재 DBMS

- ORDBMS
 - 2세대 DBMS + 3세대 DBMS
- 객체 관계형 데이터베이스 관리 시스템

핵심요약

1. 데이터베이스의 개념

■ 데이터베이스의 정의

- 데이터베이스는 어느 한 조직의 다양한 응용 프로그램들이 공동으로 사용하는 데이터들을 통합하여 저장한 운영 데이터의 집합
- 데이터베이스는 공용 데이터(Shared Data)임
- 데이터베이스는 통합된 데이터(Integrated Data)임
- 데이터베이스는 저장된 데이터(Stored Data)임
- 데이터베이스의 데이터는 운영 데이터(Operational Data)임

■ 데이터베이스의 특징

- 실시간 접근성 (Real-time Accessibility)
- 계속적인 변화 (Continuous Evolution)
- 동시 공용 (Concurrent Sharing)
- 내용에 의한 참조 (Content References)

핵심요약

2. 데이터베이스 관리 시스템

■ 데이터베이스 관리 시스템의 정의

- Database Management System(DBMS)
- 데이터베이스를 생성 및 관리해주는 기능을 제공하는 소프트웨어 패키지/시스템
- 데이터와 응용 프로그램 사이의 중계자
- 모든 사용자와 응용 프로그램들이 데이터베이스를 공유할 수 있도록 지원해 주는 범용 목적의 소프트웨어 시스템

■ 데이터베이스 관리 시스템의 기능

■ 데이터 정의 기능

- 여러 사용자의 데이터를 통합하여 저장하고 공유할 수 있도록 데이터 모델에 따라서 정의하는 기능

■ 데이터 조작 기능

- 사용자와 데이터베이스 간의 의사소통
- 데이터베이스의 접근 및 조작(삽입, 삭제, 변경 및 검색) 기능 제공

■ 데이터 제어 기능

- 데이터 일관성(Consistency)과 무결성(Integrity), 보안(Security)을 유지하는 기능

핵심요약

2. 데이터베이스 관리 시스템

■ 데이터베이스 관리 시스템의 역사

■ 1세대 DBMS

- IDS (Integrated Data Store) : 최초의 범용 목적의 DBMS
- IMS(Information Management System) DBMS

■ 2세대 DBMS

- Relational Data Model, SQL : IBM의 E.F. Codd
- Commercial DBMS
- Oracle, DB2, Ingress, Sybase, Informix

■ 3세대 DBMS

- 데이터의 복잡성 증가
- 새로운 데이터 모델의 대두
- 2세대 DBMS+3세대 DBMS ORDBMS
 - ⇒ 객체 관계형 데이터베이스 관리 시스템



SQL 활용

관계형 데이터 모델



한국기술교육대학교
온라인평생교육원

학습내용

- 관계형 데이터 모델의 구조와 연산
- 관계형 데이터 모델의 제약조건

학습목표

- 관계형 데이터 모델의 구조 및 연산을 사용할 수 있다.
- 관계형 데이터 모델의 제약조건을 설명할 수 있다.

● 관계형 데이터 모델의 구조와 연산

1. 데이터 모델

◆ 모델 $D = \langle S, O, C \rangle$

- Structure : 구조
 - 데이터의 구조
 - 정적 성질, 개체 타입과 이들 간의 관계를 명세함
- Operation : 연산
 - 데이터의 동적 성질
 - 개체 인스턴스를 처리하는 작업에 대한 명세
 - 데이터 조작 기법
- Constraint : 제약조건
 - 데이터의 논리적 제약
 - 구조로부터 파생, 의미적 제약
 - 데이터 조작의 한계를 표현한 규정
- 데이터 모델의 예

예 정수

- 정수(Integer)의 구조
 - … -2, -1, 0, 1, 2, 3, …
- 연산
 - 사칙 연산 : +, -, ×, /
- 제약조건 : 정수의 세계로 한정함
 - 정수/정수 \Rightarrow 정의할 수 없는 경우가 있음
 - $5/2 \Rightarrow 2.5$: 정수가 아님

● 관계형 데이터 모델의 구조와 연산

2. 관계형 데이터 모델의 구조와 연산

◆ 관계형 데이터 모델을 사용하는 이유

- 모델의 구조가 단순함
- 집합 이론(Set Theory)이라는 수학적 이론에 기반하여 모델이 강건(Sound)함
- SQL이라는 간단한 비절차적 언어로 사용하기 쉬움

◆ 관계형 데이터 모델

- 구조 : 릴레이션(또는 테이블)
- 연산 : 관계 대수(Relational Algebra)
- 제약조건 : 무결성(Integrity) 제약조건

◆ 관계형 데이터 모델의 구조

- 릴레이션
- 2차원 테이블 형태
- 테이블의 행(Row) \Rightarrow 튜플
 - 릴레이션의 튜플들의 집합
- 테이블의 열(Column) \Rightarrow 속성(Attribute)
 - 도메인(Domain) : 속성이 가질 수 있는 값의 범위

예 학생

학번	이름	주소	학년	스키마
100	김진현	서울	4	
101	박재영	대전	1	
102	김홍연	제주	3	튜플

속성

● 관계형 데이터 모델의 구조와 연산

2. 관계형 데이터 모델의 구조와 연산

◆ 릴레이션의 특징

① 튜플의 유일성

- 릴레이션은 튜플의 집합임
- 집합은 중복을 허용하지 않음

② 튜플의 무순서성

- 릴레이션은 튜플의 집합임
- 집합에서 원소들 간의 순서는 없음

③ 속성(Attribute)의 무순서성

- 릴레이션 스키마는 속성(Attribute)들의 집합임

④ 속성(Attribute)의 원자성(Atomicity)

- 속성(Attribute)의 값은 원자값임
- 논리적으로 더 분해할 수 없음

◆ 관계형 데이터 모델의 연산

- 관계 대수(Relational Algebra)

- 관계 대수와 SQL

- 관계 대수 : 시스템 관점
- SQL : 사용자 위주

● 관계형 데이터 모델의 구조와 연산

2. 관계형 데이터 모델의 구조와 연산

◆ 관계 대수 연산자의 분류

● 일반 집합 연산자

- 합집합(UNION, \cup)
- 교집합(INTERSECT, \cap)
- 차집합(DIFFERENCE, $-$)
- 카티션 프로덕트(CARTESIAN PRODUCT, \times)
- 합병호환성
- \cup , \cap , $-$ 연산의 피연산자(릴레이션)들이 지켜야 할 제약조건
 - ① 차수(Degree : 속성의 수)가 같아야 함
 - ② 대응되는 속성(Attribute) 쌍 별로 타입(또는 도메인)이 같아야 함
 - ③ 대응되는 속성(Attribute) 쌍 별로 의미(Semantic)가 같아야 함
 - ①, ② 조건의 시스템에서 확인해줌
 - ③ 조건은 사람이 해야 함

예 몸무게(실수) \cup 키(실수)

- ①, ② 조건은 만족, 그러나 몸무게와 키를 합친 결과는 의미 없음

● 관계형 데이터 모델의 구조와 연산

2. 관계형 데이터 모델의 구조와 연산

◆ 관계 대수 연산자의 분류

- 순수관계 연산자 : 릴레이션이 2차원 구조이기 때문에 유도되는 연산자
 - 셀렉트(Select : $\sigma(\Sigma)$) 수평적 부분집합
 - A, B가 릴레이션 R의 속성(Attribute)일 때

$$\begin{aligned}\sigma_{A\theta v}(R) &= \{ r \mid r \in R \text{ and } r.A \theta v \} \\ \sigma_{A\theta v}(R) &= \{ r \mid r \in R \text{ and } r.A \theta r.B \}\end{aligned}$$

- θ (Theta) : 비교 연산자
- $\{<, >, \leq, \geq, =, \neq\}, v$: 상수
- 프로젝트(Project : $\Pi(P)$) 수직적 부분집합
 - A릴레이션 R(X)에서

$$Y \subseteq X \text{이고 } Y = \{B_1, B_2, \dots, B_m\} \text{ 이면,} \\ \Pi_Y(R) = \{ \langle r.B_1, \dots, r.B_m \rangle \mid r \in R \}$$

- 프로젝트(Π)의 결과는 선택 조건을 만족하는 릴레이션의 수직적 부분집합(Vertical Subset)
 - 폐쇄성질
 - 중복된 튜플들은 제거됨
- 관계표현식의 예제

Π 이름, 성적 ($\sigma_{\text{학번}=300}(\text{학생})$)

학생	이름	학번	학과	성적
김홍연		100	컴공	2.8
강성민		200	컴공	3.9
빈준길		300	전자	2.3
이석주		400	기계	3.5

● 관계형 데이터 모델의 구조와 연산

2. 관계형 데이터 모델의 구조와 연산

◆ 관계 대수 연산자의 분류

- 순수관계 연산자 : 릴레이션이 2차원 구조이기 때문에 유도되는 연산자
 - 조인(JOIN, \bowtie)

$R(X), S(Y), A \in X, B \in Y$ 에 대하여

$$R \bowtie_{A \theta B} S = \{ r \cdot s \mid r \in R \text{ and } s \in S \text{ and } (r.A \theta s.B) \}$$

A, B : 조인 속성(Join Attribute)

결과 차수 = R의 차수 + S의 차수

- 조인의 예

$\Pi_{\text{교수이름}} (\text{학생} \bowtie_{\text{지도교수번호}=\text{교번}} \text{교수}) = \text{이교수}$

학생	학번	이름	지도교수번호
	100	개똥이	3
	200	소똥이	1
	300	말똥이	2

교수	교번	교수이름
	1	민교수
	2	박교수
	3	이교수

▪ 자연 조인(\bowtie_N)

- 일반적으로 아무 말 없이 조인이라고 하면 “자연 조인”을 말함
- 두 테이블에 공통으로 나타나는 속성에 대한 동등 조인으로 간주

● 관계형 데이터 모델의 제약조건

1. 키(Key)

◆ 키(Key)의 개념

- 하나의 테이블 내에서 각 튜플의 유일하게 식별할 수 있는 속성(Attribute)들의 집합
- 실생활의 키
 - 하나의 자물쇠를 열수 있는 열쇠(키)는 오직 하나임
 - 생활의 편리성을 위하여 키를 복제하여 쓰지만 논리적으로는 하나임

◆ 후보키(Candidate Key)

- 한 릴레이션 $R(A_1, \dots, A_n)$ 에 대한 속성의 집합 $K=\{A_i, \dots, A_j\}$ 으로 다음과 같은 성질을 만족함
 - 유일성(Uniqueness) : 서로 다른 두 튜플의 속성집합 K 의 값은 같지 않음
 - 최소성(Minimality) : K 는 서로 다른 두 튜플을 식별하기 위한 최소한의 속성들로만 이루어져 있음
- 릴레이션의 특징 : 튜플의 유일성
- 후보키 : 튜플의 유일성을 유지시키는 최소 속성 집합

학번	이름	지도교수번호
100	개똥이	3
200	소똥이	1
300	말똥이	2

- 유일성(Uniqueness)
 - {학번, 이름, 주소}가 같은 튜플은 없음
 - {학번, 이름}이 같은 튜플도 없음
 - {학번}이 같은 튜플도 없음
- 최소성
 - {학번}이 최소로 이루어져 있음

후보키는 {학번}임

● 관계형 데이터 모델의 제약조건

1. 키(Key)

◆ 수퍼키(Super Key)

- 유일성을 만족하는 속성 집합
- 최소성을 만족하지 않아도 됨
- 일반적으로 후보기는 수퍼키의 부분 집합임 \Leftrightarrow 수퍼키는 후보기를 포함함

◆ 기본키(Primary Key)

- 하나의 릴레이션에는 후보기가 여러 개 있을 수 있음
- 여러 개의 후보기 중 DBA가 지정한 하나의 키임

◆ 대체키(Alternative Key)

- 후보기 중 기본키를 제외한 나머지 후보기

학번	이름	주소	주민번호
100	개똥이	천안	xxx-xxxx1
200	소똥이	천안	xxx-xxxx4
300	말똥이	천안	xxx-xxxx2

- 후보기
 - {학번}
 - {주민번호}
- 수퍼키
 - 학번, 이름, 주소, 주민번호}
 - {학번, 이름, 주소}
 - {학번, 이름}
 - {이름, 주소, 주민번호} ...
- 기본키 : {학번}
- 대체키 : {주민번호}

● 관계형 데이터 모델의 제약조건

1. 키(Key)

◆ 외래키

- 한 릴레이션 R1의 튜플과 다른 릴레이션 R2의 하나의 튜플과의 연관 관계를 표시하기 위하여 사용함
- R1의 속성집합 FK의 도메인이 R2의 기본키 일 때, FK를 R1의 외래키라 함
- R1을 참조 릴레이션, R2를 피참조 릴레이션이라고 함
 - R1과 R2가 다른 릴레이션일 필요는 없음

사번	이름	직급	관리자
1	개	사장	-
2	소	부장	1
3	말	과장	2
4	닭	사원	3

◆ 개체 무결성

- 의미 : 서로 다른 두 튜플은 같을 수 없음
- 정의 : 기본키 값은 언제고 어느 때고 NULL 값일 수 없음

● 관계형 데이터 모델의 제약조건

1. 키(Key)

◆ 참조 무결성

- 외래키 값은 피참조 릴레이션의 기본키 값이거나 NULL 값임
- 추가 지정을 통해 NULL을 가질 수 없다고 제약을 걸 수 있음

학생		외래키	기본키	교수
학번	이름	지도교수번호	교번	교수이름
100	개똥이	3	1	민교수
200	소똥이	1	2	박교수
300	말똥이	2	3	이교수
400	쥐똥이	-		
500	닭똥이	9		

- 쥐똥이는 신입생이라 아직 지도교수 배정 안됨 \Rightarrow 아직 모름 \Rightarrow NULL

학생		외래키	기본키	교수
학번	이름	지도교수번호	교번	교수이름
100	개똥이	3	1	민교수
200	소똥이	1	2	박교수
300	말똥이	2	3	이교수
400	쥐똥이	-		
500	닭똥이	9		

- 닭똥이는 지교교수가 9번이라고 하지만 교수테이블에는 9번 교수가 없음
 \Rightarrow 참조 무결성 위반
- 이런 튜플은 존재할 수 없음

● 관계형 데이터 모델의 제약조건

1. 키(Key)

◆ 도메인 무결성

- 속성(Attribute) 값은 해당 속성(Attribute) 도메인에 속한 값을 중 하나이어야 함

예 대학생의 학년 도메인 : 1, 2, 3, 4

- 9학년 또는 10학년은 존재할 수 없음

- DBMS는 데이터베이스의 상태 변화(삽입, 갱신, 삭제)에도 항상 무결성 제약조건을 검사하고 유지시킴

핵심요약

1. 관계형 데이터 모델의 구조와 연산

■ 데이터 모델

■ 구조

- 데이터의 구조
- 정적 성질, 개체 탑과 이들 간의 관계를 명세

■ 연산

- 데이터의 동적 성질
- 개체 인스턴스를 처리하는 작업에 대한 명세
- 데이터 조작 기법

■ 제약조건

- 데이터의 논리적 제약
- 구조로 부터 파생, 의미적 제약
- 데이터 조작의 한계를 표현한 규정

핵심요약

1. 관계형 데이터 모델의 구조와 연산

■ 관계형 데이터 모델의 구조와 연산

■ 관계형 데이터 모델의 구조

- 릴레이션
- 2차원 테이블 형태
- 테이블의 행(Row) → 튜플
- 테이블의 열(Column) → 속성(Attribute)

■ 일반 집합 연산자

- 합집합(UNION, \cup)
- 교집합(INTERSECT, \cap)
- 차집합(DIFFERENCE, $-$)
- 카티션 프로덕트(CARTESIAN PRODUCT, \times)

■ 순수관계 연산자

- 실렉트(SELECT, σ)
- 프로젝트(PROJECT, Π)
- 조인(JOIN, \bowtie)
- 디비전(DIVISION, \div)

핵심요약

2. 관계형 데이터 모델의 제약조건

■ 키

- 하나의 테이블 내에서 각 튜플의 유일하게 식별할 수 있는 속성(Attribute)들의 집합
- 하나의 자물쇠를 열수 있는 열쇠(키)는 오직 하나임
- 후보키 (Candidate Key)
 - 한 릴레이션 R (A_1, \dots, A_n)에 대한 속성의 집합
 - 유일성, 최소성
- 수퍼키(Super Key)
 - 유일성을 만족하는 속성 집합
 - 최소성을 만족하지 않아도 됨
- 기본키(Primary Key)
 - 하나의 릴레이션에는 후보키가 여러 개 있을 수 있음
 - 여러 개의 후보키 중 DBA가 지정한 하나의 키
- 대체키(Alternative Key)
 - 후보키 중 기본키를 제외한 나머지 후보키

핵심요약

2. 관계형 데이터 모델의 제약조건

■ 개체 무결성, 참조 무결성, 도메인 무결성

■ 개체 무결성

- 의미 : 서로 다른 두 튜플은 같을 수 없음
- 정의 : 기본키 값은 언제고 어느 때고 NULL값일 수 없음

■ 참조 무결성

- 외래키 값은 피참조 릴레이션의 기본키 값이거나 NULL 값임
- 추가 지정을 통해 NULL을 가질 수 없다고 제약을 걸을 수 있음

■ 도메인 무결성

- 속성(Attribute) 값은 해당 속성(Attribute) 도메인에 속한 값을 중 하나이어야 함
- DBMS는 데이터베이스의 상태 변화(삽입, 갱신, 삭제)에도 항상 무결성 제약조건을 검사하고 유지시킴



SQL 활용

SQL의 개념과 T-SQL



한국기술교육대학교
온라인평생교육원

학습내용

- SQL의 개념
- T-SQL

학습목표

- SQL의 특징에 대해 설명할 수 있다.
- MS-SQL에 존재하는 테이블의 구조를 설명할 수 있다.

● SQL의 개념

1. SQL의 역사와 특징

◆ SQL의 역사

- SEQUEL(Structured English Query Language)
 - 1974년, IBM San Jose Lab(현재 IBM Almaden 연구소)
 - 최초의 관계형 데이터베이스 관리 시스템 프로토타입인 SystemR을 위한 데이터베이스 언어로 개발됨
- SQL
 - 1986년 ANSI에서 관계형 데이터베이스 표준언어로 인증
 - SQL 지속적인 개선 진행
 - 1986년 : SQL-86 또는 SQL1
 - 1992년 : SQL/92, SQL-92 또는 SQL2
 - 1999년 : SQL-99 또는 SQL3
 - 2003년 : SQL4, SQL-2003(객체 지향 개념 추가)

● SQL의 개념

1. SQL의 역사와 특징

◆ SQL의 특징

① SQL은 무엇인가?

- 종합 데이터베이스 언어 \Leftrightarrow 데이터 정의(DDL), 조작(DML), 제어(DCL)
 - 무엇(What) 을 표시하며 어떻게(How)는 표시하지 않음
 - 어떻게는 DBMS가 알아서 처리함

② 왜 관계 대수식 대신 SQL을 사용하는가?

- 관계 대수식 연산자 기호는 키보드로 표기하기 어려움

③ 관계 대수식(Relational Algebra)과 SQL과의 차이점은 무엇인가?

- 관계 대수식
 - Relation
 - 튜플의 집합(Set)
 - 중복을 허용하지 않음
- SQL
 - 튜플의 백(Bag)
 - 튜플들 간의 순서는 없으나 중복을 허용함


- 집합을 유지하려면 SQL의 결과에서 항상 중복되는 내용을 제거해야 함
- 중복을 제거하는 것 = 튜플을 정렬(Sorting)하는 것과 같은 문제임
- 결과 생성 시 시간이 오래 걸림 \Leftrightarrow 성능 문제 발생

● SQL의 개념

2. SQL 기본 구문

◆ DDL 문 : 데이터 정의문

● 테이블 생성 : CREATE 문

```
CREATE TABLE 테이블명  
( 속성명 속성타입 [제약조건],  
속성명 속성타입,  
...  
)
```

- 제약 조건 : NOT NULL, PRIMARY KEY, UNIQUE

● 테이블 삭제 : DROP문

```
DROP TABLE 테이블명
```

예 학생(STUDENT) 테이블을 제거하시오.

```
DROP TABLE STUDENT
```

● 테이블 구조 변경 : ALTER문

- 속성 추가

```
ALTER TABLE 테이블명(ADD 속성명 속성타입)
```

- 속성 제거

```
ALTER TABLE 테이블명(DROP 속성명 속성타입)
```

- 속성 타입 변경

```
ALTER TABLE 테이블명(ALTER 속성명 속성타입)
```

● SQL의 개념

2. SQL 기본 구문

◆ DML 문 : 데이터 조작문

● 튜플 삽입 : INSERT 문

```
INSERT INTO 테이블명(속성명, 속성명, ... )  
VALUES (속성값, 속성값, ...)
```

● 튜플 변경 : UPDATE문

```
UPDATE 테이블명  
SET 속성명= 수식  
[WHERE 조건]
```

● 튜플 삭제 : DELETE문

```
DELETE FROM 테이블명  
[WHERE 조건]
```

● 트랜잭션 관련

```
SELECT 속성명, 속성명, ...  
FROM 테이블명  
[WHERE 조건]
```

● SQL의 개념

2. SQL 기본 구문

◆ DCL 문 : 데이터 제어문

- 트랜잭션 관련

COMMIT

ROLLBACK

- 사용자 권한 제어 관련

GRANT

REVOKE

DENY

● T-SQL

1. MS-SQL

◆ 상업용 DBMS들

DBMS	DBMS	특징
Access	Microsoft	원도우즈 플랫폼으로 중소 규모 데이터베이스를 위한 데스크톱용 DBMS
SQL Server	Microsoft	저렴한 제품 가격으로 Windows NT 플랫폼에서 최적의 성능을 발휘
Informix	IBM	성능이 뛰어나며 병렬처리를 위한 멀티스레드(Multithread)를 지원
DB2	IBM	다수 사용자가 다수 관계형 데이터베이스를 동시에 접근할 수 있는 대형 데이터베이스를 위한 시스템
Oracle	Oracle	PC급에서 메인프레임급까지 모두 설치할 수 있으며, 분산처리 지원 기능이 우수함
MySQL	MySQL AB	다양한 플랫폼과 API를 지원하는 비상업용 DBMS

◆ MS-SQL Editions

- MS-SQL Server
 - Microsoft에서 제공하는 데이터베이스 관리 시스템
- Edition 종류
 - Express
 - Workgroup
 - Standard
 - Enterprise

- 
- 실습을 위하여 본 과목에서는 MS-SQL Server 2014 Express version을 사용함
 - T-SQL의 MS-SQL Server에서 사용되는 SQL문을 말함

● T-SQL

2. MS-SQL의 설치와 구동

◆ MS-SQL 받기

● URL 접속

The screenshot shows the Microsoft Developer Network (msdn.microsoft.com) with the URL <https://msdn.microsoft.com/sqlserver2014expressresources.aspx>. The page is titled "Microsoft SQL Server 2014 Express 평가 리소스" (Evaluation Resources). It features a large blue button labeled "지금 시작하기" (Start Now) with a white arrow. Below it are four sections: "지금 시작" (Start Now), "지금 사용해 보기" (Use Now), "온라인 학습" (Online Learning), and "소설 미디어" (eBooks). Each section has a small icon and a brief description.

<https://msdn.microsoft.com/sqlserver2014expressresources.aspx>

* SQL 서버 2014 버전을 선택하십시오 인터넷 다운로드하고 싶습니다

- SQL 서버 2014 익스프레스하여 LocalDB 32 비트
- SQL 서버 2014 익스프레스 64 비트하여 LocalDB
- SQL 서버 익스프레스 2014 32 비트
- SQL 서버 2014 익스프레스 64 비트
- 도구와 SQL 서버 2014 익스프레스 32 비트
- 도구의 64 비트와 SQL 서버 2014 익스프레스
- SQL 서버 2014 관리 스튜디오 익스프레스 32 비트
- SQL 서버 2014 관리 스튜디오 익스프레스 64 비트
- 고급 서비스와 SQL 서버 2014 익스프레스 32 비트
- 고급 서비스 64 비트와 SQL 서버 2014 익스프레스

* SQL 서버의 언어 2014 익스프레스 W / 도구 32 비트 다운로드를 선택하세요

- 중국어 (간체)
- 중국어 (번체)
- 영어
- 프랑스어
- 독일어
- 이탈리아어
- 일본어
- 한국어
- 포르투갈어 (브라질)
- 러시아어
- 스페인어

① 64bit 운영체제 사용시에는
“도구의 64비트와 SQL 서버
2014 익스프레스” 선택

② 웹페이지 하단의 “동의함” 단추를
선택하여 MS-SQL 설치 파일
다운로드

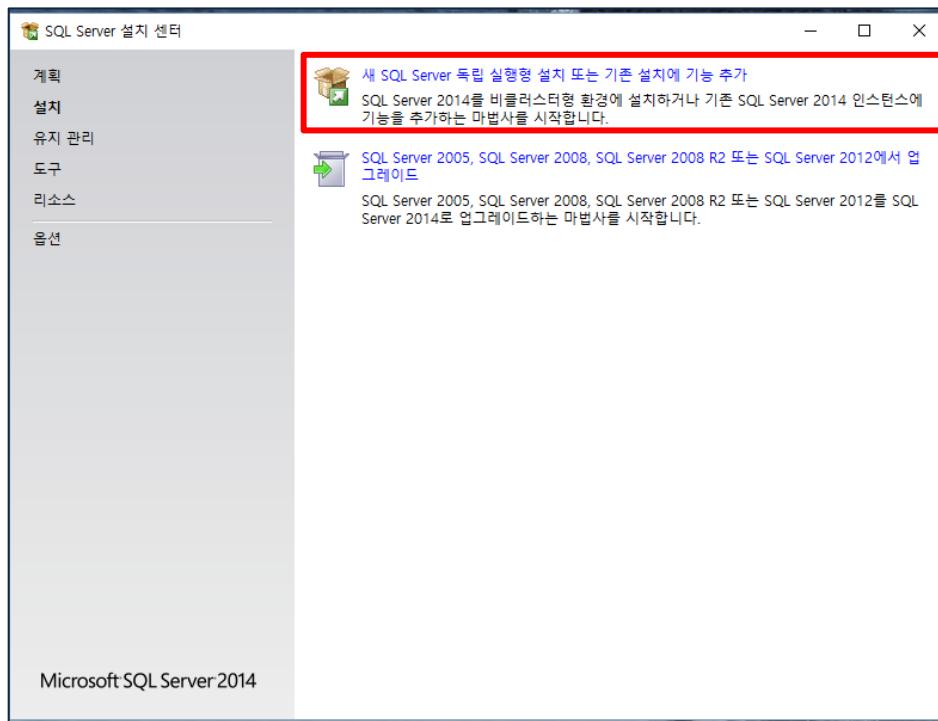
- 32bit의 경우:
SQLEXPRESS_x86_KOR.exe
- 64bit의 경우:
SQLEXPRESS_x64_KOR.exe

● T-SQL

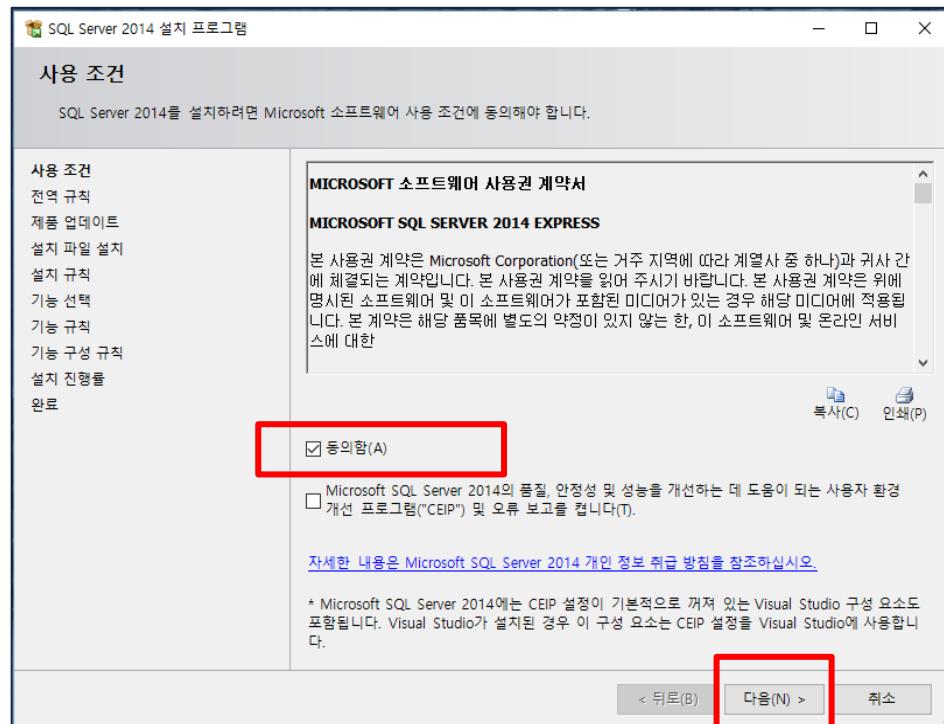
2. MS-SQL의 설치와 구동

◆ MS-SQL 설치

① 새 SQL Server 독립 실행형 설치 또는 기존 설치에 기능 추가 선택



② 사용 조건 - 사용조건 확인 후 동의 체크, 다음 버튼 선택

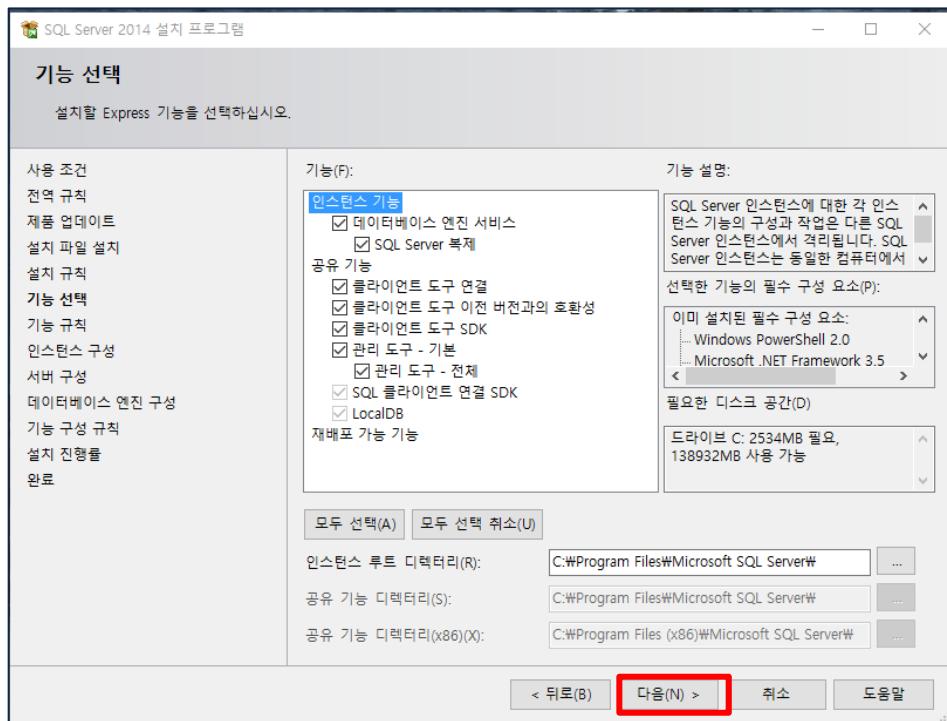


● T-SQL

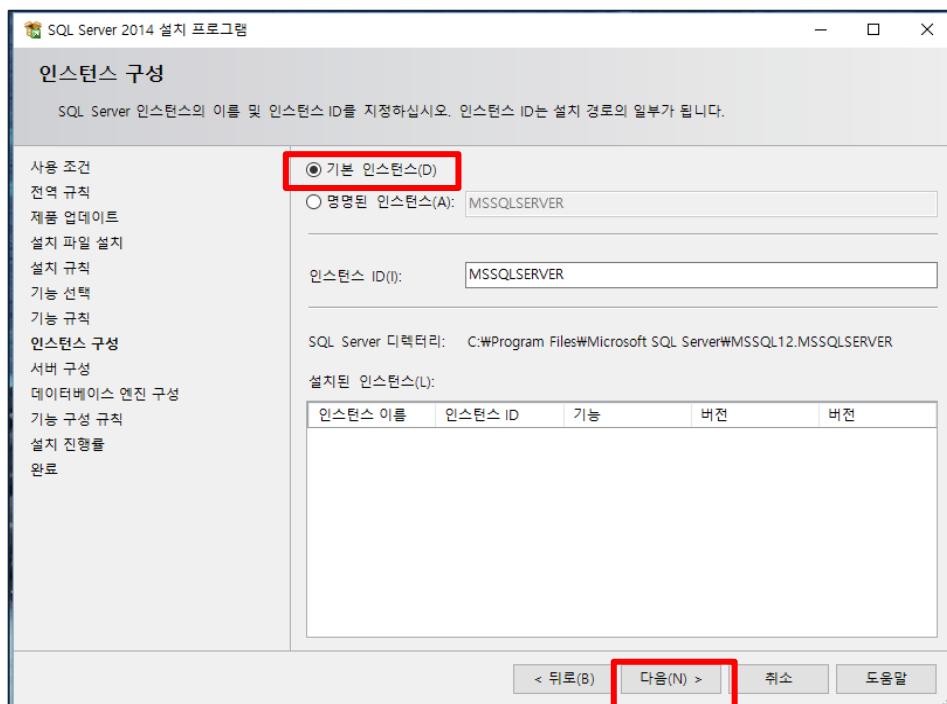
2. MS-SQL의 설치와 구동

◆ MS-SQL 설치

③ 기능 선택 - 설치할 Express 기능 선택 후 다음 버튼 선택



④ 인스턴스 구성 - 기본 인스턴스 선택 후 다음 버튼 선택

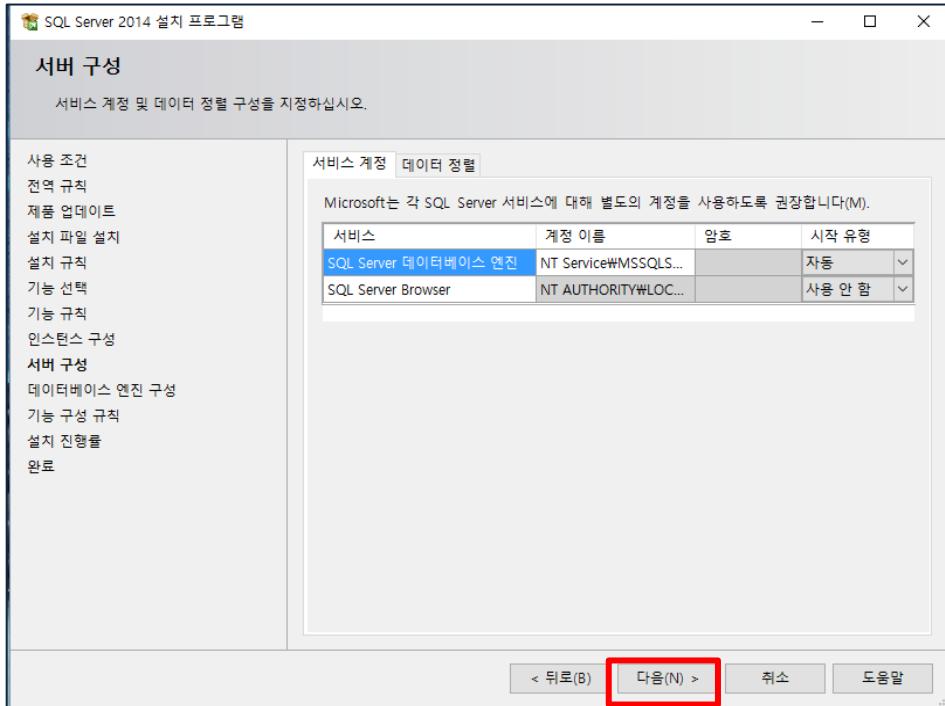


● T-SQL

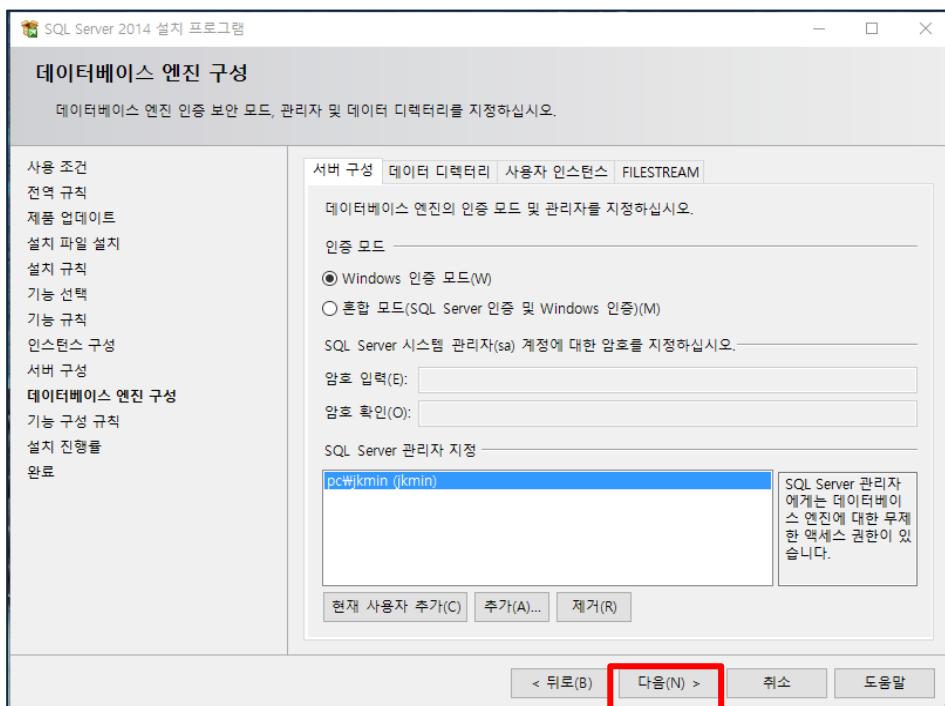
2. MS-SQL의 설치와 구동

◆ MS-SQL 설치

- ⑤ 서버구성 - SQL Server 데이터베이스 엔진 시작유형 자동으로 설정한 후 다음 버튼 선택



- ⑥ 데이터베이스 엔진 구성 - SQL Server 관리자 지정 후 다음 버튼 선택

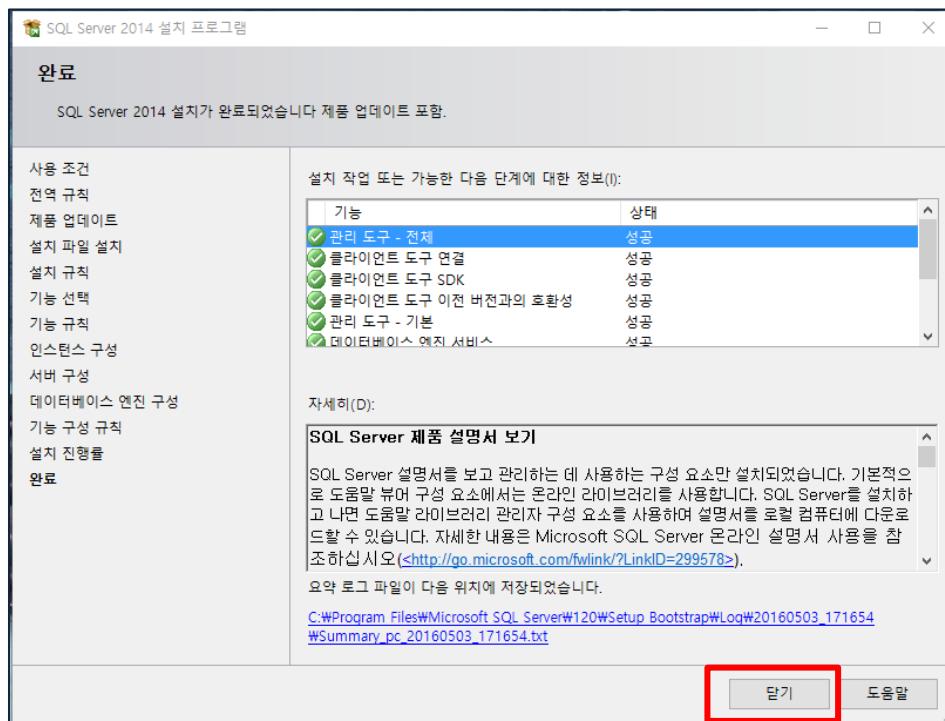
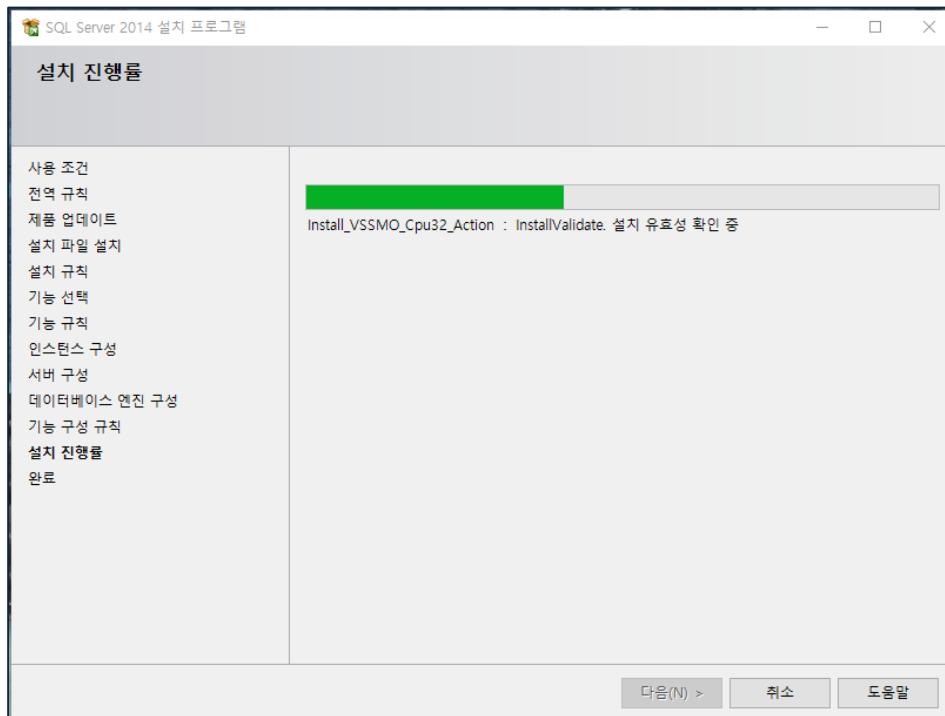


● T-SQL

2. MS-SQL의 설치와 구동

◆ MS-SQL 설치

⑦ 설치 확인 후 닫기

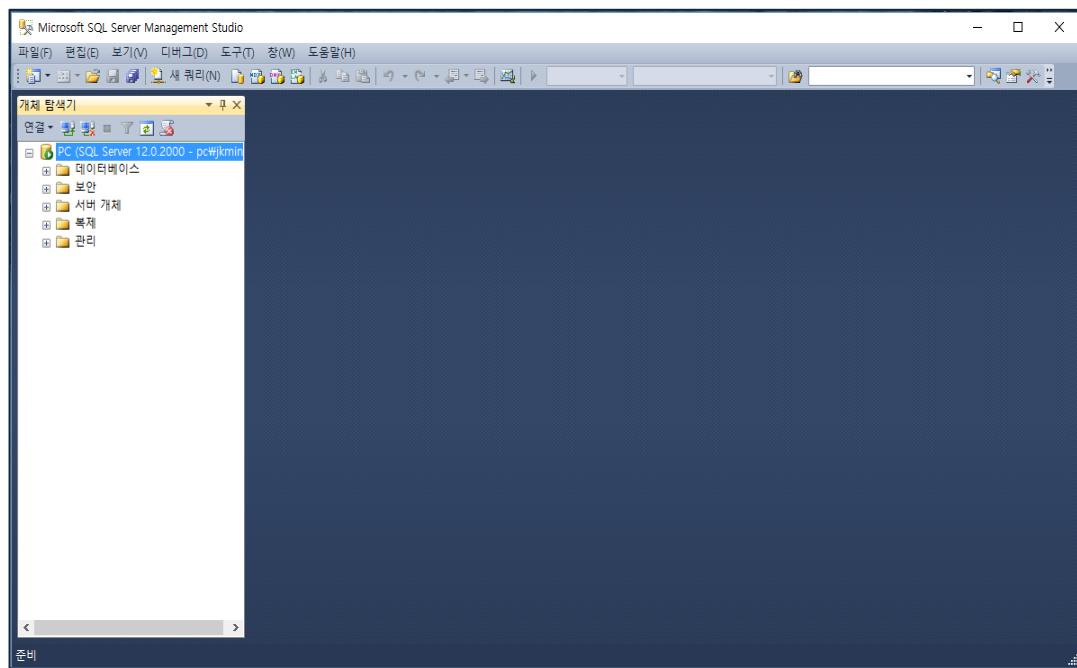
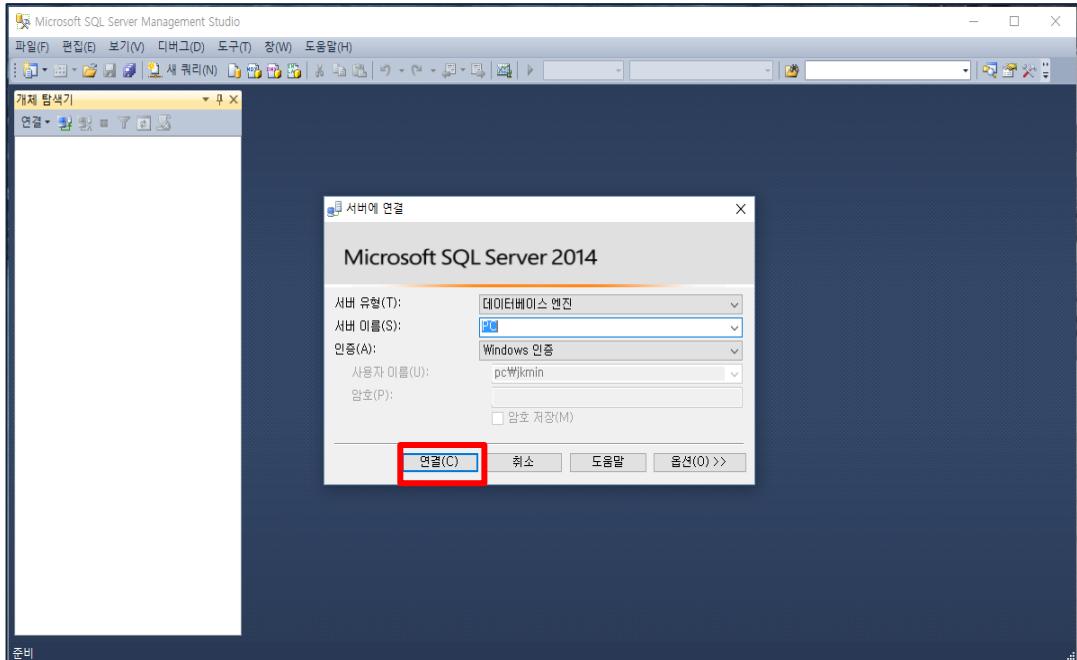


● T-SQL

2. MS-SQL의 설치와 구동

◆ MS-SQL 구동

● Database

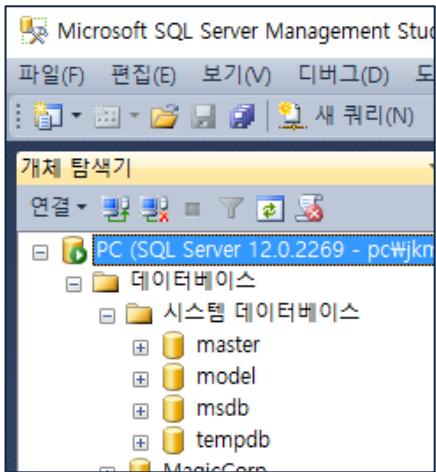


● T-SQL

2. MS-SQL의 설치와 구동

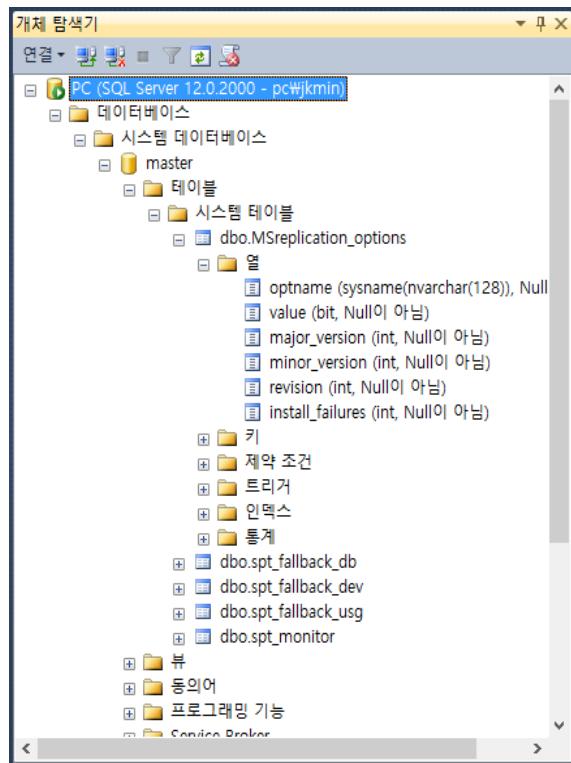
◆ MS-SQL 구동

● Database



- System Database
 - master : 시스템 관리용
 - model : 새 데이터베이스 생성을 위한 템플릿
 - msdb : SQL Server agent를 위한 데이터베이스
 - tempdb : 질의 임시결과 유지를 위한 테이블
- User Database
 - SSMS이나 T-SQL(SQL of MS SQL)을 이용해서 생성

● 테이블 구조 파악



- 데이터베이스 내에 있는 테이블의 구조는 개체 탐색기를 이용하여 손쉽게 파악이 가능함

핵심요약

1. SQL의 개념

■ SQL의 역사

- SEQUEL(Structured English Query Language)
 - 1974년, IBM San Jose Lab(현재 IBM Almaden 연구소)
 - 최초의 관계형 데이터베이스 관리 시스템 프로토타입인 SystemR을 위한 데이터베이스 언어로 개발됨
- SQL
 - 1986년 ANSI에서 관계형 데이터베이스 표준언어로 인증

■ SQL의 특징

- SQL이란?
 - 종합 데이터베이스 언어
 - 데이터 정의(DDL), 조작(DML), 제어(DCL)
 - 무엇(What) 을 표시하며 어떻게(How)는 표시하지 않음
 - 어떻게는 DBMS가 처리함

핵심요약

1. SQL의 개념

■ SQL의 기본 구문

■ DDL 문 : 데이터 정의문

- 테이블 생성 : CREATE 문

```
CREATE TABLE 테이블명  
( 속성명 속성타입 [제약조건],  
속성명 속성타입,  
...  
)
```

- 테이블 삭제 : DROP문

```
DROP TABLE 테이블명
```

■ DDL 문 : 데이터 정의문

- 테이블 구조 변경 : ALTER문

▶ 속성 추가

```
ALTER TABLE 테이블명(ADD 속성명 속성타입)
```

▶ 속성 제거

```
ALTER TABLE 테이블명 (DROP 속성명 속성타입)
```

▶ 속성 타입 변경

```
ALTER TABLE 테이블명 (ALTER 속성명 속성타입)
```

핵심요약

2. T-SQL

■ MS-SQL(MS-SQL Editions)

■ MS-SQL Server

- Microsoft에서 제공하는 데이터베이스 관리 시스템

■ Edition의 종류

- Express

- Workgroup

- Standard

- Enterprise

■ MS-SQL의 설치와 구동

■ MS-SQL 구동 - Database

- System Database

▶ master : 시스템 관리용

▶ model : 새 데이터베이스 생성을 위한 템플릿

▶ msdb : SQL Server agent를 위한 데이터베이스

▶ tempdb : 질의 임시결과 유지를 위한 테이블

- User Database

▶ SSMS이나 T-SQL(SQL of MS SQL)을 이용해서 생성



SQL 활용

데이터 구조 생성과 변경



한국기술교육대학교
온라인평생교육원

학습내용

- 테이블 생성
- 테이블 변경

학습목표

- 데이터베이스에 테이블을 생성하고 데이터를 저장할 수 있다.
- 테이블의 구조를 변경할 수 있다.

● 테이블 생성

1. 데이터베이스 생성

◆ 데이터베이스 - 밤상

- 테이블(그릇)을 올리기 전에 데이터베이스(밥상)부터 만들어야 함

◆ MS-SQL에서 데이터베이스 만들기

T-SQL (Transact-SQL)	SSMS (SQL Server Management Studio)
Text입력	GUI 이용

◆ T-SQL을 이용하여 데이터베이스 만들기

- DB 이름: test01
- 사용자 DB를 생성하려면, master DB를 사용해야 함
 - USE master
 - CREATE DATABASE test01

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer pane shows a tree structure with 'master' selected. In the center, the 'SQLQuery1.sql - PC.master (pc\jkmin (52))*' query window contains the following T-SQL code:

```
USE master
CREATE DATABASE test01
```

Below the query window, the 'Messages' pane displays the message: '명령이 완료되었습니다.' (The command has been completed successfully.). At the bottom of the screen, the status bar shows 'PC(12.0 RTM) | pc\jkmin (52) | master | 00:00:01 | 0개의 행'.

● 테이블 생성

1. 데이터베이스 생성

◆ SSMS를 이용하는 방법(GUI 이용)

- DB 이름: test02
- 논리적 DB → 물리적 파일
 - .mdf (for data), .ldf (for log), .ndf (for large DB -optional)



● 테이블 생성

2. 기본 데이터 타입

◆ DB(밥상)를 만들었으면 테이블(밥그릇)을 만들어야 함

◆ 어떤 모양의 테이블을 만들까?

- MS-SQL에서 제공하는 기본 속성 타입

◆ 숫자 타입

- bit : 1bit
- tinyint : 0~255 (정수)
- smallint : $-2^{15} \sim 2^{15}-1$ (정수)
- int : $-2^{31} \sim 2^{31}-1$ (정수)
- bigint : $-2^{63} \sim 2^{63}-1$ (정수)
- decimal[(p,s)][, numeric([p,s])] : decimal(5,2) → 123.45
- float : 4byte, 8byte float(실수)
- real : 4byte float(실수)
- datetime : 1755/1/1~9999/12/31 (8byte) (날짜)
- smalldatetime : 1990/01/01~2079/6/6 (8byte) (날짜)

◆ 날짜(시간) 타입

- datetime : 1755/1/1~9999/12/31 (8byte) (날짜)
- smalldatetime : 1990/01/01~2079/6/6 (4byte) (날짜)

◆ 문자 타입

- char[(n)] : 고정길이 문자열
- varchar[(n)] : 가변 길이 문자열
- nchar[(n)] : for unicode(2byte) → n → 2n bytes (유니코드를 위한 고정문자열)
- nvarchar[(n)] : for unicode(유니코드를 위한 가변길이문자열)

◆ IDENTITY

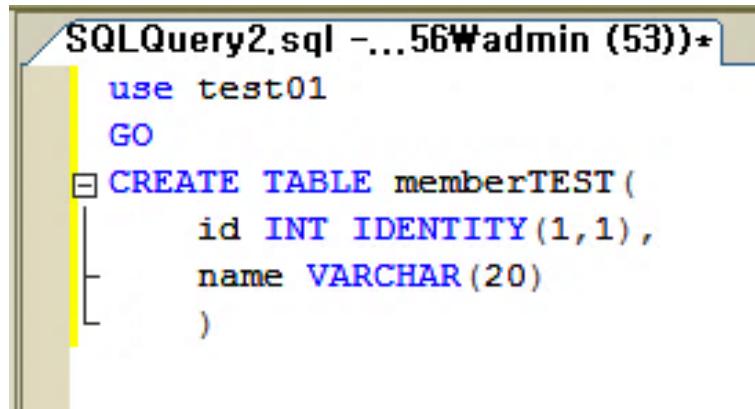
- 자동 증가 속성 타입
- IDENTITY(10, 2) → 초기값 10, 2씩 증가
 - 10, 12, 14, ...

● 테이블 생성

3. 테이블 생성과 튜플 추가

◆ 테이블의 생성

```
CREATE TABLE 테이블명  
(속성명 속성타입 [제약조건],  
 속성명 속성타입,  
 ...  
)
```



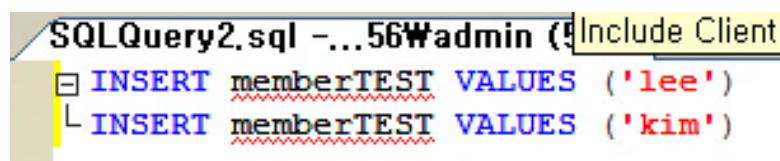
```
use test01
GO
CREATE TABLE memberTEST (
    id INT IDENTITY(1,1),
    name VARCHAR(20)
)
```

- test01 DB에 테이블을 만들 것이므로 use test01 기입

◆ 튜플 추가

```
INSERT INTO 테이블명(속성명, 속성명, ... )  
VALUES (속성값, 속성값, ... )
```

- INTO는 생략 가능



```
INSERT memberTEST VALUES ('lee')
INSERT memberTEST VALUES ('kim')
```

- id 속성은 자동 증가 타입임으로 값을 지정할 수 없음

● 테이블 생성

3. 테이블 생성과 튜플 추가

◆ 추가된 튜플의 검색

```
SELECT 속성명, 속성명, ...
  FROM 테이블명
 [ WHERE 조건 ]
```

The screenshot shows the SSMS interface with the following details:

- Query window title: SQLQuery2.sql - ...56Wadmin (53)*
- Query text: select * from memberTEST
- Results tab selected, showing the following table output:

	id	name
1	1	lee
2	2	kim

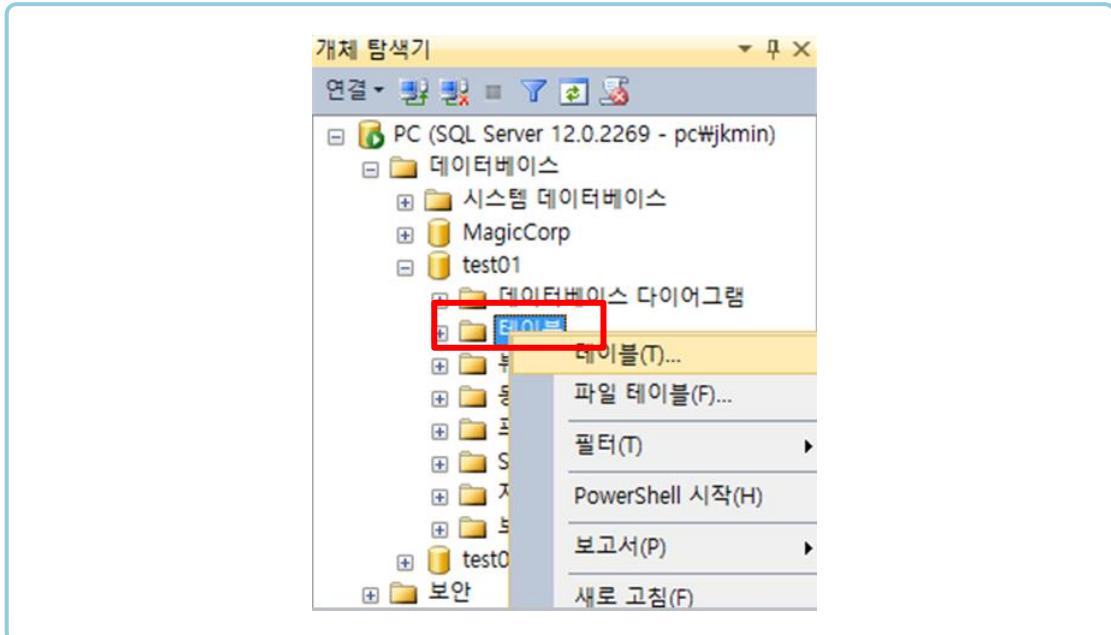
● 테이블 생성

3. 테이블 생성과 튜플 추가

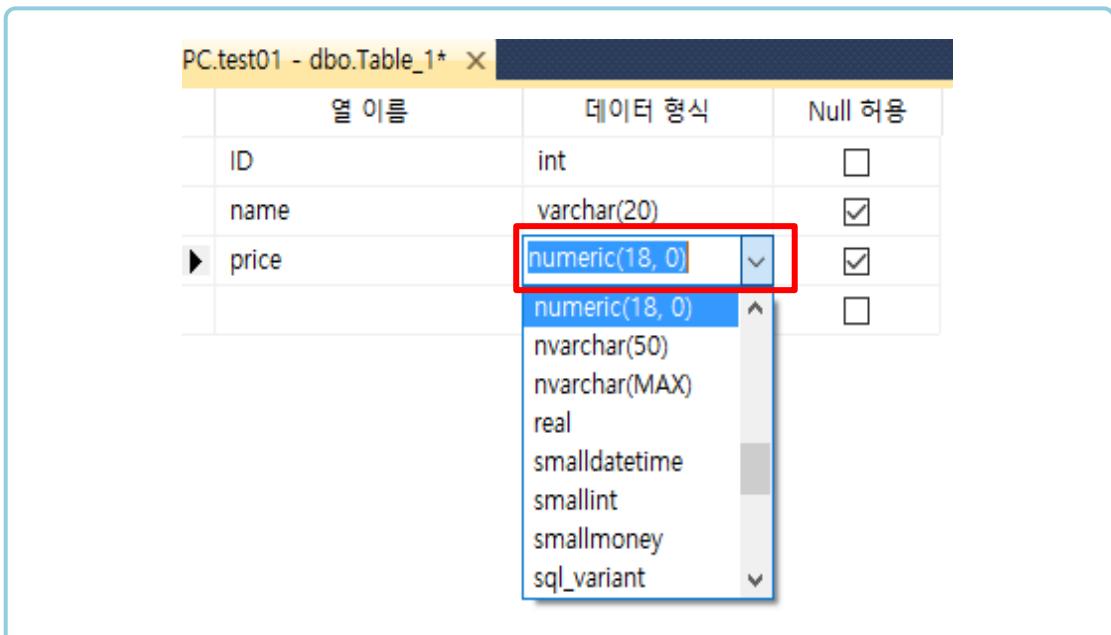
◆ 테이블의 생성

- SSMS를 이용한 테이블 만들기

① 테이블 선택(마우스 오른쪽 버튼 선택)



② 속성 이름과 타입 선택



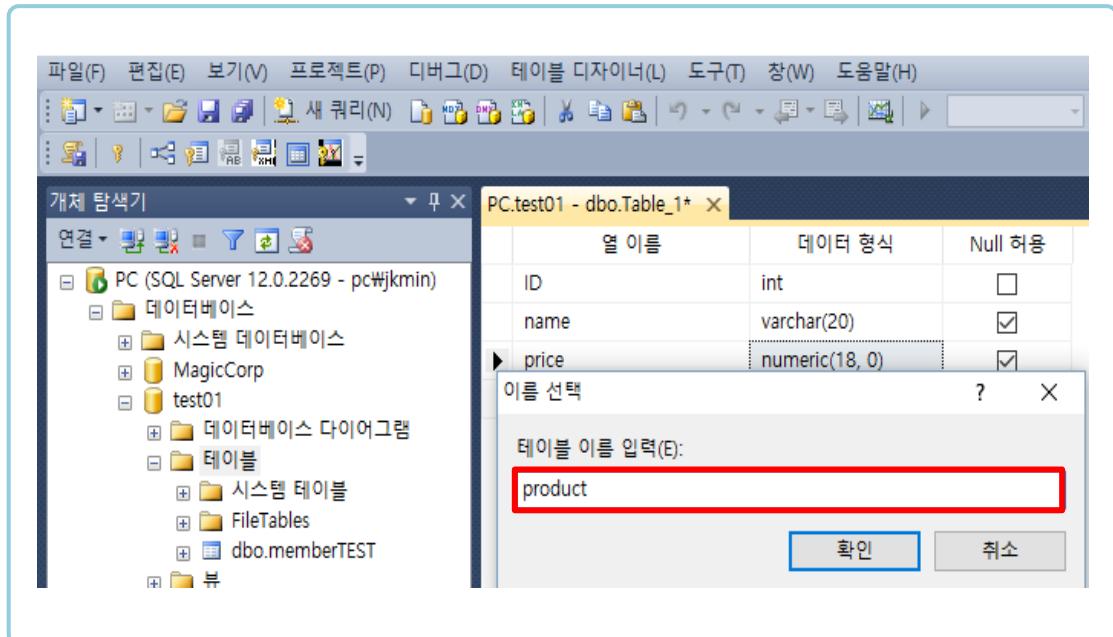
● 테이블 생성

3. 테이블 생성과 튜플 추가

◆ 테이블의 생성

- SSMS를 이용한 테이블 만들기

③ 테이블 이름 입력



● 테이블 변경

1. ADD, ALTER, DROP column

◆ 테이블의 구조 변경(ALTER TABLE)

- ADD column : 속성 추가
- ALTER column : 속성 타입 변경
- DROP column : 속성 제거

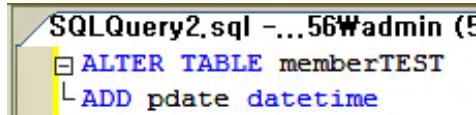
```
ALTER TABLE 테이블명{  
    {ADD|ALTER|DROP} [COLUMN] 속성명 [타입]  
}
```

● 주의 사항

- ALTER column의 경우 속성값의 범위를 증가 시키는 경우에는 문제가 없지만 범위를 감소시킬 경우에는 현재 테이블이 저장된 속성값들에 따라서 허용이 안될 수도 있음
 - 예) 이름 속성값으로 “김이름씨”, “박이름씨”, “이름이이빠요” 등이 저장되어 있을 때
- 속성 타입은 varchar(3)로 변경하려고 하면 안됨
 - 모두 6byte 이상임(한글 1자는 2byte)
- varchar(50)으로는 변경 가능함

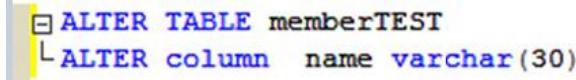
◆ ADD column : 속성 추가

- datetime형으로 pdate 속성을 memberTEST 테이블에 추가함



```
SQLQuery2.sql -... 56Wadmin (E  
ALTER TABLE memberTEST  
    ADD pdate datetime
```

- name 속성의 타입을 varchar(30)으로 변경함



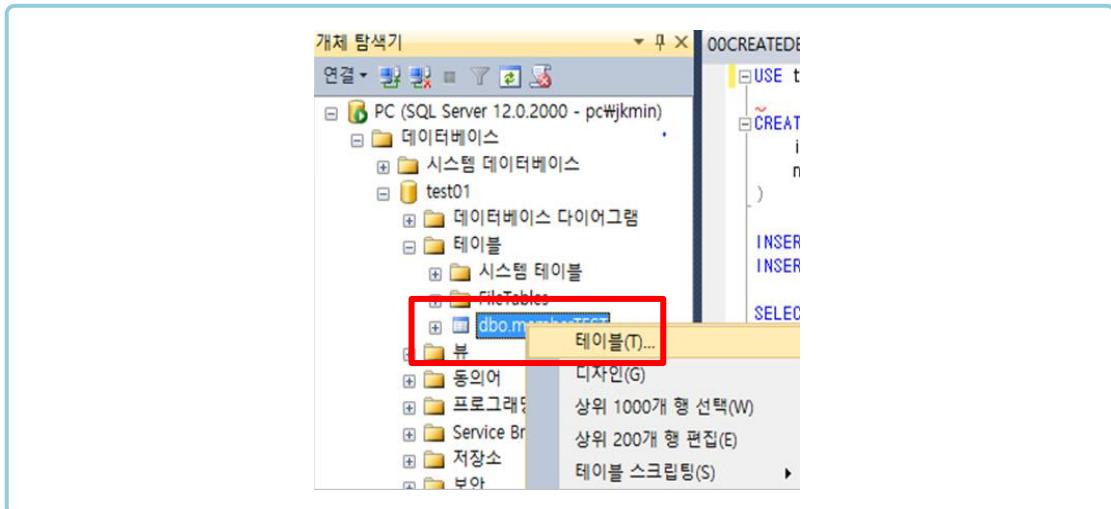
```
ALTER TABLE memberTEST  
    ALTER column name varchar(30)
```

● 테이블 변경

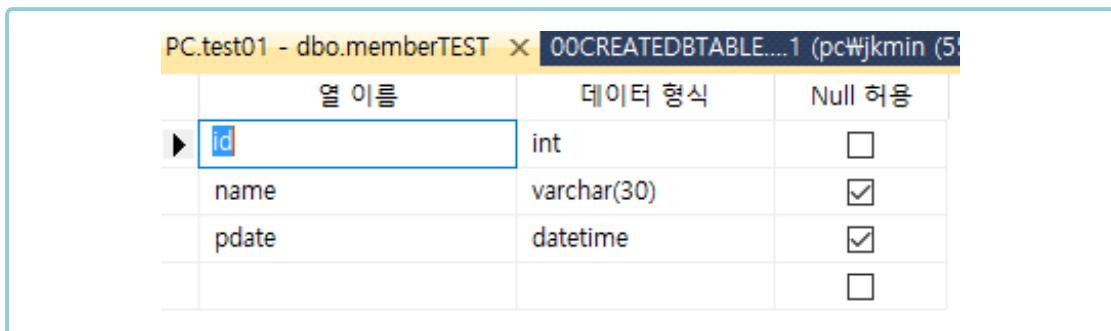
1. ADD, ALTER, DROP column

◆ SSMS에서 테이블의 구조를 보는 방법

① 테이블 선택(마우스 오른쪽 버튼 클릭)



② 디자인 선택



● 테이블 변경

2. Drop table과 TRUNCATE table

◆ 테이블을 지우기

```
DROP TABLE table_name
```

◆ 테이블의 모든 내용을 지우기, 단 테이블은 남기기

```
TRUNCATE TABLE table_name
```

◆ 주의 사항

- CREATE, ADD, ALTER, DROP, TRUNCATE 등은 모두 데이터 정의어 (DDL)임
- 명령문이 수행되고 나면 회복이 불가능함

핵심요약

1. 테이블 생성

■ T-SQL을 사용하는 방법

- DB 이름: test01
- 사용자 DB를 생성하려면, master DB를 사용해야 함

```
USE master  
CREATE DATABASE test01
```

■ drop 테이블과 truncate 테이블

- DB 이름: test02
- 논리적 DB → 물리적 파일

```
.mdf (for data), .ldf (for log),  
.ndf( for large DB -optional)
```

2. 테이블 변경

■ add, alter, drop 컬럼

- ADD column : 속성 추가
 - datetime형으로 pdate 속성을 memberTEST 테이블에 추가함
- alter column : 속성 추가
 - name 속성의 타입을 varchar(30)으로 변경함
- DROP column : 속성 제거

■ drop 테이블과 truncate 테이블

- 테이블을 지우기

```
DROP TABLE table_name
```

- 테이블의 모든 내용을 지우기, 단 테이블은 남기기

```
TRUNCATE TABLE table_name
```



SQL 활용

제약조건



한국기술교육대학교
온라인평생교육원

학습내용

- 제약조건
- 제약조건 변경

학습목표

- 테이블에 제약조건을 설정할 수 있다.
- 제약 조건을 변경할 수 있다.

● 제약조건

1. 데이터 무결성

◆ NOT NULL

- NULL 값 허용 불가

 예 학생 테이블에서 학생의 이름은 NULL값일 수 없음

◆ UNIQUE

- 하나의 테이블 내에서 한번만 나옴
- 주로 대체키 설정 시 사용됨

◆ PRIMARY KEY

- 기본키
- 의미 : UNIQUE + NOT NULL

◆ FOREIGN KEY

- 외래키

◆ CHECK

- 도메인 무결성

● 제약조건

2. 제약조건의 설정

- ◆ 테이블을 만들 때 속성에 제약조건 지정하기

```
CREATE TABLE 테이블명  
( 속성명 속성타입 [ [제약조건명] 제약조건],  
속성명 속성타입,  
...  
)
```



```
name varchar(20) NOT NULL  
id int CONSTRAINT PK_01 PRIMARY KEY
```

- ◆ 제약조건 설정 테이블 예제

```
③ CREATE TABLE customer(  
    id VARCHAR(20) CONSTRAINT PK_id PRIMARY KEY,  
    pwd VARCHAR(20) CONSTRAINT NN_pwd NOT NULL,  
    name VARCHAR(20) CONSTRAINT NN_name NOT NULL,  
    phone1 VARCHAR(3) NULL,  
    phone2 VARCHAR(8) NULL,  
    birthYear int NULL,  
    address VARCHAR(100) NULL  
)  
  
④ INSERT INTO customer  
VALUES ('one', '1111', 'Kim', '010', '77727777', 1988, 'Daejeon')  
  
⑤ INSERT INTO customer  
VALUES ('two', '2222', 'Lee', '010', '12324567', 1979, 'Seoul')
```

● 제약조건

2. 제약조건의 설정

◆ 제약조건 위반 튜플 삽입 예

```
INSERT INTO customer
VALUES(NULL, '2222', 'Lee', '010', '12324567', 1979, 'Seoul')

Msg 515, Level 16, State 2, Line 1
Cannot insert the value NULL into column 'id', table 'test01.dbo.customer';
The statement has been terminated.
```

◆ 참조 무결성 제약조건

- 외래키 값은 다른 테이블의 기본키 값을 중에 하나이어야 함

예) 사원 테이블의 부서번호는 부서 테이블의 기본키 값을 중 하나이어야 함

속성명 [CONSTRAINTS 제약조건명]
REFERENCE 참조테이블명(속성명)

◆ 참조 무결성 설정 및 삽입 오류의 예

```
CREATE TABLE orders (
    oseq      int IDENTITY(1,1) CONSTRAINT PK_oseq PRIMARY KEY,
    quantity  varchar(20) NULL,
    indate    datetime NULL,
    id        varchar(20) CONSTRAINT FK_id REFERENCES customer(id)
)

INSERT INTO orders(quantity, id) VALUES(5, 'ONE')
| 
SELECT * FROM orders

INSERT INTO orders(quantity, id) VALUES(5, 'test')

(1 row(s) affected)

(1 row(s) affected)
Msg 547, Level 16, State 0, Line 14
The INSERT statement conflicted with the FOREIGN KEY constraint "FK_id". The statement has been terminated.
```

● 제약조건

2. 제약조건의 설정

◆ Check 제약조건

- 도메인 무결성 제약조건

- 입력 값의 제한

예

```
birthYear int CHECK(birthYear >= 1900)
```

```
Age int CHECK(Age between 1 and 150)
```

◆ 테이블 수준 제약조건

- 속성 단위로 제약조건 설정은 표현에 있어서 제약이 따름
 - 복합키의 경우(여러 개의 속성이 합쳐져서 키가 됨) 등
- 구문

```
CREATE TABLE 테이블명(  
속성명 속성타입  
...  
[CONSTRAINT 제약조건명] 제약조건(속성명)  
)
```

◆ 테이블 수준 제약조건 설정의 예

```
CREATE TABLE customer3(  
    name VARCHAR(20),  
    phone VARCHAR(11),  
    birthday DATETIME,  
    address VARCHAR(100)  
CONSTRAINT customer3_COMBO_PK PRIMARY KEY(name, phone)  
)
```

Column Name	Data Type	Allow Nulls
name	varchar(20)	<input type="checkbox"/>
phone	varchar(11)	<input type="checkbox"/>
birthday	datetime	<input checked="" type="checkbox"/>
address	varchar(100)	<input checked="" type="checkbox"/>

● 제약조건 변경

1. 제약조건의 추가 및 제거

◆ 테이블을 생성한 후에 제약조건을 추가하거나 제거할 필요성이 있음

- 제약조건도 테이블의 구조 정보에 속함으로 ALTER TABLE을 이용함
- 추가

ADD CONSTRAINT

- 제거

DROP CONSTRAINT

◆ 제약조건 이름을 지정해 두어야 추가나 제거가 쉬움

ALTER TABLE 테이블명

ADD [CONSTRAINT 제약조건명] 제약조건 (속성명)

ALTER TABLE 테이블명

DROP CONSTRAINT 제약조건명

◆ 제약조건 변경 예

```
CREATE TABLE orders5 (
    oseq      int IDENTITY(1,1),
    quantity  varchar(20) NULL,
    indate    datetime NULL,
    id        varchar(20),
    pcode     varchar(20)
)

ALTER TABLE orders5
ADD CONSTRAINT PK_oseq5 PRIMARY KEY(oseq)

ALTER TABLE orders5
ADD CONSTRAINT FK_id5
FOREIGN KEY(id) REFERENCES customer(id)

ALTER TABLE orders5
DROP CONSTRAINT FK_id5
```

핵심요약

1. 제약조건

■ 데이터무결성

■ NOT NULL

- NULL 값 허용 불가

■ UNIQUE

- 하나의 테이블 내에서 한번만 나옴

- 주로 대체키 설정 시 사용됨

■ PRIMARY KEY

- 기본키(UNIQUE + NOT NULL)

■ FOREIGN KEY

- 외래키

■ CHECK

- 도메인 무결성

핵심요약

1. 제약조건

■ 제약조건의 설정

- 테이블을 만들 때 속성에 제약조건 지정하기

```
CREATE TABLE 테이블명  
( 속성명 속성타입 [ [제약조건명] 제약조건],  
속성명 속성타입,  
...  
)
```

- 참조 무결성 제약조건

- 외래키 같은 다른 테이블의 기본키 값들 중에 하나이어야 함

```
속성명 [CONSTRAINTS 제약조건명]  
REFERENCE 참조테이블명(속성명)
```

- Check 제약조건

- 도메인 무결성 제약조건 : 입력 값의 제한

```
속성명 [CONSTRAINTS 제약조건명]  
REFERENCE 참조테이블명(속성명)
```

- 테이블 수준 제약조건

- 속성 단위로 제약조건 설정은 표현에 있어서 제약이 따름

```
CREATE TABLE 테이블명(  
속성명 속성타입  
...  
[CONSTRAINT 제약조건명] 제약조건(속성명)  
)
```

핵심요약

2. 제약조건 변경

■ 제약조건의 추가 및 제거

- 테이블을 생성한 후에 제약조건을 추가하거나 제거할 필요성이 있음
- 제약조건도 테이블의 구조 정보에 속함으로 ALTER TABLE을 이용함
 - 추가 : ADD CONSTRAINT
 - 제거 : DROP CONSTRAINT
- 제약조건 이름을 지정해 두어야 추가나 제거가 쉬움

- 추가

```
ALTER TABLE 테이블명  
ADD [CONSTRAINT 제약조건명] 제약조건 (속성명)
```

- 제거

```
ALTER TABLE 테이블명  
DROP CONSTRAINT 제약조건명
```



SQL 활용

데이터 검색



한국기술교육대학교
온라인평생교육원

학습내용

- 간단한 데이터 검색
- 복잡한 데이터 검색

학습목표

- 테이블에 저장된 데이터를 검색할 수 있다.
- 다양한 검색 조건을 지정할 수 있다.

● 간단한 데이터 검색

1. AS 키워드와 *

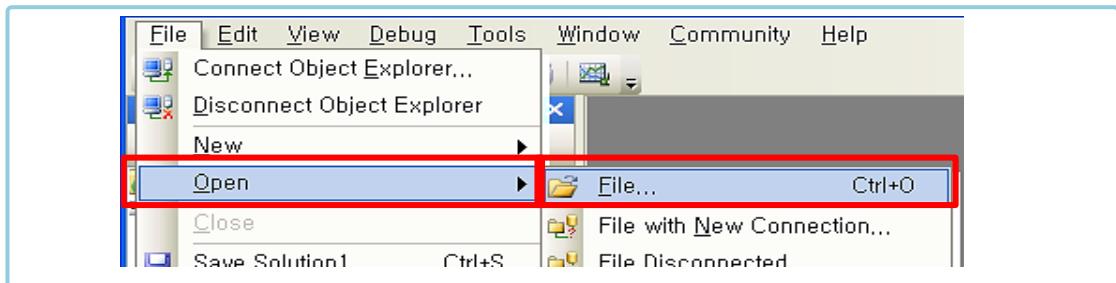
◆ 실습용 데이터 생성

- 실습을 위한 테이블 및 튜플들을 생성함

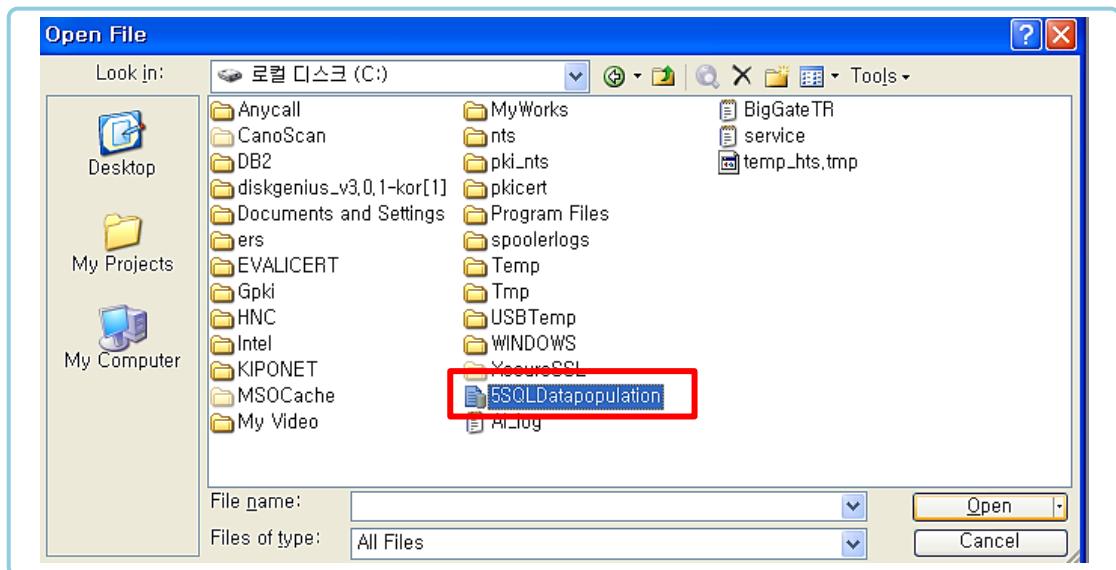
5SQLDatapopulate.sql

- DB 이름 : MagicCorp
- 테이블 : DEPARTMENT, EMPLOYEE, SALGRADE

① File ⇒ Open ⇒ File…



② 실습용 파일 선택

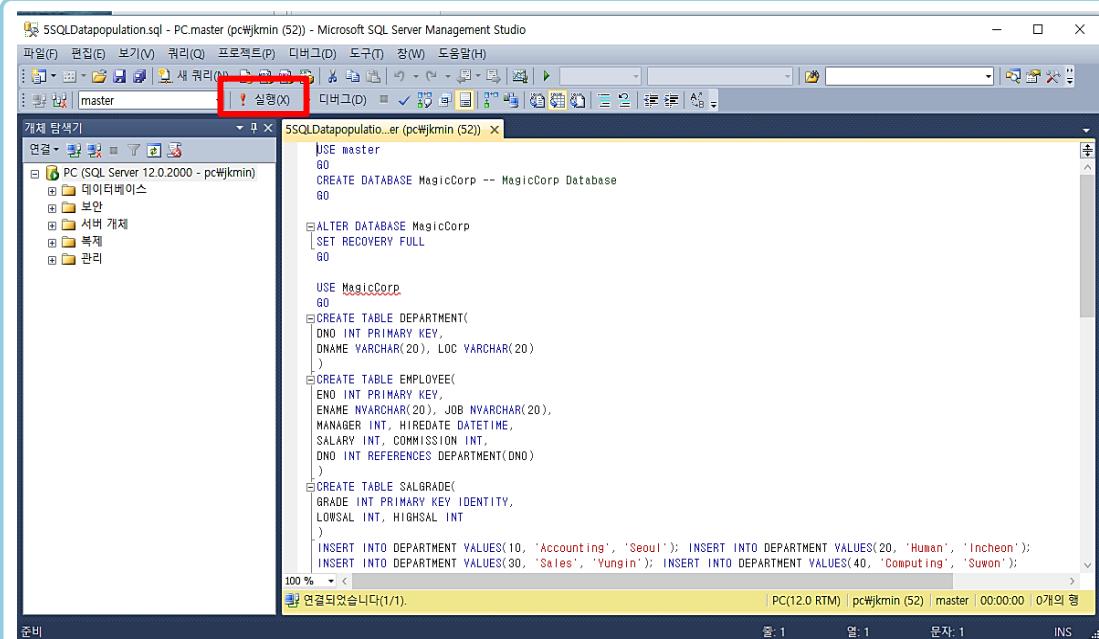


● 간단한 데이터 검색

1. AS 키워드와 *

◆ 실습용 데이터 생성

③ 실행



The screenshot shows the Microsoft SQL Server Management Studio interface. A red box highlights the '실행(X)' (Execute) button in the toolbar. The left pane shows the 'master' database context. The right pane displays a script named 'SSQLDatapopulation.sql' with the following content:

```
USE master
GO
CREATE DATABASE MagicCorp -- MagicCorp Database
GO

ALTER DATABASE MagicCorp
SET RECOVERY FULL
GO

USE MagicCorp
GO
CREATE TABLE DEPARTMENT(
    DNO INT PRIMARY KEY,
    DNAME VARCHAR(20), LOC VARCHAR(20)
)
CREATE TABLE EMPLOYEE(
    ENO INT PRIMARY KEY,
    ENAME NVARCHAR(20), JOB NVARCHAR(20),
    MANAGER INT, HIREDATE DATETIME,
    SALARY INT, COMMISSION INT,
    DNO INT REFERENCES DEPARTMENT(DNO)
)
CREATE TABLE SALGRADE(
    GRADE INT PRIMARY KEY IDENTITY,
    LWSAL INT, HIGHSAL INT
)
INSERT INTO DEPARTMENT VALUES(10, 'Accounting', 'Seoul'); INSERT INTO DEPARTMENT VALUES(20, 'Human', 'Incheon');
INSERT INTO DEPARTMENT VALUES(30, 'Sales', 'Yungin'); INSERT INTO DEPARTMENT VALUES(40, 'Computing', 'Suwon');
```

The status bar at the bottom indicates the connection is established (연결되었습니다(1/1)), the session ID is 1, the date and time are 00:00:00 | 07/08/2012, and there are 0 errors.

● 간단한 데이터 검색

1. AS 키워드와 *

◆ 무조건 검색

```
SELECT 속성명1, 속성명2, ...
FROM 테이블명
```

Q 모든 부서 정보 검색

```
USE MagicCorp
GO

SELECT DNO, DNAME, LOC FROM DEPARTMENT
```

Results

	DNO	DNAME	LOC
1	10	Accounting	Seoul
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

◆ *

- 모든 속성명을 쓰기 힘듦
 - “*” 를 사용함
 - SELECT절에서 *는 모든 속성이란 의미임

Q 모든 부서 정보 검색

```
07STARSQL.sql - J...ministrator (52)
USE MagicCorp
GO

SELECT * FROM DEPARTMENT
```

Results

	DNO	DNAME	LOC
1	10	Accounting	Seoul
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

● 간단한 데이터 검색

1. AS 키워드와 *

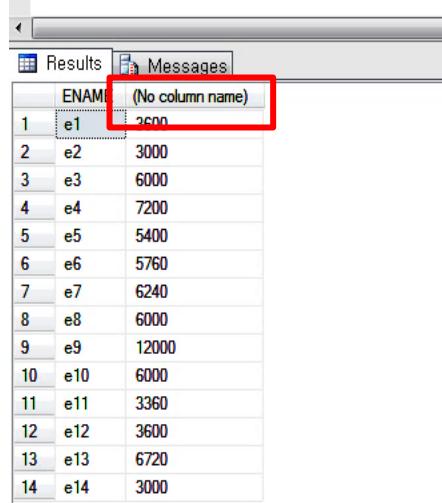
◆ AS 키워드

- ① 질의 결과의 속성명을 바꾸어서 나타나게 함
- ② 질의 결과에 수식을 넣을 수 있음
- ③ 속성명이 없음
- ④ AS 키워드로 속성명을 부여함

Q 사원 테이블에서 사원명과 봉급*12 검색

▪ 속성명이 없음

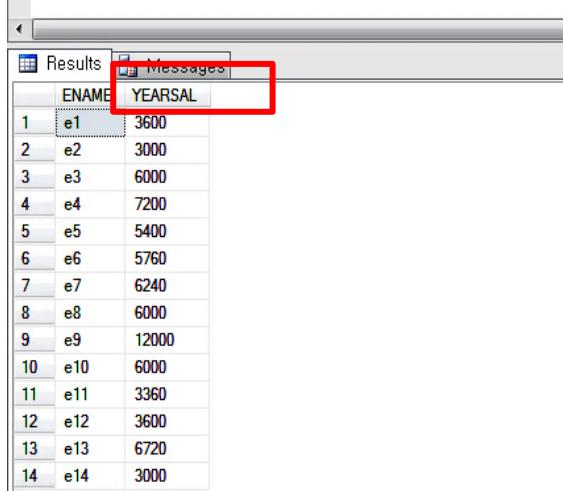
```
USE MagicCorp  
GO  
  
SELECT ENAME, SALARY*12 FROM EMPLOYEE
```



	ENAME (No column name)
1	e1 3600
2	e2 3000
3	e3 6000
4	e4 7200
5	e5 5400
6	e6 5760
7	e7 6240
8	e8 6000
9	e9 12000
10	e10 6000
11	e11 3360
12	e12 3600
13	e13 6720
14	e14 3000

▪ 속성명 생성

```
USE MagicCorp  
GO  
  
SELECT ENAME, SALARY*12 AS YEARSAL FROM EMPLOYEE
```



	ENAME	YEARSAL
1	e1	3600
2	e2	3000
3	e3	6000
4	e4	7200
5	e5	5400
6	e6	5760
7	e7	6240
8	e8	6000
9	e9	12000
10	e10	6000
11	e11	3360
12	e12	3600
13	e13	6720
14	e14	3000

● 간단한 데이터 검색

2. 간단한 조건 검색

◆ WHERE 절

- Q 사원 테이블(EMPLOYEE)에서 사원번호(ENO) 110번의 이름(ENAME)과 부서번호(DNO) 검색
- 조건 : 사원번호 110번
 - WHERE 절을 사용함

The screenshot shows a SQL query window with the following code:

```
USE MagicCorp
GO

SELECT ENAME, DNO
FROM EMPLOYEE
WHERE ENO = 110
```

Below the query window is a results grid with two columns: ENAME and DNO. There is one row of data with values 'e10' and '10' respectively.

◆ WHERE 절

WHERE 절에서 문자, 숫자, 날짜의 크기나 순서를 비교함

- 같다 : =
- 같지 않다 : !=, <>
- 크다 : >
- 크거나 같다 : >=
- 작다 : <
- 작거나 같다 : <=

◆ 논리 연산자

WHERE 절에서 여러 개의 조건을 결합할 경우

- X AND Y : X, Y가 참일 때 참을 반환
- X OR Y : X나 Y가 참일 때 참을 반환
- NOT X : X가 거짓일 때 참을 반환

● 간단한 데이터 검색

2. 간단한 조건 검색

Q 사원 테이블에서 부서번호(DNO)가 20번이고 봉급(SALARY)이 400이상인
사원의 이름(ENAME)과 직책(JOB) 검색

- 조건 : DNO = 20 AND SALARY >= 400

The screenshot shows a SQL query window with the following code:

```
USE MagicCorp
GO

SELECT ENAME, JOB FROM EMPLOYEE
WHERE DNO = 20 AND SALARY >= 400
```

Below the query window is a results grid titled "Results". The grid displays the following data:

	ENAME	JOB
1	e4	chief
2	e9	ceo
3	e13	chief

◆ DISTINCT

- SQL은 Bag 을 기반으로 함
 - 중복된 것들도 다 나옴
- 중복된 것을 제거하고 한번만 나오게 하는 방법
 - DISTINCT를 사용함



SELECT DISTINCT 속성명 ...

● 간단한 데이터 검색

2. 간단한 조건 검색

◆ DISTINCT

- SQL은 Bag 을 기반으로 함
 - 중복된 것들도 다 나옴
- 중복된 것을 제거하고 한번만 나오게 하는 방법
 - DISTINCT를 사용함

예

SELECT DISTINCT 속성명 ...



사원 테이블에서 모든 직급(JOB) 검색

```
USE MagicCorp  
GO  
  
SELECT JOB from EMPLOYEE
```

	JOB
1	staff
2	deputy
3	section
4	chief
5	section
6	chief
7	chief
8	senior
9	ceo



사원 테이블에서 모든 직급을 중복 없이 검색

```
USE MagicCorp  
GO  
  
SELECT DISTINCT JOB from EMPLOYEE
```

	JOB
1	ceo
2	chief
3	deputy
4	section
5	senior
6	staff

● 복잡한 데이터 검색

1. BETWEEN, IN, IS NULL

◆ BETWEEN a AND b

- 검색 조건의 상한과 하한을 지정함
 - 속성 X가 10보다 크거나 같고 50보다 작거나 같음
 - $X \geq 10 \text{ AND } X \leq 50$

X BETWEEN 10 AND 50

- Q 사원 테이블에서 봉급이 400보다 크거나 같고 600보다는 작거나 같은
사원들의 정보 검색

```
USE MagicCorp
GO

SELECT * FROM EMPLOYEE
WHERE SALARY BETWEEN 400 AND 600
```

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	103	e3	section	105	2005-02-10 00:00:00.000	500	100	30
2	104	e4	chief	108	2003-09-02 00:00:00.000	600	NULL	20
3	105	e5	section	105	2005-04-07 00:00:00.000	450	200	30
4	106	e6	chief	108	2003-10-09 00:00:00.000	480	NULL	30
5	107	e7	chief	108	2004-01-08 00:00:00.000	520	NULL	10
6	108	e8	senior	103	2004-03-08 00:00:00.000	500	0	30
7	110	e10	section	103	2005-04-07 00:00:00.000	500	NULL	10
8	113	e13	chief	103	2002-10-09 00:00:00.000	560	NULL	20

● 복잡한 데이터 검색

1. BETWEEN, IN, IS NULL

◆ IN(a, b, c, …)

- 속성값이 a, b, c, … 중 하나라도 일치하면 참
 - 속성 X가 10이거나 20이거나 30임
 - $X = 10 \text{ OR } X = 20 \text{ OR } X = 30$

X IN(10, 20, 30)



직급이 'section'이거나 'senior'인 사원들의 이름과 직급 검색

```
USE MagicCorp
GO

SELECT ENAME, JOB FROM EMPLOYEE
WHERE JOB IN ('section', 'senior')
```

	ENAME	JOB
1	e3	section
2	e5	section
3	e8	senior
4	e10	section

● 복잡한 데이터 검색

1. BETWEEN, IN, IS NULL

◆ IS NULL

- NULL 값은 어떤 비교를 하든 **거짓**임

- 사원 테이블에서 COMMISSION 값이 NULL인 튜플들이 있을 경우

The screenshot shows a SQL query window with the following content:

```
USE MagicCorp
GO

SELECT * FROM EMPLOYEE
```

The results grid displays the following data:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	101	e1	staff	113	2007-03-01 00:00:00.000	300	NULL	20
2	102	e2	deputy	105	2007-04-02 00:00:00.000	250	80	30
3	103	e3	section	105	2005-02-10 00:00:00.000	500	100	30
4	104	e4	chief	108	2003-09-02 00:00:00.000	600	NULL	20
5	105	e5	section	105	2005-04-07 00:00:00.000	450	200	30
6	106	e6	chief	108	2003-10-09 00:00:00.000	480	NULL	30

● 복잡한 데이터 검색

1. BETWEEN, IN, IS NULL

◆ IS NULL

- NULL 값은 어떤 비교를 하든 **거짓**임

- 사원 테이블에서 COMMISSION 값이 NULL인 튜플들이 있을 경우
 - X = NULL \Rightarrow X가 NULL 값이어도 이 결과는 거짓
 - 결과가 없음

The screenshot shows a SQL query window with the following content:

```
USE MagicCorp
GO

SELECT * FROM EMPLOYEE
WHERE COMMISSION = NULL
```

Below the query window is a results grid with the following columns: ENO, ENAME, JOB, MANAGER, HIREDATE, SALARY, COMMISSION, DNO. The results grid is currently empty, indicating no rows were found.

\Rightarrow IS NULL을 이용함

The screenshot shows a SQL query window with the following content:

```
USE MagicCorp
GO

SELECT * FROM EMPLOYEE
WHERE COMMISSION IS NULL
```

Below the query window is a results grid with the following columns: ENO, ENAME, JOB, MANAGER, HIREDATE, SALARY, COMMISSION, DNO. The results grid contains the following data:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	101	e1	staff	113	2007-03-01 00:00:00.000	300	NULL	20
2	104	e4	chief	108	2003-09-02 00:00:00.000	600	NULL	20
3	106	e6	chief	108	2003-10-09 00:00:00.000	480	NULL	30
4	107	e7	chief	108	2004-01-08 00:00:00.000	520	NULL	10
5	109	e9	ceo	NULL	1996-10-04 00:00:00.000	1000	NULL	20
6	110	e10	section	103	2005-04-07 00:00:00.000	500	NULL	10

● 복잡한 데이터 검색

1. BETWEEN, IN, IS NULL

◆ IS NULL

- NULL 값은 어떤 비교를 하든 **거짓**임
 - NULL 값이 아닌 것들을 찾는 방법
⇒ IS NOT NULL을 이용함

The screenshot shows a SQL query window and a results grid. The query is:

```
USE MagicCorp
GO

SELECT * FROM EMPLOYEE
WHERE COMMISSION IS NOT NULL
```

The results grid displays four rows of employee data where the Commission column is not null. The columns are ENO, ENAME, JOB, MANAGER, HIREDATE, SALARY, COMMISSION, and DNO.

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	102	e2	deputy	105	2007-04-02 00:00:00.000	250	80	30
2	103	e3	section	105	2005-02-10 00:00:00.000	500	100	30
3	105	e5	section	105	2005-04-07 00:00:00.000	450	200	30
4	108	e8	senior	103	2004-03-08 00:00:00.000	500	0	30

● 복잡한 데이터 검색

2. 문자열 검색

◆ LIKE 연산자

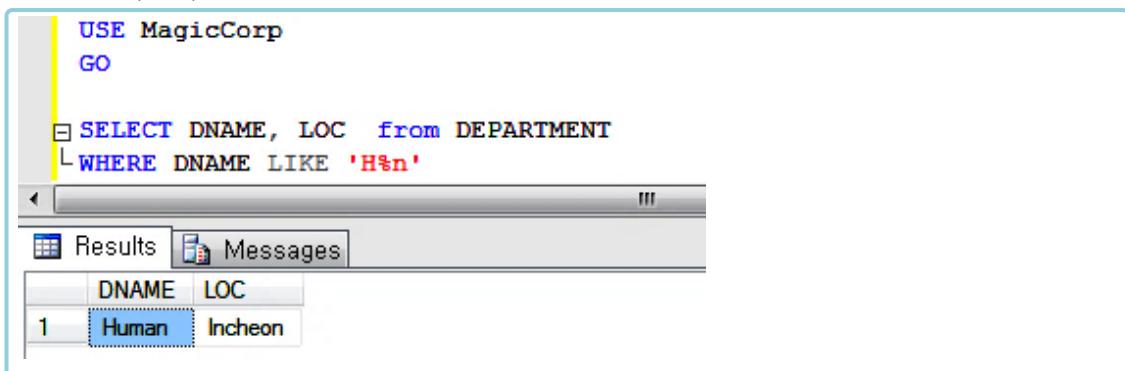
- 컬럼에 저장된 문자열 중에서 LIKE 연산자에서 지정한 문자 패턴과 부분적으로 일치하면 참이 되는 연산자

◆ 부분 문자열 검색에 사용되는 패턴

- % : 임의의 길이의 문자열
- _ : 글자 한자

◆ 부분 문자열 검색에 사용되는 패턴

- Q 부서 테이블에서 부서명(DNAME)이 H로 시작하고 n으로 끝나는 부서의 위치(Loc) 검색



The screenshot shows a SQL query window with the following content:

```
USE MagicCorp
GO

SELECT DNAME, LOC  from DEPARTMENT
WHERE DNAME LIKE 'H%n'
```

The results pane displays the following data:

	DNAME	LOC
1	Human	Incheon

● 복잡한 데이터 검색

2. 문자열 검색

◆ 질의문의 결과는 테이블에 입력된 순서대로 출력됨

- 데이터의 출력 순서를 특정 속성값을 기준으로 오름차순 또는 내림차순으로 정렬해야 하는 경우가 자주 발생함

The screenshot shows a SQL query interface with the following details:

- SQL Statement:

```
USE MagicCorp
GO

SELECT ENAME, SALARY, DNO from EMPLOYEE
```
- Results Window:
 - Tab: Results
 - Data Table:

	ENAME	SALARY	DNO
1	e1	300	20
2	e2	250	30
3	e3	500	30
4	e4	600	20
5	e5	450	30
6	e6	480	30
7	e7	520	10
8	e8	500	30

● 복잡한 데이터 검색

3. ORDER BY 절

◆ ORDER BY 절

ORDER BY {column_name} [ASC|DESC]

- ASC : 오름차순으로, 기본값(생략가능)
- DESC : 내림차순, 생략불가능

Q 봉급(Salary) 기준 내림차순으로 사원들의 이름, 봉급, 부서 번호 출력

```
SELECT ENAME, SALARY, DNO from EMPLOYEE
ORDER BY SALARY DESC
```

Results Messages

	ENAME	SALARY	DNO
1	e9	1000	20
2	e4	600	20
3	e13	560	20
4	e7	520	10
5	e8	500	30
6	e3	500	30
7	e10	500	10
8	e6	480	30

◆ 다중 속성 정렬

Q 봉급(Salary) 기준 내림차순으로 사원들의 이름, 봉급, 부서 번호 출력

- 봉급이 같은 경우에는 부서번호가 낮은 순으로(오름차순) 정렬함

```
USE MagicCorp
GO

SELECT ENAME, SALARY, DNO from EMPLOYEE
ORDER BY SALARY DESC, DNO ASC
```

여기서도 ASC 생략 가능

Results Messages

	ENAME	SALARY	DNO
1	e9	1000	20
2	e4	600	20
3	e13	560	20
4	e7	520	10
5	e10	500	10
6	e8	500	30

핵심요약

1. 간단한 데이터 검색

■ 제약조건의 설정

■ 무조건 검색

```
SELECT 속성명1, 속성명2, …  
FROM 테이블명
```

■ *

- 모든 속성명을 쓰기 힘들 경우 사용
- SELECT절에서 *는 모든 속성이란 의미임

■ AS 키워드

- AS 키워드로 속성명을 부여함

■ 간단한 조건 검색

■ WHERE 절

■ 비교 연산자

- 같다 : =
- 같지 않다 : !=, <>
- 크다 : >
- 크거나 같다 : >=
- 작다 : <
- 작거나 같다 : <=

■ 논리 연산자

- WHERE 절에서 여러 개의 조건을 결합할 경우
- X AND Y : X, Y가 참일 때 참을 반환
- X OR Y : X나 Y가 참일 때 참을 반환
- NOT X : X가 거짓일 때 참을 반환

핵심요약

1. 간단한 데이터 검색

■ DISTINCT

- SQL은 Bag을 기반으로 함
 - 중복된 것들도 다 제시 됨
- 중복된 것을 제거하고 한번만 나오게 하려면?
 - DISTINCT를 사용함

핵심요약

2. 복잡한 데이터 검색

■ BETWEEN, IN, IS NULL

■ BETWEEN a AND b

- 검색 조건의 상한과 하한을 지정함

■ IN(a, b, c,⋯)

- 속성값이 a, b, c ,⋯ 중 하나라도 일치하면 참

■ IS NULL

- NULL 값은 어떤 비교를 하든 거짓임

■ 문자열 검색

■ LIKE 연산자

- 컬럼에 저장된 문자열 중에서 LIKE 연산자에서 지정한 문자 패턴과 부분적으로 일치하면 참이 되는 연산자

■ ORDER BY 절

- 질의문의 결과는 테이블에 입력된 순서대로 출력

- 데이터의 출력 순서를 특정 속성값을 기준으로 오름차순 또는 내림차순으로 정렬해야 하는 경우가 자주 발생함

ORDER BY {column_name} [ASC|DESC]

▶ ASC : 오름차순으로, 기본값 (생략가능)

▶ DESC : 내림차순, 생략불가능



SQL 활용

데이터 삽입과 변경



한국기술교육대학교
온라인평생교육원

학습내용

- INSERT 절
- UPDATE와 DELETE

학습목표

- 다양한 INSERT 구문을 이용하여 데이터를 삽입할 수 있다.
- UPDATE와 DELETE 구문을 사용하여 저장된 데이터를 수정, 삭제할 수 있다.

● INSERT 절

1. 다양한 INSERT 구문

◆ 단일행 입력

한번에 하나의 튜플을 테이블에 입력하는 방법

```
INSERT INTO 테이블명 [(속성명, …, 속성명)]
VALUES (값, …, 값)
```

- “INTO”는 생략이 가능함
- 테이블명에 명시한 속성에 VALUES절에 지정한 값을 입력함
- 테이블명에 속성을 명시하지 않으면 테이블 생성시 정의한 컬럼 순서와 동일한 순서로 입력함



실습을 위하여 사원과 같은 구조의 테이블 EMPTEST 생성

```
USE master
GO
CREATE TABLE EMPTEST (
    ENO INT,
    ENAME NVARCHAR(20), JOB NVARCHAR(20),
    MANAGER INT, HIREDATE DATETIME,
    SALARY INT, COMMISSION INT,
    DNO INT
)
```

Messages

Command(s) completed successfully

- EMPTEST 테이블에 사원 정보 삽입
- 50, “홍길동”, “staff”, NULL, 2012-10-01, 500, 30, 10

```
USE MagicCorp
GO

INSERT INTO EMPTEST
VALUES (50, '홍길동', 'staff', NULL, '2012-10-01', 500, 30, 10)
```

Messages

(1 row(s) affected)

● INSERT 절

1. 다양한 INSERT 구문

◆ NULL의 입력

데이터를 입력하는 시점에서 해당 속성값을 모르거나, 미확정일 때 사용함

NOT NULL 조건이 지정된 경우 입력이 불가능함

● 묵시적인 방법

- INSERT INTO 절에 해당 속성명 생략

● 명시적인 방법

- VALUES 절에 있는 속성값에 NULL을 사용

- ① EMPTEST 테이블에 묵시적인 방법을 이용하여 사원 번호(51)와
사원이름('심청이') 입력, 나머지는 NULL 입력

```
USE MagicCorp
GO

INSERT INTO EMPTEST (ENO, ENAME)
VALUES (51, '심청이')

Messages
(1 row(s) affected)
```

- ② EMPTEST 테이블에 명시적인 방법을 이용하여 사원 번호(52)와
사원이름('임꺽정') 입력, 나머지는 NULL 입력

```
USE MagicCorp
GO

INSERT INTO empitest
VALUES (52, '임꺽정', NULL, NULL, NULL, NULL, NULL, NULL)

Messages
(1 row(s) affected)
```

● INSERT 절

1. 다양한 INSERT 구문

◆ 서브 쿼리를 이용한 데이터 삽입

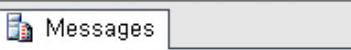
- 서브 쿼리의 결과를 테이블에 삽입함
 - 한번에 여러 튜플을 넣을 수 있음

```
INSERT INTO 테이블  
subquery
```

- 서브쿼리의 결과 집합은 INSERT 명령문에 지정된 칼럼 개수와 데이터 타입이 일치해야 함

Q 사원 테이블에서 부서번호 30인 사원들을 EMPTEST에 삽입

```
USE MagicCorp  
GO  
  
INSERT INTO empptest  
SELECT *  
FROM EMPLOYEE  
WHERE DNO = 30
```



(6 row(s) affected)

● INSERT 절

1. 다양한 INSERT 구문

◆ 질의 결과 테이블 만들기

- 질의 결과를 바로 테이블 만들고 저장하기

```
SELECT 컬럼리스트 INTO 대상테이블  
FROM 테이블  
WHERE 조건
```

- Q 부서 테이블에서 부서번호 30인 부서의 부서번호와 부서명을 DEPTEST란 테이블로 저장

```
USE MagicCorp  
GO  
  
SELECT DNO, DNAME INTO DEPTEST  
FROM DEPARTMENT  
WHERE DNO = 30
```

Messages
(1 row(s) affected)

- 오라클의 경우에는 구문이 다소 다름

```
CREATE TABLE 테이블명  
AS  
SELECT 컬럼리스트  
FROM 테이블  
[WHERE 조건]
```

- ① 질의 결과로 만든 테이블은 기존 테이블의 속성명과 타입을 그대로 적용함
- ② NOT NULL 조건을 그대로 적용함
- ③ 다른 제약조건은 적용되지 않음

● INSERT 절

1. 다양한 INSERT 구문

◆ 테이블 구조의 복사

- 상황에 따라서 기존 테이블과 동일한 구조를 지니는 테이블을 생성할 필요가 있음
 - 구조만 복사하고 튜플들은 복사하고 싶지 않은 경우
- SELECT ~ INTO ~ 구문을 이용함
- WHERE 조건에 항상 거짓이 되는 조건을 기술함

예

1 > 2

Q DEPARTMENT의 구조만 복사하여 DEPT_COPY 테이블 생성

The screenshot shows a database interface with a query window and a messages window.

Query Window:

```
SELECT * INTO DEPT_COPY
FROM DEPARTMENT
WHERE 1 > 2
```

Messages Window:

Messages

(0 row(s) affected)

● INSERT 절

1. 다양한 INSERT 구문

◆ 테이블의 구조 검색문

● 오라클

- 명령어 : DESCRIBE[DESC] 테이블명

● MS SQL

- 저장 프로시저 : sp_help 테이블명
- 저장 프로시저를 수행시키기 위하여 EXEC 명령어가 필요함

Q DEPTEST의 튜플들과 구조정보 검색

The screenshot shows the SSMS interface with a query window titled "SQLQuery1.sql - P...rp". The query is:

```
USE MagicCorp
GO

SELECT * FROM DEPTEST
EXEC sp_help DEPTEST
```

The results are displayed in three tabs: "결과" (Results), "메시지" (Messages), and "상세" (Details). The "결과" tab shows the data from the DEPTEST table:

	DNO	DNAME
1	30	Sales

The "Messages" tab shows the output of the EXEC sp_help command:

	Name	Owner	Type	Created_datetime
1	DEPTEST	dbo	user table	2016-05-03 23:29:03,363

The "Details" tab shows the column metadata for the DEPTEST table:

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	DNO	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	DNAME	varchar	no	20			yes	no	yes	Korean_Wansung_CI_AS

● INSERT 절

1. 다양한 INSERT 구문

◆ VALUES를 이용한 다중행 입력

- MS-SQL 2008 부터는 서브 쿼리가 아닌 VALUES를 이용해서도 다중행 삽입이 가능함

```
INSERT INTO table1  
VALUES (속성값들), (속성값들)
```

The screenshot shows a SQL query window with the following content:

```
INSERT INTO DEPT01  
VALUES (200, 'dept001', 'seoul'), (210, 'dept002', 'sangju')
```

Below the query window, there is a message pane labeled "Messages" which displays the result of the execution:

```
(2 row(s) affected)
```

● UPDATE와 DELETE

1. UPDATE 구문

◆ 데이터 수정

- UPDATE 문 : 테이블에 저장된 데이터를 수정하기 위한 데이터 조작어

UPDATE 테이블명

SET column = 값, ...

[WHERE 조건]

- WHERE 절이 생략되면 테이블의 모든 행이 수정됨

Q EMPTEST에 있는 모든 사원의 봉급 10% 인상

```
USE MagicCorp
GO

SELECT * FROM emptest

UPDATE emptest
SET SALARY = SALARY*1.1

SELECT * FROM emptest
```

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	NULL	2012-10-01 00:00:00.000	500	30	10
2	51	심청미	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	102	e2	deputy	105	2007-04-02 00:00:00.000	250	80	30
5	103	e3	section	105	2005-02-10 00:00:00.000	500	100	30
6	105	e5	section	105	2005-04-07 00:00:00.000	450	200	30
7	106	e6	chief	108	2003-10-09 00:00:00.000	480	NULL	30
8	108	e8	senior	103	2004-03-08 00:00:00.000	500	0	30
9	111	e11	staff	107	2007-03-01 00:00:00.000	280	NULL	30

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	NULL	2012-10-01 00:00:00.000	550	30	10
2	51	심청미	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	102	e2	deputy	105	2007-04-02 00:00:00.000	275	80	30
5	103	e3	section	105	2005-02-10 00:00:00.000	550	100	30
6	105	e5	section	105	2005-04-07 00:00:00.000	495	200	30
7	106	e6	chief	108	2003-10-09 00:00:00.000	528	NULL	30
8	108	e8	senior	103	2004-03-08 00:00:00.000	550	0	30
9	111	e11	staff	107	2007-03-01 00:00:00.000	308	NULL	30

● UPDATE와 DELETE

1. UPDATE 구문

◆ 데이터 수정

Q EMPTEST에 있는 사원들 중 30번 부서에 속한 사원들의 직급 모두 staff로 변경

```
USE MagicCorp
GO

SELECT * FROM empptest

UPDATE empptest
SET JOB = 'staff'
WHERE DNO = 30

SELECT * FROM empptest
```

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	NULL	2012-10-01 00:00:00.000	550	30	10
2	51	심청미	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	102	e2	deputy	105	2007-04-02 00:00:00.000	275	80	30
5	103	e3	section	105	2005-02-10 00:00:00.000	550	100	30
6	105	e5	section	105	2005-04-07 00:00:00.000	495	200	30
7	106	e6	chief	108	2003-10-09 00:00:00.000	528	NULL	30
8	108	e8	senior	103	2004-03-08 00:00:00.000	550	0	30

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	NULL	2012-10-01 00:00:00.000	550	30	10
2	51	심청미	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	102	e2	staff	105	2007-04-02 00:00:00.000	275	80	30
5	103	e3	staff	105	2005-02-10 00:00:00.000	550	100	30
6	105	e5	staff	105	2005-04-07 00:00:00.000	495	200	30
7	106	e6	staff	108	2003-10-09 00:00:00.000	528	NULL	30
8	108	e8	staff	103	2004-03-08 00:00:00.000	550	0	30
9	111	e11	staff	107	2007-03-01 00:00:00.000	308	NULL	30

● UPDATE와 DELETE

1. UPDATE 구문

◆ 서브 쿼리를 이용한 데이터 수정

- UPDATE문의 SET 절에서 서브 쿼리를 이용함
- 다른 테이블에 저장된 데이터를 검색하여 한번에 여러 속성값을 수정할 수 있음
- SET 절의 속성명은 서브 쿼리의 속성명과 달라도 됨

```
UPDATE table  
SET 속성1 = (SELECT ~ FROM ~ WHERE~)  
[WHERE 조건]
```

- Q EMPTEST 테이블에서 사원번호 50번인 사원의 관리자번호
⇒ 사원테이블의 사원번호 101번의 값으로 변경

```
USE MagicCorp  
GO  
  
UPDATE emptest  
SET MANAGER = (SELECT MANAGER FROM EMPLOYEE WHERE ENO = 101)  
WHERE ENO = 50  
  
SELECT *  
FROM emptest
```

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	113	2012-10-01 00:00:00.000	550	30	10
2	51	심청이	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	102	e2	staff	105	2007-04-02 00:00:00.000	275	80	30
5	103	e3	staff	105	2005-02-10 00:00:00.000	550	100	30
6	105	e5	staff	105	2005-04-07 00:00:00.000	495	200	30
7	106	e6	staff	108	2003-10-09 00:00:00.000	528	NULL	30
8	108	e8	staff	103	2004-03-08 00:00:00.000	550	0	30
9	111	e11	staff	107	2007-03-01 00:00:00.000	308	NULL	30

● UPDATE와 DELETE

1. UPDATE 구문

◆ 복수 속성값 변경

- 하나 이상의 속성값 한번에 변경하기
 - SET 절에 (속성명 = 값), (속성명= 값), … 으로 작성함

```
UPDATE table  
SET 속성1 = 값, 속성2 = 값, …  
[WHERE 조건]
```

Q EMPTEST에서 eno가 51인 튜플

⇒ SALARY를 300으로 변경

⇒ COMMISSION은 50으로 변경

```
SELECT * FROM emptest  
  
UPDATE emptest  
SET SALARY = 300, COMMISSION = 50  
WHERE eno = 51  
  
SELECT * FROM emptest
```

100 % <

결과 메시지

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	NULL	2012-10-01 00:00:00,000	550	30	10
2	51	심청미	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	102	e2	staff	105	2007-04-02 00:00:00,000	275	80	30
5	103	e3	staff	105	2005-02-10 00:00:00,000	550	100	30
6	105	e5	staff	105	2005-04-07 00:00:00,000	495	200	30
7	106	e6	staff	108	2003-10-09 00:00:00,000	528	NULL	30
8	108	e8	staff	103	2004-03-08 00:00:00,000	550	0	30

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	NULL	2012-10-01 00:00:00,000	550	30	10
2	51	심청미	NULL	NULL	NULL	300	50	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL

● UPDATE와 DELETE

2. DELETE 구문

◆ 데이터 삭제

- DELETE 문 : 테이블에 저장된 데이터 삭제를 위한 조작어

```
DELETE [FROM] 테이블  
[WHERE 조건]
```

- WHERE 절이 생략되면 테이블의 모든 행을 삭제함

Q DEPTEST에 있는 모든 정보를 삭제

The screenshot shows a SQL Server Management Studio (SSMS) interface. In the query window, the following T-SQL script is executed:

```
USE MagicCorp
GO

SELECT *
FROM DEPTEST

DELETE DEPTEST

SELECT *
FROM DEPTEST
```

The results grid displays the initial state of the DEPTEST table:

DNO	DNAME
1	Sales

After executing the DELETE statement, the table is empty, as shown by the second execution of the SELECT * statement.

● UPDATE와 DELETE

2. DELETE 구문

◆ 데이터 삭제

Q EMPTEST에서 급여가 400 미만인 사원 제거

```
USE MagicCorp
GO

SELECT *
FROM empptest

DELETE empptest
WHERE SALARY < 400

SELECT *
FROM empptest
```

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홍길동	staff	113	2012-10-01 00:00:00.000	550	30	10
2	51	심청미	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	102	e2	staff	105	2007-04-02 00:00:00.000	275	80	30
5	103	e3	staff	105	2005-02-10 00:00:00.000	550	100	30
6	105	e5	staff	105	2005-04-07 00:00:00.000	495	200	30
7	106	e6	staff	108	2003-10-09 00:00:00.000	528	NULL	30
8	108	e8	staff	103	2004-03-08 00:00:00.000	550	0	30
9	111	e11	staff	107	2007-03-01 00:00:00.000	308	NULL	30

● UPDATE와 DELETE

2. DELETE 구문

◆ 서브쿼리를 이용한 데이터 삭제

- WHERE 절에서 서브 쿼리를 이용함
- 다른 테이블에 저장된 데이터를 검색하여 한번에 여러 행을 삭제함
- WHERE 절의 속성명은 서브 쿼리의 속성명과 달라도 됨

Q 부서명이 Accounting인 부서에 속한 사원들을 EMPTEST 테이블에서 삭제하기

The screenshot shows a SQL query window with the following content:

```
USE MagicCorp
GO

SELECT *
FROM empptest

DELETE EMPTEST
WHERE DNO = (SELECT DNO FROM DEPARTMENT WHERE DNAME = 'Accounting')

SELECT *
FROM empptest
```

The Results tab displays the following data:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	50	홀길동	staff	113	2012-10-01 00:00:00.000	550	30	10
2	51	심청미	NULL	NULL	NULL	NULL	NULL	NULL
3	52	임꺽정	NULL	NULL	NULL	NULL	NULL	NULL
4	103	e3	staff	105	2005-02-10 00:00:00.000	550	100	30
5	105	e5	staff	105	2005-04-07 00:00:00.000	495	200	30
6	106	e6	staff	108	2003-10-09 00:00:00.000	528	NULL	30
7	108	e8	staff	103	2004-03-08 00:00:00.000	550	0	30

핵심요약

1. INSERT 절

■ 다양한 INSERT 구문

■ 단일행 입력

- 한번에 하나의 튜플을 테이블에 입력하는 방법

■ NULL의 입력

- 데이터를 입력하는 시점에서 해당 속성값을 모르거나, 미확정일 때 사용함
- NOT NULL 조건이 지정된 경우 입력이 불가능함

■ 서브 쿼리를 이용한 데이터 삽입

- 한번에 여러 튜플을 넣을 수 있음

■ 질의 결과 테이블 만들기

- 질의 결과로 만든 테이블은 기존 테이블의 속성명과 타입을 그대로 적용함
- NOT NULL 조건을 그대로 적용함
- 다른 제약조건은 적용되지 않음

■ 테이블 구조의 복사

- 상황에 따라서 기존 테이블과 동일한 구조를 지니는 테이블을 생성할 필요가 있음

■ 테이블의 구조 검색문

- 오라클

```
DESCRIBE[DESC] 테이블명
```

- MS SQL

```
sp_help 테이블명
```

■ VALUES를 이용한 다중행 입력

- MS-SQL 2008 부터는 서브 쿼리가 아닌 VALUES를 이용해서도 다중행 삽입이 가능함

핵심요약

2. UPDATE구문

■ 데이터 수정

■ 데이터 수정

- UPDATE 문은 테이블에 저장된 데이터를 수정하기 위한 데이터 조작어임

■ 서브 쿼리를 이용한 데이터 수정

- UPDATE문의 SET절에서 서브 쿼리를 이용함

- 다른 테이블에 저장된 데이터를 검색하여 한번에 여러 속성값을 수정할 수 있음

- SET절의 속성명의 서브 쿼리의 속성명과 달라도 됨

■ 복수 속성값 변경

- SET 절에 (속성명 = 값), (속성명= 값), … 으로 작성

■ 데이터 삭제

- DELETE 문의 테이블에 저장된 데이터 삭제를 위한 조작어

```
DELETE [FROM] 테이블  
[WHERE 조건]
```

■ 서브쿼리를 이용한 데이터 삭제

- WHERE 절에서 서브 쿼리를 이용함

- 다른 테이블에 저장된 데이터를 검색하여 한번에 여러 행을 삭제함

- WHERE 절의 속성명은 서브 쿼리의 속성명과 달라도 됨



SQL 활용

트랜잭션



한국기술교육대학교
온라인평생교육원

학습내용

- 트랜잭션
- 트랜잭션 제어문(TCL)

학습목표

- 트랜잭션을 이해하고 설명할 수 있다.
- 트랜잭션 제어문(TCL)을 이용하여 데이터를 원상태로 돌릴 수 있다.

● 트랜잭션

1. 트랜잭션의 개념

◆ 트랜잭션이란?

- 트랜잭션(Transaction) : 논리적인 일의 단위
- 기본 설정

하나의 SQL은 하나의 트랜잭션임

- 여러 개의 SQL문들이 합쳐져서 하나의 트랜잭션이 될 수도 있음

◆ 트랜잭션의 활용



항공기 예약



은행



신용 카드 처리



대형 할인점

대규모 데이터베이스를 수백, 수천 명 이상의 사용자들이 동시에 접근함

많은 사용자들이 동시에 데이터베이스의 서로 다른 부분 또는
동일한 부분을 접근하면서 데이터베이스를 사용함

⇒ 동시성

◆ 트랜잭션의 활용

● 동시성 제어

- 동시에 수행되는 트랜잭션들이 데이터베이스에 미치는 영향 = 이들을 순차적으로 수행하였을 때 데이터베이스에 미치는 영향과 같도록 보장함
- 다수 사용자가 데이터베이스를 동시에 접근하도록 허용하면서 데이터베이스의 일관성을 유지함
- 여러 사용자나 여러 응용 프로그램들이 동시에 수행되어도 서로 간섭하지 못하도록 보장함

⇒ 트랜잭션 단위, 동시성 제어

● 회복

- 데이터베이스를 갱신하는 도중에 시스템 고장 시에도 데이터베이스의 일관성을 유지함

⇒ 트랙잭션 단위 회복

● 트랜잭션

1. 트랜잭션의 개념

◆ 트랜잭션이 없다면?

① 은행 계좌 이자 증가

- 전체 계좌들에 대한 이자가 모두 계산되어야 함
- 만약 일부 계좌 이자만 증가되고 컴퓨터가 다운되었다가 재가동 된다면?
⇒ 처음부터 다시 계산하면 이중 이자 계산이 됨

② 다양한 예약 시스템

- 항공권, 극장 등의 예약 시스템
- 좌석을 선점하고 돈을 내기 전에 시스템이 다운됨
- 돈은 내지 않았지만 좌석을 잡았기 때문에 해당 좌석은 절대 다시 잡을 수 없어짐

③ 은행 계좌 이체 : A계좌에서 100원을 빼서 B계좌에 넣기

```
UPDATE ACCOUNT
```

```
SET BALANCE = BALANCE - 100
```

```
WHERE ID = A
```



```
UPDATE ACCOUNT
```

```
SET BALANCE = BALANCE + 100
```

```
WHERE ID = B
```

● 트랜잭션

1. 트랜잭션의 개념

◆ 트랜잭션이 없다면?

③ 은행 계좌 이체 : A계좌에서 100원을 빼서 B계좌에 넣기

- 계좌 이체 시 장애발생

```
UPDATE ACCOUNT  
SET BALANCE = BALANCE - 100  
WHERE ID = A
```



```
UPDATE ACCOUNT  
SET BALANCE = BALANCE + 100  
WHERE ID = B
```

- A 통장에서 돈만 빠져 나가고 B통장에 돈이 안 들어옴
- ⇒ 은행이 고객의 돈을 횡령한 것이 됨

- 횡령을 피하기 위해 먼저 B 계좌에 돈을 입금…

```
UPDATE ACCOUNT  
SET BALANCE = BALANCE + 100  
WHERE ID = B
```



```
UPDATE ACCOUNT  
SET BALANCE = BALANCE - 100  
WHERE ID = A
```

- B 통장에서 돈이 들어 왔는데 A 통장에 돈이 안 빠짐
- ⇒ 은행에 막대한 손실이 발생함

- 두 개의 SQL을 모아서 **하나의 트랜잭션**(계좌이체 업무)으로 관리함

두 DML문은 하나의 업무에 속한 작업들임

● 트랜잭션

2. 트랜잭션의 특성

◆ ACID

① Atomicity : 원자성

- 한 트랜잭션 내의 모든 연산들이 완전히 수행되거나 전혀 수행되지 않음(**All or Nothing**)을 의미함
- DBMS의 회복 모듈은 시스템이 다운되는 경우에, 부분적으로 데이터베이스를 갱신한 트랜잭션의 영향을 취소함으로써
- 트랜잭션의 원자성을 보장함
- 완료된 트랜잭션이 갱신한 사항은 트랜잭션의 영향을 재수행함으로써 트랜잭션의 원자성을 보장함

② Consistency : 일관성

- 어떤 트랜잭션이 수행되기 전에 데이터베이스가 일관된 상태를 가렸다면 트랜잭션이 수행된 후에 데이터베이스는 또 다른 일관된 상태를 가짐
- 트랜잭션이 수행되는 도중에는 데이터베이스가 일시적으로 일관된 상태를 갖지 않을 수 있음

③ Isolation : 격리성

- 고립성이라고도 함
- 한 트랜잭션이 데이터를 갱신하는 동안 이 트랜잭션이 완료되기 전에는 갱신 중인 데이터를 다른 트랜잭션들이 접근하지 못하도록 해야 함
- 다수의 트랜잭션들이 동시에 수행되더라도 그 결과는 어떤 순서에 따라 트랜잭션들을 하나씩 차례대로 수행한 결과와 같아야 함
- DBMS의 동시성 제어 모듈이 트랜잭션의 고립성을 보장함
- DBMS는 응용들의 요구사항에 따라 다양한 **고립 수준**(Isolation Level)을 제공함

④ Durability : 영속성

- 일단 한 트랜잭션이 완료되면 이 트랜잭션이 갱신한 것은 그 후에 시스템에 고장이 발생하더라도 손실되지 않음
- 완료된 트랜잭션의 효과는 시스템이 고장 난 경우에도 데이터베이스에 반영됨
- DBMS의 회복 모듈은 시스템이 다운되는 경우에도 트랜잭션의 지속성을 보장함

● 트랜잭션

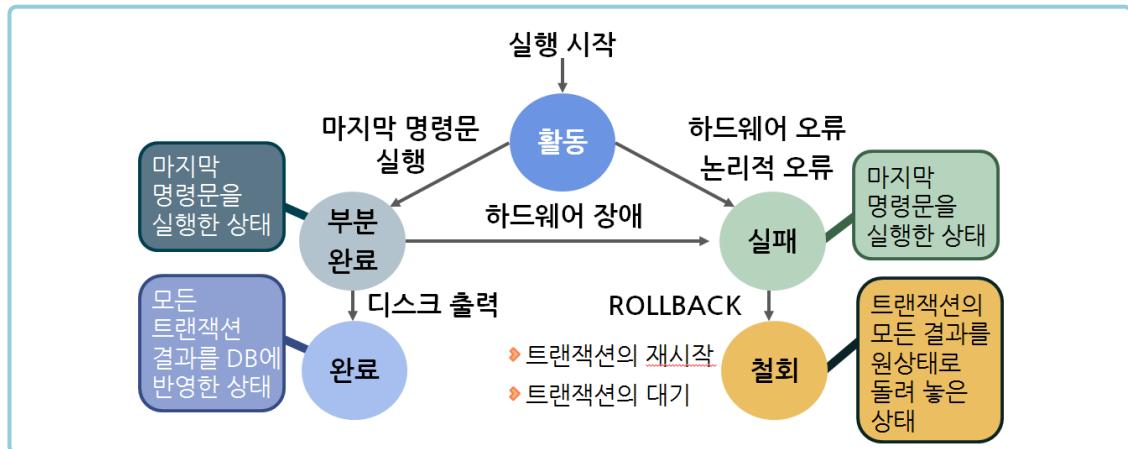
2. 트랜잭션의 특성

◆ ACID와 DB 기능

- ACID 특성과 DB의 기능은 모두 다 연관이 있음(연관성이 높은 순서)
 - DB 회복 기능 : 원자성과 지속성에 연관됨
 - DB 동시성 제어 : 일관성과 고립성에 연관
 - 무결성 제약 조건 : 일관성과 관련

3. 트랜잭션의 상태

◆ 트랜잭션의 상태 변화



● 트랜잭션 제어문(TCL)

1. 트랜잭션 제어문(TCL)

◆ COMMIT

- 트랜잭션의 마지막 명령어가 수행되었음을 나타냄
- 트랜잭션에 의한 변경을 확정
- COMMIT된 트랜잭션은 철회가 불가능함
- COMMIT 명령문 실행하기 전에 하나의 트랜잭션 변경한 결과를 다른 트랜잭션에서 접근할 수 없도록 방지하여 일관성을 유지함

◆ ROLLBACK

- 트랜잭션의 변경을 취소하고 트랜잭션 종료

◆ SAVEPOINT

- 현재 트랜잭션에서 ROLLBACK 시킬 위치 지정
- 대규모 트랜잭션(복수개의 명령어들로 이루진 트랜잭션)에서 오류 발생이 전체 트랜잭션을 취소시키는 것이 큰 부담이 될 수 있음
- 실패한 일정부분만 취소 시키도록 함

● 트랜잭션 제어문(TCL)

2. 트랜잭션 모드

◆ MS-SQL은 3가지의 트랙잭션 모드를 지원함

① 자동 커밋 트랜잭션

- 하나의 명령문이 하나의 트랙잭션이 됨
- MS-SQL에서 기본 모드임

② 명시적 트랜잭션

- 명시적으로 사용자가 트랙잭션을 정의하는 형태
- BEGIN TRAN ~ COMMIT TRAN(또는 ROLLBACK TRAN)으로 이루어짐

③ 묵시적 트랜잭션

- 자동 커밋 트랜잭션의 반대되는 개념
- 사용자가 COMMIT TRAN(또는 ROLLBACK TRAN)을 입력하기 전까지 복수 개의 명령문을 하나의 트랜잭션으로 간주함
- BEGIN TRAN이 필요 없음
- 묵시적 트랜잭션의 설정

```
SET IMPLICIT TRANSACTIONS {ON|OFF}
```

- 트랜잭션 종료마다 사용자가 반드시 COMMIT / ROLLBAK을 명령문을 실행시켜야 함
- 고급 사용자가 아닌 이상 가능한 사용하지 않는 것이 좋음

● 트랜잭션 제어문(TCL)

3. 트랜잭션 제어문(TCL) 활용

◆ 간단한 트랜잭션 철회

Q 실습을 위하여 DEPARTMENT 테이블 내용을 DEPT01로 복사하기

```
USE MagicCorp
GO

SELECT * INTO DEPT01 FROM DEPARTMENT
SELECT * FROM DEPT01
```

	DNO	DNAME	LOC
1	10	Accounting	Seoul
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

Q 트랜잭션을 시작한 후 DEPT01 테이블의 내용을 모두 지우고 트랜잭션 취소를 시켜보기

- ① 트랜잭션을 시작
- ② DEPT01 테이블 내용 지우기 ⇒ DEPT01 내용 보기
- ③ ROLLBACK ⇒ DEPT01 내용 보기

```
USE MagicCorp
GO

BEGIN TRAN
DELETE DEPT01
SELECT * FROM DEPT01
ROLLBACK TRAN
SELECT * FROM DEPT01
```

	DNO	DNAME	LOC
1	10	Accounting	Seoul
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

● 트랜잭션 제어문(TCL)

3. 트랜잭션 제어문(TCL) 활용

◆ 오류발생에 따른 트랜잭션 철회

- 트랜잭션을 구성하는 명령문들 중 : 오류 발생 \Rightarrow 트랜잭션 철회

오류 발생하지 않음 \Rightarrow 완료

- MS-SQL에서 명령문의 오류는 @@ERROR라는 변수에 저장됨
- T-SQL에서 IF ~ ELSE ~ 및 GOTO 같은 구문을 사용할 수 있음

- DEPT01 테이블의 DNO는 NULL값이 올 수 없음
 - 테이블 구조정보 보는 법 : EXEC sp_help 테이블명

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLength
1	DNO	int	no	4	10	0	no	(n/a)	(n/a)
2	DNAME	varchar	no	20			yes	no	yes
3	LOC	varchar	no	20			yes	no	yes

- 하나의 트랙잭션을 이용함

- ① DEPT01 테이블에서 부서번호 10번 튜플을 삭제함
- ② (NULL, 'PRODUCT', 'Seoul') 튜플을 삽입 \Rightarrow 오류 발생
- ③ 오류가 발생하면 해당 트랜잭션을 ROLLBACK함

```
USE MagicCorp
GO

BEGIN TRAN

DELETE FROM DEPT01 WHERE DNO = 10
SELECT * FROM DEPT01

INSERT INTO DEPT01 VALUES (NULL, 'PRODUCT', 'Seoul')

IF @@ERROR <> 0 GOTO ERROR_ROLLBACK

COMMIT TRAN
RETURN

ERROR_ROLLBACK:
ROLLBACK TRAN
GO

SELECT * FROM DEPT01
GO
```

	DNO	DNAME	LOC
1	20	Human	Incheon
2	30	Sales	Yungin
3	40	Computing	Suwon

	DNO	DNAME	LOC
1	10	Accounting	Seoul
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

● 트랜잭션 제어문(TCL)

3. 트랜잭션 제어문(TCL) 활용

◆ SAVEPOINT를 이용한 트랜잭션 부분 철회

- 트랜잭션 내에서 SAVEPOINT의 지정

SAVE TRAN 저장점명

- 트랜잭션 내에 저장점명을 다르게 하면 여러 개의 SAVEPOINT를 지정할 수 있음

- 저장점 위치로 취소

ROLLBACK TRAN 저장점명

- 하나의 트랙잭션을 이용함

① DEPT01 테이블에서 부서번호 10번 튜플을 삭제함

② 저장점 설정함

③ (50, 'PRODUCT', 'Seoul') 추가함

④ 저장점으로 ROLLBACK

⑤ (60, DESIGN, 'Jeju') 추가함

⇒ SAVEPOINT에 의하여 (50, 'PRODUCT', 'Seoul')을 삽입되지 않을 것을 알 수 있음

⇒ (60, 'DESIGN', 'Jeju')는 삽입되었음

```
USE MagicCorp
GO

BEGIN TRAN

DELETE FROM DEPT01 WHERE DNO = 10

SAVE TRAN svpoint1

INSERT INTO DEPT01 VALUES (50, 'PRODUCT', 'Seoul')

ROLLBACK TRAN svpoint1

INSERT INTO DEPT01 VALUES(60, 'DESIGN', 'Jeju')

COMMIT TRAN
GO

SELECT * FROM DEPT01
```

	DNO	DNAME	LOC
1	60	DESIGN	Jeju
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

핵심요약

1. 트랜잭션

■ 트랜잭션의 개념

- 트랜잭션(Transaction)
 - 논리적인 일의 단위
- 기본 설정 : 하나의 SQL은 하나의 트랜잭션이다
- 여러 개의 SQL문들이 합쳐서 하나의 트랜잭션이 될 수도 있음
- 트랜잭션의 활용
 - 동시성 제어 : 여러 사용자나 여러 응용 프로그램들이 동시에 수행되어도 서로 간섭하지 못하도록 보장함
 - 회복 : 데이터베이스를 갱신하는 도중에 시스템 고장 시에도 데이터베이스의 일관성을 유지함

■ 트랜잭션의 특성

- 원자성(Atomicity)
 - 한 트랜잭션 내의 모든 연산들이 완전히 수행되거나 전혀 수행되지 않음
- 일관성(Consistency)
 - 어떤 트랜잭션이 수행되기 전에 데이터베이스가 일관된 상태를 가졌다면 트랜잭션이 수행된 후에 데이터베이스는 또 다른 일관된 상태를 가짐
- 격리성(Isolation)
 - 한 트랜잭션이 데이터를 갱신하는 동안 이 트랜잭션이 완료되기 전에는 갱신 중인 데이터를 다른 트랜잭션들이 접근하지 못하도록 해야 함
- 영속성(Durability)
 - 일단 한 트랜잭션이 완료되면 이 트랜잭션이 갱신한 것은 그 후에 시스템에 고장이 발생하더라도 손실되지 않음

핵심요약

1. 트랜잭션

■ 트랜잭션의 상태

- 부분완료 : 마지막 명령문을 실행한 상태
- 완료 : 모든 트랜잭션 결과를 DB 에 반영한 상태
- 실패 : 트랜잭션의 실패
- 철회 : 트랜잭션의 모든 결과를 원상태로 돌려 놓은 상태

핵심요약

2. 트랜잭션 제어문(TCL)

■ 트랜잭션 제어문

■ COMMIT

- 트랜잭션의 마지막 명령어가 수행되었음을 나타냄

■ ROLLBACK

- 트랜잭션의 변경을 취소하고 트랜잭션 종료

■ SAVEPOINT

- 현재 트랜잭션에서 ROLLBACK 시킬 위치 지정

■ 트랜잭션 제어문(TCL) 활용

■ 간단한 트랜잭션 철회

■ 오류발생에 따른 트랜잭션 철회

- 트랜잭션을 구성하는 명령문들 중에서 오류가 발생되면 트랜잭션을 철회하고 그렇지 않으면 완료하는 것이 필요

■ SAVEPOINT를 이용한 트랜잭션 부분 철회

- 트랜잭션 내에서 SAVEPOINT의 지정

SAVE TRAN 저장점명

- 저장점 위치로 취소

ROLLBACK TRAN 저장점명



SQL 활용

조인 질의문



한국기술교육대학교
온라인평생교육원

학습내용

- 조인
- 다양한 조인 구문

학습목표

- 조인을 사용하여 두 개 이상의 테이블로부터 데이터를 조회하는 SQL문을 작성할 수 있다.
- 조인을 표현하는 다양한 SQL 문법을 설명할 수 있다.

● 조인

1. 조인의 개념

◆ 조인

- 하나의 SQL 질의문에 의해서 여러 테이블에 저장된 데이터를 한번에 조회할 수 있는 기능
- 두 개 이상의 테이블을 ‘결합’ 한다는 의미

◆ 조인의 필요성

하나의 SQL 질의문이 하나의 테이블만 검색할 수 있다고 하는 경우

Q 사번이 103인 사원의 부서명을 알고 싶을 때

- ① 사번이 103번인 사원의 부서 번호를 파악함
 - ② 해당 부서 번호와 같은 부서 번호를 가지고 부서명을 부서테이블에서 검색함
- ⇒ 매우 불편함

The screenshot shows two side-by-side SQL query windows. Both windows have the following code:

```
USE MagicCorp
GO

SELECT DNO
FROM EMPLOYEE
WHERE ENO = 103
```

The results window for the left query shows a single row with DNO value 30, which is highlighted with a red box.

The right window has the same initial code:

```
USE MagicCorp
GO

SELECT DNAME
FROM DEPARTMENT
```

But it also includes an additional WHERE clause at the bottom:

```
WHERE DNO = 30
```

This WHERE clause is also highlighted with a red box. An arrow points from the red box on the left window's results to the red box on the right window's WHERE clause, illustrating the redundancy and inconvenience of performing two separate queries without joins.

● 조인

2. 간단한 조인

◆ SQL에서 간단한 조인 표기법

- ① FROM 절에 조인에 참여하는 두 테이블을 기록함
 - 콤마(,)로 구분함
- ② WHERE 절에 조인 조건을 기술함

Q 사번이 103인 사원의 부서명을 알고 싶을 때

- 사원 정보 : EMPLOYEE
- 부서 정보 : DEPARTMENT
- 조인 조건

```
EMPLOYEE.DNO = DEPARTMENT.DNO
```

The screenshot shows a SQL query window in SSMS. The query is:

```
USE MagicCorp
GO

SELECT DNAME
FROM EMPLOYEE, DEPARTMENT
WHERE ENO= 103 and EMPLOYEE.DNO = DEPARTMENT.DNO
```

The results grid shows one row:

DNAME
1 Sales

● 조인

2. 간단한 조인

◆ 조인문 작성 시 유의 사항

● 컬럼 이름의 모호성

- 서로 다른 두 테이블의 컬럼(속성) 명이 같을 경우



DNO = DNO

⇒ DBMS에서 어느 속성이 어느 테이블에 있는 것인지 알 수 없어짐

The screenshot shows a SQL query window with the following code:

```
USE MagicCorp
GO

SELECT DNAME
FROM EMPLOYEE, DEPARTMENT
WHERE ENO= 103 and DNO = DNO
```

Below the query window is a 'Messages' pane displaying two error messages:

```
Msg 209, Level 16, State 1, Line 4
Ambiguous column name 'DNO'.
Msg 209, Level 16, State 1, Line 4
Ambiguous column name 'DNO'.
```

■ 해결방법

- 컬럼 이름 앞에 테이블 이름을 접두사로 사용함
- 테이블 이름과 컬럼 이름은 점(.)으로 구분함



DEPARTMENT.DNO = EMPLOYEE.DNO

● 테이블 별명

- 테이블의 이름이 긴 경우 SQL문 작성이 힘듦
 - 테이블 이름 대신 별명 사용이 가능함
 - FROM 절에 테이블 이름 다음에 공백을 두고 별명을 정의함

● 별명 작성 규칙

- 별명은 최대 30자까지 가능함
- 하나의 SQL에서 테이블 이름과 별명을 혼용해서 쓸 수 없음
- 테이블의 별명은 해당 SQL문에서만 유효함

● 조인

2. 간단한 조인

◆ 조인문 작성 시 유의 사항

Q 사번이 103인 사원의 부서명을 알고 싶을 때(별명 사용)

```
USE MagicCorp
GO

SELECT DNAME
FROM EMPLOYEE E, DEPARTMENT D
WHERE ENO= 103 and E.DNO = D.DNO
```

Results Messages

DNAME
1 Sales

- 별명을 정의한 후에는 별명만을 써야 함

```
USE MagicCorp
GO

SELECT DNAME
FROM EMPLOYEE E, DEPARTMENT D
WHERE ENO= 103 and E.DNO = DEPARTMENT.DNO
```

Messages

Msg 4104, Level 16, State 1, Line 4
The multi-part identifier "DEPARTMENT.DNO" could not be bound.

● 다양한 조인 구문

1. 다양한 조인들

◆ 카티잔 프로덕트(Cartesian Product : X)

- 두 테이블에 속한 튜플들의 모든 가능한 쌍을 생성함
- 일반적인 방법
 - FROM 절에 두 개 이상의 테이블명을 기록함
 - WHERE 절에는 조인 조건을 기술하지 아니함

예

```
SELECT * FROM EMPLOYEE, DEPARTMENT
```

The screenshot shows a SQL query window and its results. The query is:

```
USE MagicCorp
GO
select *
from employee, department
```

The results grid has 11 rows and 14 columns. The columns are labeled: ENO, ENAME, JOB, MANAGER, HIREDATE, SALARY, COMMISSION, DNO, DNO, DNAME, LOC. The data represents all possible combinations of employees from the 'employee' table and departments from the 'department' table.

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO	DNO	DNAME	LOC
1	101	e1	staff	1013	2007-03-01 00:00:00.000	300	NULL	20	10	Accounting	Sec
2	102	e2	deputy	1005	2007-04-02 00:00:00.000	250	80	30	10	Accounting	Sec
3	103	e3	section	1005	2005-02-10 00:00:00.000	500	100	30	10	Accounting	Sec
4	104	e4	chief	1008	2003-09-02 00:00:00.000	600	NULL	20	10	Accounting	Sec
5	105	e5	section	1005	2005-04-07 00:00:00.000	450	200	30	10	Accounting	Sec
6	106	e6	chief	1008	2003-10-09 00:00:00.000	480	NULL	30	10	Accounting	Sec
7	107	e7	chief	1008	2004-01-08 00:00:00.000	520	NULL	10	10	Accounting	Sec
8	108	e8	senior	1003	2004-03-08 00:00:00.000	500	0	30	10	Accounting	Sec
9	109	e9	ceo	NULL	1996-10-04 00:00:00.000	1000	NULL	20	10	Accounting	Sec
10	110	e10	section	1003	2005-04-07 00:00:00.000	500	NULL	10	10	Accounting	Sec
11	111	e11	staff	1007	2007-03-01 00:00:00.000	280	NULL	30	10	Accounting	Sec

◆ 동등 조인

조인 조건이 “=” 인 경우

- 조인 조건으로 일반적으로 “=”을 많이 씀

● 다양한 조인 구문

1. 다양한 조인들

◆ 자연 조인

조인 조건을 명시하지 않고 조인한다고 할 때
두 테이블에 공통으로 나타나는 속성의 동등 조인으로 생각함

- 동등 조인과 질의 결과의 구조(스키마)가 똑같지는 않음

- MS-SQL

- 자연 조인을 **명시적으로 지원하지는 않음**

- Oracle

- 자연 조인을 지원하며 FROM 절에 다음과 같이 씀

FROM 테이블명 NATURAL JOIN 테이블명

⇒ WHERE 절에 조인 조건을 쓰지 않음

◆ 쎄타 조인

조인 조건으로 <, >, <=, >=, != 등을 쓸 수 있음

- 일반적인 조인 조건에 대하여 쎄타 조인이라고 함

예 다른 사원의 봉급 보다 많은 봉급을 받는 사원들의 이름 찾기

```
USE MagicCorp
GO

SELECT E1.ENAME
FROM EMPLOYEE E1, EMPLOYEE E2
WHERE E1.SALARY > E2.SALARY
```

The screenshot shows a SQL query results window. The query selects employee names (ENAME) from the EMPLOYEE table where the salary is greater than another employee's salary. The results are displayed in a table with columns labeled ENAME and row numbers 1 through 10. The first row shows 'e3'.

	ENAME
1	e3
2	e4
3	e5
4	e6
5	e7
6	e8
7	e9
8	e10
9	e13
10	e1

● 다양한 조인 구문

1. 다양한 조인들

◆ 써타 조인

- 써타 조인의 예

Q SALGRADE 테이블에서 각 봉급의 하한과 상한에 따른 등급(Grade)이 정해져 있음

```
select *  
from SALGRADE
```

Results

	GRADE	LOWSAL	HIGHSAL
1	1	901	1000
2	2	501	900
3	3	401	500
4	4	301	400
5	5	201	300

Q 각 사원의 급여에 따라서 SALGRADE를 참조하여 사원의 등급 출력

- 조인 조건

lowsal <= salary and salary <= highsal

```
select ENAME, GRADE  
from EMPLOYEE, SALGRADE  
WHERE lowsal <= SALARY and SALARY <= HIGHSAL
```

Results

	ENAME	GRADE		ENAME	GRADE	
1	e9	1		8	e8	3
2	e4	2		9	e10	3
3	e7	2		10	e1	5
4	e13	2		11	e2	5
5	e3	3		12	e11	5
6	e5	3		13	e12	5
7	e6	3		14	e14	5

● 다양한 조인 구문

1. 다양한 조인들

◆ 씨타 조인

● 씨타 조인의 예

Q 각 사원의 급여에 따라서 SALGRADE를 참조하여 사원의 등급 출력

- SALGRADE의 lowsal과 highsال은 상한과 하한임
- between 연산자로 변환할 수 있음

```
select ENAME, GRADE
  from EMPLOYEE, SALGRADE
 WHERE SALARY between LOWSAL and HIGHSAL
```

	ENAME	GRADE		ENAME	GRADE	
1	e9	1		8	e8	3
2	e4	2		9	e10	3
3	e7	2		10	e1	5
4	e13	2		11	e2	5
5	e3	3		12	e11	5
6	e5	3		13	e12	5
7	e6	3		14	e14	5

● 다양한 조인 구문

1. 다양한 조인들

◆ 셀프 조인

하나의 테이블 내에 있는 컬럼끼리 연관시켜 조인이 필요한 경우

- Q 각 사원의 이름과 그 사원의 관리자 이름 검색

The screenshot shows the SQL query window with the following code:

```
USE MagicCorp
GO
select*
from employee
```

The results grid displays 10 rows of data from the EMPLOYEE table. The columns are ENO, ENAME, JOB, MANAGER, and HIREDATE. The MANAGER column contains values such as 1013, 1005, etc., which correspond to the ENO values in the same row. Row 10 (ENO 110) has a NULL value in the MANAGER column.

	ENO	ENAME	JOB	MANAGER	HIREDATE
1	101	e1	staff	1013	2007-03-01
2	102	e2	deputy	1005	2007-04-02
3	103	e3	section	1005	2005-02-10
4	104	e4	chief	1008	2003-09-02
5	105	e5	section	1005	2005-04-07
6	106	e6	chief	1008	2003-10-09
7	107	e7	chief	1008	2004-01-08
8	108	e8	senior	1003	2004-03-08
9	109	e9	ceo	NULL	1996-10-04
10	110	e10	section	1003	2005-04-07

- 사원 이름 : EMPLOYEE.NAME
- 각 사원의 관리자 번호 : EMPLOYEE.MANAGER
- 관리자 이름 : EMPLOYEE.NAME
- 사원의 관리자 또한 사원임
- 조인 대상 테이블이 두 개인데 동일한 테이블임
 - 물리적으로 1개임
 - 논리적으로 서로 다른 테이블이라고 생각하면 됨
⇒ 이외에는 다른 조인과 똑같음
- 어떻게 서로 다른 테이블이라고 SQL로 할까?
 - 별명을 이용함

● 다양한 조인 구문

1. 다양한 조인들

◆ 셀프 조인

Q 각 사원의 이름과 그 사원의 관리자 이름 검색

```
SELECT E.ENAME as [employeeName], M.ENAME as [managerName]
FROM EMPLOYEE E, EMPLOYEE M
WHERE E.MANAGER = M.ENO
```

The screenshot shows a database query results window. At the top, there are tabs for 'Results' and 'Messages'. The 'Results' tab is selected, displaying a table with two columns: 'employeeName' and 'managerName'. The table has 13 rows, indexed from 1 to 13. Row 1 is currently selected, highlighting the 'employeeName' cell for 'e1'. The data is as follows:

	employeeName	managerName
1	e1	e13
2	e2	e5
3	e3	e5
4	e4	e8
5	e5	e5
6	e6	e8
7	e7	e8
8	e8	e3
9	e10	e3
10	e11	e7
11	e12	e6
12	e13	e3
13	e14	e6

● 다양한 조인 구문

1. 다양한 조인들

◆ 다중 조인

● 조인 질의의 경우

- 조인에 참여하는 테이블이 2개 \Rightarrow 보통 '2중 조인'(2-way join)이라고 함

● 다중 조인(m-way join)

- 경우에 따라서 여러 테이블들 간의 조인이 필요한 경우도 있음

Q 쎈타 조인의 예제를 확장하여 각 사원별 이름, 급여 등급, 그리고 부서명 검색

- 사원 테이블의 부서 번호를 통해 부서 테이블을 참조하여 부서명을 가지고 와야 함
- 사원 테이블의 급여로부터 급여등급 테이블의 등급을 가지고 와야 함
 \Rightarrow 3개의 테이블을 조인해야 함(3중 조인)
- 이름 : employee 테이블
- 급여등급 : salgrade 테이블
- 부서명 : department 테이블
- 조인조건

employee와 salgrade

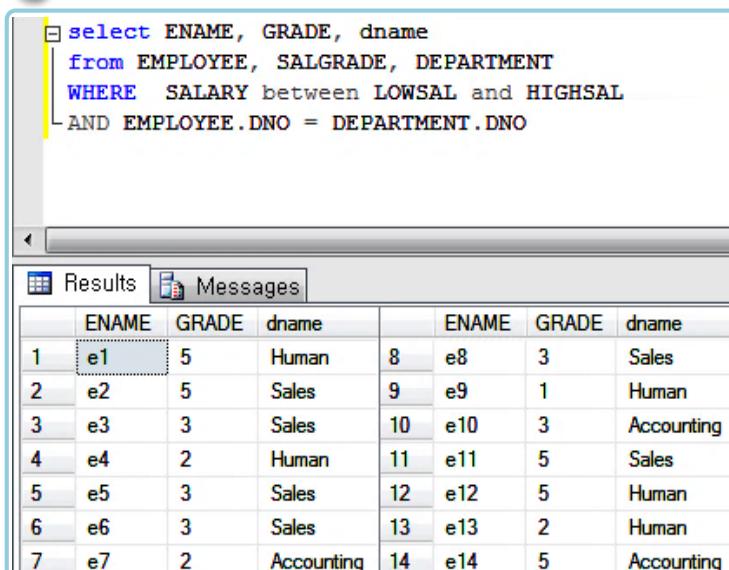
salary between lowsal and highsal

employee와 department

dno = dno

Q 사원별 이름, 급여등급, 부서명 출력

```
select ENAME, GRADE, dname
  from EMPLOYEE, SALGRADE, DEPARTMENT
 WHERE SALARY between LOWSAL and HIGHSAL
 AND EMPLOYEE.DNO = DEPARTMENT.DNO
```



The screenshot shows the Oracle SQL Developer interface. At the top, there is a code editor window containing the SQL query. Below it is a results grid showing the output of the query. The results grid has two sections: 'Results' and 'Messages'. The 'Results' section displays the following data:

	ENAME	GRADE	dname		ENAME	GRADE	dname
1	e1	5	Human	8	e8	3	Sales
2	e2	5	Sales	9	e9	1	Human
3	e3	3	Sales	10	e10	3	Accounting
4	e4	2	Human	11	e11	5	Sales
5	e5	3	Sales	12	e12	5	Human
6	e6	3	Sales	13	e13	2	Human
7	e7	2	Accounting	14	e14	5	Accounting

● 다양한 조인 구문

1. 다양한 조인들

◆ ANSI 조인

SQL을 표준화할 때 만든 ANSI 표준 문법

- 기존 SQL과 차이점
 - 조인 조건을 WHERE로 표현하지 않고 FROM 절에 표현함
- ANSI 조인의 종류
 - CROSS JOIN
 - INNER JOIN
 - OUTER JOIN

⇒ T-SQL 문법과 다소 상이해서 지원되지 않는 부분도 있음
- 크로스 조인(Cross Join)
 - 카티샨 프로덕트(Cartesian Product)의 다른 표현 법

FROM 테이블명 CROSS JOIN 테이블명

```
USE MagicCorp
GO

SELECT EMPLOYEE.ENAME, DEPARTMENT.DNAME
FROM EMPLOYEE CROSS JOIN DEPARTMENT
```

The screenshot shows a SQL query results window. At the top, there is a command line interface with the following SQL code:

```
USE MagicCorp
GO

SELECT EMPLOYEE.ENAME, DEPARTMENT.DNAME
FROM EMPLOYEE CROSS JOIN DEPARTMENT
```

Below the command line, there are two tabs: "Results" and "Messages". The "Results" tab is selected and displays the query results in a grid format. The grid has two columns: ENAME and DNAME. The data consists of 15 rows, where the first 14 rows have ENAME values from 1 to 14 and DNAME value 'Accounting'. The last row (row 15) has ENAME value 'e1' and DNAME value 'Human'. The grid has 15 columns, corresponding to the 14 rows from the first table plus one column for the second table.

	ENAME	DNAME		ENAME	DNAME
1	e1	Accounting	9	e9	Accounting
2	e2	Accounting	10	e10	Accounting
3	e3	Accounting	11	e11	Accounting
4	e4	Accounting	12	e12	Accounting
5	e5	Accounting	13	e13	Accounting
6	e6	Accounting	14	e14	Accounting
7	e7	Accounting	15	e1	Human
8	e8	Accounting			

● 다양한 조인 구문

1. 다양한 조인들

◆ ANSI 조인

- 내부 조인(Inner Join)

- 일반적인 조건의 Ansi 조인 표기법

FROM 테이블명 INNER JOIN 테이블명 ON 조인조건

FROM 테이블명 JOIN 테이블명 ON 조인조건 INNER 생략 가능

```
USE MagicCorp
GO

select ename, dname
from employee INNER JOIN department ON (employee.dno = department.dno )
```

Results Messages

	ename	dname
1	e1	Human
2	e2	Sales
3	e3	Sales
4	e4	Human
5	e5	Sales
6	e6	Sales
7	e7	Accounting
8	e8	Sales
9	e9	Human
10	e10	Accounting
11	e11	Sales
12	e12	Human
13	e13	Human
14	e14	Accounting

● 다양한 조인 구문

1. 다양한 조인들

◆ ANSI 조인

- 내부 조인(Inner Join)

```
USE MagicCorp
GO

select ename, dname
from employee JOIN department ON employee.dno = department.dno
```

The screenshot shows the SQL Server Management Studio interface with the 'Results' tab selected. The results grid displays the output of the provided SQL query, which joins the 'employee' and 'department' tables on their common 'dno' column. The columns are labeled 'ename' and 'dname'. The data shows 14 rows, each containing an employee name and their corresponding department name.

	ename	dname
1	e1	Human
2	e2	Sales
3	e3	Sales
4	e4	Human
5	e5	Sales
6	e6	Sales
7	e7	Accounting
8	e8	Sales
9	e9	Human
10	e10	Accounting
11	e11	Sales
12	e12	Human
13	e13	Human
14	e14	Accounting

● 다양한 조인 구문

1. 다양한 조인들

◆ 외부 조인

- 일반적인 조인은 조인 조건을 만족하는 튜플들만이 조인 결과에 나옴
⇒ 정보의 손실
- 조인 조건을 만족하지 않는 튜플들도 결과로 보고 싶은 경우
⇒ 조인에 참여하는 테이블에 속한 모든 튜플을 출력함
- ANSI 조인 표기법 : 명시적 표기법
 - LEFT OUTER JOIN : 왼쪽 테이블에 있는 튜플들은 다 나옴
 - RIGHT OUTER JOIN : 오른쪽 테이블에 있는 튜플들은 다 나옴
 - FULL OUTER JOIN : 양쪽 테이블에 있는 튜플들은 다 나옴



각 사원의 이름과 그 사원의 관리자 이름 검색

- 관리자가 없는 사원들도 질의 결과에 포함되어야 함

```
USE MagicCorp
select member.ename as [empolyname], manager.ename as [managename]
from employee member LEFT OUTER JOIN employee manager
ON member.manager = manager.eno
```

The screenshot shows the SQL Server Management Studio interface with the 'Results' tab selected. The results grid displays two columns: 'empolyname' and 'managename'. The data consists of 14 rows, indexed from 1 to 14. Row 9 is highlighted with a blue selection box, showing 'e9' in the first column and 'NULL' in the second column, demonstrating that employees without managers are included in the result set.

	empolyname	managename
1	e1	e13
2	e2	e5
3	e3	e5
4	e4	e8
5	e5	e5
6	e6	e8
7	e7	e8
8	e8	e3
9	e9	NULL
10	e10	e3
11	e11	e7
12	e12	e6
13	e13	e3
14	e14	e6

핵심요약

1. 조인

■ 조인의 개념

- 하나의 SQL 질의문에 의해서 여러 테이블에 저장된 데이터를 한번에 조회할 수 있는 기능
- 두 개 이상의 테이블을 ‘결합’ 한다는 의미
- 조인의 필요성
 - 하나의 SQL 질의문이 하나의 테이블만 검색할 수 있다고 하면 너무 불편함

■ 간단한 조인

- SQL에서 간단한 조인 표기법
 - FROM 절에 조인에 참여하는 두 테이블을 기록함
 - WHERE 절에 조인 조건을 기술함
- 조인문 작성 시 유의 사항
 - ① 컬럼 이름의 모호성
 - 컬럼 이름 앞에 테이블 이름을 접두사로 사용함
 - 테이블 이름과 컬럼 이름은 점(.)으로 구분함
 - ② 테이블의 이름이 긴 경우 테이블 이름 대신 별명 사용 가능
 - 하나의 SQL에서 테이블 이름과 별명을 혼용해서 쓸 수 없음

핵심요약

2. 다양한 조인 구문

■ 다양한 조인들

■ 카티샨 프로덕트(Cartesian Product)

- 두 테이블에 속한 튜플들의 모든 가능한 쌍을 생성함
- FROM 절에 두 개 이상의 테이블명을 기록
- WHERE 절에는 조인 조건을 기술하지 아니함

■ 동등 조인

- 조인 조건이 “=” 인 경우

■ 자연 조인

- 조인 조건을 명시 하지 않고 조인한다고 할 때 두 테이블에 공통으로 나타나는 속성의 동등조인으로 생각
- MS-SQL에서는 자연 조인을 명시적으로 지원하지는 않음

■ 쎄타 조인

- 조인 조건으로 <, >, <=, >=, != 등을 쓸 수 있음
- 일반적인 조인 조건에 대하여 쎄타 조인이라고 함

■ 셀프 조인

- 하나의 테이블 내에 있는 컬럼끼리 연관시켜 조인이 필요한 경우
- 조인 대상 테이블이 두 개인데 동일한 테이블임
- 물리적으로 1개이나 논리적으로 서로 다른 테이블이라고 생각하면 됨

■ 다중 조인

- 조인 질의의 경우 조인에 참여하는 테이블이 2개로 이를 보통 ‘2중 조인’(2-way join)이라고 함
- 경우에 따라서 여러 테이블들 간의 조인이 필요한 경우도 있음

핵심요약

2. 다양한 조인 구문

■ 다양한 조인들

■ ANSI 조인

- SQL을 표준화할 때 만든 ANSI 표준 문법
- 기존 SQL과 차이점은 조인 조건을 WHERE로 표현하지 않고 FROM 절에 표현함
- 크로스 조인(Cross Join) : 카티산 프로덕트(Cartesian Product)의 다른 표현 법
FROM 테이블명 CROSS JOIN 테이블명

- 내부 조인 (Inner Join) : 일반적인 조건의 Ansi 조인 표기법

FROM 테이블명 INNER JOIN 테이블명 ON 조인조건

FROM 테이블명 JOIN 테이블명 ON 조인조건

- 외부 조인

- ▶ 일반적인 조인은 조인 조건을 만족하는 튜플들만이 조인 결과에 나옴
- ▶ 조인 조건을 만족하지 않는 튜플들도 결과로 보고 싶을 경우 : 조인에 참여하는 테이블에 속한 모든 튜플 출력
- ▶ ANSI 조인 표기법 : 명시적 표기법
 - ① LEFT OUTER JOIN : 왼쪽 테이블에 있는 튜플들은 다 나옴
 - ② RIGHT OUTER JOIN : 오른쪽 테이블에 있는 튜플들은 다 나옴
 - ③ FULL OUTER JOIN : 양쪽 테이블에 있는 튜플들은 다 나옴



SQL 활용

중첩 질의문



한국기술교육대학교
온라인평생교육원

학습내용

- 중첩 질의문의 개요
- 다양한 중첩 질의문

학습목표

- 중첩 질의문의 개념에 대해 이해하고, 단일행 서브쿼리와 다중행 서브 쿼리에 대해 설명할 수 있다.
- 다양한 중첩 질의문을 적용하여 두 개 이상의 테이블로부터 데이터를 조회할 수 있다.

● 중첩 질의문의 개요

1. 중첩 질의문의 개념

하나의 SQL문의 결과를 다른 SQL문에 전달함

두 개의 SQL문을 하나의 SQL로 처리함

◆ 조인 질의문과 중첩 질의문

- 이론적으로 중첩 질의문은 조인 구문과 표현능력이 동일함
- 중첩 질의문의 필요성 ⇒ 조인의 필요성과 동일함

◆ 중첩 질의문의 필요성

하나의 SQL 질의문이 하나의 테이블만 검색할 수 있다고 하는 경우

Q 사번이 103인 사원의 부서명을 알고 싶을 때

- ① 사번이 103번인 사원의 부서 번호를 파악함
 - ② 해당 부서 번호와 같은 부서 번호를 가지고 부서명을 부서테이블에서 검색함
- ⇒ 매우 불편함 ⇒ 조인 구문 사용(조인 구문 어려움) ⇒ **중첩 질의문**

The image shows two side-by-side SQL query windows in SSMS.

Left Window (Outer Query):

```
USE MagicCorp
GO

SELECT DNO
FROM EMPLOYEE
WHERE ENO = 103
```

Right Window (Inner Query):

```
USE MagicCorp
GO

SELECT DNAME
FROM DEPARTMENT
WHERE DNO = 30
```

A red arrow points from the highlighted WHERE clause in the right window to the highlighted WHERE clause in the results of the left window, illustrating how the value from the first query is passed to the second.

Results for Left Window:

DNO
30

Results for Right Window:

DNAME
Sales

● 중첩 질의문의 개요

1. 중첩 질의문의 개념

◆ 중첩 질의문의 표현

- SQL문 안에 SQL문이 포함되어 있음

The screenshot shows a SQL query window in Microsoft SQL Server Management Studio. The query is:

```
USE MagicCorp
GO

SELECT DNAME
FROM DEPARTMENT
WHERE DEPARTMENT.DNO = (SELECT DNO
                           FROM EMPLOYEE
                           WHERE EMPLOYEE.ENO= 103
                         )
```

The results pane shows a single row with the value 'Sales' in the DNAME column.

DNAME
1 Sales

● 중첩 질의문의 개요

2. 단일행 서브 쿼리와 다중행 서브 쿼리

◆ 단일행 서브 쿼리

- 서브 쿼리의 결과로 **하나의 튜플만이 반환됨**
- 서브 쿼리의 검색 조건이 후보키에 연관되어 있을 경우가 많음

◆ 다중행 서브 쿼리

- 서브 쿼리의 결과로 **여러 개의 튜플들이 반환됨**



단일행 서브 쿼리와 다중행 서브 쿼리를 구분해야 할까?

일반적인 비교 연산자인 $=$, $<$, \leq , $>$, \geq , \neq 등은 속성값 간의 비교 연산임

- 집합에 대한 비교 연산이 안됨

◆ 단일행 서브 쿼리의 예

Q 사원 번호 110번과 같은 부서에 **근무**하는 사원들의 **사원 번호**와 **부서번호** 검색

- 사원번호가 기본키 임으로 사원번호 110번은 1명 밖에 없음
⇒ 단일행 서브 쿼리
- $=$, $<$, \leq , $>$, \geq , \neq 등을 사용할 수 있음

The screenshot shows a SQL query window with the following code:

```
USE MagicCorp
GO

SELECT ENO, DNO
FROM EMPLOYEE
WHERE EMPLOYEE.DNO = (SELECT DNO
                      FROM EMPLOYEE
                      WHERE EMPLOYEE.ENO= 110)
```

The results grid displays the following data:

	ENO	DNO
1	107	10
2	110	10
3	114	10

● 중첩 질의문의 개요

2. 단일행 서브 쿼리와 다중행 서브 쿼리

◆ 다중행 서브 쿼리 - 단일행 비교 연산자 사용 시 오류

Q 봉급이 500이상인 사원과 같은 부서에 근무하는 사원들의 이름, 봉급, 부서번호 구하기

The screenshot shows a SQL query window with the following code:

```
USE MagicCorp
GO

SELECT ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE DNO = (SELECT DNO
               FROM EMPLOYEE
               WHERE SALARY >= 500)
```

Below the code, there are two tabs: "Results" and "Messages". The "Messages" tab is selected, displaying the following error message:

Msg 512, Level 16, State 1, Line 2
Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <=, >, >= or when the subquery is used as an expression.

◆ 다중행 비교 연산자

● IN

- 속성값이 여러 값을 중 하나이기만 하면 참
- “= OR”的 의미

● ANY 또는 SOME

- 메인 쿼리 비교 조건에서 서브 쿼리의 결과와 하나라도 일치하면 참
- IN과의 차이점은 >, >=, <=, <과 같은 범위 비교와도 같이 사용이 가능함
- = ANY와 = SOME은 IN과 같은 의미임

● ALL

- 메인 쿼리 비교 조건에서 서브 쿼리의 결과와 모두 일치하면 참

● 중첩 질의문의 개요

2. 단일행 서브 쿼리와 다중행 서브 쿼리

◆ 다중행 서브 쿼리 : IN

- Q 봉급이 500이상인 사원과 같은 부서에 근무하는 사원들의 이름, 봉급, 부서번호 구하기

```
USE MagicCorp
GO

SELECT ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE DNO IN (SELECT DNO
               FROM EMPLOYEE
               WHERE SALARY >= 500)
```

	ENAME	SALARY	DNO		ENAME	SALARY	DNO
1	e1	300	20	8	e8	500	30
2	e2	250	30	9	e9	1000	20
3	e3	500	30	10	e10	500	10
4	e4	600	20	11	e11	280	30
5	e5	450	30	12	e12	300	20
6	e6	480	30	13	e13	560	20
7	e7	520	10	14	e14	250	10

◆ 다중행 서브 쿼리 : ANY

- Q 부서 번호 20에 근무하는 한 직원의 봉급 보다 많은 봉급을 받는 직원들의 이름, 봉급, 부서번호 출력

```
USE MagicCorp
GO

SELECT ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE SALARY > ANY (SELECT SALARY
                      FROM EMPLOYEE
                      WHERE DNO = 20)
```

	ENAME	SALARY	DNO
1	e3	500	30
2	e4	600	20
3	e5	450	30
4	e6	480	30
5	e7	520	10
6	e8	500	30
7	e9	1000	20
8	e10	500	10
9	e13	560	20

◆ 다중행 서브 쿼리 : ALL

- Q 부서 번호 10에 근무하는 모든 직원들의 봉급 보다 많은 봉급을 받는 직원들의 이름, 봉급, 부서번호 출력

```
USE MagicCorp
GO

SELECT ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY
                      FROM EMPLOYEE
                      WHERE DNO = 10)
```

	ENAME	SALARY	DNO
1	e4	600	20
2	e9	1000	20
3	e13	560	20

● 중첩 질의문의 개요

2. 단일행 서브 쿼리와 다중행 서브 쿼리

◆ 다중행 비교 연산자

- EXISTS
 - 서브 쿼리의 결과가 하나라도 존재하면 참이 되는 연산자
- NOT EXISTS
 - EXISTS와 상반되는 연산자

Q 봉급과 커미션의 합이 500이 넘는 사원이 존재하면 모든 사원의 이름 출력

The screenshot shows a SQL query window and a results grid. The query is:

```
USE MagicCorp
GO

SELECT ENAME
FROM EMPLOYEE
WHERE EXISTS (SELECT *
               FROM EMPLOYEE
               WHERE SALARY+COMMISSION > 500)
```

The results grid displays 14 rows of employee names (e1 to e14) from the EMPLOYEE table.

	ENAME		ENAME
1	e1	8	e8
2	e2	9	e9
3	e3	10	e10
4	e4	11	e11
5	e5	12	e12
6	e6	13	e13
7	e7	14	e14

● 다양한 중첩 질의문

1. 다중 컬럼 서브 쿼리

◆ 다중 컬럼 서브 쿼리란?

- 다중 컬럼 서브 쿼리 : 서브 쿼리의 결과가 여러 개의 속성들로 구성되어 주 쿼리의 조건과 비교하는 서브 쿼리임
 - 복수 개의 서브 쿼리들로 구성됨
 - 메인 쿼리와 서브 쿼리의 비교 대상 칼럼을 분리하여 개별적으로 비교한 후 AND 연산에 의해 최종 결과를 출력함

◆ 다중 컬럼 서브 쿼리의 예



사원번호 101인 사원과 동일 부서에 동일한 급여를 지급받는 직원 구하기

```
SELECT ENO, ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE DNO IN (SELECT DNO
               FROM EMPLOYEE
               WHERE ENO = 101)
AND SALARY IN (SELECT SALARY
                FROM EMPLOYEE
                WHERE ENO = 101)
```

	ENO	ENAME	SALARY	DNO
1	101	e1	300	20
2	112	e12	300	20

● 다양한 중첩 질의문

2. 상호 연관 서브 쿼리

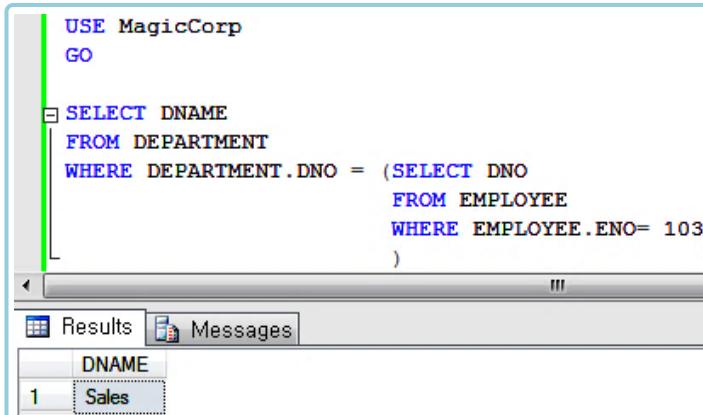
◆ 비상호 연관 서브 쿼리

- 앞서 보았던 서브 쿼리들은 서브 쿼리의 결과가 메인 쿼리에서 검사하는 튜플에는 영향 받지 않고 그 결과가 일정함
⇒ 부서 테이블의 어떤 튜플들을 검색하던지 간에 서브 쿼리

```
SELECT DNO FROM
EMPLOYEE WHERE
EMPLOYEE.ENO =103
```



항상 일관되게 부서번호 30을 반환함



The screenshot shows an SQL query window in SSMS. The code is:

```
USE MagicCorp
GO

SELECT DNAME
FROM DEPARTMENT
WHERE DEPARTMENT.DNO = (SELECT DNO
                        FROM EMPLOYEE
                        WHERE EMPLOYEE.ENO= 103)
```

The 'Results' tab is selected, showing the output:

DNAME
1 Sales

● 다양한 중첩 질의문

2. 상호 연관 서브 쿼리

◆ 상호 연관 서브 쿼리

메인 쿼리결과 서브 쿼리 간에 검색 결과를 교환하는 서브 쿼리

- 메인 쿼리와 서브 쿼리 간의 결과를 교환하기 위하여 서브 쿼리의 WHERE 조건절에서 메인 쿼리의 테이블과 연결함
- 서브 쿼리의 조건절에 메인 쿼리에서 사용하는 테이블의 속성이 나타남



메인 쿼리에서 어떤 튜플에 대한 조건을 비교하는가에

따라서 서브 쿼리의 결과가 다르게 나타남

● 상호 연관 서브 쿼리 사용법

```
SELECT 속성리스트  
FROM table1  
WHERE table1.속성 비교연산자 (
```

```
SELECT 속성리스트  
FROM table2  
WHERE table2.속성 비교연산자 table1 속성)
```

● 주의 사항

- 메인 쿼리에서 table1에 속한 튜플을 하나씩 접근하여 WHERE 절 수행 시 서브 쿼리가 반복적으로 수행됨으로 성능이 매우 떨어질 수 있음
⇒ 조인 구문을 이용하는 것이 더 효율적임

● 다양한 중첩 질의문

2. 상호 연관 서브 쿼리

◆ 상호 연관 서브 쿼리

Q 각 사원에 대하여 관리자와 동일 부서에서 근무하는 사원들의 이름, 급여, 부서 번호 출력

- 사원마다 관리자가 달라짐

```
SELECT ENO, ENAME, SALARY, DNO
FROM EMPLOYEE E
WHERE DNO IN (SELECT DNO
               FROM EMPLOYEE M
               WHERE E.MANAGER = M.ENO)
```

	ENO	ENAME	SALARY	DNO
1	101	e1	300	20
2	102	e2	250	30
3	103	e3	500	30
4	105	e5	450	30
5	106	e6	480	30
6	108	e8	500	30

● 다양한 중첩 질의문

3. 중첩 질의문 작성 시 주의점

- ◆ 다중행 서브 쿼리 시 단일행 비교 연산자와 사용하는 경우

- 중첩 질의문 사용 시 오류가 없도록 IN, ANY, ALL을 기본적으로 사용함

```
USE MagicCorp
GO

SELECT ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE DNO = (SELECT DNO
              FROM EMPLOYEE
              WHERE SALARY >= 500)

Msg 512, Level 16, State 1, Line 2
Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=,
```

- 서브 쿼리 내에서는 ORDER BY 절을 사용하면 안됨

```
SELECT ENO, ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE DNO IN (SELECT DNO
               FROM EMPLOYEE
               WHERE ENO = 101
               ORDER BY DNO)

Msg 1033, Level 15, State 1, Line 7
The ORDER BY clause is invalid in views, inline functions, derived tables, subqueries, and common table express
```

- 서브 쿼리의 결과가 NULL일 경우, 메인 쿼리의 결과 또한 NULL임

```
SELECT ENO, ENAME, SALARY, DNO
FROM EMPLOYEE
WHERE DNO IN (SELECT DNO
               FROM EMPLOYEE
               WHERE ENO = 500)

Results Messages
```

● 다양한 중첩 질의문

3. 중첩 질의문 작성 시 주의점

◆ 다중행 서브 쿼리 시 단일행 비교 연산자와 사용하는 경우

- 서브 쿼리가 NULL을 반환할 경우, 메인 쿼리에서 결과를 생성하고 싶으면 “NOT EXISTS” 를 사용함

```
SELECT ENO, ENAME, SALARY, DNO
  FROM EMPLOYEE
 WHERE NOT EXISTS (SELECT DNO
                      FROM EMPLOYEE
                     WHERE ENO = 500)
```

	ENO	ENAME	SALARY	DNO		ENO	ENAME	SALARY	DNO
1	101	e1	300	20	8	108	e8	500	30
2	102	e2	250	30	9	109	e9	1000	20
3	103	e3	500	30	10	110	e10	500	10
4	104	e4	600	20	11	111	e11	280	30
5	105	e5	450	30	12	112	e12	300	20
6	106	e6	480	30	13	113	e13	560	20
7	107	e7	520	10	14	114	e14	250	10

핵심요약

1. 중첩 질의문의 개요

■ 중첩 질의문의 개념

- 하나의 SQL문의 결과를 다른 SQL문에 전달함
- 두 개의 SQL문을 하나의 SQL로 처리함
- 이론적으로 중첩 질의문은 조인 구문과 표현능력이 동일함
- SQL문 안에 SQL문이 포함되어 있음

■ 단일행 서브 쿼리와 다중행 서브 쿼리

■ 단일행 서브 쿼리

- 서브 쿼리의 결과로 하나의 튜플만이 반환됨

■ 다중행 서브 쿼리

- 서브 쿼리의 결과로 여러 개의 튜플들이 반환됨

■ 단일행 서브 쿼리와 다중행 서브 쿼리를 구분해야 하는 이유

- 일반적인 비교 연산자인 $=$, $<$, \leq , $>$, \geq , \neq 등은 속성값 간의 비교 연산임

핵심요약

1. 중첩 질의문의 개요

■ 단일행 서브 쿼리와 다중행 서브 쿼리

■ 다중행 비교 연산자

① IN

- 속성값이 여러 값들 중 하나이기만 하면 참
- “= OR”의 의미

② ANY 또는 SOME

- 메인 쿼리 비교 조건에서 서브 쿼리의 결과와 하나라도 일치하면 참
- IN과의 차이점은 >, >=, <=, <과 같은 범위 비교와도 같이 사용 가능함

③ ALL

- 메인 쿼리 비교 조건에서 서브 쿼리의 결과와 모두 일치하면 참

④ EXISTS 연산자

- 서브 쿼리의 결과가 하나라도 존재하면 참이 되는 연산자

⑤ NOT EXISTS

- EXISTS와 상반되는 연산자

핵심요약

2. 다양한 중첩 질의문

■ 다중 컬럼 서브 쿼리

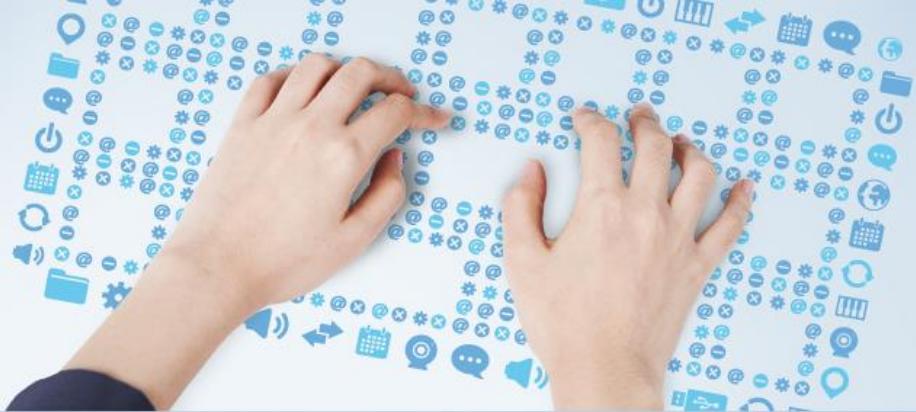
- 서브 쿼리의 결과가 여러 개의 속성들로 구성되어 주 쿼리의 조건과 비교하는 서브 쿼리임
- 복수 개의 서브 쿼리들로 구성됨
- 메인 쿼리와 서브 쿼리의 비교 대상 칼럼을 분리하여 개별적으로 비교한 후 AND 연산에 의해 최종 결과를 출력함

■ 상호 연관 서브 쿼리

- 비상호 연관 서브 쿼리
 - 서브 쿼리의 결과가 메인 쿼리에서 검사하는 튜플에는 영향 받지 않고 그 결과가 일정함
- 상호 연관 서브 쿼리
 - 메인 쿼리결과 서브 쿼리 간에 검색 결과를 교환하는 서브 쿼리
- 주의 사항
 - 메인 쿼리에서 table1에 속한 튜플을 하나씩 접근하여 WHERE 절 수행 시 서브 쿼리가 반복적으로 수행됨으로 성능이 매우 떨어질 수 있음

■ 중첩 질의문 작성 시 주의점

- 중첩 질의문 사용 시 오류가 없도록 IN, ANY, ALL을 기본적으로 사용함
- 서브 쿼리 내에서는 ORDER BY 절을 사용하면 안됨
- 서브 쿼리의 결과가 NULL일 경우, 메인 쿼리의 결과 또한 NULL임
- 서브 쿼리가 NULL을 반환할 경우, 메인 쿼리에서 결과를 생성하고 싶으면 “NOT EXISTS”를 사용함



SQL 활용

집합 연산자와 집단 연산자



한국기술교육대학교
온라인평생교육원

학습내용

- 집합 연산자
- 집단 연산자

학습목표

- 집합 연산자를 이용하여 질의문을 작성할 수 있다.
- 집단 함수와 집단 연산자를 이용하여 다양한 통계정보를 추출할 수 있다.

● 집합 연산자

1. 집합 연산자

◆ 집합 연산자

- 테이블을 구성하는 튜플 집합에 대한 테이블의 부분 집합을 결과로 반환하는 연산자
- UNION : 합집합
- INTERSECT : 교집합
- EXCEPT : 차집합(Oracle에선 MINUS로 사용)

◆ UNION의 예

Q 부서 번호 10인 사원들과 직급이 staff인 사원들 검색

```
USE MagicCorp
GO

DECLARE @emp1 TABLE (
    ENO INT,
    ENAME NVARCHAR(20),
    JOB NVARCHAR(10),
    MANAGER INT,
    HIREDATE DATE,
    SALARY INT,
    COMMISSION DECIMAL(4,2),
    DNO INT
)

DECLARE @emp2 TABLE (
    ENO INT,
    ENAME NVARCHAR(20),
    JOB NVARCHAR(10),
    MANAGER INT,
    HIREDATE DATE,
    SALARY INT,
    COMMISSION DECIMAL(4,2),
    DNO INT
)

INSERT INTO @emp1
SELECT * FROM EMPLOYEE WHERE DNO = 10

INSERT INTO @emp2
SELECT * FROM EMPLOYEE WHERE JOB = 'staff'

SELECT * FROM @emp1
UNION
SELECT * FROM @emp2
```

The screenshot shows a SQL query results window. At the top, there are tabs for 'Results' and 'Messages'. The 'Results' tab is selected, displaying a table with the following data:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	101	e1	staff	113	2007-03-01 00:00:00.000	300	NULL	20
2	107	e7	chief	108	2004-01-08 00:00:00.000	520	NULL	10
3	110	e10	section	103	2005-04-07 00:00:00.000	500	NULL	10
4	111	e11	staff	107	2007-03-01 00:00:00.000	280	NULL	30
5	112	e12	staff	106	2007-08-09 00:00:00.000	300	NULL	20
6	114	e14	staff	106	2007-11-09 00:00:00.000	250	NULL	10

● 집합 연산자

1. 집합 연산자

◆ INTERSECT의 예

Q 부서 번호 10이고 직급이 staff인 사원들 검색

```
USE MagicCorp
GO

(SELECT *
FROM EMPLOYEE
WHERE DNO = 10)
INTERSECT
(SELECT *
FROM EMPLOYEE
WHERE JOB = 'staff')
```

The screenshot shows the SQL Server Management Studio interface with the 'Results' tab selected. The results grid displays one row of data:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	114	e14	staff	106	2007-11-09 00:00:00.000	250	NULL	10

◆ EXCEPT의 예

Q 부서 번호 10인 사원들 중 직급이 staff인 사원들 검색

```
USE MagicCorp
GO

(SELECT *
FROM EMPLOYEE
WHERE DNO = 10)
EXCEPT
(SELECT *
FROM EMPLOYEE
WHERE JOB = 'staff')
```

The screenshot shows the SQL Server Management Studio interface with the '결과' (Results) tab selected. The results grid displays two rows of data:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	107	e7	chief	108	2004-01-08 00:00:00.000	520	NULL	10
2	110	e10	section	103	2005-04-07 00:00:00.000	500	NULL	10

● 집합 연산자

2. UNION과 UNION ALL

집합 연산자를 대상 테이블을 집합으로 봄

- 따라서 결과도 집합임 \Rightarrow 중복을 허용하지 않음

필요에 따라서 중복된 결과도 보고 싶은 경우



UNION ALL을 사용

◆ UNION ALL의 사용

Q 부서 번호 10인 사원들과 직급이 staff인 사원들 검색(중복 허용)

```
USE MagicCorp
GO

DECLARE @emp1 AS TABLE (
    SELECT * 
    FROM EMPLOYEE
    WHERE DNO = 10)
UNION ALL
DECLARE @emp2 AS TABLE (
    SELECT * 
    FROM EMPLOYEE
    WHERE JOB = 'staff')
```

Results Messages

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	107	e7	chief	108	2004-01-08 00:00:00.000	520	NULL	10
2	110	e10	section	103	2005-04-07 00:00:00.000	500	NULL	10
3	114	e14	staff	106	2007-11-09 00:00:00.000	250	NULL	10
4	101	e1	staff	113	2007-03-01 00:00:00.000	300	NULL	20
5	111	e11	staff	107	2007-03-01 00:00:00.000	280	NULL	30
6	112	e12	staff	106	2007-08-09 00:00:00.000	300	NULL	20
7	114	e14	staff	106	2007-11-09 00:00:00.000	250	NULL	10

● 집합 연산자

3. 외부 합집합

◆ 합병 호환성

\cup , \cap , - 연산의 피연산자(릴레이션)들이 지켜야 할 제약 조건

- 차수(Degree : 속성의 수)가 같아야 함
- 대응되는 속성 쌍 별로 타입(또는 도메인)이 같아야 함
- 대응되는 속성 쌍 별로 의미(Semantic)가 같아야 함

◆ 합병 호환성의 불일치

두 질의 결과가 합병 호환성을 만족하지 않음

합집합하고 싶은 경우



외부 합집합($U+$)

◆ SQL에서의 외부 합집합

- 두 질의 결과의 속성 수와 타입이 일치되게 만듦
 - NULL은 모든 속성에서 사용할 수 있는 속성값임
- ⇒ 이를 이용하여 강제로 동일한 구조가 되게 함

● 집합 연산자

3. 외부 합집합

◆ SQL에서의 외부 합집합

- 두 질의 결과의 속성 수와 타입이 일치되게 만듦
- NULL은 모든 속성에서 사용할 수 있는 속성값임

⇒ 이를 이용하여 강제로 동일한 구조가 되게 함

- 외부 합집합의 예

- 두 질의 결과에서 공통인 속성은 dno밖에 없음
- 다른 것들은 NULL로 출력하도록 함

⇒ 속성의 수 : 각각 4개, 타입도 일치함

⇒ 합집합 할 수 있음

```
(select eno, ename, dno , NULL from EMPLOYEE)
UNION
(select NULL, NULL, dno, dname from DEPARTMENT)
```

	eno	ename	dno	(No column name)		eno	ename	dno	(No column name)
1	101	e1	20	NULL		10	e10	10	NULL
2	102	e2	30	NULL		11	e11	30	NULL
3	103	e3	30	NULL		12	e12	20	NULL
4	104	e4	20	NULL		13	e13	20	NULL
5	105	e5	30	NULL		14	e14	10	NULL
6	106	e6	30	NULL		15	NULL	10	Accounting
7	107	e7	10	NULL		16	NULL	20	Human
8	108	e8	30	NULL		17	NULL	30	Sales
9	109	e9	20	NULL		18	NULL	40	Computing

● 집단 연산자

1. 집단 함수

◆ 집단 함수란?

- 테이블의 전체 행을 하나 이상의 컬럼을 기준으로 그룹화하여 해당 그룹 별 통계 값을 출력하는 함수

Q 전체 직원의 평균 봉급을 구하기

The screenshot shows a SQL query window with the following content:

```
USE MagicCorp
GO

SELECT AVG(SALARY)
FROM EMPLOYEE
```

The results pane shows a single row with the value 463.

(No column name)
1 463

◆ 집단 함수의 종류

- SUM : 그룹의 합계
- AVG : 그룹의 평균
- COUNT : 그룹의 개수
- MAX : 그룹의 최대값
- MIN : 그룹의 최소값
- STDEV : 그룹의 표준편차
- VAR : 그룹의 분산

일반적으로 집단 함수는 NULL값을 제외한 속성값들의 통계 값을 반환함

● 집단 연산자

1. 집단 함수

◆ 집단 함수의 예



사원의 최대 봉급, 최소 봉급 구하기

```
USE MagicCorp
GO

SELECT MAX(SALARY) AS MAXSAL, MIN(SALARY) AS MINSAL
FROM EMPLOYEE
```

Results Messages

	MAXSAL	MINSAL
1	1000	250

● 집단 연산자

1. 집단 함수

◆ 분산(VAR)

- 각 값이 평균과 얼마나 떨어져 있는지에 대한 통계 값
- 각 값과 평균의 차에 대한 차(즉, 편차)의 제곱의 평균
 - 편차는 음수이거나 양수 일수 있음으로 편차를 제곱하여 양수로 나타내고 이들의 합에 대한 평균을 구함

$$(\sum_{i=1,n} (x_i - M)^2)/N$$

- x_i : 각 값
- M : 평균
- N : 값의 개수

◆ 표준 편차(STDEV)

- 분산의 경우 편차에 대한 제곱으로 나타내므로 평균과의 단위가 맞지 않음
- 단위를 맞추기 위하여 분산의 제곱근을 표준편차로 사용함

$$STDEV = VAR^{1/2}$$

◆ 분산(VAR)과 표준 편차(STDEV)

Q 사원들의 급여 평균, 분산, 표준편차 구하기

- NULL값을 제외한 모든 행의 평균, 분산, 표준편차가 구해짐

```
SELECT AVG(salary), VAR(salary), STDEV(salary)
FROM EMPLOYEE
```

The screenshot shows the SQL Server Management Studio interface with the following details:

- Query Editor:** Contains the T-SQL query: `SELECT AVG(salary), VAR(salary), STDEV(salary) FROM EMPLOYEE`.
- Results Tab:** Displays the output of the query. The results table has three columns: (No column name), (No column name), and (No column name). The single row contains values: 463, 38593.9560439561, and 196.45344497859.
- Messages Tab:** Shows no messages or errors.

● 집단 연산자

1. 집단 함수

◆ COUNT(*)와 COUNT(속성명)

● COUNT(*)

- 테이블에서 조건을 만족하는 행의 개수를 반환하는 함수
- *는 모든 속성들이란 의미임

● COUNT(속성명)

- 속성값이 NULL이 아닌 속성값의 개수

● COUNT(DISTINCT 속성명)

- 속성값이 NULL이 아니며 중복되지 않는 속성값들의 개수
- SUM, AVG도 DISTINCT를 쓸 수 있음
- MIN, MAX에서는 DISTINCT가 의미가 없음

Q 사원 테이블의 **튜플수**와 **COMMISION**의 개수 구하기

```
USE MagicCorp
GO

SELECT COUNT(*) AS NUMROW, COUNT(COMMISSION)
FROM EMPLOYEE
```

Results Messages

NUMROW	(No column name)
1	14

Q 사원의 직급(JOB)의 수와 중복되지 않는 직급(JOB)의 수 구하기

```
USE MagicCorp
GO

SELECT COUNT(JOB) AS NUMJOB, COUNT(DISTINCT JOB) AS NUMDISTINCTJOB
FROM EMPLOYEE
```

Results Messages

NUMJOB	NUMDISTINCTJOB
1	14

● 집단 연산자

2. GROUP BY와 HAVING

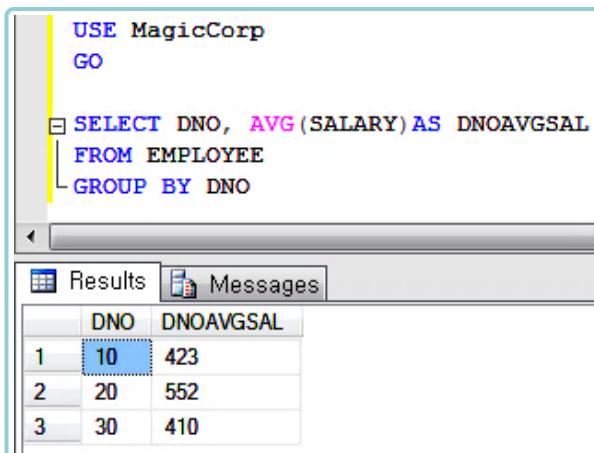
◆ GROUP BY

- 특정 속성을 기준으로 테이블 전체를 그룹으로 나누기 위한 절

예) 부서별 사원의 평균 봉급

```
SELECT 컬럼_리스트  
FROM 테이블명  
WHERE 조건  
GROUP BY 컬럼_리스트
```

Q 부서별 사원들의 평균 봉급과 부서번호 검색하기



USE MagicCorp
GO

```
SELECT DNO, AVG(SALARY) AS DNOAVGSAL  
FROM EMPLOYEE  
GROUP BY DNO
```

	DNO	DNOAVGSAL
1	10	423
2	20	552
3	30	410

● 집단 연산자

2. GROUP BY와 HAVING

◆ GROUP BY 사용 시 주의점

- ① SELECT 절에는 집단 연산자나 GROUP BY에 사용한 속성명만을 사용할 수 있음
- ② 공통되는 속성값으로 그룹핑을 했으므로, 각 그룹에서 개별 튜플을 접근할 수 없음

Q 부서별 사원들의 평균 봉급과 부서번호, 사원이름 검색하기(오류)

```
USE MagicCorp
GO

SELECT DNO, AVG(SALARY) AS DNOAVGSAL, ENAME
FROM EMPLOYEE
GROUP BY DNO
```

Messages

Msg 8120, Level 16, State 1, Line 2
Column 'EMPLOYEE.ENAME' is invalid in the select list because it is not contained in either GROUP BY or HAVING clause.

◆ HAVING

- 각 그룹에 대한 제약 조건을 기술할 때 사용함
- HAVING 절은 GROUP BY 절의 종속절임
 - GROUP BY없이 HAVING은 나타날 수 없음
- WHERE 절은 테이블 전체에 대한 제약 조건을 나타냄

Q 부서의 최대 봉급이 500초과인 부서에 대해서만 부서별 사원들의 평균 봉급과 부서 번호 출력하기

```
USE MagicCorp
GO

SELECT DNO, AVG(SALARY) AS DNOAVGSAL
FROM EMPLOYEE
GROUP BY DNO
HAVING MAX(SALARY) > 500
```

Results

DNO	DNOAVGSAL
10	423
20	552

● 집단 연산자

3. ROLLUP과 CUBE

◆ 다중 속성 GROUP BY

- 하나 이상의 속성들을 이용하여 그룹을 나누고, 그룹별로 다시 서브 그룹을 나누고자 할 때 사용함

GROUP BY 컬럼1, 컬럼2, …, 컬럼ⁿ

Q 부서별 직급별 사원들의 봉급 합 구하기

```
USE MagicCorp
GO

select dno, job, sum(salary)  as [AVGSAL]
from employee
group by dno, job
```

	dno	job	AVGSAL
1	20	ceo	1000
2	10	chief	520
3	20	chief	1160
4	30	chief	480
5	30	deputy	250
6	10	section	500
7	30	section	950
8	30	senior	500
9	10	staff	250
10	20	staff	600
11	30	staff	280

● 집단 연산자

3. ROLLUP과 CUBE

◆ ROLLUP 연산자

- GROUP BY 절의 그룹 조건에 따라서 그룹핑 하고 각 그룹에 대한 부분합을 구하는 연산자
- GROUP BY 절에 n개의 속성 명이 있으면, **n+1**개의 그룹핑 조합이 나옴
- GROUP BY **c₁, c₂, c₃** 일 때
 - 각 그룹 **c₁, c₂, c₃** 별 합
 - 각 그룹 **c₁, c₂** 별 합
 - 그룹 **c₁** 별 합
 - 전체 합



ROLLUP 연산자를 이용하여 부서별 직급별 사원들의 봉급 합 구하기

```
USE MagicCorp
GO

select dno, job, sum(salary) as [totalSAL]
from employee
group by dno, job with rollup
```

The screenshot shows a SQL query window and its results. The query uses the 'rollup' keyword to group employees by department (dno) and job, and then provides a summary row for each department. The results table has three columns: dno, job, and totalSAL.

dno	job	totalSAL	
1	10	chief	520
2	10	section	500
3	10	staff	250
4	10	NULL	1270
5	20	ceo	1000
6	20	chief	1160
7	20	staff	600
8	20	NULL	2760
9	30	chief	480
10	30	deputy	250
11	30	section	950
12	30	senior	500
13	30	staff	280
14	30	NULL	2460
15	NULL	NULL	6490

● 집단 연산자

3. ROLLUP과 CUBE

◆ CUBE 연산자

- GROUP BY 절의 그룹 조건에 따라서 그룹핑하고 각 그룹의 조합에 따른 부분합을 구하는 연산자
- GROUP BY 절에 n개의 속성명이 있으면 2^n 개의 그룹핑 조합이 나옴
- GROUP BY c_1, c_2, c_3 일 때
 - 각 그룹 c_1, c_2, c_3 별 합
 - 각 그룹 c_1, c_2 별 합, c_1, c_3 별 합, c_2, c_3 별 합
 - 그룹 c_1 별 합, c_2 별 합, c_3 별 합
 - 전체 합



CUBE 연산자를 이용하여 부서별 직급별 사원들의 봉급 합 구하기

```
USE MagicCorp
GO

select dno, job, sum(salary)  as [totalSAL]
from employee
group by dno, job with cube
```

The screenshot shows the SQL Server Management Studio interface with a query window and a results grid.

Query window content:

```
USE MagicCorp
GO

select dno, job, sum(salary)  as [totalSAL]
from employee
group by dno, job with cube
```

Results grid:

	dno	job	totalSAL		dno	job	totalSAL	
1	20	ceo	1000		12	30	senior	500
2	NULL	ceo	1000		13	NULL	senior	500
3	10	chief	520		14	10	staff	250
4	20	chief	1160		15	20	staff	600
5	30	chief	480		16	30	staff	280
6	NULL	chief	2160		17	NULL	staff	1130
7	30	deputy	250		18	NULL	NULL	6490
8	NULL	deputy	250		19	10	NULL	1270
9	10	section	500		20	20	NULL	2760
10	30	section	950		21	30	NULL	2460
11	NULL	section	1450					

● 집단 연산자

3. ROLLUP과 CUBE

◆ GROUPING SETS 함수

여러 개의 GROUP조건을 표시하고 싶은 경우

예) 부서 별 급여 합과 직급별 급여 합을 한번에 보고 싶음

부서, 직급별 합을 보고 싶지 않은 경우



GROUPING SETS 함수를 이용함

```
select dno, job, sum(salary) as [totalSAL]
from employee
group by GROUPING SETS( (dno), (job) )
```

Results Messages

	dno	job	totalSAL
1	NULL	ceo	1000
2	NULL	chief	2160
3	NULL	deputy	250
4	NULL	section	1450
5	NULL	senior	500
6	NULL	staff	1130
7	10	NULL	1270
8	20	NULL	2760
9	30	NULL	2460

← GROUP BY job의 결과

← GROUP BY dno의 결과

핵심요약

1. 집합 연산자

■ 집합 연산자

- 테이블을 구성하는 튜플 집합에 대한 테이블의 부분 집합을 결과로 반환하는 연산자
 - UNION : 합집합
 - INTERSECT : 교집합
 - EXCEPT : 차집합(Oracle에선 MINUS로 사용)

■ UNION과 UNION ALL

- 집합 연산자를 대상 테이블을 집합으로 봄
- 따라서 결과도 집합임
 - 중복을 허용하지 않음
- 필요에 따라서 중복된 결과도 보고 싶은 경우
 - UNION ALL을 사용함

■ 외부 합집합

- 합병 호환성
 - \cup , \cap , $-$ 연산의 피연산자(릴레이션)들이 지켜야 할 제약 조건
- 합병 호환성의 불일치
 - 두 질의 결과는 합병 호환성을 만족하지 않음
 - 합병 호환성이 만족되지 않는 두 테이블의 합집합 구하기 : 외부 합집합(U^+)
- SQL에서의 외부 합집합
 - 두 질의 결과의 속성 수와 타입이 일치되게 만듦
 - NULL은 모든 속성에서 사용할 수 있는 속성값임
 - 이를 이용하여 강제로 동일한 구조가 되게 함

핵심요약

2. 집단 연산자

■ 집단 함수

- 테이블의 전체 행을 하나 이상의 컬럼을 기준으로 그룹화 하여 그 그룹 별 통계값을 출력하는 함수

- SUM : 그룹의 합계
- AVG : 그룹의 평균
- COUNT : 그룹의 개수
- MAX : 그룹의 최대값
- MIN : 그룹의 최소값
- STDEV : 그룹의 표준편차
- VAR : 그룹의 분산

■ 분산(VAR)

- 각 값이 평균과 얼마나 떨어져 있는지에 대한 통계값
- 각 값과 평균의 차에 대한 차(즉, 편차)의 제곱의 평균

$$(sum_{i=1,n} (x_i - M)^2)/N$$

■ 표준 편차(STDEV)

- 분산의 경우 편차에 대한 제곱으로 나타냄으로 평균과의 단위가 맞지 않음
- 단위를 맞추기 위하여 분산의 제곱근을 표준편차로 사용함

$$STDEV = VAR^{1/2}$$

핵심요약

2. 집단 연산자

■ 집단 함수

■ COUNT(*)

- 테이블에서 조건을 만족하는 행의 개수를 반환하는 함수

■ COUNT(속성명)

- 속성값이 NULL아닌 속성값의 개수

■ COUNT(DISTINCT 속성명)

- 속성값이 NULL이 아니며 중복되지 않는 속성값들의 개수

■ GROUP BY와 HAVING

■ GROUP BY

- 특정 속성을 기준으로 테이블 전체를 그룹으로 나누기 위한 절

```
SELECT 컬럼_리스트  
FROM 테이블명  
WHERE 조건  
GROUP BY 컬럼_리스트
```

- SELECT 절에는 집단 연산자나 GROUP BY에 사용한 속성명 만을 사용할 수 있음
- 공통되는 속성값으로 그룹핑을 했으므로, 각 그룹에서 개별 튜플을 접근할 수 없음

■ HAVING

- 각 그룹에 대한 제약 조건을 기술할 때 사용함
- HAVING 절은 GROUP BY 절의 종속절임
- WHERE 절은 테이블 전체에 대한 제약 조건을 나타냄

핵심요약

2. 집단 연산자

■ ROLLUP과 CUBE

■ 다중 속성 GROUP BY

- 하나 이상의 속성들을 이용하여 그룹을 나누고, 그룹별로 다시 서브 그룹을 나누고자 할 때

GROUP BY 컬럼1, 컬럼2, …, 컬럼ⁿ

■ ROLLUP 연산자

- GROUP BY 절의 그룹 조건에 따라서 그룹핑하고 각 그룹에 대한 부분합을 구하는 연산자
- GROUP BY 절에 n개의 속성 명이 있으면, n+1개의 그룹핑 조합이 나옴

■ CUBE 연산자

- GROUP BY 절의 그룹 조건에 따라서 그룹핑하고 각 그룹의 조합에 따른 부분합을 구하는 연산자
- GROUP BY 절에 n 개의 속성명이 있으면 2n개의 그룹핑 조합이 나옴

■ GROUPING SETS 함수

- 경우에 따라서 여러 개의 GROUP조건을 표시하고 싶은 경우, 부서, 직급별 합을 보고 싶지 않은 경우 GROUPING SETS 함수를 이용함



SQL 활용

순위 계산



한국기술교육대학교
온라인평생교육원

학습내용

- 순위 함수
- 그룹 별 순위

학습목표

- 질의 결과에 순위를 지정하는 순위 함수를 사용하여 순위를 추출할 수 있다.
- 순위 함수를 적용하여 그룹별 순위를 부여할 수 있다.

● 순위 함수

1. TOP() 함수

◆ 질의 결과 튜플 수의 제한

- 질의 결과는 ORDER BY 절을 이용하여 정렬할 수 있음
- ORDER BY 정렬 기준에서 특정 등수 / 비율까지만 보고 싶은 경우
⇒ Top(n) 함수를 이용함

```
SELECT TOP(n) 속성명
```

```
...
```

```
ORDER BY 속성명
```



사원들 중 급여 기준 5등까지만 결과로 출력하기

```
USE MagicCorp
GO
select top(5) *
from employee
order by salary desc
```

The screenshot shows the SQL Server Management Studio interface. On the left, the query window displays the T-SQL code. On the right, the 'Results' tab shows the output of the query. The results are a table with columns: ENO, ENAME, JOB, MANAGER, HIREDATE, SALARY, COMMISSION, and DNO. The data for the top 5 employees is as follows:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	109	e9	ceo	NULL	1996-10-04 00:00:00.000	1000	NULL	20
2	104	e4	chief	1008	2003-09-02 00:00:00.000	600	NULL	20
3	113	e13	chief	1003	2002-10-09 00:00:00.000	560	NULL	20
4	107	e7	chief	1008	2004-01-08 00:00:00.000	520	NULL	10
5	103	e3	section	1005	2005-02-10 00:00:00.000	500	100	30

- 5등에 동률이 있을 경우 임의로 한 개만 출력함

● 순위 함수

1. TOP() 함수

- ◆ 동률이 있을 때 모두 보고 싶은 경우

- WITH TIES를 사용함

```
SELECT TOP(n) WITH TIES 속성명
```

...

```
ORDER BY 속성명
```

- Q 사원들 중 급여 기준 5등까지만 결과로 출력하기

```
USE MagicCorp  
GO  
select top(5) WITH TIES *  
from employee  
order by salary desc
```

The screenshot shows the SQL Server Management Studio interface. In the 'Results' tab, the output of the query is displayed in a grid. The columns are labeled ENO, ENAME, JOB, MANAGER, HIREDATE, SALARY, COMMISSION, and DNO. The data shows seven rows of employee information, ordered by salary in descending order. The first row has ENO 109 and ENAME e9.

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	109	e9	ceo	NULL	1996-10-04 00:00:00.000	1000	NULL	20
2	104	e4	chief	1008	2003-09-02 00:00:00.000	600	NULL	20
3	113	e13	chief	1003	2002-10-09 00:00:00.000	560	NULL	20
4	107	e7	chief	1008	2004-01-08 00:00:00.000	520	NULL	10
5	108	e8	senior	1003	2004-03-08 00:00:00.000	500	0	30
6	103	e3	section	1005	2005-02-10 00:00:00.000	500	100	30
7	110	e10	section	1003	2005-04-07 00:00:00.000	500	NULL	10

- 5등에 동률이 있을 경우 모두 출력함

● 순위 함수

1. TOP() 함수

◆ 정렬 기준 특정 비율까지만 보고 싶은 경우

- Top(n) PERCENT를 이용함
- 상위 n%까지만을 출력하게 됨
 - WITH TIES와도 같이 쓸 수 있음

```
SELECT TOP(n) PERCENT [WITH TIES] 속성명
```

...

```
ORDER BY 속성명
```



사원들 중 급여 기준 20%까지만 결과로 출력하기

```
USE MagicCorp
GO

select top(20) PERCENT *
from employee
order by salary desc
```

100 %

결과 메시지

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	109	e9	ceo	NULL	1996-10-04 00:00:00,000	1000	NULL	20
2	104	e4	chief	108	2003-09-02 00:00:00,000	600	NULL	20
3	113	e13	chief	103	2002-10-09 00:00:00,000	560	NULL	20

● 순위 함수

2. RANK() 함수

TOP() 함수를 쓰면 결과 수를 제한함

- TOP() 함수는 등수를 구할 수 없음

◆ RANK 함수

- 각 튜플에 등수를 표시함

RANK 함수 over (order by 속성명 [asc|desc])

- 속성 기준 오름차순(asc) 또는 내림차순(desc)으로 정렬된 상태에 대하여 등수 지정
 - 다양함 RANK 함수가 있음

◆ RANK() 함수

```
SELECT 속성명, RANK () OVER (ORDER BY 속성명 [asc|desc] )
```

- 동률에 대하여 동일 등수 배정
 - 비연속식 등수 배정

예 1,2,2,4,⋯

Q 사원에 대하여 이름, 급여, 급여에 대한 내림차순 RANK() 값 출력하기

```
USE MagicCorp
GO

select ename, salary, rank() over(order by salary desc) as rank
from employee
```

	ename	salary	rank
1	e9	1000	1
2	e4	600	2
3	e13	560	3
4	e7	520	4
5	e8	500	5
6	e3	500	5
7	e10	500	5
8	e6	480	8
9	e5	450	9

● 순위 함수

2. RANK() 함수

◆ DENSE_RANK() 함수

```
SELECT 속성명, DENSE_RANK () OVER (ORDER BY 속성명)
```

- 동률에 대하여 동일 등수 배정
 - 연속식 등수 배정

예 1,2,2,3,⋯

Q 사원에 대하여 이름, 급여, 급여에 대한 DENSE_RANK() 값 출력하기

```
USE MagicCorp
GO

select ename, salary, dense_rank() over(order by salary desc) as rank
from employee
```

100 % <

결과 메시지

	ename	salary	rank
1	e9	1000	1
2	e4	600	2
3	e13	560	3
4	e7	520	4
5	e8	500	5
6	e3	500	5
7	e10	500	5
8	e6	480	6
9	e5	450	7

● 순위 함수

2. RANK() 함수

◆ ROW_NUMBER() 함수

```
SELECT 속성명, ROW_NUMBER () OVER (ORDER BY 속성명)
```

- 동률에 대하여 임의 등수 배정
 - 연속식 등수 배정

예 1,2,3,4,⋯

Q 사원에 대하여 이름, 급여, 급여에 대한 ROW_NUMBER() 값 출력하기

```
USE MagicCorp
GO

select ename, salary, ROW_NUMBER() over(order by salary desc) as rank
from employee
```

100 % < <

결과 메시지

	ename	salary	rank
1	e9	1000	1
2	e4	600	2
3	e13	560	3
4	e7	520	4
5	e8	500	5
6	e3	500	6
7	e10	500	7
8	e6	480	8
9	e5	450	9

● 순위 함수

2. RANK() 함수

◆ NTILE(n) 함수

- 전체 튜플을 n 개로 균등 분할하여 순위 지정

- 예
- 결과 튜플이 20개이고 n 이 10이면, 1등 2개, 2등 2개, …, 10등 2개로 등수 지정
 - 결과 튜플수가 n 으로 나누어 떨어지지 않으면 1등부터 추가적으로 배정함
 - 결과 튜플이 22개이고 n 이 10이면 1등 3개, 2등 3개, 3등 2개, …, 10등 2개로 등수 지정

Q 사원에 대하여 이름, 급여, 급여에 대한 내림차순으로 5등분 하여 등분순위 출력하기

```
USE MagicCorp  
GO  
  
select ename, salary, NTILE(5) over(order by salary desc) as rank  
from employee
```

100 % <

결과 메시지

	ename	salary	rank		ename	salary	rank
1	e9	1000	1	8	e6	480	3
2	e4	600	1	9	e5	450	3
3	e13	560	1	10	e1	300	4
4	e7	520	2	11	e12	300	4
5	e8	500	2	12	e11	280	4
6	e3	500	2	13	e14	250	5
7	e10	500	3	14	e2	250	5

● 그룹 별 순위

1. 그룹 별 순위 지정

◆ 기준 RANK() 함수 문법

- 전체 결과에 대한 속성값 기준 등수 지정이 됨
 - 특정 그룹별 순위 지정은 어떻게 할까?
 - 부서별로 구분해서 각 부서 내에서 봉급 순위를 알아봄

◆ PARTITION BY 속성명

- 튜플들을 속성값에 따라서 그룹핑함
- 각 그룹에 대하여 순위 함수를 적용함

```
RANK() over (PARTITION BY dno ORDER BY salary desc)
```

- “DNO별로 분류하고 각 분류된 소그룹에서 salary기준 내림차순하고 순위를 나타내시오.”라는 의미임

Q DNO별로 분류하고 각 분류된 소그룹에서 salary기준 내림차순하고 순위 나타내기

```
USE MagicCorp
GO

select ename, salary, dno,
       rank() over(partition by dno order by salary desc) as rank_dept
from employee
```

The screenshot shows the SQL Server Management Studio interface with a query window and a results grid. The query window contains the T-SQL code provided above. The results grid displays 14 rows of data from the employee table, partitioned by department (dno) and ordered by salary in descending order. The 'rank_dept' column shows the ranking within each department.

	ename	salary	dno	rank_dept		ename	salary	dno	rank_dept	
1	e7	520	10	1		8	e1	300	20	4
2	e10	500	10	2		9	e8	500	30	1
3	e14	250	10	3		10	e3	500	30	1
4	e9	1000	20	1		11	e6	480	30	3
5	e4	600	20	2		12	e5	450	30	4
6	e13	560	20	3		13	e11	280	30	5
7	e12	300	20	4		14	e2	250	30	6

● 그룹 별 순위

1. 그룹 별 순위 지정

- ◆ 그룹별 특정 등수의 정보를 보고 싶은 경우

- WHERE 절을 같이 활용함

```
RANK() over (PARTITION BY dno ORDER BY salary desc) AS 속성명
```

...

WHERE 속성명 = 등수



각 부서에서 급여 순위 2등인 사원의 부서번호, 이름과 급여 출력하기

- 인라인 뷰를 사용함

The screenshot shows a SQL query being run in SSMS. The query uses an inline view (temp) to rank employees by department and salary, then filters for those ranked 2nd in each department.

```
USE MagicCorp
GO

select *
FROM
    (SELECT dno, ename, salary, rank() over(partition by dno order by salary desc) as rank_val
     FROM EMPLOYEE) as temp
WHERE rank_val =2
```

The results grid shows two rows of data:

	dno	ename	salary	rank_val
1	10	e10	500	2
2	20	e4	600	2

● 그룹 별 순위

2. 그룹 별 집단 함수

- ◆ 그룹 별 집단 함수의 적용

```
SELECT 집단 함수 ~ GROUP BY~
```

- ◆ PARTITION BY를 이용해서도 그룹 별 집단 함수를 적용할 수 있음

```
SELECT 집단함수() OVER (PARTITION BY 속성명)
```

- Q PARTITION BY를 이용하여 부서별 급여의 평균 출력하기

```
USE MagicCorp  
GO  
  
select dno,  
       AVG(salary) over(partition by dno ) as avg_sal_dept  
  from employee
```



The screenshot shows the SQL Server Management Studio interface with the following details:

- Query Editor window:
 - Text area contains the SQL code provided above.
 - Results pane shows the output of the query:
- Results pane tabs: "결과" (Results) is selected, showing the query results; "메시지" (Messages) is also present.
- Query results table:

	dno	avg_sal_dept
1	10	423
2	10	423
3	10	423
4	20	552
5	20	552
6	20	552
7	20	552
8	20	552

● 그룹 별 순위

3. 행 순서 함수

◆ 행 순서 함수란?

- 행 순서 함수 : 정렬된 대상에서 특정 순위의 튜플들을 추출할 필요가 있을 때 사용되는 함수
 - FIRST_VALUE 함수
 - 정렬 대상에서 첫 번째 데이터 추출
 - LAG / LEAD 함수
 - 지정된 순서에서 선행 / 후행 데이터를 참조하는 함수

◆ FIRST_VALUE 함수

Q 각 부서별 최고 급여액 출력하기

```
USE MagicCorp
GO

select DISTINCT dno, FIRST_VALUE(salary)
           over(partition by dno order by salary desc) as higest_sal
      from employee
```

100 % <

	dno	higest_sal
1	10	520
2	20	1000
3	30	500

● 그룹 별 순위

3. 행 순서 함수

◆ LAG / LEAD 함수

- 정렬 기준 선행 값 / 후행 값을 추출함

Q 각 사원별 이름, 급여와 급여 순위 상 선행 순위의 급여, 급여 순위 상 후행 순위의 급여 출력하기

```
USE MagicCorp
GO

select DISTINCT ename, salary,
               LAG(salary, 1) over(order by salary desc) as LAG_VAL,
               LEAD(salary, 1) over(order by salary desc) as LEAD_VAL
        from employee
```

100 % <

	ename	salary	LAG_VAL	LEAD_VAL
1	e1	300	450	300
2	e10	500	500	480
3	e11	280	300	250
4	e12	300	300	280
5	e13	560	600	520
6	e14	250	280	250
7	e2	250	250	NULL
8	e3	500	500	500
9	e4	600	1000	560

핵심요약

1. 순위 함수

■ Top 함수

■ 질의 결과 튜플 수의 제한

- 질의 결과는 ORDER BY 절의 이용하여 정렬할 수 있음
- ORDER BY 정렬 기준에서 특정 등수 / 비율까지만 보고 싶은 경우
- Top(n) 함수를 이용함

```
SELECT TOP(n) 속성명
```

```
...
```

```
ORDER BY 속성명
```

- 동률이 있을 때 모두 보고 싶은 경우

- WITH TIES를 사용함

```
SELECT TOP(n) WITH TIES 속성명
```

```
...
```

```
ORDER BY 속성명
```

■ 정렬 기준 특정 비율까지만 보고 싶은 경우

- Top(n) PERCENT를 이용함
- 상위 n%까지만을 출력하게 됨
- WITH TIES와도 같이 쓸 수 있음

```
SELECT TOP(n) PERCENT [WITH TIES] 속성명
```

```
...
```

```
ORDER BY 속성명
```

핵심요약

1. 순위 함수

■ RANK 함수

- TOP() 함수를 쓰면 결과 수를 제한함
- TOP() 함수는 등수를 구할 수 없음
- RANK 함수 : 각 튜플에 등수를 표시함

```
RANK함수 over (order by 속성명 [asc|desc] )
```

■ RANK() 함수

```
SELECT 속성명, RANK () OVER (ORDER BY 속성명 [asc|desc] )
```

- 동률에 대하여 동일 등수 배정
- 비연속식 등수 배정

■ DENSE_RANK() 함수

```
SELECT 속성명, DENSE_RANK () OVER (ORDER BY 속성명)
```

- 동률에 대하여 동일 등수 배정
- 연속식 등수 배정

■ ROW_NUMBER() 함수

```
SELECT 속성명, ROW_NUMBER () OVER (ORDER BY 속성명)
```

- 동률에 대하여 임의 등수 배정
- 연속식 등수 배정

■ NTILE(n) 함수

- 전체 튜플을 num개로 균등 분할하여 순위 지정

핵심요약

2. 그룹 별 순위

■ 그룹 별 순위 지정

■ 기준 RANK함수 문법

- 전체 결과에 대한 속성값 기준 등수 지정이 됨

■ PARTITION BY 속성명

- 튜플들을 속성값에 따라서 그룹핑함

- 각 그룹에 대하여 순위 함수를 적용함

```
RANK() over (PARTITION BY dno ORDER BY salary desc)
```

■ 그룹별 특정 등수의 정보를 보고 싶은 경우

- WHERE 절을 같이 활용함

```
RANK() over (PARTITION BY dno ORDER BY salary desc) AS 속성명
```

...

WHERE 속성명 = 등수

■ 그룹 별 집단 함수

■ 그룹 별 집단 함수의 적용

```
SELECT 집단 함수 ~ GROUP BY~
```

■ PARTITION BY를 이용해서도 그룹 별 집단 함수를 적용할 수 있음

```
SELECT 집단함수() OVER (PARTITION BY 속성명)
```

■ 행 순서 함수

■ 정렬된 대상에서 특정 순위의 튜플들을 추출할 필요가 있을 때 사용되는 함수

■ FIRST_VALUE 함수

- 정렬 대상에서 첫 번째 데이터 추출

■ LAG / LEAD 함수

- 지정된 순서에서 선행 / 후행 데이터를 참조하는 함수



SQL 활용

인덱스와 뷰



한국기술교육대학교
온라인평생교육원

학습내용

- 인덱스
- 뷰

학습목표

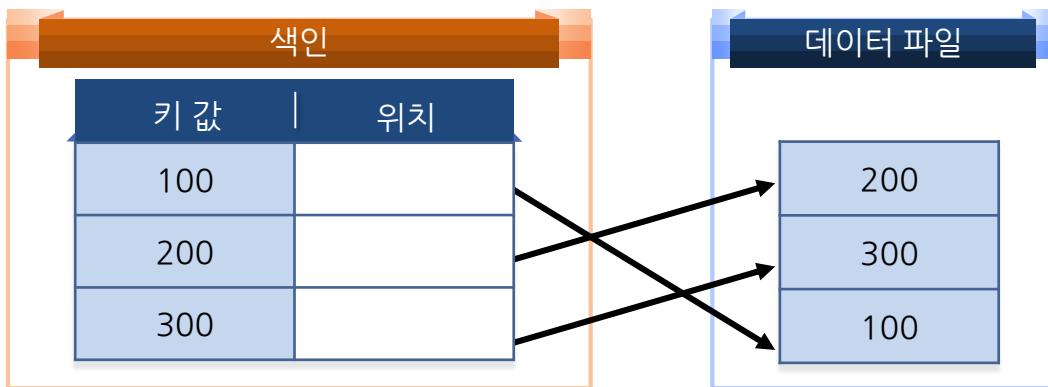
- 데이터에 빠르게 접근하기 위해 다양한 종류의 인덱스를 생성하고 활용할 수 있다.
- 뷰의 개념을 이해하고 뷰를 생성할 수 있다.

● 인덱스

1. 인덱스의 개념

◆ 인덱스(Index)

- 검색 성능을 향상 시키기 위한 부가적인 자료 구조
- 질의 명령문의 검색 속도를 향상시키기 위해 칼럼에 대해 생성하는 객체
- 포인터를 이용하여 테이블에 저장된 데이터를 랜덤 액세스하기 위한 목적으로 사용함



◆ 인덱스가 효율적인 경우

- WHERE 절이나 조인 조건절에서 자주 사용되는 칼럼의 경우
- 전체 데이터 중에서 10~15% 이내의 데이터를 검색하는 경우
- 두 개 이상의 칼럼이 WHERE 절이나 조인 조건에서 자주 사용되는 경우
- 테이블에 저장된 데이터의 변경이 드문 경우
 - 색인은 부가적인 자료 구조임
 - 데이터 삽입 시 비효율적임

● 인덱스

1. 인덱스의 개념

◆ 인덱스 생성 / 삭제 구문

● 색인 생성

```
CREATE INDEX 색인명  
ON 테이블명(속성명, 속성명, …)
```

● 색인 삭제

```
DROP INDEX 색인명  
ON 테이블명
```

● 인덱스

2. 인덱스의 종류

- ① 고유 인덱스 vs 비고유 인덱스
- ② 단일 인덱스 vs 결합 인덱스
- ③ DESCENDING INDEX
- ④ 집중 인덱스 vs 비집중 인덱스

◆ 고유 인덱스 vs 비고유 인덱스

- 고유 인덱스
 - 유일 값을 가지는 속성에 대하여 생성하는 색인
 - 각 키 값은 테이블의 하나의 튜플과 연관됨
- 비고유 인덱스
 - 중복된 값을 가지는 속성에 생성하는 인덱스
 - 키 값은 여러 개의 튜플들과 연관됨

● 기본키

- ① 테이블이 기본키에 대해서는 자동으로 고유색인이 생성됨
 - ⇒ Primary Index
 - 기본키는 중복을 허용하지 않음
- ② 새로운 튜플을 삽입 할 때마다 키값이 고유값인지 검사해야 함
- ③ 테이블에 속한 튜플들이 많다면 매우 느림



고유 색인을 이용함

● 관계형 테이블의 검색

- ① 테이블 검색 시 기본키만을 사용하지 않음
 - 예 학생 테이블에서 학번이 100번인 학생 검색하기
 - ② 실제로는 학생을 검색할 때는 학번보다 이름을 이용하는 경우가 더 많음
 - ③ 검색을 빨리 하려면 조건에 많이 사용되는 컬럼에 대하여 색인을 생성함
- ⇒ Secondary Index

● 인덱스

2. 인덱스의 종류

◆ 고유 인덱스 vs 비고유 인덱스

● 고유 인덱스의 생성

- 고유 인덱스를 생성할 때는 UNIQUE 키워드를 사용함

Q 부서 테이블에 부서 이름에 대하여 고유 색인 생성하기

```
CREATE UNIQUE INDEX idx_dname_unique  
ON DEPARTMENT (dname)
```

Messages

Command(s) completed successfully.

● 비고유 인덱스의 생성

- UNIQUE 없이 색인을 생성하면 비고유 색인이 됨

Q 부서 테이블에 부서 위치에 대하여 비고유 색인 생성하기

```
CREATE INDEX idx_loc_unique  
ON DEPARTMENT (loc)
```

Messages

Command(s) completed successfully.

● 인덱스

2. 인덱스의 종류

◆ 단일 인덱스 vs 결합인덱스

- 단일 인덱스
 - 하나의 속성만으로 구성된 색인
 - 앞에서 보인 예들은 단일 인덱스들임
- 결합 인덱스
 - 두 개 이상의 속성들에 대하여 생성된 색인
- 결합 인덱스의 생성



직원 테이블에서 부서 번호와 급여에 대하여 결합 인덱스 생성하기

```
CREATE INDEX idx_dnosalary  
ON EMPLOYEE (dno, salary)
```

Messages

Command(s) completed successfully.

● 인덱스

2. 인덱스의 종류

◆ DESCENDING INDEX

- 일반적인 색인들은 속성값에 대하여 오름차순으로 정렬되어 저장됨
 - DESCENDING INDEX : 특별히 속성별로 정렬 순서를 지정하여 결합 인덱스를 생성하는 방법
 - 색인 생성 시에 각 속성별로 정렬순서(DESC, ASC)를 정해줌
- DESCENDING INDEX의 생성

Q 사원에 대하여 부서 번호는 오름차순, 급여는 내림차순으로 하여 색인 생성하기

```
CREATE INDEX idx_dnosalary_desc  
ON EMPLOYEE(dno asc, salary desc)
```

Messages

Command(s) completed successfully.

◆ 집중 인덱스 vs 비집중 인덱스

- 집중 인덱스
 - 테이블의 튜플이 저장된 물리적 순서 해당 색인의 키값 순서와 동일하게 유지되도록 구성된 색인
 - 기본키에 대하여 생성된 색인은 집중 인덱스임
 - 테이블의 튜플들이 기본키에 오름차순으로 정렬되어 저장되어 있고 기본키 색인 또한 기본키에 따라서 오름차순으로 정렬되어 있음
 - 집중 인덱스는 하나의 테이블에 대하여 하나만 생성할 수 있음
- 비집중 인덱스
 - 집중 인덱스가 아닌 인덱스들

● 인덱스

3. 인덱스의 활용

- ◆ 질의 수행 시 인덱스를 사용하는지 확인하기

Q 사원 이름이 'e1'인 사원의 정보 검색하기

```
SELECT * FROM EMPLOYEE WHERE ENAME = 'e1'
```

```
USE MagicCorp
GO

SELECT * FROM EMPLOYEE WHERE ENAME = 'e1'
```

100 % <

결과 메시지

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	101	e1	staff	113	2007-03-01 00:00:00,000	300	NULL	20

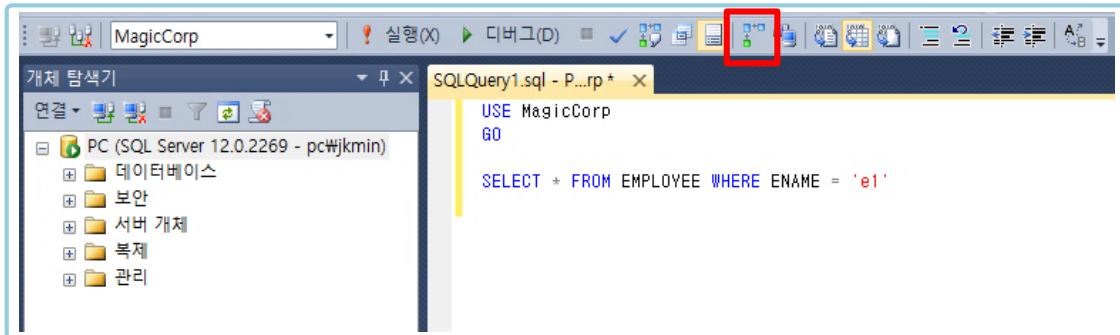
● 인덱스

3. 인덱스의 활용

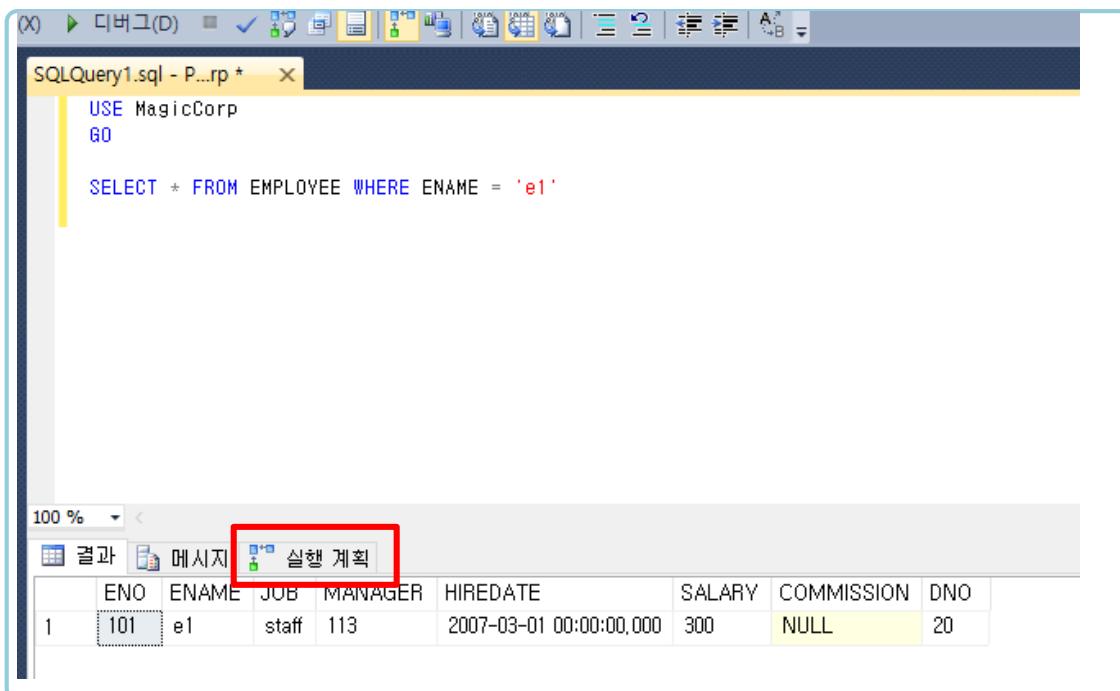
◆ 질의 수행 시 인덱스를 사용하는지 확인하기

● MS-SQL에서 질의 수행 계획 보는 방법

① 질의 수행 전 클릭해서 수행 계획 보기 선택



② 실행 계획 탭이 생김

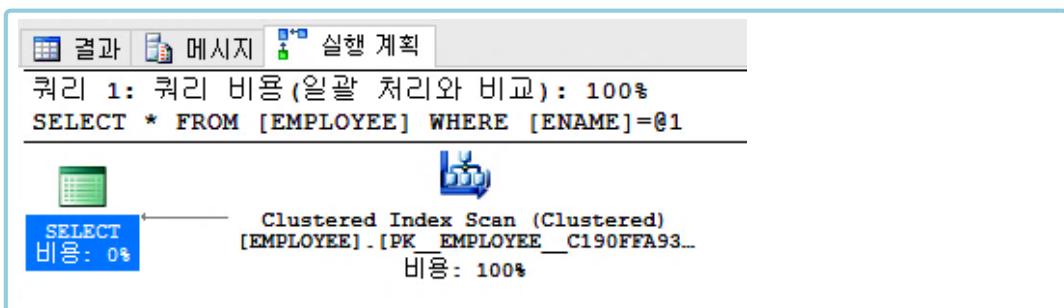


● 인덱스

3. 인덱스의 활용

◆ 질의 수행 시 인덱스를 사용하는지 확인하기

- MS-SQL에서 질의 수행 계획 보는 방법
 - 질의 수행 방법
 - 기본키(ENO)에 생성된 색인
 - 처음부터 scan하는 형태로 이용함



● 인덱스

3. 인덱스의 활용

◆ 질의 수행 시 인덱스를 강제로 사용하게 하기

- ① 질의는 eno로 탐색하는 것이 아니라 ENAME을 가지고 탐색하는 질의임

```
SELECT * FROM EMPLOYEE WHERE ENAME = 'e1'
```

- ② employee테이블의 ename을 가지고 색인 emp_name_idx를 만듦

⇒ 질의 수행기가 해당 색인을 사용하지 않음

강제로 emp_name_idx를 사용하게 할 수 없을까?

```
USE MagicCorp
GO

CREATE INDEX emp_name_idx
ON EMPLOYEE(ename)
```

- ③ FROM 절에 WITH(INDEX= INDEX_NAME)을 추가하여 강제로 특정 색인을 사용하게 함

- 앞선 질의에서 emp_name_idx를 사용하게 함

The screenshot shows the SQL query window and the results grid. The query is:

```
USE MagicCorp
GO

SELECT * FROM EMPLOYEE WITH (INDEX = emp_name_idx) WHERE ENAME = 'e1'
```

The results grid shows one row:

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	101	e1	staff	113	2007-03-01 00:00:00,000	300	NULL	20

- ④ 색인을 사용하는지 질의수행 계획 확인

- emp_name_idx를 통해서 이름이 e1인 튜플의 기본키값을 파악함 (index Seek)
- 파악된 기본키를 이용하여 기본 색인(PK_EMPLOYEE...)을 검색하여 결과를 찾도록 수행됨

● 뷰

1. 뷰의 개념

◆ 뷰(View)란?

- 하나 이상의 기본 테이블이나 다른 뷰를 이용하여 생성되는 가상 테이블
 - 기본 테이블은 디스크에 공간이 할당되어 데이터를 저장함
 - 뷰는 데이터 딕셔너리(Data Dictionary) 테이블에 뷰에 대한 정의(SQL문)만 저장되어 디스크 저장 공간 할당이 이루어지지 않음
 - 전체 데이터 중에서 일부만 접근할 수 있도록 함
 - 뷰에 대한 수정 결과는 뷰를 정의한 기본 테이블에 적용됨
 - 뷰를 정의한 기본 테이블에서 정의된 무결성 제약조건은 그대로 유지됨

◆ 뷰의 필요성

- 사용자마다 특정 객체만 조회할 수 있도록 할 필요가 있음
 - 모든 직원에 대한 정보를 모든 사원이 볼 수 있도록 하면 안 됨
 - 복잡한 질의문을 단순화 할 수 있음
 - 데이터의 중복성을 최소화할 수 있음
- 예) 판매부(Sale)에 속한 사원들을 따로 관리하고 싶은 경우
- ⇒ 판매부에 속한 사원들만을 사원테이블에서 찾아서 다른 테이블로 만들면 중복성이 발생함
 - ⇒ 이럴 때 뷰가 필요함

◆ 뷰의 장·단점

- 장점
 - 논리적 독립성을 제공함
 - 데이터의 접근 제어(보안)
 - 사용자의 데이터 관리 단순화
 - 여러 사용자의 다양한 데이터 요구 지원
- 단점
 - 뷰의 정의 변경 불가
 - 삽입, 삭제, 갱신 연산에 제한이 있음

● 뷰

1. 뷰의 개념

◆ 뷰의 생성

- 뷰의 생성 구문

```
CREATE VIEW 뷰이름  
AS SQL문(select 문)
```

- 뷰의 삭제 구문

```
DROP VIEW 뷰이름
```

Q 사원 테이블에 부서번호 30인 사원들의 뷰 생성하기

```
USE MagicCorp  
GO  
  
CREATE VIEW EMP30  
AS  
SELECT *  
FROM EMPLOYEE  
WHERE DNO = 30
```

Messages
Command(s) completed successfully.

Q 뷔를 이용하여 부서번호 30인 사원들 중 급여가 500이상인 사원들의 이름 구하기

```
USE MagicCorp  
GO  
  
SELECT ENAME  
FROM EMP30  
WHERE SALARY >= 500
```

Results Messages

ENAME
1 e3
2 e8

● 뷰

1. 뷰의 개념

◆ 뷰의 종류

- 단순 뷰
 - 하나의 기본 테이블 위에 정의된 뷰
- 복합 뷰
 - 두 개 이상의 기본 테이블로부터 파생된 뷰

◆ 뷰에 대한 간접 연산

- 무결성 제약 조건, 표현식, 집단연산, GROUP BY 절의 유무에 따라서 DML(Data Manipulation Language)문 사용이 제한적임
 - 데이터 조작 언어(DML : Data Manipulation Language) : INSERT, DELETE, UPDATE, SELECT 문과 같이 데이터의 삽입, 삭제, 변경, 검색을 할 수 있게 하는 데이터 조작문



사원 테이블에서 **평균 연봉**을 구하는 뷰 생성하기

```
USE MagicCorp
GO

CREATE VIEW EMPAVGSAL
AS
SELECT AVG(SALARY) AS SALAVG
FROM EMPLOYEE

Messages
Command(s) completed successfully.
```

● 뷰

1. 뷰의 개념

◆ 뷰에 대한 간접 연산

Q 평균 연봉 뷰에 대하여 평균 연봉 10 증가시키기

```
USE MagicCorp
GO

UPDATE EMPAVGSAL
SET SALAVG = SALAVG+10
```

Messages

Msg 4406, Level 16, State 1, Line 3
Update or insert of view or function 'EMPAVGSL' failed because it contains a derived or

- 뷰가 집단연산의 결과일 경우, 뷰를 통한 간접 연산은 불가능함

● 뷰

2. 인라인 뷰

◆ 인라인 뷰란?

- 하나의 질의문 내에서만 생성되어 사용 되어지고 질의문 수행 종료 후에는 사라지는 뷰
 - 뷰의 명시적인 선언(즉, Create View 문)이 없음
 - FROM 절에서 참조하는 테이블의 크기가 클 경우, 필요한 행과 속성만으로 구성된 집합으로 정의하여 질의문을 효율적으로 구성함
 - FROM 절에서 서브 쿼리를 사용하여 생성하는 임시 뷰

◆ 인라인 뷰의 예제

Q 부서별 평균 급여 파악하기

- ⇒ 부서 번호로 나와 있음
- ⇒ 부서명도 알고 싶음
- ⇒ 사원 테이블과 부서 테이블 조인이 필요함

```
SELECT DNO, AVG(salary) as AVG_SAL
FROM EMPLOYEE
GROUP BY DNO
```

Results

DNO	AVG_SAL
1	10
2	20
3	30

```
SELECT DNAME, AVG(SALARY)
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.DNO = E.DNO
GROUP BY DNAME
```

Results

DNAME	(No column name)
1 Accounting	423
2 Human	552
3 Sales	410

● 뷰

2. 인라인 뷰

◆ 인라인 뷰의 예제

Q 인라인 뷰를 이용하여 부서별 부서명, 평균 급여 출력하기

- FROM 절
 - inline view S 선언
 - 부서번호 및 평균
- WHERE 절
 - 부서테이블과 S와의 조인

```
SELECT DNAME, AVG_SAL
  FROM (SELECT DNO, AVG(salary) as AVG_SAL
         FROM EMPLOYEE
        GROUP BY DNO) as S, DEPARTMENT D
 WHERE S.DNO = D.DNO
```

The screenshot shows a database query results window. At the top, there are tabs for 'Results' and 'Messages'. The 'Results' tab is selected, displaying a table with three rows. The table has two columns: 'DNAME' and 'AVG_SAL'. The data is as follows:

	DNAME	AVG_SAL
1	Accounting	423
2	Human	552
3	Sales	410

● 뷰

2. 인라인 뷰

◆ WITH 절

- 인라인 뷰의 또 다른 정의 방법
 - FROM 절에 임시 질의 결과를 정의하는 대신 WITH 절을 이용하여 임시 테이블을 생성함

WITH 임시테이블명(속성명)
AS (SELECT ~ FROM ~ WHERE)

Q WITH 절을 사용하여 부서별 급여평균, 부서명을 출력하기

- WITH 절의 AS문 이후의 질의 결과를 S라는 임시 테이블로 생성함
- 메인 질의문에서는 S 테이블과 부서 테이블의 조인으로 표현함

```
WITH S (DNO, AVG_SAL)
AS (SELECT DNO, AVG(salary)
     FROM EMPLOYEE
     GROUP BY DNO)
SELECT DNAME, AVG_SAL
      FROM S, DEPARTMENT
     WHERE DEPARTMENT.DNO = S.DNO
```

The screenshot shows the SQL Server Management Studio interface with the 'Results' tab selected. The results grid displays the following data:

	DNAME	AVG_SAL
1	Accounting	423
2	Human	552
3	Sales	410

● 뷰

2. 인라인 뷰

◆ 뷰의 정의 보기

- 뷔의 정의 내용을 보고 싶을 경우
 - SP_HELPTEXT라는 저장 프로시저를 이용함
- 저장 프로시저를 수행하는 명령문
 - EXEC

Q EMP30 뷔의 정의 파악하기

The screenshot shows the SSMS interface with the following details:

- The query window contains the command: `EXEC SP_HELPTEXT EMP30`.
- The results pane is displayed, showing the definition of the EMP30 view.
- The results pane has two tabs: "Results" (selected) and "Messages".
- The results table has a single column named "Text".
- The output is as follows:

Text
1 CREATE VIEW EMP30
2 AS
3 SELECT *
4 FROM EMPLOYEE
5 WHERE DNO = 30

핵심요약

1. 인덱스

■ 인덱스의 개념

■ 인덱스의 개념

- 검색 성능을 향상 시키기 위한 부가적인 자료 구조
- SQL 명령문의 검색 속도를 향상시키기 위해 칼럼에 대해 생성하는 객체
- 포인트를 이용하여 테이블에 저장된 데이터를 랜덤 액세스하기 위한 목적으로 사용함

■ 인덱스가 효율적인 경우

- WHERE 절이나 조인 조건절에서 자주 사용되는 칼럼의 경우
- 전체 데이터 중에서 10~15% 이내의 데이터를 검색하는 경우
- 두 개 이상의 칼럼이 WHERE 절이나 조인 조건에서 자주 사용되는 경우
- 테이블에 저장된 데이터의 변경이 드문 경우

■ 색인 생성

```
CREATE INDEX 색인명  
ON 테이블명(속성명, 속성명, …)
```

■ 색인 삭제

```
DROP INDEX 색인명  
ON 테이블명
```

핵심요약

1. 인덱스

■ 인덱스의 종류

■ 고유 인덱스

- 유일 값을 가지는 속성에 대하여 생성하는 색인
- 각 키 값은 테이블의 하나의 튜플과 연관됨

■ 비고유 인덱스

- 중복된 값을 가지는 속성에 생성하는 인덱스
- 키 값은 여러 개의 튜플들과 연관됨

■ 단일 인덱스

- 하나의 속성만으로 구성된 색인
- 앞에서 보인 예들은 단일 인덱스들임

■ 결합 인덱스

- 두 개 이상의 속성들에 대하여 생성된 색인

■ DESCENDING INDEX

- 일반적인 색인들은 속성값에 대하여 오름차순으로 정렬되어 저장됨
- 특별히 속성별로 정렬 순서를 지정하여 결합 인덱스를 생성하는 방법
- 색인 생성 시에 각 속성별로 정렬순서(DESC, ASC)를 정해줌

■ 집중 인덱스

- 테이블의 튜플이 저장 된 물리적 순서 해당 색인의 키값 순서와 동일하게 유지되도록 구성된 색인
- 기본키에 대하여 생성된 색인은 집중 인덱스임
- 테이블의 튜플들이 기본키에 오름차순으로 정렬되어 저장되어 있고 기본키 색인 또한 기본키에 따라서 오름차순으로 정렬되어 있음
- 집중 인덱스는 하나의 테이블에 대하여 하나만 생성할 수 있음

■ 비집중 인덱스

- 집중 인덱스가 아닌 인덱스들

핵심요약

1. 인덱스

■ 인덱스의 활용

- 질의 수행 시 인덱스를 사용하는지 확인하기
 - ① 질의 수행 시 인덱스를 강제로 사용하게 하기
 - ② employee테이블의 ename을 가지고 색인 emp_name_idx을 만듦
 - ③ FROM 절에 WITH(INDEX= INDEX_NAME)을 추가하여 강제로 특정 색인을 사용하게 함
 - ④ 색인을 사용하는지 질의수행 계획 확인

핵심요약

2. 뷰

■ 뷰의 개념

■ 뷰의 개념

- 하나 이상의 기본 테이블이나 다른 뷰를 이용하여 생성되는 가상 테이블
- 기본 테이블은 디스크에 공간이 할당되어 데이터를 저장함
- 뷰는 데이터 딕셔너리(Data Dictionary) 테이블에 뷰에 대한 정의(SQL문)만 저장되어 디스크 저장 공간 할당이 이루어지지 않음
- 전체 데이터 중에서 일부만 접근할 수 있도록 함
- 뷰에 대한 수정 결과는 뷰를 정의한 기본 테이블에 적용됨
- 뷰를 정의한 기본 테이블에서 정의된 무결성 제약조건은 그대로 유지됨

■ 뷰의 필요성

- 사용자마다 특정 객체만 조회할 수 있도록 할 필요가 있음
- 복잡한 질의문을 단순화 할 수 있음
- 데이터의 중복성을 최소화할 수 있음

■ 뷰의 장점

- 논리적 독립성을 제공함
- 데이터의 접근 제어(보안)
- 사용자의 데이터 관리 단순화
- 여러 사용자의 다양한 데이터 요구 지원

■ 뷰의 단점

- 뷰의 정의 변경 불가
- 삽입, 삭제, 갱신 연산에 제한이 있음

핵심요약

2. 뷰

■ 뷰의 개념

■ 뷰의 생성 구문

```
CREATE VIEW 뷰이름  
AS SQL문(select 문)
```

■ 뷔의 삭제 구문

```
DROP VIEW 뷔이름
```

■ 뷔의 종류

- 단순 뷔 : 하나의 기본 테이블 위에 정의된 뷔
- 복합 뷔 : 두 개 이상의 기본 테이블로부터 파생된 뷔

■ 뷔에 대한 간접 연산

- 무결성 제약 조건, 표현식, 집단연산, GROUP BY 절의 유무에 따라서 DML문의 사용이 제한적임

핵심요약

2. 뷰

■ 인라인 뷰

■ 인라인 뷰란?

- 하나의 질의문 내에서만 생성되어 사용 되어지고 질의문 수행 종료 후에는 사라지는 뷰
- 뷰의 명시적인 선언(즉, Create View 문)이 없음
- FROM 절에서 참조하는 테이블의 크기가 클 경우, 필요한 행과 속성만으로 구성된 집합으로 정의하여 질의문을 효율적으로 구성함
- FROM 절에서 서브 쿼리를 사용하여 생성하는 임시 뷰

■ WITH 절

- 인라인 뷰의 또 다른 정의 방법
- FROM 절에 임시 질의 결과를 정의하는 대신 WITH 절을 이용하여 임시 테이블을 생성함

WITH 임시테이블명(속성명)
AS (SELECT ~ FROM ~ WHERE)

■ 뷰의 정의 보기

- 뷰의 정의 내용을 보고 싶을 경우 SP_HELPTEXT라는 저장 프로시저를 이용함
- 저장 프로시저를 수행하는 명령문 : EXEC



SQL 활용

사용자 관리



한국기술교육대학교
온라인평생교육원

학습내용

- 보안
- 권한 부여

학습목표

- 보안에 대한 기본 개념을 설명할 수 있다.
- 사용자 권한 부여를 위한 DCL문을 작성할 수 있다.

● 보안

1. 통제

◆ 보안

- 불법적인 데이터의 폭로나 변경 또는 파괴로부터 데이터베이스를 보호하는 것

◆ 보안에 대한 통제

- ① 법적, 윤리적 통제
 - 법, 윤리 \Rightarrow 심리적 보안
- ② 행정, 관리적 통제
 - 오용을 탐지하고 방지함
- ③ 물리적 통제
 - 적극적, 물리적 보안으로 위반을 예방, 탐지함
- ④ 기술적 통제
 - 하드웨어 통제
 - 소프트웨어 통제
 - 데이터베이스 통제
 - DBMS 보안 서브 시스템 \Rightarrow 접근 제어

● 보안

2. 접근 제어

◆ 권한이 부여되지 않은 데이터의 검색이나 변경을 방지함

① 직접 접근 제어

- 사용자 신분증 확인(ID)
- 신분증 본인 확인을 위한 인증(PASSWORD)
- 요청 데이터 객체에 대한 요청 연산 권한(권한 부여)

② 간접 접근 제어

- 한 장소에서 다른 장소로의 데이터 흐름 제어
- 개인의 비밀 데이터로부터 작성된 통계정보에 대한 추론 제어
- 전송이나 저장 데이터의 암호화 시스템 작동과 사용자 상호작용의 감시

◆ 접근 제어 구조

- 신분증
 - 지문, 성문, ID
- 인증
 - 권한 부여 테이블
 - 사용자, 접근 가능한 데이터와 연산
 - 데이터베이스 정보
 - 요구되는 연산
 - 메인 메모리에 있는 권한 부여 테이블
 - 사용자 활동 로깅

◆ 권한 부여 규정

- 권한 부여 규정은 DCL로 명세함
- 명세된 규정은 데이터 딕셔너리(Data Dictionary)에서 관리함

● 권한 부여

1. 뷰 기반 기법

◆ 뷰 기반 기법이란?

- 뷰를 이용한 권한 부여
- 특정 뷰에 대하여 특정 사용자만 보도록 지정함
- 민감한 데이터를 권한이 없는 사용자로부터 은닉할 수 있음
- 릴레이션의 수직적 / 수평적 서브셋을 제한할 수 있음

◆ 뷰 기반 기법의 문제점

```
CREATE VIEW ST1  
AS SELECT SNO, NAME, SAL  
FROM STUDENT  
WHERE YEAR ≤ 4
```

◆ 튜플 삽입의 제약

```
INSERT INTO S1(SNO, NAME, YEAR): <'E5', 'LEE', 5>
```

◆ 뷰 기반 기법의 문제점

- 알려진 값의 NULL 값
- ST는 DEPTNO가 12인 뷰인데 삽입될 때는 12대신 NULL이 들어감

```
CREATE VIEW ST2  
AS SELECT SNO, YEAR  
FROM STUDENT  
WHERE DEPTNO=12
```

```
INSERT INTO ST2 (SNO, YEAR):  
<'E5', 2>
```

● 권한 부여

2. GRANT / REVOKE 기법

◆ GRANT / REVOKE

- 특정 데이터와 연산을 특정 사용자만 수행할 수 있도록 권한 부여하는 DCL 문
 - GRANT문
 - 자신에게 허용된 권한을 다른 사용자에게 부여하는 구문
 - REVOKE문
 - 다른 사용자에게 허용한 권한을 철회하는 구문
 - DENY문
 - 다른 사용자에게 특정 권한을 불허하는 구문

◆ GRANT 구문

```
GRANT [권한ALL] ON 데이터객체 TO 사용자
```

- 데이터객체가 테이블 또는 뷰일 경우
 - 권한 : SELECT, INSERT, UPDATE, DELETE, REFERENCE 등 사용 권한
- 데이터객체가 데이터베이스일 경우
 - 권한 : CREATE [DB, TABLE, VIEW] 등의 권한
 - 주의점 : DROP 권한은 일반적으로 생성자(주인)만 가짐
- ALL : 모든 권한을 말함

◆ REVOKE / DENY 구문

```
REVOKE 권한 ON 데이터객체 TO 사용자
```

```
DENY 권한 ON 데이터객체 TO 사용자
```

● 권한 부여

3. MS-SQL에서의 권한 부여

◆ 인증 모드

- MS-SQL은 인증과 권한 부여가 분리되어 있음
 - **인증**을 위해서는 로그인 객체가 필요함
 - ⇒ 인증 : MS-SQL Server에 접속할 수 있다는 것
 - **권한 부여**를 위해서는 사용자 객체가 필요함
 - 인증되었다고 모든 객체(테이블 등)을 접근할 수 있다는 것은 아님
 - 사용자마다 다른 권한을 가질 수 있음
- 윈도우 인증
 - 별도의 ID나 비밀번호 없이 Windows에 접속한 사용자로 MS-SQL에 연결할 수 있도록 하는 인증방식
- SQL-Server 인증
 - 윈도우 인증과는 무관하게 SQL-Server에 등록된 로그인 계정으로 인증
 - Windows 운영체제의 보안과는 상관없이 SQL-Server 계정으로 접속 가능
 - 보안의 취약성으로 MS사에서는 권장하지 않음
 - 실무에서는 SQL-Server 인증을 빈번히 사용함
 - 보안이 상대적으로 취약하지만 외부 컴퓨터에서 SQL-Server에 접근하여 사용하려면 SQL-Server 인증이 보다 편리함

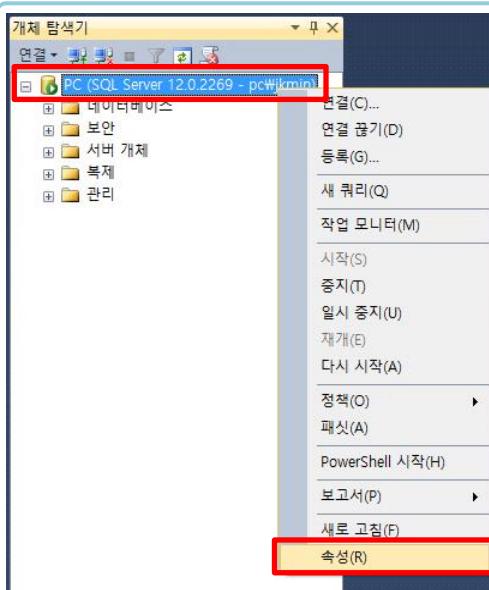
컴퓨터 전체를 사용할 수 있는 윈도우 인증보다는 DBMS 인증이 낫다.

● 권한 부여

3. MS-SQL에서의 권한 부여

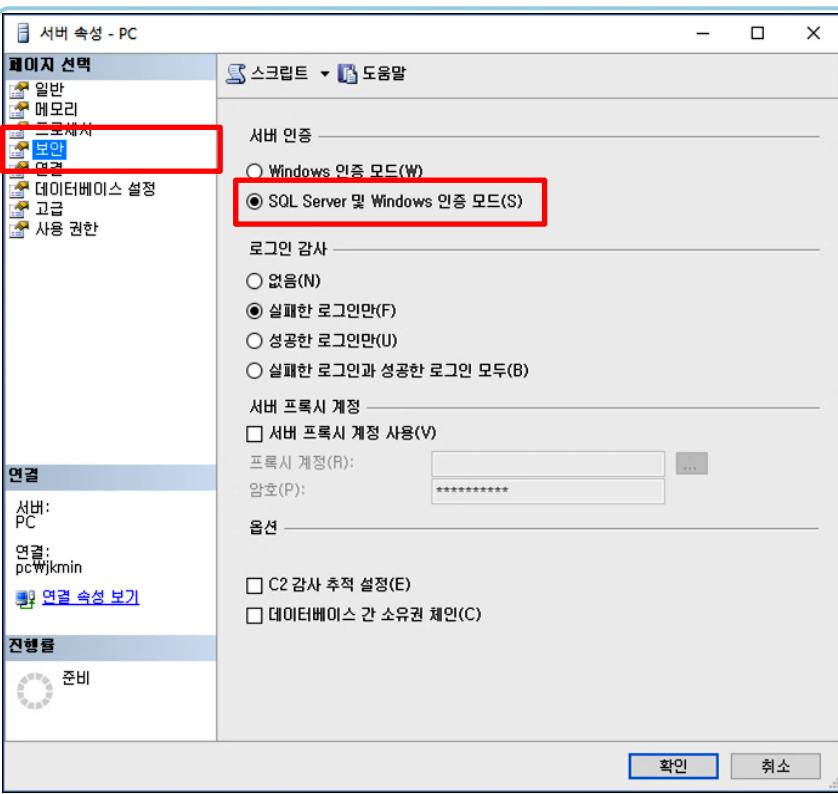
◆ 인증 모드

● 인증모드 변경



● 인증모드 변경

⇒ 이후 컴퓨터 재시작 필수



● 권한 부여

3. MS-SQL에서의 권한 부여

◆ DB 사용자

- 로그인이 되었다고 MS-SQL Server가 관리하는 모든 데이터베이스들을 자동 접근할 수 있다면 심각한 보안 위협이 됨



데이터베이스 별로 사용자 등록을 해야 함

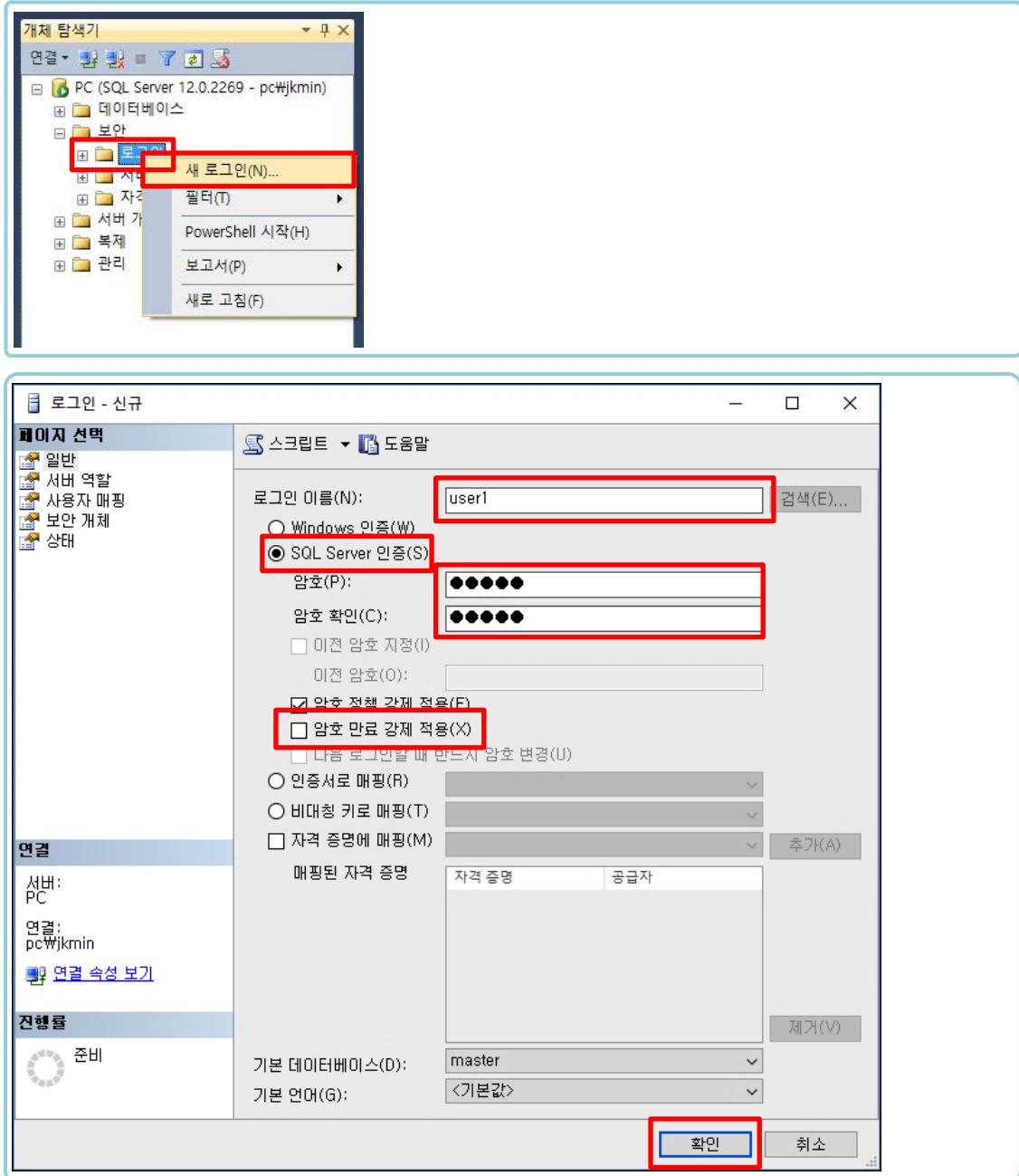
◆ Login 객체 생성 및 사용자 등록

- SSMS를 이용하여 user1 로그인 객체를 생성하고 MagicCorp 데이터베이스에 사용자를 등록함
 - SQL-Server 인증 방식으로 만듦

● 권한 부여

3. MS-SQL에서의 권한 부여

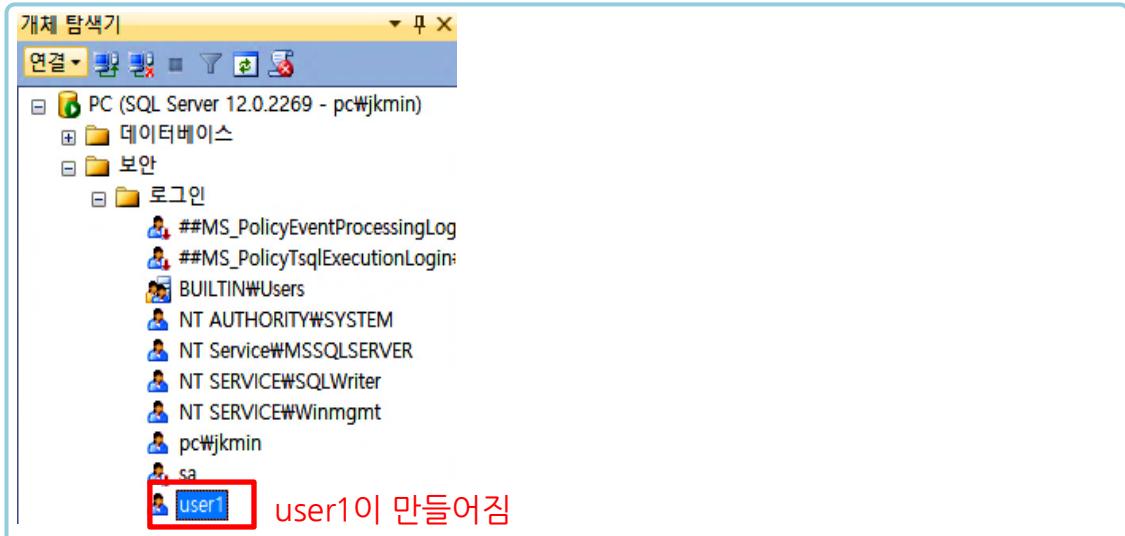
① Login 객체 생성



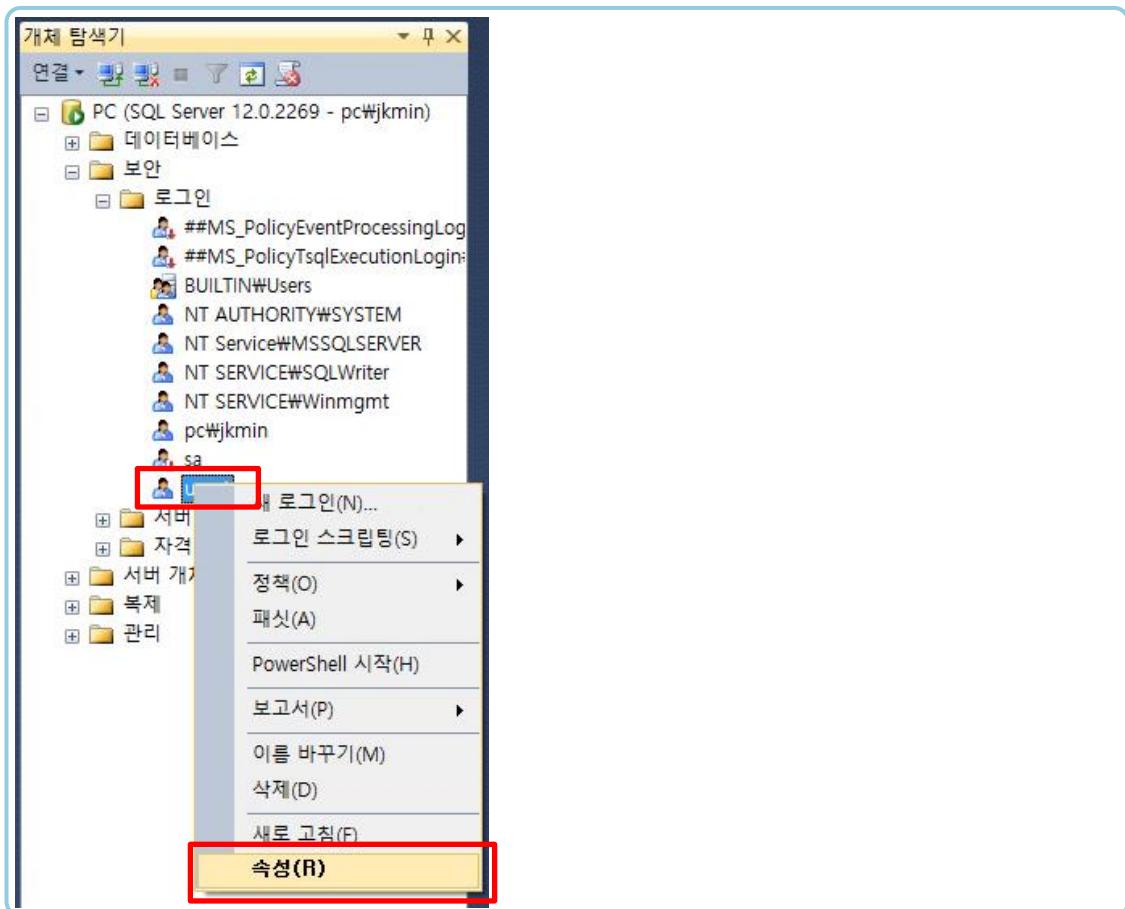
● 권한 부여

3. MS-SQL에서의 권한 부여

① Login 객체 생성



② 사용자 등록

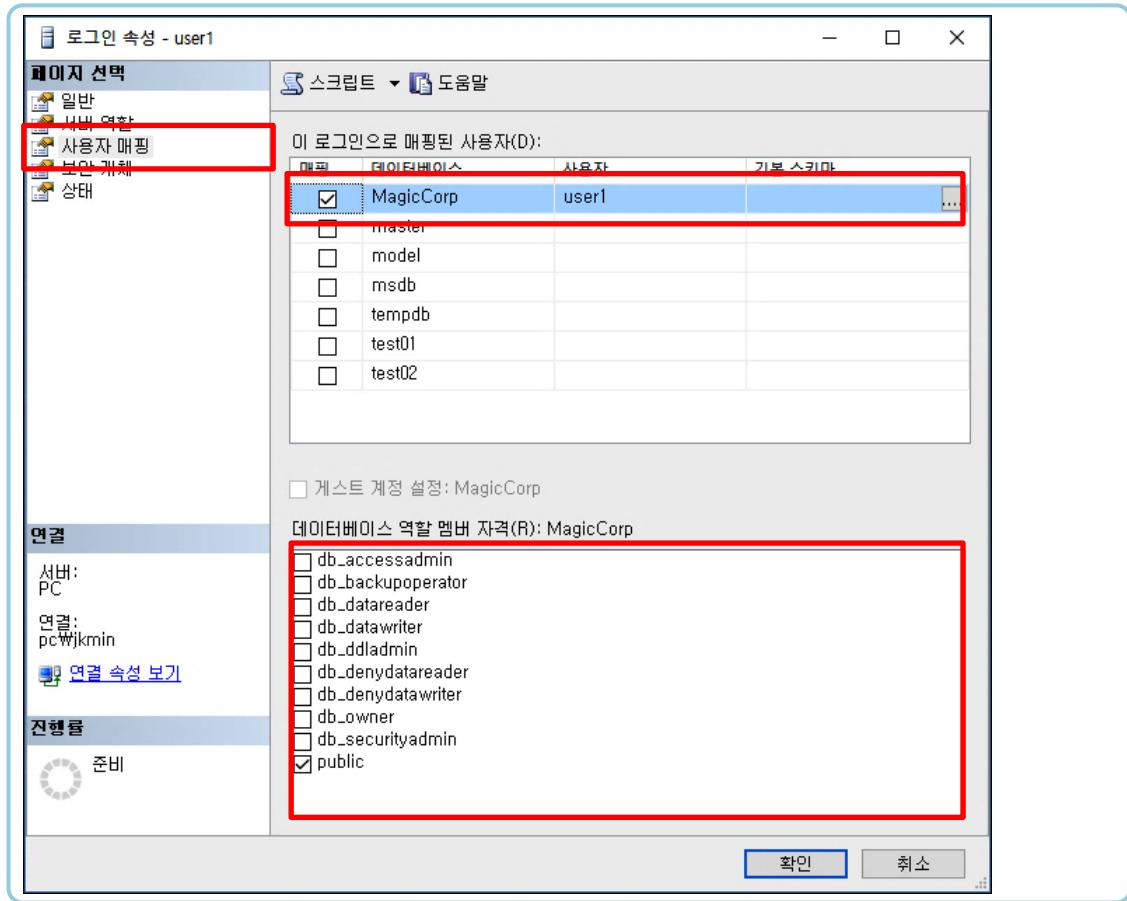


● 권한 부여

3. MS-SQL에서의 권한 부여

② 사용자 등록

- 데이터베이스 역할(Rule) 멤버 자격
 - 사용자가 해당 DB에 어떤 역할인지 지정함

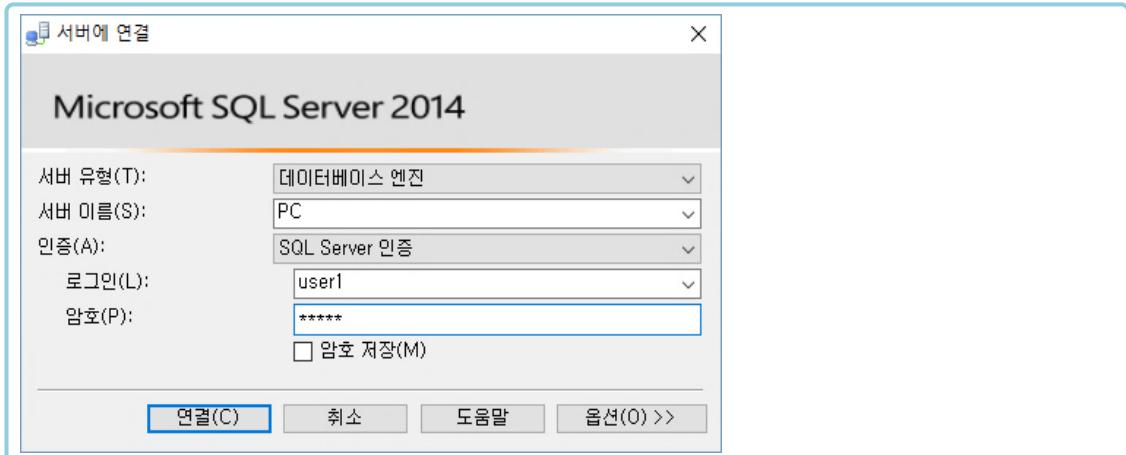


- 데이터베이스 역할(Rule) 멤버 자격
 - 주요 역할 멤버
 - db_accessadmin : 로그인에 대한 추가나 제거 권한
 - db_owner : DB의 모든 구성 및 유지 작업 가능
 - public : 디폴트로 부여되는 최소한의 권한

● 권한 부여

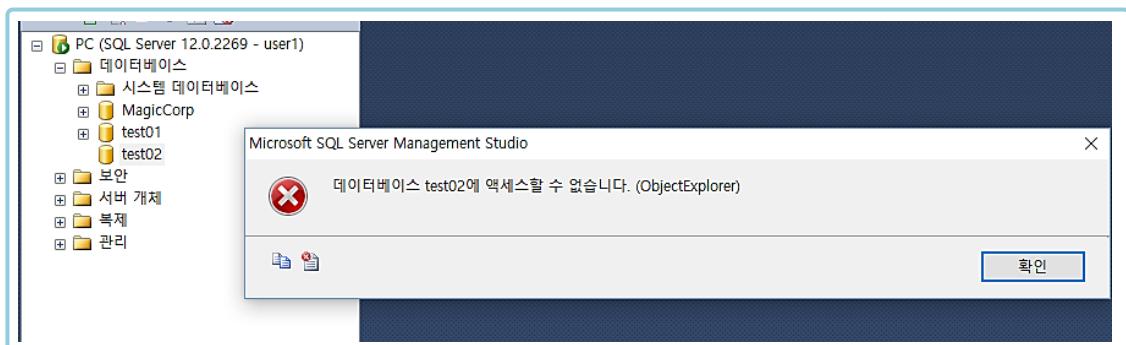
3. MS-SQL에서의 권한 부여

- ③ MS-SQL 재 시작 후 SQL-Server 인증 방식으로 해서 user1 로그인

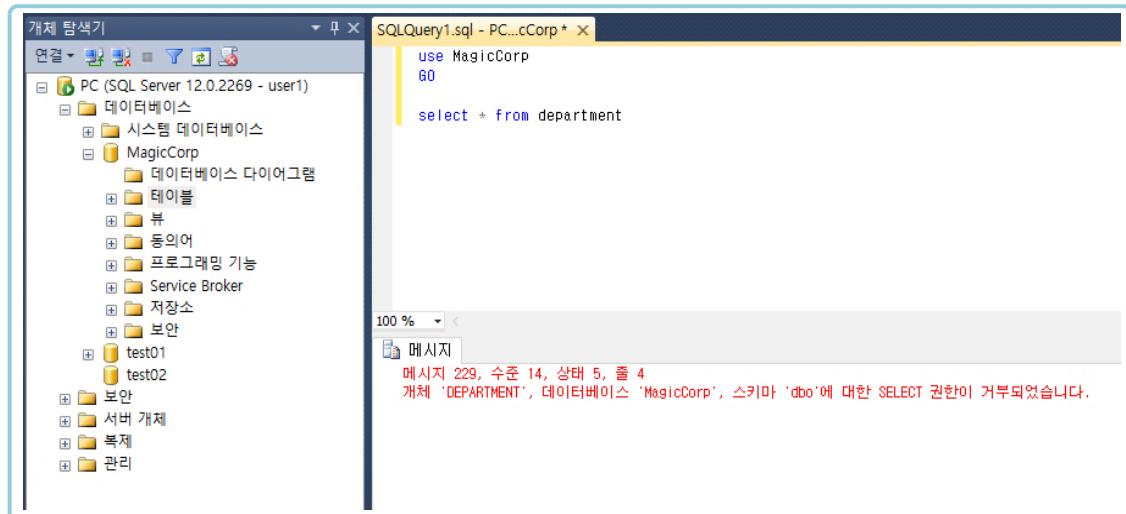


- ④ user1 로그인

- test02 DB에는 역할이 없으므로 접근 불가



- MagicCorp에 접근은 가능하지만 권한 부여된 것이 없어서 테이블 접근은 불가능함



● 권한 부여

3. MS-SQL에서의 권한 부여

⑤ T-SQL을 이용한 로그인 객체와 사용자 등록

- MS-SQL을 종료하고 재 시작함
- Windows 인증 모드로 연결하여 관리자가 되어야 함
- user3 계정을 만듦

```
USE master
GO

CREATE LOGIN [user3]
WITH PASSWORD = '12345',
DEFAULT_DATABASE = [master],
CHECK_POLICY = ON,
CHECK_EXPIRATION = OFF
GO

USE MagicCorp
GO
CREATE USER [user3]
FOR LOGIN [user3]
GO
```

100 % < 메시지
명령이 완료되었습니다.

⑥ user3에게 DEPARTMENT 테이블에 대한 검색(Select) 및 수정(Update) 권한 부여(T-SQL 이용)

- 허가를 거부하거나 해지하려면 DENY 또는 REVOKE를 씀

```
use MagicCorp
GO

GRANT select, update
ON department
to user3
GO
```

100 % < 메시지
명령이 완료되었습니다.

● 권한 부여

3. MS-SQL에서의 권한 부여

- ⑦ user3로 로그인하여 DEPARTMENT 검색

The screenshot shows a SQL query window titled "SQLQuery1.sql - PC...cCorp *". The query is:

```
use MagicCorp  
GO  
select * from DEPARTMENT
```

The results grid displays the following data:

	DNO	DNAME	LOC
1	10	Accounting	Seoul
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

- ⑧ user3로 로그인하여 DEPARTMENT에 새로운 튜플 삽입

- insert 권한이 없음으로 삽입이 불가능 함

The screenshot shows a SQL query window titled "SQLQuery1.sql - PC...cCorp *". The query is:

```
use MagicCorp  
GO  
insert into DEPARTMENT values (50, 'newPart', 'Incheon')
```

The results grid shows an error message in the "메시지" (Message) tab:

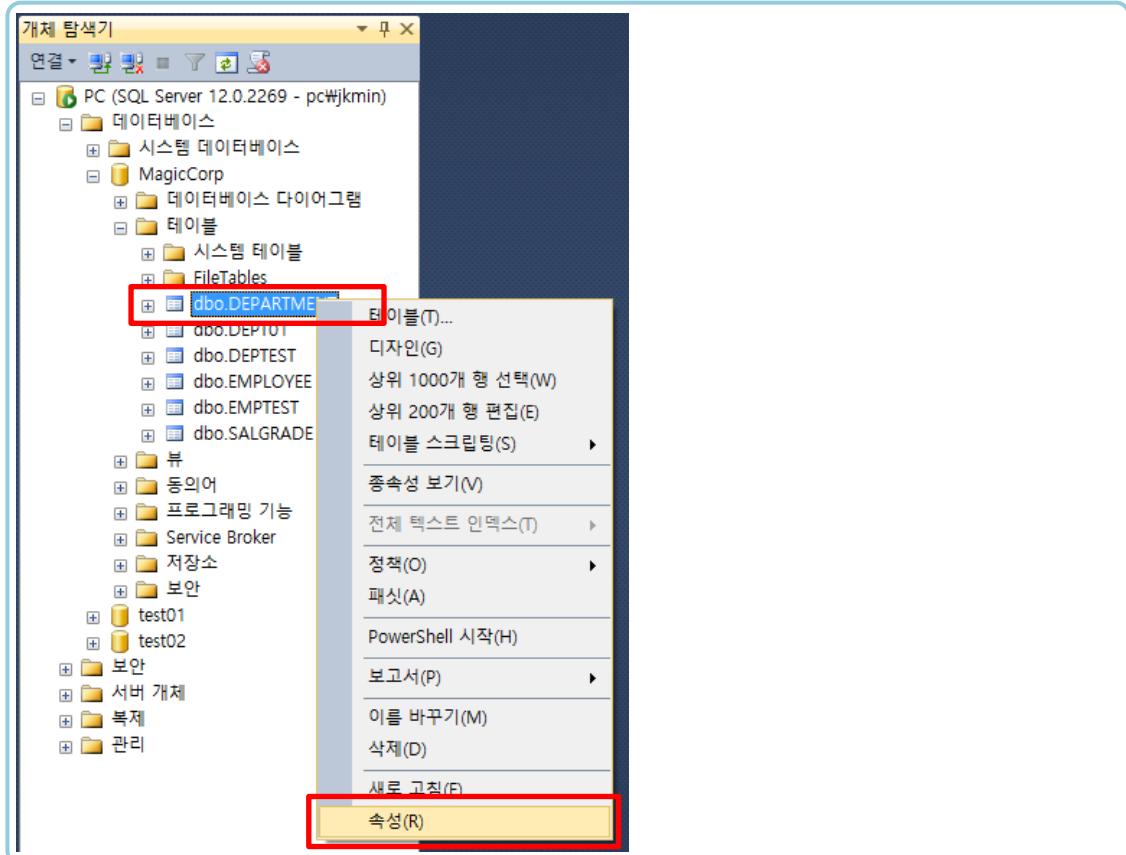
메시지 229, 수준 14, 상태 5, 줄 4
개체 'DEPARTMENT', 데이터베이스 'MagicCorp', 스키마 'dbo'에 대한 INSERT 권한이 거부되었습니다.

● 권한 부여

3. MS-SQL에서의 권한 부여

⑨ 관리자로 로그인하여 user3의 DEPARTMENT 테이블에 insert 권한 추가 부여

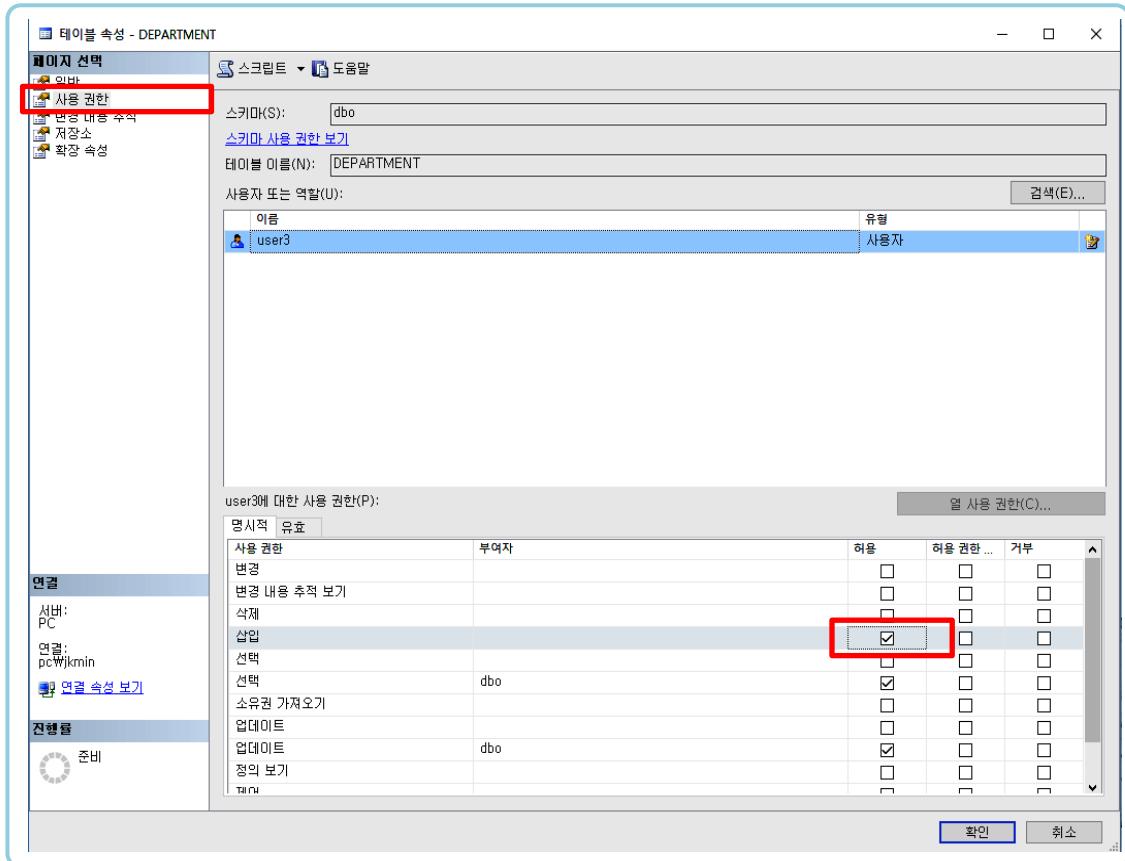
- insert 권한이 없음으로 삽입이 불가능 함



● 권한 부여

3. MS-SQL에서의 권한 부여

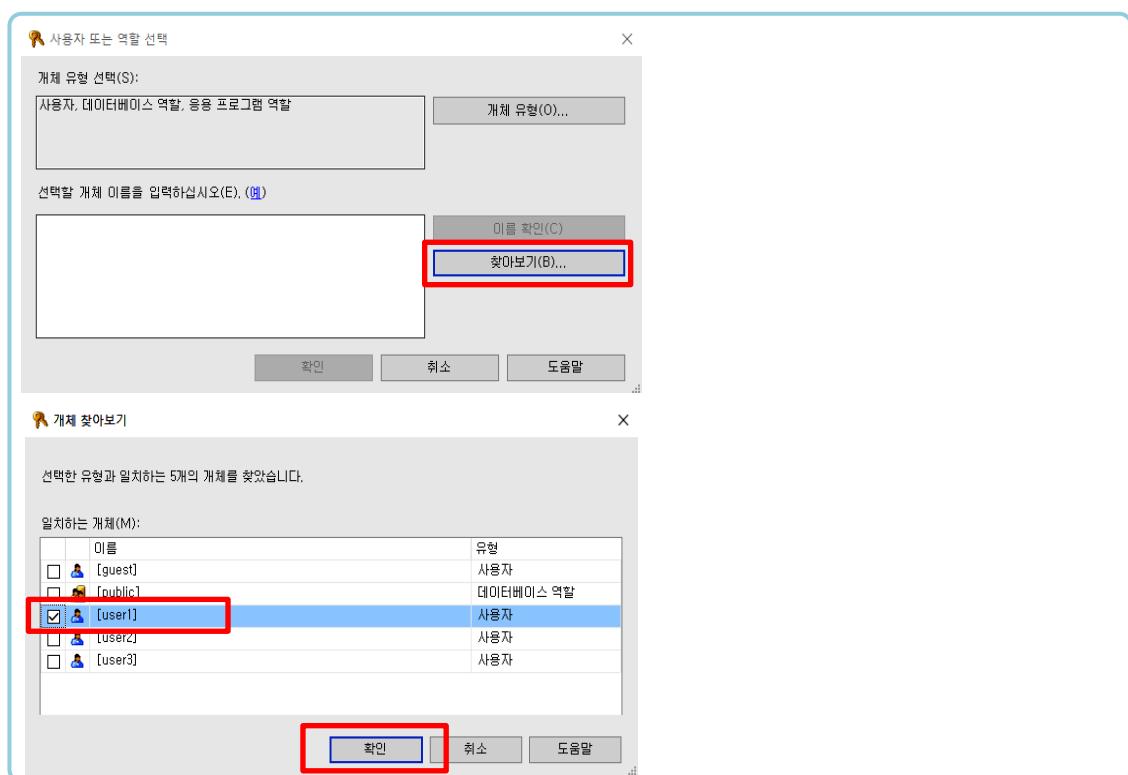
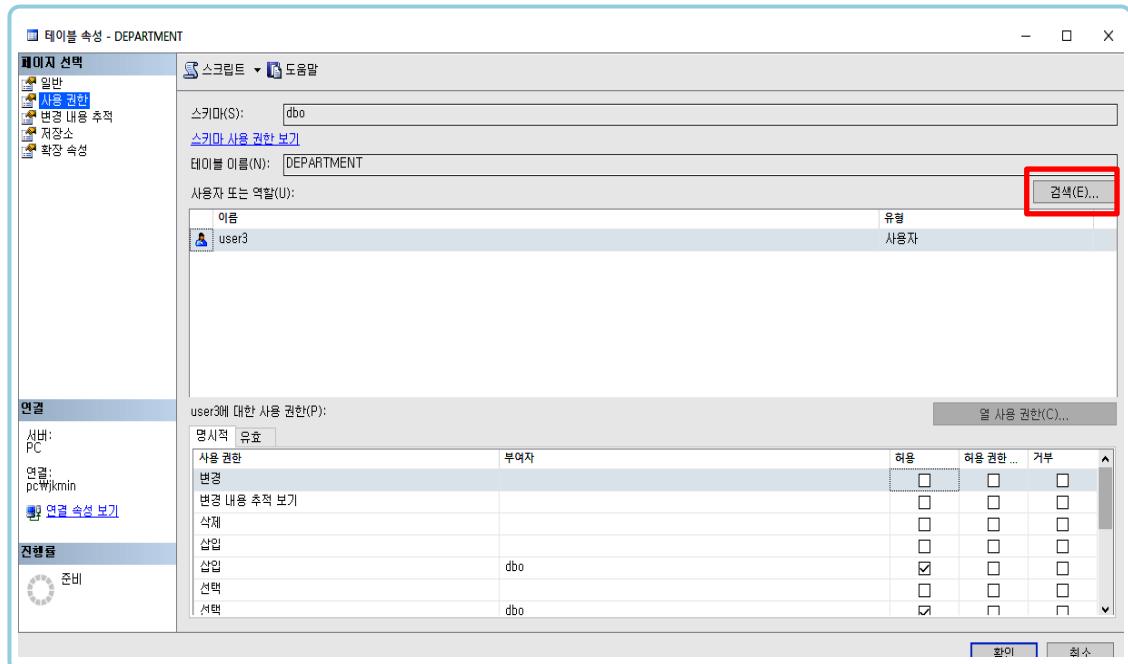
- ⑨ 관리자로 로그인하여 user3의 DEPARTMENT 테이블에 insert 권한 추가 부여



● 권한 부여

3. MS-SQL에서의 권한 부여

- ⑩ 관리자로 로그인하여 user1에게도 DEPARTMENT 테이블에 select insert 권한 추가 부여



● 권한 부여

3. MS-SQL에서의 권한 부여

- ⑩ 관리자로 로그인하여 user1에게도 DEPARTMENT 테이블에 select insert 권한 추가
부여

사용자 또는 역할(U):

이름	유형
user1	사용자
user3	사용자

user1에 대한 사용 권한(P):

명시적		默示적	
사용 권한	부여자	허용	허용 권한 ...
변경		<input type="checkbox"/>	<input type="checkbox"/>
변경 내용 추적 보기		<input type="checkbox"/>	<input type="checkbox"/>
삭제		<input type="checkbox"/>	<input type="checkbox"/>
삽입		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
선택		<input type="checkbox"/>	<input type="checkbox"/>
소유권 가져오기		<input type="checkbox"/>	<input type="checkbox"/>
업데이트		<input type="checkbox"/>	<input type="checkbox"/>

확인

- ⑪ 이후 user1, user3로 로그인 시 DEPARTMENT 테이블에 대한 검색 / 삽입이 가능해짐

핵심요약

1. 보안

■ 통제

■ 보안

- 테이블을 구성하는 튜플 집합에 대한 테이블의 부분 집합을 결과로 반환하는 연산자

■ 보안에 대한 통제

① 법적, 윤리적 통제

- 법, 윤리 \Rightarrow 심리적 보안

② 행정, 관리적 통제

- 오용을 탐지하고 방지함

③ 물리적 통제

- 적극적, 물리적 보안으로 위반을 예방, 탐지함

④ 기술적 통제

- 하드웨어 통제

- 소프트웨어 통제

- 데이터베이스 통제 : DBMS 보안 서브 시스템 \Rightarrow 접근 제어

핵심요약

1. 보안

■ 접근 제어

■ 권한이 부여되지 않은 데이터의 검색이나 변경을 방지함

① 직접 접근 제어

- 사용자 신분증 확인(ID)
- 신분증 본인 확인을 위한 인증(PASSWORD)
- 요청 데이터 객체에 대한 요청 연산 권한(권한 부여)

② 간접 접근 제어

- 한 장소에서 다른 장소로의 데이터 흐름 제어
- 개인의 비밀 데이터로부터 작성된 통계정보에 대한 추론 제어
- 전송이나 저장 데이터의 암호화 시스템 작동과 사용자 상호작용의 감시

■ 접근 제어 구조

- 신분증 : 지문, 성문, ID
- 인증 : 권한 부여 테이블(사용자, 접근 가능한 데이터와 연산)

데이터베이스 정보

요구되는 연산(메인 메모리에 있는 권한 부여 테이블, 사용자 활동 로깅)

■ 권한 부여 규정

- 권한 부여 규정은 DCL로 명세함
- 명세된 규정은 데이터 딕셔너리에서 관리함

핵심요약

2. 권한 부여

■ 뷰 기반 기법

- 뷰 기반 기법이란?
 - 뷰를 이용한 권한 부여
 - 특정 뷰에 대하여 특정 사용자만 보도록 지정함
 - 민감한 데이터를 권한이 없는 사용자로부터 은닉할 수 있음
 - 릴레이션의 수직적 / 수평적 서브셋을 제한할 수 있음

■ 뷰 기반 기법의 문제점

```
CREATE VIEW ST1  
    AS SELECT SNO, NAME, SAL  
        FROM STUDENT  
        WHERE YEAR = 4
```

- 튜플 삽입의 제약

```
INSERT INTO ST1(SNO, NAME, YEAR): <'E5', 'LEE', 5>
```

■ 뷰 기반 기법의 문제점

- 알려진 값의 NULL값 : ST는 DEPTNO가 12인 뷰인데 삽입될 때는 12대신 NULL이 들어감

```
CREATE VIEW ST2  
    AS SELECT SNO, YEAR  
        FROM STUDENT  
        WHERE DEPTNO=12
```

```
INSERT INTO ST2 (SNO, YEAR):  
<'E5', 2>
```

핵심요약

2. 권한 부여

■ GRANT / REVOKE 기법

- 특정 데이터와 연산을 특정 사용자만 수행할 수 있도록 권한 부여하는 DCL 문
 - GRANT문 : 자신에게 허용된 권한을 다른 사용자에게 부여하는 구문
 - REVOKE문 : 다른 사용자에게 허용한 권한을 철회하는 구문
 - DENY문 : 다른 사용자에게 특정 권한을 불허하는 구문

■ GRANT 구문

GRANT [권한|ALL] ON 데이터객체 TO 사용자

- 데이터객체가 테이블 또는 뷰일 경우 : SELECT, INSERT, UPDATE, DELETE, REFERENCE 등 사용 권한
- 데이터객체가 데이터베이스일 경우 : CREATE [DB, TABLE, VIEW] 등의 권한
 - ▶ 주의점 : DROP 권한은 일반적으로 생성자(주인)만 가짐
- ALL : 모든 권한을 말함

■ REVOKE / DENY 구문

REVOKE 권한 ON 데이터객체 TO 사용자

DENY 권한 ON 데이터객체 TO 사용자

핵심요약

2. 권한 부여

■ MS-SQL에서의 권한 부여

■ 인증 모드

- MS-SQL은 인증과 권한 부여가 분리되어 있음

▶ 인증을 위해서는 로그인 객체가 필요함

▶ 권한 부여를 위해서는 사용자 객체가 필요함

- 윈도우 인증

▶ 별도의 ID나 비밀번호 없이 Windows에 접속한 사용자로 MS-SQL에

연결할 수 있도록 하는 인증방식

- SQL-Server 인증

▶ 윈도우 인증과는 무관하게 SQL-Server에 등록된 로그인 계정으로 인증

▶ Windows 운영체제의 보완과는 상관없이 SQL-Server 계정으로 접속 가능

▶ 보완의 취약성으로 MS사에서는 권장하지 않음

▶ 실무에서는 SQL-Sever 인증을 빈번히 사용함

- 인증모드 변경 이후 컴퓨터 재시작 필수임

■ DB 사용자

- 로그인이 되었다고 MS-SQL Server가 관리하는 모든 데이터베이스들을 자동

접근할 수 있다면 심각한 보안 위협이 됨

▶ MS-SQL은 인증과 권한 부여가 분리되어 있음

■ Login 객체 생성 및 사용자 등록

- SSMS를 이용하여 user1 로그인 객체를 생성하고 MagicCorp 데이터베이스에

사용자를 등록함

▶ SQL-Server 인증 방식으로 만듦



SQL 활용

저장 프로시저와 사용자 정의 함수



한국기술교육대학교
온라인평생교육원

학습내용

- 프로시저
- 사용자 정의 함수

학습목표

- 프로시저의 개념을 이해하고 프로시저의 매개변수를 활용하여 프로시저를 작성할 수 있다.
- 사용자 정의 함수와 프로시저의 차이점을 알고, 테이블을 반환하는 함수를 작성할 수 있다.

● 프로시저

1. 프로시저의 개념

◆ 프로시저

- 자주 사용되는 질의문들을 하나로 묶어서 저장해두고 필요할 때마다 명령문처럼 실행할 수 있도록 해주는 것
 - 선택적으로 매개변수를 받아 일련의 질의문을 실행시켜 결과를 돌려주는 것
 - 범용 언어의 프로시저 ⇒ 함수와 유사한 개념

◆ 일반 질의문과의 차이점

- 일반 질의문
 - 사용자 또는 응용 프로그램이 실행하고자 하는 SQL문을 DBMS에 전송하고 그 결과를 받음
 - 대량의 복잡한 질의문들이 반복적으로 입력되면 그만큼 시스템에 부담이 됨
 - DBMS에도 처리해야 하는 일이 늘어남
- 프로시저
 - 프로시저 내용은 DBMS에 포함되어 있고 실행 방안도 미리 작성되어 있음
 - 사용자나 응용 프로그램은 프로시저 이름과 매개변수 값(필요 시)만을 전송하면 됨
⇒ 복잡한 SQL문의 단순화

◆ 프로시저 생성 구문

```
CREATE [PROCEDURE|PROC] 프로시저이름
AS
BEGIN SQL문 END //BEGIN END는 SQL문이 하나만 있다면 생략 가능
```

● 프로시저

1. 프로시저의 개념

◆ 프로시저 실행 문법

```
EXEC 프로시저이름
```

◆ 프로시저 수정 문법

```
ALTER PROCEDURE 프로시저이름  
AS SQL문
```

```
DROP PROCEDURE 프로시저이름
```

● 프로시저

1. 프로시저의 개념

◆ 단순 프로시저 생성 및 호출

Q MagicCorp DB에 employ 테이블에서 사번이 109인 사원의 이름, 직급, 급여를 검색하는 emp_pro 프로시저 생성하기

```
USE MagicCorp
GO

CREATE PROCEDURE emp_pro
AS
SELECT ENAME, JOB, SALARY
FROM EMPLOYEE
WHERE ENO = 109
```

Messages
Command(s) completed successfully.

Q emp_pro 호출하기

```
USE MagicCorp
GO

EXEC EMP_PRO
```

Results Messages

	ENAME	JOB	SALARY
1	e9	ceo	1000

Q emp_pro 프로시저를 수정하여 사원번호 110번의 정보 검색하기

```
USE MagicCorp
GO

ALTER PROCEDURE emp_pro
AS
SELECT ENAME, JOB, SALARY
FROM EMPLOYEE
WHERE ENO = 110
```

→

```
USE MagicCorp
GO

EXEC EMP_PRO
```

Results Messages

	ENAME	JOB	SALARY
1	e10	section	500

Messages
Command(s) completed successfully.

● 프로시저

2. 프로시저의 매개변수

◆ 매개 변수

- 프로시저 실행 시 조건값 등을 변경 할 수 없을까?
 - 사원번호 109번에 대한 정보를 추출하는 저장 프로시저를 생성하고 이를 110번에 대한 정보를 추출하도록 저장 프로시저 변경
⇒ 저장 프로시저 수행 시 사원번호를 입력으로 주어 해당 사원정보를 추출하도록 할 수 없을까?



매개 변수를 사용함

- 저장 프로시저 수행 시 수행 질의문에 특정 값을 매개변수로 전달할 수 있도록 하여 다양한 조건을 하나의 질의문으로 수행할 수 있도록 지원해 줌

◆ 입력 매개변수의 선언

● 생성

```
CREATE PROCEDURE 프로시저이름  
@매개변수명 타입, …  
AS SQL문
```

● 실행

```
EXEC 프로시저이름 매개변수값
```

● 프로시저

2. 프로시저의 매개변수

◆ 입력 매개변수 프로시저 예제

- Q MagicCorp DB에 employ 테이블에서 임의의 사번을 입력 받아 해당 사원의 사번, 이름, 직급, 급여를 검색하는 emp_pro_para 프로시저 생성하기

```
USE MagicCorp
GO

CREATE PROCEDURE emp_pro_para
@enumber int
AS
SELECT ENO, ENAME, JOB, SALARY
FROM EMPLOYEE
WHERE ENO = @enumber
```

Messages
Command(s) completed successfully.

- Q 사원번호 101, 102등 임의 사원번호를 입력하여 emp_pro_para 수행하기

```
USE MagicCorp
GO

EXEC EMP_PRO_PARA 101
EXEC EMP_PRO_PARA 102
```

Results Messages

	ENO	ENAME	JOB	SALARY
1	101	e1	staff	300

	ENO	ENAME	JOB	SALARY
1	102	e2	deputy	250

● 프로시저

2. 프로시저의 매개변수

◆ 출력 매개변수란?

- 입력 매개변수와 반대로 프로시저의 처리 결과값을 반환하는 매개변수

◆ 출력 매개변수 선언

```
CREATE PROCEDURE 프로시저이름  
@매개변수명 타입 OUTPUT, ...  
AS  
SELECT @매개변수명= 속성명  
FROM ... WHERE...
```

◆ 출력 매개변수

- 출력 매개변수 값 받기
 - 프로시저 실행 전에 매개변수를 선언함(DECLARE 문 이용)
 - 선언된 매개변수를 출력함(SELECT 문 이용)
 - 프로시저 실행 전에 매개변수를 선언함(DECLARE 문 이용)

```
DECLARE @매개변수명
```

- 프로시저 실행

```
EXEC 프로시저명 @매개변수명 OUTPUT
```

- 선언된 매개변수를 출력함(SELECT 문 이용)

```
SELECT @매개변수명
```

● 프로시저

2. 프로시저의 매개변수

◆ 출력 매개변수 프로시저 예제

- Q 사원 테이블에서 부서번호인 사원들의 평균급여를 알아내는 프로시저 emp_out_put_para 작성하기

```
USE MagicCorp
GO

CREATE PROC emp_out_put_para
@did int,
@avg_sal int OUTPUT
AS
SELECT @avg_sal= AVG(salary)
FROM EMPLOYEE
WHERE EMPLOYEE.DNO = @did
```

- Q emp_out_put_para를 호출하여 그 결과 출력하기

- 호출 전 출력 매개변수 값을 받을 변수 선언 필요

```
USE MagicCorp
GO

DECLARE @AVG_SALY INT

EXEC emp_out_put_para 30, @AVG_SALY OUTPUT

SELECT @AVG_SALY
```

100 % <

결과		메시지
(열 이름 없음)		
1	410	

● 사용자 정의 함수

1. 프로시저와의 차이점

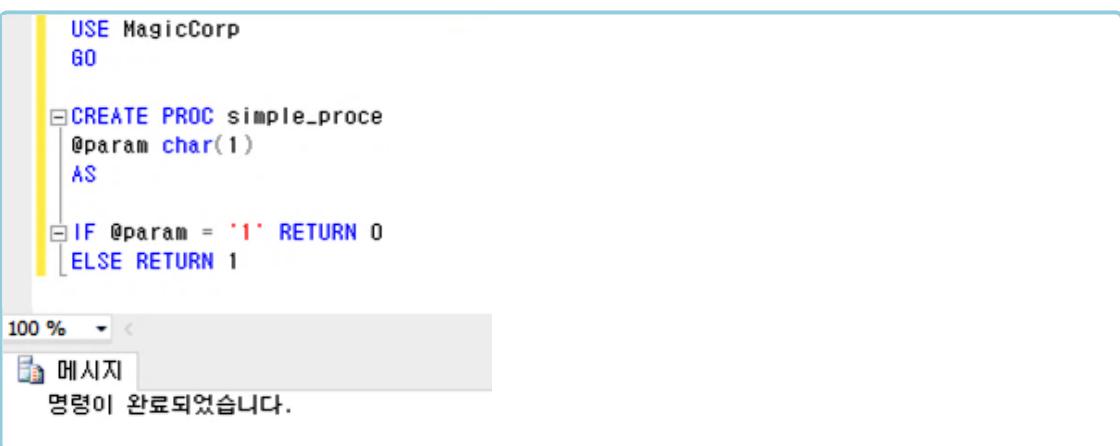
◆ 사용자 정의 함수

- 프로시저와 달리 RETURN문을 이용하여 하나의 값을 반드시 반환해야 함

◆ 프로시저

- RETURN문을 이용하여 값을 반환함

⇒ 프로시저의 반환 값은 int 형만 가능하고 프로시저의 결과가 아닌 상태값 만을 반환함



```
USE MagicCorp
GO

CREATE PROC simple_proce
@param char(1)
AS

IF @param = '1' RETURN 0
ELSE RETURN 1
```

◆ 사용자 정의 함수의 특징

- ① 사용자 정의 함수는 프로시저와 달리 다양한 타입의 값을 반환할 수 있음
- ② 함수이므로 질의문 내에서 사용이 가능함

◆ 사용자 정의 함수의 선언

- 기본적으로 프로시저와 유사함

```
CREATE FUNCTION 함수명
(@매개변수명 타입, … )
RETURNS 반환타입
AS
[BEGIN] SQL 문 [END]
```

● 주의점

- SQL문 내에는 반드시 RETURN문이 있어야 함

● 사용자 정의 함수

1. 프로시저와의 차이점

◆ 사용자 정의 함수의 예

Q 임의의 부서 번호를 입력하면 해당 부서 사원들의 최대 급여를 반환하는 함수 MAX_SAL 생성하기

```
USE MagicCorp
GO

CREATE FUNCTION MAX_SAL
(@param int)
RETURNS int
AS
BEGIN
DECLARE @MAX_VAL int

SELECT @MAX_VAL = MAX(salary)
FROM EMPLOYEE
WHERE DNO = @param

RETURN @MAX_VAL
END
```

100 % < 메시지
명령이 완료되었습니다.

Q MAX_SAL 함수를 이용하여 부서번호 30의 최대 급여와 동일한 급여를 받는 사원 정보를 출력하기

```
USE MagicCorp
GO

SELECT *
FROM EMPLOYEE
WHERE SALARY = dbo.MAX_SAL(30)
```

100 % < 결과
결과 메시지

	ENO	ENAME	JOB	MANAGER	HIREDATE	SALARY	COMMISSION	DNO
1	103	e3	section	105	2005-02-10 00:00:00,000	500	100	30
2	108	e8	senior	103	2004-03-08 00:00:00,000	500	0	30
3	110	e10	section	103	2005-04-07 00:00:00,000	500	NULL	10

- 사용자 정의 함수 호출 시에는 반드시 접두사로 “dbo.”을 넣어야 함

● 사용자 정의 함수

2. 테이블 반환 함수

◆ 테이블 반환 함수

- 함수 결과로 테이블을 반환하는 함수

◆ 프로시저 + 뷰

- 함수 결과로 테이블이 반환됨으로 SELECT 문의 FROM 절 등에서 쓸 수 있음
⇒ 뷰와 유사한 성격

◆ 테이블 반환 함수의 선언

```
CREATE FUNCTION 함수명
( @매개변수명 타입, ….)
RETURNS @반환변수 TABLE (테이블정의)
AS
[BEGIN] SQL 문 [END]
```

● 사용자 정의 함수

2. 테이블 반환 함수

◆ 테이블 반환 함수의 예

Q 부서번호를 입력 받아 해당 부서 사원들의 **사원번호와 이름을 반환하는 사용자 정의 함수** EMP_DEPT 생성하기

```
USE MagicCorp
GO

CREATE FUNCTION EMP_DEPT
(@param int)
RETURNS @emp_dept_table TABLE (
    emp_id int,
    emp_name varchar(20)
)
AS
BEGIN

    INSERT INTO @emp_dept_table
    SELECT EMPLOYEE.ENO, EMPLOYEE.ENAME
    FROM EMPLOYEE
    WHERE DNO = @param

    RETURN
END
```

100 % <

메시지
명령이 완료되었습니다.

Q 사용자 정의 함수 EMP_DEPT를 이용하여 20번 부서의 **사원번호, 사원명 출력하기**

```
USE MagicCorp
GO

SELECT *
FROM dbo.EMP_DEPT(20)
```

	결과	메시지
1	emp_id 101	emp_name e1
2	104	e4
3	109	e9
4	112	e12
5	113	e13

핵심요약

1. 프로시저

■ 프로시저의 개념

- 자주 사용되는 질의문들을 하나로 묶어서 저장해두고 필요할 때마다 명령문처럼 실행할 수 있도록 해주는 개념
 - 선택적으로 매개변수를 받아 일련의 질의문을 수행해서 결과를 돌려주는 것
 - 범용 언어의 프로시저 / 함수와 유사한 개념임

■ 일반 질의문과의 차이점

① 일반 질의문

- 사용자 또는 응용 프로그램이 실행하고자 하는 SQL문을 DBMS에 전송하고 그 결과를 받음
- 대량의 복잡한 질의문들이 반복적으로 입력되면 그만큼 시스템에 부담이 됨
- DBMS에도 처리해야 하는 일이 늘어나 시스템에 더 큰 부담이 됨

② 프로시저

- 프로시저 내용은 DBMS에 포함되어 있고 실행 방안도 미리 작성되어 있음
- 사용자나 응용 프로그램은 프로시저 이름과 매개변수 값(필요 시)만을 전송하면
⇒ 복잡한 SQL문의 단순화

핵심요약

1. 프로시저

■ 프로시저의 개념

■ 프로시저 생성 구문

```
CREATE [PROCEDURE|PROC] 프로시저이름  
AS  
BEGIN SQL문 END //BEGIN END는 SQL문이 하나만 있다면 생략 가능
```

■ 프로시저 실행 문법

```
EXEC 프로시저이름
```

■ 프로시저 수정 문법

```
ALTER PROCEDURE 프로시저이름  
AS SQL문
```

```
DROP PROCEDURE 프로시저이름
```

핵심요약

1. 프로시저

■ 프로시저의 매개변수

- 프로시저 실행 시 조건값 등을 변경 할 수 없을까?
 - 사원번호 109번에 대한 정보를 추출하는 저장 프로시저를 생성하고 이를 110번에 대한 정보를 추출하도록 저장 프로시저 변경
 - 저장 프로시저 수행 시 사원번호를 입력으로 주어 해당 사원정보를 추출하도록 할 수 없을까?
 - ⇒ 매개 변수를 사용함
 - 저장 프로시저 수행 시 수행 질의문에 특정 값을 매개변수로 전달할 수 있도록 하여 다양한 조건을 하나의 질의문으로 수행할 수 있도록 지원해 줌

■ 입력 매개변수의 선언

- 생성

```
CREATE PROCEDURE 프로시저이름  
@매개변수명 타입, ...  
AS SQL문
```

- 실행

```
EXEC 프로시저이름 매개변수값
```

핵심요약

1. 프로시저

■ 프로시저의 매개변수

■ 출력 매개변수

- 입력 매개변수와 반대로 프로시저의 처리 결과값을 반환하는 매개변수
- 출력 매개변수 선언

```
CREATE PROCEDURE 프로시저이름  
@매개변수명 타입 OUTPUT, ...  
AS  
SELECT @매개변수명= 속성명  
FROM ... WHERE...
```

- 출력 매개변수 값 받기

- ▶ 프로시저 실행 전에 매개변수를 선언함(DECLARE문 이용)
- ▶ 선언된 매개변수를 출력함(SELECT문 이용)
- ▶ 프로시저 실행 전에 매개변수를 선언함(DECLARE문 이용)

```
DECLARE @매개변수명
```

- 프로시저 실행

```
EXEC 프로시저명 @매개변수명 OUTPUT
```

- 선언된 매개변수를 출력함(SELECT 문 이용)

```
SELECT @매개변수명
```



SQL 활용

트리거



한국기술교육대학교
온라인평생교육원

학습내용

- 무결성 규정
- 트리거 활용

학습목표

- 무결성의 의미 및 무결성 규정에 대하여 설명할 수 있다.
- 트리거의 구동 방법을 이해하고, 특정 사건에 대한 트리거를 작성할 수 있다.

● 무결성 규정

1. 무결성의 의미

◆ 무결성

- 정밀성, 정확성, 정당성
- 허가 받은 사용자가 수행하는 갱신 작업에서 의미적 오류를 방지함
- 의미적 제약의 개념

◆ 무결성 서브 시스템



- 일관성
 - 데이터베이스에서 병행 트랜잭션들이 상호 작용으로부터 의미상 모순이 없도록 하는 것
- 신뢰성
 - 시스템의 오동작으로부터 오류가 발생하지 않도록 하는 것
- DBMS의 한 구성 요소
- 무결성 규정을 유지 관리함
- 데이터베이스의 무결성을 유지함
- 트랜잭션이 수행하는 갱신 연산이 무결성 규정을 위반하지 않는가를 감시함
 - 위반 시 거부, 보고, 취소 / 복귀를 수행함

● 무결성 규정

2. 제약 조건

◆ 무결성 규정 대상

① 도메인

- 형식
- 타입
- 범위

② 기본키, 외래키

- 개체 무결성(Entity Integrity)
- 참조 무결성(Referencial Integrity)

③ 종속성(묵시적 제약조건)

- 함수 종속
- 다치 종속
- 조인 종속

④ 관계

- 내부 관계
- 외부 관계

◆ 도메인 무결성 대상

● 도메인 정의

- 도메인 이름
- 데이터 형

● 삽입이나 갱신 연산에 적용

● 무결성 규정

2. 제약 조건

◆ 릴레이션의 무결성 규정

- 릴레이션을 조작하는 과정에서의 의미적 제약조건을 명세함
- 연산 수행 전 / 후에 대한 제약 조건을 규정함
 - 삽입
 - 삭제
 - 갱신
- 분류
 - ① 상태 제약과 과도 제약
 - ② 집합 제약과 튜플 제약
 - ③ 즉시 제약과 지연 제약

◆ 상태 제약과 과도 제약

- 상태 제약
 - 릴레이션 상태에 대한 제약
 - 일관성 있는 상태 유지
 - 정적 제약
 - 각 릴레이션 상태가 모두 만족해야 하는 규정
 - 데이터베이스 상태의 유효성



● 무결성 규정

2. 제약 조건

◆ 상태 제약과 과도 제약

● 상태 제약

- 키 속성의 제약 : 유일성
- NULL 값의 제약 : 이름은 NULL 값일 수 없음

▪ 관계 제약 : 참조 무결성

예 모든 학생은 하나의 학과에만 속함

▪ 도메인 제약 : 유효한 값

예 학생의 학년은 1, 2, 3, 4 중에 하나이어야 함

▪ 의미 무결성 제약

예 직원의 월급은 그의 관리자의 월급보다 높을 수 없음

● 과도 제약

▪ 동적 제약

- 데이터베이스의 한 상태에서 다른 상태로 변환되는 과정에서 적용되는 규정
- 데이터베이스 상태의 변환 전과 후의 비교
 - 변환 전과 후에 모두 적용됨

예 월급은 감소될 수 없음

◆ 집합 제약과 튜플 제약

● 집합 제약

▪ 튜플 집합 전체에 대한 제약

예 직원 전체의 평균 급여는 300을 넘을 수 없음

● 튜플 제약

▪ 처리되고 있는 튜플에만 적용됨

예 직원 급여의 최대는 500을 넘을 수 없음

⇒ 한 직원의 급여를 변경할 때 500이 넘는지만 감시하면 됨

● 무결성 규정

2. 제약 조건

◆ 즉시 제약과 지연 제약

- 즉시 제약

- 삽입 / 삭제 / 갱신 연산이 수행될 때마다 적용되는 제약 규정

- 지연 제약

- 트랜잭션이 완전히 수행된 후에 적용되는 제약 규정

● 트리거 활용

1. 트리거의 개념

◆ 트리거(TRIGGER)

- 방아쇠, 제동기, 계기, 유인, 자극이란 사전적 의미
- DBMS에서 특정 사건이 발생 시 자동으로 일련의 과정이 수행되는 프로시저

◆ 프로시저 Vs. 트리거

- 프로시저 : 사용자가 직접 EXEC 명령어를 이용하여 프로시저를 수행함
- 트리거 : 특정 조건을 만족하면 자동으로 수행되도록 하는 저장 프로시저
 - ⇒ 특정 사건이 발생될 때만 실행되는 프로시저
 - ⇒ 사용자가 트리거를 따로 호출할 필요 없음

◆ 무결성과 트리거(TRIGGER)

- 트리거는 데이터의 변경이 발생될 때 수행됨
 - ⇒ 데이터 변경 시 무결성에 문제가 발생되면 이를 보완할 수 있도록 자동으로 프로시저를 수행하도록 트리거를 정의해 놓으면 무결성을 유지시킬 수 있음
- 단점 : 테이블 선언 시 정의한 제약조건에 비하여 성능이 저하됨
- 장점 : 프로시저와 더불어 데이터베이스 내에 업무 규칙을 구현할 수 있음

◆ 수행 기점에 따른 트리거의 분류

- AFTER 트리거
 - 이벤트(삽입 / 삭제 / 변경) 발생 직후 실행되는 트리거
 - 테이블에 대해서만 작성됨
- BEFORE 트리거
 - 이벤트(삽입 / 삭제 / 변경) 발생 이전에 실행되는 트리거
 - 일반적으로 BEFORE 트리거는 지원되지 않음
- INSTEAD OF 트리거
 - 이벤트(삽입 / 삭제 / 변경) 발생 시 해당 이벤트 대신 구동되는 트리거
 - ⇒ 다른 작업을 수행하는 트리거
 - INSTEAD OF 트리거를 활용하여 BEFORE 트리거 같은 역할을 수행시킬 수 있음

● 트리거 활용

1. 트리거의 개념

◆ inserted와 deleted 테이블

- 트리거 작동 시 생성되는 임시 테이블
 - 사건에 따라서 둘 중 하나 또는 둘 다 만들어짐

사건	inserted 테이블	deleted 테이블
삽입	방금 삽입된 튜플이 복사됨	-
변경	변경된 튜플이 복사됨	변경 전 튜플을 보관함
삭제	-	방금 삭제된 튜플을 보관함

● 트리거 활용

2. 트리거의 구동

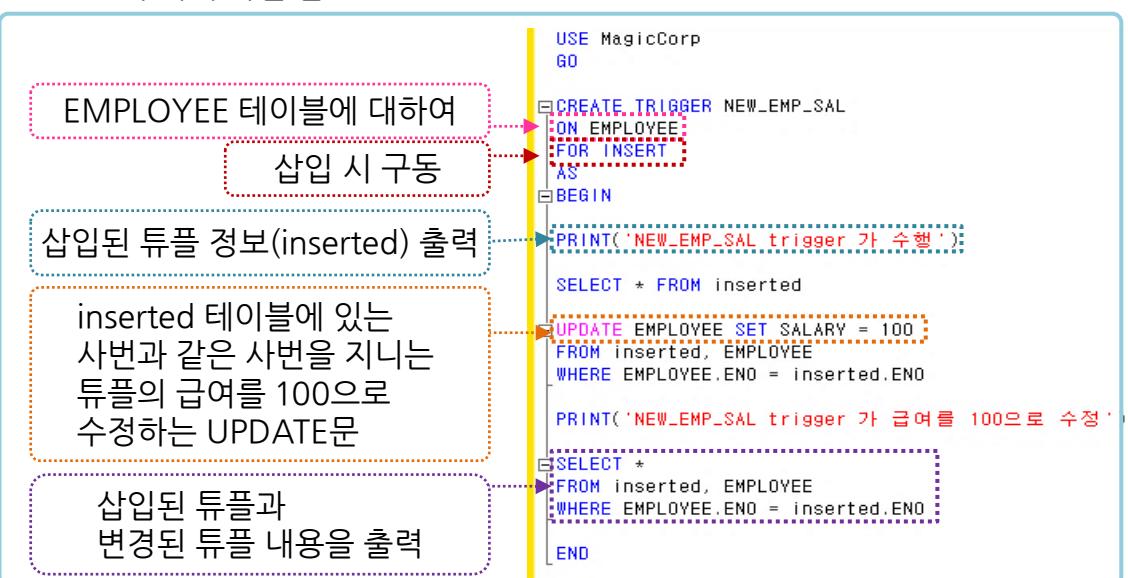
◆ 트리거 생성 문법

```
CREATE TRIGGER 트리거명  
ON 테이블명  
[for / after / instead of] [insert / update / delete]  
AS  
SQL문
```

- for는 after와 같은 것임
- ON 테이블에 의해 테이블에 내용이 추가 / 삭제되면 inserted 또는 deleted라는 가상 테이블에 자동으로 추가되고 이를 이용하여 트리거를 수행시키게 됨

◆ AFTER 트리거(또는 FOR 트리거) 구동 예

- 새로운 업무 규칙
 - 신입 직원들의 급여는 무조건 100임
 - 기존 직원들을 이 규칙에 적용 받지 않음
 - Employee 테이블에 새로운 튜플이 들어 올 때마다 salary 속성을 자동으로 100이 되게 하면 됨



● 트리거 활용

2. 트리거의 구동

◆ AFTER 트리거(또는 FOR 트리거) 구동 예

Q 사원 테이블이 300번인 사원 삽입하기

```
USE MagicCorp
GO

insert into EMPLOYEE(ENO, ENAME) values (300, 'new_emp')

100 % < 결과 메시지


|   | ENO | ENAME   | JOB  | MANAGER | HIREDATE | SALARY | COMMISSION | DNO  |
|---|-----|---------|------|---------|----------|--------|------------|------|
| 1 | 300 | new_emp | NULL | NULL    | NULL     | NULL   | NULL       | NULL |



|   | ENO | ENAME   | JOB  | MANAGER | HIREDATE | SALARY | COMMISSION | DNO  | ENO | ENAME   | JOB  | MANAGER | HIREDATE | SALARY | COMMISS |
|---|-----|---------|------|---------|----------|--------|------------|------|-----|---------|------|---------|----------|--------|---------|
| 1 | 300 | new_emp | NULL | NULL    | NULL     | NULL   | NULL       | NULL | 300 | new_emp | NULL | NULL    | NULL     | 100    | NULL    |



급여가 100으로 설정됨



결과 메시지



NEW_EMP_SAL trigger 가 수행



(1개 행이 영향을 받음)



(1개 행이 영향을 받음)  
NEW_EMP_SAL trigger 가 급여를 100으로 수정



(1개 행이 영향을 받음)



(1개 행이 영향을 받음)


```

● 트리거 활용

2. 트리거의 구동

◆ INSTEAD OF 트리거 구동 예

● INSTEAD OF 트리거

- 뷰나 테이블에 삽입 / 삭제 / 변경 연산에 대응되어 다른 작업을 수행하는 트리거
- ① DEPARTMENT 테이블에 갱신이 일어나면 갱신이 불가능하다고 메시지 출력하기
- 갱신은 일어나지 않게 하기

```
USE MagicCorp
GO

CREATE TRIGGER NO_UPDATE
ON DEPARTMENT
INSTEAD OF UPDATE
AS
BEGIN
PRINT( 'DEPARTMENT에 대한 UPDATE는 하지 마시오' )
END
```

- ② DEPARTMENT의 LOC 속성값을 모두 'SEOUL'로 변경하기

- 트리거로 인하여 안됨

```
USE MagicCorp
GO

SELECT * FROM DEPARTMENT
UPDATE DEPARTMENT
SET LOC = 'SEOUL'
SELECT * FROM DEPARTMENT
```

	DNO	DNAME	LOC
1	10	Accounting	Seoul
2	20	Human	Incheon
3	30	Sales	Yungin
4	40	Computing	Suwon

결과	메시지
	(4개 행이 영향을 받음) DEPARTMENT에 대한 UPDATE는 하지 마시오

(4개 행이 영향을 받음)
DEPARTMENT에 대한 UPDATE는 하지 마시오
(4개 행이 영향을 받음)
(4개 행이 영향을 받음)

● 트리거 활용

3. DDL 트리거

◆ DDL 트리거

- CREATE, ALTER, DROP과 같은 DDL문이 발생시 구동되는 트리거
 - DML 트리거와 유사함
 - 정의 시 ON 테이블명 대신 ON DATABASE를 사용함
 - INSTEAD OF 트리거는 지원 안 함

```
CREATE TRIGGER 트리거명  
ON DATABASE  
{FOR|AFTER} {DROP_TABLE|CREATE_TABLE|ALTER_TABLE}  
AS SQL문
```

- Q 테이블 삭제 시 자동으로 ROLLBACK이 발생되어 테이블이 삭제되지 않도록 하는 DDL 트리거 만들기

```
USE MagicCorp  
GO  
  
CREATE TRIGGER ROLLBACK_TRIGGER  
ON DATABASE  
FOR DROP_TABLE  
AS  
BEGIN  
PRINT('DDL Trigger: ROLLBACK')  
  
ROLLBACK TRANSACTION  
  
END
```

- EMPLOYEE 테이블을 삭제해도 삭제되지 않음

```
USE MagicCorp  
GO  
  
DROP TABLE EMPLOYEE  
GO  
  
SELECT * FROM EMPLOYEE  
GO
```

100 % < >
결과 메시지
DDL Trigger: ROLLBACK
메시지 3609, 수준 16, 상태 2, 줄 4
트리거가 발생하여 트랜잭션이 종료되었습니다. 일괄 처리가 중단되었습니다.
(15개 행이 영향을 받음)

● 트리거 활용

3. DDL 트리거

◆ 트리거의 변경과 삭제

- DCL문임으로 DROP과 ALTER문을 씀

DROP TRIGGER 트리거명

ALTER TRIGGER 트리거명

핵심요약

1. 무결성 규정

■ 무결성의 의미

■ 무결성

- 정밀성, 정확성, 정당성
- 허가 받은 사용자가 수행하는 갱신 작업에서 의미적 오류를 방지함
- 의미적 제약의 개념

■ 무결성 서브 시스템

- 사용자 요청 \Rightarrow 보안 시스템 \Rightarrow 갱신연산 \Rightarrow 무결성 서브시스템 \Rightarrow 갱신 \Rightarrow 정확한 DB
- 무결성 규정을 유지 관리함
- 데이터베이스의 무결성을 유지함
- 트랜잭션이 수행하는 갱신 연산이 무결성 규정을 위반하지 않는가를 감시함
 - ▶ 위반 시에는 거부, 보고, 취소 / 복귀를 수행함

핵심요약

1. 무결성 규정

■ 제약 조건

■ 무결성 규정 대상

- 도메인 : 형식, 타입, 범위
- 기본키, 외래키 : 개체 무결성(Entity Integrity), 참조 무결성(Referencial Integrity)
- 종속성 (묵시적 제약조건) : 함수 종속, 다치 종속, 조인 종속
- 관계 : 내부 관계, 외부 관계

■ 도메인 무결성 대상

- 도메인 정의 : 도메인 이름, 데이터 형
- 삽입이나 갱신 연산에 적용

■ 릴레이션의 무결성 규정

- 릴레이션을 조작하는 과정에서의 의미적 제약조건을 명세함
- 연산 수행 전 / 후에 대한 제약 조건을 규정함
 - ▶ 삽입
 - ▶ 삭제
 - ▶ 갱신
- 분류
 - ▶ 상태 제약과 과도 제약
 - ▶ 집합 제약과 튜플 제약
 - ▶ 즉시 제약과 지연 제약

핵심요약

1. 무결성 규정

■ 제약 조건

■ 상태 제약

- 릴레이션 상태에 대한 제약
- 일관성 있는 상태 유지 : 정적 제약

- ▶ 각 릴레이션 상태가 모두 만족해야 하는 규정
- ▶ 데이터베이스 상태의 유효성

- 키 속성의 제약 : 유일성
- NULL 값의 제약 : 이름은 NULL 값일 수 없음
- 관계 제약 : 참조무결성
- 도메인 제약 : 유효한 값
- 의미 무결성 제약

■ 과도 제약

- 동적 제약

- 데이터베이스의 한 상태에서 다른 상태로 변환되는 과정에서 적용되는 규정
- 데이터베이스 상태의 변환 전과 후의 비교

- ▶ 변환 전과 후에 모두 적용됨

예 》 월급은 감소될 수 없음

핵심요약

1. 무결성 규정

■ 제약 조건

■ 집합 제약

- 튜플 집합 전체에 대한 제약

■ 튜플 제약

- 처리되고 있는 튜플에만 적용됨

■ 즉시 제약

- 삽입 / 삭제 / 갱신 연산이 수행될 때마다 적용되는 제약 규정

■ 지연 제약

- 트랜잭션이 완전히 수행된 후에 적용되는 제약 규정

핵심요약

2. 트리거 활용

■ 트리거의 개념

■ DBMS에서 특정 사건이 발생 시 자동으로 일련의 과정이 수행되는 프로시저

- 프로시저

▶ 사용자가 직접 EXEC 명령어를 이용하여 프로시저를 수행함

- 트리거

▶ 특정 조건을 만족하면 자동으로 수행되도록 하는 저장 프로시저

▶ 특정 사건이 발생될 때만 실행되는 프로시저

▶ 사용자가 트리거를 따로 호출할 필요 없음

■ 무결성과 트리거(TRIGGER)

- 트리거는 데이터의 변경이 발생될 때 수행됨

▶ 데이터 변경 시 무결성에 문제가 발생되면 이를 보완할 수 있도록 자동으로 프로시저를 수행하도록 트리거를 정의해 놓으면 무결성을 유지시킬 수 있음

- 단점

▶ 릴레이션 선언 시 정의한 제약조건에 비하여 성능이 저하됨

- 장점

▶ 프로시저와 더불어 데이터베이스 내에 업무 규칙을 구현할 수 있음

핵심요약

2. 트리거 활용

■ 트리거의 개념

■ 수행 기점에 따른 트리거의 분류

- AFTER 트리거

- ▶ 이벤트(삽입 / 삭제 / 변경) 발생 직후 실행되는 트리거
- ▶ 테이블에 대해서만 작성됨

- BEFORE 트리거

- ▶ 이벤트(삽입 / 삭제 / 변경) 발생 이전에 실행되는 트리거
- ▶ 일반적으로 BEFORE 트리거는 지원되지 않음

- INSTEAD OF 트리거

- ▶ 이벤트(삽입 / 삭제 / 변경) 발생 시 해당 이벤트 대신 구동되는 트리거
- ▶ 즉, 다른 작업을 수행하는 트리거
- ▶ INSTEAD OF 트리거를 활용하여 BEFORE 트리거 같은 역할을 수행시킬 수 있음

■ inserted와 deleted 테이블

사건	inserted 테이블	deleted 테이블
삽입	방금 삽입된 튜플이 복사됨	-
변경	변경된 튜플이 복사됨	변경 전 튜플을 보관함
삭제	-	방금 삭제된 튜플을 보관함

핵심요약

2. 트리거 활용

■ 트리거의 구동

■ 트리거 생성 문법

```
CREATE TRIGGER 트리거명  
ON 테이블명  
[for / after / instead of]  
[insert / update / delete]  
AS  
SQL문
```

- for는 after와 같은 것임
- on 테이블에 의해 테이블에 내용이 추가 / 삭제되면 inserted 또는 deleted라는 가상 테이블에 자동으로 추가되고 이를 이용하여 트리거를 수행시키게 됨

■ AFTER 트리거(또는 FOR 트리거) 구동 예

- 새로운 업무 규칙

- ▶ 신입 직원들의 급여는 무조건 100임
- ▶ 기존 직원들을 이 규칙에 적용 받지 않음
- ▶ Employee 테이블에 새로운 튜플이 들어 올 때마다 salary속성을 자동으로 100이 되게 하면 됨

- INSTEAD OF 트리거

- ▶ 뷰나 테이블에 삽입 / 삭제 / 변경 연산에 대응되어 다른 작업을 수행하는 트리거

핵심요약

2. 트리거 활용

■ 트리거의 구동

■ DDL 트리거

- CREATE, ALTER, DROP과 같은 DDL문이 발생시 구동되는 트리거
- 정의 시 ON 테이블명 대신 ON DATABASE를 사용함
- INSTEAD OF 트리거는 지원 안 함

```
CREATE TRIGGER 트리거명  
ON DATABASE  
{FOR|AFTER} {DROP_TABLE|CREATE_TABLE|ALTER_TABLE}  
AS SQL문
```

- 트리거의 변경과 삭제

▶ DCL문임으로 DROP과 ALTER문을 씀

```
DROP TRIGGER 트리거명
```

```
ALTER TRIGGER 트리거명
```