



# SQL 활용

## 인덱스와 뷰



한국기술교육대학교  
온라인평생교육원

## 학습내용

- 인덱스
- 뷰

## 학습목표

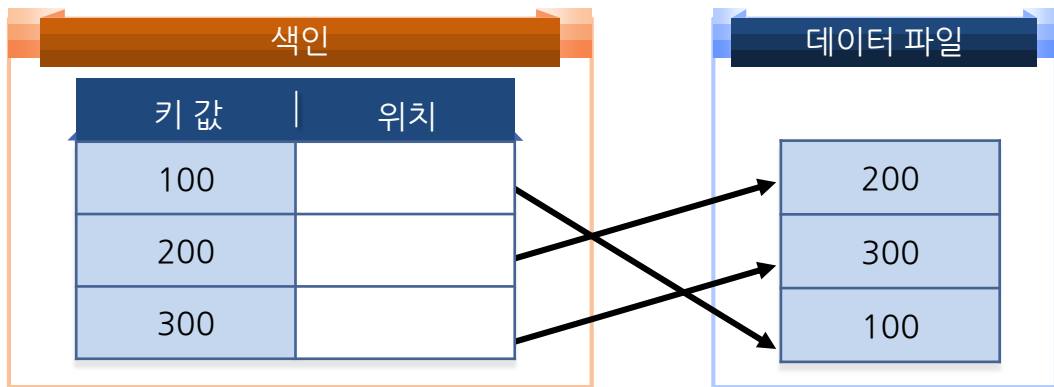
- 데이터에 빠르게 접근하기 위해 다양한 종류의 인덱스를 생성하고 활용할 수 있다.
- 뷰의 개념을 이해하고 뷰를 생성할 수 있다.

## ● 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

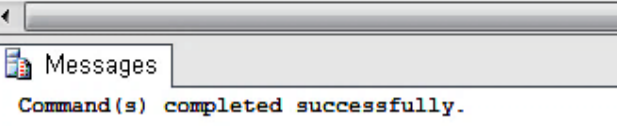
#### ◆ 고유 인덱스 vs 비고유 인덱스

- 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)
```



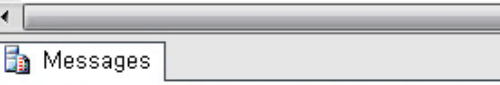
Messages  
Command(s) completed successfully.

- 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

**Q** 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

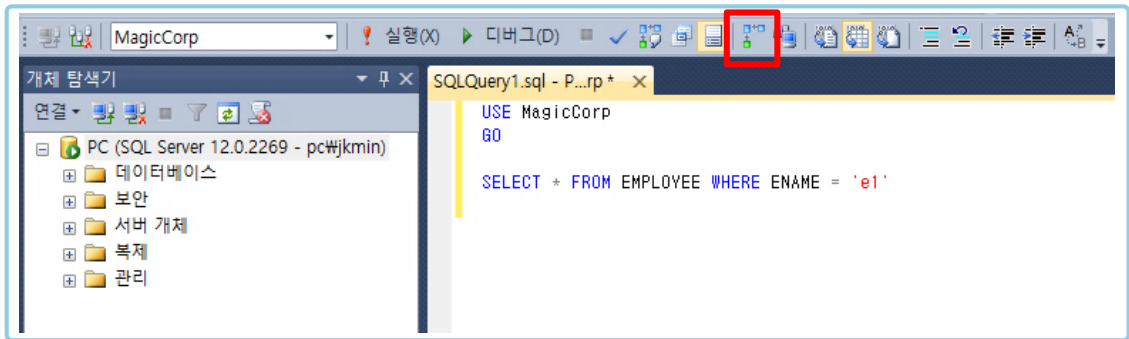
## ● 인덱스

### 3. 인덱스의 활용

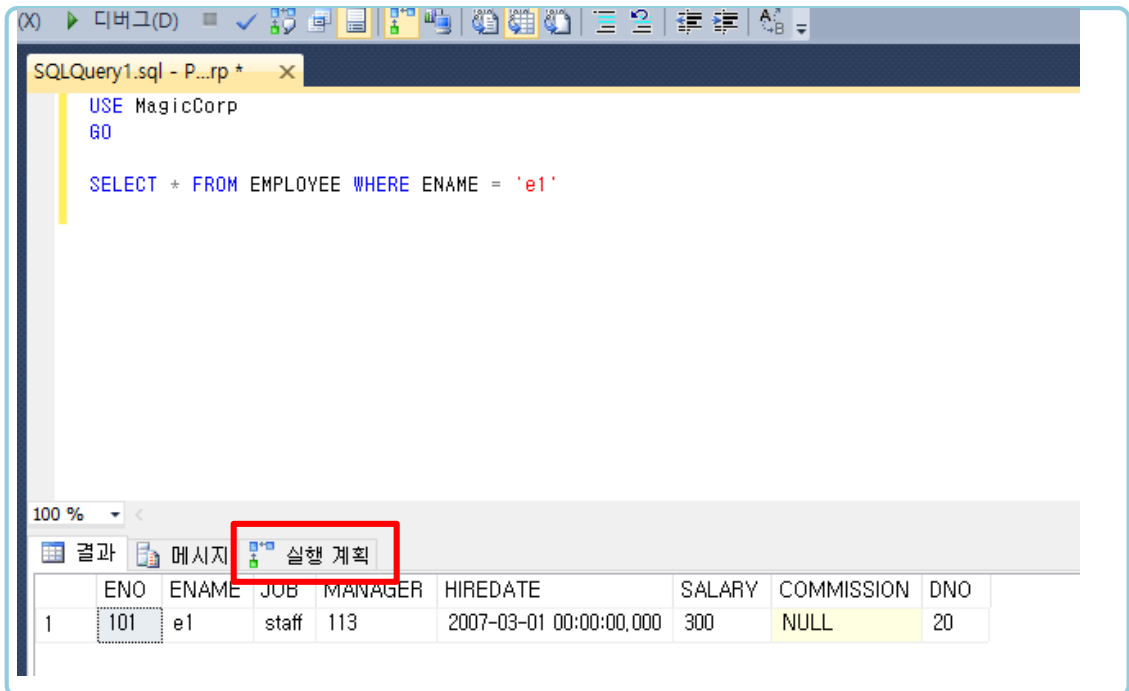
#### ◆ 질의 수행 시 인덱스를 사용하는지 확인하기

##### ● MS-SQL에서 질의 수행 계획 보는 방법

##### ① 질의 수행 전 클릭해서 수행 계획 보기 선택



##### ② 실행 계획 탭이 생김

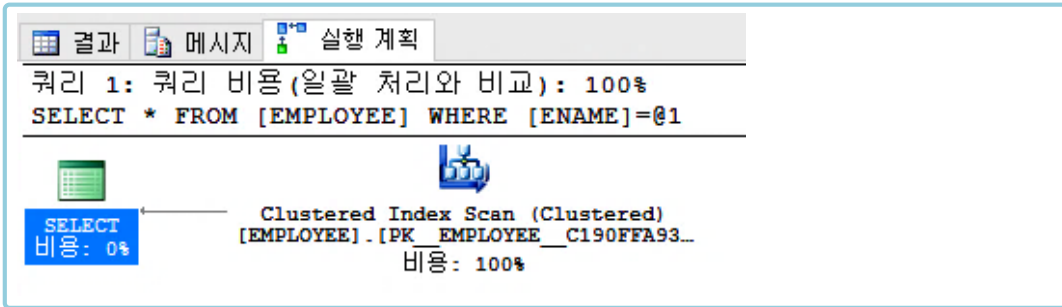


## ● 인덱스

### 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 만들

⇒ 질의 수행기가 해당 색인을 사용하지 않음

강제로 emp\_name\_idx를 사용하게 할 수 있을까?

```
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

```
USE MagicCorp
GO

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

100 % <

결과 메시지 실행 계획

	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 디셔너리(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 직원들을 따로 관리하고 싶은 경우

⇒ 판매부에 속한 직원들만을 직원테이블에서 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 뷰를 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 갱신 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
```

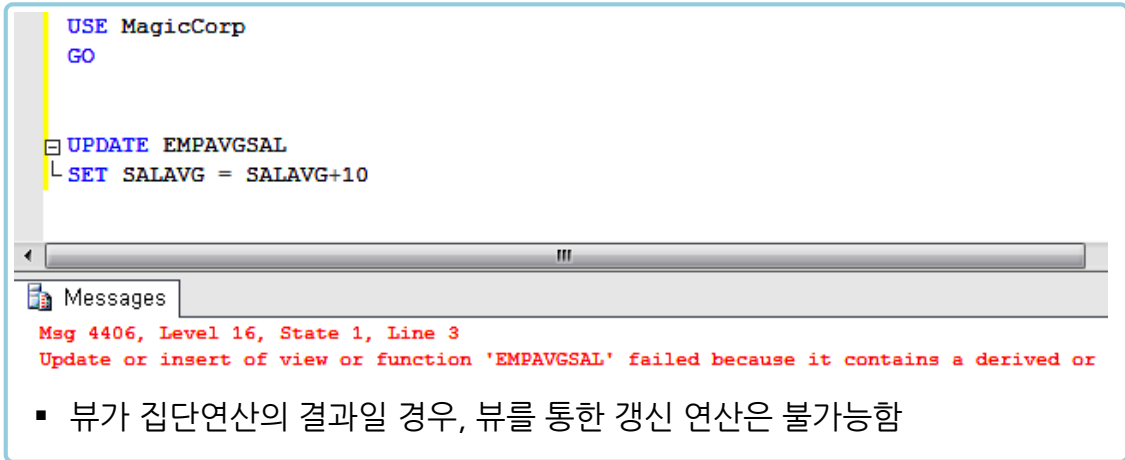
Messages  
Command(s) completed successfully.

## ● 뷰

### 1. 뷰의 개념

#### ◆ 뷰에 대한 갱신 연산

Q 평균 연봉 뷰에 대하여 평균 연봉 10 증가시키기



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

```
USE MagicCorp
GO

UPDATE EMPAVGSAL
SET SALAVG = SALAVG+10
```

Below the query, a scroll bar is visible. At the bottom, a 'Messages' pane displays the following error message:

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

- 뷰가 집단연산의 결과일 경우, 뷰를 통한 갱신 연산은 불가능함



## ● 뷰

### 2. 인라인 뷰

#### ◆ 인라인 뷰란?

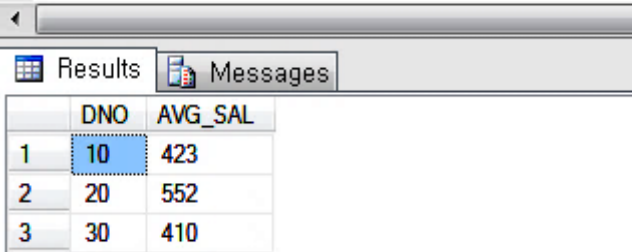
- 하나의 질의문 내에서만 생성되어 사용 되어지고 질의문 수행 종료 후에는 사라지는 뷰
  - 뷰의 명시적인 선언(즉, Create View 문)이 없음
  - FROM 절에서 참조하는 테이블의 크기가 클 경우, 필요한 행과 속성만으로 구성된 집합으로 정의하여 질의문을 효율적으로 구성함
  - FROM 절에서 서브 쿼리를 사용하여 생성하는 임시 뷰

#### ◆ 인라인 뷰의 예제

##### Q 부서별 평균 급여 파악하기

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

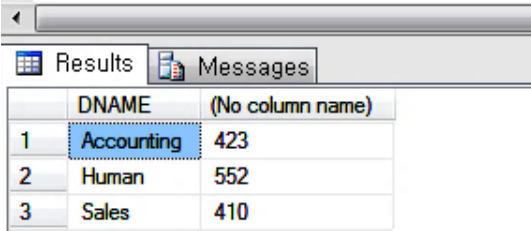
```
SELECT DNO, AVG(salary) as AVG_SAL
FROM EMPLOYEE
GROUP BY DNO
```



The screenshot shows a SQL query window with the query: `SELECT DNO, AVG(salary) as AVG_SAL FROM EMPLOYEE GROUP BY DNO`. Below the query, there is a 'Results' tab showing a table with two columns: 'DNO' and 'AVG\_SAL'. The table contains three rows of data.

	DNO	AVG_SAL
1	10	423
2	20	552
3	30	410

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



The screenshot shows a SQL query window with the query: `SELECT DNAME, AVG(SALARY) FROM DEPARTMENT D, EMPLOYEE E WHERE D.DNO = E.DNO GROUP BY DNAME`. Below the query, there is a 'Results' tab showing a table with two columns: 'DNAME' and '(No column name)'. The table contains three rows of data.

	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
```

Results		Messages
	DNAME	AVG_SAL
1	Accounting	423
2	Human	552
3	Sales	410

## ● 뷰

### 2. 인라인 뷰

#### ◆ WITH 절

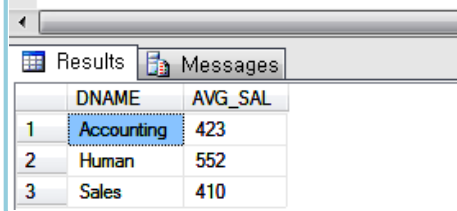
- 인라인 뷰의 또 다른 정의 방법
  - FROM 절에 임시 질의 결과를 정의하는 대신 WITH 절을 이용하여 임시 테이블을 생성함

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 a SQL query window with the following SQL code:

```
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
```

Below the query window, there is a 'Results' tab showing the output of the query:

	DNAME	AVG_SAL
1	Accounting	423
2	Human	552
3	Sales	410

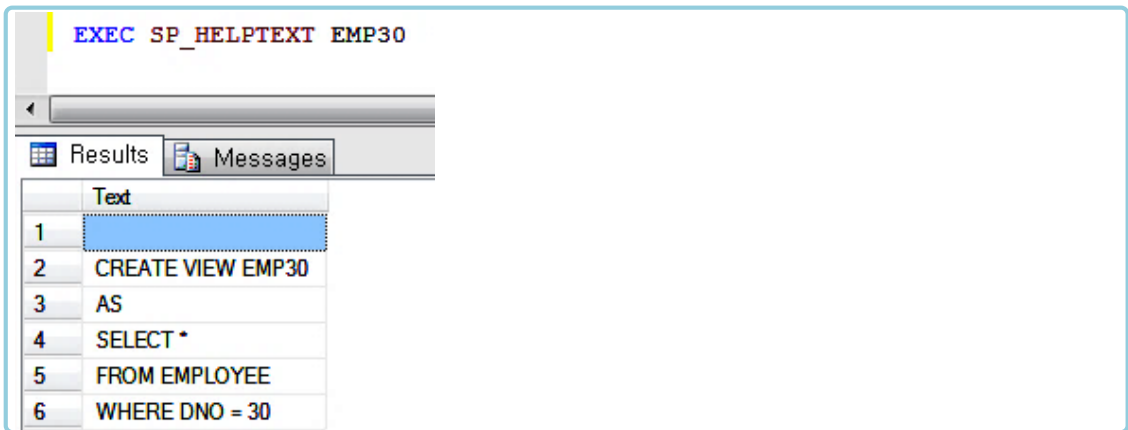
## ● 뷰

### 2. 인라인 뷰

#### ◆ 뷰의 정의 보기

- 뷰의 정의 내용을 보고 싶을 경우
  - SP\_HELPTEXT라는 저장 프로시저를 이용함
- 저장 프로시저를 수행하는 명령문
  - EXEC

#### Q EMP30 뷰의 정의 파악하기



The screenshot shows a SQL Server query window with the command `EXEC SP_HELPTEXT EMP30` entered. Below the command, the 'Results' tab is active, displaying the definition of the EMP30 view. The definition is as follows:

	Text
1	
2	CREATE VIEW EMP30
3	AS
4	SELECT *
5	FROM EMPLOYEE
6	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 WHERE절이나 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)
```

#### ■ 뷰의 삭제 구문

```
DROP VIEW 뷰이름
```

#### ■ 뷰의 종류

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

#### ■ 뷰에 대한 갱신 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
- FROM절에서 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