데이터베이스

트랜잭션(Transaction)과 뷰(View)

학습목표

- 트랜잭션의 개념을 설명할 수 있으며, 트랜잭션 명령문을 이용하여 트랜잭션을 완료 및 복귀 시킬 수 있다.
- 뷰의 특징 및 제약 사항을 설명할 수 있다.

학습내용

- 트랜잭션(Transaction)
- 뷰(View)

트랜잭션(Transaction)

트랜잭션의 개념

01 트랜잭션(Transaction)이란?

- 논리적인 일의 단위
- 기본설정

> 하나의 SQL은 하나의 트랜잭션임

  - 여러 개의 SQL문들이 합쳐서 하나의 트랜잭션이 될 수도 있음

02 트랜잭션의 활용

- 항공기 예약, 은행, 신용 카드 처리, 대형 할인점 등에서는 대규모 데이터베이스를 수백, 수천 명 이상의 사용자들이 동시에 접근
- 많은 사용자들이 동시에 데이터베이스의 서로 다른 부분 또는 동일한 부분을 접근하면서 데이터베이스를 사용

항공기 예약

은행 업무


신용카드 처리

트랜잭션(Transaction)

트랜잭션의 개념

02 트랜잭션의 활용

동시성 제어

- 동시에 수행되는 트랜잭션들이 데이터베이스에 미치는 영향은 이들을 순차적으로 수행하였을 때 데이터베이스에 미치는 영향과 같도록 보장
- 다수 사용자가 데이터베이스를 동시에 접근하도록 허용하면서 데이터베이스의 일관성 유지
- 여러 사용자나 여러 응용 프로그램들이 동시에 수행되어도 서로 간섭하지 못하도록 보장
- 트랜잭션 단위의 동시성 제어

회복

- 데이터베이스를 갱신하는 도중 시스템 고장 시에도 데이터베이스의 일관성을 유지
- 트랜잭션 단위 회복

트랜잭션(Transaction)

트랜잭션의 개념

03 트랜잭션이 없을 경우

01 은행 계좌의 이자 증가

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

02 항공권, 극장 등의 다양한 예약 시스템

- 만약, 좌석을 선점하고 돈을 내기 전에 시스템이 다운 된다면?
  - 돈은 내지 않았지만 좌석을 잡았기 때문에 해당 좌석은 다시 잡을 수 없어짐

트랜잭션(Transaction)

트랜잭션의 개념

03 트랜잭션이 없을 경우

03 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
```

- 계좌 이체 시 장애 발생

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

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

A통장에서 돈만 빠져 나가고 B통장에 돈이 들어오지 않음

은행이 고객의 돈을 횡령한 것!

트랜잭션(Transaction)

트랜잭션의 개념

03 트랜잭션이 없을 경우

03 은행 계좌 이체

- 횡령을 피하기 위해 먼저 B계좌에 돈을 입금

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

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

장애발생

B통장에는 입금이 되었는데, A통장에는 인출이 안됨

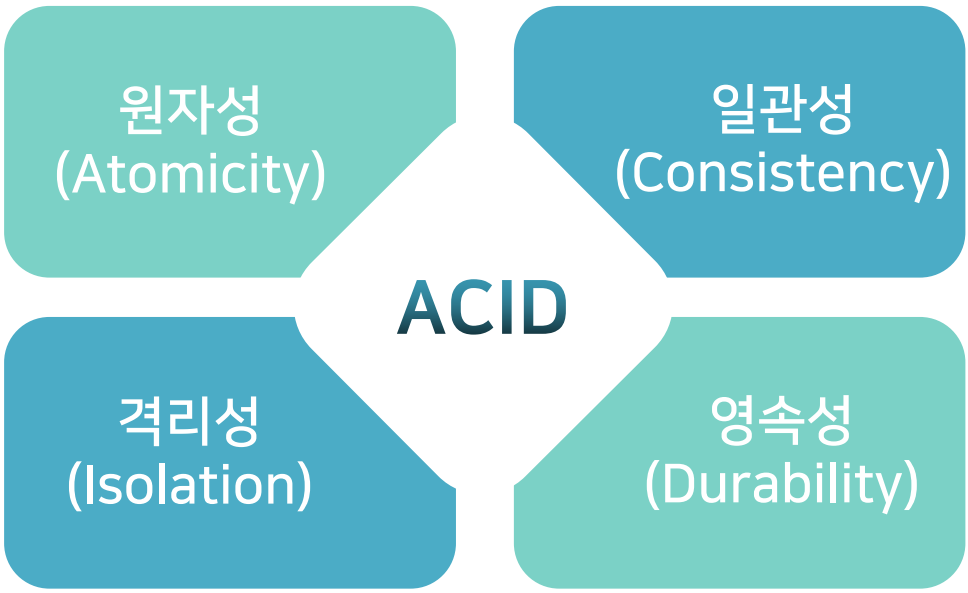
은행의 막대한 손실 발생

두 개의 SQL을 모아서 하나의 트랜잭션(계좌이체 업무)으로 관리하자!
→ 두 DML문은 하나의 업무에 속한 작업들임

트랜잭션(Transaction)

트랜잭션의 개념

04 트랜잭션의 특성



※ ACID : Atomicity, Consistency, Isolation, Durability의 약자

트랜잭션(Transaction)

트랜잭션의 개념

04 트랜잭션의 특성

원자성(Atomicity)

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

일관성(Consistency)

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

트랜잭션(Transaction)

트랜잭션의 개념

04 트랜잭션의 특성

격리성(Isolation)

- 고립성이라고도 하며, 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 보장
- DBMS는 응용들의 요구사항에 따라 다양한 고립 수준(Isolation level)을 제공

지속성(Durability)

- 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 보장

트랜잭션(Transaction)

트랜잭션의 개념

05 ACID와 DB 기능

ACID 특성과 DB의 기능은 모두 다 연관이 있지만, 특별히 더 관련성이 높은 것을 따지면 다음과 같음

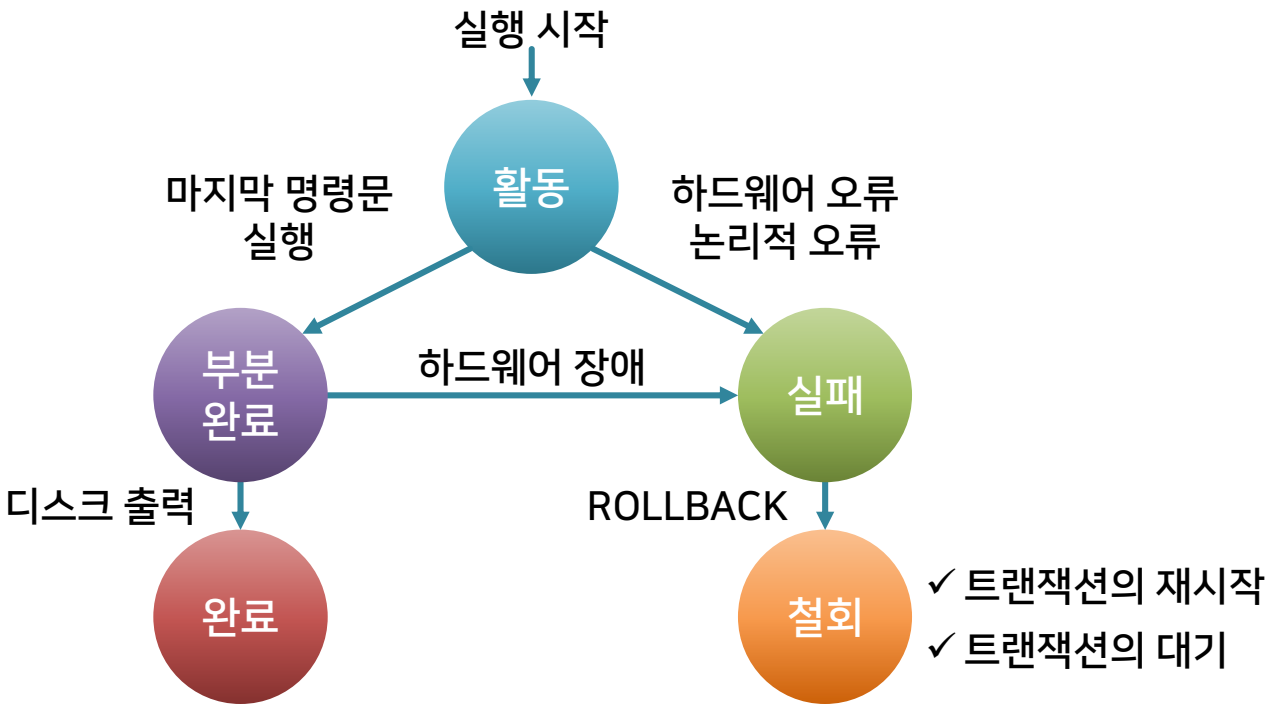
DB 회복 기능	원자성과 지속성에 연관
DB 동시성 제어	일관성과 고립성에 연관
무결성 제약 조건	일관성과 연관

트랜잭션(Transaction)

트랜잭션의 개념

06 트랜잭션의 상태

트랜잭션의 상태 변화



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

트랜잭션(Transaction)

트랜잭션 제어문(TCL)

01 트랜잭션 제어 명령어

COMMIT

01	트랜잭션의 마지막 명령어가 수행되었음을 나타냄
02	트랜잭션에 의한 변경 확정
03	COMMIT 된 트랜잭션은 철회 불가능
04	COMMIT 명령문을 실행하기 전에 하나의 트랜잭션을 변경한 결과를 다른 트랜잭션에서 접근할 수 없도록 방지하여 일관성을 유지

ROLLBACK

01	트랜잭션의 변경을 취소하고 트랜잭션을 종료

SAVEPOINT

01	현재 트랜잭션에서 ROLLBACK 시킬 위치를 지정
04	대규모 트랜잭션(복수개의 명령어들로 이루어진 트랜잭션)에서의 오류 발생이 전체 트랜잭션을 취소 시키는 것이 큰 부담이 될 수 있음

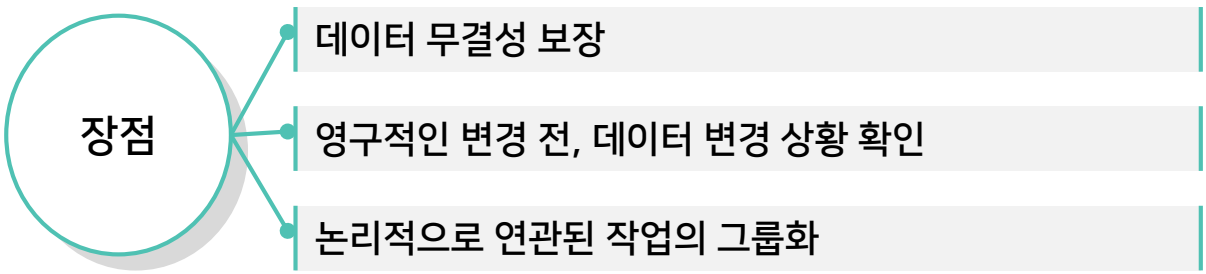
- 실패한 일정 부분만 취소 시키도록 함

트랜잭션(Transaction)

트랜잭션 제어문(TCL)

01 트랜잭션 제어 명령어

COMMIT과 ROLLBACK 명령문의 장점



02 트랜잭션 모드

MS-SQL은 3가지의 트랜잭션 모드를 지원

트랜잭션(Transaction)

트랜잭션 제어문(TCL)

02 트랜잭션 모드

자동 커밋 트랜잭션

- 하나의 명령문이 하나의 트랜잭션이 됨
- MS-SQL에서의 기본 모드

명시적 트랜잭션

- 명시적으로 사용자가 트랜잭션을 정의하는 형태

```
BEGIN TRAN ~ COMMIT TRAN(또는 ROLLBACK TRAN)
```

묵시적 트랜잭션

- 자동 커밋 트랜잭션의 반대되는 개념
- 사용자가 COMMIT TRAN(또는 ROLLBACK TRAN)을 입력하기 전까지 복수개의 명령문을 하나의 트랜잭션으로 간주

▶ BEGIN TRAN이 필요 없음

- 묵시적 트랜잭션의 설정

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

- 트랜잭션의 종료마다 사용자가 반드시 COMMIT/ROLLBAK 명령문을 실행시켜야 함

트랜잭션(Transaction)

트랜잭션 제어문(TCL)

03 TCL 활용

간단한 트랜잭션을 철회하는 방법

예 | 실습을 위하여 DEPARTMENT 테이블 내용을 DEPT01로 복사해봅시다.

```
USE MagicCorp
GO

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

SELECT * INTO DEPT01 FROM DEPARTMENT
SELECT * FROM DEPT01 입력 후 복사

100 %

결과 | 메시지

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

트랜잭션(Transaction)

트랜잭션 제어문(TCL)

03 TCL 활용

간단한 트랜잭션을 철회하는 방법

예 | 트랜잭션을 시작한 후 DEPT01 테이블의 내용을 모두 지우고 트랜잭션 취소 시키기

- 트랜잭션 시작
- DEPT01 테이블 내용 지우기
- DEPT01 내용 보기
- ROLLBACK
- DEPT01 내용 보기

```
BEGIN TRAN
DELETE DEPT01
SELECT * FROM DEPT01
ROLLBACK TRAN
SELECT * FROM DEPT01 입력 후 복사
```

```
USE MagicCorp
GO
BEGIN TRAN
DELETE DEPT01
SELECT * FROM DEPT01
ROLLBACK TRAN
SELECT * FROM DEPT01
```

결과 메시지

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

트랜잭션(Transaction)

트랜잭션 제어문(TCL)

03 TCL 활용

오류발생에 따른 트랜잭션을 철회하는 방법

- 트랜잭션을 구성하는 명령문들 중에서 오류가 발생되면 트랜잭션을 철회하고, 그렇지 않으면 완료하는 것이 필요함
  - 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 방법

```
EXEC sp_help 테이블명
```

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenN
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

트랜잭션(Transaction)

트랜잭션 제어문(TCL)

03 TCL 활용

오류발생에 따른 트랜잭션을 철회하는 방법

- 하나의 트랜잭션을 이용
  - DEPT01 테이블에서 부서번호 10번 튜플 삭제
  - (NULL, 'PRODUCT', 'Seoul') 튜플 삽입
    → 오류 발생
  - 오류가 발생하면 해당 트랜잭션을 ROLLBACK

```
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
```

100 %

결과 메시지

	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(Transaction)

트랜잭션 제어문(TCL)

03 TCL 활용

- SAVEPOINT를 이용한 트랜잭션을 부분 철회하는 방법
  - 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 트랜잭션을 이용
    - DEPT01 테이블에서 부서번호 10번 튜플 삭제
    - 저장점 설정
    - (50, 'PRODUCT', 'Seoul') 추가
    - 저장점으로 rollback
    - (60, DESIGN,'Jeju') 추가

```
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
```

100 %

결과 | 메시지

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

뷰(View)

뷰의 개념

01 뷰(View)란?

- 하나 이상의 기본 테이블이나 다른 뷰를 이용하여 생성되는 가상 테이블
  - 기본 테이블은 디스크에 공간이 할당되어 데이터를 저장
  - 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 적용
- 뷰를 정의한 기본 테이블에서 정의된 무결성 제약조건은 그대로 유지

뷰(View)

뷰의 개념

02 뷰의 필요성

01 사용자마다 특정 객체만 조회할 수 있도록 할 필요가 있음

- 모든 직원에 대한 정보를 모든 사원이 볼 수 있도록 하면 안됨

02 복잡한 질의문을 단순화 할 수 있음

03 데이터의 중복성을 최소화할 수 있음

» 예 | 판매부(Sale)에 속한 사원들을 따로 관리 하고 싶을 때, 판매부에 속한 사원들만을 사원 테이블에서 찾아서 다른 테이블로 만들면 중복성 발생

이러한 상황에 뷰가 필요함

뷰(View)

뷰의 개념

03 뷰의 장·단점

장점

- 논리적 독립성 제공
- 데이터의 접근 제어(보안)
- 사용자의 데이터 관리의 단순화
- 여러 사용자의 다양한 데이터 요구를 지원

단점

- 뷰의 정의 변경 불가
- 삽입, 삭제, 갱신 연산에 제한이 있음

뷰(View)

뷰의 개념

04 뷰의 생성

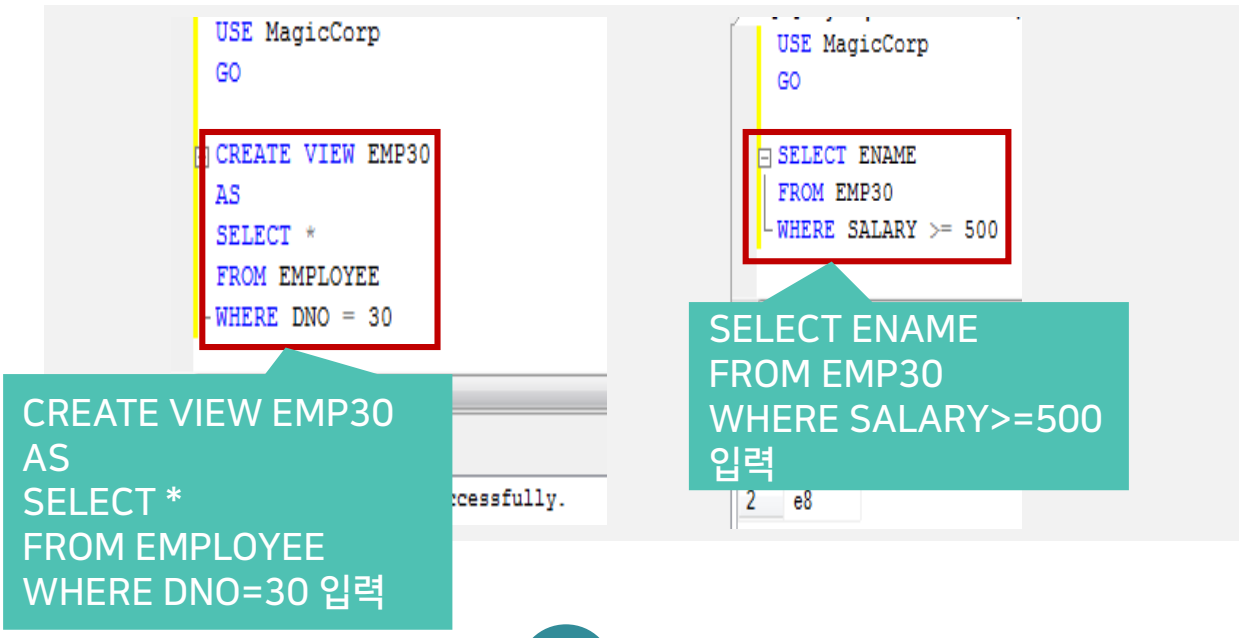
생성 구문

```
CREATE VIEW 뷰이름
AS SQL문(Select문)
```

삭제 구문

```
DROP VIEW 뷰이름
```

예 | 사원 테이블에 부서번호 30인 사원들의 View를 생성하시오.
뷰를 이용하여 부서번호 30인 사원들 중 급여가 500 이상인 사원들의 이름을 구하시오.



뷰(View)

뷰의 개념

05 뷰의 종류

단순뷰	하나의 기본 테이블 위에 정의된 뷰
복합뷰	두 개 이상의 기본 테이블로부터 파생된 뷰

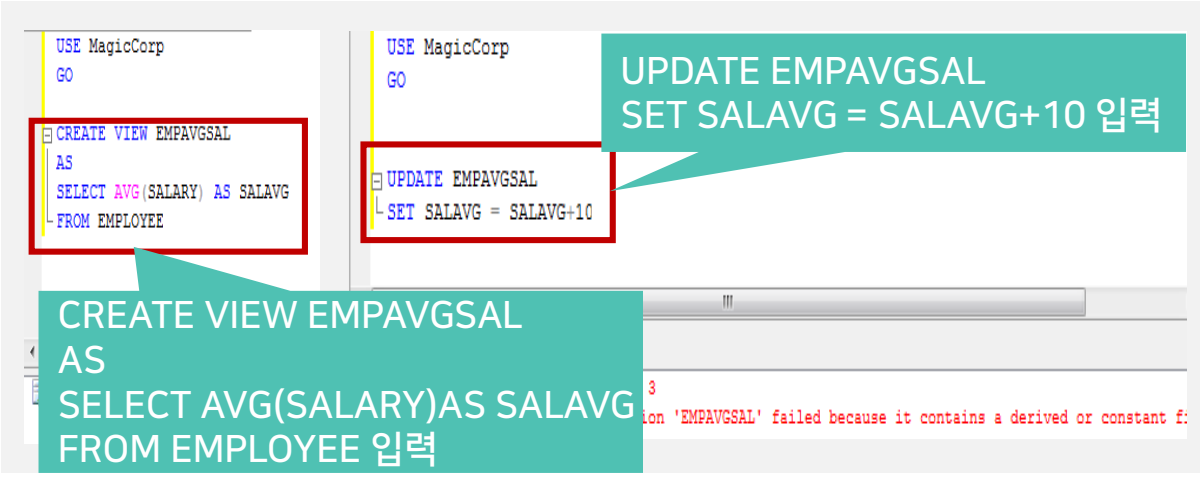
뷰(View)

뷰의 개념

06 뷰에 대한 갱신 연산

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

예 | 사원 테이블에서 평균 연봉을 구하는 View를 생성하자.
평균 연봉 View에 대하여 평균 연봉을 10 증가시키자.



- 갱신 연산이 불가한 환경

View의 결과가 통계 요약을 가지고 있는 경우

통계값이므로 기본 테이블의 어느 튜플에 반영할지 파악하기 어려움

View가 두 개 이상의 테이블로부터(조인 연산) 파생된 경우

여러 개의 테이블로부터 생성된 것이므로
어느 테이블에 속한 튜플을 고쳐야 할지 파악하기 어려움

뷰(View)

인라인뷰

01 인라인뷰(Inline view)

하나의 질의문 내에서만 생성되어 사용되고 질의문 수행 종료 후에는 사라지는 뷰

01 뷰의 명시적인 선언 (즉, Create View문)이 없음

02 FROM절에서 참조하는 테이블의 크기가 클 경우, 필요한 행과 속성만으로 구성된 집합으로 정의하여 질의문을 효율적으로 구성

03 FROM절에서 서브 쿼리를 사용하여 생성하는 임시뷰

뷰(View)

인라인뷰

02 인라인뷰의 예제

예 | 부서별 평균 급여를 파악하자.

부서 번호로 나와 있음 ➡ 부서명도 알고 싶음 ➡ 사원 테이블과 부서 테이블 조인 필요

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

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

	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
```

SELECT DNAME, AVG(SALARY)
FROM DEPARTMENT D, EMPLOYEE E
GROUP BY DNAME 입력

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

뷰(View)

인라인뷰

02 인라인뷰의 예제

예 | 인라인뷰를 이용하여 부서별 부서명, 평균 급여를 출력하자.

- FROM절
  - 인라인뷰 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
```

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 입력

뷰(View)

인라인뷰

03 WITH절

인라인뷰의 또 다른 정의 방법

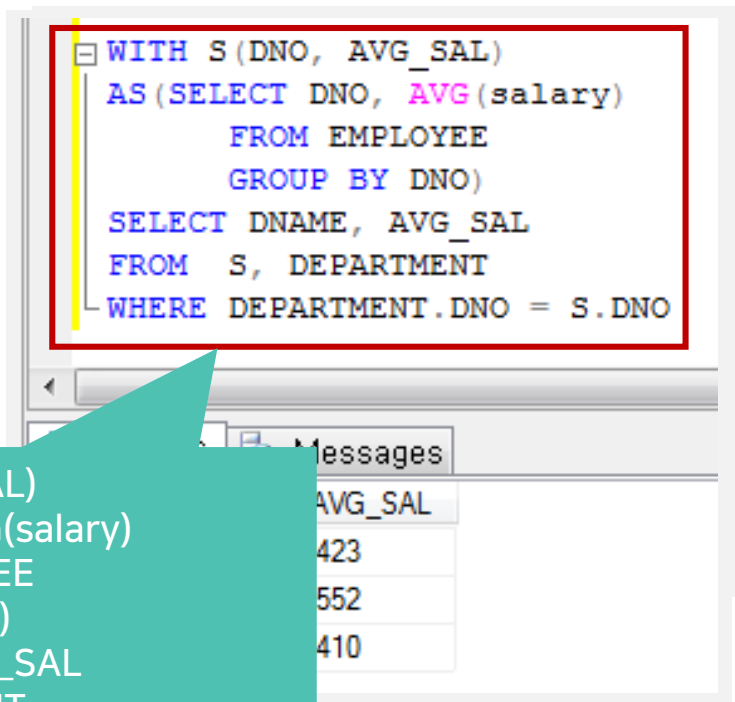
- FROM절에 임시 질의 결과를 정의하는 대신 WITH절을 이용하여 임시 테이블을 생성

```
WITH 임시 테이블명(속성명)
AS (SELECT ~ FROM ~ WHERE)
```

예 | WITH절을 사용하여 부서별 급여평균, 부서명을 출력하자.

- WITH절의 AS문 이후의 질의 결과를 S라는 임시 테이블을 생성
- 메인 질의문에서는 S테이블과 부서 테이블의 조인으로 표현

```
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
```



WHIT S(DNO, AVG_SAL)
AS(SELECT DNO, AVG(salary)
FROM EMPLOYEE
GROUP BY DNO)
SELECT DNAME, AVG_SAL
FROM S, DEPARTMENT
WHERE DPARTMENT.DNO=S.DNO 입력

뷰(View)

인라인뷰

04 뷰의 정의 보기

- 뷰의 정의 내용을 보고 싶을 경우
  - SP_HELPTEXT라는 저장 프로시져를 이용
- 저장 프로시져를 수행하는 명령문: EXEC

예 | EMP30 뷰의 정의를 파악하자.

```
EXEC SP_HELPTEXT EMP30
```

EXEC SP_HELPTEXT EMP30 입력

Results | Messages

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

1 트랜잭션(Transaction)

- ✓ 논리적인 일의 단위로 여러 개의 SQL문들이 합쳐서 하나의 트랜잭션이 될 수도 있음
- ✓ 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

2 뷰(View)

- ✓ 뷰는 하나 이상의 기본 테이블이나 다른 뷰를 이용하여 생성되는 가상 테이블로 데이터 딕셔너리(Data dictionary) 테이블에 뷰에 대한 정의(SQL문)만 저장되어 디스크 저장 공간 할당이 이루어지지 않음

- ✓ 뷰의 필요성
  - 사용자 마다 특정 객체만 조회할 수 있도록 할 필요가 있음
  - 복잡한 질의문을 단순화 할 수 있음
  - 데이터의 중복성을 최소화할 수 있음