

초보자를 위한
Oracle SQL Database



트랜잭션



한국기술교육대학교
온라인평생교육원

학습내용

- 트랜잭션
- COMMIT, ROLLBACK

학습목표

- 트랜잭션에 대하여 설명할 수 있다.
- COMMIT, ROLLBACK을 이해하고 사용할 수 있다.

트랜잭션

트랜잭션

트랜잭션

트랜잭션이란

트랜잭션이란

- DBMS에서 발생하는 1개 이상의 SQL 명령들을 하나의 논리 집합으로 묶어 놓은 단위
- 5개의 SQL 문이 하나의 작업으로 진행이 될 때, 5개의 SQL 문이 정상적으로 완료 되었으면 정상 종료되고, 중간에 하나라도 오류가 발생하면 모든 명령이 취소되는 ALL-OR-NOTHING 방식으로 처리되어야 함



트랜잭션

트랜잭션

트랜잭션의 특징

트랜잭션의 특징 ACID

A 원자성 : ATOMICITY

C 일관성 : CONSISTENCY

I 격리성 : ISOLATION

D 지속성 : DURABILITY



트랜잭션



트랜잭션

트랜잭션

트랜잭션의 특징

트랜잭션의 특징 ACID

A

원자성 : ATOMICITY

- ALL-OR-NOTHING을 보장함
- 부분적으로 실행되는 것 없이, 모두 성공적으로 실행되거나, 모두 실행되지 않은 상태를 유지하는 것을 말함



트랜잭션

트랜잭션

트랜잭션의 특징

트랜잭션의 특징 ACID

C

일관성 : CONSISTENCY

- 성공적으로 트랜잭션이 완료되면 일관적인 데이터베이스 상태를 유지하는 것을 말함
- SELECT하는 시점에 데이터베이스에 변경이 확정된 데이터만 읽어 항상 일관적인 데이터를 조회



트랜잭션

트랜잭션

트랜잭션

트랜잭션의 특징

트랜잭션의 특징 ACID

I 격리성 : ISOLATION

- 트랜잭션이 실행되고 있을 때 다른 트랜잭션의 작업이 끼어들지 못하도록 보장함
- 트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과가 만들어지는 것을 막기 위해 트랜잭션은 격리되어야 함



트랜잭션

트랜잭션

트랜잭션의 특징

트랜잭션의 특징 ACID

D 지속성 : DURABILITY

- 성공적으로 수행된 트랜잭션은 데이터베이스에 영구적으로 반영되어야 함



트랜잭션

트랜잭션 트랜잭션

트랜잭션 예시

트랜잭션의 예: 은행에서 계좌 이체하기



1 B 계좌에 +10,000원 입금요청

2 A 계좌 잔금확인

3 A 계좌 -10,000원

4 B 계좌 상태확인

5 B 계좌에 +10,000원 입금완료

트랜잭션 트랜잭션

트랜잭션 예시

은행을 이용하는 A씨가 자신의 계좌에서 B씨에게 10,000원을 송금하는 과정

1. B계좌에 10,000원을 입금 요청
2. A 계좌 잔금확인
3. A 계좌에서 -10,000원
4. B 계좌 상태 확인
5. B 계좌에 +10,000원
6. B 계좌 10,000원 입금완료

모든 과정이 정상적으로 실행이 되었으며 데이터베이스에 수정된 내용이 반영(ALL) 됨

트랜잭션

트랜잭션 트랜잭션

트랜잭션 예시

트랜잭션 오류발생의 예: 은행에서 계좌 이체하기



트랜잭션 트랜잭션

트랜잭션 예시

은행을 이용하는 A씨가 자신의 계좌에서 B씨에게 10,000원을 송금하는 과정

1. B계좌에 10,000원을 입금 요청
2. A 계좌 잔금확인
3. A 계좌에서 -10,000원
4. B 계좌 상태 확인
5. B 계좌에 +10,000원
6. B 계좌 10,000원 입금완료

과정 중 4번에서 어떤 이유로 오류가 발생하게 되어 A계좌에서는 10,000원이 빠졌는데 B계좌에 입금이 안되면 큰 문제 발생

이 때문에 송금하는 과정의 SQL 명령을 하나의 트랜잭션으로 처리하여 중간에 오류가 발생하게 되면 모든 과정이 취소(NOTHING)가 되어 ALL-OR-NOTHING을 보장해야 함

트랜잭션

트랜잭션

트랜잭션

트랜잭션 관리

트랜잭션 제어어(TCL)를 사용

COMMIT

ROLLBACK

SAVEPOINT

SET TRANSACTION



트랜잭션

트랜잭션

트랜잭션 관리

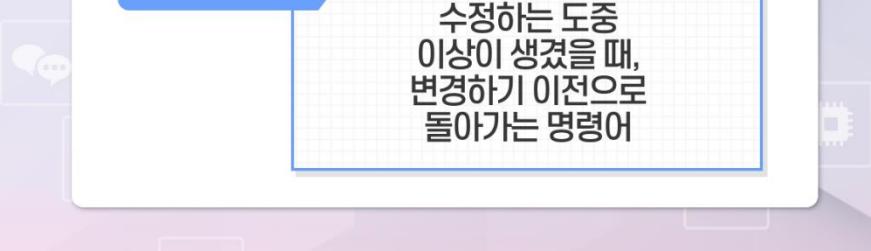
트랜잭션 제어어(TCL)를 사용

COMMIT

한 트랜잭션에서 데이터 조작이 정상적으로 완료되었으면, 그 결과를 데이터베이스에 반영하는 명령어

ROLLBACK

트랜잭션 내에서 데이터를 수정하는 도중 이상이 생겼을 때, 변경하기 이전으로 돌아가는 명령어



트랜잭션

트랜잭션

트랜잭션

트랜잭션 관리

트랜잭션 제어어(TCL)를 사용

SAVEPOINT

트랜잭션의 특정 지점에 이름을 지정하고,
그 지점 이전에 수행한 작업에 영향을 주지 않고
그 지점 이후에 수행한 작업을 롤백(ROLLBACK)
할 수 있음

트랜잭션

트랜잭션

트랜잭션 관리

트랜잭션 제어어(TCL)를 사용

SET TRANSACTION

현재 트랜잭션을 읽기 전용 또는 읽기/쓰기로 설정하거나,
분리 수준을 설정,
롤백 세그먼트를 지정,
트랜잭션에 이름을 지정

- SET TRANSACTION READ WRITE : 트랜잭션을 Read Write로 설정. 기본값.
- SET TRANSACTION READ ONLY : 트랜잭션을 ReadOnly로 설정.
- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE : 트랜잭션이 수행 중일 때 INSERT/DELETE도 제한.

트랜잭션

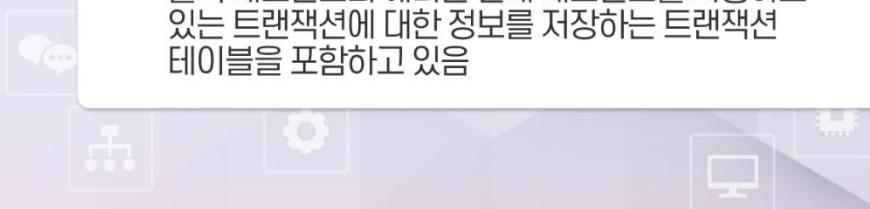


트랜잭션 트랜잭션

ROLLBACK SEGMENT

ROLLBACK SEGMENT

- 롤백 세그먼트란 트랜잭션을 실행할 때 그 이전 이미지를 기록하기 위해 사용되는 원형(CIRCULAR)구조의 세그먼트
- 롤백 세그먼트는 트랜잭션이 데이터베이스의 데이터를 변경할 때 수정되기 전의 파일, 블록 ID 같은 블럭 정보 및 데이터를 저장
- 롤백 세그먼트의 헤더는 현재 세그먼트를 사용하고 있는 트랜잭션에 대한 정보를 저장하는 트랜잭션 테이블을 포함하고 있음



트랜잭션 트랜잭션

트랜잭션의 시작과 종료

새 트랜잭션이 시작하는 경우

- 클라이언트가 접속한 직후
- COMMIT 명령을 실행한 직후
- ROLLBACK 명령을 실행한 직후
- DDL, DCL 관련 명령을 실행한 직후



트랜잭션

트랜잭션

트랜잭션

트랜잭션의 시작과 종료

트랜잭션이 종료하는 경우

- 클라이언트가 접속을 종료한 직후
- COMMIT 명령을 실행한 직후
- ROLLBACK 명령을 실행한 직후
- DDL, DCL 관련 명령을 실행한 직후



트랜잭션

트랜잭션

트랜잭션의 시작과 종료

자동 커밋(AUTO COMMIT)

- DDL, DCL 관련 명령을 실행하면
자동으로 트랜잭션이 종료되는 현상



COMMIT, ROLLBACK

트랜잭션

COMMIT, ROLLBACK 문

COMMIT 문

COMMIT 문

- COMMIT은 하나의 트랜잭션인 여러 DML(INSERT, DELETE 등) 명령어를 정상적으로 데이터베이스에 반영하라는 명령어
- 즉, 하나의 트랜잭션 과정을 정상적으로 종료하고 새로운 트랜잭션을 시작
- 이때 COMMIT 명령을 사용하지 않아도 AUTO COMMIT이 되는 경우가 있음
- DDL(CREATE, ALTER, DROP, RENAME 등) 및 DCL(GRANT, REVOKE) 명령문을 수행하면 AUTO COMMIT이 되고 데이터베이스를 정상적으로 종료하였을 경우에도 AUTO COMMIT이 됨
- 자동으로 적용되는 AUTO COMMIT은 꼭 기억할 것

트랜잭션

COMMIT, ROLLBACK 문

COMMIT 문

트랜잭션 1	트랜잭션 2	트랜잭션 3
INSERT ➔ UPDATE ➔ INSERT	INSERT ➔ UPDATE ➔ DELETE	UPDATE ➔ INSERT ➔ DELETE

트랜잭션 1 시작

COMMIT
트랜잭션 1 종료/DB 반영
트랜잭션 2 시작

COMMIT
트랜잭션 2 종료/DB 반영
트랜잭션 3 시작

COMMIT, ROLLBACK

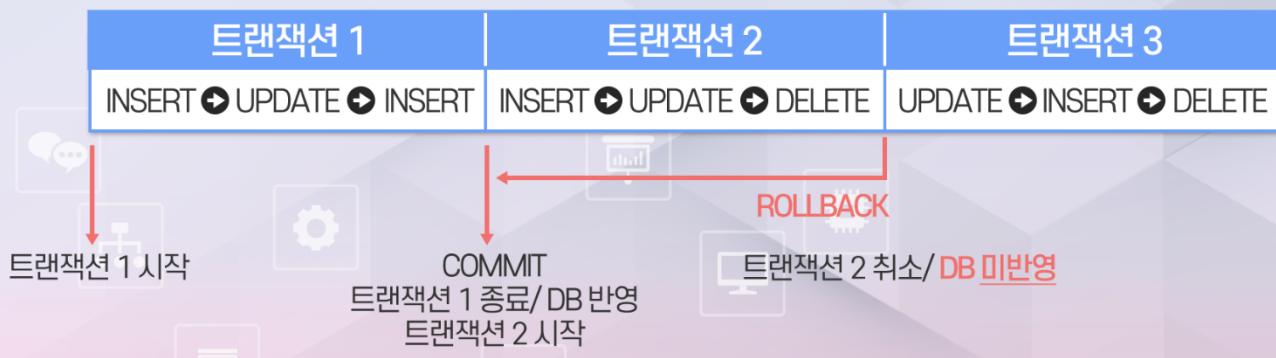
트랜잭션

COMMIT, ROLLBACK 문

ROLLBACK

ROLLBACK

- ROLLBACK은 오류나 어떤 문제가 발생하였을 때 하나의 트랜잭션을 취소하는 명령
- 시스템에 비정상적인 종료가 발생하였다면 자동으로 ROLLBACK이 됨
- COMMIT를 한번 적용하면 COMMIT 이전으로 되돌릴 수 없음



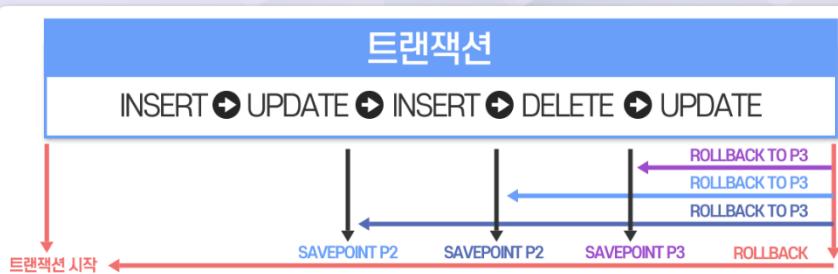
트랜잭션

COMMIT, ROLLBACK 문

SAVEPOINT

SAVEPOINT

- SAVEPOINT는 하나의 트랜잭션을 작게 나누어 저장하는 역할을 함
- “SAVEPOINT SAVEPOINT명”으로 지정하고 저장된 SAVEPOINT에 “ROLLBACK TO SAVEPOINT명”을 이용하여 해당 SAVEPOINT까지 ROLLBACK할 수 있음
- SAVEPOINT는 하나의 트랜잭션 안에 여러 개 지정 가능
- 주의할 점은 P1 지점으로 ROLLBACK을 하였으면 P2 자동 삭제, 전체 ROLLBACK을 하였으면 모든 SAVEPOINT 지점 삭제



COMMIT, ROLLBACK



트랜잭션

COMMIT, ROLLBACK 문

COMMIT, ROLLBACK 실습 예

워크시트 질의 작성기

```

1 CREATE TABLE TEST (NO NUMBER(3), NAME VARCHAR2(10));
2 INSERT INTO TEST VALUES(1, 'KIM');
3 INSERT INTO TEST VALUES(2, 'LEE');
4 INSERT INTO TEST VALUES(3, 'PARK');
5 SELECT * FROM TEST;

```

1: Table TEST이(가) 생성되었습니다.

2: 1 행 이(가) 삽입되었습니다.

3: 1 행 이(가) 삽입되었습니다.

4: 1 행 이(가) 삽입되었습니다.

5:

NO NAME

1 KIM

2 LEE

3 PARK

트랜잭션

COMMIT, ROLLBACK 문

COMMIT, ROLLBACK 실습 예

```

5 SELECT * FROM TEST;
6 ROLLBACK;
7 SELECT * FROM TEST;
8 INSERT INTO TEST VALUES(4, 'CHOI');
9 INSERT INTO TEST VALUES(5, 'JEONG');
10 SELECT * FROM TEST;
11 COMMIT;

```

NO NAME

1 KIM

2 LEE

3 PARK

6: 롤백 완료.

7: 선택된 행 없음

8: 1 행 이(가) 삽입되었습니다.

9: 1 행 이(가) 삽입되었습니다.

10:

NO NAME

4 CHOI

5 JEONG

11: 커밋 완료.

12:

NO NAME

COMMIT, ROLLBACK

트랜잭션

COMMIT, ROLLBACK 문

COMMIT, ROLLBACK 실습 예

```

11: COMMIT;
12: SELECT * FROM TEST;
13: INSERT INTO TEST VALUES(6, 'KANG');
14: SELECT * FROM TEST;
15: ROLLBACK;

```

```

11: 커밋 완료.
12:
   NO NAME
   -----
   4 CHOI
   5 JEONG
13: 1 행 이(가) 삽입되었습니다.
14:
   NO NAME
   -----
   4 CHOI
   5 JEONG
   6 KANG
15: 롤백 완료.

```

트랜잭션

COMMIT, ROLLBACK 문

COMMIT, ROLLBACK 실습 예

```

15: ROLLBACK;
16: SELECT * FROM TEST;
17: DROP TABLE TEST;

```

```

15: 롤백 완료.
16:
   NO NAME
   -----
   4 CHOI
   5 JEONG
17: Table TEST01(가) 삭제되었습니다.

```

COMMIT, ROLLBACK



트랜잭션

COMMIT, ROLLBACK 문

◆ SAVEPOINT 실습 예

```
워크시트 1. 질의 작성기
1 CREATE TABLE TEST(NO NUMBER(3),NAME VARCHAR2(10));
2 INSERT INTO TEST VALUES(1,'KIM');
3 INSERT INTO TEST VALUES(2,'LEE');
4 INSERT INTO TEST VALUES(3,'PARK');
5 SELECT * FROM TEST;
6 SAVEPOINT P1;
```

1: Table TEST이 (가) 생성되었습니다.
 2: 1 행 이 (가) 삽입되었습니다.
 3: 1 행 이 (가) 삽입되었습니다.
 4: 1 행 이 (가) 삽입되었습니다.
 5:
 NO NAME

 1 KIM
 2 LEE
 3 PARK
 6: Savepoint0이 (가) 생성되었습니다.



트랜잭션

COMMIT, ROLLBACK 문

◆ SAVEPOINT 실습 예

```
6 SAVEPOINT P1;
7 INSERT INTO TEST VALUES(4,'CHOI');
8 SELECT * FROM TEST;
9 ROLLBACK TO P1;
10 SELECT * FROM TEST;
```

6: Savepoint0이 (가) 생성되었습니다.
 7: 1 행 이 (가) 삽입되었습니다.
 8:
 NO NAME

 1 KIM
 2 LEE
 3 PARK
 4 CHOI
 9: 둘백 완료.
 10:
 NO NAME

 1 KIM
 2 LEE
 3 PARK



COMMIT, ROLLBACK

트랜잭션

COMMIT, ROLLBACK 문

SAVEPOINT 실습 예

```

10: SELECT * FROM TEST;
11: INSERT INTO TEST VALUES(5, 'JEONG');
12: INSERT INTO TEST VALUES(6, 'KANG');
13: SELECT * FROM TEST;
14: SAVEPOINT P2;
```

```

10:
      NO NAME
-----
      1 KIM
      2 LEE
      3 PARK
11: 1 행이(가) 삽입되었습니다.
12: 1 행이(가) 삽입되었습니다.
13:
      NO NAME
-----
      1 KIM
      2 LEE
      3 PARK
      5 JEONG
      6 KANG
14: Savepoint이(가) 생성되었습니다.
```

트랜잭션

COMMIT, ROLLBACK 문

SAVEPOINT 실습 예

```

14: SAVEPOINT P2;
15: ROLLBACK TO P1;
16: SELECT * FROM TEST;
17: ROLLBACK TO P2;
```

```

14: Savepoint이(가) 생성되었습니다.
15: 롤백 완료.
16:
      NO NAME
-----
      1 KIM
      2 LEE
      3 PARK
17:
명령의 17 행에서 시작하는 중 오류 발생 -
ROLLBACK TO P2
오류 보고 -
ORA-01086: 'P2' 저장점이 이 세션에 설정되지 않았거나 부적합합니다.
01086. 00000 - "savepoint '%s' never established in this session or is invalid"
*Cause: An attempt was made to roll back to a savepoint that was never
        established in this session, or was invalid.
*Action: Try rolling back to the savepoint from the session where it is estab
```

COMMIT, ROLLBACK



트랜잭션

COMMIT, ROLLBACK 문

◆ SAVEPOINT 실습 예

```
17: ROLLBACK TO P2;
18: SELECT * FROM TEST;
19: DROP TABLE TEST;
```

17:
명령의 17 행에서 시작하는 중 오류 발생 -
ROLLBACK TO P2
오류 보고 -
ORA-01086: 'P2' 저장점이 이 세션에 설정되지 않았거나 부적합합니다.
01086. 00000 - "savepoint '%s' never established in this session or is invalid'
*Cause: An attempt was made to roll back to a savepoint that was never
established in this session, or was invalid.
*Action: Try rolling back to the savepoint from the session where it is estab
18:
NO NAME

1 KIM
2 LEE
3 PARK
19: Table TEST01(가) 삭제되었습니다.



실습하기

트랜잭션

실습하기

- 트랜잭션을 테스트할 테이블을 생성하고 데이터를 입력하기 바랍니다.

```
CREATE TABLE "TEST_TRANSACTION"
(
    "NO" NUMBER(3,0),
    "NAME" VARCHAR2(10),
    "HEIGHT" NUMBER(3,0),
    "WEIGHT" NUMBER(3,0)
);
```

```
INSERT INTO TEST_TRANSACTION (NO,NAME,HEIGHT,WEIGHT) VALUES (1,'KIM',180,95);
INSERT INTO TEST_TRANSACTION (NO,NAME,HEIGHT,WEIGHT) VALUES (2,'LEE',175,88);
INSERT INTO TEST_TRANSACTION (NO,NAME,HEIGHT,WEIGHT) VALUES (3,'PARK',177,85);
INSERT INTO TEST_TRANSACTION (NO,NAME,HEIGHT,WEIGHT) VALUES (4,'JEONG',170,76);
INSERT INTO TEST_TRANSACTION (NO,NAME,HEIGHT,WEIGHT) VALUES (5,'CHOI',163,67);
INSERT IN
```

- + Oracle Database 19c 버전 사용
- + 최신 버전인 Oracle Database 21c는 현재 cloud에서만 사용 가능하며, 지원 기간은 2년

트랜잭션 적용하기

실습단계

트랜잭션 적용하기

트랜잭션 테스트 테이블 생성하기

번호를 저장할 3자리 숫자 컬럼

이름을 저장할 10자리 문자 컬럼

키를 저장할 3자리 숫자 컬럼

몸무게를 저장할 3자리 숫자 컬럼

초기 입력 데이터들

SET TRANSACTION READ ONLY;

트랜잭션을 ReadOnly로 설정

SET TRANSACTION READ WRITE;

트랜잭션을 Read Write로 설정. 기본값.

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

각 SQL문장의 실행 시점의 스냅샷을 사용

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;



실습하기

실습단계

트랜잭션 내의 모든 문장을 완료할 때까지 트랜잭션 시작 시점의 데이터 값을 사용

테이블 생성 SQL 문

명령문 실행 아이콘 또는 키보드에서 CTRL+ENTER키로 실행

TEST_TRANSACTION 테이블 생성

CTRL+ENTER키로 데이터 입력 실행

입력 데이터 확인

COMMIT으로 데이터베이스에 저장

READ ONLY TRANSACTION 시작

데이터 입력 테스트

읽기 전용 트랜잭션이라 입력 오류 발생

데이터 수정 테스트

역시, 읽기 전용 트랜잭션이라 입력 오류 발생

ROLLBACK 수행

READ ONLY TRANSACTION 종료

READ WRITE TRANSACTION 시작

데이터 입력 테스트

데이터 입력 성공 확인

테이블에 있는 모든 데이터 삭제 테스트

데이터 삭제 확인

ROLLBACK 명령으로 현 트랜잭션 작업 취소

ROLLBACK 되었는지 확인

COMMIT으로 트랜잭션 종료하고 데이터베이스에 저장

Oracle 서버 가상 머신에서 작업

SQL 명령으로 오라클에 접속

ISOLATION LEVEL READ COMMITTED TRANSACTION 시작

트랜잭션은 새로운 접속 시 자동으로 시작

서버에서 데이터 수정 작업을 실행

서버에서 COMMIT 실행하여 데이터베이스에 저장

원격지 접속에서 수정된 값이 적용되었는지 확인하면 수정되어 있음

테이블에 있는 모든 데이터 삭제 테스트



실습하기

실습단계

데이터 삭제 확인

ROLLBACK 명령으로 현 트랜잭션 작업 취소

ROLLBACK 되었는지 확인

COMMIT으로 트랜잭션 종료하고 데이터베이스에 저장

ISOLATION LEVEL SERIALIZABLE TRANSACTION 시작

데이터 입력 테스트

데이터 입력 성공 확인

서버에서 데이터 수정 작업을 실행

서버에서 데이터 수정 성공 확인

서버에서 COMMIT 실행하여 데이터베이스에 저장

원격지 접속에서 수정된 값이 적용되었는지 확인하면 수정 안 되어 있음

즉, 트랜잭션 시작 시점의 데이터 값을 유지

ROLLBACK 명령으로 현 트랜잭션 작업 취소

ROLLBACK 되었는지 확인

COMMIT으로 트랜잭션 종료하고 데이터베이스에 저장