DataBase & Oracle DBMS

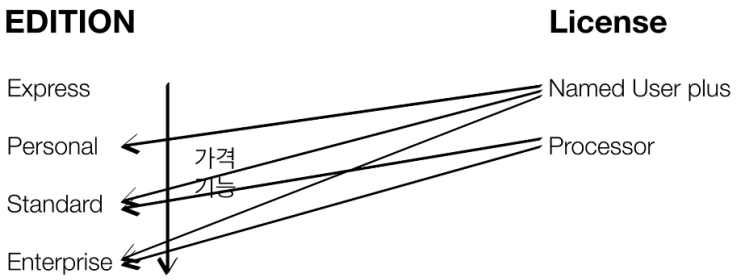
Oracle - 1. 수업 소개

Graph형, Document형, Object형(객체형), Hierarchical형(계층형), Key-Value형, Relational형(관계형) 등 많은 DBMS가 존재하는데 압도적으로 많이 사용되는 DBMS는 Relational형(관계형)이다.

Relational형(관계형) DBMS의 핵심은 표이다. 인류가 만든 수많은 정보 시각화 도구 중 가장 위대한 성취 2가지를 뽑아보자면 좌표 평면과 표이다. 이 복잡한 세상에서 어떤 정보든 간에 x, y 혹은 열과 행의 틀에 따라서 정보를 정리 정돈할 수 있다면(2차원으로 표현할 수 있다면) 우리는 정보를 쉬우면서도 심도있게 따져볼 수 있다.

관계형 DBMS는 표라고 하는 위대한 성취를 기계화 한 것이다. 정보를 일단 표로 표현하기만 하면 정보를 매우 이해하기 쉬워진다. 이렇게 만들어진 표를 기계화 시키면 기계가 갖고 있는 엄청난 능력을 가진 표 로봇을 만들수 있다. 지금부터 우리는 정보기술의 위대한 성취인 관계형 데이터베이스, 그 중에서도 가장 인기있는 제품인 Oracle을 탐험해보자.

Oracle - 2. 가격 정책



Oracle - 3. 설치

접속 방법 :

```
sqlplus system/Silcroad105
```

Oracle - 4. 나 이거 할 줄 알아'의 최소 단위들

1. 설치
2. CRUD (Create, Read, Update, Delete)
3. Group
4. Run

Oracle - 5-1. 사용자와 스키마

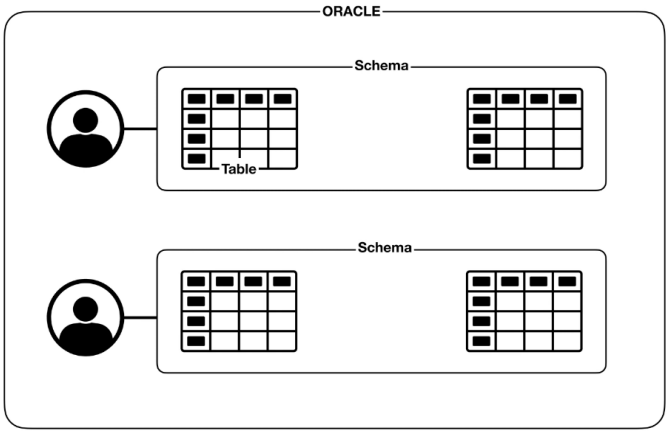
오라클 데이터베이스에서 우리가 하려고 하는 최종적인 일은 표(Table)에 정보를 기록하고, 기록된 정보를 읽는 것이다. 그러기 위해서는 우리는 표를 만들어야한다.

우리가 표를 만들다보면 표가 점점 많아질 것이다. 우리는 표가 많아지면 자연스럽게 서로 연관된 표를 그룹핑하고 싶어지고 그룹핑을 하기 위한 스키마라는 체계가 필요하다.

스키마는 서로 연관된 표들을 그룹핑하는 것이다. 스키마의 더 정확한 정의는 스키마에 속한 표들을 정의(설명)하는 것(표가 어떤 형태를 갖추고, 어떤 정보를 갖는지 등) 이다.

스키마를 사용하기 위해서 필수적으로 같이 생각해야되는 개념은 사용자라는 개념이다. 여러 컴퓨터들이 오라클 데이터베이스가 설치된 컴퓨터에 네트워크로 접속하여 작업을 하기 때문에 오라클에는 여러 사용자를 만들 수 있고 각각의 사용자는 자신이 관리하는 테이블에 접속할 수 있다.

사용자를 생성하게 되면 사용자가 속하는 스키마가 만들어지게 된다. 스키마와 사용자가 서로 같은 것은 아니지만 사용자를 만들면 스카마가 생성이 되고 그 스키마를 관리하는 것은 또 사용자이기 때문에 이 두 가지는 함께 생각할 필요가 있다.



Oracle - 5-2. 사용자 생성

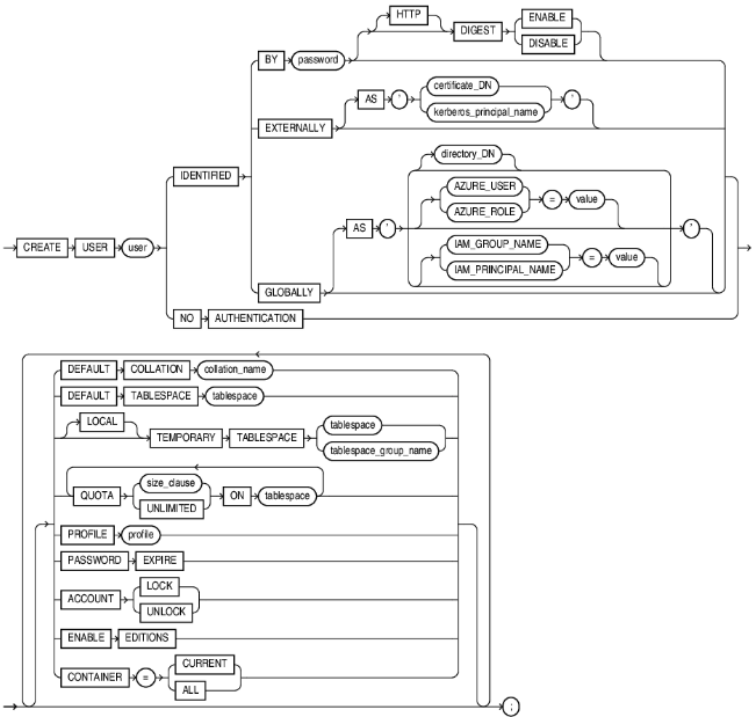
오라클에서 사용자와 스키마를 생성해보자.

사용자를 생성하기 위해서는 관리자 권한(system)으로 로그인 해야한다.

```
[silc@lms ~]$ sqlplus

SQL*Plus: Release 18.0.0.0.0 - Production on 화 7월 9 21:40:55 2024
Version 18.4.0.0.0

Copyright (c) 1982, 2018, Oracle.  All rights reserved.

사용자명 입력: system
비밀번호 입력:
마지막 성공한 로그인 시간: 화 7월  09 2024 21:23:46 -04:00

다음에 접속됨:
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0
```

Syntax

create_user::=



네모는 필수로 작성해야 할 내용이고, 타원은 바꿔야 할 내용이다.

IDENTIFIED를 쓰고 BY, EXTERNALLY, GLOBALLY 셋 중 하나 써야한다. 비밀번호를 이용해서 사용자를 생성할거기 때문에 BY를 쓸 것이다.

```
SQL> create user minsung identified by silcroad;
create user minsung identified by silcroad
            *
1행에 오류:
ORA-65096: 공통 사용자 또는 롤 이름이 부적합합니다.
```

CDB, PDB 개념 때문에 위와 같이 오류가 발생한다.

```
SQL> ALTER SESSION SET "_ORACLE_SCRIPT" = TRUE;

세션이 변경되었습니다.

SQL> create user minsung identified by silcroad;

사용자가 생성되었습니다.
```

ALTER SESSION SET "_ORACLE_SCRIPT" = TRUE; 를 입력하면 해결된다.

minsung이라는 사용자가 생성되었다. 즉, minsung이라고 하는 스키마도 생성되었다.

이제 minsung이라고 하는 사용자로 로그인해보자.

로그인이 되지 않는다. 왜냐하면 minsung이라는 사용자가 아무 권한이 없기 때문이다.

Oracle - 5-3. 사용자 권한 부여

사용자에게 권한을 부여하는 방법을 살펴보자.

```
grant dba to minsung;
```

grant : 권한 부여 명령어

dba : database administrator(데이터베이스 관리자)의 약자로 모든 일을 할 수 있는 권한

→ minsung 사용자에게 dba 권한을 주겠다.

실제로 데이터베이스를 관리하는 사용자와 데이터베이스를 이용하는 사용자를 이원화시켜서 관리하는 것이 가장 이상적이다.

minsung이라는 사용자는 데이터베이스를 사용할 사용자이기 때문에 원칙대로라면 minsung에게 필요한 최소한의 권한만 주는 것이 바람직하다. 하지만 교육을 위해서 가장 강력한 권한인 dba를 부여하자.

Oracle - 6. 테이블 생성

위에서 사용자를 만들었으니 사용자에 해당하는 스키마도 만들어졌을 것이고, 그 안에 표(Table)을 만들어보자.

아래와 같은 코드를 이용하여 만들 수 있다.

```
CREATE TABLE 테이블 명 (
    [컬럼명 1] [데이터 타입] [기본 값 (생략 가능)] [NULL (생략 가능)],
    [컬럼명 2] [데이터 타입] [기본 값 (생략 가능)] [NULL (생략 가능)],
    [컬럼명 3] [데이터 타입] [기본 값 (생략 가능)] [NULL (생략 가능)]
);
```

Table 명 : topic

id	title	description	created
1	ORACLE	ORACLE is ...	2019-7-29
2	MySQL		2019-7-1
3	MSSQL	MSSQL is ...	2019-1-1

위 Table을 만들어보자.

```
CREATE TABLE topic (
    id NUMBER NOT NULL,
    title VARCHAR2(50) NOT NULL,
    description VARCHAR2(4000) NULL,
    created DATE NOT NULL
);
```

테이블 명: topic

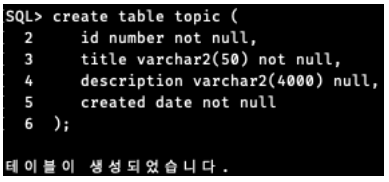
NUMBER : INT type

VARCHAR2(50) : CHAR type, 50보다 큰 문자가 들어오면 뒤가 잘림

VARCHAR2(4000) : CHAR type, 4000보다 큰 문자가 들어오면 뒤가 잘림, VARCAHR은 최대 4000글자가 들어올 수 있음

VARCHAR이 아닌 VARCHAR2를 사용하는 이유 : ORACLE에서 VARCHAR와 VARCHAR2는 현재 동의어지만 추후를 위해 VARCHAR2 사용을 권장한다. VARCHAR 데이터 타입의 경우 ORACLE 측에서 변경이 예정되어있기 때문에 변경사항이 반영되었을 때 그로인한 오류가 발생할 수 있다.

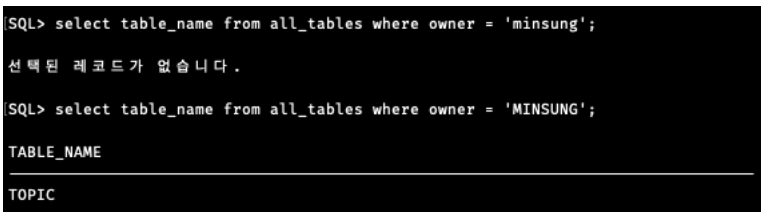
DATE : DATE 타입



표(Table)을 만들었으므로, 어떤 표가 현재 나의 스키마에 존재하는지 확인해보자.

```
SELECT table_name FROM all_tables WHERE OWNER = 'MINSUNG';
```

주의할 점 : 특정 소유자의 테이블을 조회할 때는 OWNER 조건을 대문자로 작성해야 한다.



Oracle - 7. 행 추가

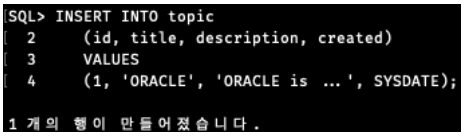
Table을 만들었으니, Table 안에 데이터를 CRUD하자.

Table 명 : topic

id	title	description	created
1	ORACLE	ORACLE is ...	24/07/10
2	MySQL	MySQL is ...	24/07/10
3	SQL Server	SQL Server is ...	24/07/10

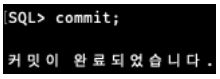
```
INSERT INTO topic
    (id, title, description, created)
    VALUES
    (1, 'ORACLE', 'ORACLE is ...', SYSDATE);
```

SYSDATE : 현재 시간을 적음



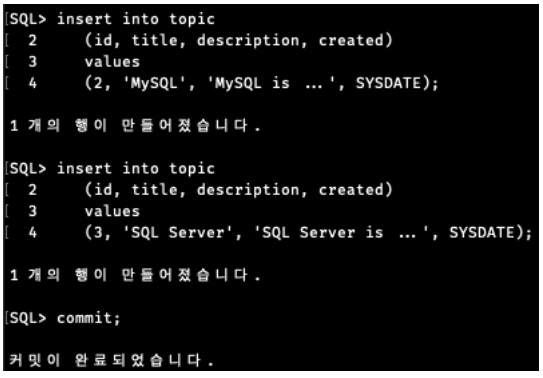
수정 작업(행 추가)를 했다면 commit이라는 명령을 내려야 함. commit을 하지 않는다면 반영이 되지 않아 다른 사람은 내가 추가한 것을 모르게 된다.

```
commit;
```




```
INSERT INTO topic
    (id, title, description, created)
    VALUES
    (2, 'MySQL', 'MySQL is ...', SYSDATE);

INSERT INTO topic
    (id, title, description, created)
    VALUES
    (3, 'SQL Server', 'SQL Server is ...', SYSDATE);
```

```
commit;
```



Oracle - 8. SQL이란?

엑셀과 같은 스프레드시트는 데이터(행)을 추가하는데 한계가 있다. 데이터를 65,000개 이상을 넣으면 파일을 분활해야할 수도 있다.

데이터베이스는 저장 장치의 한계만 없다면 데이터를 1억, 10억, 100억개 .. 를 넣을 수 있다. 이러한 엄청나게 많은 데이터에서 원하는 데이터를 읽는데 0.1초 밖에 걸리지 않는다.

그리고 데이터베이스는 명령어를 통해서 데이터베이스를 제어할 수 있다. 즉, 컴퓨터 프로그램을 통해서 명령어를 자동으로 만들어서 자동으로 만들어진 명령어를 데이터베이스 시스템에게 보내주기만 하면 데이터베이스 시스템이 그 명령어를 받아서 처리해줄 수 있다. 이것이 함의하는 것은 자동화이다.
엑셀은 사용자가 직접 하나하나 입력해야되기 때문에 자동화하는 것이 어렵다. 하지만 데이터베이스시스템은 명령어를 통해서 제어할 수 있기 때문에 자동화를 할 수 있다는 엄청난 장점을 가지고 있다. php, 파이썬, 자바 등을 이용해서 데이터베이스 시스템을 자동으로 제어를 하여 엄청난 일들을 할 수 있다.

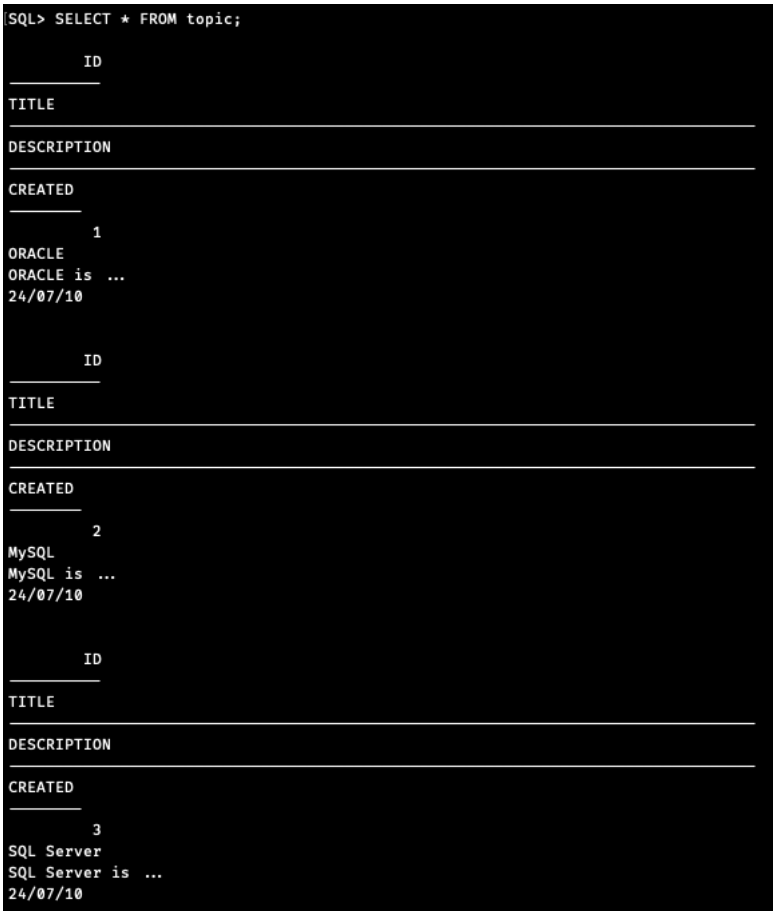
즉, 명령어를 이용해서 데이터베이스를 제어할 수 있다 라는 것인데 이러한 명령어를 SQL(Structured Query Language, 구조화된 정보를 처리하도록 요청하는 컴퓨터 언어)라고 한다.

Oracle - 9-1. 행 읽기 : Select 문의 기본 형식

모든 행과 모든 열을 가져오기 :

```
SELECT * FROM topic;
```

topic에서 모든 컬럼을 가져오겠다. (* : 모든 컬럼)

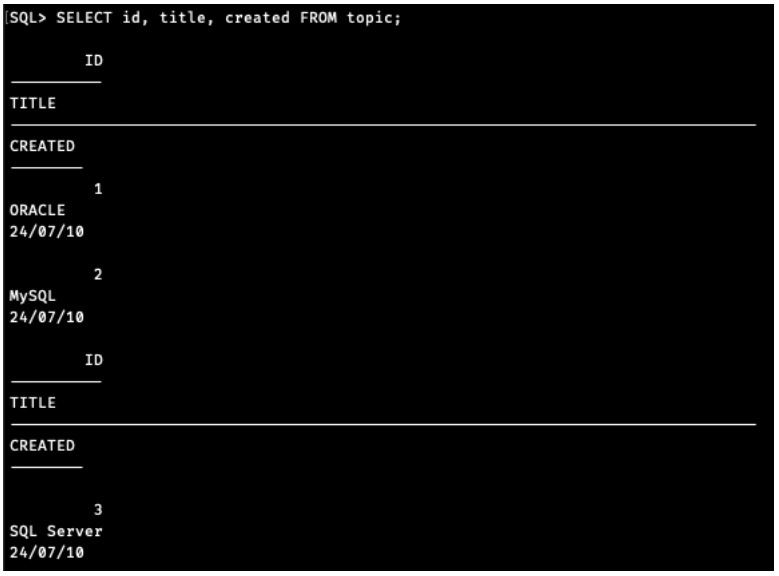


Oracle - 9-2. 행 읽기 : 행과 컬럼 제한하기

우리가 보고 싶은 컬럼이나 행이 전체가 아닐 수도 있다.

만약 id, title, created 컬럼만 보고싶다면 (description 컬럼을 보고싶지 않으면) 아래와 같은 명령어를 입력하면 된다.

```
SELECT id, title, created FROM topic;
```

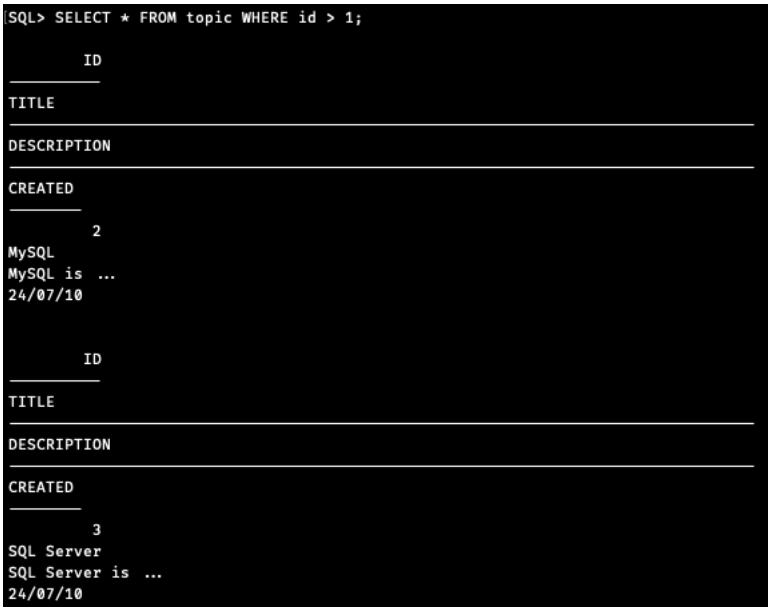


만약 id가 1번인 행만을 보고싶다면 아래와 같은 명령어를 입력하면 된다.

```
SELECT * FROM topic WHERE id = 1;
```

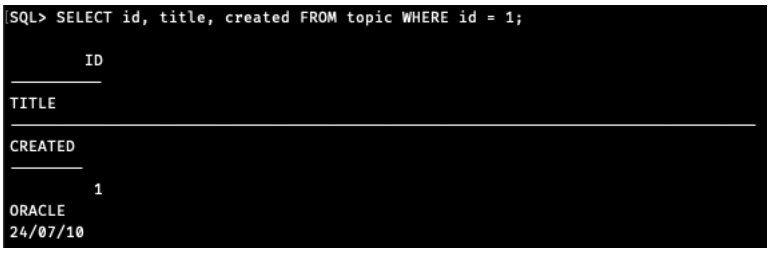
만약 id가 1보다 큰 행을 보고싶다면 아래와 같은 명령어를 입력하면 된다.

```
SELECT * FROM topic WHERE id > 1;
```



만약 id가 1인 id, title, created 컬럼만 보고싶다면 아래과 같은 명령어를 입력하면 된다.

```
SELECT id, title, created FROM topic WHERE id = 1;
```



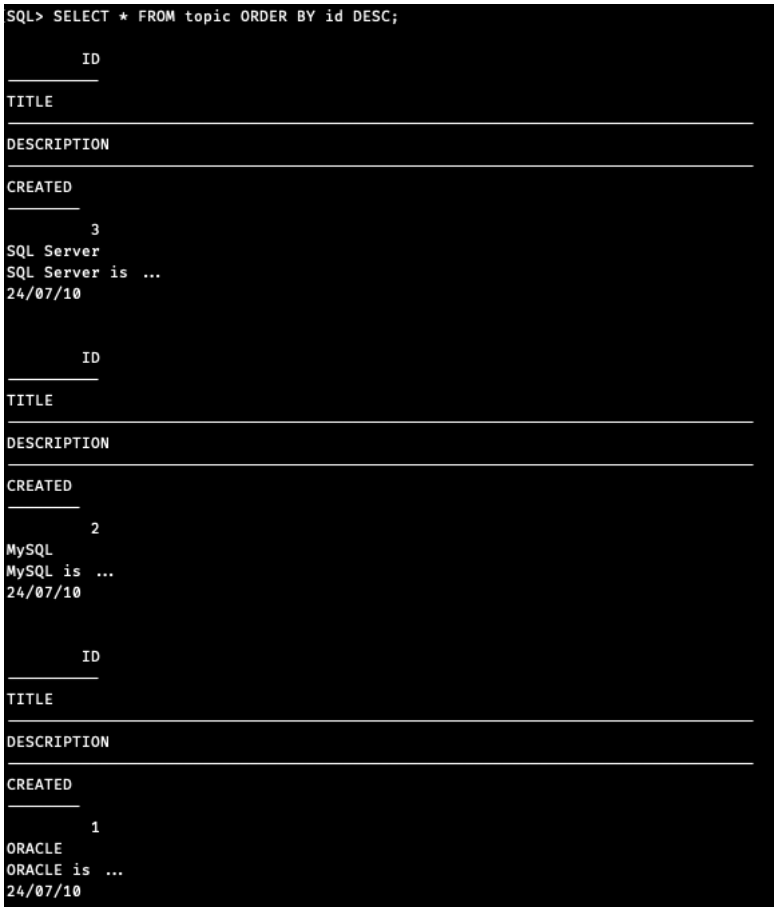
Oracle - 9-3. 행 읽기 : 정렬과 페이징

출력되는 결과의 정렬 상태와 행의 개수를 바꾸는 방법을 살펴보자.

정렬 :

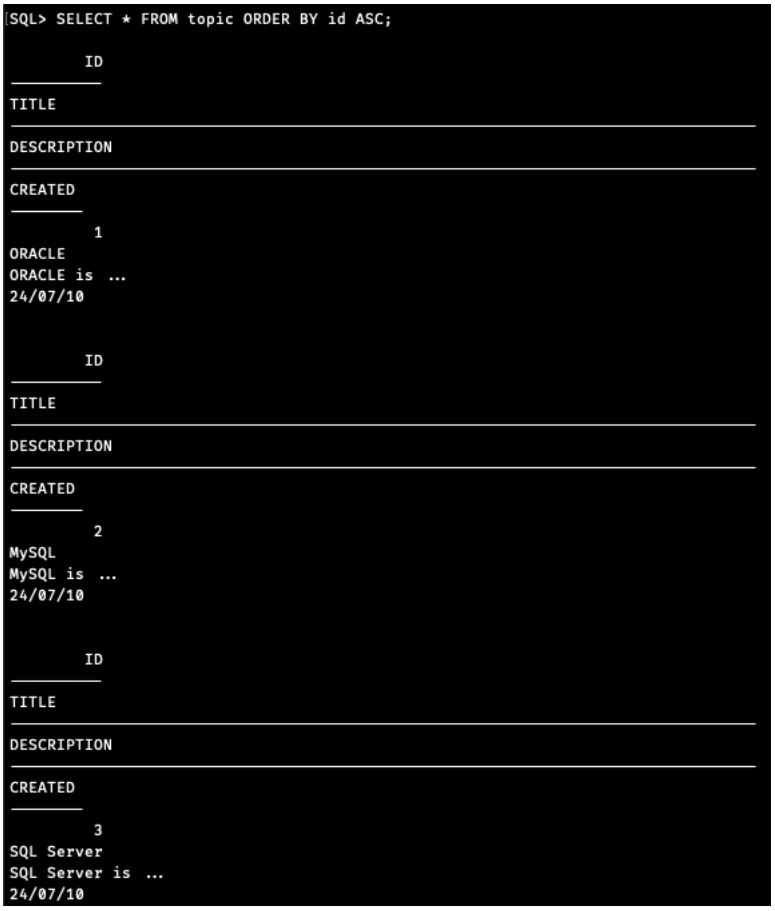
id 기준 내림차순 정렬 :

```
SELECT * FROM topic ORDER BY id DESC;
```



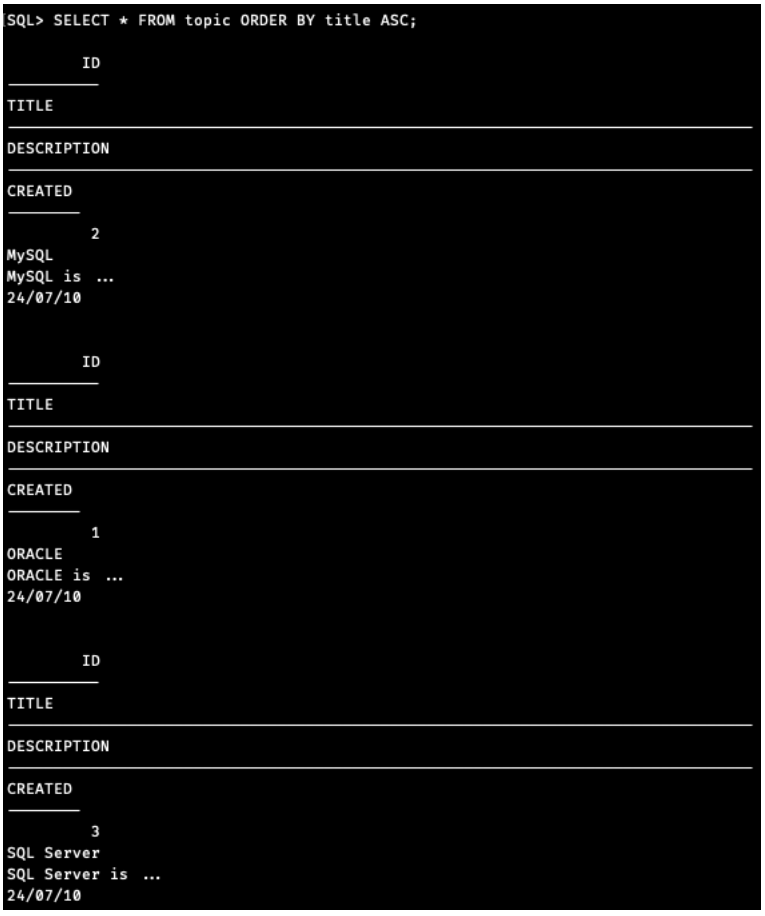
id 기준 오름차순 정렬 :

```
SELECT * FROM topic ORDER BY id ASC;
```



title 기준 오름차순 정렬 :

```
SELECT * FROM topic ORDER BY title ASC;
```

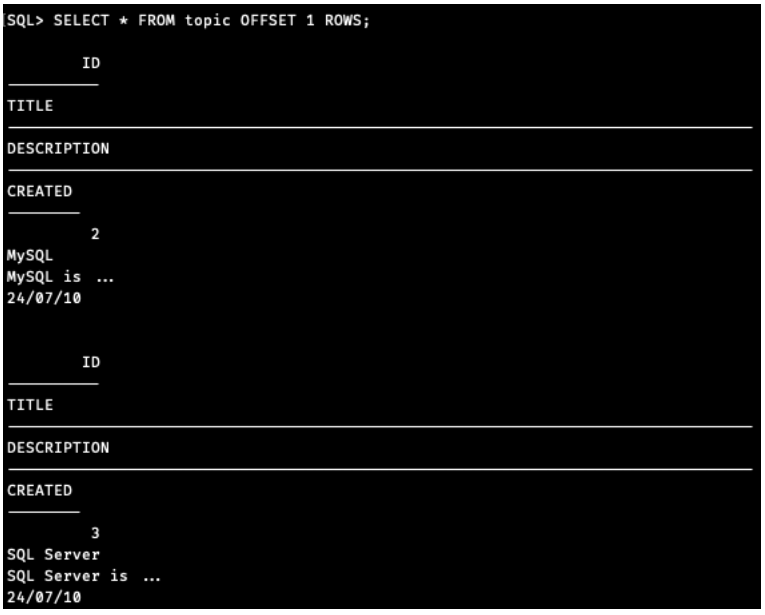


행의 개수 변경 - page :

데이터베이스에 데이터가 1억건이면 SELECT * FROM topic; 을 실행하면 과부하가 걸린다. 따라서 이럴 경우 우리가 원하는 행만 가져와야 한다. 이러한 기법을 page라고 한다.

```
SELECT * FROM topic OFFSET 1 ROWS;
```

OFFSET 1 : 0번째 이후에 있는 행들만 가져온다.



```
SELECT * FROM topic OFFSET 2 ROWS;
```

OFFSET 2 : 1번째 이후에 있는 행들만 가져온다.

→ OFFSET n : n-1번째 이후에 있는 행들만 가져온다.

fetch : n번째 이후에 있는 행들만 가져오는데 그 중에서 몇개를 가져올 것인가

```
SELECT * FROM topic OFFSET 1 ROWS FETCH NEXT 1 ROWS ONLY;
```

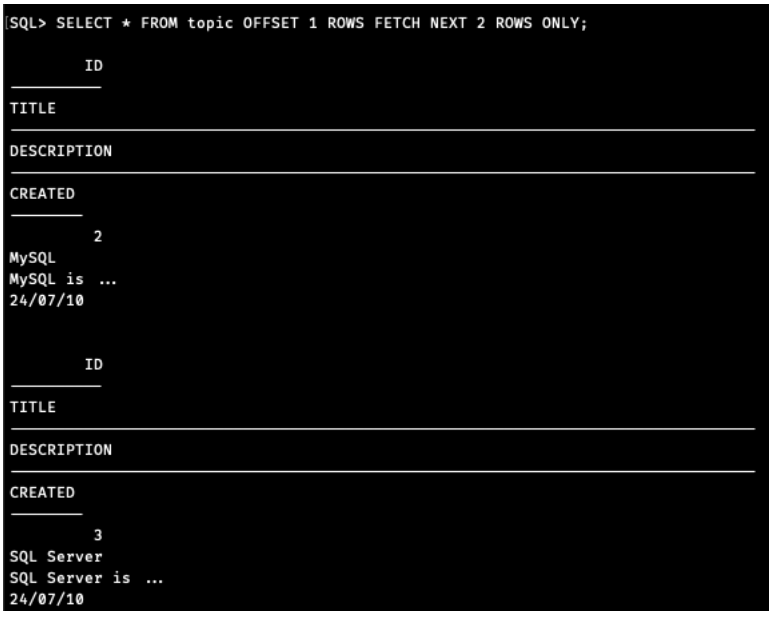
0번째 이후에 있는 행들만 가져오는데 0번째 이후에 1개만 가져오겠다.

→ 1개의 페이지가 1개의 행을 갖는다.


```
SELECT * FROM topic OFFSET 1 ROWS FETCH NEXT 2 ROWS ONLY;
```

0번째 이후에 있는 행들만 가져오는데 0번째 이후에 2개만 가져오겠다.

→ 1개의 페이지가 2개의 행을 갖는다.



Oracle - 10. 행 수정

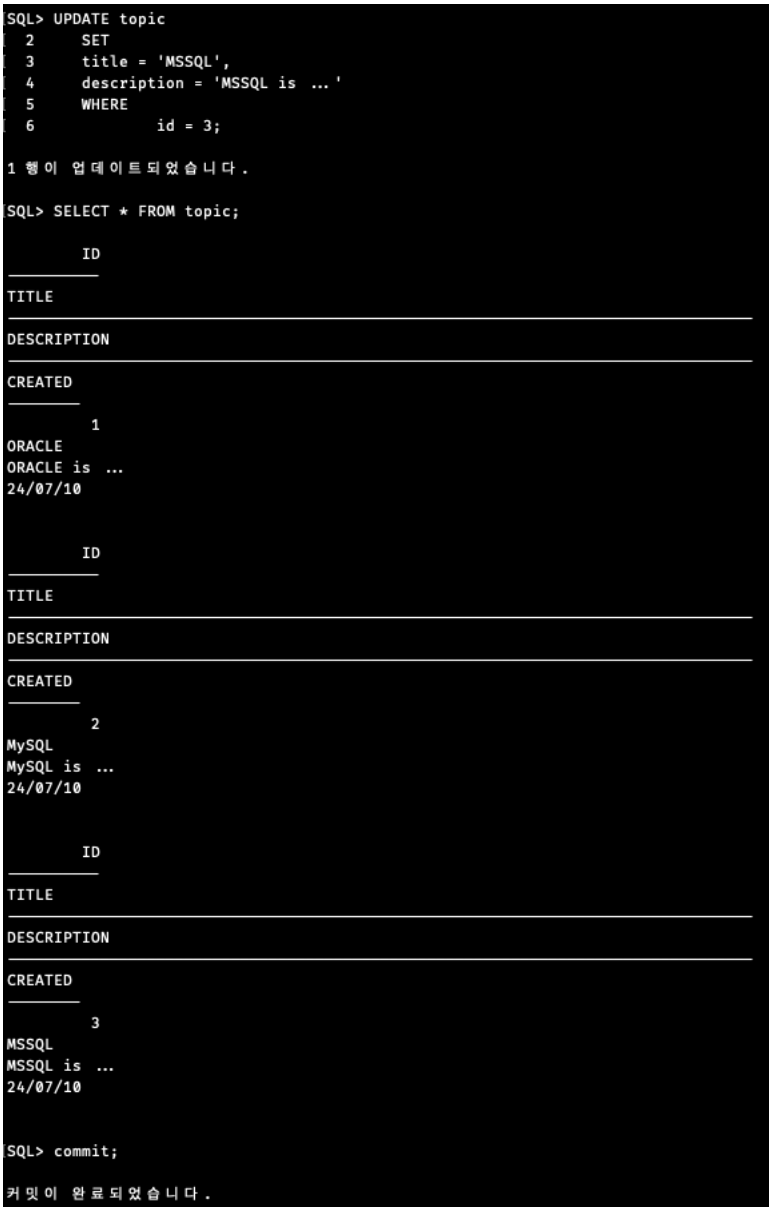
행을 수정해보자.

id 값이 3인 행의 title 값을 SQL Server에서 MSSQL이라고 바꾸고, description 값을 SQL Server is ...에서 MSSQL is ...로 바꾸자.

```
UPDATE topic
    SET
    title = 'MSSQL',
    description = 'MSSQL is ...'
    WHERE
        id = 3;
```

수정 작업을 하였으므로 commit;을 해주자.

```
commit;
```



Oracle - 11. 행 삭제

행 삭제는 주의하며 해야한다.

id가 3인 행을 삭제하고 싶다면 아래 명령어를 입력하면 된다.

```
DELETE FROM topic WHERE id = 3;
```

```
commit;
```

Oracle - 12. PRIMARY KEY

주민번호, 학번 처럼 중복되면 안되는 식별자가 존재한다.

위와 같은 표에서는 id가 중복되면 안된다.

근데 우리가 데이터를 넣다가 실수로 중복된 id를 넣을 수도 있다. 이를 방지하기 위해 primary key를 지정 해줘야한다.

테이블을 생성할 때 or 테이블을 생성한 후에 alter라는 명령어를 통해서 primary key를 지정 할 수 있다.

topic table을 버리고, id에 primary key를 지정해서 다시 만들어주자.

```
DROP TABLE topic;
```

```
CREATE TABLE topic (
      id NUMBER NOT NULL,
    title VARCHAR2(50) NOT NULL,
    description VARCHAR2(4000) NULL,
    created DATE NOT NULL,
    CONSTRAINT PK_TOPIC PRIMARY KEY(id)
);
```

CONSTRAINT : 제약 조건 (primary key는 table에 존재하지 않는 값만을 넣을 수 있다는 제약 조건임.)

PK_TOPIC : primary key의 고유한 이름

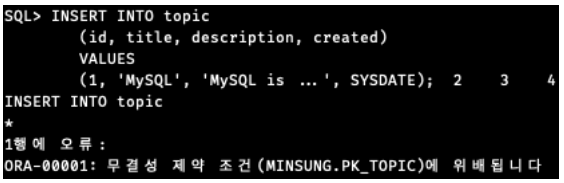
PRIMARY KEY : 어떤 제약 조건 ? PRIMARY KEY라는 제약 조건

(id) : id가 PRIMARY KEY

(id, title) : id, title이 PRIMARY KEY

```
INSERT INTO topic
    (id, title, description, created)
    VALUES
    (1, 'ORACLE', 'ORACLE is ...', SYSDATE);

INSERT INTO topic
    (id, title, description, created)
```

```
    VALUES
    (1, 'MySQL', 'MySQL is ...', SYSDATE);

INSERT INTO topic
    (id, title, description, created)
    VALUES
    (2, 'MySQL', 'MySQL is ...', SYSDATE);

INSERT INTO topic
    (id, title, description, created)
    VALUES
    (3, 'SQL Server', 'SQL Server is ...', SYSDATE);
```




```
commit;
```

Oracle - 13. SEQUENCE

id 값을 primary key로 지정하면 id 값의 중복이 없다고 확신할 수 있다.

primary key로 뭘 해야할까?

만약 4번째의 값을 추가 하려면 아래 코드(id = 4)를 입력하면 된다.

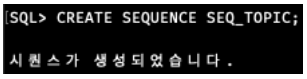
```
INSERT INTO topic
    (id, title, description, created)
    VALUES
    (4, 'MongoDB', 'MongoDB is ...', SYSDATE);
```

즉, 기존 테이블의 id 값 중 가장 큰 것을 찾아서 가장 큰 것에 1을 더한 행을 추가하면 된다.

근데 이것은 하다 보면 귀찮기도 하고, 내가 이 값을 찾아내는 동안에 누군가가 값을 찾아 넣어버리면 문제가 생긴다. → Oracle이 이것을 해줬으면 좋겠다. → SEQUENCE 사용

SEQ_TOPIC라는 SEQUENC를 CREATE :

```
CREATE SEQUENCE SEQ_TOPIC;
```



SEQUENCE를 만들었으니 이 SEQUENCE를 이용해서 자동으로 1씩 증가 시키는 primary key를 만들자.

```
INSERT INTO topic
    (id, title, description, created)
    VALUES
    (SEQ_TOPIC.NEXTVAL, 'MongoDB', 'MongoDB is ...', SYSDATE);
```

SEQ_TOPIC.NEXTVAL : SEQ_TOPIC의 다음 값(VALUE)

우리의 테이블에서 기존의 id = 1, 2, 3와 SEQ_TOPIC이 충돌할 수도 있으므로 기존의 데이터를 삭제하자.

```
DELETE FROM topic;
```

```
commit;
```

```
INSERT INTO topic
    (id, title, description, created)
    VALUES
    (SEQ_TOPIC.NEXTVAL, 'ORACLE', 'ORACLE is ...', SYSDATE);

INSERT INTO topic
    (id, title, description, created)
    VALUES
    (SEQ_TOPIC.NEXTVAL, 'MySQL', 'MySQL is ...', SYSDATE);

INSERT INTO topic
    (id, title, description, created)
    VALUES
    (SEQ_TOPIC.NEXTVAL, 'SQL Server', 'SQL Server is ...', SYSDATE);
```

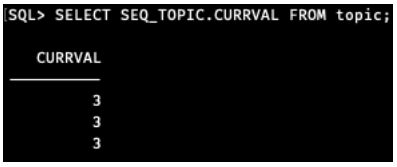
```
commit;
```

id가 무조건 1씩 증가하므로 값을 추가할 때 값이 충돌하지 않을까라는 생각은 안해도 된다.

sequence의 현재 값을 알고 싶을 때는 아래 명령어를 입력하면 된다.

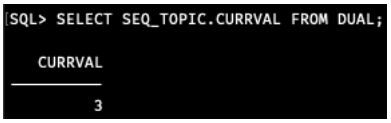
```
SELECT SEQ_TOPIC.CURRVAL FROM topic;
```

SEQ_TOPIC.CURRVAL : SEQ_TOPIC의 현재 값(CURRVAL)



위 명령어를 입력하면 topic 테이블에 있는 행의 개수 만큼 똑같은 값들이 출력된다. 이를 해결하기 위해 아래 명령어(DUAL : 가상의 느낌이 나는 특수한 테이블)를 입력하면 된다.

```
SELECT SEQ_TOPIC.CURRVAL FROM DUAL;
```



sequence는 primary key와 항상 같이 기억할 것 → 이 2가지는 같이 있을 때 강력해짐.

Oracle - 14. 서버와 클라이언트

지금까지 우리가 작업을 하면서, 한대의 컴퓨터에 Oracle 서버를 설치하고 또 SQL plus를 이용해서Oracle을 제어를 해왔다.

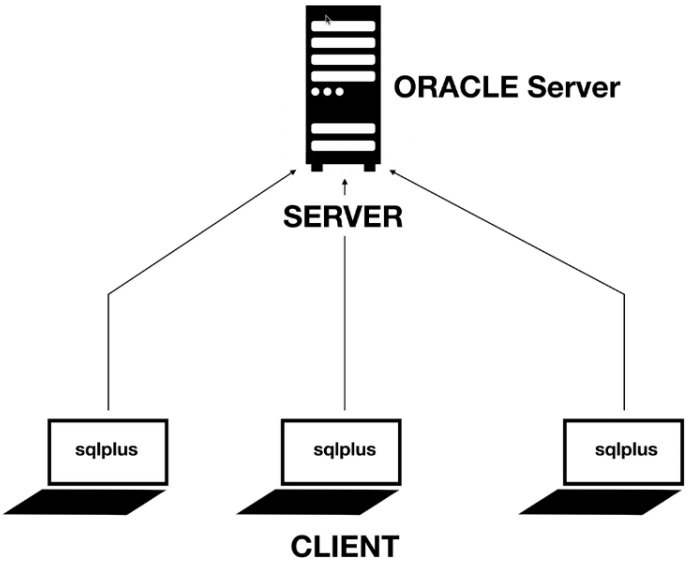
이렇게 비싸고 복잡한 프로그램을 한대의 컴퓨터에서만 사용하는 것은 너무 아깝다.
그리고 이렇게 되면 여러 사람이 사용 한다고 하면 하나의 컴퓨터를 사용하려고 줄을 서야하는가?

아니다. 인터넷이 연결되어 있는 시대인 만큼 Oracle을 1대의 컴퓨터에 설치하고 또 n대의 컴퓨터에는 sqlplus를 설치해서 인터넷을 통해서 Oracle이 설치 되어있는 컴퓨터의 Oracle을 제어할 수 있다.

이것을 살펴보기 위해서는 인터넷의 개념을 알아야 한다. 인터넷에 연결되어 있는 컴퓨터 한대 한대를 호스트라고 부른다.

데이터베이스의 호스트 이름이 뭐냐 라고 물어보면 데이터베이스가 설치 되어있는 컴퓨터의 ip나 도메인을 물어보는 것이다.

위 상황에서는 호스트가 Oracle이 설치 되어있는 컴퓨터, sql plus가 설치 되어있는 컴퓨터로 (1 + n)개이다.
네트워크가 연결되어있는 컴퓨터 한대 한대를 부르는 호스트라는 표현은 표현력이 부족하다. (1+n)대의 컴퓨터는 서로 역할이 다르다. sql plus가 설치 되어있는 컴퓨터는 정보를 요청하고, Oracle이 설치 되어있는 컴퓨터는 정보를 응답한다. 역할에 따라서 각자를 부르는 표현이 있으면 좋겠다 라는 생각을 하면서 만든 용어가 바로 client (sql plus가 설치 - 뜻 : 정보를 요청한다 = 고객)와 server (oracle이 설치 - 뜻 : 정보를 응답한다 = 서비스를 제공 = 사업자)이다.



server, client이 2가지가 인터넷을 이해하는 핵심적인 개념이다. server에 oracle을 설치하고, client에 sqlplus를 설치한다. 그러면 client가 서로 다른 공간에 있어도 server를 제어할 수 있다. 즉, 서로 다른 공간에 있어도 Oracle 데이터베이스를 사용할 수 있게 된다.

이 맥락에서 server에 설치되어있는 소프트웨어를 oracle database server라고 부르고, client에 설치되어 있는 것들을 oracle database client라고 부르고 이에 속하는 구체적인 제품은 sqlplus이다. (sqlplus는 oracle을 구매하면 기본적으로 제공하는 client이다. oracle server와 sqlplus는 같은 것이 아님.)

Oracle - 16.1 테이블의 분해 조립 - 분해하기

관계형 데이터베이스의 장점 :

1. 관계형 데이터베이스는 표라고 하는 형태로 데이터를 다룬다.
2. 표를 사용하다보면 표가 비대해지는데 이때 관계형 데이터베이스를 이용하면 표를 필요에 따라서 쪼갤 수 있고, 필요에 따라서 결합할 수 있다.

topic :

id	title	description	created
1	ORACLE	ORACLE is ...	2019-7-29
2	MySQL	MySQL is ...	2019-7-1
3	SQL Server	SQL Server is ...	2019-8-1

우리가 이런 표를 유지하다보면 정보가 복잡해 질 수 있다. 예를 들어, 각각의 글을 누가 썼나 라는 정보가 추가될 수도 있다.

topic :

id	title	description	created	name	profile
1	ORACLE	ORACLE is ...	2019-7-29	egoing	developer
2	MySQL	MySQL is ...	2019-7-1	egoing	developer
3	SQL Server	SQL Server is ...	2019-8-1	duru	DBA

이 표에 문제점이 있을까? 행이 1억개가 있다고 상상해보자.

1. 만약 egoing이라는 사람이 1억개의 글을 작성했는데 이 사람이 profile을 manager로 바꾸고 싶다. 1개를 바꾸는데 1초가 걸린다면 1억개를 바꾸는데 3년이 걸린다.
2. 만약 1행의 egoing과 2행의 egoing이 이름, 직업이 같은 동명이인이라고 생각해보자. 어떻게 구별해야할까?
3. 만약 taeho라는 저자가 아직 글을 안썼다면 taeho의 존재를 이 표에서 나타낼 수 없다. 즉, taeho는 글을 쓰기 전까지 정보시스템에서 없는 사람이 된다.

→ 이러한 문제점을 극복하는 방법은 표를 쪼개는 것이다.

topic :

id	title	description	created	author_id
1	ORACLE	ORACLE is ...	2019-7-29	1
2	MySQL	MySQL is ...	2019-7-1	1
3	SQL Server	SQL Server is ...	2019-8-1	2

author :

id	name	profile
1	egoing	manager
2	duru	DBA
3	taeho	data scientist
4	egoing	manager

첫번째 행을 읽기 : 첫 번째 행은 oracle이고 author_id가 1번이니까 1번을 author 테이블에서 찾는다. (egoing이라는 사람이 썼고 이 사람은 manager이다 라는 것을 알 수 있다.)

표를 쪼개니까,

1. 예전에는 1억개 행의 egoing이 profile을 developer에서 manager로 바뀌었다면, 1억번 수정해야되므로 비효율적이었다. 근데 표를 쪼개니까 author 테이블의 하나의 행만 manager로 수정하면 topic 테이블의 1억개의 행이 1번 id를 가르킨다고 하더라도 수정은 1번만 하면 되는 폭팔적인 효과를 가질 수 있게 된다.
2. 이름이 egoing이고 직업도 manager인 동명이인이 존재한다면, 이 사람은 id 값이 1번인 행과 별도의 id 값(4)을 가지고 있는 행을 통해 다른 사람인 것을 구별할 수 있게 되었다.
3. 이 table 상에서 분명히 저잔데 글을 한번도 쓴적이 없는 사람(taeho)은 author 테이블안에 존재한다.

→ 표를 쪼개는 것으로 엄청난 효과를 얻을 수 있다.

Oracle - 16.2 테이블의 분해 조립 - 조립하기

표를 쪼개기 전 : 읽기는 좋은데 수정(쓰기)이 힘들다.

표를 쪼개기 후 : 수정(쓰기)는 좋은데 읽기가 힘들다.

→ 장점과 단점을 합치자 : JOIN - 표를 쪼개고 필요할 때 마다 결합하자.

즉, 표를 쪼개서 저장하지만 읽을 때는 결합된 형태로 저장된 것처럼 읽자.

topic 테이블의 옆에 author 테이블을 붙여서 읽자. 기준은 topic 테이블의 author_id와 author 테이블의 id가 같을때이다. :

```
SELECT * FROM topic LEFT JOIN author ON topic.author_id = author.id;
```

id	title	description	created	author_id	id	name	profile
1	ORACLE	ORACLE is ...	2019-7-29	1	1	egoing	manager
2	MySQL	MySQL is ...	2019-7-1	1	1	egoing	manager
3	SQL Server	SQL Server is ...	2019-8-1	2	2	duru	DBA

Oracle - 16.3 테이블의 분해 조립 - 분해 실행하기

author 테이블을 추가하자.

```
CREATE TABLE author (
    id NUMBER NOT NULL,
    name VARCHAR2(20) NOT NULL,
    profile VARCHAR2(50),
    CONSTRAINT PK_AUTHOR PRIMARY KEY(id)
);
```

```
CREATE SEQUENCE SEQ_AUTHOR;
```

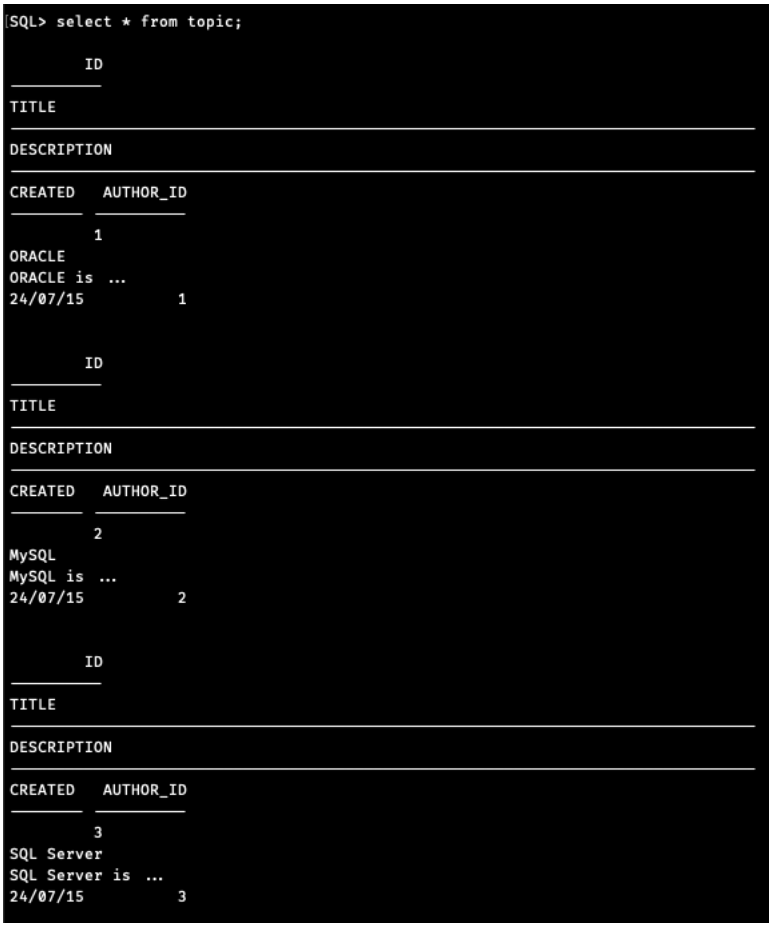
```
INSERT INTO author (id, name, profile) VALUES (SEQ_AUTHOR.nextval, 'egoing', 'developer');
INSERT INTO author (id, name, profile) VALUES (SEQ_AUTHOR.nextval, 'duru', 'DBA');
INSERT INTO author (id, name, profile) VALUES (SEQ_AUTHOR.nextval, 'taeho', 'data scientis
t');
```

```
commit;
```

```
ALTER TABLE TOPIC ADD (AUTHOR_ID NUMBER);
```

```
UPDATE TOPIC SET AUTHOR_ID = 1 WHERE id = 1;
UPDATE TOPIC SET AUTHOR_ID = 2 WHERE id = 2;
UPDATE TOPIC SET AUTHOR_ID = 3 WHERE id = 3;
```

```
commit;
```



author 테이블의 Primary Key 값을 AUTHOR_ID에 저장했다. 이렇게 연관된 다른 테이블의 Primary Key 값을 적어놓은 열을 foreign key라고 부른다.

Oracle - 16.4 테이블의 분해 조립 - 조립 실행하기 JOIN

topic 테이블과 author 테이블을 분리해서 저장했다.

저장은 분리해서 했지만 읽어올때 결합해서 마치 결합된 상태로 저장한 것 같은 환상을 만들어내보자. → JOIN

```
SELECT * FROM topic LEFT JOIN author ON topic.author_id = author.id;
```

이렇게 하면 id 값이 2개(topic의 id, author의 id)이므로 헷갈린다.

해결 방안 :

```
SELECT topic.id, title, name FROM topic LEFT JOIN author ON topic.author_id = author.id;
```

topic의 id, author의 id로 id가 2개이므로 topic.id라고 구분을 지어줘야한다.

이렇게 하면 작성한 나는 topic의 id 인 줄 알 수 있는데, 다른 사람들은 모를 수도 있다.

해결 방안 :

```
SELECT topic.id TOPIC_ID, title, name FROM topic LEFT JOIN author ON topic.author_id = aut
hor.id;
```

topic.id TOPIC_ID라고 하면 topic.id는 화면에 표시될 때 TOPIC_ID로 표시가 된다.

테이블의 이름에 대한 별명도 정할 수 있다 :

```
SELECT T.id TOPIC_ID, title, name FROM topic T LEFT JOIN author A ON T.author_id = A.id;
```

topic T라고 하면 topic은 T가 된다. 따라서 이 명령문의 topic을 사용하는 부분에서 topic을 T라고 할 수 있다.
author A라고 하면 author은 A가 된다. 따라서 이 명령문의 topic을 사용하는 부분에서 author을 A라고 할 수 있다.

topic의 id가 1인 행만 가져오고 싶을 때 :

```
SELECT T.id TOPIC_ID, title, name FROM topic T LEFT JOIN author A ON T.author_id = A.id WH
ERE T.id = 1;
```

WHERE T.id = 1 : topic의 id가 1인 행만 가져온다.

[데이터베이스 개요]

- 데이터베이스가 뭔가?
  - 개념적으로 보면, 데이터베이스는 데이터를 구조화/정규화 저장, 관리를 하기 위한 시스템이다.
    - DB Management System(DBMS)를 이용해서 데이터베이스를 쉽고 편하고 다양한 기능을 통해서 데이터베이스를 잘 관리하기 위한 시스템

- 데이터베이스를 크게 분류해보자.
  - 관계형 데이터베이스 (Relational database)
    - 데이터는 테이블 형태로 관리 / SQL 이용해서 데이터를 핸들링 한다
  - 비관계형 데이터베이스 (NoSQL database)
    - 데이터는 테이블 형태 외의 다양한 형태로 관리 (json 포맷 field:value / 그래프 데이터 등 ...) / MongoDB, 분산형 Apache 카산드라

- Quiz) Bigdata를 들어봤는가 ?
  - 전통적인 데이터 처리 프로그램으로 핸들링 하기 힘든 데이터들 (ex : OTT / 스트리머의 실시간 스트리밍 데이터들 같은 것)
  - 데이터의 규모(volumn) : 방대함
  - 속도 : 데이터가 생성되는 속도 / 소비하는 속도 → 빠르다 (스트리밍 데이터)
  - 데이터의 형태가 굉장히 다양 : 구조적인 데이터(테이블) / 반구조적인 데이터(json) / 비구조적인 데이터(동영상)

- 관계형 모델
  - 관계형 모델이란 뭘까?
    - 데이터베이스의 여러 모델이 있을텐데 그 중 하나. 결국은 데이터베이스의 모델 종류 중 한가지.

        - 데이터를 테이블 형태로 저장하고, 생성한 테이블들의 관계를 정의해서 데이터를 구성하는 모델.
    - 관계형 모델 특징
        - 데이터 독립성 : 물리적 / 논리적 데이터 독립성 제공 (데이터 블록(실제 데이터를 저장하는 물리적 공간) → 익스텐트(Extent) → 데이터 세그먼트 / 테이블 → 테이블 스페이스)
            - 데이터의 독립성 : 데이터 내부에서 데이터 파일의 저장소를 변경하고, 사이즈를 변경하는 등 뭘 해도 외부에서 봤을 땐 동일하게 동작을 해야한다. 그 이유가 뭘까?

              → 테이블 만들고 데이터 넣고 조회도 해보고 했었음. 근데 그게 실제로 내가 넣었던 order list에 데이터들이 데이터베이스가 105번에 설치되어있는것임. 이때 거기의 물리적인 파일 위치가 어디있는지 신경을 썼었나? table을 가지고 데이터만 만졌음 (데이터를 핸들링하기 위한 논리적인 개념)

              dbms는 이러한 논리적인 개념들은 프로그램이 올라와있으니깐 실제로 데이터가 논리적인 하드웨어 위에 저장이 되어야 할거아니야. → 이거는 dbms가 알아서 관리를 해줌. 우리는 전혀 모르는 형태로 테이블을 만들고 설계를 하고 조작을 했잖아? 우리가 실제로 이 테이블을 어느 데이터 블록에 명시를 하고 이 블록에 이 데이터들이 넣어라 명령 함. → 우리는 하드웨어 신경을 안씀. 어플리케이션 레벨만 신경 씀. → 하드웨어는 dbms가 알아서 해줌.

              우리가 신경써야되는건 논리적인 구조(데이터들을 어떻게 결합시키고, 핸들링하고) 임.

              오라클 dbms가 알아서 해줌. ⇒ 독립성

              근데 만약 독립적이지 않으면 이렇게 논리적으로 우리가 모델링을 할 수가 없음.
        - 데이터 무결성 : 데이터의 정확성과 일관성을 유지 (ex : pk(primary key)가 있는 열은 데이터의 unique를 보장해야 함)
        - SQL 사용 : 구조적 질의 언어를 사용해 데이터를 정의, 조작, 검색함 (DDL / DML / TCL)
        - entity의 관계성 : 테이블 간의 관계를 정의하고, 데이터를 결합하거나 핸들링 함
    - 관계형 모델에서 많이 언급되는 개념
        - Schema : 데이터베이스의 구조를 정의함
        - table : 데이터를 row/column 형태로 저장하는 구조(entity)
        - row : 테이블 내의 record를 나타냄. record는 각 column에 해당하는 값을 가지고 있음.
        - column : 테이블 내의 데이터의 속성. 고유한 이름하고 타입을 가지고 있음.
        - key : row를 식별할 수 있는 column의 집합 (PK/FK)

[실습]

- 실제 상황을 가정하고 그거에 맞춰서 실제로 만져보자
    - 예시를 위한 상황을 가정
        - 인터넷 의류 쇼핑몰을 운영하고 있음. 사장임.
        - 내가 지금 팔고 있는 의류는 총 5가지 (티셔츠, 바지, 양말, 신발, 모자)
        - 쇼핑몰에 가입한 회원들도 있어야 함
        - 회원들은 쇼핑몰에서 판매되는 의류 목록을 보고 물건을 사야함.
        - 사장은 회원들이 주문한 주문 목록을 확인한 후 물건을 준비하고 배송을 진행함.
    - 요구사항
        - 쇼핑몰 회원은 여러개 물품을 주문할 수 있음.
        - 회원이 주문한 물품에 대한 데이터는 따로 관리됨.
        - 사장은 주문 건을 확인해서, 각 주문건에 대한 상태를 '준비', '배송', '완료' 상태로 관리함.
        - 회원의 정보 (고객 번호 / 이름 / 나이)
        - 물품의 정보 (물품 번호 / 물품명 / 설명)
        - 회원정보와 물품정보 사이에는 관계성이 존재해야 한다. 관계성에 대한 부분을 아래 테이블에서 정의할 것 이다. 주문목록의 정보 (주문번호 / 물품번호 / 고객번호 / 고객이름 / 주문상태)

- E-R 모델을 이용한 데이터베이스 설계
    - Entity - Relationship 모델 개념

- 엔티티 : 데이터베이스에서 저장해야 할 object
- 속성 (Attribute) : 엔티티의 특성이나 정보 (column)
- Relationship : 엔티티 간의 관계. 연관성.

- E-R 모델을 이용한 데이터베이스설계 과정

1. 요구사항 수집 및 분석
2. E-R 다이어그램 작성
   - CUSTOM_LIST : 1명의 고객은 여러개의 물건을 주문할 수 있음
   - PRODUCT_LIST : 1개의 물건은 여러 고객한테 판매될 수 있음
   - ORDER_LIST : CUSTOM_LIST(eneity)와 PRODUCT_LIST(entity) 간의 관계(N)
   - CUSTOM_LIST
     - 고객번호(유니크한 값)
     - 이름 (유니크한 값)
     - 나이
   - PRODUCT_LIST
     - 물품번호(유니크한 값)
     - 물품명
     - 설명
   - ORDER_LIST
     - 주문번호(유니크한 값)
     - 물품번호(물품번호는 PRODUCT_LIST 의 물품번호를 따를 것임)
     - 고객번호(고객번호는 CUSTOM_LIST 의 고객번호를 따를 것임)
     - 고객이름(고객이름는 CUSTOM_LIST 의 이름를 따를 것임)
     - 주문상태(주문이 들어온 경우에는, 주문상태가 존재해야 됨)
3. E-R 다이어그램에서 명시한 내용을 바탕으로 스키마 변환

```
create table CUSTOM_LIST(
고객번호 primary key (유니크한 값)
이름    (유니크한 값)
나이
);
```

   - PRODUCT_LIST
     - 물품번호(유니크한 값)
     - 물품명
     - 설명
   - ORDER_LIST
     - 주문번호(유니크한 값)
     - 물품번호(물품번호는 PRODUCT_LIST의 물품번호를 따를 것임)
     - 고객번호(고객번호는 CUSTOM_LIST의 고객번호를 따를 것임)
     - 고객이름(고객이름는 CUSTOM_LIST 의 이름를 따를 것임) 주문상태(주문이 들어온 경우에는, 주문상태가 존재해야 됨)

   상에서 데이터베이스를 만들 것이다
   -> DB 접속해서 DBMS 상에서 실제로 쿼리를 수행해서 데이터베이스를 만든다

4. 3번에서 만든거 가지고 실제 DBMS 상에서 데이터베이스를 만들 것이다 -> DB 접속해서 DBMS 상에서 실제로 쿼리를 수행해서 데이터베이스를 만든다
    - 고객정보
      create table customer_list (
      customer_id number,
      customer_name varchar2(20),
      age number
      );
      alter table customer_list add constraint pk_customer_list primary key (customer_id, customer_name);
    - 물품정보
      create table product_list (
      product_id number primary key,
      product_name varchar2(30),
      description varchar2(100)
      );
    - 주문목록
      create table order_list (
      order_id number primary key,
      order_product_id number,
      order_customer_id number,
      order_customer_name varchar2(20),
      status varchar2(30) default 'prepare' not null
      );
      alter table order_list add constraint fk_product_list foreign key (order_product_id) references customer_list (product_id);
      alter table order_list add constraint fk_customer_list foreign key (order_customer_id, order_customer_name) references customer_list (customer_id, customer_name);

- 컴퓨터에서 test.sql을 작성하고 oracle 접속해서 @test.sql 하면 test.sql 에 있는 내용이 sqlplus에 한꺼번에 다 작성됨.

- 테이블 생성하고 commit을 안했는데 왜 테이블이 롤백되지 않았을까?
    - SQL을 종류에 따라 분류를 하면
        - DML : (Insert/Update/Delete)
        - DDL : (Create/Alter table)
        - TCL : (COMMIT/ROLLBACK)

      DML은 TCL과 연관되어 있어 commit 해야함.

      하지만 DDL은 TCL과 연관이 없어 commit 안해도 됨.

cre_obj.sql :

```
drop table order_list purge;
drop table customer_list purge;
drop table product_list purge;
```

order_list, customer_list, product_list 테이블을 해당 테이블이 Recycle Bin에 남지 않게(purge) 삭제한다.

```
create table product_list (
product_id number primary key,
product_name varchar2(30) not null,
```

```
description varchar2(100)
);
```

product_list 테이블을 생성한다.

```
create table customer_list (
customer_id number,
customer_name varchar2(20),
age number
);
```

customer_list 테이블을 생성한다.

```
alter table customer_list add constraint pk_customer_list primary key(customer_id, custome
r_name);
```

- alter table customer_list : customer_list 테이블의 구조를 변경하기 위한 명령어이다. 테이블에 새로운 컬럼을 추가하거나, 기존 컬럼을 수정하거나, 테이블에 제약 조건을 추가할 때 사용된다.
- add constraint pk_customer_list : 새로운 제약 조건을 추가한다. pk_customer_list는 새로 추가될 제약 조건의 이름이다.
- primary key(customer_id, customer_name) : customer_id와 customer_name 컬럼을 조합하여 primary key로 설정한다.

```
create table order_list (
order_id number primary key,
order_product_id number,
order_customer_id number,
order_customer_name varchar2(30),
status varchar2(30) default 'prepare' not null
);
```

order_list 테이블을 생성한다.

- status varchar2(30) default 'prepare' not null : 최대 30자의 문자열을 저장할 수 있는 varchar2 데이터 유형 status 열을 정의한다. 기본 값은 prepare로 설정되며, NULL 값은 가질 수 없다.

```
alter table order_list add constraint fk_product_list foreign key (order_product_id) refer
ences product_list (product_id);
```

- alter table order_list : order_list 테이블의 구조를 변경한다.
- add constraint fk_product_list : 새로운 제약 조건을 추가하는데, 이 제약 조건의 이름은 fk_product_list이다.
- foreign key (order_product_id) : order_list 테이블의 order_product_id 열을 foreign key로 설정한다.
- references product_list (product_id) :
  - 이 foreign key가 참조하는 테이블과 열을 지정한다. 여기서는 product_list 테이블의 product_id 열을 참조한다.
  - 즉, order_list 테이블의 order_product_id 열 값은 product_list 테이블의 product_id 열에 존재하는 값이여야 한다.

```
alter table order_list add constraint fk_customer_list foreign key (order_customer_id, o
rder_customer_name) references customer_list (customer_id, customer_name);
```

- alter table order_list : order_list 테이블의 구조를 변경한다.
- add constraint fk_customer_list : 새로운 제약 조건을 추가하는데, 이 제약 조건의 이름은 fk_customer_list이다.

- foreign key (order_customer_id, order_customer_name) : order_list 테이블의 order_customer_id와 order_customer_name 열을 조합하여 foreign key로 설정한다.
- references customer_list (customer_id, customer_name) :
  - 이 foreign key가 참조하는 테이블과 열을 지정한다. 여기서는 customer_list 테이블의 customer_id와 customer_name 열을 참조한다.
  - 즉, order_list 테이블의 order_customer_id와 order_customer_name 열 값의 조합은 customer_list 테이블의 customer_id와 customer_name 열 값의 조합 중 하나 여야 한다.

init_data.sql :

```
insert into product_list (product_id, product_name, description)
values (10, 'T-shirt', 'a summer necessity');
insert into product_list (product_id, product_name, description)
values (20, 'Pants', 'so cool item');
insert into product_list (product_id, product_name, description)
values (30, 'Socks', 'foot needs it');
insert into product_list (product_id, product_name, description)
values (40, 'Shoes', 'foot needs it, too');
insert into product_list (product_id, product_name, description)
values (50, 'Hat', 'a fashionable item');
commit;

insert into customer_list (customer_id, customer_name, age)
values (100, 'shkang', 38);
insert into customer_list (customer_id, customer_name, age)
values (200, 'mslee', 23);
insert into customer_list (customer_id, customer_name, age)
values (300, 'older man', 99);
commit;
```

scenario.sql :

```
-- 쇼핑몰의 회원은 의류 목록을 보고 주문을 한다.
--- shkang 고객의 경우 모자를 골라 주문을 한다
select * from product_list;
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
values (1, 5, 1, 'shkangg');
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
values (1, 5, 100, 'shkangg');
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
values (1, 5, 100, 'shkang');
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
values (1, 50, 100, 'shkang');
commit;
select * from order_list;

--- mslee 고객의 경우 티셔츠와 바지를 골라 주문을 한다
select * from product_list;
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
```

```
values (2, 10, 200, 'mslee');
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
values (3, 20, 200, 'mslee');
commit;
select * from order_list;

--- older man 고객의 경우 모자와 양말을 골라 주문을 한다
select * from product_list;
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
values (4, 30, 300, 'older man');
insert into order_list (order_id, order_product_id, order_customer_id, order_customer_nam
e)
values (5, 50, 300, 'older man');
commit;
select * from order_list;

-- 사장은 주문목록을 확인 후 older man 고객의 물건을 먼저 준비하여 배송 후, 주문 목록의 상태를 변경한다.
select * from order_list;
select * from order_list where ORDER_CUSTOMER_NAME='older man';

update order_list set STATUS='deliverying' where ORDER_CUSTOMER_ID=300;
commit;
select * from order_list;

-- 사장은 주문목록을 확인 후 mslee 고객의 바지를 배송 했는데, 로켓배송이라 바로 배송처리 되었다.
select * from order_list where ORDER_CUSTOMER_NAME='mslee';

select order_list.ORDER_ID, order_list.ORDER_PRODUCT_ID, product_list.PRODUCT_NAME, order_
list.ORDER_CUSTOMER_NAME from order_list
join product_list on order_list.ORDER_PRODUCT_ID = product_list.PRODUCT_ID
where ORDER_CUSTOMER_NAME='mslee';

update order_list set STATUS='completed' where ORDER_ID=3;
commit;
select * from order_list where ORDER_CUSTOMER_NAME='mslee';
select * from order_list;

-- 완료된 주문건은 삭제한다.
delete from order_list where STATUS='completed';
commit;
select * from order_list;
```

[실습 2]

ddl.sql :

```
-- 테이블 생성 1 : 제약 조건까지 한번에
create table emp (
    no varchar2(100) not null,
    name varchar2(100) default '홍길동' not null,
    addr varchar2(100) null,
    constraint emp_pk primary key (no)
);
```

```
-- 테이블 생성 2 : 제약 조건은 ALTER 문으로
create table emp (
    no varchar2(100) not null,
    name varchar2(100) default '홍길동' not null,
    addr varchar2(100) null
);

alter table emp add constraint emp_pk primery key (no);

-- 테이블 생성 3 : PK를 칼럼 선언할 때 한번에
create table emp (
        no varchar2(100) primary key,
        name varchar2(100) default '홍길동' not null,
        addr varchar2(100) null
);

-- 다른 테이블에서 데이터까지 복사해서 만들기
create table emp as select * from emp2; -- 모든 컬럼
create table emp as select name from emp2; -- 특정 컬럼
create table emp as select * from emp2 where 1 = 2; -- 데이터를 제외한 테이블 구조만 (1 = 2 하는
항상 거짓인 조건을 사용해 데이터를 선택하지 않음)

-- 참조 관계 정의
--- 1. 부모
create table 부서 (
    부서번호 char(10),
    부서명 char(10),
    constraint 제약조건이름 primary key (부서번호)
);

--- 2. 자식
create table 직원 (
    직원번호 char(10),
    부서명 char(10),
    constraint 제약조건이름 primary key (직원번호),
    constraint 제약조건이름 foreign key (부서명) references 부서 (부서명) on delete cascade
    --- on delete cascade : 옵션으로 부서 테이블의 특정 부서가 삭제되면, 해당 부서를 참조하는 직원 테이블
의 모든 행도 자동으로 삭제됨
);

-- 인덱스 생성
create index idx_emp on emp (name);
--- 이 구문은 emp 테이블의 name 컬럼에 대해 인덱스를 생성한다.
--- create index idx_emp : idx_emp라는 이름의 인덱스를 생성한다.
--- on emp (name) : emp 테이블의 name 컬럼에 대해 인덱스를 생성한다.
--- 인덱스를 통해 검색 속도를 향상시키고, 정렬 및 집계 성능을 개선하고 join 성능을 향상시킨다.

-- 제약조건 생성
ALTER TABLE EMP ADD CONSTRAINT EMP_PK PRIMARY KEY (NO);

ALTER TABLE EMP ADD CONSTRAINT EMP_CK CHECK(LENGTH(NAME) > 2);
--- name 컬럼의 값이 2자 이하인 행을 삽입하거나 업데이트하려고 할 때 오류가 발생한다.

-- 컬럼 추가
alter table emp add (phone varchar2(100));

-- 칼럼 명 변경
ALTER TABLE EMP RENAME COLUMN PHONE TO PHONE_NUM;
```

```
-- 칼럼 타입 변경
ALTER TABLE EMP MODIFY (PHONE_NUM VARCHAR2(200));

-- 칼럼 삭제
ALTER TABLE EMP DROP COLUMN PHONE_NUM;


-- 테이블 전체 삭제
DROP TABLE EMP;

-- 테이블은 삭제하지 않음. 데이터 전체 삭제. 롤백 불가능, 자동 커밋, 데이터 저장공간 모두 Release
TRUNCATE FROM EMP;

-- 테이블 이름 변경
RENAME EMP TO EMP_AFTER;
```

dml.sql :

```
-- 데이터 삽입 : 전체 컬럼 데이터 입력시
INSERT INTO EMP VALUES ('1', '홍길동', '영등포');

-- 데이터 삽입 : 특정 컬럼 데이터 입력시
INSERT INTO EMP (NO, NAME) VALUES ('1', '홍길동');

-- 여러 행을 한번에 삽입
INSERT INTO EMP
    SELECT NO, NAME, ADDR FROM EMP2 WHERE ADDR = '영등포';

-- 데이터 수정
UPDATE EMP SET ADDR = '강남' WHERE NAME = '홍길동';

-- 데이터 일부 삭제
DELETE FROM EMP WHERE NO = '1';

-- 데이터 전체 삭제. 롤백가능, 직접 커밋, 데이터 저장공간은 그대로 살아있음
DELETE FROM EMP;

-- 여러 테이블의 데이터를 한 테이블에 모으기
MERGE INTO 타겟테이블
    USING 소스테이블
    ON (타겟테이블.컬럼1 = 소스테이블.컬럼1)
    WHEN MATCHED THEN
        ... -- ON절 조건이 성립한다면 해당 쿼리 수행
    WHEN NOT MATCHED THEN
        ... -- ON절 조건이 성립하지 않는다면 해당 쿼리 수행
```

[데이터베이스 서버, 클라이언트]

- 테이블 쪼개는거, 결합(join)하는 것이 중요함 → 이를 위한 설계(모델링)가 엄청 중요함

- 서버, 클라이언트
    - 105번 장비에 oracle, silc, 등 여러 계정이 있는데 silc 계정에서는 sqlplus로 oracle 접속이 가능하지만, oracle 계정에서는 sqlplus로 oracle 접속이 안됨. (oracle 계정은 oracle을 설치해서 생긴 것임. 즉, oracle을 설치 한 계정임.)

- 왜 안될까?
  - silc 계정에는 sqlplus가 설치되어 있고, oracle에는 sqlplus가 설치되어있지 않아서? → 아니다. oracle 계정은 oracle을 설치한 계정이므로 sqlplus가 설치되어 있음.
  - 환경 변수를 silc 계정에다가만 주었기 때문이다.
  - 즉, oracle 계정에다가 oracle을 설치했지만 환경 변수를 세팅하지 않아서 sqlplus로 접속을 못하는 것임.
  - oracle 계정에서 ORACLE 환경 변수를 확인해보면 아래와 같이 적혀있는 것이 없음.

```
[oracle@lms ~]$ env | grep ORACLE
```

  - silc 계정에서 ORACLE 환경 변수를 확인해보면 아래와 같이 적혀있음.

```
[silc@lms ~]$ env | grep ORACLE
ORACLE_SID=XE
ORACLE_HOME=/opt/oracle/product/18c/dbhomeXE
```

- oracle 계정에서 sqlplus로 oracle에 접속하려면, oracle 계정에 환경 변수를 주면 된다.

```
[oracle@lms ~]$ env | grep ORACLE
ORACLE_SID=XE
ORACLE_HOME=/opt/oracle/product/18c/dbhomeXE
[oracle@lms ~]$ sqlplus minsung/silcroad

SQL*Plus: Release 18.0.0.0.0 - Production on 금 7월 19 01:56:53 2024
Version 18.4.0.0.0

Copyright (c) 1982, 2018, Oracle.  All rights reserved.

마지막 성공한 로그인 시간: 금 7월  19 2024 01:55:47 -04:00

다음에 접속됨:
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0

SQL>
```

  환경 변수를 세팅하니 접속이 가능하다.

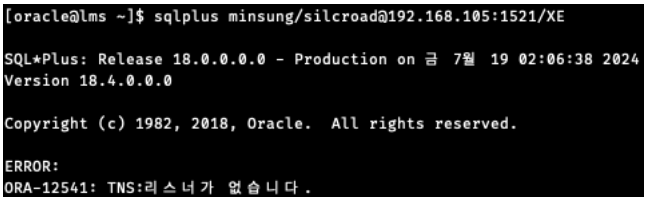
- 데이터베이스 서버에 접속할 때 sqlplus(클라이언트)라는 것을 이용해서 로컬(현재 장비)만 접속이 가능할까?
- 아니다. 다른 서버에도 접속 가능하다.
- 통신을 하기 위한 프로토콜이 몇가지 있다. 기본적으로 tcp(네트워크를 이용한 통신), 두번째로 프로세스 간 통신(네트워크 없이 프로세스끼리 통신함.)이 있다.
- 기본적으로 데이터베이스를 사용하는 주된 이유가 나 혼자만 쓰는 것이 아니라 많은 사람들이 데이터를 핸들링 하기 위해서 사용한다.
- 유저들이 데이터베이스 서버에 접속하기 위해 데이터베이스 서버가 설치되어있는 로컬에서만 접속해서 써야한다 ? → 이건 너무 힘들다.
- 보통은 클라이언트에서 네트워크로 접속해서 사용한다.

- 대다수는 아래처럼 로컬에서 접속하지 않는다.

```
sqlplus minsung/silcroad
```

- 아래처럼 네트워크 주소를 작성해주면, 해당 네트워크 노드의 데이터베이스 엔진에 접속 가능하다.

```
sqlplus minsung/silcroad@192.168.105:1521/XE
```



- 하지만 접근이 되지 않는다. 왜 그럴까?
- 192.168.105:1521에 리스너 접근 허용을 안줬기 때문이다.

- 로컬(127.0.0.1)는 리스너 접근 허용을 줬기 때문에 아래처럼 네트워크 주소를 작성해주면 접속이 가능하다.

```
sqlplus minsung/silcroad@127.0.0.1:1521/XE
```

- 강팀장님께서 1522번에 리스너를 띄워놔서 아래처럼 작성해도 접속이 가능하다.

```
sqlplus minsung/silcroad@192.168.105:1522/XE
```

[Database Object]

DataBase Object(데이터베이스 객체)는 데이터베이스 내에서 관리되고 조작되는 구조화된 데이터의 유형을 말한다. DataBase Object(데이터베이스 객체)에는 테이블 뿐만 아니라 다양한 종류의 객체가 포함된다. 주요 데이터베이스 객체의 종류는 다음과 같다.

1. 테이블(Table) :
   - 행(row)과 열(column)로 구성된 데이터 저장 구조이다.
   - 각 열은 특정 데이터 타입을 가지며, 각 행은 데이터 레코드를 나타낸다.
2. 뷰(View) :
   - 하나 이상의 테이블에서 파생된 가상 테이블이다.
   - 쿼리(데이터베이스에 요청) 결과를 저장하며, 물리적으로 데이터가 저장되지 않고 쿼리가 실행될 때마다 최신 데이터를 반환한다.
3. 인덱스(Index) :
   - 테이블의 데이터 검색 속도를 높이기 위해 사용하는 데이터 구조이다.
   - 특정 열 또는 열 조합에 대해 생성된다.
4. 시퀀스(Sequence) :
   - 숫자의 자동 증가를 제공하는 객체이다.
   - 주로 유니크 한 키 값을 생성하는 데 사용된다.
5. 트리거(Trigger) :
   - 특정 이벤트(예: 데이터 삽입, 갱신, 삭제)가 발생할 때 자동으로 실행되는 SQL 코드 블록이다.
     - SQL 코드 블록 : 하나 이상의 SQL 문을 묶어서 하나의 논리적 단위로 처리하는 것
   - 데이터 무결성 유지, 감사 로그 기록 등의 목적으로 사용된다.
6. 스토어드 프로시저(Stroed Procedure) :
   - 데이터베이스 내에 저장된 SQL 코드 블록이다.
   - 특정 작업을 수행하기 위해 호출될 수 있으며, 매개변수를 받아들일 수 있다.
7. 함수(Function) :
   - 입력값을 받아서 특정 작업을 수행한 후 결과를 반환하는 SQL 코드 블록이다.
   - 사용자 정의 함수로서 복잡한 쿼리 논리를 단순화하는 데 사용된다.
8. 패키지(Package) :

- 관련된 스토어드 프로시저와 함수를 그룹화 한 객체이다.
- 모듈 단위로 코드 관리를 용이하게 한다.

9. 동의어(Synonym) :
   - 데이터베이스 객체에 대한 별칭이다.
   - 주로 객체의 복잡한 이름을 간단하게 하거나, 객체의 위치를 추상화하는 데 사용된다.
10. 물리적 저장 구조(Physical Storage Structures) :
    - 데이터 파일(Data Files), 테이블스페이스(Tablespaces), 세그먼트(Segments), 익스텐트(Extents), 블록(Blocks) 등.
    - 데이터가 실제로 저장되는 물리적 저장 구조이다.

[데이터베이스의 물리적 저장 구조]

- Oracle 데이터베이스의 물리적 저장 구조 :
  1. 데이터 파일 (Data Files)
     - 정의 : 데이터 파일은 Oracle 데이터베이스의 물리적 저장 구조이다. 테이블, 인덱스, 클러스터 등의 데이터베이스 객체의 실제 데이터가 저장되는 곳이다.
     - 위치 : 데이터 파일은 디스크에 물리적으로 존재한다. 파일 시스템의 특정 디렉토리 내에 저장된다. 예를 들어, /u01/app/oracle/oradata/DB_NAME/users01.dbf와 같은 경로에 저장될 수 있다.
     - 관리 : 데이터 파일은 테이블스페이스의 일부로 관리된다. 하나의 테이블스페이스는 여러 개의 데이터 파일을 가질 수 있다.
  2. 테이블스페이스 (Tablespaces)
     - 정의 : 테이블스페이스는 논리적 저장 구조로, 하나 이상이 데이터 파일을 포함하는 논리적 저장 단위이다. 테이블, 인덱스 등의 데이터베이스 객체는 테이블스페이스 내에 저장된다.
     - 역할 : 테이블스페이스는 데이터베이스 객체를 논리적으로 그룹화하여 관리하고, 특정 저장소 관리 정책을 적용하는 데 사용된다.
     - 예시 : USERS 테이블스페이스가 있으며, 이 테이블스페이스는 여러 데이터 파일로 구성될 수 있다. 예를 들어, users01.dbf, users02.dbf 등

- 확인 방법 :
  1. 테이블 스페이스 확인 :

     먼저, 현재 사용자 스키마에 있는 모든 테이블을 조회한다.

     ```
     SELECT table_name FROM user_tables;
     ```

     특정 테이블이 저장된 테이블스페이스를 확인한다.

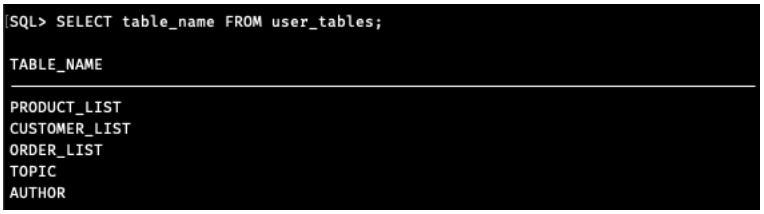
     ```
     SELECT table_name, tablespace_name
     FROM user_tables
     WHERE table_name = 'MY_TABLE';
     ```

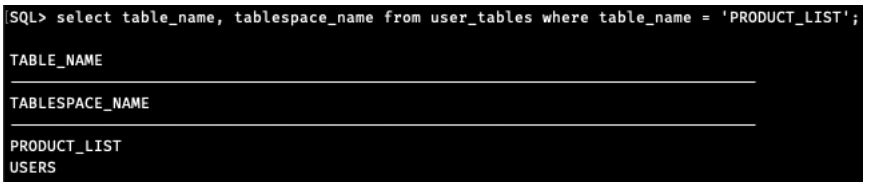
  2. 테이블스페이스에 포함된 데이터 파일 경로 확인 :

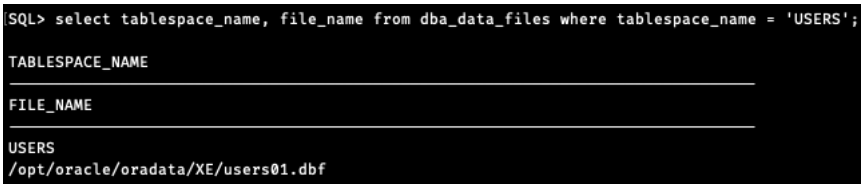
     ```
     SELECT tablespace_name, file_name
     FROM dba_data_files
     WHERE tablespace_name = 'USERS';
     ```

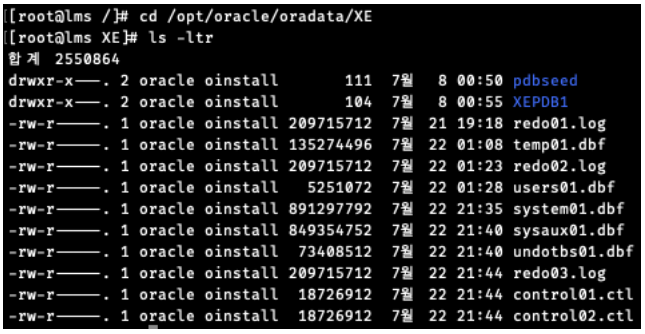
  3. 경로에 접속 :

     ```
     cd /opt/oracle/oradata/XE
     ```









[REDO, UNDO]

Oracle 데이터베이스에서 REDO와 UNDO는 데이터 무결성을 유지하고, 데이터 복구를 지원하는 중요한 역할을 한다.

REDO (Redo Log) :

- 정의
  - Redo Log : 데이터베이스에서 발생한 모든 변경 작업을 기록하는 로그 파일이다. 데이터베이스가 손상되거나 시스템 장애가 발생했을 때 데이터 복구를 위해 사용된다.
- 역할
  - 데이터 복구 : 시스템 장애나 오류가 발생했을 때, Redo 로그를 사용하여 최근의 모든 변경 작업을 다시 적용하여 데이터베이스를 복구한다.
  - 커밋된 트랜잭션 보장 : 트랜잭션이 커밋될 때, 그 트랜잭션의 모든 변경 사항이 Redo 로그에 기록된다. 이는 시스템 장애가 발생해도 커밋된 트랜잭션이 손실되지 않도록 보장한다.
    - 트랜잭션 : 데이터베이스의 상태를 변화시키기 위한 일련의 작업들을 하나의 논리적인 단위로 묶은 것
- 구조
  - Redo Log Files : 실제로 Redo 정보를 저장하는 파일이다. 일반적으로 두 개 이상의 파일 그룹으로 구성되어 있으며, 그룹 간에 로테이션하면서 사용된다.
  - Log Buffer : 메모리 내에서 Redo 로그 정보를 일시적으로 저장하는 버퍼이다. 변경 사항이 발생할 때마다 Redo 로그는 먼저 이 버퍼에 기록된다.
- 동작 방식
  1. 트랜잭션이 시작되고 데이터가 변경된다.

2. 변경된 데이터는 메모리의 데이터 버퍼 캐시에 저장된다.
3. 동시에 변경 사항이 Redo 로그 버퍼에 기록된다.
4. 트랜잭션이 커밋되면 Redo 로그 버퍼의 내용이 디스크의 Redo 로그 파일에 기록된다.

UNDO (Undo Tablespace) :

- 정의
  - Undo Tablespace : 트랜잭션이 수행한 변경 사항을 원래 상태로 되돌리기 위해 이전 데이터를 저장하는 공간이다.
- 역할
  - 롤백 (Rollback) : 트랜잭션이 중단되거나 취소되면 변경 사항을 원래 상태로 되돌리기 위해 Undo 데이터를 사용한다.
  - 읽기 일관성 (Read Consistency) : 트랜잭션이 실행되는 동안 다른 사용자가 일관된 데이터 뷰를 볼 수 있도록 보장한다. 예를 들어, 다른 사용자가 데이터를 읽을 때 트랜잭션이 아직 완료되지 않았다면 Undo 데이터를 사용하여 트랜잭션 시작 시점의 데이터를 제공한다.
  - 플래시백 기능 (Flashback) : Oracle 데이터베이스는 Undo 데이터를 사용하여 과거 시점의 데이터 상태를 복원하는 플래시백 기능을 지원한다.
- 구조
  - Undo Segments : Undo 데이터가 저장되는 세그먼트이다. 여러 Undo 세그먼트가 하나의 Undo 테이블스페이스에 존재할 수 있다.
  - Undo Tablespace : Undo 세그먼트를 포함하는 테이블스페이스이다. 이는 데이터베이스의 변경 사항을 롤백하고 읽기 일관성을 제공하기 위해 사용된다.
- 동작 방식
  1. 트랜잭션이 시작되고 데이터가 변경된다.
  2. 변경되기 이전의 데이터가 Undo 세그먼트에 기록된다.
  3. 트랜잭션이 커밋되거나 롤백되면 Undo 데이터를 사용하여 데이터의 원래 상태를 복원하거나 트랜잭션을 취소할 수 있다.

[REDO, UNDO 설정]

Oracle 데이터베이스를 설치하면, redo와 undo는 기본적으로 설정되고 구성된다. 이들은 데이터베이스의 중요한 구성 요소이며, 데이터베이스 무결성 및 복구 기능을 보장하는 데 필수적이다. 각 구성 요소가 어떻게 설정되고 동작하는지 살펴보자.

Redo Log 설정 :

- 기본 설정
  1. Redo Log 파일 : Oracle 데이터베이스를 설치할 때, 설치 과정에서 기본적으로 여러 개의 Redo Log 파일 그룹이 생성된다. 각 그룹은 하나 이상의 Redo Log 파일을 포함한다. 이들은 데이터베이스의 모든 변경 사항을 기록한다.
  2. Log Buffer : Redo Log 버퍼는 메모리 내에 위치한 영역으로, 데이터베이스에서 발생하는 변경 사항이 디스크의 Redo Log 파일에 기록되기 전에 임시로 저장된다.
- 설정 확인 :

  SQLPLUS를 사용하여 현재 Redo Log 설정을 확인할 수 있다.

```
-- Redo Log 파일 목록 조회
SELECT group#, member
FROM v$logfile;

-- Redo Log 그룹 상태 조회
SELECT group#, status, bytes
FROM v$log;
```

```
SQL> select group#, member
  2  from v$logfile;

    GROUP#
----------
MEMBER
--------------------------------------------------------------------------------
         3
/opt/oracle/oradata/XE/redo03.log

         2
/opt/oracle/oradata/XE/redo02.log

         1
/opt/oracle/oradata/XE/redo01.log


SQL> select group#, status, bytes
  2  from v$log;

    GROUP# STATUS                                           BYTES
---------- ------------------------------------------- ----------
         1 INACTIVE                                    209715200
         2 INACTIVE                                    209715200
         3 CURRENT                                     209715200
```

Undo 설정 :

- 기본 설정
  1. Undo Tablespace : Oracle 데이터베이스 설치 시, 하나 이상의 Undo 테이블스페이스가 생성된다. 이 테이블스페이스는 트랜잭션이 수행한 변경 사항을 원래 상태로 되돌리기 위한 정보를 저장한다.
  2. Automatic Undo Management : 대부분의 현대적인 Oracle 데이터베이스는 자동 Undo 관리 모드를 사용한다. 이는 Oracle이 자동으로 Undo Tablespace를 관리하고, 필요한 경우 자동으로 크기를 조절한다.
- 설정 확인

  SQLPLUS를 사용하여 현재 Undo Tablespace 설정을 확인할 수 있다.

```
-- 현재 사용 중인 Undo Tablespace 확인
SELECT tablespace_name
FROM dba_tablespaces
WHERE contents = 'UNDO';

-- Undo Tablespace 상태 및 파일 정보 조회
SELECT tablespace_name, file_name, bytes, status
FROM dba_data_files
WHERE tablespace_name IN (
  SELECT tablespace_name
  FROM dba_tablespaces
  WHERE contents = 'UNDO'
);
```

```
SQL> select tablespace_name
  2  from dba_tablespaces
  3  where contents = 'UNDO';

TABLESPACE_NAME
--------------------------------------------------------------------------------
UNDOTBS1
```

```
SQL> select tablespace_name, file_name, bytes, status
  2  from dba_data_files
  3  where tablespace_name in (
  4     select tablespace_name
  5     from dba_tablespaces
  6     where contents = 'UNDO'
  7  );

TABLESPACE_NAME
--------------------------------------------------------------------------------
FILE_NAME
--------------------------------------------------------------------------------
     BYTES STATUS
---------- ---------------------
UNDOTBS1
/opt/oracle/oradata/XE/undotbs01.dbf
  73400320 AVAILABLE
```

Redo 및 Undo 설정 변경 :

필요에 따라 Redo Log와 Undo Tablespace 설정을 변경할 수 있다.

- Redo Log 파일 추가

```
ALTER DATABASE ADD LOGFILE GROUP 4 ('/u01/app/oracle/oradata/ORCL/redo04.log') SIZE 50
M;
```

- Undo Tablespace 추가

```
CREATE UNDO TABLESPACE undotbs2 DATAFILE '/u01/app/oracle/oradata/ORCL/undotbs02.dbf' S
IZE 200M;
```

[데이터베이스 장애 처리]

어떻게 로그로 복구가 이루어지나?

복구에는 2가지 종류가 있는데, 사용자의 요청 또는 오류 발생 등으로 인해서 시스템이 트랜잭션을 철회하는 경우와 소프트웨어 문제나 하드웨어 문제 등으로 인해서 장애가 발생하고 데이터베이스 시스템이 재시작 복구(restart recovery)하는 경우가 있다.

트랜잭션 철회는 어떻게 ?

트랜잭션을 철회하는 경우는 시스템은 정상적으로 동작하고 있는 중이며 특정 트랜잭션만 철회하는 경우인데, 이 때 트랜잭션의 철회는 다음과 같이 이루어진다. 먼저 로그를 역방향으로 탐색하면서 트랜잭션 수행 순서의 역순으로 UNDO를 수행해야 정확하게 UNDO가 이루어질 수 있다.

UNDO를 수행하고 나면 해당 UNDO 작업에 대한 보상 로그 레코드(CLR, Compensation Log Record)라고 하는 REDO 전용 로그를 쓰게 되는데, UNDO를 하고 난 이후에 다시 UNDO를 해서 복구가 잘못 이루어지지 않도록 하기 위함이다. CLR은 이전 로그 레코드 위치를 UNDO 로그의 이전 로그를 가리키도록 하여 이후에는 한 번 UNDO된 로그를 다시 접근하여 재차 UNDO하게 되는 일이 발생되지 않도록 해준다. 이전 로그를 계속 탐색하면서 해당 트랜잭션의 시작 로그까지 도달하면 해당 트랜잭션의 철회 복구가 완료된 것이다.

장애로 인해 재시작하면 어떻게 복구가 되나?

장애 발생 이후 데이터베이스가 재시작 복구하는 경우에는 크게 3단계로 복구가 이루어진다.

1단계는 로그 분석 단계로, 마지막 체크포인트(checkpoint) 시점부터 최근 로그(EOL, End Of Log)까지 탐색하면서 어디서부터 시스템이 복구를 시작해야 하는지, 어느 트랜잭션들을 복구해야 하는지 등등을 알아내는 단계이다.

2단계는 REDO 복구 단계로 복구를 시작해야 하는 시점부터 장애 발생 직전 시점까지 REDO가 필요한 모든 로그를 REDO 복구를 하는 단계이다. 이 단계에서는 심지어 실패한 트랜잭션의 REDO 로그조차도 REDO를 하게 되는데, 언뜻 보면 불필요한 것으로 생각되지만 이렇게 하면 이후의 복구 단계를 매우 간단하게 하는 효과를 가져다 준다. 이 단계에서는 모든 트랜잭션에 대해서 REDO 복구만 한다는 점이 중요한데, 이러한 REDO 복구가 완료된 시점의 데이터베이스 상태는 장애 발생 시점의 상태와 같게 된다. 이전 상황을 그대로 재현하여 복원한다는 의미로 이 REDO 복구에서 이루어지는 작업을 repeating history라고 부른다.

마지막 3단계는 UNDO 복구 단계로 로그를 최신 시점부터 다시 역방향으로 탐색하면서 UNDO 복구가 필요한 로그들에 대해서 UNDO 복구를 수행한다. 여기서 수행하는 UNDO는 결국 트랜잭션 철회 시에 수행하는 UNDO와 같은 방식으로, repeating history를 통해 데이터베이스 상태를 장애 시점까지 복원해두고 UNDO 복구를 여러 트랜잭션의 철회로 간단하게 해결할 수 있다. 한 트랜잭션만 철회시키는

것이 아니라 여러 트랜잭션을 철회시킨다는 차이점만 존재한다. 이 단계의 UNDO 복구를 개별 트랜잭션의 UNDO와 구별하여 Global UNDO라고도 부른다.

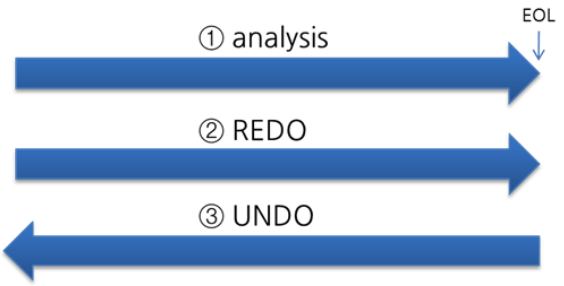


그림 4 재시작 복구 단계와 로그 접근 방향

로그를 통한 복구 과정 중에 특정 로그가 UNDO 내지는 REDO 복구가 필요한 것인지를 판단해야 할 필요가 있다. 이미 로그가 반영되었다면 그 로그에 대한 복구 연산은 필요치 않은데 이는 어떻게 해결할까? 모든 로그에는 LSN이라고 하는 식별자가 있는데, 데이터베이스의 모든 페이지는 page LSN을 가지고 있다. 이 page LSN은 페이지가 갱신될 때마다 해당 로그의 LSN으로 갱신된다. 즉, 모든 페이지는 해당 페이지를 마지막으로 갱신한 로그의 식별자를 포함하고 있으므로, 로그에 적용해야 할지 여부는 해당 로그의 LSN과 page LSN을 비교함으로써 판단할 수 있다. Page LSN이 어떤 로그의 LSN보다 예전 것이라면 해당 페이지는 반드시 해당 로그로 복구로 복구되어야 한다는 것을 의미하며, 반대로 page LSN이 해당 로그의 LSN과 같거나 더 최신의 값을 가지고 있다면 이 페이지는 해당 로그보다 나중에 쓰인 로그로 이미 갱신되었다는 것을 의미하므로 복구가 필요치 않다는 것을 의미한다. CUBRID는 page LSN으로 페이지 시작 부분의 8바이트의 공간을 사용하므로, 기본 16KB의 페이지를 사용하는 경우 실제 데이터가 저장되는 공간은 page LSN을 위한 공간을 제외한 16376 바이트가 된다.

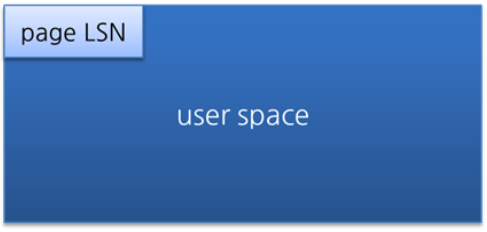


그림 6 데이터베이스의 페이지 구성

백업을 이용한 미디어 복구는 어떻게?

디스크 미디어(media)의 문제가 생겼을 때 수행하는 미디어 복구, 일명 아카이브(archive) 복구가 있는데, 이는 데이터베이스의 백업으로부터 복구를 하는 것을 의미한다. 데이터베이스 백업 기법에는 여러 가지가 있는데, 데이터베이스가 수행 도중에 트랜잭션들의 수행을 방해하지 않고 현재 스냅샷(snapshot)을 그대로 복사하는 퍼지(fuzzy) 백업이 CUBRID를 포함한 상용 DBMS가 사용하는 기법이다.

트랜잭션이 수행하고 있는 도중에 데이터베이스 이미지를 복사하는 것이기 때문에 미처 커밋하지 못한 일부 트랜잭션의 이미지가 복사될 수도 있고, 커밋한 트랜잭션의 데이터가 아직 반영되지 못한 채로 복사가 될 수도 있다. 이렇게 퍼지하게 복사한 데이터베이스 백업으로 어떻게 복원(restore)를 할까? 역시나 답은 로그에 있다. 미디어 복구 시에는 데이터베이스 백업과 (이에 포함된) 로그, 혹시 남아 있다면 장애 시점의 로그까지 활용하여 복구를 하게 되는데, 데이터베이스 백업은 데이터베이스 파일을 복사한 것이므로 이를 새로 복사해 둔 후 데이터베이스를 재시작한다고 생각하면, 미디어 복구 문제는 위에서 설명한 장애 발생 이후에 재시작 복구 작업과 결국 같은 문제가 된다. 결국 로그를 읽어서 퍼지하게 복사했던 데이터베이스 이미지에서 아직 미처 반영되지 못한 커밋했던 트랜잭션들을 다시 REDO해 주고, 결국 커밋 레코드가 포함되지 않은 트랜잭션들은 UNDO해 주면 된다. 이러한 미디어 복구 시점의 재시작 복구를 특별히 roll-forward 복구라고 부르기도 한다.

미디어 장애가 발생했을 때 마지막 데이터베이스 백업 이후의 모든 로그가 남아 있다면 장애 시점까지 손실 없이 데이터베이스를 복원할 수 있다. 불행히도 백업 이후의 일부 로그가 유실되었다면 최소한 백업 시점의 일관성이 유지되는 데이터베이스 시점까지는 복원이 가능하다. 미디어 복구를 이용하여 특정 시점으로 데이터베이스를 복원하는 것도 가능한데, 이는 roll-forward 과정을 현재 시점까지 전체를 수행하는 것이 아니라 DBA가 원하는 특정 시점까지만 수행하면 된다.