3. 테이블의 생성과 수정 그리고 삭제(1)[테이블의 생성]

3.1 테이블의 생성

테이블이란?

1. 테이블은 데이터베이스의 기본적인 데이터 저장 단위 입니다.
2. 데이터베이스 테이블은 사용자가 접근 가능한 모든 데이터를 보유하며 레코드와 컬럼으로 구성됩니다.

   관계형 데이터베이스가 아닌 예전의 데이터베이스 용어에서는 파일과 테이블이, 필드와 컬럼이, 그리고 레코드와 행이 동일시 되었습니다.
3. 테이블은 시스템내에서 독립적으로 사용되길 원하는 엔티티를 표현할 수 있습니다.
4. 테이블은 두 엔티티간의 관계를 표현할 수 있습니다.

   즉, 테이블은 고용자와 그들의 작업 숙련도 혹은 제품과 주문과의 관계를 표현하는데 사용될 수 있습니다. 테이블내에 있는 외래키(Foreign Key)는 두 엔티티 사이의 관계를 표현하는데 사용됩니다.
5. 비록 "테이블"이라는 말이 더 많이 사용되지만 테이블의 형식어는 "릴레이션"입니다.

컬럼

- 테이블의 각 컬럼은 엔티티의 한 속성을 표현한다.

행(ROW, 레코드)

- 테이블의 데이터는 행에 저장됩니다.

※테이블 생성시 제한사항과 고려할점

- 테이블 이름과 컬럼은 항상 알파벳 문자로 시작해야 하며 A~Z까지의 문자, 0~9까지의 숫자, 그리고 $,#,_(UnderBar)를 사용할 수 있습니다.(* 공백 사용 불가능)
- 테이블의 컬럼 이름은 30자를 초과할 수 없고, 예약어를 사용할 수 없습니다.
- 오라클 테이블 한 계정안에서 테이블 이름은 다른 테이블 이름과 달리 유사해야 합니다.
- 한 테이블 안에서 컬럼 이름은 같을 수 없으며 다른 테이블에서의 컬럼 이름과는 같을 수 있다.

```
** 테이블의 생성 문법(Syntax)*****************************************
*  CREATE TABLE [ schema. ] table_name
*    ( column datatype
*      [, column datatype .... ]
*      );
**********************************************************************
```

- schema : 테이블의 소유자
- table_name : 테이블 이름
- column : 컬럼의 이름
- datatype : 컬럼의 데이터 유형
- TABLESPACE : 테이블이 데이터를 저장 할 테이블스페이스를 지정합니다.

** 테이블의 생성 예제 **

1. 회원테이블만들기

회원테이블:USERTBL							
아이디	이름	생년	지역	국번	전화번호	키	가입일
USERID	USERNAME	BIRTH	LOCAL	MOBILE	MOBILE2	HEIGHT	WRITEDATE
VARCHAR	VARCHAR	INT	VARCHAR	VARCHAR	VARCHAR	INT	DATETIME
PRIMARY KEY							
NOT NULL	NOT NULL	NOT NULL	NOT NULL	NULL	NULL	NULL	NULL

```
SQL> CREATE TABLE USERTBL(
        USERID      VARCHAR(30)     PRIMARY KEY,
        (컬럼)       (데이터타입)      (제약조건)
        USERNAME    VARCHAR(30) NOT NULL,
        BIRTH       INT NOT NULL,
        LOCAL       VARCHAR(10) NOT NULL,
        HIREDATE    DATE,
        MOBILE      VARCHAR(3),
        MOBILE2     VARCHAR(10),
        HEIGHT      INT,
        WRITEDATE   DATETIME);
```

테이블이 생성되었습니다.

2. 제품테이블 만들기

제품테이블:PRODTBL				
제품명	가격	제조일자	제조회사	남은수량
PRODUCTNAME	PRICE	OPENDATE	COMPANY	STOCK
VARCHAR	INT	DATETIME	VARCHAR	INT
PRIMARY KEY				
NOT NULL	0	NULL	NULL	0

```
SQL> CREATE TABLE PRODTBL(
        PRODUCTNAME VARCHAR(50) PRIMARY KEY,
        PRICE INT DEFAULT 0,
        OPENDATE DATETIME,
        COMPANY VARCHAR(30),
        STOCK INT DEFAULT 0) ;
```

테이블이 생성되었습니다.

3. 구매테이블만들기

구매테이블:BUYTBL						
일련번호	아이디	물품명	분류	단가	수량	구매일
BUYNO	USERID	PRODUCTNAME	GROUPNAM	PRICE	AMOUNT	BUYDATE
INT	VARCHAR	VARCHAR	VARCHAR	INT	INT	DATETIME
PRIMARY KEY	FOREIGN KEY	FOREIGN KEY				
	NOTNULL	NOT NULL	NOT NULL	0	0	NOW()
AUTO_INCREMENT						

SQL>

3. 테이블의 생성과 수정 그리고 삭제(3)[데이터 타입]

3.3 데이터 타입

테이블을 생성할 때 각 컬럼에 지정할 수 있는 데이터 타입들입니다.

1) 정수형

DATA TYPE	설 명
BIT(M)	비트값 타입. 즉, 0과 1로 구성되는 binary 값을 저장한다. (M : 1~64, 생략 시 기본값은 1 로 설정)
BOOL	0은 false, 0이 아닌 값은 true 로 간주하는 논리형 데이터 ENUM(Y,N) 또는 TINYINT(1) 로 대체하여 사용하는 것을 권장
TINYINT(M)	부호 있는 수는 -128 ~ 127 부호 없는 수는 0 ~ 225 까지 표현 (1바이트)
SMALLINT(M)	부호 있는 수는 -32768 ~ 32767 부호 없는 수는 0 ~ 65535 까지 표현 (2바이트)
MEDIUMINT(M)	부호 있는 수는 -8388608 ~ 8388607 부호 없는 수는 0 ~ 16777215 까지 표현 (3바이트)
INT(M) INTEGER(M)	부호 있는 수는 -2147483648 ~ 2147483647 부호 없는 수는 0 ~ 4294967295 까지 표현 (4바이트)
BIGINT(M)	부호 있는 수는 -9223372036854775808 ~ 9223372036854775807 부호 없는 수는 0~18446744073709551615 (8바이트)

2) 고정소수점형

DATA TYPE	설 명
DECIMAL(M,D) NUMERIC	M자리 정수(정밀도)와 D자리 소수점(스케일)으로 표현 최대 65자리까지 표현할 수 있다.
BOOL	0은 false, 0이 아닌 값은 true 로 간주하는 논리형 데이터 ENUM(Y,N) 또는 TINYINT(1) 로 대체하여 사용하는 것을 권장

3) 날짜형

DATA TYPE	설 명
DATE	날짜를 표현하는 타입 (3바이트) 1000-01-01 ~ 9999-12-31
DATETIME	날짜와 시간을 같이 나타내는 타입 (8바이트) 1000-01-01 00:00:00 ~ 9999-12-31 23:59:59

TIMESTAMP	1970-01-01 00:00:00 ~ 2037-01-19 03:14:07 INSERT, UPDATE 연산에 유리하다. (4바이트)
TIME	시간을 표현하는 타입 (3바이트) -838:59:59 ~ 838:59:59
YEAR	연도를 나타낸다. (1바이트) 1901 ~ 2155, 70 ~ 69 (1970~2069) 부호 없는 수는 0 ~ 16777215 까지 표현 (3바이트)

4) 문자형

DATA TYPE	설 명
CHAR(M)	고정 길이를 가지는 문자열을 저장한다. (M : 0~255)
VARCHAR(M)	가변 길이를 가지는 문자열을 저장하며, 후행 공백을 제거하지 않는다. (M : 0~65,535) M이 0~255 이면 문자길이+1byte, ~65,535 이면 문자길이 +2byte
TINYBLOB TINYTEXT	1~255 개의 가변 길이를 가지는 문자열을 저장한다. (문자길이 +1byte)
BLOB TEXT	1~65,535 개의 가변 길이를 가지는 문자열을 저장한다. (문자길이+2byte) BLOB 는 바이너리 데이터, TEXT 는 문자 데이터 저장에 유리하다.
MEDIUMBLOB MEDIUMTEXT	1~16,777,215 개의 가변 길이를 가지는 문자열을 저장한다. (문자길이+3byte)
LONGBLOB LONGTEXT	1~429,496,729 개의 가변 길이를 가지는 문자열을 저장한다. (문자길이+4byte)
ENUM	문자 형태인 value 를 숫자로 저장하여 최대 65,535 개의 문자열 중 한가지를 반환 255 이하 value 는 1바이트, 65,535 이하 value 는 2바이트
SET	비트 연산 열거형, ENUM 형과 동일하게 문자열 값을 정수값으로 매핑하여 저장한다.

3. 테이블의 생성과 수정 그리고 삭제(2)[테이블 제약조건]

3.2 테이블의 제약조건

제약조건 (Constraint)

제약조건이란 **테이블에 부적절한 자료가 입력되는 것을 방지**하기 위해서 여러 가지 규칙을 적용해 놓는거라 생각하면 됩니다. 간단하게 테이블안에서 데이터의 성격을 정의하는 것이 바로 제약조건 입니다.

- 데이터의 무결성 유지를 위하여 사용자가 지정할 수 있는 성질 입니다.
- 모든 CONSTRAINT는 데이터 사전(DICTIONARY)에 저장 됩니다.
- 의미있는 이름을 부여했다면 CONSTRAINT를 쉽게 참조할 수 있습니다.
- 표준 객체 명명법을 따르는 것이 좋습니다.
- 제약조건은 테이블을 생성할 당시에 지정할 수도 있고,
  테이블 생성 후 구조변경(ALTER) 명령어를 통해서도 추가가 가능합니다.
- NOT NULL제약조건은 반드시 컬럼 레벨에서만 정의가 가능합니다.

****NOT NULL 조건 :** 컬럼을 필수 필드화 시킬 때 사용합니다.

```
SQL> CREATE TABLE emp(
        ename VARCHAR2(20)   NOT NULL );
```

이런식으로 하면 ename 컬럼에는 꼭 데이터를 입력해야만 합니다.
여기서 emp_nn_ename은 (테이블이름_제약조건이름_컬럼이름) 형식으로 CONSTRAINT NAME을 정의 합니다.
CONSTRAINT NAME은 USER_CONSTRAINTS 뷰(VIEW)를 통해서 확인할 수 있습니다.

******* 제약조건 확인하기 ********

```
SQL> SELECT   * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
        WHERE TABLE_NAME='MEMBER';
```

	CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	TABLE_SCHEMA	TABLE_NAME	CONSTRAINT_TYPE	ENFORCED
▸	def	mydb	PRIMARY	mydb	member	PRIMARY KEY	YES

** UNIQUE 조건 **

데이터의 유일성을 보장(중복되는 데이터가 존재할 수 없습니다.)

자동으로 index가 생성 됩니다.

```
SQL> ALTER TABLE DEPT ADD CONSTRAINT DEPT_DEPTNO UNIQUE(DEPTNO);
```

테이블이 변경 되었습니다.

이런식으로 하면 deptno 컬럼에 중복된 데이터가 들어갈 수 없습니다.

-- 제약조건의 삭제

```
SQL> ALTER TABLE emp
       DROP CONSTRAINT DEPT_DEPTNO;
```

테이블이 변경 되었습니다.

** CHECK 조건 **

컬럼의 값을 어떤 특정 범위로 제한 할 수 있습니다.

```
SQL> ALTER TABLE emp
       ADD CONSTRAINT emp_ck_sal
       CHECK (SAL >= 50 AND SAL<= 10000 );
```

테이블이 변경 되었습니다.

comm컬럼은 체크조건에서 제한을 하고 있으므로 1에서 100까지의 값만을 가질 수 있습니다.

또 체크 조건에서는 IN 연산자를 사용할 수 있습니다.

```
SQL> ALTER TABLE emp
       ADD CONSTRAINT emp_ck_comm
       CHECK ( comm IN (10000,20000,30000,40000,50000));
```

테이블이 변경 되었습니다.

comm 컬럼은 10000,20000,30000,40000,50000의 값만을 가질 수 있습니다.

**** DEFAULT 조건 ****

컬럼 기본값 지정 : 데이터 입력시에 입력을 하지 않아도 지정된 값이 입력될수 있습니다.

```
SQL> CREATE TABLE emp(
        hiredate DATE DEFAULT NOW() );
```

이런식으로 하면 hiredate 컬럼에 INSERT를 하지 않아도 오늘 날짜가 들어갑니다.

**** FOREIGN KEY 지정 ****

기본키를 참조하는 컬럼 또는 컬럼들의 집합입니다.

※ 외래키를 가지는 컬럼의 데이터 형은 외래키가 참조하는 기본키의 컬럼과 데이터형이 일치해야 합니다. 이를 어기면 참조무결성 제약에의해 테이블을 생성할 수 없습니다.

※ 외래키에 의해 참조되고 있는 기본키는 삭제할 수 없다.

※ ON DELETE CASCADE 연산자와 함께 정의된 외래키의 데이터는 그 기본키가 삭제 될 때 같이 삭제 됩니다.

```
SQL> ALTER TABLE emp ADD CONSTRAINT emp_fk_deptno
        FOREIGN KEY (deptno) REFERENCES dept(deptno) ON DELETE CASCADE;
```

테이블이 변경되었습니다.

ON DELETE CASCADE

: 다른 테이블의 기존 행에 있는 외래 키에서 참조하는 키가 포함된 행을 삭제하려고 하면 해당 외래 키가 포함되어 있는 모든 행도 삭제되도록 지정

**** 외래키 삭제하기**

SQL> ALTER TABLE [테이블명] DROP FOREIGN KEY [제약조건이름];

4. 데이터 조작어(DML)(5)[트랜잭션(commit,rollback)]

4.5 트랜잭션 (Commit과 Rollback)

**** 데이터베이스 TRANSACTION ****

- 트랜잭션은 데이터 처리의 한 단위 입니다.
- 오라클 서버에서 발생하는 SQL문들을 하나의 논리적인 작업단위로써 성공하거나 실패하는 일련의 SQL문을 트랜잭션이라 보시면 됩니다.
- ORACLE SERVER는 TRANSACTION을 근거로 데이터의 일관성을 보증 합니다.
- TRANSACTION은 데이터를 일관되게 변경하는 DML문장으로 구성됩니다
  (COMMIT, ROLLBACK, SAVEPOINT)

1. TRANSACTION의 시작
   - 실행 가능한 SQL문장이 제일 처음 실행될 때

2. TRANSACTION의 종료
   - COMMIT이나 ROLLBACK
   - DDL이나 DCL문장의 실행(자동 COMMIT)
   - 기계 장애 또는 시스템 충돌(crash)
   - deadlock 발생
   - 사용자가 정상 종료

**** COMMIT과 ROLLBACK ****

AUTOCOMMIT 확인하기

	@@autocommit
▶	1

1: TRUE(자동커밋), 2:FALSE

COMMIT : 변경사항 저장

ROLLBACK : 변경사항 취소

1. AUTOCOMMIT해제

SQL> SET AUTOCOMMIT=0;

**** Commit 과 Rollback 예제 ****

이전의 커밋(COMMIT)이 일어난 뒤부터 다음의 커밋(COMMIT)전 까지의 작업이 하나의 트랜잭션 이며, 커밋과 롤백(ROLLBACK)은 이러한 트랜잭션 단위로 데이터 베이스에서 발생한 작업을 저장, 삭제하는 일입니다.

- **Automatic commit** : DDL(Create, Alter, Drop), DCL(Grant, Revoke)
- **Automatic Rollback** : 비정상적인 종료, system failure

**** 예제 ****

SQL> DELETE FROM emp **WHERE** empno = 7521;

-> 한 개의 행이 삭제 되었습니다.

SQL> COMMIT;

-> 커밋이 완료 되었습니다.

한 개의 행을 삭재하고, COMMIT 문으로 데이터를 영구히 저장했습니다. 이 것은 하나의 트랜잭션이 여기서 종료되고 새로운 트랜잭션이 발생하는 것을 말합니다.

SQL> SELECT empno **FROM** emp **WHERE** empno = 7521;

-> 선택된 레코드가 없습니다.

SQL> INSERT INTO emp(empno, ename, hiredate) **VALUES**(9000, 'test', sysdate);

-> 한 개의 행이 작성되었습니다.

SQL> COMMIT;

-> 커밋이 완료 되었습니다.

SQL> DELETE FROM emp **WHERE** empno = 9000;

-> 한개의 행이 삭제 되었습니다.

SQL> SELECT empno **FROM** emp **WHERE** empno = 9000;

-> 선택된 레코드가 없습니다.

위의 예제를 보면 empno가 9000번인 데이터를 Insert한후 Commit으로 데이터를 저장한 다음에 데이터를 다시 삭제한 후 Select를 해보면 데이터가 검색되지 않는 것을 알 수 있습니다.

하지만 다른 유저에서는 커밋이나 롤백을 하기 전까지 이전에 Insert한 empno가 9000번인 데이터를 조회하면 데이터가 검색 됩니다.

데이터베이스에서의 이런 기능을 읽기 일관성이라고 합니다.

SQL> ROLLBACK;

-> 롤백이 완료 되었습니다.

(이전에 트랜잭션(커밋)이 발생하고나서 지금 발생한 ROLLBACK문 전까지의 작업의 취소를 말합니다.)

검색을 해보면 커밋이 완료된 시점의 레코드 하나가 검색 됩니다.

SQL> SELECT empno FROM emp WHERE empno = 9000;

```
   EMPNO
----------
      9000
```

-> 한 개의 행이 선택되었습니다.

**** SAVEPOINT 와 ROLLBACK TO ****

SAVEPOINT는 사용자가 트랜잭션의 작업을 여러개의 세그먼트로 분할할 수 있도록 하는 특별한 작업입니다.

SAVEPOINT는 **부분적인 롤백을** 가능하게 하기 위해 트랜잭션에 대한 중간점을 정의 합니다.

SQL> INSERT INTO emp(empno, ename, hiredate) **VALUES**(10000, 'test2', sysdate);

-> 한 개의 행이 작성되었습니다.

SQL> SAVEPOINT A;

-> 저장점이 생성 되었습니다. (여기서 SAVEPOINT를 생성했습니다.)

SQL> INSERT INTO emp(empno, ename, hiredate) VALUES(10001, 'test3', sysdate);

-> 한 개의 행이 작성되었습니다.

SQL> INSERT INTO emp(empno, ename, hiredate) VALUES(10002, 'test4', sysdate);

-> 한 개의 행이 작성되었습니다.

SQL> DELETE FROM emp WHERE empno IN(10000, 10001, 10002);

-> 세 개의 행이 삭제 되었습니다.

SQL> SELECT empno, ename FROM emp WHERE empno IN(10000, 10001, 10002);

-> 선택된 행이 없습니다.

SQL> ROLLBACK TO SAVEPOINT A;

-> 롤백이 완료 되었습니다. (SAVEPOINT까지만 롤백이 시행 됩니다.)

SQL> SELECT empno, ename **FROM** emp **WHERE** empno **IN**(10000, 10001, 10002);

-> 한 개의 행이 선택되었습니다.

```
     EMPNO ENAME
---------- ---------------
     10000 test2
```

SAVEPOINT까지만 롤백이 실행되었습니다. 그 결과 첫 번째 데이터는 그대로 남고, SAVEPOINT 후에 실행된 데이터 입력은 삭제되었습니다.