

Oracle SQL

오라클이란?

Oracle Corporation이란 미국의 기업에서 만든 데이터 베이스 관리 시스템 입니다.

Oracle 은 Database에서 가장 많이 쓰이며, 기능 또한 가장 좋은 것으로 알려진 최고의 Database 입니다.

Oracle 을 잘 다룬다고 하면 Database 에 대해서 전문적인 지식을 가지고 있다고 할 만큼 Database = Oracle 라는 인식이 저변에 확대되어 있는 것이 사실입니다.

Microsoft 사의 MS-SQL 버전이 높아지면서 많은 향상된 기능들과 편리한 기능들 그리고 Microsoft 의 OS와 접목된 최적화된 설계로 Oracle 의 시장을 넘보고 있지만 아직까지 Oracle 은 Database 분야에서 선두의 자리를 고수하고 있습니다.

Database 라는 것이 이제는 몰라서는 안 되는 IT 기술의 하나로 되고, 또한 Database 라는 곳이 쓰이는 곳이 많다 보니, DBA 의 중요성은 날로 높아진다고 할 수 있습니다. 이와 함께 많은 사람들이 관심을 갖게된 Oracle의 자격증이 OCP-DBA 자격증입니다.

Database 와 같은 경우는 처음에 공부하기가 조금은 힘든 분야입니다.

용어도 생소하고, 또한 개념 이해도 힘들기 때문에, 처음에 많은 시간을 투자하셔서 공부를 해야 후에 다른 어떤 Database 를 접하게 되어도 접근이 빠릅니다.

사실, Oracle 은 Database 가 복잡하고 많은 기능들이 있다 보니, 처음에 접근하기 힘든 것이 사실입니다만 Oracle DB 를 이해하고 난 다음에 다른 DB 를 공부하는 것은 그만큼 쉬운 것이 사실입니다.

공부를 하시면서 가장 유의할 점은 이해 위주의 공부를 하시라는 것입니다.

www.bpan.com에 있던 내용을 적어 보았습니다.

1. SQL 문의 종류
2. ORACLE사용자의 생성과 권한의 설정
 - 2.1. 사용자의 생성
 - 2.2. User의 변경 및 삭제
 - 2.3. 권한과 룰
 - 2.3.1. 시스템 권한(System Privileges)
 - 2.3.2. 객체 권한(Object Privileges)
 - 2.3.3. 룰(Role)
 - 2.3.4. 오라클 데이터베이스를 설치하면 기본적으로 생성되는 Role
3. 테이블의 생성과 수정 그리고 삭제
 - 3.1. 테이블의 생성
 - 3.2. 테이블의 제약조건
 - 3.3. 오라클 데이터 타입
 - 3.4. 테이블의 관리
4. 데이터 조작용어(DML)
 - 4.1. 데이터의 삽입, 수정, 삭제
 - 4.2. SELECT문 및 연산자
 - 4.3. 예명(Alias)
 - 4.4. 조인(Join)
 - 4.4.1. Equi Join, Non_Equi Join, Self Join
 - 4.4.2. Outer Join
 - 4.5. 트랜잭션(commit과 rollback)
 - 4.6. Commit과 Rollback 예제

- 5. 내장 함수(Sing-Row Functions)
 - 5.1. 숫자함수(Number Functions)
 - 5.2. 문자열 처리 함수(Character Functions)
 - 5.3. 날짜 처리 함수(Date Functions)
 - 5.4. 변환 함수(Conversion Functions)
 - 5.5. General Functions
 - 5.6. 기타 함수들
- 6. 그룹 함수(Group Functions)
 - 6.1. Group Function의 종류
 - 6.2. Group By절과 Having절
- 7. 서브쿼리(Subquery)
 - 7.1. 서브쿼리(Subquery)란?
 - 7.2. 단일 행(Sing-Row) 서브쿼리
 - 7.3. 다중 행(Multiple-Row) 서브쿼리
 - 7.4. 다중 열(Multiple-Column) 서브쿼리
 - 7.5. FROM절상의 서브쿼리(INLINE VIEW)
 - 7.6. 상관관계 서브쿼리
 - 7.7. 집합 쿼리(UNION, INTERSECT, MINUS)
- 8. 데이터 사전(Data Dictionary)
- 9. 오라클 객체
 - 9.1. 인덱스(Index)
 - 9.2. VIEW 테이블
 - 9.3. SEQUENCE(시퀀스)
 - 9.4. SYNONYM(동의어)

1. SQL문의 종류

DDL (Data Definition Language) : 데이터와 그 구조를 정의 합니다.

| SQL문 | 내 용 |
|--------|--------------------------------------|
| CREATE | 데이터베이스 객체를 생성 합니다. |
| DROP | 데이터베이스 객체를 삭제 합니다. |
| ALTER | 기존에 존재하는 데이터베이스 객체를 다시 정의하는 역할을 합니다. |

DML (Data Manipulation Language) : 데이터의 검색과 수정등의 처리

| SQL문 | 내 용 |
|--------|-----------------------------|
| INSERT | 데이터베이스 객체에 데이터를 입력 |
| DELETE | 데이터베이스 객체에 데이터를 삭제 |
| UPDATE | 기존에 존재하는 데이터베이스 객체안의 데이터 수정 |
| SELECT | 데이터베이스 객체로부터 데이터를 검색 |

DCL (Data Control Language) : 데이터베이스 사용자의 권한을 제어

| SQL문 | 내 용 |
|--------|-----------------------------|
| GRANT | 데이터베이스 객체에 권한을 부여 합니다. |
| REVOKE | 이미 부여된 데이터베이스객체의 권한을 취소합니다. |

2. ORACLE사용자의 생성과 권한의 설정

2.1. 오라클을 설치하면 기본적으로 생성되는 유저

오라클을 설치하면 기본적으로 생성되는 유저들

◆ 오라클을 설치하면 아래의 유저를 포함한 여러 유저들이 기본적으로 생성이 됩니다.

| 유저명 | 비밀번호 |
|--------|-------------------|
| SYS | CHANGE_ON_INSTALL |
| SYSTEM | MANAGER |
| SCOTT | TIGER |

◆ SYS

- 데이터베이스의 모든 기본 테이블과 뷰는 SYS스키마에 저장 됩니다.
- 기본테이블과 뷰는 oracle을 운영하는데 꼭 필요 합니다.
- SYS스키마의 테이블은 data dictionary의 무결성 유지관리를 위해 oracle에 의해 처리됩니다.
- 대부분의 database사용자는 SYS계정으로 접속하지 말아야 합니다.

◆ SYSTEM

- 관리정보를 화면으로 보여주는 추가 테이블과 뷰, 오라클 도구가 사용하는 내부테이블과 뷰를 만들 수 있습니다.
- SYSTEM유저는 모든 시스템 권한을 가지고 있습니다.

☞ 참 고

- SYS와 SYSTEM은 사용자를 만들거나 데이터베이스를 관리 할 수 있는 권한(DBA)을 가지고 있습니다.
막강한 어드민 유저라고 생각 하시면 됩니다.
- SYS나 SYSTEM유저에 테이블을 생성하거나, 일반 데이터들을 Insert하는 방법은 아주 좋지 않은 방법 입니다.
SYS나 SYSTEM유저는 데이터베이스를 관리하기 위해서만 사용되어야 합니다.

◆ SCOTT

- SCOTT유저는 일반 사용자로 오라클의 기본적인 SQL문을 테스트를 하기 위한 테이블과 데이터들이 있습니다.

2. ORACLE사용자의 생성과 권한의 설정

2.2. 사용자의 생성

사용자 생성 문법

```
CREATE USER user_name  
IDENTIFIED [BY password | EXTERNALLY ]  
[ DEFAULT TABLESPACE tablespace ]  
[ TEMPORARY TABLESPACE tablespace ]  
[ QUOTA { integer [ K | M ] | UNLIMITED } ON tablespace ]  
[ PASSWORD EXPIRE ]  
[ ACCOUNT { LOCK | UNLOCK } ]  
[ PROFILE { profile | DEFAULT } ]
```

- **user_name** : 생성될 사용자 이름
- **BY password** : 사용자가 데이터베이스에 의해 인증되도록 지정하며, 데이터베이스 유저 로그인시 사용하는 비밀번호입니다.
- **EXTERNALLY** : 사용자가 운영 체제에 의해서 인증되도록 지정합니다.
- **DEFAULT TABLESPACE**는 사용자 스키마를 위한 기본 테이블 스페이스를 지정 합니다.
- **TEMPORARY TABLESPACE**는 사용자의 임시 테이블 스페이스를 지정합니다.
- **QUOTA**절을 사용하여 사용자가 사용할 테이블 스페이스의 영역을 할당합니다.
- **PASSWORD EXPIRE** : 사용자가 SQL*PLUS를 사용하여 데이터베이스에 로그인할 때 암호를 재설정하도록 합니다.
(사용자가 데이터베이스에 의해 인증될 경우에만 적합한 옵션입니다.)
- **ACCOUNT LOCK/UNLOCK** : 사용자 계정을 명시적으로 잠그거나 풀 때 사용할 수 있습니다.(UNLOCK이 기본값입니다.)
- **PROFILE**: 자원 사용을 제어하고 사용자에게 사용되는 암호 제어 처리 방식을 지정하는데 사용됩니다.

2. ORACLE사용자의 생성과 권한의 설정

2.2. 사용자의 생성

새로운 USER를 생성하기 위해서는 **CREATE USER**문을 이용하면 됩니다.
USER를 생성하기 위해서는 USER생성 권한이 있는 사용자로 접속해야 합니다.

아래는 유저를 생성하는 아주 간단한 예제 입니다.

SQL PLUS를 실행시키고 SCOTT/TIGER로 접속을 합니다.

SQL>**CREATE USER TEST IDENTIFIED BY TEST;**

1행에 오류:

ORA-01031: 권한이 불충분합니다

※ SCOTT USER는 사용자 생성 권한이 없어서 사용자를 생성할 수 없습니다.

SQL>CONN SYSTEM/MANAGER — DBA Role이 있는 유저로 접속합니다.

SQL>**CREATE USER TEST IDENTIFIED BY TEST;** — USER를 다시 생성합니다.
사용자가 생성되었습니다.

위의 유저생성 예제는 테이블스페이스를 지정하지 않았습니다.
아래 [참고]의 내용은 유저를 생성 할 때 테이블스페이스를 지정해 주지 않을 경우의 문제 입니다.
테이블스페이스에 대한 강좌는 어드민 강좌를 참고해 주세요

※ 참고

- 임시 테이블스페이스를 지정해 주지 않으면 시스템 테이블스페이스가 기본으로 지정 되지만 시스템 테이블스페이스에 단편화가 발생할 수 있으므로 사용자를 생성할 때 임시 테이블스페이스를 따로 지정해 주는 것이 좋습니다.
- 또한 DEFAULT TABLESPACE도 사용자를 생성할 때 지정해 주지 않으면 기본적으로 시스템 테이블스페이스가 지정이 됩니다. 하지만 사용자를 생성할때 DEFAULT TABLESPACE를 지정을 해서 사용자가 소유한 데이터와 객체들의 저장 공간을 별도로 관리를 해야 합니다.
시스템 테이블스페이스는 본래의 목적(모든 데이터 사전 정보와, 저장 프로시저, 패키지, 데이터베이스 트리거등을 저장)을 위해서만 사용되어야 하지 일반사용자의 데이터 저장용으로 사용 되어서는 안됩니다.

2. ORACLE사용자의 생성과 권한의 설정

2.2. 사용자의 생성

새로 생성한 USER로 접속해 볼까요..

```
SQL> CONN TEST/TEST
```

ERROR:

ORA-01045: 사용자 TEST는 CREATE SESSION 권한을 가지고있지 않음; 로그온이 거절되었습니다

- 새로 생성한 TEST USER는 권한이 없어서 접근할 수가 없습니다.
- 모든 USER는 권한이 있고 권한에 해당하는 역할만 할 수 있습니다.
- TEST라는 USER를 사용하기 위해서도 권한을 부여해주어야 합니다.

```
SQL> CONN SYSTEM/MANAGER
```

연결되었습니다.

```
SQL> GRANT connect, resource TO TEST ;
```

권한이 부여되었습니다.

```
SQL> CONN TEST/TEST
```

연결되었습니다.

※ 권한에 대한 자세한 설명은 권한 설정에서 학습 하겠습니다.

※ 참고 테이블 스페이스란 ?

- 오라클 서버가 데이터를 저장하는 논리적인 구조입니다.
- 테이블스페이스는 하나 또는 여러개의 데이터파일로 구성되는 논리적인 데이터 저장 구조입니다.

테이블 스페이스에 대한 자세한 내용은 오라클 어드민의 테이블스페이스 강좌에서 학습하겠습니다.

2. ORACLE사용자의 생성과 권한의 설정

2.3. 사용자의 변경 및 삭제

● USER 변경하기 위해서는 **ALTER USER** 문을 사용합니다.

※ ALTER USER문으로 변경 가능한 옵션

- | | |
|------------------|-----------------|
| - 비밀번호 | - 운영체제 인증 |
| - 디폴트 테이블 스페이스 | - 임시 테이블 스페이스 |
| - 테이블 스페이스 분배 할당 | - 프로파일 및 디폴트 역할 |

사용자 수정 문법

```
ALTER USER user_name  
[ IDENTIFIED {BY password | EXTERNALLY } ]  
[ DEFAULT TABLESPACE tablespace ]  
[ TEMPORARY TABLESPACE tablespace ]  
[ PASSWORD EXPIRE ]  
[ ACCOUNT { LOCK | UNLOCK } ]
```

User의 Password 수정 예제

```
SQL>CONN SYSTEM/MANAGER      -- SYSTEM USER로 접속합니다.  
  
SQL>ALTER USER scott IDENTIFIED BY lion;  -- scott USER의 비밀번호를 lion으로 수정합니다.  
  
SQL>conn scott/lion           -- scott USER의 비밀번호가 수정된 것 을 확인할 수 있습니다.  
  
SQL>conn system/manager  
  
SQL>ALTER USER scott IDENTIFIED BY tiger;  -- scott USER의 비밀번호를 처음처럼 수정합니다.
```

2. ORACLE사용자의 생성과 권한의 설정

2.3. 사용자의 변경 및 삭제

사용자 삭제 문법

DROP USER user_name [**CASCADE**]

SQL>DROP USER scott; -- scott 유저를 삭제 하는 예제 입니다.

SQL>**DROP USER** scott **CASCADE**;

-- scott 유저가 객체를 소유하고 있을 경우에는 CASCADE옵션을 추가해서 삭제해야 합니다.

※ **CASCADE**를 사용하게 되면 사용자 이름과 관련된 모든 데이터베이스 스키마가 데이터 사전으로부터 삭제되며 모든 스키마 객체들 또한 물리적으로 삭제 됩니다.

데이터베이스에 등록된 사용자 정보의 확인

데이터베이스에 등록된 사용자를 조회하기 위해서는 DBA_USERS라는 데이터사전을 조회하면 됩니다.
SQL*Plus를 실행시켜 system/manager 로 접속을 합니다.

SQL>SELECT username, default_tablespace, temporary_tablespace
FROM ALL_USERS;

| USERNAME | DEFAULT_TABLESPACE | TEMPORARY_TABLESPACE |
|----------|--------------------|----------------------|
| SYS | SYSTEM | TEMP |
| SYSTEM | TOOLS | TEMP |
| CTXSYS | DRSYS | DRSYS |
| SCOTT | SYSTEM | SYSTEM |
| STORM | STORM | STORMTEMP |

...

위와 같이 유저와 테이블 스페이스에 대한 정보가 화면에 나옵니다.

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

권한이란?

오라클에서 권한(Privilege)은 특정 타입의 SQL문을 실행하거나 데이터베이스나 데이터 베이스 객체에 접근할 수 있는 권리입니다.

시스템 권한(System Privileges)

시스템 권한

- 시스템 권한은 사용자가 데이터베이스에서 특정 작업을 수행 할 수 있도록 합니다
- 약 126개의 시스템 권한이 있으며 그 수는 계속 증가하고 있습니다.
- 권한의 ANY 키워드는 사용자가 모든 스키마에서 권한을 가짐을 의미 합니다.
- **GRANT** 명령은 사용자 또는 Role에 대해서 권한을 부여 합니다.
- **REVOKE** 명령은 권한을 삭제 합니다.

시스템 권한의 종류 몇 가지

- CREATE SESSION : 데이터 베이스를 연결할 수 있는 권한
- CREATE ROLE : 오라클 데이터베이스 역할을 생성할 수 있는 권한
- CREATE VIEW : 뷰의 생성 권한
- ALTER USER : 생성한 사용자의 정의를 변경할 수 있는 권한
- DROP USER : 생성한 사용자를 삭제 시키는 권한

시스템 권한 부여 문법

```
GRANT [system_privilege | role] TO [user | role | PUBLIC]
[WITH ADMIN OPTION]
```

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

[문법설명]

- system_privilege : 부여할 시스템 권한의 이름
- role : 부여할 데이터베이스 역할의 이름
- user, role : 부여할 사용자 이름과 다른 데이터 베이스 역할 이름
- **PUBLIC** : 시스템 권한, 또는 데이터베이스 역할을 모든 사용자에게 부여할 수 있습니다.
- **WITH ADMIN OPTION** : 권한을 부여 받은 사용자도 부여 받은 권한을 다른 사용자 또는 역할로 부여할 수 있게 되며, 만약 사용자가 WITH ADMIN OPTION과 같이 역할을 부여 받는다면 부여된 역할은 그 사용자에게 의해 변경 또는 삭제 될 수 있습니다.

시스템 권한 부여 예제

```
SQL>GRANT CREATE USER, ALTER USER, DROP USER TO scott
WITH ADMIN OPTION.
권한이 부여되었습니다.
```

★설명 : scott 사용자에게 사용자를 생성, 수정, 삭제 할 수 있는 권한을 부여 했습니다.
scott 사용자도 다른 사용자에게 그 권한을 부여 할 수 있습니다.

시스템 권한 박탈 문법

```
REVOKE [system_privilege | role] FROM [user | role | PUBLIC]
```

```
SQL>REVOKE CREATE USER, ALTER USER, DROP USER
FROM scott
권한이 회수되었습니다.
```

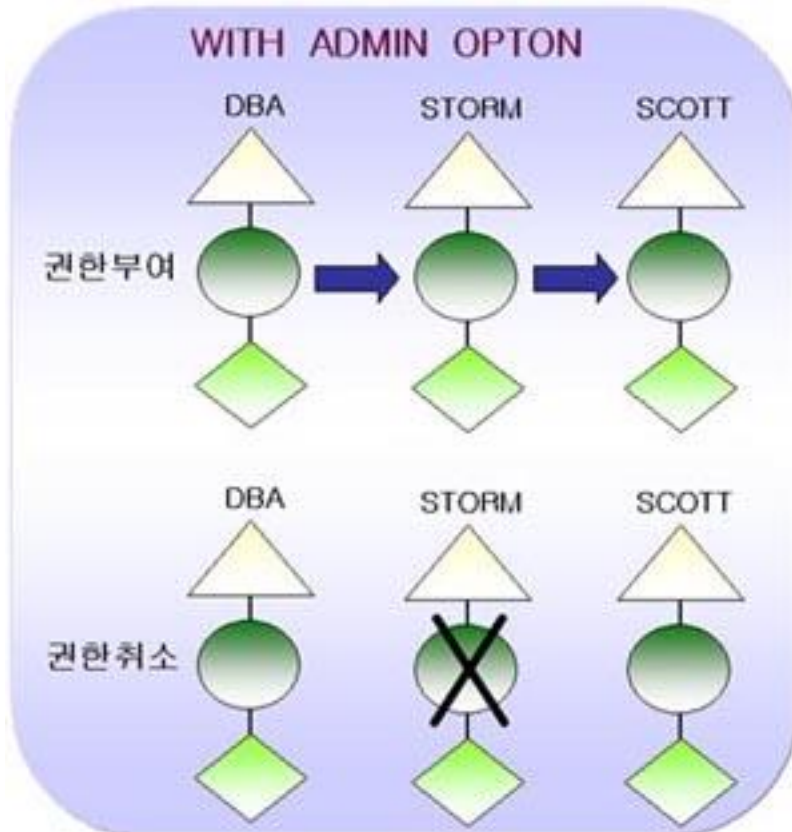
★설명 : scott 사용자에게 부여한 생성, 수정, 삭제 권한을 회수합니다,

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

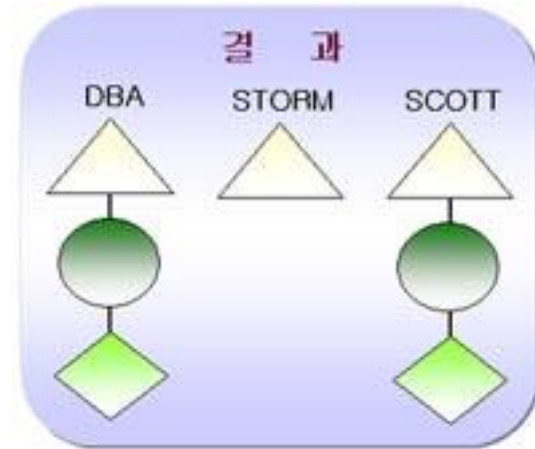
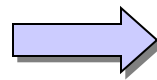
WITH ADMIN OPTION을 사용하여 시스템 권한 취소

WITH ADMIN OPTION을 사용하여 시스템 권한을 부여했어도 시스템 권한을 취소 할 때는 연쇄적으로 취소 되지 않습니다.



시나리오

1. DBA가 STORM에게 WITH ADMIN OPTION을 사용하여 CREATE TABLE 시스템 권한을 부여 합니다.
2. STORM이 테이블을 생성 합니다.
3. STORM이 CREATE TABLE 시스템 권한을 SCOTT에게 부여 합니다.
4. SCOTT가 테이블을 생성 합니다.
5. DBA가 STORM에게 부여한 CREATE TABLE 시스템 권한을 취소 합니다.



결과

- STORM의 테이블은 여전히 존재하지만 새 테이블을 생성할 수 있는 권한은 없습니다.
- SCOTT는 여전히 테이블과 새로운 테이블을 생성 할 수 있는 CREATE TABLE권한을 가지고 있습니다.

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

객체 권한(Object Privileges)

◆ Object Privileges(객체 권한)

객체 권한은 사용자가 소유하고 있는 특정한 객체를 다른 사용자들이 액세스 하거나 조작 할 수 있게 하기 위해서 생성을 합니다.

- 테이블이나 뷰, 시퀀스, 프로시저, 함수, 또는 패키지 중 지정된 한 오브젝트에 특별한 작업을 수행 할 수 있게 합니다.
- Object 소유자는 다른 사용자에게 특정 Object Privileges를 부여 할 수 있습니다.
- PUBLIC으로 권한을 부여하면 회수할 때도 PUBLIC으로 해야 합니다.
- 기본적으로 사용자가 소유한 오브젝트에 대한 모든 권한은 자동적으로 획득 됩니다
- WITH GRANT OPTION 옵션은 룰 에 권한을 부여할 때는 사용할 수 없습니다

Object별 부여할 수 있는 권한

| 객체권한 | 테이블(Table) | 뷰(View) | 시퀀스(Sequence) | 프로시저(Procedure) |
|------------|------------|---------|---------------|-----------------|
| ALTER | ○ | | ○ | |
| DELETE | ○ | ○ | | |
| EXECUTE | | | | ○ |
| INDEX | ○ | | | |
| INSERT | ○ | ○ | | |
| REFERENCES | ○ | | | |
| SELECT | ○ | ○ | ○ | |
| UPDATE | ○ | ○ | | |

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

객체 권한 부여 문법

```
GRANT object_privilege [column]
ON object
TO {user[,user] | role | PUBLIC}
[WITH GRANT OPTION]
```

이 전페이지 표에서 맨 왼쪽에 있는 ALTER, DELETE, EXECUTE.. 등등은 object_privilege란에 오면 되고, 맨 윗줄에 있는 테이블, 뷰, 시퀀스, 프로시저 등등은 ON 다음에 있는 object에 입력하면 됩니다.

◆ 문법 설명

- object_privilege : 부여할 객체 권한의 이름
- object : 객체명
- user, role : 부여할 사용자 이름과 다른 데이터 베이스 역할 이름
- PUBLIC : 오브젝트 권한, 또는 데이터베이스 역할을 모든 사용자에게 부여할 수 있습니다.
- WITH GRANT OPTION : 권한을 부여 받은 사용자도 부여 받은 권한을 다른 사용자 또는 역할로 부여할 수 있게 됩니다.

객체 권한 부여 예제

```
SQL>GRANT SELECT, INSERT
ON emp
TO scott
WITH GRANT OPTION
권한이 부여되었습니다.
```

★설명 : scott 사용자에게 emp테이블을 SELECT, INSERT할 수 있는 권한을 부여했습니다.
scott 사용자도 다른 사용자에게 그 권한을 부여 할 수 있습니다..

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

객체 권한 회수 문법

```
REVOKE {privilege[,privilege,..] | ALL}  
ON object  
FROM {user[,user] | role | PUBLIC}  
[CASCADE CONSTRAINTS]
```

◆ 문법 설명

- 객체 권한의 철회는 그 권한을 부여한 부여자만이 수행 할 수 있습니다.
- CASCADE CONSTRAINTS : 이 명령어의 사용으로 REFERENCES객체 권한에서 사용된 참조 무결성 제한을 같이 삭제 할 수 있습니다.
- WITH GRANT OPTION으로 객체 권한을 부여한 사용자의 객체 권한을 철회하면, 권한을 부여 받은 사용자가 부여한 객체 권한 또한 같이 철회되는 종속철회가 발생합니다.

객체 권한 회수 예제

```
SQL>REVOKE SELECT, INSERT  
ON emp  
FROM scott
```

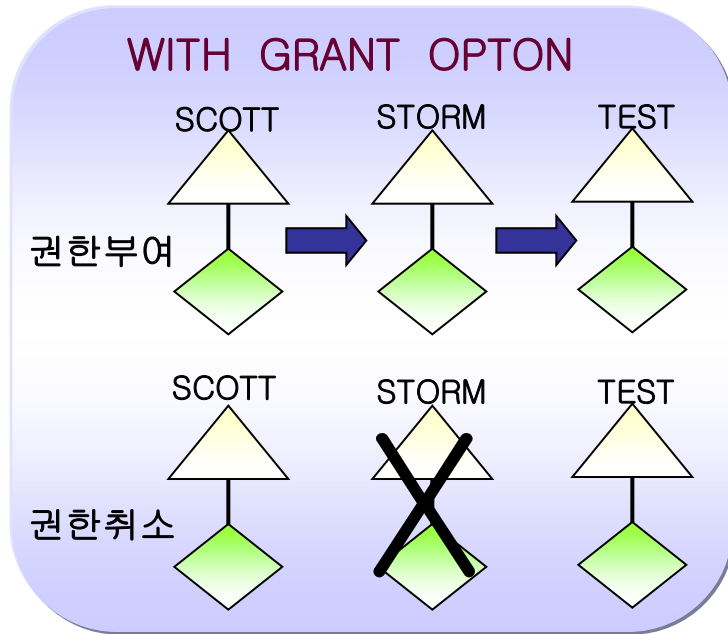
- *설명 : scott 사용자에게부여한 emp테이블에 대한 SELECT, INSERT권한이 회수 됩니다..
만약 scott사용자가 다른 사용자에게 SELECT, INSERT권한을 부여했으면.. 그 권한들도 같이 철회가 됩니다.

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

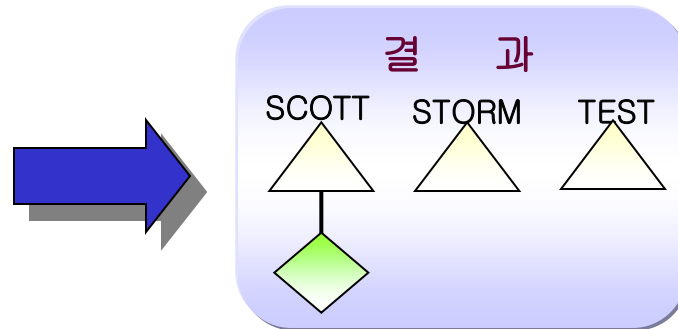
WITH GRANT OPTION을 사용하여 객체 권한 취소

WITH GRANT OPTION을 사용하여 부여한 객체 권한을 취소하면 취소 작업이 연쇄적으로 수행 됩니다.
WITH ADMIN OPTION과 비교해 보세요



시나리오

1. SCOTT가 STORM에게 WITH GRANT OPTION을 사용하여 emp테이블의 SELECT 권한을 부여 합니다.
2. STORM이 emp테이블의 SELECT권한을 TEST에게 부여 합니다.
3. SCOTT가 STORM에게 부여한 emp테이블의 SELECT 권한을 취소 합니다



결과

- SCOTT가 STORM에게 부여한 emp테이블에 대한 SELECT 권한을 취소하면 STORM이 부여한 TEST유저가 emp테이블을 SELECT할 수 있는 권한도 자동으로 취소가 됩니다.

2. ORACLE사용자의 생성과 권한의 설정

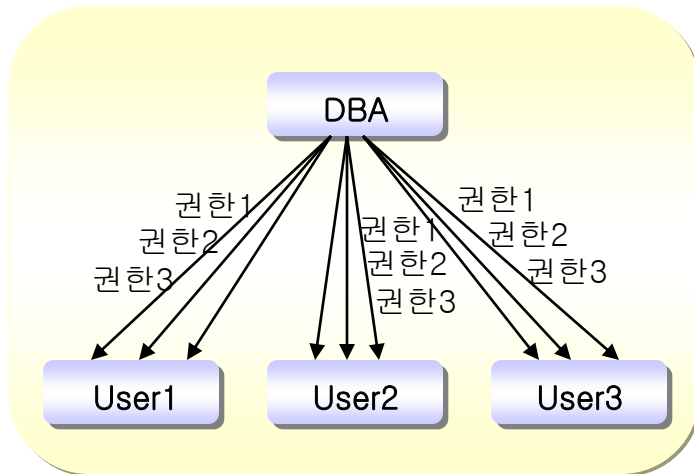
2.4. 권한(Privileges)과 룰(Role)

룰 (Role)

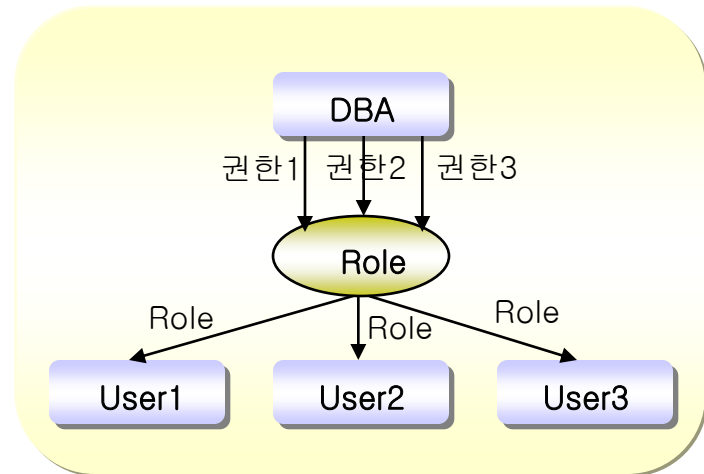
◆ ROLE이란 사용자에게 허가할 수 있는 권한들의 집합 이라고 할 수 있습니다.

- ROLE을 이용하면 권한 부여와 회수를 쉽게 할 수 있습니다.
- ROLE은 Create Role권한을 가진 User에 의해서 생성 됩니다.
- 한 사용자가 여러개의 ROLL을 ACCESS할 수 있고, 여러 사용자에게 같은 ROLE을 부여할 수 있습니다.
- 시스템 권한을 부여하고, 취소할 때와 동일한 명령을 사용하여 사용자에게 부여하고, 취소 합니다.
- 사용자는 ROLE에 ROLE을 부여할 수 있습니다.
- 오라클 데이터베이스를 설치하면 기본적으로 CONNECT, RESOURCE, DBA ROLE이 제공 됩니다.

아래의 그림처럼 DBA가 유저들에게 권한을 부여할 때 일일이 권한 하나하나씩을 지정을 한다면 몹시 불편할 것 입니다. DBA가 USER의 역할에 맞도록 Role을 생성하여서 Role만 유저에게 지정을 한다면 보다 효율적으로 유저들의 권한을 관리 할 수 있습니다.



[일반적으로 권한을 부여하는 방법]



[룰을 생성하여 권한을 부여하는 방법]

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 롤(Role)

롤의 생성 문법

```
CREATE ROLE role_name
```

◆ ROLE의 부여 순서

- ① ROLE의 생성 : CREATE ROLE manager
- ② ROLE에 권한 부여 : GRANT create session, create table TO manager
- ③ ROLE을 사용자 또는 ROLE에게 부여 : GRANT manager TO scott, test;

```
SQL> CREATE ROLE manager
```

-- role을 생성 합니다.

```
SQL> GRANT create session, create table TO manager
```

-- role에 권한을 부여 합니다.

```
SQL> GRANT manager TO scott, test;
```

-- 권한이 부여된 role을 user나 role에 부여 합니다.

롤 관련 데이터 사전

| Dictionary 뷰 | 테이블(Table) |
|---------------------|-----------------------------|
| ROLE_SYS_PRIVS | Role에 부여된 시스템 권한 |
| ROLE_TAB_PRIVS | Role에 부여된 테이블 권한 |
| USER_ROLE_PRIVS | 현재 사용자가 ACCESS할 수 있는 ROLE |
| USER_TAB_PRIVS_MADE | 현재 사용자의 객체에 부여한 객체 권한 |
| USER_TAB_PRIVS_RECD | 현재 사용자의 객체에 부여된 객체 권한 |
| USER_COL_PRIVS_MADE | 현재 사용자 객체의 특정 컬럼에 부여한 객체 권한 |
| USER_COL_PRIVS_RECD | 현재 사용자 객체의 특정 컬럼에 부여된 객체 권한 |

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

오라클 데이터베이스를 생성하면 기본적으로 생성되는 룰(Role)

오라클 데이터베이스를 생성하면 기본적으로 몇 가지의 Role이 생성 됩니다.
DBA_ROLES 데이터 사전을 통하여 미리 정의된 Role을 조회 할 수 있습니다.

```
SQL>SELECT * FROM DBA_ROLES;
```

| ROLE | PASSWORD |
|-------------------|----------|
| CONNECT | NO |
| RESOURCE | NO |
| DBA | NO |
| EXP_FULL_DATABASE | NO |
| | |

이 외에도 많이 ROLE이 존재합니다. 그 중에서 가장 많이 사용하는 3가지만 설명 하겠습니다.

◆ DBA Role

- 모든 시스템 권한이 부여된 Role입니다.
- DBA Role은 데이터베이스 관리자에게만 부여해야 합니다.

2. ORACLE사용자의 생성과 권한의 설정

2.4. 권한(Privileges)과 룰(Role)

◆ RESOURCE

- Store Procedure나 Trigger와 같은 PL/SQL을 사용할 수 있는 권한 들로 이루어져 있습니다.
- PL/SQL을 사용하려면 RESOURCE Role을 부여해야 합니다.
- 유저를 생성하면 일반적으로 CONNECT, RESOURCE 룰 을 부여 합니다.

```
SQL>SELECT grantee, privilege
      FROM DBA_SYS_PRIVS
      WHERE grantee = 'RESOURCE';
```

◆ CONNECT

- 오라클에 접속 할 수 있는 세션 생성 및 테이블을 생성하거나 조회 할 수 있는 가장 일반적인 권한들로 이루어져 있습니다.
- CONNECT Role이 없으면 유저를 생성하고서도 Oracle에 접속 할 수가 없습니다.
- 아래의 명령어로 CONNECT Role이 어떤 권한으로 이루어져 있는지 확인 할 수 있습니다.

```
SQL>SELECT grantee, privilege
      FROM DBA_SYS_PRIVS
      WHERE grantee = 'CONNECT';
```

| GRANTEE | PRIVILEGE |
|---------|----------------------|
| CONNECT | ALTER SESSION |
| CONNECT | CREATE CLUSTER |
| CONNECT | CREATE DATABASE LINK |
| CONNECT | CREATE SEQUENCE |
| CONNECT | CREATE SESSION |

3. 테이블의 생성과 수정 그리고 삭제

3.1. 테이블의 생성

테이블이란 ?

1. 테이블은 오라클 데이터베이스의 기본적인 데이터 저장 단위입니다.
2. 데이터베이스 테이블은 사용자가 접근 가능한 모든 데이터를 보유하며 레코드와 컬럼으로 구성 됩니다.
3. 테이블은 시스템내에서 독립적으로 사용되길 원하는 엔티티를 표현할 수 있습니다.
예를 들면, 회사에서의 고용자나 제품에 대한 주문은 테이블로 표현 가능합니다.
4. 테이블은 두 엔티티간의 관계를 표현할 수 있습니다.
즉 테이블은 고용자와 그들의 작업 숙련도 혹은 제품과 주문과의 관계를 표현하는데 사용될 수 있습니다.
테이블내에 있는 외래 키(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 고려 할 점

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

3. 테이블의 생성과 수정 그리고 삭제

3.1. 테이블의 생성

테이블 생성 문법

```
CREATE TABLE [schema.]table_name
( column datatype
  [, column datatype ...]
)
[TABLESPACE tablespace ]
[PCTFREE integer ]
[PCTUSED integer ]
[INITRANS integer ]
[MAXTRANS integer ]
[STORAGE storage-clause ]
[LOGGING | NOLOGGING ]
[CACHE | NOCACHE ] ;
```

- schema : 테이블의 소유자
- table_name: 테이블 이름
- column: 컬럼의 이름
- datatype: 컬럼의 데이터 유형
- TABLESPACE: 테이블이 데이터를 저장 할 테이블스페이스를 지정 합니다.
- PCTFREE : 블록내에 이미 존재하고 있는 Row에 Update가 가능하도록 예약시켜 놓는 블록의 퍼센트 값을 지정 합니다.
- PCTUSED : 테이블 데이터가 저장될 블록의 행 데이터 부분의 크기를 퍼센트지로 지정 합니다.
PCTFREE에 의해 지정된 크기만큼 Block이 차면 PCTUSED 값보다 작아져야 새로운 행 삽입이 가능 합니다.

- INITRANS : 하나의 데이터 블록에 지정될 초기 트랜잭션의 값을 지정합니다. (기본값은 1)
- MAXTRANS: 하나의 데이터 블록에 지정될 수 있는 트랜잭션 최대 수를 지정 합니다. (기본값은 255)
- STORAGE: 익스텐트 스토리지에 대한 값을 지정 합니다.
- LOGGING: 테이블에 대해 이후의 모든 작업이 리두 로그 파일 내에 기록 되도록 지정합니다. (default)
- NOLOGGING: 리두 로그 파일에 테이블의 생성과 특정 유형의 데이터 로드를 기록하지 않도록 지정 합니다.
- CACHE : 전체 테이블 스캔(full table scan)이 수행될 때 읽어 들인 블록이 버퍼 캐쉬 내의 LRU 리스트의 가장 최근에 사용된 것의 자리에 위치 하도록 지정 합니다.
- NOCACHE : 전체 테이블 스캔(full table scan)이 수행될때 읽어 들인 블록이 버퍼 캐쉬 내의 LRU 리스트의 가장 최근에 사용 되지 않은 것의 자리에 위치하도록 지정 합니다.

* PCTFREE, PCTUSED에 대한 자세한 강좌는 오라클 어드민 강좌를 참고해 주세요.

3. 테이블의 생성과 수정 그리고 삭제

3.1. 테이블의 생성

테이블 생성 예제

◆ DEPT2 테이블 생성 예제 입니다.

```
SQL>CREATE TABLE DEPT2(  
    DEPTNO    NUMBER CONSTRAINT dept_pk_deptno PRIMARY KEY,  
    - (컬럼)   (데이터타입)                (제약조건)  
    DNAME     VARCHAR2(40),  
    LOC       VARCHAR2(50))  
    PCTFREE 20  
    PCTUSED 50 ;
```

테이블이 생성되었습니다.

테이블 생성시 주의사항

- 테이블 이름을 지정하고 각 컬럼들은 괄호 "()" 로 묶어 지정합니다.
- 컬럼뒤에 데이터 타입은 꼭 지정되어야 합니다.
- 각 컬럼들은 콤마", "로 구분되고, 항상 끝은 세미콜론"; " 으로 끝납니다.
- 한 테이블 안에서 컬럼 이름은 같을 수 없으며 다른 테이블에서의 컬럼 이름과는 같을 수 있습니다.

유저가 소유한 모든 테이블 조회

◆ USER_TABLES 데이터사전을 조회 하면 유저가 소유한 테이블을 확인 할 수 있습니다.

```
SQL>SELECT table_name FROM USER_TABLES;  
TABLE_NAME
```

```
-----  
BONUS  
CRETABLE
```

3. 테이블의 생성과 수정 그리고 삭제

3.2. 테이블의 제약조건

제약조건이란 ?

제약조건이란 테이블에 부적절한 자료가 입력되는 것을 방지하기 위해서 여러 가지 규칙을 적용해 놓는 거라 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.

| 제 약 조 건 | 설 명 |
|-----------------|--|
| PRIMARY KEY(PK) | 유일하게 테이블의 각행을 식별(NOT NULL과 UNIQUE조건을 만족) |
| FOREIGN KEY(FK) | 열과 참조된 열 사이의 외래키 관계를 적용하고 설정합니다. |
| UNIQUE key(UK) | 테이블의 모든 행을 유일하게 하는 값을 가진 열(NULL을 허용) |
| NOT NULL(NN) | 열은 NULL값을 포함할 수 없습니다. |
| CHECK(CK) | 참이어야 하는 조건을 지정함(대부분 업무 규칙을 설정) |

3. 테이블의 생성과 수정 그리고 삭제

3.2. 테이블의 제약조건

NOT NULL 조건 : 컬럼을 필수 필드화 시킬 때 사용합니다.

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

위와 같이 테이블을 생성하면 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 확인 할 수 있습니다.

```
SQL> SELECT CONSTRAINT_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'EMP' ;  
CONSTRAINT_NAME
```

```
emp_nn_ename          → 이런 식으로 제약조건의 이름을 확인할 수 있습니다.
```

UNIQUE 조건

UNIQUE 조건 : 데이터의 유일성을 보장 합니다. (중복되는 데이터가 존재할수 없습니다.)
유니크 조건을 생성하면 자동으로 index가 생성됩니다.

```
SQL> ALTER TABLE emp ADD CONSTRAINT emp_uk_deptno UNIQUE (deptno) ;  
테이블이 변경되었습니다.
```

- 위와 같이 제약조건을 생성하면 deptno 컬럼에 중복된 데이터가 들어갈 수 없습니다.

-- 제약 조건의 삭제

```
SQL> ALTER TABLE emp DROP CONSTRAINT emp_uk_deptno ;  
테이블이 변경되었습니다.
```

3. 테이블의 생성과 수정 그리고 삭제

3.2. 테이블의 제약조건

CHECK 조건 : 컬럼의 값을 어떤 특정 범위로 제한할 수 있습니다.

```
SQL>ALTER TABLE emp ADD CONSTRAINT emp_ck_comm CHECK (comm >= 10 AND comm <= 100000) ;  
테이블이 변경되었습니다.
```

- comm컬럼은 체크조건에서 제한을 하고 있으므로 1에서 100까지의 값만을 가질 수 있습니다.
또 체크 조건에서는 IN 연산자를 사용할 수 있습니다.

```
SQL>ALTER TABLE emp DROP CONSTRAINT emp_ck_comm ;  
테이블이 변경되었습니다.
```

-- 제약 조건의 삭제

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

-- IN 연산자를 이용해서 CHECK 제약 조건을 생성

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

DEFAULT(컬럼 기본값) 지정

데이터 입력시에 입력을 하지 않아도 지정된 값이 디폴트로 입력이 됩니다.

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

위와 같이 디폴트 값을 설정하면 hiredate 컬럼에 INSERT를 하지 않아도 오늘 날짜가 들어갑니다.

3. 테이블의 생성과 수정 그리고 삭제

3.2. 테이블의 제약조건

PRIMARY KEY(기본키) 지정

PRIMARY KEY : Primary Key(기본키)는 UNIQUE 와 NOT NULL의 결합과 같습니다.

- 기본키는 그 데이터 행을 대표하는 컬럼으로서의 역할을 수행하여 다른 테이블에서 외래키들이 참조할 수 있는 키로서의 자격을 가지고 있습니다. 이를 참조 무결성이라 합니다.
- UNIQUE 조건과 마찬가지로 기본키를 정의하면 자동으로 인덱스를 생성하며 그 이름은 기본 키 제약 조건의 이름과 같습니다.

```
SQL>CREATE TABLE emp(  
    empno NUMBER CONSTRAINT emp_pk_empno PRIMARY KEY ) ;
```

위와 같이 제약조건을 설정 하면 empno 컬럼에 UNIQUE 제약조건과 NOT NULL제약조건을 가지게 됩니다.

FOREIGN KEY(외래 키)지정

FOREIGN KEY(외래 키)지정 : 기본키를 참조하는 컬럼 또는 컬럼들의 집합입니다.

- 외래키를 가지는 컬럼의 데이터 형은 외래키가 참조하는 기본키의 컬럼과 데이터형이 일치해야 합니다. 이를 어기면 참조무결성 제약에 의해 테이블을 생성 할 수 없습니다.
- 외래키에 의해 참조되고 있는 기본 키는 삭제 할 수 없습니다.
- ON DELETE CASCADE 연산자와 함께 정의된 외래키의 데이터는 그 기본키가 삭제 될 때 같이 삭제됩니다.

```
SQL>ALTER TABLE emp ADD CONSTRAINT emp_fk_deptno  
    FOREIGN KEY (deptno) REFERENCES dept(deptno) [on delete cascade] <- []는 필수 아님  
테이블이 변경되었습니다.
```

위와 같이 제약조건을 생성하면 emp 테이블의 deptno 컬럼은 dept 테이블에 deptno 컬럼을 참조하는 외래키를 가지게 됩니다.

3. 테이블의 생성과 수정 그리고 삭제

3.3. 오라클 데이터 타입

오라클 테이블을 생성 할 때 각 컬럼에 지정 할 수 있는 데이터 타입들 입니다.

| DATA TYPE | 설 명 |
|-------------|--|
| VARCHAR2(n) | 가변 길이 문자 데이터(1~4000byte) |
| CHAR(n) | 고정 길이 문자 데이터(1~2000byte) |
| NUMBER(p,s) | 전체 p자리 중 소수점 이하 s자리(p:1~38, s:-84~127) |
| DATE | 7Byte(BC 4712년 1월 1일부터 AD 9999년 12월 31일) |
| LONG | 가변 길이 문자 데이터(1~2Gbyte) |
| CLOB | 단일 바이트 가변 길이 문자 데이터(1~4Gbyte) |
| RAW(n) | n Byte의 원시 이진 데이터(1~2000) |
| LONG RAW | 가변 길이 원시 이진 데이터(1~2Gbyte) |
| BLOB | 가변 길이 이진 데이터(1~4Gbyte) |
| BFILE | 가변 길이 외부 파일에 저장된 이진 데이터(1~4Gbyte) |

숫자 데이터(Numeric Data)

- 오라클 데이터베이스에서 숫자는 항상 가변 길이 데이터로 저장되며 유효 자릿수 38자리까지 저장할 수 있습니다.

3. 테이블의 생성과 수정 그리고 삭제

3.3. 오라클 데이터 타입

문자 데이터(Character Data)

- 문자 데이터는 데이터베이스에 고정 길이, 또는 가변 길이 문자열로 저장될 수 있습니다.
- CHAR와 NCHAR같은 고정 길이 문자 데이터 유형은 고정 길이까지 공백으로 채워서 저장합니다.
- NCHAR는 고정 폭(fixed-width), 또는 가변 폭(variable-width) character set의 저장을 가능하게 하는 NLS 데이터 유형입니다.
최대 크기는 한 문자를 저장하는데 필요한 바이트 수에 따라 결정되며 한 행 당 2000 바이트가 상한입니다.
기본값은 character set에 따라 1 문자, 또는 1 바이트입니다.
- 가변 길이 문자 데이터 유형은 실제 컬럼 값을 저장하는데 필요한 바이트만큼만을 사용하며 각 행에 따라 그 크기가 다양합니다. VARCHAR2와 NVARCHAR2 가 있습니다.

날짜(DATE) 데이터 유형

- 오라클 서버는 날짜를 7 바이트, 고정 길이 필드(field)로 저장합니다.
- 오라클 DATE는 항상 시간을 포함합니다.

RAW 데이터 유형

- 크기가 적은 이진 데이터의 저장에 사용 합니다.

3. 테이블의 생성과 수정 그리고 삭제

3.3. 오라클 데이터 타입

크기가 큰 오브젝트를 저장하기 위한 데이터 유형

- ① 크기가 큰 오브젝트를 저장하기 위한 데이터 유형에는 LONG과 LONG RAW, LOB데이터 유형이 있습니다.
- ② LONG데이터 유형은 2GB의 문자열 데이터를 저장 할 수 있습니다.
- ③ 오라클은 LOB을 저장하기 위한 여섯 가지 데이터 유형을 제공합니다.
 - 큰 고정 폭(fixed-width) 문자 데이터를 위한 CLOB과 LONG
 - 큰 고정 폭 국가 character set 데이터를 위한 NCLOB
 - 구조화되지 않은 데이터를 저장하기 위한 BLOB과 LONG RAW
 - 구조화되지 않은 데이터를 운영 체제 파일에 저장하기 위한 BFILE

LONG과 LOB 데이터 유형 비교

| LONG, LONG RAW | LOB |
|-----------------------|---------------------|
| 테이블에 컬럼 하나만 생성 할 수 있음 | 테이블에 여러개의 컬럼 생성이 가능 |
| 2GB | 4GB |
| SELECT결과로 데이터를 리턴 | SELECT결과로 위치를 리턴 |
| 데이터를 직접 저장 | 데이터를 직접 또는 간접 저장 |
| 오브젝트 유형을 지원하지 않음 | 오브젝트 유형 지원 |

3. 테이블의 생성과 수정 그리고 삭제

3.4. 테이블의 관리

테이블 컬럼의 관리

테이블의 컬럼은 ADD, MODIFY, DROP 연산자를 통해서 관리 할 수 있습니다.

ADD 연산자

ADD 연산자 : 테이블에 새로운 컬럼을 추가 할 때 사용 합니다.

```
SQL>ALTER TABLE emp ADD (addr VARCHAR2(50));
```

VARCHAR2의 데이터 형을 가지는 addr 컬럼이 emp 테이블에 추가 됩니다.

MODIFY 연산자

MODIFY 연산자 : 테이블의 컬럼을 수정 하거나 NOT NULL 컬럼으로 변경 할 수 있습니다.

```
SQL>ALTER TABLE emp MODIFY (ename VARCHAR2(50) NOT NULL) ;
```

-> ename 컬럼이 VARCHAR2 50자리로 수정 됩니다.

※ 컬럼이 이미 데이터를 가지고 있을 경우 다른 데이터형으로 변경이 불가능합니다.

DROP 연산자

◆ 컬럼의 삭제 예제

```
SQL>ALTER TABLE table_name DROP COLUMN column_name
```

-- 컬럼의 삭제는 오라클 8i 버전 부터 지원을 합니다.

◆ 제약 조건의 삭제 예제

-- CASCADE 연산자와 함께 사용하면 외래키에 의해 참조되는 기본키도 삭제될 수 있습니다.

```
SQL>ALTER TABLE emp DROP CONSTRAINT emp_pk_empno CASCADE;
```

3. 테이블의 생성과 수정 그리고 삭제

3.4. 테이블의 관리

테이블 정보의 관리

기존 테이블의 복사

- 기존 테이블을 부분, 또는 완전히 복사할 때에 서브쿼리를 가진 CREATE TABLE 명령어를 사용해서 쉽게 테이블을 복사 할 수 있습니다.
- 하지만 제약 조건, 트리거, 그리고 테이블 권한은 새로운 테이블로 복사되지 않습니다.
- 제약조건은 NOT NULL 제약조건만 복사 됩니다.

[문법]

```
CREATE TABLE [schema.]table_name  
[ LOGGING | NOLOGGING ]  
[ ... ]  
AS  
subquery
```

[예제]

```
SQL>CREATE TABLE emp2  
AS  
SELECT * FROM emp;  
테이블이 생성되었습니다.
```

테이블의 테이블스페이스 변경

Oracle8i이전 버전에서는 export를 해서 다시 import를 해야지만 테이블스페이스를 변경 할 수 있었지만
오라클 8i부터는 ALTER TABLE ~ MOVE TABLESPACE 명령어로 쉽게 테이블의 테이블스페이스를 변경 할 수 있습니다.

```
ALTER TABLE table_name MOVE TABLESPACE tablespace_name;
```

```
SQL>ALTER TABLE emp MOVE TABLESPACE test;  
테이블이 변경되었습니다  
→emp 테이블의 테이블스페이스를 test로 변경 했습니다.
```

3. 테이블의 생성과 수정 그리고 삭제

3.4. 테이블의 관리

테이블 정보의 변경

- 보통 테이블의 정보를 변경하는 이유는 스토리지 파라미터와 블록 할용파라미터를 변경하기 위해서 사용 합니다.
- 테이블 정보의 변경시 INITIAL의 값은 변경 할 수 없습니다.

```
ALTER TABLE [schema.]table_name  
[ STORAGE storage-clause ]  
[ PCTFREE integer ]  
[ PCTUSED integer ]  
[ INITRANS integer ]  
[ MAXTRANS integer ]
```

STORAGE-CLAUSE에 들어올 수 있는 스토리지 파라미터를 정리하면 아래와 같습니다.

- NEXT
다음 번 생성될 익스텐트의 크기를 Byte단위로 지정합니다. 이후의 익스텐트 크기는 PCTINCREASE만큼씩 증가 됩니다.
- PCTINCREASE
마지막 생성된 익스텐트의 바로 다음에 생성될 익스텐트의 증가율을 퍼센트지로 지정 합니다.
- MINEXTENTS
최초 생성되는 익스텐트의 수를 지정 합니다.
- MAXEXTENTS
생성될 수 있는 최대 익스텐트의 수를 지정 합니다.

기타 블록관련 파라미터는 테이블의 생성 강좌를 참고하세요..

3. 테이블의 생성과 수정 그리고 삭제

3.4. 테이블의 관리

테이블의 TRUNCATE

- 테이블을 Truncate하면 테이블의 모든 행이 삭제되고 사용된 공간이 해제 됩니다.
- TRUNCATE TABLE은 DDL명령이므로 롤백 데이터가 생성되지 않습니다.
DELETE명령으로 데이터를 지우면 롤백명령어로 복구 할 수 있지만 TRUNCATE로 데이터를 삭제하면 롤백을 할 수가 없습니다.
- 행의 인덱스도 같이 잘려 나갑니다.
- 외래키가 참조중인 테이블은 TRUNCATE 할 수 없습니다.
- TRUNCATE명령을 사용하면 삭제 트리거가 실행되지 않습니다.

```
TRUNCATE TABLE [schema.]table_name ;
```

DROP TABLE (테이블을 삭제할 때 사용)

```
DROP TABLE [schema.]table_name [CASCADE CONSTRAINTS] ;
```

```
SQL>DROP TABLE emp ;
```

```
SQL>DROP TABLE emp CASCADE CONSTRAINT;
```

CASCADE CONSTRAINT

➔ 외래키에 의해 참조되는 기본키를 포함한 테이블일 경우 기본키를 참조하던 외래 키 조건도 같이 삭제 됩니다.

4. 데이터 조작용(DML)

4.1. 데이터의 삽입, 수정, 삭제

INSERT

INSERT명령어는 테이블 안에 데이터를 삽입하는 역할을 합니다. .

```
INSERT INTO table_name(column1, column2,...)
VALUES (데이터, '데이터',...);
```

- 실제 데이터는 VALUES 괄호()안에 입력하고 문자열은 단일 따옴표(' ')로 둘러쌉니다.
- 각각의 데이터 구분은 ","로 합니다.
- 테이블 이름 옆에 ()생략시에는 모든 컬럼을 VALUES()안에 입력 시킵니다

◆ 모든 데이터를 입력할 경우

```
SQL>INSERT INTO EMP VALUES(7369, 'SMITH', 'CLERK', 7902, TO_DATE('80/12/17'), 800, NULL, 20);
```

◆ 원하는 데이터만 입력할 경우

```
SQL>INSERT INTO DEPT (DEPTNO, DNAME) VALUES(10, 'ACCOUNTING' );
```

◆ SELECT 문장을 이용한 INSERT

```
INSERT INTO table_name(column1, column2,...)
SELECT column1, column2,...
FROM table_name
WHERE 조건;
```

```
SQL>INSERT INTO DEPT
      SELECT * FROM SCOTT.DEPT ;
```

4. 데이터 조작용(DML)

4.1. 데이터의 삽입, 수정, 삭제

UPDATE

UPDATE 명령어는 테이블 안의 데이터를 수정 합니다.

```
UPDATE table_name  
SET column1 = 값(고칠내용), column2 = 값, ...  
WHERE 조건
```

SQL>UPDATE EMP SET DEPTNO = 30 WHERE EMPNO = 7902 ; → 사원번호가 7902번인 사람의 부서 번호가 30번으로 수정됨

SQL>UPDATE EMP SET SAL = SAL * 1.1 WHERE DEPTNO = 20 ; → 20부서의 사원들의 급여가 10% 인상됨

SQL>UPDATE EMP SET HIREDATE = SYSDATE → 모든 사원의 입사일이 오늘로 수정됨

DELETE

DELETE 명령어는 테이블 안의 데이터를 삭제 합니다.

```
DELETE FROM table_name WHERE 조건 ;
```

SQL>DELETE FROM EMP WHERE EMPNO = 7902 ; → 사원번호가 7902번인 사람의 데이터가 삭제 되었습니다.

SQL>DELETE FROM EMP WHERE SAL < (SELECT AVG(SAL) FROM EMP) ; → 평균급여보다 적게 받는 사원 삭제

SQL>DELETE FROM EMP ; → 테이블의 모든 행이 삭제 됩니다.

4. 데이터 조작용(DML)

4.2. SELECT문 및 연산자

SELECT 문

SELECT문은 데이터베이스로부터 저장되어 있는 데이터를 검색하는데 사용합니다.

```
SELECT      [DISTINCT] {*, column [alias], ...}  
FROM        table_name  
[WHERE      condition]  
[ORDER BY {column, expression} [ASC | DESC]];
```

- DISTINCT : 중복되는 행을 제거하는 옵션입니다.
- * : 테이블의 모든 column을 출력 합니다.
- alias : 해당 column에 대해서 다른 이름을 부여할 때 사용합니다.
- table_name : 질의 대상 테이블명
- WHERE : 조건을 만족하는 행들만 검색
- condition : column, 표현식, 상수 및 비교 연산자
- ORDER BY : 질의 결과 정렬을 위한 옵션(ASC:오름차순(Default),DESC내림차순)

SQL문의 작성 방법

- SQL 문장은 대소문자를 구별하지 않습니다.
- SQL 문장은 한 줄 또는 여러 줄에 입력될 수 있습니다.
- 일반적으로 키워드는 대문자로 입력합니다.
다른 모든 단어, 즉 테이블 이름, 열 이름은 소문자로 입력합니다.(권장)
- 가장 최근의 명령어가 1개가 SQL buffer에 저장됩니다.
- SQL문 마지막 절의 끝에 ";"를 기술하여 명령의 끝을 표시 합니다.

[예제]

```
SQL>SELECT empno 사번, ename 성명 FROM emp WHERE deptno = 10
```

4. 데이터 조작용어(DML)

4.2. SELECT문 및 연산자

WHERE절에 올 수 있는 연산자들.

| 연산자 | 설 명 |
|---------------------|---|
| BETWEEN a AND b | a와b사이의 데이터를 출력 합니다.(a, b값 포함) |
| IN (list) | list의 값 중 어느 하나와 일치하는 데이터를 출력 합니다. |
| LIKE | 문자 형태로 일치하는 데이터를 출력 합니다.(%, _사용) |
| IS NULL | NULL값을 가진 데이터를 출력 합니다. |
| NOT BETWEEN a AND b | a와b사이에 있지않은 데이터를 출력 합니다.(a, b값 포함하지 않음) |
| NOT IN (list) | list의 값과 일치하지 않는 데이터를 출력 합니다. |
| NOT LIKE | 문자 형태와 일치하지 않는 데이터를 출력 합니다. |
| IS NOT NULL | NULL값을 갖지 않는 데이터를 출력 합니다. |

IN 과 NOT IN 연산자

◆ IN 연산자

```
SQL> SELECT empno, ename FROM emp WHERE empno IN (7900, 7934) ;
```

--> 사번이 7900, 7934번인 사원의 사번과 성명 출력

◆ NOT IN 연산자

```
SQL> SELECT empno, ename FROM emp WHERE empno NOT IN (7900, 7934);
```

--> 사번이 7900, 7934번이 아닌 사원의 사번과 성명 출력

4. 데이터 조작용(DML)

4.2. SELECT문 및 연산자


BETWEEN 연산자

```
SQL> SELECT empno, ename FROM emp WHERE sal BETWEEN 3000 AND 5000 ;
```

→ 급여가 3000에서 5000사이인 사원만 보여줍니다.

BETWEEN 연산자(AND를 이용해 두 조건을 결합한 검색과 같은 결과값을 보여줍니다.)

LIKE 연산자

- 검색 STRING 값에 대한 와일드 카드 검색을 위해서 LIKE 연산자를 사용 합니다.
- % : 여러개의 문자열을 나타내는 와일드 카드
- _ : 단 하나의 문자를 나타내는 와일드 카드
- ESCAPE : 와일드 카드 문자를 일반문자 처럼 사용하고 싶은 경우에 사용합니다.
 WHERE name LIKE '%aW_y%' ESCAPE 'W' ;
- LIKE 연산자는 대소문자를 구분합니다.
- Upper() 함수를 이용해 대소문자 구분 없이 출력 할 수 있습니다.

[예제]

```
SQL> SELECT empno, ename FROM emp WHERE UPPER(ename) like '%K%';
```

'K' 문자가 들어있는 사원 정보를 보여줍니다.

upper()라는 함수는 k가 들어가 있는 것도 대문자 'K'로 인식하기 때문에 데이터들을 보여줍니다.

※ '_'를 이용한 LIKE 검색

```
SQL> SELECT empno, ename FROM emp WHERE UPPER(ename) like '_I%'
```

'_'는 한 문자를 나타냅니다. 'I' 문자가 두 번째 문자에 위치한 사원들의 정보를 보여줍니다.

4. 데이터 조작용(DML)

4.3. 예명(Alias)

테이블 예명(Alias)

◆ 테이블 예명(Alias)

- 테이블 alias로 column을 단순, 명확히 할 수 있습니다.
- 현재의 SELECT 문장에 대해서만 유효합니다.
- 테이블 alias는 길이가 30자 까지 가능하나 짧을수록 더욱 좋습니다.
- 테이블 alias는 의미가 있어야 합니다
- FROM절에 테이블 alias설정시 해당 테이블 alias는 SELECT문장에서 테이블 이름 대신에 사용해야 합니다.

```
SQL> SELECT a.dname, b.cnt
       FROM dept a, (SELECT deptno, COUNT(empno) cnt FROM emp GROUP BY deptno) b
       WHERE a.deptno = b.deptno
             AND b.cnt > 3
```

| DNAME | CNT |
|----------|-----|
| RESEARCH | 6 |
| SALES | 6 |

사원수가 3명이 넘는 부서의 부서명과 사원수를 보여줍니다.

위 쿼리에선 총 3개의 Alias가 사용됐습니다.

첫 번째로 DEPT테이블을 a라는 예명으로

두 번째로 부서의 사원수인 COUNT(empno)를 cnt라는 예명으로

세 번째로 부서별 사원수를 가져오는 쿼리를 b라는 예명을 주었습니다.

위 예제와 같이 예명은 컬럼에만 주는 것이 아니라, 쿼리 문 및 테이블에도 사용할 수 있습니다.

4. 데이터 조작용어(DML)

4.3. 조인 (Join)

조인(Join) ?

◆ 조인(Join) 이란

- 둘이상의 테이블을 연결하여 데이터를 검색하는 방법 입니다.
- 보통 둘 이상의 행들의 공통된 값 Primary Key 및 Foreign Key 값을 사용하여 조인 합니다.
- 그러므로 두 개의 테이블을 SELECT문장 안에서 조인하려면 적어도 하나의 컬럼이 그 두 테이블 사이에서 공유 되어야 합니다

◆ 조인 방법

- Equi join (동등 조인, 내부조인)
- Outer join
- Non-equi join
- Self join

◆ Cartesian Product (카티션 곱)

검색하고자 했던 데이터뿐 아니라 조인에 사용된 테이블들의 모든 데이터가 Retrun되는 현상

◆ Cartesian product는 다음과 같은 경우에 발생합니다.

- 조인 조건을 정의하지 않았을 경우
- 조인 조건이 잘못된 경우
- 첫 번째 테이블의 모든 행들이 두 번째 테이블의 모든 행과 조인이 되는 경우
- 테이블의 개수가 N이라면 Cartesian product를 피하기 위해서는 적어도 N-1개의 등가 조건을 SELECT 문안에 포함시켜서 다른 테이블 안에 있는 각 테이블의 컬럼이 적어도 한번은 참조되도록 해야 합니다.

4. 데이터 조작용(DML)

4.3. 조인 (Join)

Equi Join

☞ Equi Join

- 조건절 Equality Condition(=)에 의하여 조인이 이루어 집니다.
- Equi join의 성능을 높이기 위해서는 Index 기능을 사용하는 것이 좋습니다.

```
SQL>SELECT  e.ename, d.dname      ➔ WHERE 절에 조인 조건을 작성하고 column명 앞에 테이블명을 적습니다.  
      FROM    emp e , dept d  
      WHERE e.deptno = d.deptno;
```

Non-Equijoin

☞ Non-Equijoin

- Non-equijoin은 테이블의 어떤 column도 join할 테이블의 column에 일치하지 않을 때 사용하고 조인조건은 동등(=)이외의 연산자를 갖습니다. (BETWEEN AND, IS NULL, IS NOT NULL, IN, NOT IN)

```
SQL>SELECT e.ename, d.dname  
      FROM emp e, dept d  
      WHERE e.sal BETWEEN 3000 AND 4000;
```

Self Join

☞ Self Join

- Equi Join과 같으나 하나의 테이블에서 조인이 일어나는 것이 다릅니다.
- 같은 테이블에 대해 두 개의 alias를 작성함으로 FROM절에 두 개의 테이블을 사용 하는 것과 같이 합니다.

```
SQL> SELECT concat(a.ename,' ') || ' : W' || b.sal  급여  
      FROM emp a, emp b  
      WHERE a.empno = b.empno
```

4. 데이터 조작용(DML)

4.3. 조인 (Join)

Out(외부) Join

☞ Out(외부) Join

- equijoin 문장들의 한가지 제약점은 그것들이 조인을 생성하려 하는 두 개의 테이블의 두 개 컬럼에서 공통된 값이 없다면 테이블로부터 데이터를 Return하지 않는다는 것입니다.
- 정상적으로 조인 조건을 만족하지 못하는 행들을 보기위해 outer join을 사용합니다.
Outer join 연산자 "(+)"입니다.
- 조인시킬 값이 없는 조인측에 "(+)"를 위치 시킵니다.
- Outer join 연산자는 표현식의 한 편에만 올 수 있습니다.
- Outer join은 IN 연산자를 사용할 수 없고 OR 연산자에 의해 다른 하나의 조건에 연결될 수 없습니다.

-ANSI JOIN 논리적인 수행순서

1.SELECT 2.FROM 3.OUTER JOIN 4.ON 5.WHERE

```
select distinct(a.deptno),b.deptno from emp a right outer join dept b on a.deptno=b.deptno;
```

```
select distinct(a.deptno),b.deptno from dept b left outer join emp a on a.deptno=b.deptno;
```

예제1) 일반 조인의 경우

```
SQL> SELECT DISTINCT(a.deptno), b.deptno
      FROM emp a, dept b
      WHERE a.deptno = b.deptno
```

| DEPTNO | DEPTNO |
|--------|--------|
| 10 | 10 |
| 20 | 20 |
| 30 | 30 |

예제2) out join을 했을 경우

```
SQL> SELECT DISTINCT(a.deptno), b.deptno
      FROM emp a, dept b
      WHERE a.deptno(+) = b.deptno
```

| DEPTNO | DEPTNO |
|--------|--------|
| 10 | 10 |
| 20 | 20 |
| 30 | 30 |
| | 40 |

4. 데이터 조작용(DML)

4.4. 트랜잭션(commit과 rollback)

데이터베이스 TRANSACTION

데이터베이스 TRANSACTION

- ◆ 트랜잭션은 데이터 처리의 한 단위입니다.
- ◆ 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
- DDL(create,drop,alter)이나 DCL(grant,revoke)문장의 실행(자동 COMMIT)
- 기계 장애 또는 시스템 충돌(crash)
- deadlock 발생
(트랜잭션과 트랜잭션 사이에 서로가 서로를 lock 을 걸어둔 데이터를 액세스하다 둘다 waiting 에 걸리는 현상)
- 사용자가 정상 종료

③ 자동 COMMIT은 다음의 경우 발생 합니다.

- DDL,DCL문장이 완료 될때
- 명시적인 COMMIT이나 ROLLBACK없이 SQL*Plus를 정상 종료 했을 경우

④ 자동 ROLLBACK은 다음의 경우 발생 합니다.

- SQL*Plus를 비정상 종료 했을 경우
- 비정상적인 종료, system failure

4. 데이터 조작용어(DML)

4.4. 트랜잭션(commit과 rollback)

COMMIT과 ROLLBACK

COMMIT과 ROLLBACK

COMMIT : 변경사항 저장

ROLLBACK : 변경사항 취소

① COMMIT과 ROLLBACK의 장점

- 데이터의 일관성을 제공 합니다.
- 데이터를 영구적으로 변경하기 전에 데이터 변경을 확인하게 합니다.
- 관련된 작업을 논리적으로 그룹화 할 수 있습니다.
- COMMIT, SAVEPOINT, ROLLBACK 문장으로 TRANSACTION의 논리를 제어할 수 있습니다.

② COMMIT이나 ROLLBACK 이전의 데이터 상태

- 데이터 이전의 상태로 복구가 가능합니다.
- 현재 사용자는 SELECT문장으로 DML작업의 결과를 확인할 수 있습니다.
- 다른 사용자는 SELECT문장으로 현재 사용자 사용한 DML문장의 결과를 확인할 수 없습니다.
- 변경된 행은 LOCK이 설정되어서 다른 사용자가 변경할 수 없습니다.

③ COMMIT이후의 데이터 상태

- 데이터베이스에 데이터를 영구적으로 변경
- 데이터의 이전 상태는 완전히 상실
- 모든 사용자가 결과를 볼 수 있습니다.
- 변경된 행의 LOCK이 해제되고 다른 사용자가 변경할 수 있습니다.
- 모든 SAVEPOINT는 제거 됩니다.

Savepoint a;

Rollback to savepoint a;//a까지 rollback

5. 내장 함수(Sing-Row Functions)

5.1. 숫자함수(Number Functions)

ABS(n) : ABS함수는 절대값을 계산하는 함수입니다.

CEIL(n) : CEIL함수는 주어진 값보다는 크지만 가장 근접하는 최소값을 구하는 함수입니다.

EXP(n) : EXP함수는 주어진 값의 e의 승수를 나타냅니다. e는 2.71828183..입니다.

FLOOR(n) : FLOOR함수는 주어진 값보다 작거나 같은 최대 정수값을 구하는 함수입니다. (CEIL 함수와 비교해 보세요.)

LN(n) : LN함수는 주어진 값의 자연로그 값을 반환합니다.

MOD(m, n) : MOD함수는 m을 n으로 나누어 남은 값을 반환한다. n이 0일 경우 m을 반환합니다.

POWER(m, n) : POWER함수는 m의 n승 값을 계산합니다.

ROUND(n, [m]) : ROUND함수는 n값의 반올림을 하는 함수로 m은 소숫점 아래 자릿수를 나타낸다.

SQL>SELECT ROUND(192.123, 1) TEST FROM dual ; SQL>SELECT ROUND(192.123, -1) TEST FROM dual ;

| TEST | TEST |
|-------|------|
| 192.1 | 190 |

SIGN(n) : SIGN함수는 n<0일 경우 -1DFM N=0일 경우 0을 N>0일 경우 1을 반환합니다.

SQRT(n) : SQRT함수는 n값의 루트값을 계산한다. n은 양수여야 합니다.

TRUNC(n, m) : TRUNC함수는 n값을 m 소숫점 자리로 반내림한 값을 반환합니다. 9 ROUND 함수와 비교해 보세요.)

SQL>SELECT TRUNC(7.5597, 2) TEST FROM dual ; SQL>SELECT TRUNC (5254.26, -2) TEST FROM dual ;

| TEST | TEST |
|------|------|
| 7.55 | 5200 |

5. 내장 함수(Sing-Row Functions)

5.2. 문자열 처리 함수(Character Functions)

👉 CONCAT(char1, char2)

CONCAT 함수는 Concatenation의 약자로 두 문자를 결합하는 역할을 합니다. "||" 연산자와 같은 역할을 합니다.

```
SQL>SELECT CONCAT('Oracle', ' Korea') NAME FROM dual ;
```

```
NAME
```

```
-----  
Oracle Korea
```

👉 INITCAP(char) : 주어진 문자열의 첫 번째 문자를 대문자로 변환시켜 줍니다.

👉 LOWER(char) : 문자열을 소문자로 변환 시켜 줍니다.

👉 UPPER(char) : 문자열을 대문자로 변환 시켜 줍니다.

👉 LPAD(char1, n [,char2]) :왼쪽에 문자열을 끼어 넣는 역할을 합니다.

n은 반환되는 문자열의 전체 길이를 나타내며, char1의 문자열이 n보다 클 경우 char1을 n개 문자열 만큼 반환합니다.

```
SQL>SELECT LPAD('JUNG-SICK', 10, '*') NAME FROM dual ;
```

```
NAME
```

```
-----  
*JUNG-SICK
```

👉 RPAD(char1, n [,char2]) : LPAD와 반대로 오른쪽에 문자열을 끼어 넣는 역할을 합니다

```
SQL>SELECT RPAD('JUNG-SICK', 10, '*') NAME FROM dual ;
```

```
NAME
```

```
-----  
JUNG-SICK*
```

5. 내장 함수(Sing-Row Functions)

5.2. 문자열 처리 함수(Character Functions)

👉 **SUBSTR(char, m, [n])** : SUBSTR함수를 이용하여 m 번째 자리부터 길이가 n개인 문자열을 반환한 합니다.
m이 음수일 경우에는 뒤에서 M번째 문자부터 반대 방향으로 n개의 문자를 반환합니다.

```
SQL>SELECT SUBSTR('JUNG-SICK', 3, 3) NAME FROM dual ; SQL>SELECT SUBSTR('JUNG-SICK', -3, 3) NAME FROM dual ;
```

| NAME | NAME |
|------|------|
| NG- | ICK |

👉 **LENGTH(char1)** : 문자열의 길이를 리턴 합니다.

```
SQL>SELECT LENGTH('JUNG-SICK') TEST FROM dual ;
```

| TEST |
|------|
| 9 |

👉 **REPLACE(char1, str1, str2)** : REPLACE는 문자열의 특정 문자를 다른 문자로 변환 합니다.

```
SQL> SELECT REPLACE('JACK and JUE', 'J', 'BL') "Changes" FROM DUAL;
```

| Changes |
|----------------|
| BLACK and BLUE |


-- 대소문자를 구분한다는 것을 알수 있습니다.

```
SQL>SELECT REPLACE('JACK and JUE', 'j', 'BL') "Changes" FROM DUAL
```

| Changes |
|--------------|
| JACK and JUE |

5. 내장 함수(Sing-Row Functions)

5.2. 문자열 처리 함수(Character Functions)


 **INSTR** : 문자열이 포함되어 있는지를 조사하여 문자열의 위치를 반환합니다.
지정한 문자열이 발견되지 않으면 0이 반환 됩니다.

-- 지정한 문자 OK가 발견되지 않아서 0이 반환 됩니다.

```
SQL>SELECT INSTR('CORPORATE FLOOR','OK') "Instring" FROM DUAL
Instring
-----
0
```

-- OR이 있는 위치 2를 반환 합니다. 왼쪽부터 비교를 한다는 것을 알 수 있습니다.

```
SQL>SELECT INSTR('CORPORATE FLOOR','OR') "Instring" FROM DUAL
Instring
-----
2
```

 **TRIM** : 특정한 문자를 제거 합니다.

제거할 문자를 입력하지 않으면 기본적으로 공백이 제거 됩니다. 리턴값의 데이터타입은 VARCHAR2 입니다.

-- 0을 제거 합니다.

```
SQL>SELECT TRIM(0 FROM 0009872348900) "TRIM Example" FROM DUAL;
TRIM Example
-----
98723489
```

-- 어떤 문자도 입력하지 않으면 기본적으로 공백이 제거 됩니다.

-- TRIM을 사용한 위에 예제와 사용하지 않은 아래 예제의 결과 값이 다르게 나오는 것을 알 수 있습니다.

```
SQL>SELECT NVL(TRIM (' '), '공백') "TRIM Example" FROM DUAL
TRIM Example
-----
공백
```

5. 내장 함수(Sing-Row Functions)

5.3. 날짜 처리 함수(Date Functions)

👉 **LAST_DAY(d)** : LAST_DAY함수는 달의 마지막 날의 날짜를 반환합니다

```
SQL>SELECT SYSDATE TODAY, LAST_DAY(SYSDATE) LASTDAY FROM dual ;
          TODAY                LASTDAY
```

```
-----
05-JUN-2000          30-JUN-2000
```

👉 **ADD_MONTHS(a, b)** : ADD_MONTHS 함수는 a의 날짜에 b의 달을 더한 값을 반환 합니다.

```
SQL>SELECT TO_CHAR(ADD_MONTHS(SYSDATE,3), 'RRRR/MM/DD' LASTDAY) "date" FROM dual ;
          date
```

```
-----
2000/09/05
```

👉 **MONTH_BETWEEN(a1, a2)** : MONTH_BETWEEN은 a1과 a2 사이의 달의 수를 NUMBER형 타입으로 반환 합니다.

```
SQL>SELECT MONTHS_BETWEEN(TO_DATE('2000/06/05') , TO_DATE('2000/09/23')) "Date" FROM dual ;
          Date
```

```
-----
-3.880635
```

👉 **ROUND(d[,F])** : ROUND 함수는 F에 지정된 단위로 반올림 합니다, F가 연도라면 연도 단위로 반올림 합니다.

```
SQL>SELECT ROUND(TO_DATE('1998/09/11'), 'YEAR') FROM dual ;
          ROUND(TO_
```

```
-----
99-01-01
```

5. 내장 함수(Sing-Row Functions)

5.4. 변환 함수(Conversion Functions)

☞ **TO_CHAR** : TO_CHAR함수는 DATE형, NUMBER형을 VARCHAR2형으로 바껴 줍니다.

```
SQL>SELECT TO_CHAR(SYSDATE, 'MONTH') CHARTEST FROM dual ;
        CHARTEST
```

```
-----
        JUNE
```

오늘이 6월 10일 이니깐요.. 오늘이 달인 6월이 짝합니다.

☞ **TO_DATE** : TO_DATE함수는 CHAR, VARCHAR2형을 DATE 타입으로 변환합니다.

```
SQL>SELECT TO_DATE('2000/06/16','RRRR/MM/DD') FROM dual ;
        TO_DATE(
```

```
-----
        2000/06/16
```

'2000/06/16' 문자열이 날짜형으로 변합니다

☞ **TO_NUMBER** : TO_NUMBER함수는 CHAR, VARCHAR2의 데이터 타입을 숫자형식으로 변환합니다.

```
SQL>SELECT TO_NUMBER('1210616') FROM dual ;
```

```
        TO_NUMBER(
```

```
-----
        1210616
```

'1210616' 문자열이 숫자형으로 변합니다

* SQL Function에 대해서 자세히 나와 있습니다.

http://otn.oracle.com/docs/products/oracle9i/doc_library/901_doc/server.901/a90125/functions2.htm

5. 내장 함수(Sing-Row Functions)

5.5. 기타 함수(Miscellaneous Single-Row Functions)

NVL (Mysql에서는 ifnull)

- NVL 함수는 NULL값을 다른 값으로 바꿀 때 쓰입니다.
- 모든 데이터 타입에 적용 가능합니다.
- 전환되는 값의 데이터 타입을 일치시켜야 합니다

##NVL(컬럼명, null일때 출력될 값) : 해당 컬럼의 데이터 값이 NULL이면 출력될 값으로 작성해라

```
SQL>SELECT empno, NVL(comm, 0) FROM emp WHERE deptno = 30;
```

```
EMPNO NVL(COMM,0)
```

| | |
|------|------|
| 7499 | 300 |
| 7654 | 1400 |
| 7698 | 0 |
| 7900 | 0 |

Commission이 없는 사원에 대해 0으로 바꾸어서 출력합니다.

DECODE

- DECODE 함수는 데이터 들을 다른 값으로 바꾸어 줍니다.
- 형식 DECODE(VALUE, IF1, THEN1, IF2, THEN2...)
- VALUE 값이 IF1일경우에 THEN1값으로 바꾸어 주고 VALUE값이 IF2일경우에는 THEN2값으로 바꾸어 줍니다.

```
SQL> SELECT deptno, DECODE(deptno, 10 , 'ACCOUNTING', 20 , 'RESEARCH', 30 , 'SALES', 40 , 'OPERATIONS')
```

```
FROM emp ;
```


```
DEPTNO DECODE(DEP
```

| | |
|----|------------|
| 20 | RESEARCH |
| 30 | SALES |
| 10 | ACCOUNTING |

부서가 10번이면 'ACCOUNTING'를 20번이면 'RESEARCH'를 30번이면 'SALES'를 40번이면 'OPERATIONS'를 출력하는 예제 입니다.

5. 내장 함수(Sing-Row Functions)

5.5. 기타 함수(Miscellaneous Single-Row Functions)

 **DUMP** : DUMP는 바이트 크기와 해당 데이터 타입 코드를 반환합니다..

```
SQL>SELECT ename, DUMP(ename, 16) "16진수" FROM emp WHERE ename = 'ALLEN'
```

```
ename      16진수
```

```
-----  
ALLEN      Typ=1 Len=5: 41,4c,4c,45,4e
```

만약에 16대신 8을 넣으면 8진수로, 10를 넣으면 10진수로 변환이 됩니다..

16, 10, 8, 17이 올수 있는데요 17은 단일 문자열을 반환 한데요.. 한번 테스트 해보세요

Len은 ename의 해당 byte수 입니다. ..


 **GREATEST** : GREATEST함수는 검색값 중에서 가장 큰 값을 반환 합니다..

```
SQL>SELECT GREATEST(10, 100, 5, -7) FROM DUAL;
```

```
GREATEST(10,100,5,-7)
```

```
-----  
100
```

--가장 큰 수는 100이겠쥬..

 **LEAST** : LEAST함수는 GREATEST함수와 반대로 가장 작은 값을 반환합니다.

```
SQL>SELECT LEAST(10, 100, 5, -7) FROM DUAL;
```

```
LEAST(10,100,5,-7)
```

```
-----  
-7
```

5. 내장 함수(Sing-Row Functions)

5.5. 기타 함수(Miscellaneous Single-Row Functions)

- ☞ **UID** : 현재 사용자의 유일한 ID번호를 반환 합니다.
USER : 현재 오라클을 사용하는 사용자를 VARCHAR2형식으로 반환 합니다.

```
SQL> SELECT USER, UID FROM DUAL;
```

| USER | UID |
|-------|-----|
| SCOTT | 32 |

- ☞ **USERENV** : USERENV 함수는 현재 세션의 환경 정보를 반환합니다.
- ENTRYID : 사용 가능한 Auditing entry Identifier를 반환합니다.
 - LABEL : 현재 세션의 Label을 반환합니다.
 - LANGUAGE : 현재 세션에서 사용중인 언어와 테리토리 값을 반환합니다.
 - SESSIONID : Auditing(감사) Session ID를 반환 합니다.
 - TERMINAL : 현재 세션 터미널의 OS ID를 반환 합니다.

```
SQL> SELECT USERENV('LANGUAGE') FROM DUAL;
```

| USERENV('LANGUAGE') |
|--------------------------|
| KOREAN_KOREA.KO16KSC5601 |

- ☞ **VSIZ** : 해당 문자의 BYTE수를 반환 합니다. 해당 문자가 NULL이면 NULL값이 반환 됩니다.

```
SQL> SELECT VSIZ(ename), ename FROM emp WHERE deptno = 30;
```

| VSIZ(ENAME) | ENAME |
|-------------|--------|
| 5 | ALLEN |
| 4 | WARD |
| 6 | MARTIN |


6. 그룹 함수(Group Functions)

6.1. Group Function의 종류

그룹 함수란 ?

- 그룹 함수란 여러 행 또는 테이블 전체의 행에 대해 함수가 적용되어 하나의 결과값을 가져오는 함수를 말합니다.
- **GROUP BY**절을 이용하여 그룹 당 하나의 결과가 주어지도록 그룹화 할 수 있습니다.
- **HAVING**절을 사용하여 그룹 함수를 가지고 조건비교를 할 수 있습니다.
- **COUNT(*)**를 제외한 모든 그룹함수는 NULL값을 고려하지 않습니다.
- **MIN, MAX** 그룹함수는 모든 자료형에 대해서 사용 할 수 있습니다.

그룹 함수의 종류


 **COUNT** : COUNT 함수는 검색된 행의 수를 반환합니다.

```
SQL>SELECT COUNT(deptno) FROM DEPT ;
```

```
COUNT(DEPTNO)
```

```
4
```

==> 검색된 행의 총 수 4개를 반환합니다. 즉 4개의 부서가 존재합니다.

 **MAX** : MAX 함수는 컬럼중의 최대값을 반환합니다.

```
SQL>SELECT MAX(sal) salary FROM emp ;
```


```
SALARY
```

```
5000
```

==> sal컬럼중에서 제일 큰값을 반환합니다. 즉 가장 큰 급여를 반환합니다.


6. 그룹 함수(Group Functions)

6.1. Group Function의 종류

 **MIN** : MIN 함수는 컬럼중의 최소값을 반환합니다.


```
SQL>SELECT MIN(sal) salary FROM emp ;  
SALARY
```

```
-----  
800    ==>    sal컬럼중에서 가장 작은 값 반환합니다. 즉 가장 적은 급여를 반환합니다
```

 **AVG** : AVG 함수는 평균값을 반환합니다.


```
SQL>SELECT ROUND(AVG(sal),1) salary FROM emp WHERE deptno = 30  
SALARY
```

```
-----  
1566.7    ==>    30부서 사원의 평균 급여를 소수점 1자리 이하에서 반올림해서 보여줍니다.
```

 **SUM** : SUM 함수는 검색된 컬럼의 합을 반환합니다.

```
SQL>SELECT SUM(sal) salary FROM emp WHERE deptno = 30;  
SALARY
```

```
-----  
9400    ==>    30부서 사원의 급여 합계를 보여줍니다.
```

 **STDDEV** : STDDEV 함수는 표준편차를 반환합니다.

```
SQL> SELECT ROUND(STDDEV(sal),3) salary FROM emp WHERE deptno = 30 ;  
SALARY
```

```
-----  
668.331    ==>    30부서 사원의 급여 표준편차를 반환합니다.
```

6. 그룹 함수(Group Functions)

6.2. Group By절과 Having절

GROUP BY ?

- 특정한 컬럼의 데이터 들을 다른 데이터들과 비교해 유일한 값에 따라 무리를 짓습니다.
- **GROUP BY**절을 사용하여 한 테이블의 행들을 원하는 그룹으로 나눕니다.
- Column명을 GROUP함수와 SELECT절에 사용하고자 하는 경우 GROUP BY뒤에 Column명을 추가 합니다.

GROUP BY 예제

예제1) 부서별로 그룹을 지은 검색 결과 값이며 부서별로 사원수를 보여줍니다. .

```
SQL>SELECT b.deptno, COUNT(a.empno) FROM emp a, dept b WHERE a.deptno = b.deptno
      GROUP BY b.deptno
```

| DEPTNO | COUNT(*) |
|--------|----------|
|--------|----------|

| | |
|----|---|
| 10 | 3 |
| 20 | 5 |
| 30 | 6 |

예제2)업무별로 그룹하여 업무, 인원수, 평균 급여액, 최고 급여액, 최저 급여액 및 합계를 출력하라.

```
SQL>SELECT job, COUNT(empno) "인원수", AVG(sal) "평균급여액", MAX(sal) "최고급여액", MIN(sal) "최저급여액",
      SUM(sal) "급여합계"
      FROM emp GROUP BY job
```

| JOB | 인원수 | 평균급여액 | 최고급여액 | 최저급여액 | 급여합계 |
|-----------|-----|--------|-------|-------|------|
| ANALYST | 2 | 3000 | 3000 | 3000 | 6000 |
| CLERK | 4 | 1037.5 | 1300 | 800 | 4150 |
| MANAGER | 3 | 2840 | 2975 | 2695 | 8520 |
| PRESIDENT | 1 | 5000 | 5000 | 5000 | 5000 |
| SALESMAN | 4 | 1400 | 1600 | 1250 | 5600 |

6. 그룹 함수(Group Functions)

6.2. Group By절과 Having절

GROUP BY의 HAVING 절

- WHERE절에 GROUP Function을 사용할 수 없습니다.
- HAVING절은 GROUP 함수를 가지고 조건비교를 할 때 사용합니다.
- WHERE → GROUP BY → HAVING → ORDER BY순으로 쿼리문이 와야 됩니다.

예제 1) 사원수가 5명이 넘는 부서의 부서명과 사원수를 출력해라

```
SQL>SELECT b.dname, COUNT(a.empno) FROM emp a, dept b WHERE a.deptno = b.deptno
      GROUP BY dname
      HAVING COUNT(a.empno) > 5
```

| DNAME | COUNT(A.EMPNO) |
|----------|----------------|
| RESEARCH | 6 |
| SALES | 6 |

예제 2) 전체 월급이 5000을 초과하는 각 업무에 대해서 업무와 월급여 합계를 출력하여라.
단 판매원은 제외하고 월 급여 합계로 내림차순 정렬 하여라.

```
SQL>SELECT job, SUM(sal) "급여합계" FROM emp
      WHERE job NOT IN ('SALES')          -- 판매원은 제외
      GROUP BY job                        -- 업무별로 Group By
      HAVING SUM(sal) > 5000              -- 전체 월급이 5000을 초과하는
      ORDER BY SUM(sal) DESC;            -- 월급여 합계로 내림차순 정렬
```

| JOB | 급여합계 |
|----------|------|
| MANAGER | 8520 |
| ANALYST | 6000 |
| SALESMAN | 5600 |

7. 서브쿼리(Subquery)

7.1 서브쿼리(Subquery)란 ?

Subquery란 ?

- ◆ SUBQUERY는 다른 하나의 SQL 문장의 절에 NESTED된 SELECT 문장 입니다.
- ◆ SELECT, UPDATE, DELETE, INSERT와 같은 DML문과 CREATE TABLE 또는 VIEW에서 이용 될 수 있습니다.
- ◆ 알려지지 않은 조건에 근거한 값들을 검색하는 SELECT 문장을 작성하는데 유용 합니다.
- ◆ SUBQUERY는 MAIN QUERY가 실행되기 이전에 한번 실행 됩니다.

Guidelines

- ◆ SUBQUERY는 괄호로 묶어야 합니다. ◆ SUBQUERY는 연산자의 오른쪽에 나타나야 합니다.
- ◆ 두 종류의 비교 연산자들이 SUBQUERY에 사용 됩니다.
 - 단일 행 연산자(=, >, >=, <, <=, <>, !=) - 복수 행 연산자(IN, NOT IN, ANY, ALL, EXISTS)

Guidelines

- ◆ 단일 행(Sing-Row) 서브쿼리 : SELECT문장으로부터 오직 하나의 행만을 검색하는 질의입니다
- ◆ 다중 행(Multiple-Row) 서브쿼리 : SELECT문장으로부터 하나 이상의 행을 검색하는 질의입니다
- ◆ 다중 열(Multiple-Column) 서브쿼리 : SELECT문장으로부터 하나 이상의 컬럼을 검색하는 질의입니다
- ◆ FROM절상의 서브쿼리(INLINE VIEW) : FROM절상에 오는 서브쿼리로 VIEW처럼 작용 합니다.
- ◆ 상관관계 서브 쿼리 : 바깥쪽 쿼리의 컬럼 중의 하나가 안쪽 서브쿼리의 조건에 이용되는 처리 방식 입니다.

7. 서브쿼리(Subquery)

7.2. 단일 행(Sing-Row) 서브쿼리

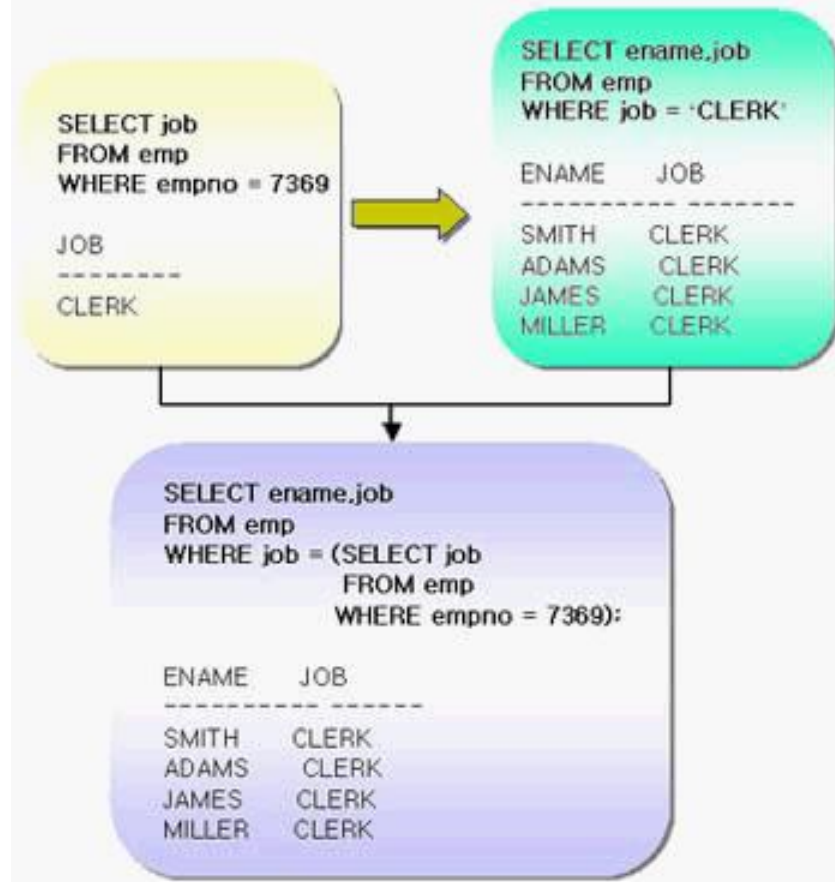
☞ 단일 행(Sing-Row) 서브쿼리 란 ?

◆ 오직 한개의 행(값)을 반환 합니다.(Return only one row)

◆ 단일 행 연산자(=, >, >=, <, <=, <>, !=) 만 사용할 수 있습니다.

```
SQL> SELECT ename, job
      FROM emp
      WHERE job = (SELECT job
                  FROM emp
                  WHERE empno = 7369);
```

위의 서브쿼리를 실행 시키면. 오른쪽의 그림처럼.
empno가 7369인 job을 먼저 검색하고..
job이 'CLERK'인 사원의 이름과 직업을 반환 합니다.



7. 서브쿼리(Subquery)

7.3. 다중 행(Multiple-Row) 서브쿼리

☞ 다중 행(Multiple-Row) 서브쿼리 란 ?

- ◆ 하나 이상의 행을 RETURN하는 SUBQUERY를 다중 행 SUBQUERY라고 합니다.
- ◆ 복수 행 연산자(IN, NOT IN, ANY, ALL, EXISTS)를 사용 할 수 있습니다.

☞ IN 연산자의 사용 예제

예제) 부서별로 가장 급여를 많이 받는 사원의 정보를 출력하는 예제 입니다.

```
SQL>SELECT empno,ename,sal,deptno
FROM emp
WHERE sal IN (SELECT MAX(sal)
              FROM emp
              GROUP BY deptno);
```

| EMPNO | ENAME | SAL | DEPTNO |
|-------|-------|------|--------|
| 7698 | BLAKE | 2850 | 30 |
| 7788 | SCOTT | 3000 | 20 |
| 7902 | FORD | 3000 | 20 |
| 7839 | KING | 5000 | 10 |

☞ ANY 연산자의 사용 예제

- ANY 연산자는 서브쿼리의 결과값중 어느 하나의 값이라도 만족이 되면 결과값을 반환 합니다.

```
SQL>SELECT ename, sal
FROM emp
WHERE deptno != 20
AND sal > ANY(SELECT sal
              FROM emp
              WHERE job='SALESMAN');
```

| ENAME | SAL |
|--------|------|
| ALLEN | 1600 |
| BLAKE | 2850 |
| CLARK | 2450 |
| KING | 5000 |
| TURNER | 1500 |
| MILLER | 1300 |

6 개의 행이 선택되었습니다.

7. 서브쿼리(Subquery)

7.3. 다중 행(Multiple-Row) 서브쿼리

☞ ALL 연산자의 사용 예제

- ALL 연산자는 서브쿼리의 결과값중 모든 결과 값이 만족 되어야만 결과값을 반환 합니다.

```
SQL>SELECT ename, sal
      FROM emp
      WHERE deptno != 20
            AND sal > ALL(SELECT sal
                          FROM emp
                          WHERE job='SALESMAN');
```

| ENAME | SAL |
|-------|------|
| BLAKE | 2850 |
| CLARK | 2450 |
| KING | 5000 |

3 개의 행이 선택되었습니다.

☞ EXISTS 연산자의 사용 예제

- EXISTS 연산자를 사용하면 서브쿼리의 데이터가 존재하는가의 여부를 먼저 따져 존재하는 값들만을 결과로 반환해 줍니다.
- SUBQUERY에서 적어도 1개의 행을 RETURN하면 논리식은 참이고 그렇지 않으면 거짓 입니다.

예제) 사원을 관리할 수 있는 사원의 정보를 보여 줍니다.

```
SELECT empno, ename, sal
FROM emp e
WHERE EXISTS (SELECT empno
              FROM emp
              WHERE e.empno = mgr)
```

| EMPNO | ENAME | SAL |
|-------|-------|------|
| 7566 | JONES | 2975 |
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7788 | SCOTT | 3000 |
| 7839 | KING | 5000 |
| 7902 | FORD | 3000 |

6 개의 행이 선택되었습니다.

7. 서브쿼리(Subquery)

7.4. 다중 열(Multiple-Column) 서브쿼리

☞ 다중 열(Multiple-Column) 서브쿼리 란 ?

◆ 다중 열 서브쿼리란 서브쿼리의 결과값이 두개 이상의 컬럼을 반환하는 서브쿼리 입니다.

☞ Pairwise(쌍비교) Subquery

◆ 서브쿼리가 한번 실행되면서 모든 조건을 검색해서 주 쿼리로 넘겨 줍니다.

ex)

```
SELECT empno, sal, deptno
FROM emp
WHERE (sal, deptno) IN ( SELECT sal, deptno
                        FROM emp
                        WHERE deptno = 30
                        AND comm is NOT NULL );
```

| EMPNO | SAL | DEPTNO |
|-------|------|--------|
| 7521 | 1250 | 30 |
| 7654 | 1250 | 30 |
| 7844 | 1500 | 30 |
| 7499 | 1600 | 30 |

☞ Null Values in a Subquery

◆ 서브쿼리에서 null값이 반환되면 주 쿼리에서는 어떠한 행도 반환되지 않습니다.

☞ Nonpairwise(비쌍비교) Subquery

◆ 서브쿼리가 여러 조건별로 사용 되어서 결과값을 주 쿼리로 넘겨 줍니다.

ex)

```
SELECT empno, sal, deptno
FROM emp
WHERE sal IN ( SELECT sal
               FROM emp
               WHERE deptno = 30
               AND comm is NOT NULL )
AND deptno IN ( SELECT deptno
                FROM emp
                WHERE deptno = 30
                AND comm is NOT NULL );
```

| EMPNO | SAL | DEPTNO |
|-------|------|--------|
| 7521 | 1250 | 30 |
| 7654 | 1250 | 30 |
| 7844 | 1500 | 30 |
| 7499 | 1600 | 30 |

7. 서브쿼리(Subquery)

7.5. FROM절상의 서브쿼리(INLINE VIEW)와 상관관계 서브쿼리

👉 FROM절상의 서브쿼리(INLINE VIEW) 란 ?

- ◆ SUBQUERY는 FROM절에서도 사용이 가능 합니다.
- ◆ INLINE VIEW란 FROM절상에 오는 서브쿼리로 VIEW처럼 작용 합니다.

예제)급여가 20부서의 평균 급여보다 크고 사원을 관리하는 사원으로서 20부서에 속하지 않은 사원의 정보를 보여주는 SQL문 입니다.

```
SELECT b.empno,b.ename,b.job,b.sal , b.deptno
FROM (SELECT empno
      FROM emp WHERE sal >(SELECT AVG(sal)
                           FROM emp
                           WHERE deptno = 20)) a,
      emp b
WHERE a.empno = b.empno
AND b.mgr is NOT NULL
AND b.deptno != 20
```

| EMPNO | ENAME | JOB | SAL | DEPTNO |
|-------|-------|---------|------|--------|
| 7698 | BLAKE | MANAGER | 2850 | 30 |
| 7782 | CLARK | MANAGER | 2450 | 10 |

👉 상관관계 서브쿼리

- ◆ 상관관계 서브쿼리란 바깥쪽 쿼리의 컬럼 중의 하나가 안쪽 서브쿼리의 조건에 이용되는 처리 방식 입니다.
- ◆ 이는 주 쿼리에서 서브 쿼리를 참조하고 이 값을 다시 주 쿼리로 반환한다는 것입니다.

예제) 사원을 관리할 수 있는 사원의 평균급여보다 급여를 많이 받는 사원의 정보를 출력

```
SELECT empno, ename, sal
FROM emp e
WHERE sal > (SELECT AVG(sal) sal
             FROM emp
             WHERE e.empno = mgr)
```

| EMPNO | ENAME | SAL |
|-------|-------|------|
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7788 | SCOTT | 3000 |
| 7839 | KING | 5000 |
| 7902 | FORD | 3000 |

7. 서브쿼리(Subquery)

7.6. 집합 쿼리(UNION, INTERSECT, MINUS)

👉 집합 쿼리(UNION, INTERSECT, MINUS)

◆ 집합 연산자를 사용시 집합을 구성할 컬러의 데이터 타입이 동일해야 합니다.

◆ UNION :합집합 ◆ UNION ALL:공통원소 두번씩 다 포함한 합집합

◆ INTERSECT:교집합 ◆ MINUS:차집합

👉 UNION

◆ UNION은 두 테이블의 결합을 나타내며, 결합시키는 두 테이블의 중복되지 않은 값들을 반환 합니다.

```
SQL>SELECT deptno FROM emp
UNION
SELECT deptno FROM dept;
```

| DEPTNO |
|--------|
| 10 |
| 20 |
| 30 |
| 40 |

👉 UNION ALL

◆ UNION과 같으나 두 테이블의 중복되는 값까지 반환 합니다.

```
SQL>SELECT deptno FROM emp
UNION ALL
SELECT deptno FROM dept;
```

| DEPTNO |
|--------|
| 20 |
| 30 |
| 30 |
| 20 |
| 10 |
| 20 |
| 10 |
| 30 |
| |

7. 서브쿼리(Subquery)

7.6. 집합 쿼리(UNION, INTERSECT, MINUS)

INTERSECT

- ◆ INTERSECT는 두 행의 집합중 공통된 행을 반환 합니다.

```
SQL>SELECT deptno FROM emp
      INTERSECT
      SELECT deptno FROM dept;
```

| DEPTNO |
|--------|
| 10 |
| 20 |
| 30 |

MINUS

- ◆ MINUS는 첫번째 SELECT문에 의해 반환되는 행중에서 두번째 SELECT문에 의해 반환되는 행에 존재하지 않는 행들을 보여 줍니다.

```
SQL>SELECT deptno FROM dept
      MINUS
      SELECT deptno FROM emp;
```

| DEPTNO |
|--------|
| 40 |

8. 데이터 사전(Data Dictionary)

8. 데이터 사전(Data Dictionary)

데이터 사전(Data Dictionary)이란 ?

- ◆ 데이터 사전(Data Dictionary)이란 읽기전용 테이블 및 뷰들의 집합으로 데이터베이스 전반에 대한 정보를 제공 합니다.
- ◆ 데이터 사전에 저장되는 내용은 아래와 같습니다.
 - 오라클의 사용자 이름
 - 오라클 권한과 롤
 - 데이터베이스 스키마 객체(Table, View, index, cluster, Synonym, Sequence..) 이름과 정의들
 - 무결성제약 조건에 관한 정보
 - 데이터베이스의 구조 정보
 - 오라클 데이터베이스의 함수 와 프로지저 및 트리거에 대한 정보
 - 기타 일반적인 DataBase 정보 들이 있습니다.
- ◆ Oracle 사용자 SYS는 데이터 디셔너리의 모든 기본 Table과 View를 소유 합니다.
- ◆ Oracle은 DDL 명령이 실행될때 마다 Data Dictionary를 Access 합니다.
- ◆ 모든 Oracle 사용자는 DB정보에 대한 읽기 전용 참조로 Data Dictionary 사용할 수 있습니다.
- ◆ DB작업동안 Oracle은 Data Dictionary를 읽어 개체의 존재여부와 사용자에게 적합한 Access 권한이 있는지를 확인 합니다. 또한 Oracle은 Data Dictionary를 계속 갱신하여 DataBase 구조, 감사, 사용자권한, 데이터등의 변경사항을 반영 합니다.
- ◆ 데이터 사전 테이블은 아래의 스크립트를 실행시켜서도 생성할 수 있습니다.
 - @\$ORACLE_HOME/rdbms/admin/sql.bsq ==> 기본 데이터사전 테이블 생성 스크립트
 - @\$ORACLE_HOME/rdbms/admin/catalog.sql ==> 자주 사용되는 데이터사전 뷰 생성 스크립트

8. 데이터 사전(Data Dictionary)

8. 데이터 사전(Data Dictionary)

User_XXXX View

- ◆ 한 특정 사용자에게 종속되어 있고, 그 사용자가 조회 가능한 데이터 사전 뷰들입니다.
- ◆ ALL_XXXX View의 모든 정보의 부분 집합입니다.
- ◆ Public Synonym을 가질 수 있습니다.

아래의 예는 Scott 사용자의 Table 조회 결과입니다.

```
SQL> SELECT table_name, tablespace_name FROM USER_TABLES;
```

| TABLE_NAME | TABLESPACE_NAME |
|------------|-----------------|
| BONUS | TOOLS |
| DEPT | TOOLS |
| DUMMY | TOOLS |

ALL_XXXX View

- ◆ 한 특정 사용자가 조회 가능한 모든 데이터사전 뷰를 의미 합니다.
- ◆ 자신이 조회하려는 객체의 주인이 아니더라도 그 객체에 접근 할 수 있는 권한을 가지고 있다면 ALL_XXX 뷰를 통하여 조회가 가능 합니다.

```
SQL> SELECT table_name, tablespace_name FROM ALL_TABLES;
```

| | |
|----------------------|--------|
| DUAL | SYSTEM |
| SYSTEM_PRIVILEGE_MAP | SYSTEM |
| TABLE_PRIVILEGE_MAP | SYSTEM |
| DEPT | TEST |
| DUMMY | TEST |
| ... | |

8. 데이터 사전(Data Dictionary)

8. 데이터 사전(Data Dictionary)

DBA_XXXX View

- ◆ DBA권한을 가진 사용자 만이 조회할 수 있는 데이터 사전으로서 모든 오라클 데이터베이스 객체에 대한 정보를 보여 줍니다.
- ◆ SELECT ANY TABLE 권한이 있는 사용자 또한 질의가 가능 합니다.
- ◆ 이러한 View에 대한 동의어는 생성되지 않으며, 다른 사용자가 질의 하려면 앞에 sys.이라는 접두어를 붙여야 합니다.

ex) `SELECT OWNER, OBJECT_NAME FROM SYS.DBA_OBJECTS;`

9. 오라클 객체

9.1. 인덱스(Index)

※ 인덱스란?

- ◆ 인덱스는 테이블이나 클러스트에서 쓰여지는 선택적인 객체로서, 오라클 데이터베이스 테이블내의 원하는 레코드를 빠르게 찾아갈 수 있도록 만들어진 데이터 구조입니다.
- ◆ 자동 인덱스 : 프라이머리 키 또는 unique 제한 규칙에 의해 자동적으로 생성되는 인덱스 입니다.
- ◆ 수동 인덱스 : CREATE INDEX 명령을 실행해서 만드는 인덱스들 입니다.

※ Index를 생성하는 것이 좋은 Column

- ① WHERE절이나 join조건 안에서 자주 사용되는 컬럼
- ② null 값이 많이 포함되어 있는 컬럼
- ③ WHERE절이나 join조건에서 자주 사용되는 두 개이상의 컬럼들
- ◆ 다음과 같은 경우에는 index 생성이 불필요 합니다.
 - ① table이 작을 때
 - ③ 테이블이 자주 갱신될 때

- ※ 오라클 인덱스는 B-tree(binary search tree)에 대한 원리를 기반으로 하고 있습니다.
B-tree인덱스는 컬럼안에 독특한 데이터가 많을 때 가장 좋은 효과를 냅니다.

이 알고리즘 원리는

- ① 주어진 값을 리스트의 중간점에 있는 값과 비교합니다.
만약 그 값이 더 크면 리스트의 아래쪽 반을 버립니다. 만약 그 값이 더 작다면 위쪽 반을 버립니다.
- ② 하나의 값이 발견될 때 까지 또는 리스트가 끝날 때까지 그와 같은 작업을 다른 반쪽에도 반복합니다.

9. 오라클 객체

9.1. 인덱스(Index)

① Bitmap 인덱스

- ◆ 비트맵 인덱스는 각 컬럼에 대해 적은 개수의 독특한 값이 있을 경우에 가장 잘 작동합니다. 그러므로 비트맵 인덱스는 B-tree 인덱스가 사용되지 않을 경우에서 성능을 향상 시킵니다.
- ◆ 테이블이 매우 크거나 수정/변경이 잘 일어나지 않는 경우에 사용할 수 있습니다. 예를 들어 여권 기록을 포함하고 있는 테이블의 성별 열이나 결혼 여부 열의 경우에는 B-트리 인덱스 보다는 비트맵 인덱스가 더 유리할 것입니다.
- ◆ 질의문이 OR 연산자를 포함하는 여러 개의 WHERE 조건을 자주 사용할 때 유리합니다

```
SQL>CREATE BITMAP INDEX emp_deptno_indx  
ON emp(deptno);
```

② Unique 인덱스

- ◆ Unique 인덱스는 인덱스를 사용한 컬럼의 중복값들을 포함하지 않고 사용할 수 있는 장점이 있습니다.
- ◆ 프라이머키 와 Unique 제약 조건시 생성되는 인덱스는 Unique 인덱스입니다.

```
SQL>CREATE UNIQUE INDEX emp_ename_indx  
ON emp(ename);
```

③ Non-Unique 인덱스

- ◆ Non-Unique 인덱스는 인덱스를 사용한 컬럼에 중복 데이터 값을 가질 수 있습니다.

```
SQL>CREATE INDEX dept_dname_indx  
ON dept(dname);
```

9. 오라클 객체

9.1. 인덱스(Index)

④ 결합 (Concatenated(=Composite)) 인덱스

◆ 복수개의 컬럼에 생성할 수 있으며 복수키 인덱스가 가질 수 있는 최대 컬럼값은 16개입니다

```
SQL>CREATE UNIQUE INDEX emp_empno_ename_indx  
ON emp(empno, ename);
```

※ 인덱스의 삭제

◆ 인덱스의 구조는 테이블과 독립적이므로 인덱스의 삭제는 테이블의 데이터에는 아무런 영향도 미치지 않습니다.

◆ 인덱스를 삭제하려면 INDEX의 소유자이거나 DROP ANY INDEX권한을 가지고 있어야 합니다.

◆ INDEX는 ALTER를 할 수 없습니다.

```
SQL>DROP INDEX emp_empno_ename_indx ;
```

※ 인덱스에 대한 정보는 USER_INDEXES 뷰 또는 USER_IND_COLUMNS 뷰를 통해 검색할 수 있습니다.

```
SQL> SELECT index_name , index_type  
FROM USER_INDEXES  
WHERE table_name='EMP';
```

| INDEX_NAME | INDEX_TYPE |
|----------------|------------|
| EMP_DEPTNO_IDX | BITMAP |
| EMP_PK_EMPNO | NORMAL |

9. 오라클 객체

9.2. VIEW 테이블

뷰란?

- ◆ 뷰는 하나의 가상 테이블이라 생각하시면 됩니다.
- ◆ 뷰는 실제 데이터가 저장 되는 것은 아니지만 뷰를 통해 데이터를 관리 할 수 있습니다.
- ◆ 뷰는 복잡한 query를 통해 얻을 수 있는 결과를 간단한 query를 써서 구할 수 있게 합니다.
- ◆ 한개의 뷰로 여러 테이블에 대한 데이터를 검색할 수 있습니다.
- ◆ 특정 평가기준에 따른 사용자 별로 다른 데이터를 액세스할 수 있도록 합니다.

뷰의 제한 조건

- ◆ 테이블에 NOT NULL로 만든 컬럼들이. 뷰에 다 포함이 되 있어야 됩니다.
- ◆ 그리고 ROWID, ROWNUM, NEXTVAL, CURRVAL 등과 같은 가상 컬럼에 대한 참조를 포함하고 있는 뷰 에는 어떤 데이터도 Insert할 수 없습니다.
- ◆ WITH READ ONLY 옵션을 설정한 뷰도 데이터를 갱신할 수 없습니다.
- ◆ WITH CHECK OPTION을 설정한 뷰는 뷰의 조건에 해당되는 데이터만 삽입, 삭제, 수정을 할 수 있습니다.

9. 오라클 객체

9.2. VIEW 테이블

👉 뷰 생성 문법

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view_name  
AS subquery  
[WITH CHECK OPTION [CONSTRAINT 제약조건]]  
[WITH READ ONLY]
```

- FORCE : 기본 테이블 유무에 관계없이 VIEW를 생성
- WITH CHECK OPTION : VIEW에 의해 액세스될 수 있는 행만이 입력되거나 변경될 수 있음을 지정 합니다.
- WITH READ ONLY : SELECT만 가능한 VIEW의 생성
- 함수를 사용한 컬럼은 반드시 ALIAS를 지정해야 합니다.

```
SQL> CREATE OR REPLACE VIEW Name_Query  
AS  
    SELECT a.ename, b.dname  
    FROM   emp a, dept b  
    WHERE  a.deptno = b.deptno  
          AND b.deptno = 20
```

이렇게 뷰를 생성해 놓고 뷰를 통해 검색을 하면 됩니다.

```
SQL>SELECT * FROM Name_Query;
```

| ENAME | DNAME |
|-------|----------|
| SMITH | RESEARCH |
| JONES | RESEARCH |

9. 오라클 객체

9.2. VIEW 테이블

WITH CHECK OPTION

- ◆ view의 조건식을 만족하는 데이터만 INSERT 또는 UPDATE가 가능하도록 하는 옵션 입니다.

```
SQL> CREATE OR REPLACE VIEW Check_Option
      AS
```

```
    SELECT empno, ename, deptno
    FROM   emp
    WHERE  deptno = 10
    WITH CHECK OPTION
```

```
SQL> INSERT INTO Check_Option(empno, ename, deptno)
      VALUES (10005, 'jain', 30);
```

```
INSERT INTO Check_Option(empno, ename, deptno)
      *
```

1행에 오류:

ORA-01402: 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다
부서 번호가 10인 사원만 INSERT, UPDATE할 수 있습니다.

WITH READ ONLY

- ◆ SELECT만 가능한 VIEW를 생성 합니다.

```
SQL> CREATE OR REPLACE VIEW Read_Only
      AS
    SELECT empno, ename, deptno
    FROM   emp
    WHERE  deptno = 10
    WITH READ ONLY
```

view created.

단순히 읽기 만 할 수 있고 데이터는 입력하지 못합니다.

- ◆ 뷰의 정보 조회 : **USER_VIEWS** 데이터 사전을 통해서 뷰에 대한 정보를 조회 할 수 있습니다.

```
SQL>SELECT view_name , text  FROM   USER_VIEWS;
```

- ◆ 뷰의 삭제

```
SQL>DROP VIEW Read_Only;
view dropped.
```

9. 오라클 객체

9.3. SEQUENCE(시퀀스)

👉 시퀀스 란?

- ◆ 유일(UNIQUE)한 값을 생성해주는 오라클 객체입니다.
- ◆ 시퀀스를 생성하면 기본키와 같이 순차적으로 증가하는 컬럼을 자동적으로 생성할수 있습니다.
- ◆ 보통 primary key 값을 생성하기 위해 사용합니다.
- ◆ 메모리에 Cache되었을 때 Sequence 값의 액세스 효율이 증가 합니다.
- ◆ Sequence는 테이블과는 독립적으로 저장되고 생성됩니다. 따라서 하나의 sequence를 여러 테이블에서 쓸 수 있습니다.

👉 시퀀스 생성 문법

```
CREATE SEQUENCE sequence_name  
[START WITH n]  
[INCREMENT BY n]  
[MAXVALUE n | NOMAXVALUE]  
[MINVALUE n | NOMINVALUE]  
[CYCLE | NOCYCLE]
```

START WITH : 시퀀스의 시작 값을 지정합니다. n을 1로 지정하면 1부터 순차적으로 시퀀스번호가 증가 합니다.

INCREMENT BY : 시퀀스의 증가 값을 말합니다. n을 2로 하면 2씩 증가합니다.

START WITH를 1로 하고 INCREMENT BY를 2으로 하면 1, 3,5,7,... 이렇게 시퀀스 번호가 증가하게 됩니다.

MAXVALUE n | NOMAXVALUE : MAXVALUE는 시퀀스가 증가할수 있는 최대값을 말합니다.
NOMAXVALUE는 시퀀스의 값을 무한대로 지정합니다.

MINVALUE n | NOMINVALUE : MINVALUE는 시퀀스의 최소값을 지정 합니다.
기본값은 1이며, NOMINVALUE를 지정할 경우 최소값은 무한대가 됩니다

9. 오라클 객체

9.3. SEQUENCE(시퀀스)

```
SQL>CREATE SEQUENCE emp_seq  
      START WITH 1  
      INCREMENT BY 1  
      MAXVALUE 100000 ;
```

➔ 시작 값이 1이고 1씩 증가하고, 최대값이 100000만이 되는 시퀀스를 생성했습니다.

```
SQL>INSERT INTO emp(empno, ename, hiredate ) VALUES(emp_seq.NEXTVAL, 'julia' , sysdate);
```

➔ empno는 컬럼값을 입력할 때 일일이 다음 값을 기억하지 않아도 NEXTVAL을 사용하여 자동으로 입력할 수 있습니다.

- CURRVAL : 현재 값을 반환 합니다. .
- NEXTVAL : 현재 시퀀스값의 다음 값을 반환 합니다.

```
SQL>SELECT emp_seq.CURRVAL FROM DUAL ;  
      CURRVAL  
      -----  
              1
```

```
SQL>SELECT emp_seq.NEXTVAL FROM DUAL ;  
      NEXTVAL  
      -----  
              2
```

시퀀스 사용 규칙

- ◆ NEXTVAL, CURRVAL을 사용할 수 있는 경우
 - subquery가 아닌 select문
 - insert문의 select절
 - insert문의 value절
 - update문의 set절
- ◆ NEXTVAL, CURRVAL을 사용할 수 없는 경우
 - view의 select절
 - group by, having, order by절이 있는 select문
 - create table, alter table 명령의 default값
 - distinct 키워드가 있는 select문
 - select, delete, update의 subquery

9.3. SEQUENCE(시퀀스)

시퀀스의 수정 및 삭제

```
ALTER SEQUENCE sequence_name  
[INCREMENT BY n]  
[MAXVALUE n | NOMAXVALUE]  
[MINVALUE n | NOMINVALUE]  
[CYCLE | NOCYCLE]
```

- ◆ **START WITH**는 수정할 수 없습니다.
- ◆ **START WITH** 절이 없다는 점을 빼고는 **CREATE SEQUENCE**와 같습니다.

```
SQL>ALTER SEQUENCE emp_seq  
      INCREMENT BY 2  
      CYCLE;
```

➔ 2씩 증가하고, 최대값을 넘으면 다시 처음부터 순환하도록 수정하였습니다.

- ◆ **DROP** 문으로 필요하지 않은 시퀀스는 삭제 할수 있습니다.

```
SQL>DROP SEQUENCE PRD_SEQ;  
sequence dropped.
```


9. 오라클 객체

9.4. SYNONYM(동의어)

시노님 이란?

- ◆ 시노님은 오라클 객체(테이블, 뷰, 시퀀스, 프로시저)에 대한 대체이름(Alias)를 말합니다.
- ◆ Synonym은 실질적으로 그 자체가 Object가 아니라 Object에 대한 직접적인 참조입니다.
- ◆ 시노님을 사용하는 이유는..
 - ① 데이터베이스의 투명성을 제공하기 위해서 사용 한다고 생각하면 됩니다. 시노님은 다른 유저의 객체를 참조할 때 많이 사용을 합니다.
 - ② 만약에 다른 유저의 객체를 참조할 경우가 있을 때 시노님을 생성해서 사용을 하면은 추후에 참조하고 있는 오브젝트가 이름을 바꾸거나 이동할 경우 객체를 사용하는 SQL문을 모두 다시 고치는 것이 아니라 시노님만 다시 정의하면 되기 때문에 매우 편리 합니다.
 - ③ 객체의 긴 이름을 사용하기 편한 짧은 이름으로 해서 SQL코딩을 단순화 시킬 수 있습니다.
 - ④ 또한 객체를 참조하는 사용자의 오브젝트를 감추 수 있기 때문에 이에 대한 보안을 유지할 수 있습니다.
시노님을 사용하는 유저는 참조하고 있는 객체에 대한 사용자의 object의 소유자, 이름, 서버이름을 모르고 시노님 이름만 알아도 사용 할 수 있습니다.

시노님을 사용하는 이유

- ◆ 오브젝트의 실제 이름과 소유자 그리고 위치를 감춤으로써 database 보안을 개선하는데 사용 됩니다
- ◆ Object에의 Public Access를 제공 합니다.
- ◆ Remote Database의 Table, View, Program Unit를 위해 투명성을 제공 합니다.
- ◆ Database 사용자를 위해 SQL 문을 단순화 할 수 있습니다.

9. 오라클 객체

9.4. SYNONYM(동의어)

◆ 시노님에는 두가지 종류가 있습니다.

- ① Private Synonym : 전용 시노님은 특정 사용자만 이용할수 있습니다.
- ② Public Synonym : 공용 시노님은 공용 사용자 그룹이 소유하며 그 Database에 있는 모든 사용자가 공유 합니다.

👉 시노님 생성 문법

```
CREATE [PUBLIC] SYNONYM synonym_name  
FOR object_name
```

scott USER의 emp테이블을 test USER가 사용 하는 예제.

1. 먼저 scott/tiger USER로 접속해서 test USER에게 emp테이블을 조작할 권한을 부여합니다.

```
SQL>GRANT ALL ON emp TO test;
```

➔ test user에 대하여 scott의 emp테이블을 조작할 수 있는 권한을 부여합니다.

2. test USER로 접속해 동의어를 생성합니다.

```
SQL> connect test/test
```

```
SQL> CREATE SYNONYM scott_emp FOR scott.emp ;
```

➔ scott USER가 소유하고 있는 emp 테이블에 대해 scott_emp라는 일반 시노님을 생성했습니다.
scott 사용자의 emp테이블을 test 사용자가 scitt_emp라는 동의어로 사용 합니다. .

-- 시노님을 이용한 쿼리

```
SQL> SELECT empno, ename FROM scott_emp;
```

➔ 이 두 쿼리의 결과는 같습니다.

-- 일반 테이블을 쿼리

```
SQL> SELECT empno, ename FROM scott.emp;
```

동의어 삭제 ➔ SQL> DROP SYNONYM scott_emp