

SQL

관계 대수 와 관계 해석

❖ 관계 대수(Relational Algebra)

- ✓ 관계형 데이터베이스에서 원하는 정보와 그 정보를 검색하기 위해서 어떻게 유도하는가를 기술하는 절차적인 언어
- ✓ 질의에 대한 해를 구하기 위해 수행해야 할 연산의 순서를 명시하며, 릴레이션을 처리하기 위해 연산자 와 연산 규칙을 제공
- ✓ 일반 집합 연산자
 - 합집합(\cup)
 - 교집합(\cap)
 - 차집합($-$),
 - 카티션 프로덕트(\times) : 속성의 수는 더하기 튜플의 수는 곱하기
- ✓ 순수 관계 연산자
 - Select(σ): 선택 조건을 만족하는 릴레이션의 수평적 부분집합
 - Project(π): 릴레이션의 수직적 부분집합, 중복 튜플의 생성은 제거
 - Join(\bowtie): 세타 조인, 동일 조인, 자연 조인, 자연 조인은 동일 조인의 결과 릴레이션에서 중복되는 조인 애트리뷰트를 제거하는 경우와 동일
 - Division(\div): 속성을 제외하고 나머지 속성만 추출해내는 기능

관계 대수 와 관계 해석

❖ 관계 대수(Relational Algebra)

1. 마당 서점에서 판매하는 도서 중 8,000원 이하인 도서를 검색하시오.

$$\sigma_{(가격 \leq 8000)}(\text{도서})$$

2. 신간도서 안내를 위해 카달로그 발송 주소록을 만드시오.

$$\Pi_{\text{이름, 주소, 핸드폰}}(\text{고객})$$

3. 마당서점은 지점A와 지점B가 있다. 두 지점의 도서는 각 지점에서 관리하며 릴레이션 이름은 각각 도서A, 도서B이다. 마당 서점의 도서를 하나의 릴레이션으로 보이시오.

$$\text{도서A} \cup \text{도서B}$$

4. 마당 서점의 두 지점에서 동일하게 보유하고 있는 도서 목록을 보이시오.

$$\text{도서A} \cap \text{도서B}$$



관계 대수 와 관계 해석

❖ 관계 대수(Relational Algebra)

6. 도서 릴레이션과 고객 릴레이션의 카티전 프로덕트를 구하시오.

도서A×도서B

7. 고객과 고객의 주문사항을 모두 보이시오.

고객 ⋈_{고객.주문번호=주문.주문번호} 주문

8. 마당 서점의 고객과 고객의 주문 내역을 보이시오.

(단, 고객 기준으로 주문 내역이 없는 고객도 모두 보이시오.)

고객 ⋈(고객.고객번호,주문.고객번호)주문

9. 마당 서점의 고객과 고객의 주문 내역을 보이시오.

(단, 주문 내역이 없는 고객과 고객 릴레이션에 고객번호가 없는 주문을 모두 보이시오.)

고객 ⋈(고객.고객번호,주문.고객번호)주문

10. 마당 서점의 고객과 고객의 주문 내역을 보이시오.

(단, 주문 내역 기준으로 고객 릴레이션에 고객번호가 없는 주문도 모두 보이시오.)

고객 ⋈(고객.고객번호,주문.고객번호)주문

관계 대수 와 관계 해석

❖ 관계 대수(Relational Algebra)

11. 마당 서점의 고객 중 주문 내역이 있는 고객의 고객 정보를 보이시오.

$$\pi_{\text{고객}} \bowtie_{\text{고객,고객번호, 주문,고객번호}} \text{주문}$$

12. 마당 서점의 도서 중 가격이 8,000원 이하인 도서 이름과 출판사를 모두 보이시오.

$$\pi_{\text{도서이름,출판사}} (\sigma_{\text{가격} \leq 8000} \text{도서})$$

13. 마당 서점의 박지성 고객의 거래 내역 중 주문 번호, 이름, 가격을 보이시오. (카티전프로덕트, 조인)

- 카티전프로덕트 : $\pi_{\text{주문,주문번호, 고객,이름, 주문,판매가격}} (\sigma_{\text{고객,고객번호=주문,고객번호 and 고객,이름 = '박지성'}} (\text{고객} \times \text{주문}))$

- 조인 : $\pi_{\text{주문번호, 이름, 판매가격}} (\sigma_{\text{이름='박지성'}} (\text{고객} \bowtie_{\text{고객,고객번호=주문,고객번호}} \text{주문}))$

관계 해석

❖ 관계 해석(Relational Calculus)

- ✓ 프레디키트 해석(predicate calculus)으로 질의어를 표현
- ✓ 원하는 릴레이션을 정의하는 방법을 제공하며 비절차적(non-procedural)인 언어
- ✓ 튜플 관계 해석과 도메인 관계 해석의 두 종류
- ✓ 샘플
 - 셀렉트 연산 : $\{ t \mid \text{EMPLOYEE}(t) \text{ and } t.\text{SALARY} > 5000 \}$
 - 프로젝트 연산 : $\{ t.\text{FNAME}, t.\text{LNAME} \mid \text{EMPLOYEE}(t) \}$
 - 셀렉트 + 프로젝트 : $\{ t.\text{FNAME}, t.\text{LNAME} \mid \text{EMPLOYEE}(t) \text{ and } t.\text{SALARY} > 50000 \}$



SQL

❖ SQL

- ✓ 1974년 IBM 연구소에서 개발한 SEQUEL에서 유래
- ✓ 국제 표준 데이터베이스 언어이며, 많은 회사에서 관계형 데이터베이스(RDB)를 지원하는 언어로 채택
- ✓ 관계 대수와 관계 해석을 기초로 한 혼합 데이터 언어
- ✓ 질의어지만 질의 기능만 있는 것이 아니라 정의, 조작, 제어 기능을 모두 갖추고 있음



SQL

❖ SQL의 분류

구 분	형 식	비 고
DQL (Data Query Language)	SELECT column-1, column-2, ... FROM table명 WHERE 조건절 ;	검색시 사용
DML (Data Manipulation Language)	UPDATE table명; INSERT INTO table명; DELETE table명;	변경시 사용
DDL (Data Definition Language)	CREATE TABLE table명; DROP TABLE table명; ALTER TABLE table명;	Object의 생성과 변경 시
TCL (Transaction Control Language)	COMMIT; ROLLBACK; SAVEPOINT;	Transaction 종료 및 취소
DCL (Data Control Language)	GRANT ...; REVOKE;	권한 부여 및 취소

DDL

❖ DDL

- ✓ DB 구조, 데이터 형식, 접근 방식 등 DB를 구축하거나 수정할 목적으로 사용하는 언어
- ✓ DDL은 번역한 결과가 데이터 사전(Data Dictionary)이라는 특별한 파일에 여러 개의 테이블로서 저장
- ✓ 대상
 - 스키마(Schema)
 - DBMS 특성과 구현 환경을 감안한 데이터 구조
 - 직관적으로 하나의 데이터베이스로 이해 가능
 - 도메인(Domain):
 - 속성의 데이터 타입과 크기, 제약 조건 등을 지정한 정보
 - 속성이 가질 수 있는 값의 범위로 이해 가능 공간
 - 테이블(Table): 데이터 저장 공간
 - 뷰(View): 하나 이상의 물리 테이블에서 유도되는 가상의 논리 테이블
 - 인덱스(Index): 검색을 빠르게 하기 위한 데이터 구조
- ✓ 명령어
 - ❑ 생성 : CREATE
 - ❑ 변경 : ALTER
 - ❑ 삭제 : DROP, TRUNCATE

데이터베이스 개체

구분	
Schema	테이블을 모아서 관리하기 논리적인 저장 단위 - 데이터베이스라고도 함
Domain	테이블에 저장할 수 있는 자료형이나 값에 붙인 별명
Table	데이터를 저장하는 개체
View	자주 사용하는 SELECT 구문을 저장해서 테이블 처럼 사용하는 개체
Index	데이터를 빠르게 찾을 수 있도록 해주는 개체
Synonym	개체에 대한 별명
Sequence	일련번호 - MySQL에는 없고 유사한 역할을 Auto_InCrement 가 수행

데이터베이스 개체

구 분	형 식
Anonymous Procedure	자주 사용하는 DML 문장을 하나로 묶은 개체 Declare ~ Begin ~ Exception ~ End;
Stored Procedure	Create or Replace Procedure [프로시저명] Begin Exception End;
Stored Function	연산의 결과를 반환하는 개체 Create or Replace Function [함수명] Return Begin Exception End;
Package	같이 사용되는 함수나 프로시저를 묶은 개체 Create or Replace Package [패키지명] Begin ~ End; Create or Replace Package Body Begin ~ End;
Trigger	DML 문장을 수행하기 전이나 후에 수행되는 문장 Create or Replace Trigger [트리거명] AFTER [BEFORE] [조건] Begin End;

DDL

❖ 스키마 생성

CREATE SCHEMA 스키마이름 AUTHORIZATION 사용자아이디

✓ 데이터베이스 제품에 따라 다름 – SCHEMA 대신에 DATABASE 를 사용하기도 함

❖ 도메인 생성

CREATE DOMAIN 도메인이름 AS 데이터타입

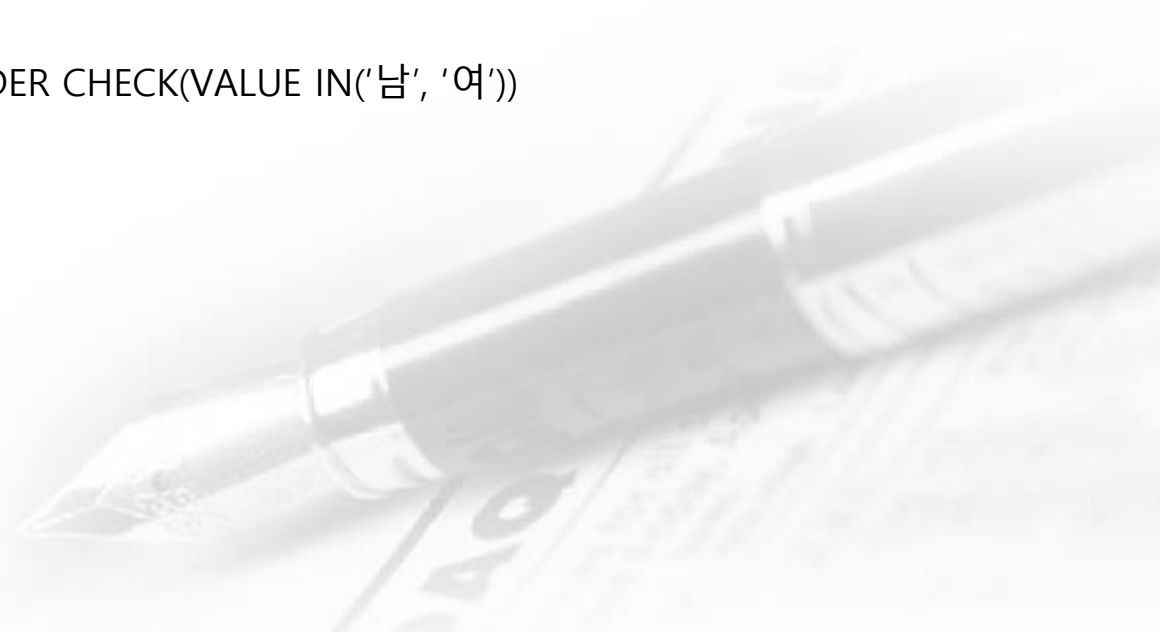
[DEFAULT 기본값]

[CONSTRAINT 제약조건명 CHECK (범위값)]

CREATE DOMAIN GENDER CHAR(3)

DEFAULT('남')

CONSTRAINT VALID_GENDER CHECK(VALUE IN('남', '여'))



DDL

❖ CREATE TABLE(테이블 생성) 문의 기본 형식

CREATE TABLE 테이블 이름

(컬럼 이름 data_type 제약조건, ...);

- ✓ 테이블 이름은 객체를 의미할 수 있는 적절한 이름을 사용 - 가능한 단수형을 권고
- ✓ 테이블 이름은 다른 테이블의 이름과 중복되지 않아야 한다.
- ✓ 한 테이블 내에서는 컬럼 이름이 중복되게 지정될 수 없다.
- ✓ 각 컬럼들은 콤마 ","로 구분되고, 테이블 생성 문의 끝은 항상 세미콜론 ";"으로 끝난다.
- ✓ 컬럼에 대해서는 다른 테이블까지 고려하여 데이터베이스 내에서는 일관성 있게 사용하는 것이 좋다.(데이터 표준화 관점 : 공통성 컬럼을 도메인으로 관리)
- ✓ 컬럼 뒤에 데이터 유형은 꼭 지정되어야 한다.
- ✓ 테이블 이름과 컬럼 이름은 문자로 시작해야 하고, 벤더 별로 길이에 대한 한계가 있다.
- ✓ 벤더에서 사전에 정의한 예약어(Reserved word)는 쓸 수 없다.
- ✓ A-Z, a-z, 0-9, _, \$, # 문자만 허용된다.
- ✓ 실제 DBMS는 팀 테이블의 TEAM_ID를 PC나 UNIX의 디렉토리 구조처럼 'DB명+DB사용자명+테이블명+컬럼명'처럼 계층적 구조를 가진 전체 경로로 관리하고 있다.
- ✓ 같은 이름을 가진 컬럼들은 기본키와 외래키의 관계를 가지는 경우가 많으며 향후 테이블 간의 조인 조건으로 주로 사용되는 중요한 연결고리 컬럼들이다.

DDL

❖ 테이블 구조 변경

- ✓ 열 추가: ALTER TABLE 테이블이름 ADD 열이름 데이터타입 [DEFAULT 값]
- ✓ 열 데이터 타입 변경: ALTER TABLE 테이블이름 MODIFY 열이름 데이터타입 [DEFAULT 값]
- ✓ 열 삭제: ALTER TABLE 테이블이름 DROP 열이름

❖ 테이블 삭제, 절단, 이름 변경

- ✓ 테이블 삭제: DROP TABLE 테이블이름
- ✓ 테이블 내용 삭제: TRUNCATE TABLE 테이블이름
- ✓ 테이블 이름 변경
 - RENAME TABLE 이전테이블이름 TO 새로운테이블이름
 - ALTER TABLE 이전테이블이름 RENAME 새로운테이블이름



DDL

❖ 제약조건

✓ 제약 조건 유형

- PRIMARY KEY
 - 테이블의 기본키를 정의함.
 - 기본으로 NOT NULL, UNIQUE 제약이 포함됨.
- FOREIGN KEY
 - 외래키를 정의함.
 - 참조 대상을 테이블이름(열이름)으로 명시해야 함.
 - 참조 무결성 위배 상황 발생 시 처리 방법으로 옵션 지정 가능 - NO ACTION, SET DEFAULT, SET NULL, CASCADE
- UNIQUE
 - 테이블 내에서 열은 유일한 값을 가져야 함.
 - 테이블 내에서 동일한 값을 가져서는 안 되는 항목에 지정함.
- NOT NULL
 - 테이블 내에서 관련 열의 값은 NULL일 수 없음
 - 필수 입력 항목에 대해 제약 조건으로 설정함
- CHECK: 개발자가 정의하는 제약 조건 상황에 따라 다양한 조건 설정 가능

DDL

❖ 제약조건

✓ 외래키 옵션

- NO ACTION: 아무런 조치를 하지 않음
- CASCADE: 같이 삭제되거나 변경
- SET NULL: NULL로 설정
- SET DEFAULT: 기본값으로 설정

- ✓ 제약 조건 활용: 테이블 생성을 위한 CREATE 문에 제약 조건을 명시하는 형태로 사용하며, ALTER를 통해 테이블의 제약 조건을 변경할 수 있다.



테이블 생성

- ❖ 테이블명 : PLAYER
- ❖ 테이블 설명 : K-리그 선수들의 정보를 가지고 있는 테이블
- ❖ 칼럼명
 - ✓ PLAYER_ID (선수ID) 문자 고정 자릿수 7자리,
 - ✓ PLAYER_NAME (선수명) 문자 가변 자릿수 20자리,
 - ✓ TEAM_ID (팀ID) 문자 고정 자릿수 3자리,
 - ✓ E_PLAYER_NAME (영문선수명) 문자 가변 자릿수 40자리,
 - ✓ NICKNAME (선수별명) 문자 가변 자릿수 30자리,
 - ✓ JOIN_YYYY (입단년도) 문자 고정 자릿수 4자리,
 - ✓ POSITION (포지션) 문자 가변 자릿수 10자리,
 - ✓ BACK_NO (등번호) 숫자 2자리,
 - ✓ NATION (국적) 문자 가변 자릿수 20자리,
 - ✓ BIRTH_DATE (생년월일) 날짜,
 - ✓ SOLAR (양/음) 문자 고정 자릿수 1자리,
 - ✓ HEIGHT (신장) 숫자 3자리,
 - ✓ WEIGHT (몸무게) 숫자 3자리,



테이블 생성

❖ Oracle

```
CREATE TABLE PLAYER (  
  PLAYER_ID CHAR(7),  
  PLAYER_NAME VARCHAR2(20),  
  TEAM_ID CHAR(3),  
  E_PLAYER_NAME VARCHAR2(40),  
  NICKNAME VARCHAR2(30),  
  JOIN_YYYY CHAR(4),  
  POSITION VARCHAR2(10),  
  BACK_NO NUMBER(2),  
  NATION VARCHAR2(20),  
  BIRTH_DATE DATE,  
  SOLAR CHAR(1),  
  HEIGHT NUMBER(3),  
  WEIGHT NUMBER(3));
```



새로운 컬럼 추가

- ❖ ALTER TABLE ADD 문은 기존 테이블에 새로운 컬럼을 추가
- ❖ 새로운 컬럼은 테이블 맨 마지막에 추가되므로 자신이 원하는 위치에 만들어 넣을 수 없음
- ❖ 이전에 추가해 놓은 데이터가 존재한다면 그 데이터에도 컬럼이 추가되지만 컬럼의 값은 NULL 값으로 설정

ALTER TABLE table_name

ADD (column_name, data_type expr, ...);



새로운 칼럼 추가

- ❖ EMP01 테이블에 문자 타입의 직급(JOB) 칼럼을 추가

```
ALTER TABLE EMP01  
ADD(JOB VARCHAR2(9));
```

	123 EMPNO ↑↓	ABC ENAME ↑↓	123 SAL ↑↓	ABC JOB ↑↓



기존 컬럼 속성 변경하기

- ❖ ALTER TABLE MODIFY 문을 다음과 같은 형식으로 사용하면 테이블에 이미 존재하는 컬럼을 변경할 수 있음

```
ALTER TABLE table_name  
MODIFY (column_name, data_type expr, ...);
```












- ❖ 컬럼을 변경한다는 것은 컬럼에 대해서 데이터 타입이나 크기, 기본 값들을 변경한다는 의미



기존 칼럼 속성 변경하기

- ❖ emp01 테이블의 직급(JOB) 칼럼을 최대 30글자까지 저장할 수 있게 변경

```
ALTER TABLE EMP01  
MODIFY(JOB VARCHAR2(30));
```

	컬럼명	#	Type	Type Mod	Not Null	디폴트	Comment
 Columns	123 EMPNO	1	NUMBER(4,0)		<input type="checkbox"/>		
 Constraints	ABC ENAME	2	VARCHAR2(20)		<input type="checkbox"/>		
 Foreign Keys	123 SAL	3	NUMBER(7,2)		<input type="checkbox"/>		
 References	ABC JOB	4	VARCHAR2(9)		<input type="checkbox"/>		
 Triggers							
 Indexes							
 Partitions							
 Privileges							
 Statistics / ...							
 DDL							
 Virtual							












기존 칼럼 삭제

- ❖ 테이블에 이미 존재하는 컬럼을 삭제
- ❖ ALTER TABLE ~ DROP COLUMN 명령어로 칼럼을 삭제

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

- ❖ EMP01 테이블의 직급 칼럼을 삭제

```
ALTER TABLE EMP01  
DROP COLUMN JOB;
```

	컬럼명	#	Type	Type Mod	Not Null	디폴트	Comment
 Columns	123 EMPNO	1	NUMBER(4,0)		<input type="checkbox"/>		
 Constraints	ABC ENAME	2	VARCHAR2(20)		<input type="checkbox"/>		
 Foreign Keys	123 SAL	3	NUMBER(7,2)		<input type="checkbox"/>		
 References							
 Triggers							
 Indexes							
 Partitions							
 Privileges							
 Statistics / ...							
 DDL							
 Virtual							

DDL

❖ 뷰 생성 및 삭제

- ✓ 생성: CREATE VIEW 뷰명[(속성명[, 속성명, ...])] AS SELECT문;
 - 속성 명을 생략하면 SELECT 구문의 속성 명을 사용
 - SELECT 문에 UNION 이나 ORDER BY를 사용할 수 없음
- ✓ 삭제: DROP VIEW 뷰명;

❖ 자주 사용되는 30번 부서에 소속된 직원들의 사번과 이름과 부서번호를 출력하기 위한 SELECT문을 하나의 뷰로 정의

```
CREATE VIEW EMP_VIEW30
AS
SELECT EMPNO, ENAME, DEPTNO
FROM EMP_COPY
WHERE DEPTNO=30;
```

❖ 뷰는 테이블에 접근(SELECT)한 것과 동일한 방법으로 결과를 얻을 수 있음

```
SELECT * FROM EMP_VIEW30;
```

❖ 뷰 구조 확인

```
DESC EMP_VIEW30;
```



DDL

❖ 인덱스 생성 및 삭제

✓ 생성

```
CREATE [UNIQUE] INDEX 인덱스명  
ON 테이블명(속성명[ASC|DESC] [, 속성명[ASC|DESC]])  
[CLUSTER] ;
```

○ UNIQUE 사용되면 중복 값이 없는 속성으로 생성

✓ 삭제: DROP INDEX 인덱스명;

✓ 수정: ALTER [UNIQUE] INDEX <index name> ON <table name> (<column(s)>);

✓ 테이블 EMP의 컬럼 중에서 이름(ENAME)에 대해서 인덱스를 생성

```
CREATE INDEX IDX_EMP_ENAME  
ON EMP(ENAME);
```

✓ EMP01 테이블의 IDX_EMP_ENAME 인덱스를 제거

```
DROP INDEX IDX_EMP_ENAME;
```



DCL

- ❖ DCL 유형
 - ✓ 사용자 권한: 접근 통제가 목적으로 사용자를 등록하고, 사용자에게 특정 데이터베이스를 사용할 수 있는 권리를 부여하는 작업
 - ✓ 트랜잭션: 안전한 거래 보장을 목적으로 동시에 다수의 작업을 독립적으로 안전하게 처리 하기 위한 상호 작용 단위
- ❖ 트랜잭션 제어를 위한 명령어 TCL(Transaction Control Language)이 있다.
- ❖ TCL과 DCL은 대상이 달라 서로 별개의 개념으로 분류할 수 있으나 제어 기능의 공통점으로 DCL의 일부로 분류하기도 한다.
- ❖ DCL 명령어
 - ✓ DCL
 - GRANT : 데이터베이스 사용자 권한 부여
 - REVOKE: 데이터베이스 사용자 권한 회수
 - ✓ TCL
 - COMMIT: 트랜잭션 확정
 - ROLLBACK: 트랜잭션 취소
 - SAVEPOINT: rollback 지점 설정



DCL

❖ 사용자 권한 부여

- ✓ 시스템 권한: GRANT 권한1, 권한2 TO 사용자계정
- ✓ 객체 권한: GRANT 권한1, 권한2 ON 객체명 TO 사용자계정
- ✓ with grant Option을 추가하면 다른 사용자에게 권한 부여 가능

구분	권한	내용
시스템 권한	CREATE USER	계정 생성 권한
	DROP USER	계정 삭제 권한
	DROP ANY TABLE	테이블 삭제 권한
	CREATE SESSION	데이터베이스 접속 권한
	CREATE TABLE	테이블 생성 권한
	CREATE VIEW	뷰 생성 권한
	CREATE SEQUENCE	시퀀스 생성 권한
	CREATE PROCEDURE	함수 생성 권한
	ALTER	테이블 변경 권한
객체 권한	INSERT	데이터 조작 권한
	DELETE	
	SELECT	
	UPDATE	PROCEDURE 실행 권한
	EXECUTE	

DCL

❖ 사용자 권한 회수

- ✓ 시스템 권한: REVOKE 권한1, 권한2 FROM 사용자계정
- ✓ 객체 권한: REVOKE 권한1, 권한2 ON 객체명 FROM 사용자계정



DML


- ❖ 데이터를 조작하는 명령어가 DML(Data Manipulation Language)
- ❖ DML 유형
 - ✓ 데이터 생성: INSERT, 삽입 형태로 신규 데이터를 테이블에 저장
 - ✓ 데이터 조회: SELECT, 테이블의 내용을 조회
 - ✓ 데이터 변경: UPDATE, 테이블의 내용을 변경
 - ✓ 데이터 삭제: DELETE, 테이블의 내용을 삭제
- ❖ 데이터 삽입
 - ✓ INSERT INTO table_name (column1, column2, ..) VALUES (value1, value2, ...);
 - ✓ INSERT INTO table_name VALUES (value1, value2, ...);
- ❖ 데이터 수정(UPDATE)
 - ✓ UPDATE table SET column1 = value1, column2 = value2, ... [WHERE 절] ;
- ❖ 데이터 삭제(DELETE)
 - ✓ DELETE FROM table [WHERE 절] ;



DML

- ❖ 새로운 데이터를 추가하기 위해서 사용할 명령어 INSERT INTO ~ VALUES ~는 컬럼 명에 기술된 목록의 수와 VALUES 다음에 나오는 괄호에 기술한 값의 개수가 동일해야 함
- ❖ 컬럼 DEPTNO에 10을 컬럼 DNAME에는 'ACCOUNTING'을, 컬럼 LOC에는 'NEW YORK'을 추가

```
INSERT INTO DEPT01  
(DEPTNO, DNAME, LOC)  
VALUES(10, 'ACCOUNTING', 'NEW YORK');
```



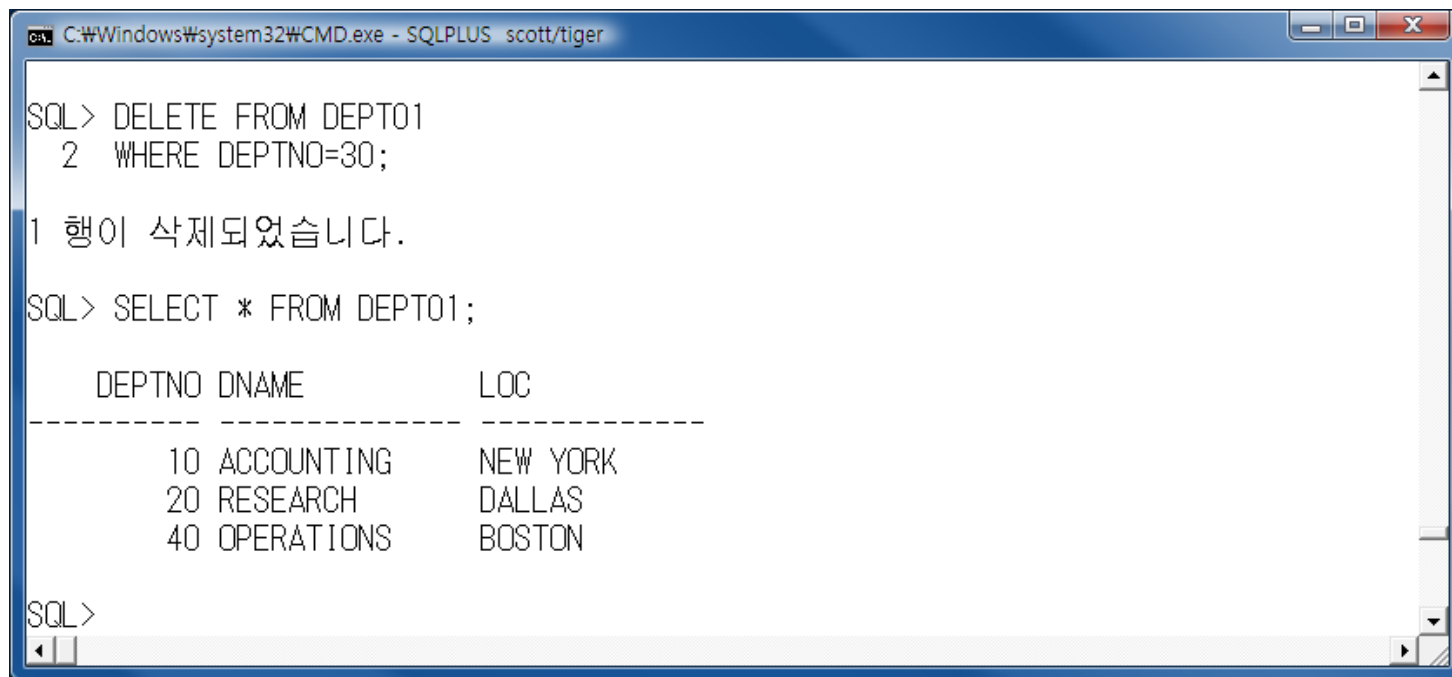
데이터 수정

- ❖ 모든 사원의 부서번호를 30번으로 수정
UPDATE EMP01
SET DEPTNO=30;
- ❖ 모든 사원의 급여를 10% 인상시키는 UPDATE
UPDATE EMP01
SET SAL = SAL * 1.1;
- ❖ 모든 사원의 입사일을 오늘로 수정
UPDATE EMP01
SET HIREDATE = SYSDATE;



데이터 삭제

- ❖ 부서 테이블에서 30번 부서만 삭제
DELETE FROM DEPT01
WHERE DEPTNO=30;



```
C:\Windows\system32\CMD.exe - SQLPLUS scott/tiger

SQL> DELETE FROM DEPT01
2  WHERE DEPTNO=30;

1 행이 삭제되었습니다.

SQL> SELECT * FROM DEPT01;

  DEPTNO DNAME          LOC
-----
10 ACCOUNTING      NEW YORK
20 RESEARCH        DALLAS
40 OPERATIONS      BOSTON

SQL>
```


DML

- ❖ 테이블 구조 조회: DESC 테이블명
- ❖ 데이터 조회: SELECT
SELECT [PREDICATE] [테이블명.]속성명 [AS 별칭][, [테이블 명.]속성명, ...] [, 그룹함수(속성명)
[AS 별칭]]
FROM 테이블명[, 테이블명, ...]
[WHERE 조건]
[GROUP BY 속성명, 속성명, ...]
[HAVING 조건]
[ORDER BY 속성명 [ASC | DESC]];



DML

❖ 데이터 조회: SELECT

- ✓ PREDICATE : 불러올 튜플 수를 제한할 명령어를 기술
 - ALL : 모든 튜플을 검색할 때 지정하는 것으로, 주로 생략
 - DISTINCT : 중복된 튜플이 있으면 그 중 첫번째 한 개만 검색함
 - DISTINCTROW : 중복된 튜플을 검색하지만 선택 된 속성의 값이 아닌 튜플 전체를 대상으로 함
- ✓ 속성명 : 검색하여 불러올 속성(열) 및 수식들을 지정함
 - 테이블을 구성하는 모든 속성을 지정할 때 는 '*' 를 기술함
 - 두 개 이상의 테이블에 공통으로 존재하는 속성은 '테이블명.속성명'으로 기술
 - AS : 속성 및 연산의 이름을 다른 제목으로 표시하 기 위해 사용함
- ✓ FROM절 : 질의에 의해 검색될 데이터들을 포함하는 테이블명을 기술함
- ✓ WHERE절 : 검색할 조건 기술함
- ✓ GROUP BY절 : 특정 속성을 기준으로 그룹화하여 검색할 때 그룹화 할 속성을 지정함
 - 일반적으로 GROUP BY절은 그룹 함수와 함께 사용 됨
 - 그룹 함수의 종류
 - COUNT(속성명) : 속성 대신에 * 사용 가능
 - MAX(속성명)
 - MIN(속성명)
 - SUM(속성명)
 - AVG(속성명)

DML

❖ 데이터 조회: SELECT

- ✓ HAVING절: GROUP BY와 함께 사용되며, 그룹에 대한 조건을 지정함
- ✓ ORDER BY절: 특정 속성을 기준으로 정렬하여 검색할 때 사용함
 - 속성명 : 정렬의 기준이 되는 속성명을 기술함
 - [ASC | DESC] : 정렬 방식으로서 'ASC'는 오름차순, 'DESC'는 내림차순 정렬을 수행하며 생략하면 오름차순으로 지정됨



SELECT

❖ 모든 열 선택

✓ SELECT 키워드에 "*" 을 사용하여 테이블의 열 데이터 모두를 조회

❖ SCOTT이 소유하고 있는 EMP Table의 모든 데이터를 조회

SELECT *

FROM emp;

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	80/12/17	800	(null)	20
2	7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
3	7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
4	7566	JONES	MANAGER	7839	81/04/02	2975	(null)	20
5	7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
6	7698	BLAKE	MANAGER	7839	81/05/01	2850	(null)	30
7	7782	CLARK	MANAGER	7839	81/06/09	2450	(null)	10
8	7788	SCOTT	ANALYST	7566	87/04/19	3000	(null)	20
9	7839	KING	PRESIDENT	(null)	81/11/17	5000	(null)	10
10	7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30
11	7876	ADAMS	CLERK	7788	87/05/23	1100	(null)	20
12	7900	JAMES	CLERK	7698	81/12/03	950	(null)	30
13	7902	FORD	ANALYST	7566	81/12/03	3000	(null)	20
14	7934	MILLER	CLERK	7782	82/01/23	1300	(null)	10

SELECT

- ❖ 특정 Column 선택
 - ✓ 테이블의 특정 Column을 검색하고자 할 경우 Column이름을 ","로 구분하여 명시함으로써 특정 Column을 조회
 - ✓ 조회 순서는 SELECT문 뒤에 기술한 Column의 순서대로 조회
- ❖ SCOTT이 소유하고 있는 EMP Table에서 empno, ename, sal, job 를 조회
SELECT empno, ename, sal, job
FROM emp;

	EMPNO	ENAME	SAL	JOB
1	7369	SMITH	800	CLERK
2	7499	ALLEN	1600	SALESMAN
3	7521	WARD	1250	SALESMAN
4	7566	JONES	2975	MANAGER
5	7654	MARTIN	1250	SALESMAN
6	7698	BLAKE	2850	MANAGER
7	7782	CLARK	2450	MANAGER
8	7788	SCOTT	3000	ANALYST
9	7839	KING	5000	PRESIDENT
10	7844	TURNER	1500	SALESMAN
11	7876	ADAMS	1100	CLERK
12	7900	JAMES	950	CLERK
13	7902	FORD	3000	ANALYST
14	7934	MILLER	1300	CLERK

SELECT

❖ 산술 표현식

- ✓ 데이터가 조회되는 방식을 수정하거나 계산을 수행하고자 할 때 산술 표현식을 사용
- ✓ 산술 표현식은 열 이름, 숫자 상수, 문자 상수, 산술 연산자를 포함할 수 있으며 연산자는 +(Add), -(Subtract), *(Multiply), /(Divide)을 사용
- ✓ SELECT문장에서는 FROM절을 제외한 절에서 사용할 수 있음
- ✓ 산술 표현식이 하나 이상의 연산자를 포함한다면 일반적인 산술 연산자 우선 순위를 적용
- ✓ 날짜는 하루를 숫자 1로 계산해서 덧셈과 뺄셈 가능

❖ emp 테이블의 Sal을 300증가 시키기 위해 덧셈 연산자를 사용하고 ename, sal, sal+300을 조회

```
SELECT ename, sal, sal+300  
FROM emp;
```

	ENAME	SAL	SAL+300
1	SMITH	800	1100
2	ALLEN	1600	1900
3	WARD	1250	1550
4	JONES	2975	3275
5	MARTIN	1250	1550
6	BLAKE	2850	3150
7	CLARK	2450	2750
8	SCOTT	3000	3300
9	KING	5000	5300
10	TURNER	1500	1800
11	ADAMS	1100	1400
12	JAMES	950	1250
13	FORD	3000	3300
14	MILLER	1300	1600

SELECT

❖ 주의

- ✓ 계산된 결과 열인 SAL+300은 EMP테이블의 새로운 열이 아니고 단지 디스플레이를 위한 것
- ✓ 디폴트로 새로운 열의 이름 sal+300은 생성된 계산식으로부터 유래
- ✓ SQL*Plus는 산술 연산자 앞뒤의 공백을 무시

❖ NULL 값의 처리

- ✓ 행이 특정 열에 대한 데이터 값이 없다면 값은 NULL
- ✓ NULL 값은 이용할 수 없거나 지정되지 않았거나, 알 수 없거나 또는 적용할 수 없는 값
- ✓ NULL 값은 0이나 공백과는 다르며 0은 숫자이며 공백은 문자
- ✓ 열이 NOT NULL로 정의되지 않았거나 열이 생성될 때 PRIMARY KEY로 정의되지 않았다면 열은 NULL 값을 포함할 수 있음
- ✓ NULL 값을 포함한 산술 표현식 결과는 NULL
- ✓ column에 데이터 값이 없으면 그 값 자체가 NULL 또는 NULL 값을 포함
- ✓ NULL 값은 1바이트의 내부 저장 장치를 오버헤드로 사용하고 있으며 어떠한 datatype의 column 들이라도 NULL 값을 포함할 수 있음

SELECT

- ❖ emp 테이블에서 empno, ename, sal, comm, sal+comm/100을 조회
SELECT empno, ename, sal, comm, sal+comm/100
FROM emp;

	EMPNO	ENAME	SAL	COMM	SAL+COMM/100
1	7369	SMITH	800	(null)	(null)
2	7499	ALLEN	1600	300	1603
3	7521	WARD	1250	500	1255
4	7566	JONES	2975	(null)	(null)
5	7654	MARTIN	1250	1400	1264
6	7698	BLAKE	2850	(null)	(null)
7	7782	CLARK	2450	(null)	(null)
8	7788	SCOTT	3000	(null)	(null)
9	7839	KING	5000	(null)	(null)
10	7844	TURNER	1500	0	1500
11	7876	ADAMS	1100	(null)	(null)
12	7900	JAMES	950	(null)	(null)
13	7902	FORD	3000	(null)	(null)
14	7934	MILLER	1300	(null)	(null)

SELECT

❖ NVL 함수

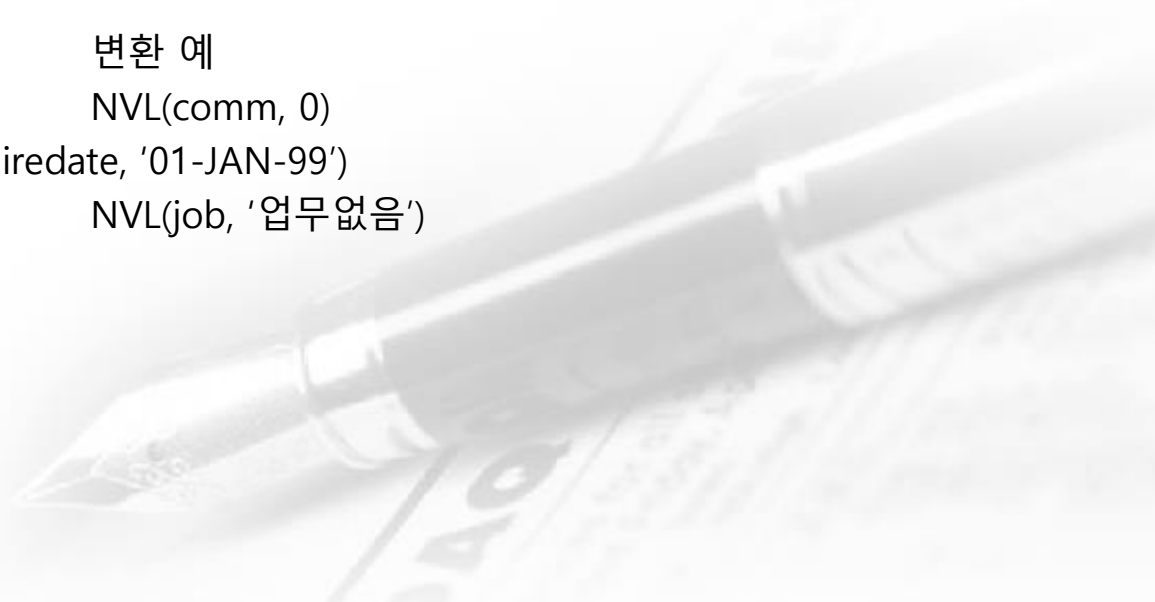
- ✓ NULL값을 특정한 값(실제 값)으로 변환하는데 사용
- ✓ 사용될 수 있는 데이터 타입은 날짜, 문자, 숫자
- ✓ NVL 함수를 사용할 때 전환되는 값은 컬럼의 데이터 타입을 준수

❖ Syntax

- ✓ NVL(expr1,expr2)
 - expr1 NULL 값을 포함하고 있는 Column이나 표현식
 - expr2 NULL 변환을 위한 목표 값

❖ 다양한 데이터형에 대한 NVL변형

- | | |
|------------------|----------------------------|
| ✓ 데이터형 | 변환 예 |
| NUMBER | NVL(comm, 0) |
| DATE | NVL(hiredate, '01-JAN-99') |
| CHAR or VARCHAR2 | NVL(job, '업무없음') |



SELECT

- ❖ emp 테이블에서 ename, sal, comm, sal*12+comm을 조회(단 comm이 NULL 이면 0로 계산)

```
SELECT ename, sal, comm, sal * 12 + NVL(comm,0)
FROM emp;
```

R2	ENAME	R2	SAL	R2	COMM	R2	SAL*12+NVL(COMM,0)
1	SMITH		800		(null)		9600
2	ALLEN		1600		300		19500
3	WARD		1250		500		15500
4	JONES		2975		(null)		35700
5	MARTIN		1250		1400		16400
6	BLAKE		2850		(null)		34200
7	CLARK		2450		(null)		29400
8	SCOTT		3000		(null)		36000
9	KING		5000		(null)		60000
10	TURNER		1500		0		18000
11	ADAMS		1100		(null)		13200
12	JAMES		950		(null)		11400
13	FORD		3000		(null)		36000
14	MILLER		1300		(null)		15600

SELECT

- ❖ 열에 별칭(Alias) 부여: 질의의 결과를 조회할 때 보통 SQL*Plus는 열 Heading으로 선택된 열 이름을 사용하는데 이 Heading은 때로 사용자가 이해하기가 어려운 경우가 있기 때문에 열 Heading을 변경하여 질의 결과를 조회하면 보다 쉽게 사용자가 이해할 수 있음
- ❖ 열 별칭(Alias) 정의
 - ✓ 열 Heading 이름을 변경
 - ✓ 계산식을 조회하는 경우에 유용
 - ✓ SELECT 절에서 열 이름 바로 뒤에 입력
 - ✓ 열 이름과 별칭 사이에 키워드 AS를 넣을 수 있음
 - ✓ 공백이나 특수 문자 또는 대문자가 있으면 이중 인용부호(" ")가 필요



SELECT

- ❖ EMP 테이블에서 ENAME를 이름으로 SAL을 급여로 해서 2개 Column의 모든 데이터를 조회

```
SELECT ename AS 이름, sal 급여  
FROM emp
```

	이름	급여
1	SMITH	800
2	ALLEN	1600
3	WARD	1250
4	JONES	2975
5	MARTIN	1250
6	BLAKE	2850
7	CLARK	2450
8	SCOTT	3000
9	KING	5000
10	TURNER	1500
11	ADAMS	1100
12	JAMES	950
13	FORD	3000
14	MILLER	1300



SELECT

- ❖ 특별히 명시되지 않았다면, SQL*Plus는 중복되지는 행을 제거하지 않고 Query 결과를 조회
- ❖ 결과에서 중복되는 행을 제거하기 위해서는 SELECT절에서 컬럼 이름 앞에 DISTINCT를 기술
- ❖ DISTINCT라는 키워드는 항상 SELECT 바로 다음에 기술
- ❖ DISTINCT뒤에 나타나는 컬럼들은 모두 DISTINCT의 영향을 받음
- ❖ DISTINCT뒤에 여러 개의 컬럼을 기술하였을 때 나타나는 행은 컬럼의 조합들이 중복되지 않게 조회



SELECT


- ❖ EMP 테이블에서 JOB을 모두 조회
SELECT job
FROM emp;

	Q2	JOB
1		CLERK
2		SALESMAN
3		SALESMAN
4		MANAGER
5		SALESMAN
6		MANAGER
7		MANAGER
8		ANALYST
9		PRESIDENT
10		SALESMAN
11		CLERK
12		CLERK
13		ANALYST
14		CLERK



SELECT

- ❖ EMP 테이블에서 JOB을 중복을 제거하고 조회
SELECT DISTINCT job
FROM emp;

 JOB
1 CLERK
2 SALESMAN
3 PRESIDENT
4 MANAGER
5 ANALYST



SELECT

- ❖ EMP 테이블에서 empno별로 job를 한번씩 조회
SELECT DISTINCT deptno,job
FROM emp;

	DEPTNO	JOB
1	20	CLERK
2	30	SALESMAN
3	20	MANAGER
4	30	CLERK
5	10	PRESIDENT
6	30	MANAGER
7	10	CLERK
8	10	MANAGER
9	20	ANALYST



SELECT

- ❖ 일반적인 경우 테이블에 있는 모든 자료를 조회할 필요없이 사용자가 원하는 자료를 조회하는 경우가 대부분이며 이러한 질의를 만족하게 하는 것이 WHERE
- ❖ WHERE절은 수행될 조건 절을 포함하며 FROM절 바로 다음에 기술

- ❖ Syntax

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

- ✓ WHERE 행들에 대한 조건을 기술하는 절
 - condition 검색할 조건을 입력: colum명, 표현식, 문자 상수, 숫자 상수, 비교 연산자로 구성
- ✓ ORDER BY 질의 결과 정렬을 위한 옵션(ASC:오름차순(Default),DESC내림차순)



SELECT

❖ WHERE절에 사용되는 연산자

- ✓ WHERE절을 사용하여 행들을 제한
- ✓ WHERE절은 FROM절 다음에 위치
- ✓ 조건은 아래의 것으로 구성
 - column 명, 표현식, 상수
 - 비교 연산자, SQL연산자, 논리연산자
 - 문자(Literal)는 작은 따옴표 안에 표기
 - 날짜도 숫자로 인식하므로 크기 비교 연산 가능

❖ 비교 연산자

연산자	의미
=	같다
>	보다 크다
>=	보다 크거나 같다
<	보다 작다
<=	보다 작거나 같다
<>, !=, ^=	같지 않다
NOT Column_name =	같지 않다
NOT Column_name >	보다 크지 않다

SELECT

- ❖ EMP 테이블에서 sal이 3000 이상인 사원의 empno, ename, job, sal을 조회

```
SELECT empno, ename, job, sal  
FROM emp  
WHERE sal >= 3000;
```

	EMPNO	ENAME	JOB	SAL
1	7839	KING	PRESIDENT	5000
2	7902	FORD	ANALYST	3000



SELECT

- ❖ EMP 테이블에서 job이 MANAGER 인 사원의 empno, ename, job, sal, deptno을 조회

```
SELECT empno, ename, job, sal, deptno  
FROM emp  
WHERE job = 'MANAGER'
```

Results:

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7566	JONES	MANAGER	2975	20
2	7698	BLAKE	MANAGER	2850	30
3	7782	CLARK	MANAGER	2450	10



SELECT

- ❖ EMP 테이블에서 hiredate가 1982년 01월 01일 이후 인 사원의 empno, ename, job, sal, hiredate, deptno 을 조회
 - ✓ 날짜 - to_date('2008/04/14 22:02:14', 'yyyy/mm/dd hh24:mi:ss')
 - ✓ 오늘 날짜 및 시간 - sysdate

```
SELECT empno, ename, job, sal, hiredate, deptno
FROM emp
WHERE hiredate >= to_date('1982/01/01', 'yyyy/mm/dd')
```

	EMPNO	ENAME	JOB	SAL	HIREDATE	DEPTNO
1	7934	MILLER	CLERK	1300	82/01/23	10

SELECT

❖ SQL연산자

연산자

설 명

AND

두 가지 조건을 모두 만족할 경우에만 검색

OR

두 가지 조건 중 하나라도 만족하는 경우에만 검색

BETWEEN a AND b

a 와 b 사이(a, b값 포함)

IN (list)

list의 값 중 어느 하나와 일치

LIKE

패턴과 일치(%,_사용)

IS NULL

NULL 값 일치

NOT BETWEEN a AND b

a 와 b사이에 있지 않음(a, b값 포함하지 않음)

NOT IN (list)

list의 값과 일치하지 않음

NOT LIKE

패턴과 일치하지 않음

IS NOT NULL

NULL과 일치하지 않음

✓ BETWEEN 연산자

- 두 값의 범위에 해당하는 행을 조회하기 위해 사용
- 작은 값을 앞에 기술하고 큰 값은 뒤에 기술

SELECT

- ❖ EMP 테이블에서 sal이 1300에서 1500 인 사원의 ename, job, sal, deptno 을 조회

```
SELECT ename,job,sal,deptno  
FROM emp  
WHERE sal BETWEEN 1300 AND 1500;
```

Results:

	ENAME	JOB	SAL	DEPTNO
1	TURNER	SALESMAN	1500	30
2	MILLER	CLERK	1300	10



SELECT

- ❖ IN 연산자: 목록에 있는 값에 대해서 조회하기 위해 IN 연산자를 사용
- ❖ EMP 테이블에서 empno가 7902,7788,7566인 사원의 empno, ename, job, sal, hiredate를 조회

```
SELECT empno, ename, job, sal, hiredate  
FROM emp  
WHERE empno IN (7902,7788,7566);
```

	EMPNO	ENAME	JOB	SAL	HIREDATE
1	7566	JONES	MANAGER	2975	81/04/02
2	7902	FORD	ANALYST	3000	81/12/03

SELECT

❖ LIKE 연산자

- ✓ STRING 값에 대한 와일드 카드 검색을 위해서 LIKE 연산자를 사용
- ✓ 검색 조건은 LITERAL 문자나 숫자를 포함
- ✓ '%'는 문자가 없거나 하나 이상의 문자
- ✓ '_'는 하나의 문자와 대치
- ✓ 패턴 일치 문자를 조합하는 것도 가능
- ✓ '%'나 '_'에 대해서 검색하기 위해서는 ESCAPE 식별자를 이용할 수 있음
- ✓ name에 값이 X_Y가 포함되어 있는 문자열을 조회하고자 할 경우 Escape를 사용
 - WHERE name LIKE '%X₩_Y%' ESCAPE '₩';



SELECT

- ❖ EMP 테이블에서 hiredate가 1982년인 사원의 empno, ename, job, sal, hiredate, deptno 를 조회

```
SELECT empno, ename, job, sal, hiredate, deptno
FROM emp
WHERE hiredate >= to_date('1982/01/01', 'yyyy/mm/dd') and
hiredate <= to_date('1982/12/31', 'yyyy/mm/dd');
```

Results:

	EMPNO	ENAME	JOB	SAL	HIREDATE	DEPTNO
1	7934	MILLER	CLERK	1300	82/01/23	10



SELECT

- ❖ EMP 테이블에서 hiredate가 1982년인 사원의 empno, ename, job, sal, hiredate, deptno 를 조회

```
SELECT empno, ename, job, sal, hiredate, deptno
FROM emp
WHERE hiredate LIKE '82%';
```

Results:

	EMPNO	ENAME	JOB	SAL	HIREDATE	DEPTNO
1	7934	MILLER	CLERK	1300	82/01/23	10



SELECT

- ❖ EMP 테이블에서 hiredate가 12월인 사원의 empno, ename, job, sal, hiredate, deptno 를 조회

```
SELECT empno, ename, job, sal, hiredate, deptno  
FROM emp  
WHERE hiredate LIKE '__12%';
```



SELECT

- ❖ IS NULL 연산자: NULL값은 값이 없거나, 알 수 없거나, 적용할 수 없다는 의미이므로 NULL값을 조회하고자 할 경우에 사용
- ❖ EMP 테이블에서 comm 이 NULL사원의 empno, ename, job, sal, comm, deptno를 조회

```
SELECT empno, ename, job, sal, comm, deptno  
FROM emp  
WHERE comm IS NULL;
```

Results:

	EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	800	(null)	20
2	7566	JONES	MANAGER	2975	(null)	20
3	7698	BLAKE	MANAGER	2850	(null)	30
4	7782	CLARK	MANAGER	2450	(null)	10
5	7788	SCOTT	ANALYST	3000	(null)	20
6	7839	KING	PRESIDENT	5000	(null)	10
7	7876	ADAMS	CLERK	1100	(null)	20
8	7900	JAMES	CLERK	950	(null)	30
9	7902	FORD	ANALYST	3000	(null)	20
10	7934	MILLER	CLERK	1300	(null)	10

SELECT

- ❖ 논리 연산자: AND, OR, NOT
- ❖ EMP 테이블에서 sal이 2800이상이고 job이 MANAGER 인 사원의 empno, ename, job, sal, hiredate, deptno를 조회

```
SELECT empno, ename, job, sal, hiredate, deptno  
FROM emp  
WHERE sal >= 2800 AND job = 'MANAGER';
```

```
SELECT empno, ename, job, sal, hiredate, deptno  
FROM emp  
WHERE job = 'MANAGER' AND sal >= 2800;
```



SELECT

- ❖ EMP 테이블에서 sal이 1100이상이거나 job이 MANAGER 인 사원의 empno, ename, job, sal, hiredate, deptno를 조회

```
SELECT empno, ename, job, sal, hiredate, deptno
FROM emp
WHERE sal >= 1100 OR job = 'MANAGER';
```

Results:

	R2	EMPNO	R2	ENAME	R2	JOB	R2	SAL	R2	HIREDATE	R2	DEPTNO
1		7499		ALLEN		SALESMAN		1600		81/02/20		30
2		7521		WARD		SALESMAN		1250		81/02/22		30
3		7566		JONES		MANAGER		2975		81/04/02		20
4		7654		MARTIN		SALESMAN		1250		81/09/28		30
5		7698		BLAKE		MANAGER		2850		81/05/01		30
6		7782		CLARK		MANAGER		2450		81/06/09		10
7		7788		SCOTT		ANALYST		3000		87/04/19		20
8		7839		KING		PRESIDENT		5000		81/11/17		10
9		7844		TURNER		SALESMAN		1500		81/09/08		30
10		7876		ADAMS		CLERK		1100		87/05/23		20
11		7902		FORD		ANALYST		3000		81/12/03		20
12		7934		MILLER		CLERK		1300		82/01/23		10

SELECT

- ❖ NOT 연산자는 BETWEEN, LIKE, IS NULL과 같은 다른 SQL연산자와 함께 사용 가능
- ❖ EMP 테이블에서 job이 MANAGER, CLERK, ANALYST가 아닌 사원의 empno, ename, job, sal, deptno를 조회

```
SELECT empno, ename, job, sal, deptno  
FROM emp  
WHERE job NOT IN ('MANAGER','CLERK','ANALYST')
```

Results:

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7499	ALLEN	SALESMAN	1600	30
2	7521	WARD	SALESMAN	1250	30
3	7654	MARTIN	SALESMAN	1250	30
4	7839	KING	PRESIDENT	5000	10
5	7844	TURNER	SALESMAN	1500	30



SELECT

- ❖ 우선 순위 규칙: 괄호>산술연산자>모든 비교 연산자>NOT>AND>OR
- ❖ emp 테이블에서 job이 SALESMAN 이거나 PRESIDENT이고 sal이 1500이 넘는 사원의 empno, ename, job, sal를 조회

```
SELECT empno, ename, job, sal
```

```
FROM emp
```

```
WHERE job = 'SALESMAN' OR job = 'PRESIDENT' AND sal > 1500;
```

Results:

	EMPNO	ENAME	JOB	SAL
1	7499	ALLEN	SALESMAN	1600
2	7521	WARD	SALESMAN	1250
3	7654	MARTIN	SALESMAN	1250
4	7839	KING	PRESIDENT	5000
5	7844	TURNER	SALESMAN	1500



SELECT

- ❖ emp 테이블에서 (job이 SALESMAN 이거나 PRESIDENT)이고 sal이 1500이 넘는 사원의 empno, ename, job, sal를 조회

```
SELECT empno, ename, job, sal
```

```
FROM emp
```

```
WHERE (job = 'SALESMAN' OR job = 'PRESIDENT') AND sal > 1500;
```

Results:

	EMPNO	ENAME	JOB	SAL
1	7499	ALLEN	SALESMAN	1600
2	7839	KING	PRESIDENT	5000



SELECT

- ❖ SELECT를 수행한 나오는 결과 행의 순서는 알 수 없음
- ❖ ORDER BY절은 행을 정렬하는데 사용
- ❖ ORDER BY절을 사용하는 경우 SELECT문의 맨 뒤에 기술해야 함
- ❖ 정렬을 위한 표현식이나 컬럼의 Alias을 명시할 수 있음

- ❖ Syntax

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

ORDER BY	검색된 행이 조회되는 순서를 명시
ASC	행의 오름차순 정렬(Default)
DESC	행의 내림차순 정렬



SELECT

- ❖ 디폴트 정렬은 오름차순
- ❖ 숫자 값은 가장 적은 값이 먼저 조회(예 : 1 ~ 999)
- ❖ 날짜 값은 가장 빠른 값이 먼저 조회(예 : 01-JAN-92 ~ 01-JAN-95)
- ❖ 문자 값은 알파벳 순서로 조회(예 : A ~ Z ~ a ~ z)
- ❖ NULL값은 오름차순에서는 제일 나중에 그리고 내림차순에서는 제일 먼저
- ❖ 행이 디스플레이 되는 순서를 큰 것을 먼저 조회하기 위해서는 ORDER BY절에서 열 이름 뒤에 DESC키워드를 명시



SELECT

- ❖ emp 테이블에서 hiredate의 오름차순으로 hiredate, empno, ename, job, sal, deptno를 조회

```
SELECT hiredate, empno, ename, job, sal, deptno  
FROM emp  
ORDER BY hiredate;
```

Results:







	HIREDATE	EMPNO	ENAME	JOB	SAL	DEPTNO
1	80/12/17	7369	SMITH	CLERK	800	20
2	81/02/20	7499	ALLEN	SALESMAN	1600	30
3	81/02/22	7521	WARD	SALESMAN	1250	30
4	81/04/02	7566	JONES	MANAGER	2975	20
5	81/05/01	7698	BLAKE	MANAGER	2850	30
6	81/06/09	7782	CLARK	MANAGER	2450	10
7	81/09/08	7844	TURNER	SALESMAN	1500	30
8	81/09/28	7654	MARTIN	SALESMAN	1250	30
9	81/11/17	7839	KING	PRESIDENT	5000	10
10	81/12/03	7900	JAMES	CLERK	950	30
11	81/12/03	7902	FORD	ANALYST	3000	20
12	82/01/23	7934	MILLER	CLERK	1300	10
13	87/04/19	7788	SCOTT	ANALYST	3000	20
14	87/05/23	7876	ADAMS	CLERK	1100	20

SELECT

- ❖ emp 테이블에서 hiredate의 내림차순으로 hiredate,empno,ename,job,sal,deptno를 조회

```
SELECT hiredate, empno, ename, job, sal, deptno
FROM emp
ORDER BY hiredate desc;
```

Results:

	 HIREDATE	 EMPNO	 ENAME	 JOB	 SAL	 DEPTNO
1	87/05/23	7876	ADAMS	CLERK	1100	20
2	87/04/19	7788	SCOTT	ANALYST	3000	20
3	82/01/23	7934	MILLER	CLERK	1300	10
4	81/12/03	7902	FORD	ANALYST	3000	20
5	81/12/03	7900	JAMES	CLERK	950	30
6	81/11/17	7839	KING	PRESIDENT	5000	10
7	81/09/28	7654	MARTIN	SALESMAN	1250	30
8	81/09/08	7844	TURNER	SALESMAN	1500	30
9	81/06/09	7782	CLARK	MANAGER	2450	10
10	81/05/01	7698	BLAKE	MANAGER	2850	30
11	81/04/02	7566	JONES	MANAGER	2975	20
12	81/02/22	7521	WARD	SALESMAN	1250	30
13	81/02/20	7499	ALLEN	SALESMAN	1600	30
14	80/12/17	7369	SMITH	CLERK	800	20


SELECT

❖ 다양한 정렬 방법

```
SELECT empno, ename, job, sal, sal*12 annsal  
FROM emp  
ORDER BY annsal;
```

```
SELECT empno, ename, job, sal, sal*12 annsal  
FROM emp  
ORDER BY sal*12;
```

```
SELECT empno, ename, job, sal, sal*12 annsal  
FROM emp  
ORDER BY 5;
```

- ❖ 하나 이상의 열로 질의 결과를 정렬
 - ❖ ORDER BY절에서 열을 명시하고, 열 이름은 콤마로 구분
 - ❖ SELECT절에 포함되지 않는 열이나 계산식으로 정렬 가능
- 

SELECT

- ❖ emp 테이블에서 deptno의 오름차순으로 정렬하고 같은 경우 sal의 내림차순으로 deptno, sal, empno, ename, job 를 조회

```
SELECT deptno, sal, empno, ename, job  
FROM emp  
ORDER BY deptno, sal DESC;
```

Results:

	R 2	DEPTNO	R 2	SAL	R 2	EMPNO	R 2	ENAME	R 2	JOB
1		10		5000		7839		KING		PRESIDENT
2		10		2450		7782		CLARK		MANAGER
3		10		1300		7934		MILLER		CLERK
4		20		3000		7788		SCOTT		ANALYST
5		20		3000		7902		FORD		ANALYST
6		20		2975		7566		JONES		MANAGER
7		20		1100		7876		ADAMS		CLERK
8		20		800		7369		SMITH		CLERK
9		30		2850		7698		BLAKE		MANAGER
10		30		1600		7499		ALLEN		SALESMAN
11		30		1500		7844		TURNER		SALESMAN
12		30		1250		7654		MARTIN		SALESMAN
13		30		1250		7521		WARD		SALESMAN
14		30		950		7900		JAMES		CLERK

SELECT

- ❖ emp 테이블에서 deptno의 오름차순으로 정렬하고 같은 경우 job의 오름차순으로 job이 같은 경우에는 sal의 내림차순으로 deptno, job, sal, empno, ename, hiredate를 조회

```
SELECT deptno, job, sal, empno, ename, hiredate
FROM emp
ORDER BY deptno, job, sal DESC;
```

Results:

	DEPTNO	JOB	SAL	EMPNO	ENAME	HIREDATE
1	10	CLERK	1300	7934	MILLER	82/01/23
2	10	MANAGER	2450	7782	CLARK	81/06/09
3	10	PRESIDENT	5000	7839	KING	81/11/17
4	20	ANALYST	3000	7788	SCOTT	87/04/19
5	20	ANALYST	3000	7902	FORD	81/12/03
6	20	CLERK	1100	7876	ADAMS	87/05/23
7	20	CLERK	800	7369	SMITH	80/12/17
8	20	MANAGER	2975	7566	JONES	81/04/02
9	30	CLERK	950	7900	JAMES	81/12/03
10	30	MANAGER	2850	7698	BLAKE	81/05/01
11	30	SALESMAN	1600	7499	ALLEN	81/02/20
12	30	SALESMAN	1500	7844	TURNER	81/09/08
13	30	SALESMAN	1250	7654	MARTIN	81/09/28
14	30	SALESMAN	1250	7521	WARD	81/02/22

그룹 함수

❖ 그룹 함수

- ✓ 하나 이상의 행을 그룹으로 묶어서 연산해서 리턴하는 함수
- ✓ 그룹화 한 이후에 사용해야 하기 때문에 HAVING, SELECT, ORDER BY 에서 사용 가능
- ✓ NULL은 제외하고 계산



그룹 함수

❖ 그룹 함수는 하나 이상의 행을 그룹으로 묶어 연산하여 총합, 평균 등 하나의 결과로 리턴

구 분	설 명
SUM	그룹의 누적 합계를 반환
AVG	그룹의 평균을 반환
COUNT	그룹의 총 개수를 반환
MAX	그룹의 최대값을 반환
MIN	그룹의 최소값을 반환
STDDEV	그룹의 표준편차를 반환
VARIANCE	그룹의 분산을 반환

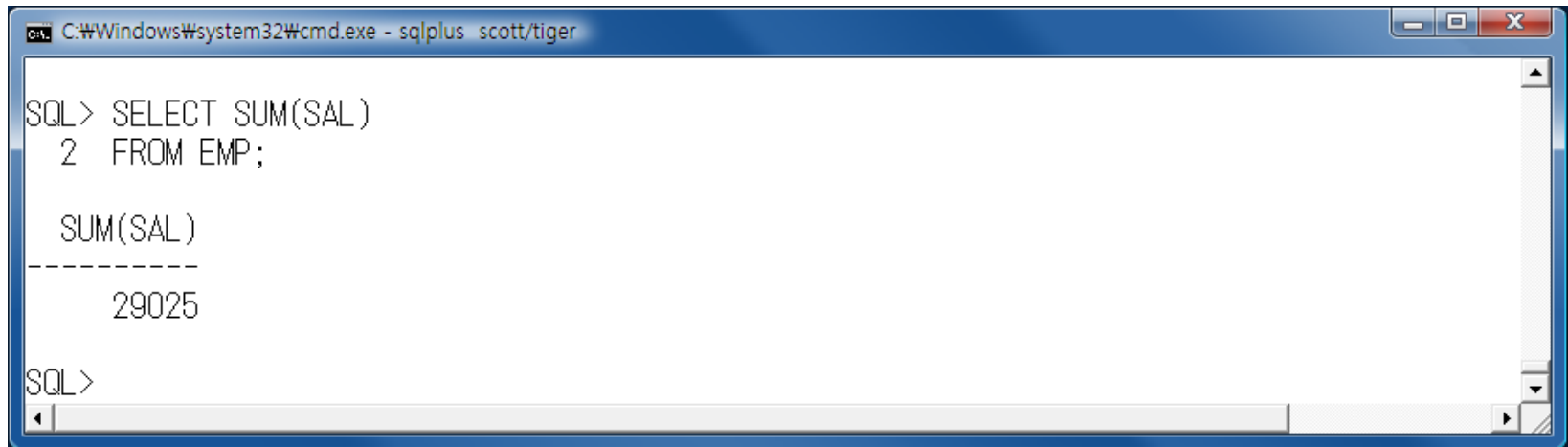
그룹 함수

❖ SUM 함수

- ✓ 해당 컬럼 값들에 대한 총합을 구하는 함수로 숫자 데이터에만 사용
- ✓ EMP 테이블에서 SAL의 합계 조회

예

```
SELECT SUM(SAL)
FROM EMP;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL command is entered and executed:

```
SQL> SELECT SUM(SAL)
2 FROM EMP;
```

The output of the query is displayed as follows:

```
SUM(SAL)
-----
29025
```

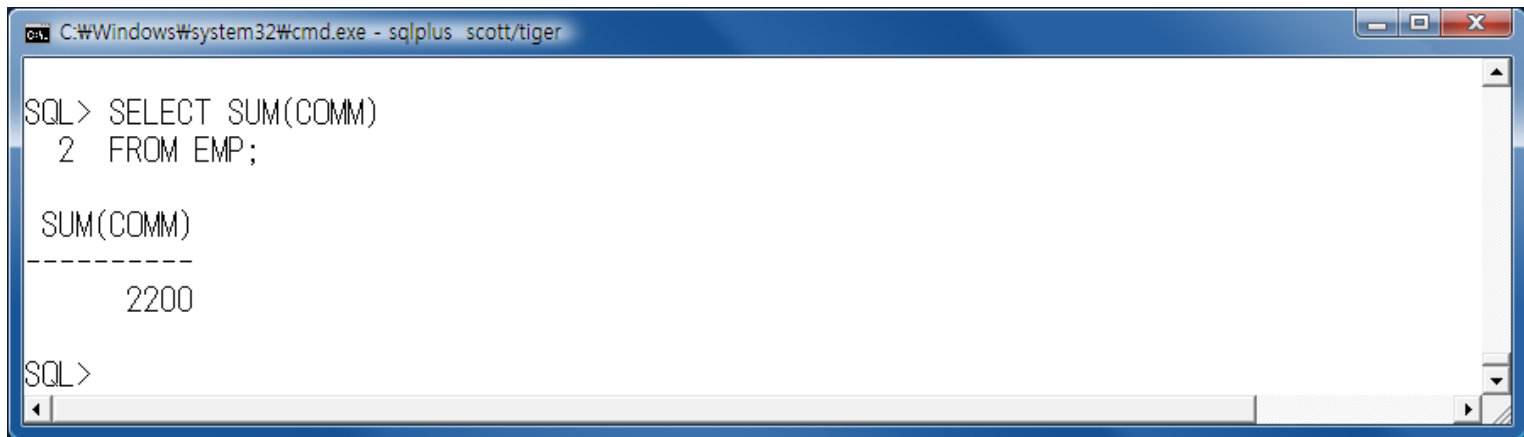
The prompt "SQL>" is visible at the bottom of the window, indicating the command is ready for the next input.

그룹 함수

- ❖ EMP 테이블에서 COMM의 합계 구하기

예

```
SELECT SUM(COMM)
FROM EMP;
```



```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger

SQL> SELECT SUM(COMM)
2  FROM EMP;

SUM(COMM)
-----
      2200

SQL>
```

- ❖ 사원 테이블을 살펴보면 커미션 컬럼에 NULL 값이 저장된 사원이 존재하는데 NULL 은 블랙 홀이므로 NULL을 저장한 컬럼과 연산한 결과도 NULL인데 커미션의 총합을 구해도 NULL 값으로 출력되지 않는데 그룹 함수는 다른 연산자와는 달리 해당 컬럼 값이 NULL 인 것을 제외하고 계산

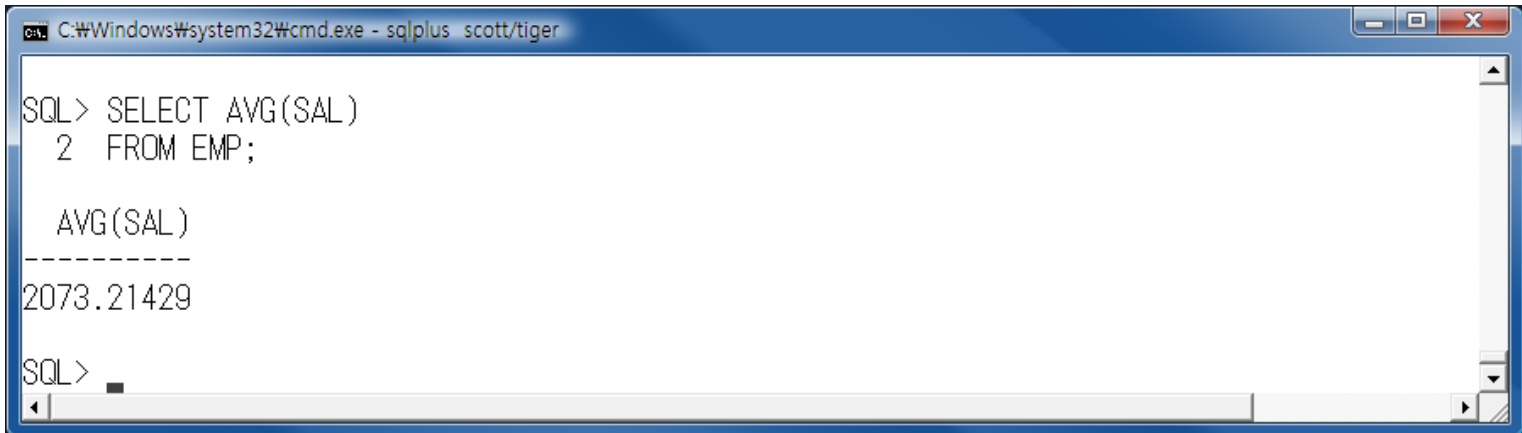
그룹 함수

❖ AVG 함수

- ✓ 해당 컬럼 값들에 대해 평균을 구하는 함수
- ✓ 해당 컬럼 값이 NULL 인 것에 대해서는 제외하고 계산
- ✓ 다음은 EMP 테이블에서 SAL의 평균 구하기

예

```
SELECT AVG(SAL)
FROM EMP;
```



```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger

SQL> SELECT AVG(SAL)
2  FROM EMP;

      AVG(SAL)
-----
2073.21429

SQL>
```

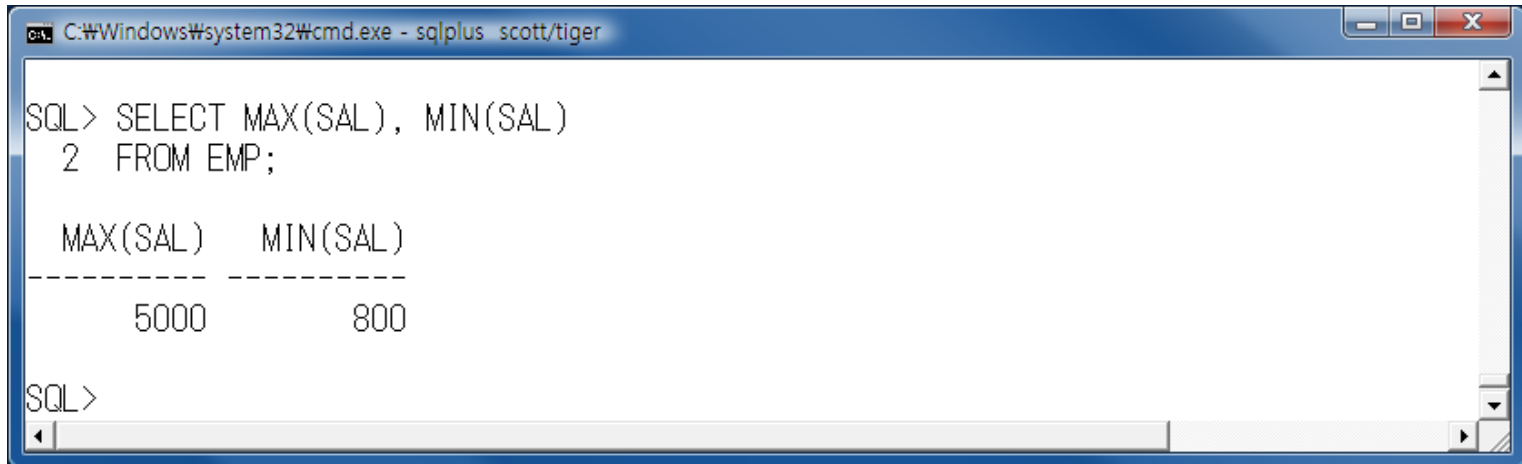
그룹 함수

❖ MAX, MIN

- ✓ 지정한 컬럼 값들 중에서 최대값을 구하는 함수가 MAX이고, 최소값을 구하는 함수가 MIN
- ✓ 숫자, 문자, 날짜 형식 모두에 사용 가능
- ✓ 다음은 가장 높은 급여와 가장 낮은 급여를 구하기

예

```
SELECT MAX(SAL), MIN(SAL)
FROM EMP;
```



```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger

SQL> SELECT MAX(SAL), MIN(SAL)
2  FROM EMP;

  MAX(SAL)    MIN(SAL)
-----
      5000         800

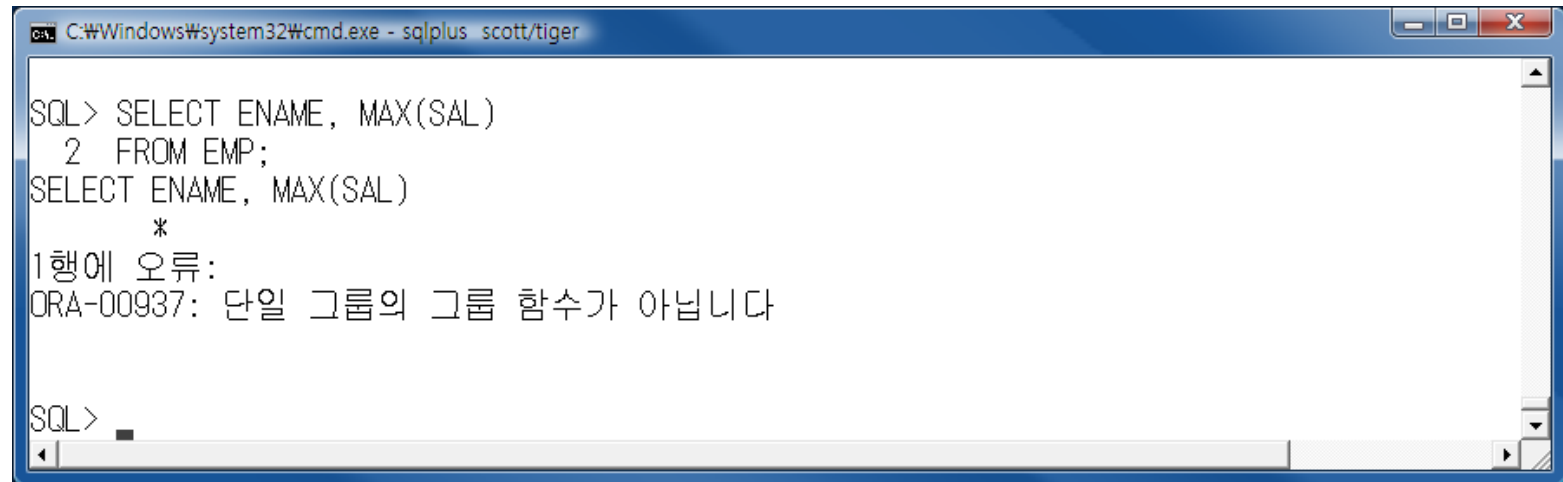
SQL>
```

그룹 함수와 단순 컬럼

- ❖ 직원들의 최대 급여는 다음과 같이 구함

예	SELECT MAX(SAL) FROM EMP;
---	------------------------------

- ❖ 아래와 같이 직원의 이름도 함께 출력하려고 하면 에러 - 그룹 함수는 그룹화 한 표현식 만 같이 출력 가능



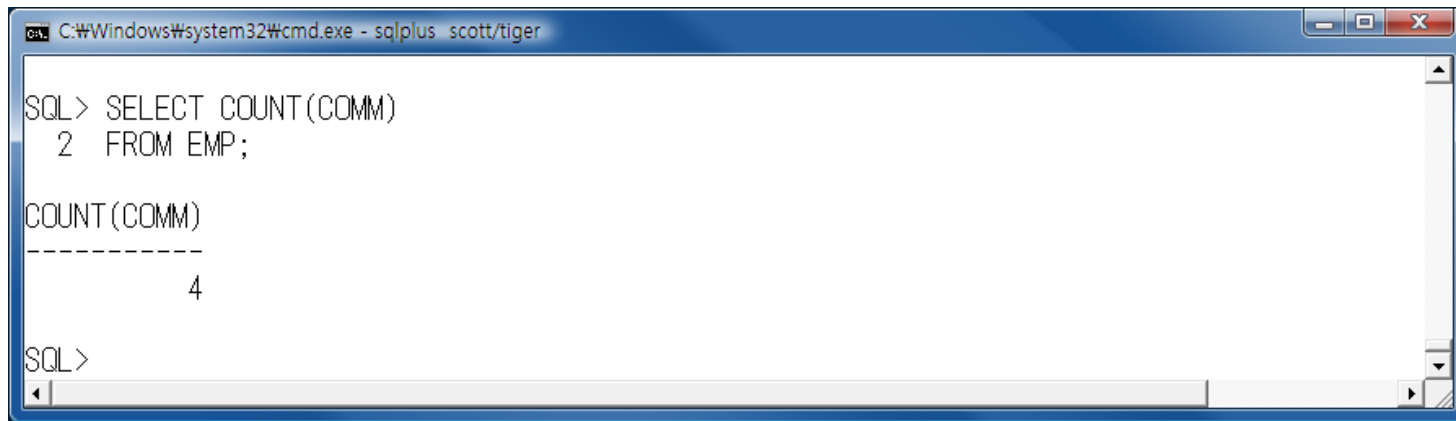
The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered: `SQL> SELECT ENAME, MAX(SAL) 2 FROM EMP;`. The output shows the first line of the query, `SELECT ENAME, MAX(SAL)`, followed by an asterisk `*` and the error message `1행에 오류: ORA-00937: 단일 그룹의 그룹 함수가 아닙니다`. The prompt `SQL>` is visible at the bottom.

COUNT 함수

- ❖ COUNT 함수는 테이블에서 조건을 만족하는 행의 개수를 반환하는 함수
- ❖ COUNT 함수에 특정 컬럼을 기술하면 해당 컬럼 값을 갖고 있는 로우의 개수를 계산하여 리턴
- ❖ 다음은 사원 테이블의 사원들 중에서 커미션을 받은 사원의 수를 구하는 예제

예

```
SELECT COUNT(COMM)  
FROM EMP;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". The prompt is "SQL>". The user has entered the SQL query "SELECT COUNT(COMM) FROM EMP;". The output shows "COUNT(COMM)" followed by a dashed line and the value "4". The prompt "SQL>" is visible again at the bottom.

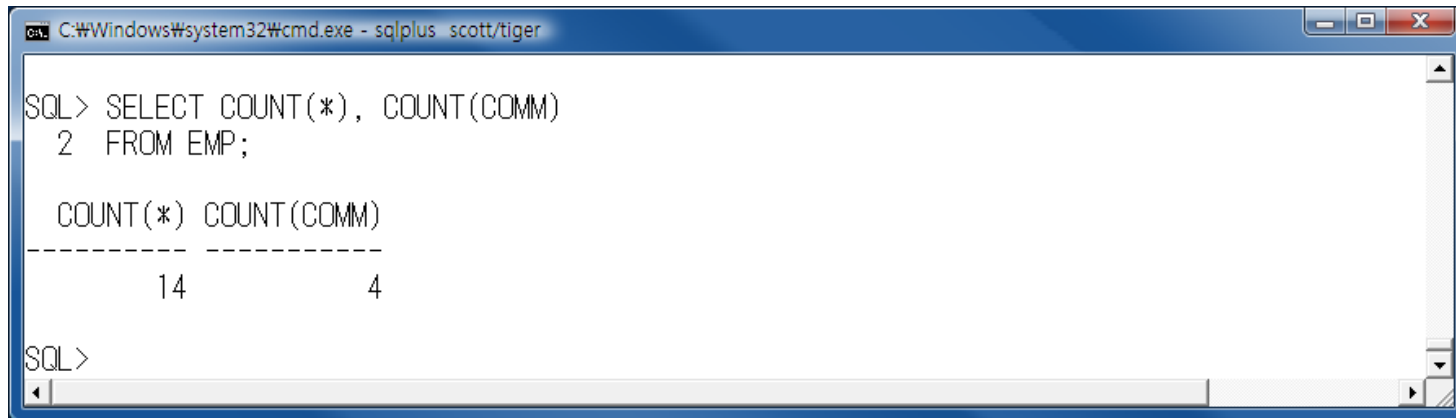
```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger  
  
SQL> SELECT COUNT(COMM)  
2  FROM EMP;  
  
COUNT(COMM)  
-----  
4  
  
SQL>
```

COUNT 함수

- ❖ COUNT 함수는 NULL 값에 대해서는 개수를 세지 않음
- ❖ COUNT 함수에 COUNT(*)처럼 *를 적용하면 테이블의 전체 로우 수를 리턴
- ❖ 전체 사원의 수와 커미션을 받는 사원의 수를 구하는 예제

예

```
SELECT COUNT(*), COUNT(COMM)
FROM EMP;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT COUNT(*), COUNT(COMM)
2 FROM EMP;
```

The output of the query is displayed as follows:

COUNT(*)	COUNT(COMM)
14	4

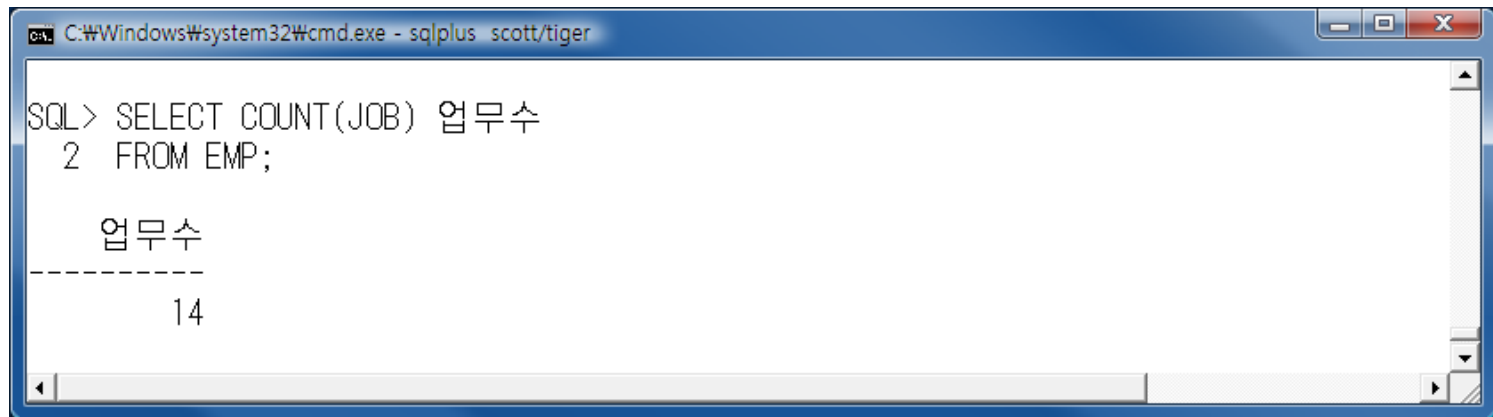
The window also shows the prompt "SQL>" at the bottom, indicating the end of the command.

COUNT 함수

- ❖ 사원 테이블에서 사원들의 직업의 개수를 조회

예

```
SELECT COUNT(JOB) 업무수  
FROM EMP;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT COUNT(JOB) 업무수  
2 FROM EMP;
```

The output of the query is displayed as follows:

업무수
14

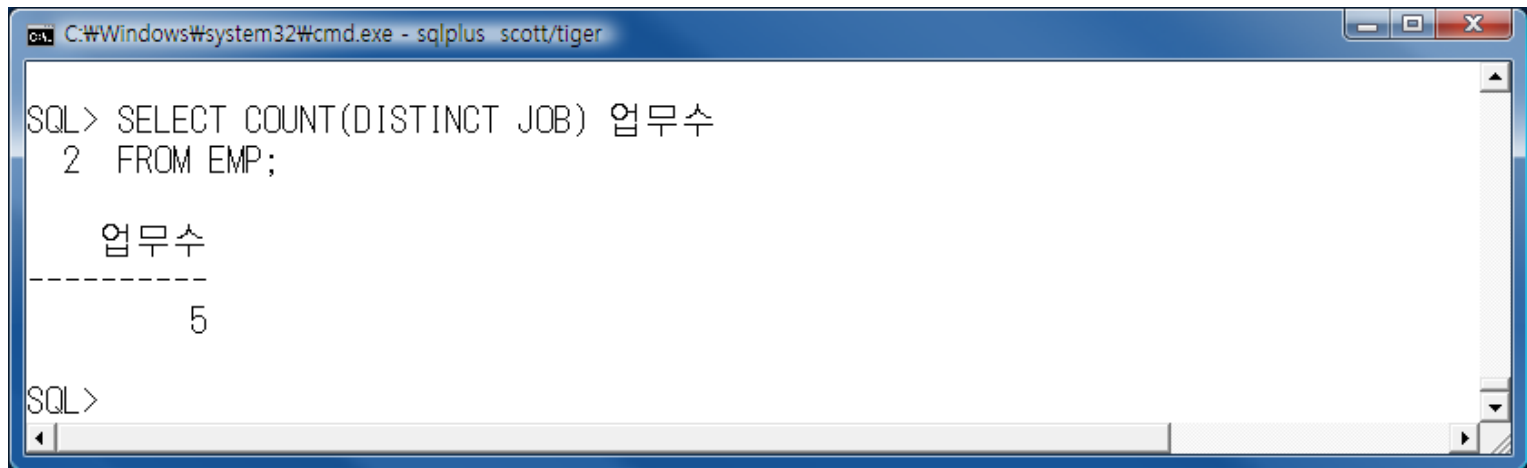
The background of the slide features a faint image of a fountain pen and some documents.

COUNT 함수

- ❖ 직업의 종류가 몇 개인지 즉, 중복되지 않은 직업의 개수를 카운트
- ❖ 중복 행 제거 키워드 DISTINCT를 써서 다음과 같이 질의

예

```
SELECT COUNT(DISTINCT JOB) 업무수  
FROM EMP;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT COUNT(DISTINCT JOB) 업무수  
2 FROM EMP;
```

The output of the query is displayed as follows:

업무수
5

The prompt "SQL>" is visible at the bottom of the window.

RANK 함수

❖ 문법

- ✓ RANK(expr) within group(ORDER BY expr)

EMP 테이블에서 sal이 3000인 사원의 순위 구하기

예

```
SELECT rank(3000) within group(ORDER BY sal desc) as 순위  
FROM EMP;
```

≡	순 위
➤	2



GROUP BY

- ❖ 특정 컬럼 값을 기준으로 그룹화 하고자 하는 경우 GROUP BY 절 뒤에 해당 컬럼을 기술

형식

```
SELECT 컬럼명, 그룹함수  
FROM 테이블명  
WHERE 조건 (연산자)  
GROUP BY 컬럼명;
```

- ❖ 합계, 평균, 최대값.이나, 최소값. 등을 어떤 컬럼을 기준으로 그 컬럼의 값 별로 보고자 할 때 GROUP BY 절 뒤에 해당 컬럼을 기술
- ❖ GROUP BY 절을 사용할 때 주의할 점은 GROUP BY 절 다음에는 컬럼의 별칭을 사용할 수 없고, 반드시 컬럼명을 기술

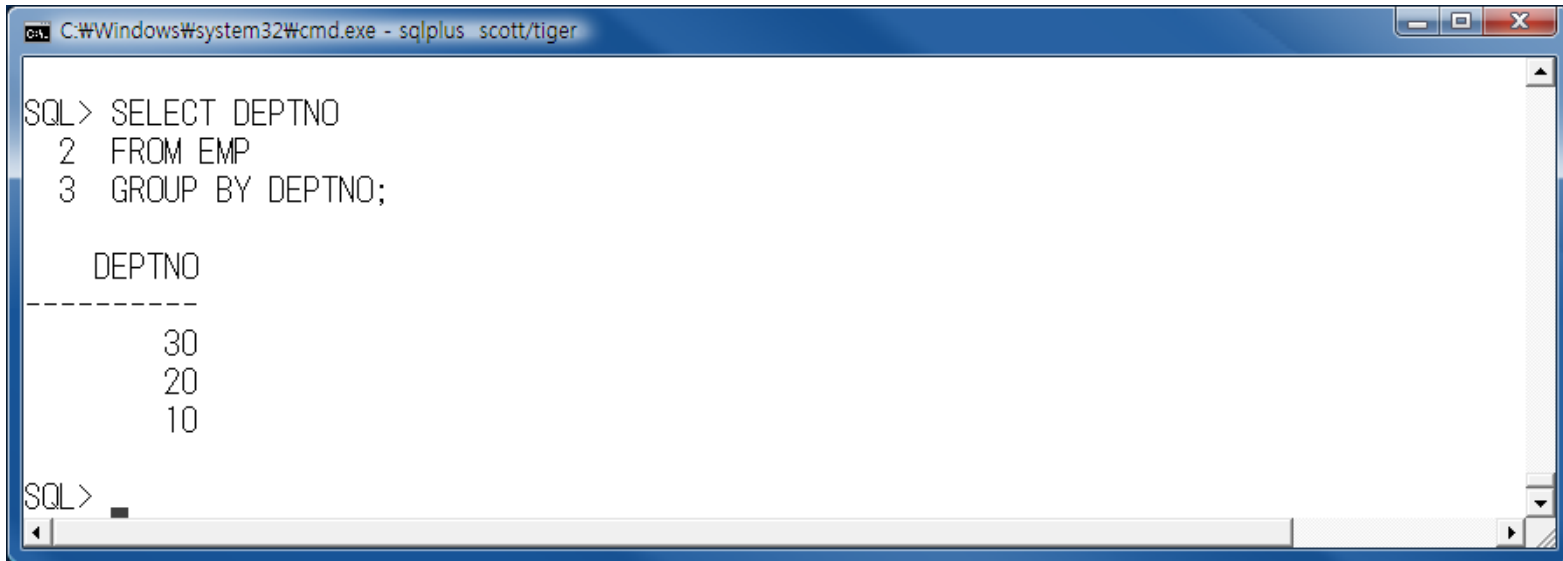


GROUP BY

- ❖ 사원 테이블을 부서 번호로 그룹화 해서 출력

예

```
SELECT DEPTNO  
FROM EMP  
GROUP BY DEPTNO;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT DEPTNO  
2 FROM EMP  
3 GROUP BY DEPTNO;
```

The output of the query is displayed as follows:

DEPTNO
30
20
10

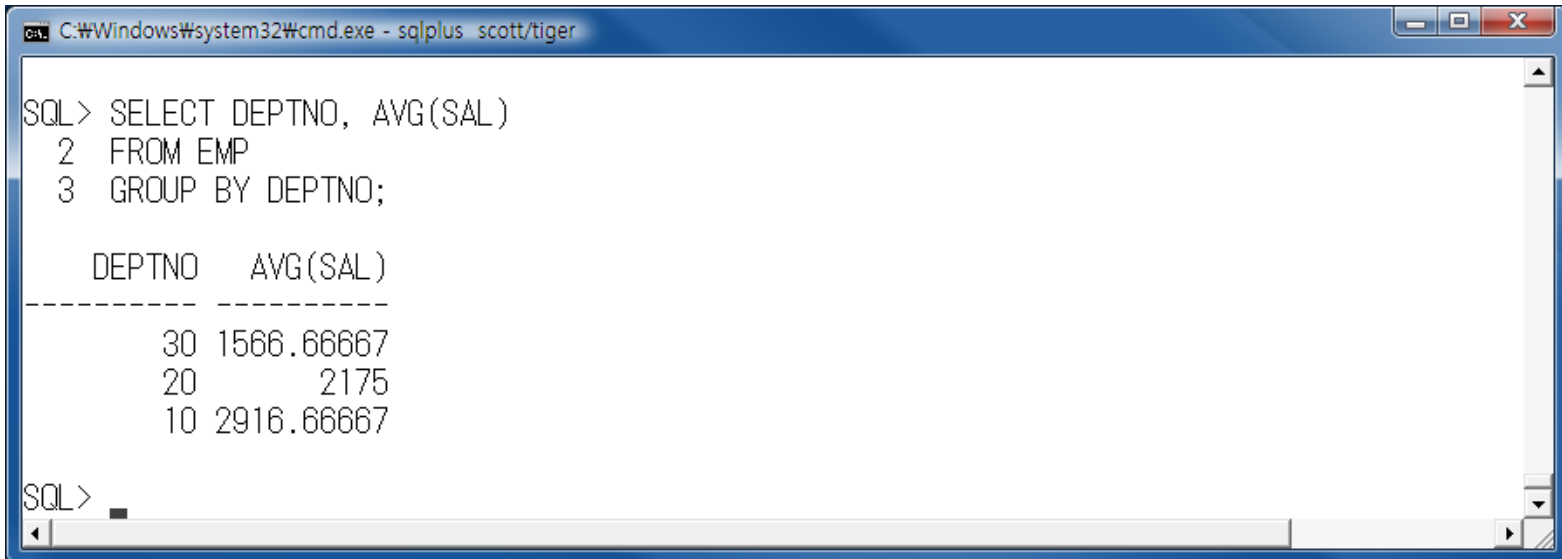
The window also shows the "SQL>" prompt at the bottom, indicating the command is ready for the next input.

GROUP BY

- ❖ EMP 테이블에서 소속 부서별(deptno) 평균 급여(sal)를 출력

예

```
SELECT DEPTNO, AVG(SAL)
FROM EMP
GROUP BY DEPTNO;
```



```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger

SQL> SELECT DEPTNO, AVG(SAL)
  2  FROM EMP
  3  GROUP BY DEPTNO;

   DEPTNO   AVG(SAL)
-----
       30  1566.66667
       20    2175
       10  2916.66667

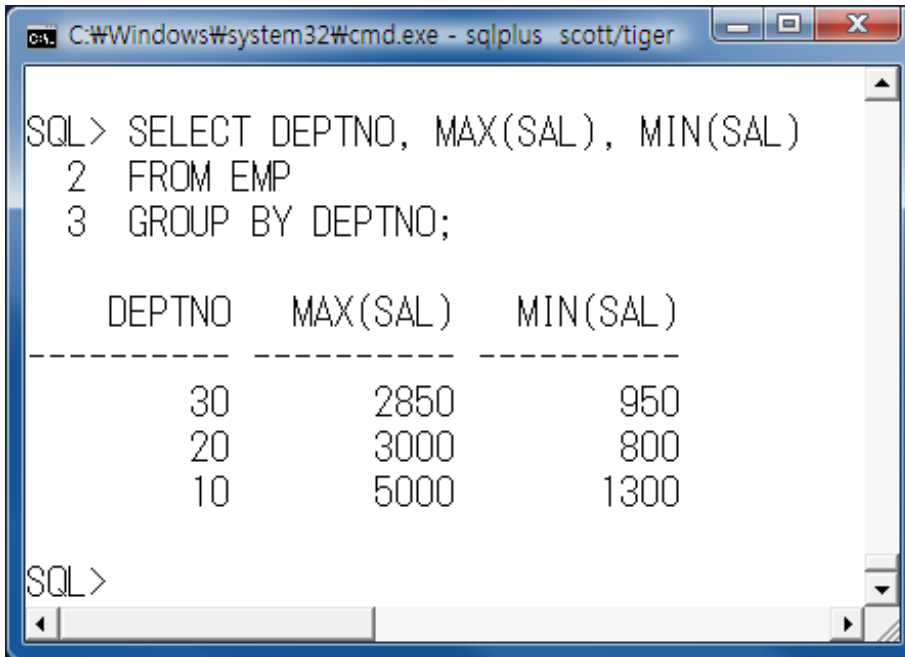
SQL>
```


GROUP BY

- ❖ EMP 테이블에서 소속 부서별(deptno) 최대 급여(sal)와 최소 급여를 출력

예

```
SELECT DEPTNO, MAX(SAL), MIN(SAL)
FROM EMP
GROUP BY DEPTNO;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT DEPTNO, MAX(SAL), MIN(SAL)
2  FROM EMP
3  GROUP BY DEPTNO;
```

The output of the query is displayed in a table format:

DEPTNO	MAX(SAL)	MIN(SAL)
30	2850	950
20	3000	800
10	5000	1300

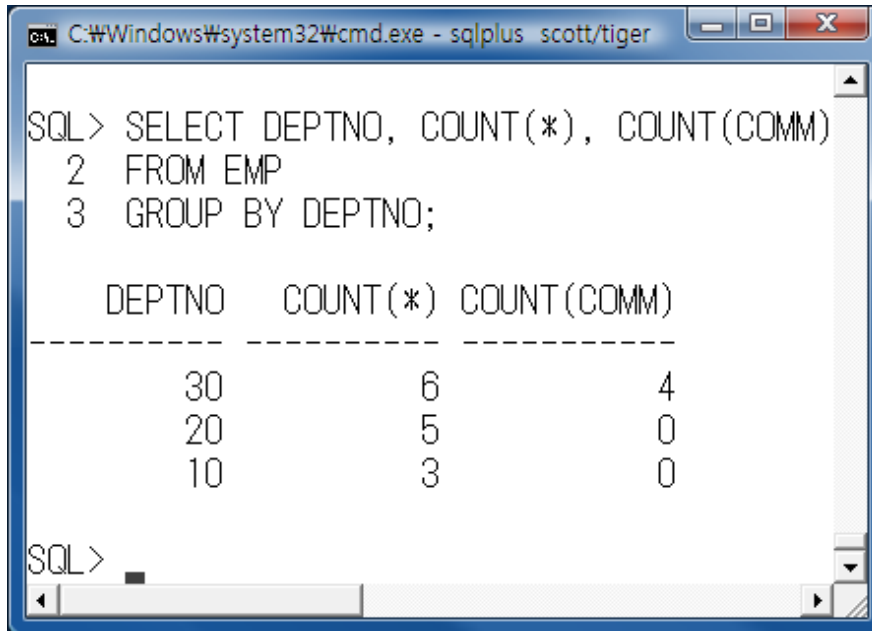
The prompt "SQL>" is visible at the bottom of the window.

GROUP BY

- ❖ 부서별(DEPTNO)로 사원 수와 커미션(COMM)을 받는 사원들의 수를 계산

예

```
SELECT DEPTNO, count(*), count(comm)
FROM EMP
GROUP BY DEPTNO;
```



```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger

SQL> SELECT DEPTNO, COUNT(*), COUNT(COMM)
2  FROM EMP
3  GROUP BY DEPTNO;

  DEPTNO    COUNT(*)  COUNT(COMM)
-----
      30          6           4
      20          5           0
      10          3           0

SQL>
```

GROUP BY

- ❖ 부서 와 직무(DEPTNO, JOB)별로 사원 수와 급여(SAL)의 합을 계산

예

```
SELECT DEPTNO, JOB, COUNT(*), SUM(SAL)
FROM EMP
GROUP BY DEPTNO, JOB
ORDER BY DEPTNO, JOB;
```

	DEPTNO	JOB	COUNT(*)	SUM(SAL)
1	10	CLERK	1	1,300
2	10	MANAGER	1	2,450
3	10	PRESIDENT	1	5,000
4	20	ANALYST	2	6,000
5	20	CLERK	2	1,900
6	20	MANAGER	1	2,975
7	30	CLERK	1	950
8	30	MANAGER	1	2,850
9	30	SALESMAN	4	5,600

GROUP BY

- ❖ 부서별 평균 급여(SAL)가 가장 큰 값 조회

예

```
SELECT MAX(AVG(SAL))  
FROM EMP  
GROUP BY DEPTNO;
```

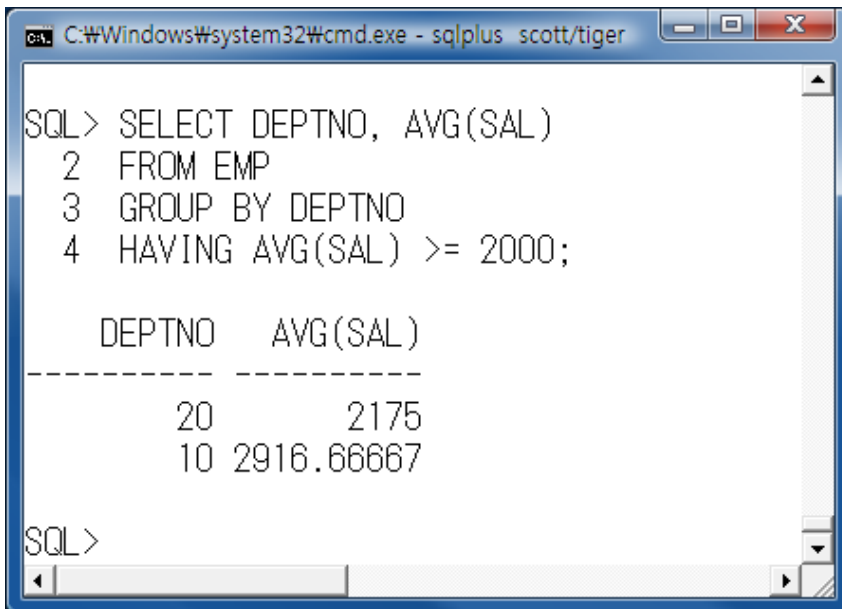
	123 MAX(AVG(SAL)) 🔍 ↕
1	2,916.6666666667

HAVING 조건

- ❖ SELECT 절에 조건을 사용하여 결과를 제한할 때는 WHERE 절을 사용하지만 그룹의 결과를 제한할 때는 HAVING 절을 사용
- ❖ 부서별로 그룹화 한 후(GROUP BY), 그룹 지어진 부서별 평균 급여가 2000 이상인(HAVING) 부서 번호와 부서별 평균 급여를 출력하는 경우

예

```
SELECT DEPTNO, ROUND(AVG(SAL))  
FROM EMP  
GROUP BY DEPTNO  
HAVING AVG(SAL) >= 2000;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT DEPTNO, AVG(SAL)  
2 FROM EMP  
3 GROUP BY DEPTNO  
4 HAVING AVG(SAL) >= 2000;
```

The output of the query is displayed as a table:

DEPTNO	AVG(SAL)
20	2175
10	2916.66667

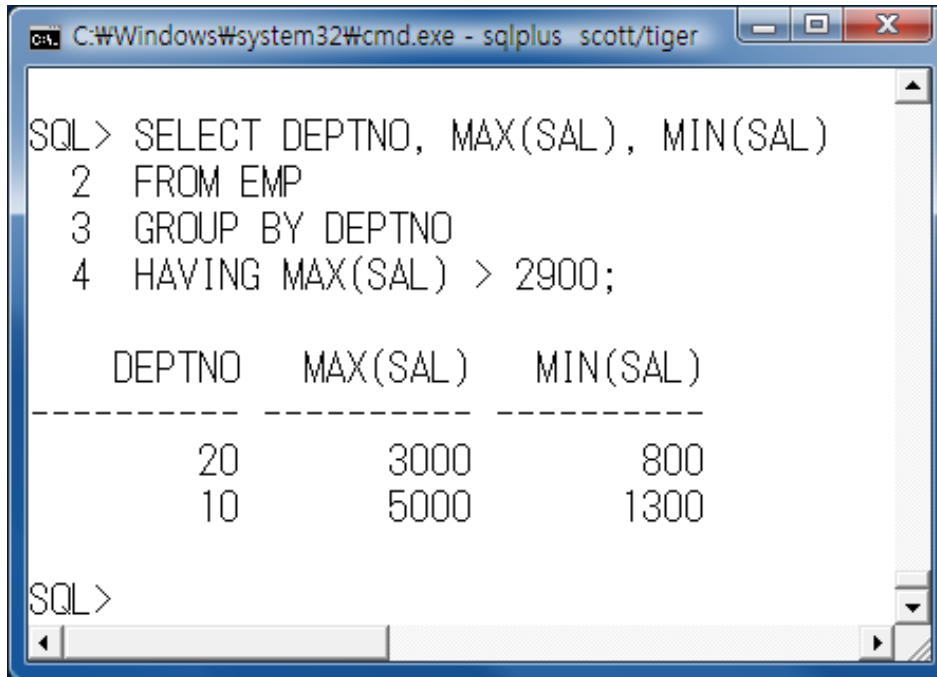
The prompt "SQL>" is visible at the bottom of the window.

HAVING 조건

- ❖ 부서의 최대 급여 와 최소 급여를 구하되 최대 급여가 2900이상인 부서만 출력

예

```
SELECT DEPTNO, MAX(SAL), MIN(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MAX(SAL) > 2900;
```



```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger

SQL> SELECT DEPTNO, MAX(SAL), MIN(SAL)
2  FROM EMP
3  GROUP BY DEPTNO
4  HAVING MAX(SAL) > 2900;

   DEPTNO   MAX(SAL)   MIN(SAL)
-----
        20        3000         800
        10        5000        1300

SQL>
```

HAVING 조건

- ❖ 비 효율적인 HAVING을 만드는 경우가 있는데 WHERE 절에서 제한이 가능한 조건을 HAVING에 기술하는 경우
- ❖ EMP 테이블에서 deptno가 10, 20인 부서의 sal의 평균을 구하는 sql

```
SELECT DEPTNO, avg(sal)
FROM EMP
GROUP BY DEPTNO
HAVING deptno in (10,20);
```

```
SELECT DEPTNO, avg(sal)
FROM EMP
WHERE deptno in(10,20)
GROUP BY DEPTNO;
```



조인의 필요성

- ❖ 특정 부서 번호에 대한 부서이름이 무엇인지는 부서(DEPT) 테이블에 있는 경우 특정 사원에 대한 부서명을 알아내기 위해서는 부서 테이블에서 정보를 가져와야 함

```
C:\Windows\system32\cmd.exe
SQL> SELECT ENAME, DEPTNO
2 FROM EMP
3 ORDER BY DEPTNO;
```

ENAME	DEPTNO
CLARK	10
KING	10
MILLER	10
JONES	20
FORD	20
ADAMS	20
SMITH	20
SCOTT	20
WARD	30
TURNER	30
ALLEN	30
JAMES	30
BLAKE	30
MARTIN	30

14 개의 행이 선택되었습니다.

```
SQL>
```

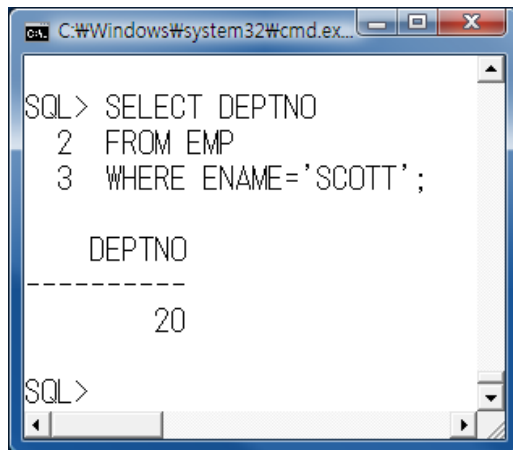
```
C:\Windows\system32\cmd.exe
SQL> SELECT DEPTNO, DNAME
2 FROM DEPT;
```

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

```
SQL>
```


조인의 필요성

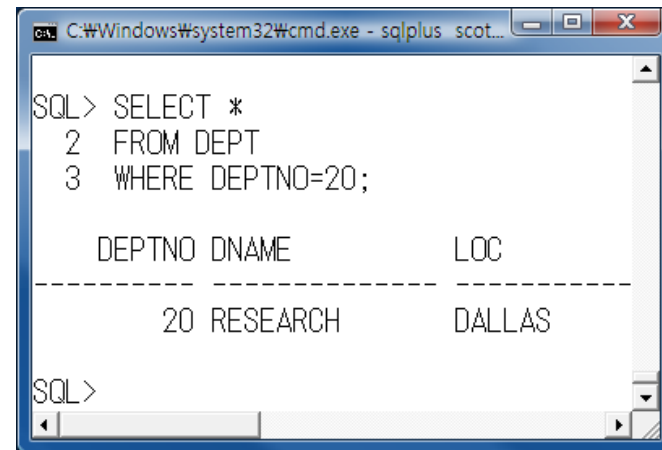
- ❖ MILLER인 사원이 소속되어 있는 부서의 이름이 무엇인지 알아보고자 하는 경우 MILLER란 사원의 부서명을 알아내는 작업 역시 사원 테이블에서 MILLER이 소속된 부서 번호를 알아낸 후에 부서 테이블에서 해당 부서 번호에 대한 부서명을 가져와야 함



```
C:\Windows\system32\cmd.exe
SQL> SELECT DEPTNO
2 FROM EMP
3 WHERE ENAME='SCOTT';

DEPTNO
-----
      20

SQL>
```



```
C:\Windows\system32\cmd.exe - sqlplus scot...
SQL> SELECT *
2 FROM DEPT
3 WHERE DEPTNO=20;

DEPTNO DNAME          LOC
-----
      20 RESEARCH      DALLAS

SQL>
```

- ❖ 실습에서처럼 원하는 정보가 두 개 이상의 테이블에 나누어져 있다면 위와 같이 여러 번 질의를 하지 않고 SQL에서는 두 개 이상의 테이블을 결합해야만 원하는 결과를 얻을 수 있을 때 한 번의 질의로 원하는 결과를 얻을 수 있는 조인 기능을 제공

JOIN

❖ JOIN

- ✓ 2개의 테이블에 대해 연관된 튜플들을 결합하여, 하나의 새로운 테이블을 생성하는 것
- ✓ JOIN은 크게 INNER JOIN과 OUTER JOIN으로 구분
- ✓ JOIN은 일반적으로 FROM절에 기술하지만 릴레이션이 사용되는 어느 곳에서나 사용 가능
- ✓ CROSS JOIN(CATESIAN PRODUCT)
 - FROM 절에 테이블 명을 2개 이상 기술한 것으로 JOIN 조건이 없는 경우
 - 2개 테이블의 모든 조합을 리턴
- ✓ INNER JOIN
 - 일반적인 JOIN의 형태로, 관계가 설정된 두 테이블에서 조인된 필드가 일치하는 행 만을 표시
 - WHERE 절 이용

```
SELECT [테이블명1.]속성명, [테이블명2.]속성명, ...  
FROM 테이블명1, 테이블명2, ...  
WHERE 테이블명1.속성명 = 테이블명2.속성명;
```
 - NATURAL JOIN

```
SELECT [테이블명1.]속성명, [테이블명2.]속성명, ...  
FROM 테이블명1 NATURAL JOIN 테이블명2;
```
 - USING 이용

```
SELECT [테이블명1.]속성명, [테이블명2.]속성명, ...  
FROM 테이블명1 JOIN 테이블명2 USING(속성명);
```

JOIN

❖ JOIN

✓ OUTER JOIN

- 릴레이션에서 JOIN 조건에 만족하지 않는 튜플도 결과로 출력하기 위한 JOIN 방법으로 LEFT OUTER JOIN, RIGHT OUTER JOIN 등이 있다.
- LEFT OUTER JOIN INNER JOIN의 결과를 구한 후, 우측 항 릴레이션의 어떤 튜플과도 맞지 않는 좌측 항의 릴레이션에 있는 튜플들에 NULL 값을 붙여서 INNER JOIN의 결과에 추가함

SELECT [테이블명1.]속성명, [테이블명2.]속성명, ...

FROM 테이블명1 LEFT OUTER JOIN 테이블명2 ON 테이블명1.속성명 = 테이블명2.속성명;

- RIGHT OUTER JOIN : INNER JOIN의 결과를 구한 후, 좌측 항 릴레이션의 어떤 튜플과도 맞지 않는 우측 항의 릴레이션에 있는 튜플들에 NULL 값을 붙여서 INNER JOIN의 결과에 추가함

SELECT [테이블명1.]속성명, [테이블명2.]속성명, ...

FROM 테이블명1 RIGHT OUTER JOIN 테이블명2 ON 테이블명1.속성명 = 테이블명2.속성명;

- + 기호를 이용해서 반대편에 추가할 수 있음
- Ansi 표준에는 Full Outer Join 있음

- ✓ Self Join: 동일한 테이블끼리 조인 하는 것

CROSS JOIN

- CROSS JOIN은 특별한 키워드 없이 SELECT 문의 FROM 절에 사원(EMP) 테이블과 부서(DEPT) 테이블을 콤마로 연결하여 연속하여 기술하는 것으로 Cartesian Product 라고도 함

예

```
SELECT *
FROM EMP, DEPT;
```

C:\WINDOWS\system32\cmd.exe - sqlplus scott/tiger

```
SOL> SELECT *
2 FROM EMP, DEPT;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	7802	80/12/17	800		20	10	ACCOUNTING	NEW YORK
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30	10	ACCOUNTING	NEW YORK
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30	10	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	7839	81/04/02	2975		20	10	ACCOUNTING	NEW YORK
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30	10	ACCOUNTING	NEW YORK
7698	BLAKE	MANAGER	7839	81/05/01	2850		30	10	ACCOUNTING	NEW YORK
7782	CLARK	MANAGER	7839	81/06/08	2450		10	10	ACCOUNTING	NEW YORK
7788	SCOTT	ANALYST	7566	87/04/19	3000		20	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT		81/11/17	5000		10	10	ACCOUNTING	NEW YORK
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30	10	ACCOUNTING	NEW YORK
7876	ADAMS	CLERK	7788	87/05/23	1100		20	10	ACCOUNTING	NEW YORK
7900	JAMES	CLERK			950		30	10	ACCOUNTING	NEW YORK
7902	FORD	ANALYST			3000		20	10	ACCOUNTING	NEW YORK
7934	MILLER	CLERK			1300		10	10	ACCOUNTING	NEW YORK
7989	SMITH	CLERK			800		20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN			1600		30	20	RESEARCH	DALLAS
7566	JONES	MANAGER			2975		20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN			1250	1400	30	20	RESEARCH	DALLAS
7698	BLAKE	MANAGER			2850		30	20	RESEARCH	DALLAS
7782	CLARK	MANAGER			2450		10	20	RESEARCH	DALLAS
7788	SCOTT	ANALYST			3000		20	20	RESEARCH	DALLAS
7839	KING	PRESIDENT			5000		10	20	RESEARCH	DALLAS
7844	TURNER	SALESMAN			1500	0	30	20	RESEARCH	DALLAS
7876	ADAMS	CLERK			1100		20	20	RESEARCH	DALLAS
7900	JAMES	CLERK			950		30	20	RESEARCH	DALLAS
7902	FORD	ANALYST			3000		20	20	RESEARCH	DALLAS
7934	MILLER	CLERK			1300		10	20	RESEARCH	DALLAS
7989	SMITH	CLERK			800		20	30	SALES	CHICAGO
7499	ALLEN	SALESMAN			1600	300	30	30	SALES	CHICAGO
7521	WARD	SALESMAN			1250	500	30	30	SALES	CHICAGO
7566	JONES	MANAGER			2975		20	30	SALES	CHICAGO
7654	MARTIN	SALESMAN			1250	1400	30	30	SALES	CHICAGO
7698	BLAKE	MANAGER			2850		30	30	SALES	CHICAGO
7782	CLARK	MANAGER			2450		10	30	SALES	CHICAGO
7788	SCOTT	ANALYST			3000		20	40	OPERATIONS	BOSTON
7839	KING	PRESIDENT			5000		10	40	OPERATIONS	BOSTON
7844	TURNER	SALESMAN			1500	0	30	40	OPERATIONS	BOSTON
7876	ADAMS	CLERK			1100		20	40	OPERATIONS	BOSTON
7900	JAMES	CLERK			950		30	40	OPERATIONS	BOSTON
7902	FORD	ANALYST			3000		20	40	OPERATIONS	BOSTON
7934	MILLER	CLERK			1300		10	40	OPERATIONS	BOSTON
7989	SMITH	CLERK			800		20	30	SALES	CHICAGO
7499	ALLEN	SALESMAN			1600	300	30	30	SALES	CHICAGO
7521	WARD	SALESMAN			1250	500	30	30	SALES	CHICAGO
7566	JONES	MANAGER			2975		20	30	SALES	CHICAGO
7654	MARTIN	SALESMAN			1250	1400	30	30	SALES	CHICAGO

C:\WINDOWS\system32\cmd.exe - sqlplus scott/tiger

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7698	BLAKE	MANAGER	7839	81/05/01	2850		30	30	SALES	CHICAGO
7782	CLARK	MANAGER	7839	81/06/08	2450		10	30	SALES	CHICAGO
7788	SCOTT	ANALYST	7566	87/04/19	3000		20	30	SALES	CHICAGO
7839	KING	PRESIDENT		81/11/17	5000		10	30	SALES	CHICAGO
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30	30	SALES	CHICAGO
7876	ADAMS	CLERK	7788	87/05/23	1100		20	30	SALES	CHICAGO
7900	JAMES	CLERK	7698	81/12/03	950		30	30	SALES	CHICAGO
7902	FORD	ANALYST	7566	81/12/03	3000		20	30	SALES	CHICAGO
7934	MILLER	CLERK	7782	82/01/23	1300		10	30	SALES	CHICAGO
7989	SMITH	CLERK	7902	80/12/17	800		20	40	OPERATIONS	BOSTON
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30	40	OPERATIONS	BOSTON
7521	WARD	SALESMAN			1250	500	30	40	OPERATIONS	BOSTON
7566	JONES	MANAGER	7839	81/04/02	2975		20	40	OPERATIONS	BOSTON
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30	40	OPERATIONS	BOSTON
7698	BLAKE	MANAGER	7839	81/05/01	2850		30	40	OPERATIONS	BOSTON
7782	CLARK	MANAGER	7839	81/06/08	2450		10	40	OPERATIONS	BOSTON
7788	SCOTT	ANALYST	7566	87/04/19	3000		20	40	OPERATIONS	BOSTON
7839	KING	PRESIDENT		81/11/17	5000		10	40	OPERATIONS	BOSTON
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30	40	OPERATIONS	BOSTON
7876	ADAMS	CLERK	7788	87/05/23	1100		20	40	OPERATIONS	BOSTON
7900	JAMES	CLERK	7698	81/12/03	950		30	40	OPERATIONS	BOSTON
7902	FORD	ANALYST	7566	81/12/03	3000		20	40	OPERATIONS	BOSTON
7934	MILLER	CLERK	7782	82/01/23	1300		10	40	OPERATIONS	BOSTON

56 개의 행이 선택되었습니다.

CROSS JOIN

- ❖ CROSS JOIN의 결과 얻어지는 컬럼의 수는 사원 테이블의 컬럼의 수(8)와 부서 테이블의 컬럼의 수(3)를 더한 것이므로 11
- ❖ 행의 수는 사원 한 명에 대해서 DEPT 테이블의 모든 행과 결합되기에 2개 테이블의 행의 개수를 곱한 것 만큼 조회
- ❖ CROSS JOIN의 결과를 보면 사원 테이블에 부서에 대한 상세정보가 결합되긴 했지만, 조인 될 때 아무런 조건을 제시하지 않았기에 사원 한 명에 대해서 DEPT 테이블의 모든 행의 데이터와 결합 된 형태이기에 CROSS JOIN의 결과는 아무런 의미를 갖지 못하는 경우가 많음
- ❖ 조인 결과가 의미를 갖으려면 조인할 때 조건을 지정해야 하는 경우가 대부분



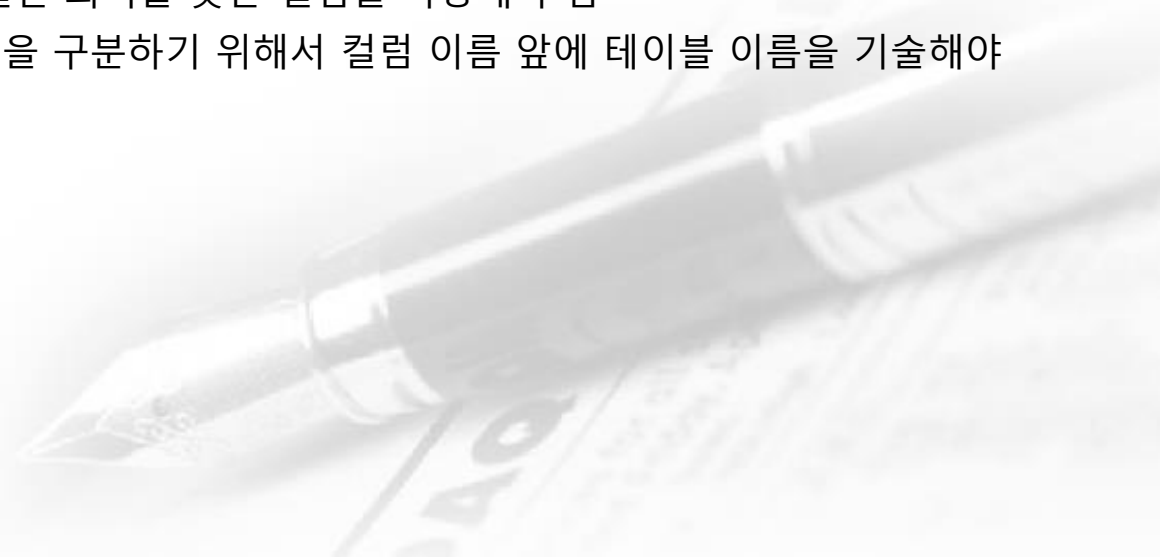
EQUI JOIN

- ❖ EQUI JOIN은 가장 많이 사용하는 조인 방법으로서 조인 대상이 되는 두 테이블에서 동일한 의미를 갖는 컬럼의 값이 일치되는 행을 연결하여 결과를 생성하는 조인 방법
- ❖ 다음은 사원 정보를 출력할 때 각 사원들이 소속된 부서의 상세 정보를 출력하기 위해서 두 개의 테이블을 조인한 경우

예

```
SELECT *  
FROM EMP, DEPT  
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

- ❖ 사원(EMP) 테이블과 부서(DEPT) 테이블의 공통 컬럼인 DEPTNO의 값이 일치(=)되는 조건을 WHERE 절에 기술하여 사용
- ❖ 테이블을 조인하려면 일치되는 동일한 의미를 갖는 컬럼을 사용해야 함
- ❖ 컬럼의 이름이 같은 경우 컬럼 이름을 구분하기 위해서 컬럼 이름 앞에 테이블 이름을 기술해야 함



EQUI JOIN

- ❖ 다음은 두 테이블을 조인한 결과로 살펴보면 다음과 같이 부서 번호를 기준으로 같은 값을 가진 사원 테이블의 컬럼과 부서 테이블의 컬럼이 결합

C:\WINDOWS\system32\cmd.exe - sqlplus scott/tiger

```
SQL> SELECT *  
2 FROM EMP, DEPT  
3 WHERE EMP.DEPTNO=DEPT.DEPTNO;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7782	CLARK	MANAGER	7839	81/06/09	2450		10	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT		81/11/17	5000		10	10	ACCOUNTING	NEW YORK
7934	MILLER	CLERK	7782	82/01/23	1300		10	10	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	7839	81/04/02	2975		20	20	RESEARCH	DALLAS
7902	FORD	ANALYST	7566	81/12/03	3000		20	20	RESEARCH	DALLAS
7876	ADAMS	CLERK	7788	87/05/23	1100		20	20	RESEARCH	DALLAS
7369	SMITH	CLERK	7902	80/12/17	800		20	20	RESEARCH	DALLAS
7788	SCOTT	ANALYST	7566	87/04/19	3000		20	20	RESEARCH	DALLAS
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30	30	SALES	CHICAGO
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30	30	SALES	CHICAGO
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30	30	SALES	CHICAGO
7900	JAMES	CLERK	7698	81/12/03	950		30	30	SALES	CHICAGO
7698	BLAKE	MANAGER	7839	81/05/01	2850		30	30	SALES	CHICAGO
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30	30	SALES	CHICAGO

14 개의 행이 선택되었습니다.

EQUI JOIN에 AND 연산

- ❖ EMP 테이블의 ENAME이 MILLER인 사원의 ENAME과 DEPT 테이블의 DNAME을 출력

예

```
SELECT ENAME, DNAME  
FROM EMP, DEPT  
WHERE EMP.DEPTNO=DEPT.DEPTNO  
AND ENAME='MILLER';
```

	ENAME	DNAME	
1	MILLER	ACCOUNTING	

컬럼명의 모호성 해결

- ❖ 양쪽 테이블에 동일한 컬럼이 존재하는 경우 컬럼 이름을 기재할 때 테이블 이름을 기재해야 하는데 그렇지 않으면 애매한 컬럼 이름이라고 에러가 발생

예

```
SELECT ENAME, DNAME, DEPTNO  
FROM EMP, DEPT  
WHERE EMP.DEPTNO = DEPT.DEPTNO  
AND ENAME='MILLER';
```

ORA-00918: column ambiguously defined

00918. 00000 - "column ambiguously defined"

*Cause:

*Action:

70행, 22열에서 오류 발생



컬럼명의 모호성 해결

- ❖ 동일한 이름의 컬럼은 컬럼명 앞에 테이블 명을 명시적으로 기술함으로써 컬럼이 어느 테이블 소속인지 구분할 수 있음

예

```
SELECT EMP.ENAME, DEPT.DNAME, EMP.DEPTNO  
FROM EMP, DEPT  
WHERE EMP.DEPTNO=DEPT.DEPTNO  
AND ENAME='MILLER';
```

	ENAME	DNAME	DEPTNO
1	MILLER	ACCOUNTING	10

테이블에 별칭 부여하기

- ❖ 테이블 이름에 별칭을 붙이는 방법은 FROM 절 다음에 테이블 이름을 명시하고 공백을 둔 다음에 별칭을 지정
- ❖ 이후 절에서 테이블 이름을 사용할 때는 별칭을 사용해야 함

예

```
SELECT E.ENAME, D.DNAME, E.DEPTNO, D.DEPTNO  
FROM EMP E, DEPT D  
WHERE E.DEPTNO = D.DEPTNO  
AND E.ENAME='MILLER';
```



HASH JOIN

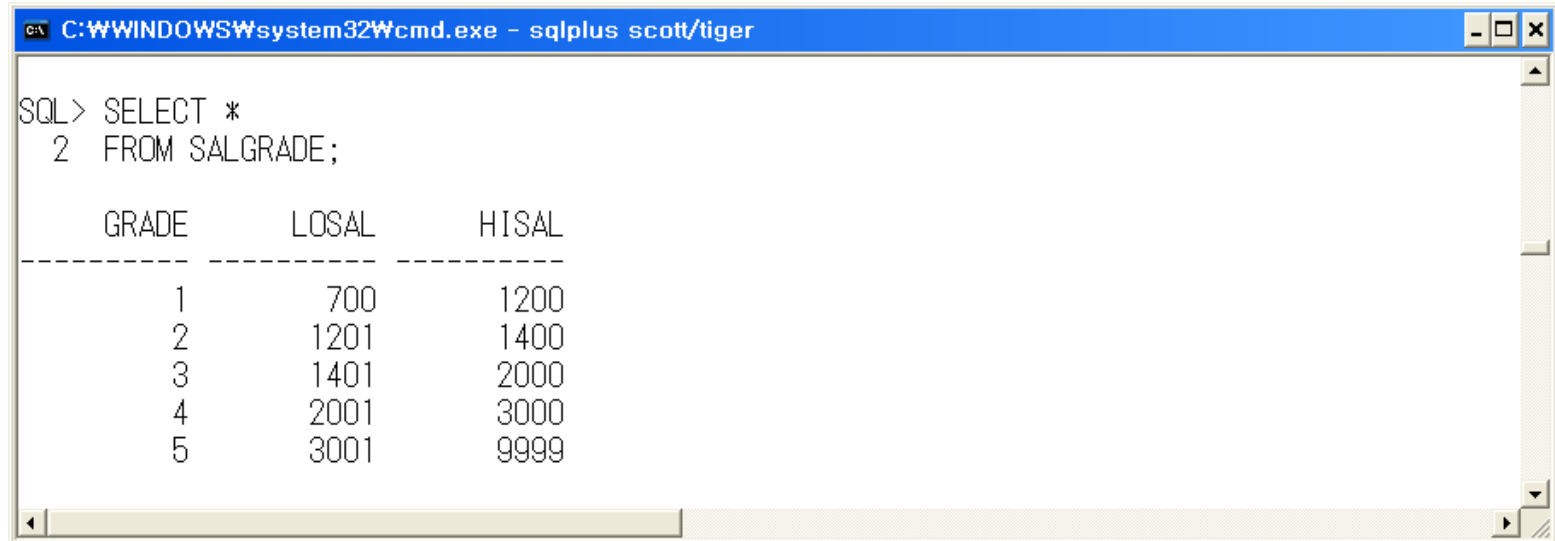
- ❖ 해시 함수는 테이블을 해시 메모리에 적재한 후에 해시 함수로써 연결하는 방법
- ❖ 해시 조인은 EQUI 조인만 사용 가능한 방법
- ❖ 해시 조인은 먼저 선행 테이블을 결정하고 선행 테이블에서 주어진 조건(Where구)에 해당하는 행을 선택하는 방식
- ❖ 해당 행이 선택되면 조인 키(Join Key)를 기준으로 해시 함수를 사용해서 해시 테이블을 메인 메모리(Main Memory)에 생성하고 후행 테이블에서 주어진 조건에 만족하는 행을 찾는 방식
- ❖ 후행 테이블의 조인 키를 사용해서 해시 함수를 적용하여 해당 버킷을 검색하는 방식



NON-EQUI JOIN

- ❖ NON-EQUI JOIN은 특정 범위 내에 있는지를 조사하기 위해서 WHERE 절에 조인 조건을 = 연산자 이외의 비교 연산자를 사용
- ❖ 급여 등급 테이블(SALGRADE)을 데이터 확인

예 SELECT * FROM SALGRADE;



```
SQL> SELECT *  
2 FROM SALGRADE;
```

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

NON-EQUI JOIN

- ❖ 급여 등급 테이블(salgrade)에는 급여에 대한 등급을 나누어 놓음
- ❖ 급여의 등급은 총 5등급으로 나누어져 있으며, 1등급은 급여가 700부터 1200 사이이고, 2등급은 1201부터 1400 사이이고, 3등급은 1401부터 2000 사이이고, 4등급은 2001부터 3000사이이고, 5등급이면 3001부터 9999사이
- ❖ 급여 등급을 5개로 나누어 놓은 salgrade에서 정보를 얻어 와서 각 사원의 급여 등급을 조회하고자 하는 경우에 사원(emp) 테이블과 급여 등급(salgrade) 테이블을 조인해야 함

예

```
SELECT ENAME, SAL, GRADE  
FROM EMP, SALGRADE  
WHERE SAL BETWEEN LOSAL AND HISAL;
```



```
SQL> SELECT ENAME, SAL, GRADE  
2 FROM EMP, SALGRADE  
3 WHERE SAL BETWEEN LOSAL AND HISAL;
```

ENAME	SAL	GRADE
SMITH	800	1
JAMES	950	1
ADAMS	1100	1
WARD	1250	2
MARTIN	1250	2
MILLER	1300	2
TURNER	1500	3
ALLEN	1600	3
CLARK	2450	4
BLAKE	2850	4
JONES	2975	4

ENAME	SAL	GRADE
SCOTT	3000	4
FORD	3000	4
KING	5000	5

14 개의 행이 선택되었습니다.

SELF JOIN

- ❖ 조인은 두 개 이상의 서로 다른 테이블을 서로 연결하는 것뿐만 아니라 하나의 테이블 내에서 조인을 해야만 원하는 자료를 얻는 경우가 발생할 수 있음 – 동일한 의미를 갖는 컬럼이 하나의 테이블에 2개 이상 존재하는 경우 : EMP 테이블에서는 MGR 이 관리자의 EMPNO
- ❖ SELF JOIN이란 자기 자신과 조인을 맺는 것
- ❖ SMITH의 매니저 이름이 무엇인지 알아내려면 어떻게?

C:\Windows\system32\cmd.exe

```
SQL> SELECT ENAME, MGR  
2 FROM EMP;
```

ENAME	MGR
SMITH	7902
ALLEN	7698
WARD	7698
JONES	7839
MARTIN	7698
BLAKE	7839
CLARK	7839
SCOTT	7566
KING	
TURNER	7698
ADAMS	7788
JAMES	7698
FORD	7566
MILLER	7782

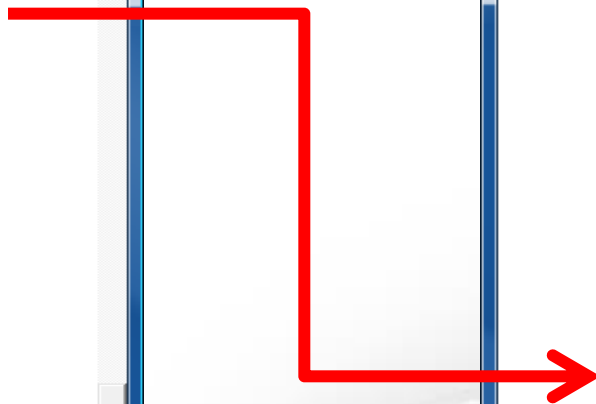
14 개의 행이 선택되었습니다.

C:\Windows\system32\cmd.exe

```
SQL> SELECT EMPNO, ENAME  
2 FROM EMP;
```

EMPNO	ENAME
7369	SMITH
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES
7902	FORD
7934	MILLER

14 개의 행이 선택되었습니다.



SELF JOIN

예

```
select employee.ename || '의 매니저는 ' || manager.ename  
from emp employee, emp manager  
where employee.mgr = manager.empno
```

Results:

	EMPLOYEE.ENAME '의 매니저는 ' MANAGER.ENAME
--	--

1	FORD의 매니저는 JONES
2	SCOTT의 매니저는 JONES
3	JAMES의 매니저는 BLAKE
4	TURNER의 매니저는 BLAKE
5	MARTIN의 매니저는 BLAKE
6	WARD의 매니저는 BLAKE
7	ALLEN의 매니저는 BLAKE
8	MILLER의 매니저는 CLARK
9	ADAMS의 매니저는 SCOTT
10	CLARK의 매니저는 KING
11	BLAKE의 매니저는 KING
12	JONES의 매니저는 KING
13	SMITH의 매니저는 FORD

OUTER JOIN

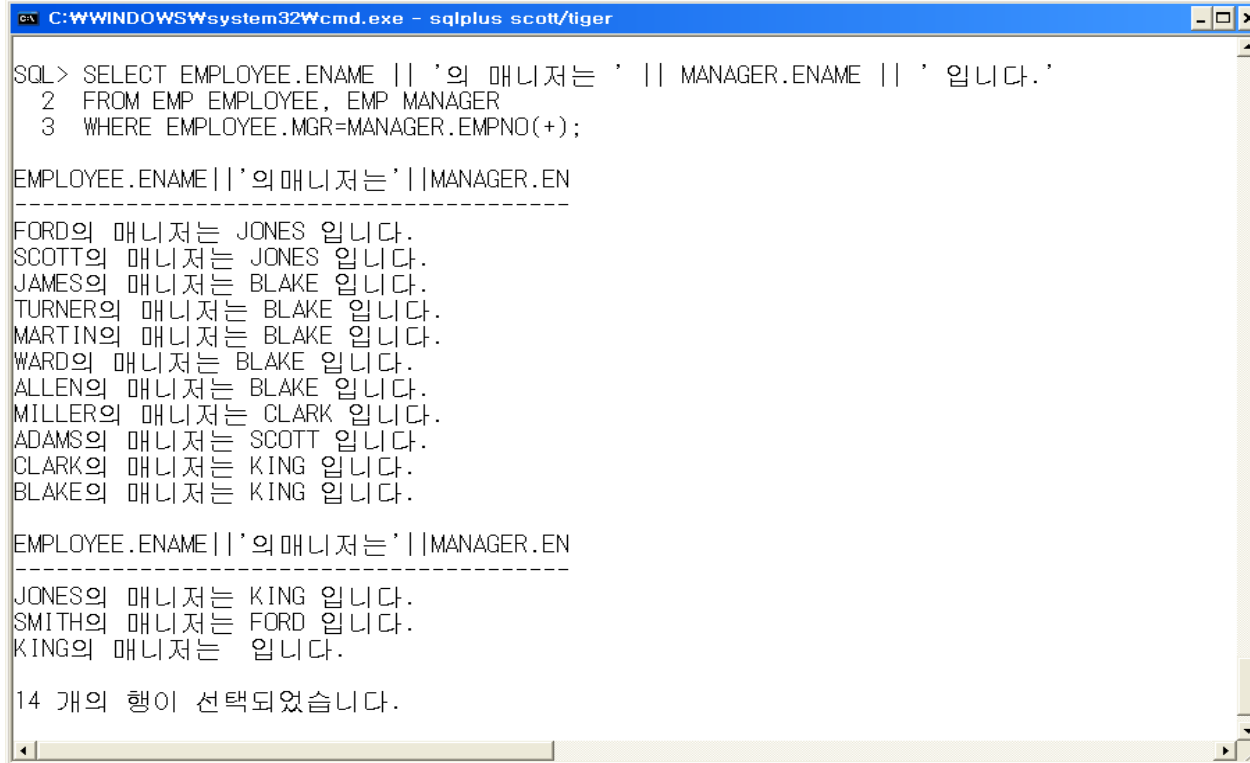
- ❖ SELF JOIN을 이용해서 특정 사원의 매니저 이름을 구했는데 결과를 살펴보면 이름이 KING인 사원 한 사람의 정보가 빠져 있음을 확인할 수 있음
- ❖ KING은 이 회사의 사장(PRESIDENT)으로 매니저가 존재하지 않으므로 MGR 컬럼 값이 NULL
- ❖ 사원 번호(EMPNO)가 NULL인 사원은 없으므로 조인 조건에 만족하지 않아서 KING은 SELF JOIN의 결과에서 배제
- ❖ 조인 조건에 만족하지 못하였더라도 해당 행을 나타내고 싶을 때에 사용하는 것이 외부 조인(OUTER JOIN)
- ❖ 외부 조인은 NULL 값이기에 배제된 행을 결과에 포함시킬 수 있으며 다음과 같이 "(+)" 기호를 조인 조건에서 정보가 부족한 컬럼 이름 뒤에 추가
- ❖ 사원 번호(EMPNO)가 NULL인 사원은 없으므로 manager.empno 뒤에 "(+)" 기호를 추가



OUTER JOIN

예

```
SELECT employee.ename || '의 매니저는'
       || manager.ename || '입니다.'
FROM emp employee, emp manager
WHERE employee.mgr = manager.empno(+);
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe - sqlplus scott/tiger". The user has entered an SQL query to perform an outer join between the EMP and EMP_MANAGER tables. The query is as follows:

```
SQL> SELECT EMPLOYEE.ENAME || '의 매니저는 ' || MANAGER.ENAME || ' 입니다.'
2  FROM EMP EMPLOYEE, EMP MANAGER
3  WHERE EMPLOYEE.MGR=MANAGER.EMPNO(+);
```

The output of the query is displayed in two sections, separated by dashed lines. The first section lists the employees and their managers:

```
EMPLOYEE.ENAME || '의 매니저는 ' || MANAGER.EN
-----
FORD의 매니저는 JONES 입니다.
SCOTT의 매니저는 JONES 입니다.
JAMES의 매니저는 BLAKE 입니다.
TURNER의 매니저는 BLAKE 입니다.
MARTIN의 매니저는 BLAKE 입니다.
WARD의 매니저는 BLAKE 입니다.
ALLEN의 매니저는 BLAKE 입니다.
MILLER의 매니저는 CLARK 입니다.
ADAMS의 매니저는 SCOTT 입니다.
CLARK의 매니저는 KING 입니다.
BLAKE의 매니저는 KING 입니다.
```

The second section shows the results for the employees who do not have a manager (JONES, SMITH, and KING):

```
EMPLOYEE.ENAME || '의 매니저는 ' || MANAGER.EN
-----
JONES의 매니저는 KING 입니다.
SMITH의 매니저는 FORD 입니다.
KING의 매니저는  입니다.
```

At the bottom of the window, it states: "14 개의 행이 선택되었습니다." (14 rows selected).

ANSI CROSS JOIN

```
SELECT *  
FROM EMP CROSS JOIN DEPT;
```

C:\Windows\system32\cmd.exe - sqlplus scott/tiger

```
SQL> SELECT *  
2 FROM EMP CROSS JOIN DEPT;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	7902	80/12/17	800		20	10	ACCOUNTING	NEW YORK
7499	ALLEN									
7521	WARD									
7566	JONES									
7654	MARTIN									
7698	BLAKE									
7782	CLARK									
7788	SCOTT									
7839	KING									
7844	TURNER									
7876	ADAMS									
7900	JAMES									
7902	FORD									
7934	MILLER									
7369	SMITH									

C:\Windows\system32\cmd.exe - sqlplus scott/tiger

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7934	MILLER	CLERK	7782	82/01/23	1300		10	30	SALES	CHICAGO
7369	SMITH	CLERK	7902	80/12/17	800		20	40	OPERATIONS	BOSTON
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30	40	OPERATIONS	BOSTON
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30	40	OPERATIONS	BOSTON
7566	JONES	MANAGER	7839	81/04/02	2975		20	40	OPERATIONS	BOSTON
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30	40	OPERATIONS	BOSTON
7698	BLAKE	MANAGER	7839	81/05/01	2850		30	40	OPERATIONS	BOSTON
7782	CLARK	MANAGER	7839	81/06/09	2450		10	40	OPERATIONS	BOSTON
7788	SCOTT	ANALYST	7566	87/04/19	3000		20	40	OPERATIONS	BOSTON
7839	KING	PRESIDENT		81/11/17	5000		10	40	OPERATIONS	BOSTON
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30	40	OPERATIONS	BOSTON
7876	ADAMS	CLERK	7788	87/05/23	1100		20	40	OPERATIONS	BOSTON
7900	JAMES	CLERK	7698	81/12/03	950		30	40	OPERATIONS	BOSTON
7902	FORD	ANALYST	7566	81/12/03	3000		20	40	OPERATIONS	BOSTON
7934	MILLER	CLERK	7782	82/01/23	1300		10	40	OPERATIONS	BOSTON

56 개의 행이 선택되었습니다.

ANSI INNER JOIN

- ❖ ANSI 조인은 FROM 다음에 INNER JOIN 이란 단어를 사용하여 조인할 테이블 이름을 명시하고 ON 절을 사용하여 조인 조건을 명시하여 다음과 같이 작성

```
SELECT * FROM table1 INNER JOIN table2  
ON table1.column1 = table2.column2
```

- ❖ ANSI 조인에서는 조인 정보를 ON절에 기술하여 조인 조건을 명확하게 지정하고 다른 조건에 대해서는 WHERE 구문에서 지정

```
SELECT ENAME, DNAME  
FROM EMP INNER JOIN DEPT  
ON EMP.DEPTNO=DEPT.DEPTNO  
WHERE ENAME='MILLER';
```



ANSI INNER JOIN

❖ USING을 이용한 조인 조건 지정하기

- ✓ 두 테이블에 각각 조인을 정의한 컬럼의 이름이 동일하다면 USING 절에서 조인할 컬럼을 지정하여 구문을 더 간단하게 표현

```
SELECT * FROM table1 JOIN table2  
USING (공통컬럼)
```

- ✓ EMP와 DEPT에 DEPTNO 라는 같은 이름의 컬럼이 있기 때문에 다음과 같이 간단하게 조인문을 기술

```
SELECT EMP.ENAME, DEPT.DNAME  
FROM EMP INNER JOIN DEPT  
USING (DEPTNO);
```



ANSI INNER JOIN

❖ NATURAL JOIN

- ✓ 두 테이블에 각각 조인을 정의한 컬럼의 이름이 동일하다면 USING 절에서 조인할 컬럼을 지정하여 구문을 더 간단하게 표현

```
SELECT * FROM table1 NATURAL JOIN table2
```

- ✓ EMP와 DEPT에 DEPTNO 라는 같은 이름의 컬럼이 있기 때문에 다음과 같이 간단하게 조인문을 기술

```
SELECT EMP.ENAME, DEPT.DNAME  
FROM EMP NATURAL JOIN DEPT;
```



ANSI OUTER JOIN

- ❖ 새로운 ANSI 구문에서 OUTER JOIN은 LEFT OUTER JOIN, RIGHT OUTER JOIN 그리고 FULL OUTER JOIN 세 가지 타입의 조인을 제공

SELECT *

FROM table1 [LEFT | RIGHT | FULL] OUTER JOIN table2

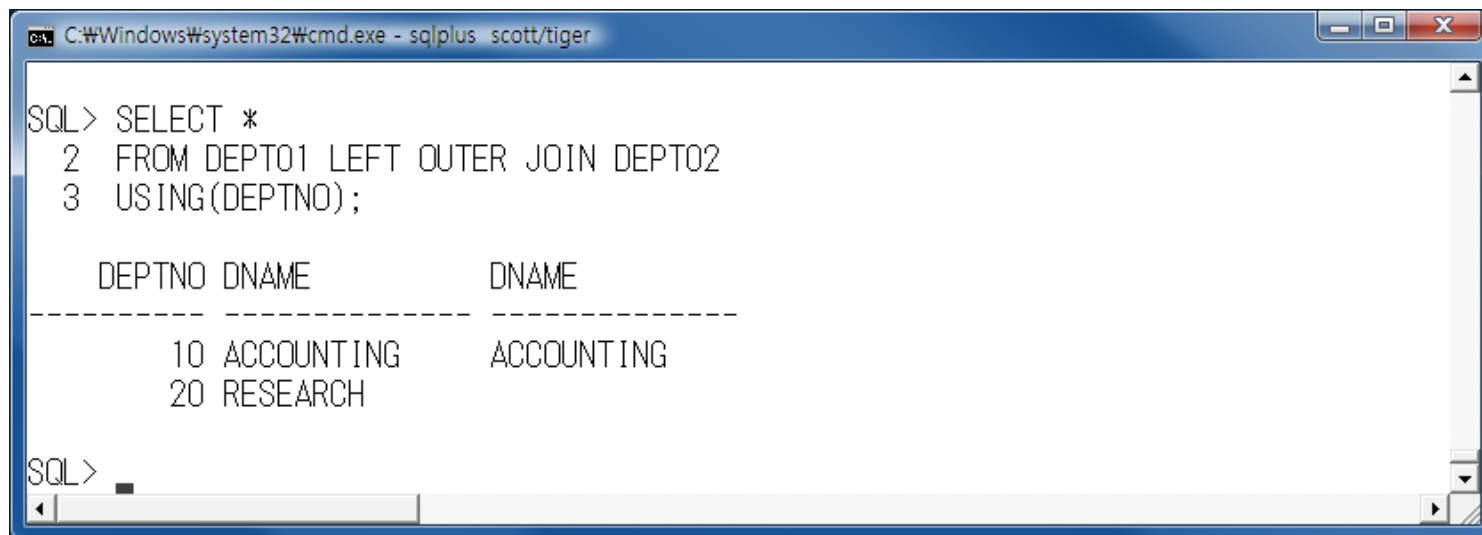
- ❖ OUTER JOIN은 어느 한쪽 테이블에는 해당하는 데이터가 존재하는데 다른 쪽 테이블에는 데이터가 존재하지 않을 경우 그 데이터가 출력되지 않는 문제점을 해결하기 위해 사용하는 조인 기법



LEFT OUTER JOIN

- ❖ DEPT01 테이블의 20번 부서와 조인할 부서번호가 DEPT02에는 없지만, 20번 부서도 출력되도록 하기 위해서 DEPT01 테이블이 왼쪽에 존재하기에 LEFT OUTER JOIN을 사용

```
SELECT *  
FROM DEPT01 LEFT OUTER JOIN DEPT02  
ON DEPT01.DEPTNO = DEPT02.DEPTNO;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT *  
2 FROM DEPT01 LEFT OUTER JOIN DEPT02  
3 USING(DEPTNO);
```

The output of the query is displayed as a table with three columns: DEPTNO, DNAME, and DNAME. The first two columns are from DEPT01, and the third column is from DEPT02. The results are as follows:

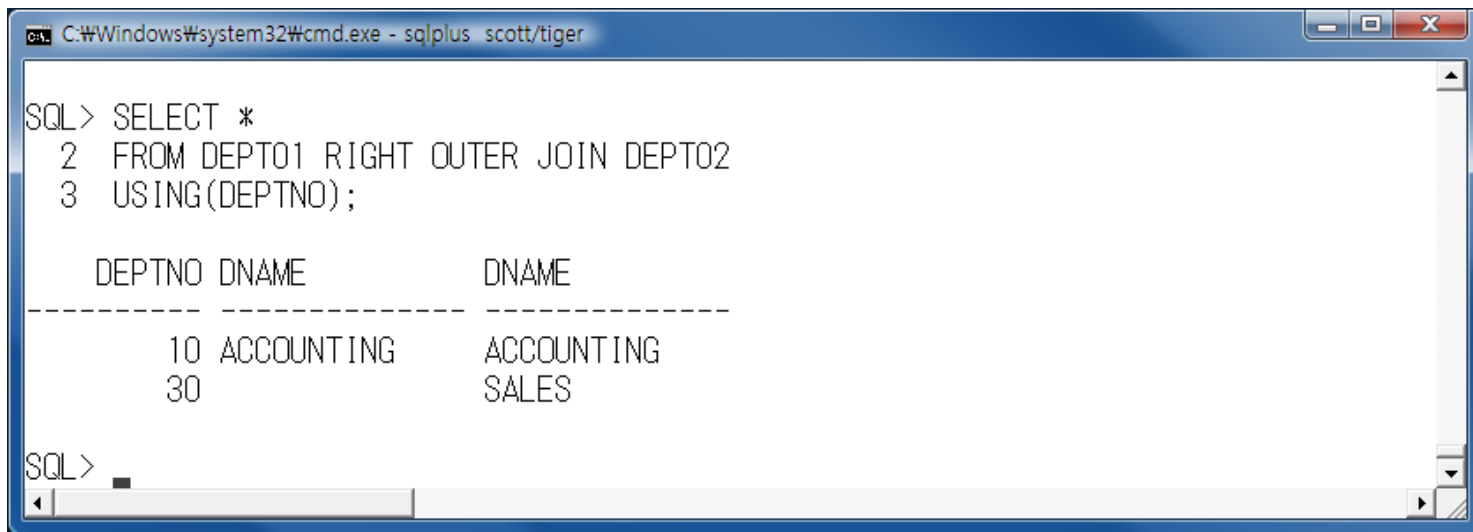
DEPTNO	DNAME	DNAME
10	ACCOUNTING	ACCOUNTING
20	RESEARCH	

The command prompt shows the SQL prompt "SQL>" at the bottom left, and the window has standard Windows window controls (minimize, maximize, close) at the top right.

RIGHT OUTER JOIN

- ❖ DEPT02 테이블에만 있는 30번 부서까지 출력되도록 하기 위해서 RIGHT OUTER JOIN을 사용

```
SELECT *  
FROM DEPT01 RIGHT OUTER JOIN DEPT02  
USING(DEPTNO);
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". The user has entered the following SQL query:

```
SQL> SELECT *  
2 FROM DEPT01 RIGHT OUTER JOIN DEPT02  
3 USING(DEPTNO);
```

The query results are displayed in a table format:

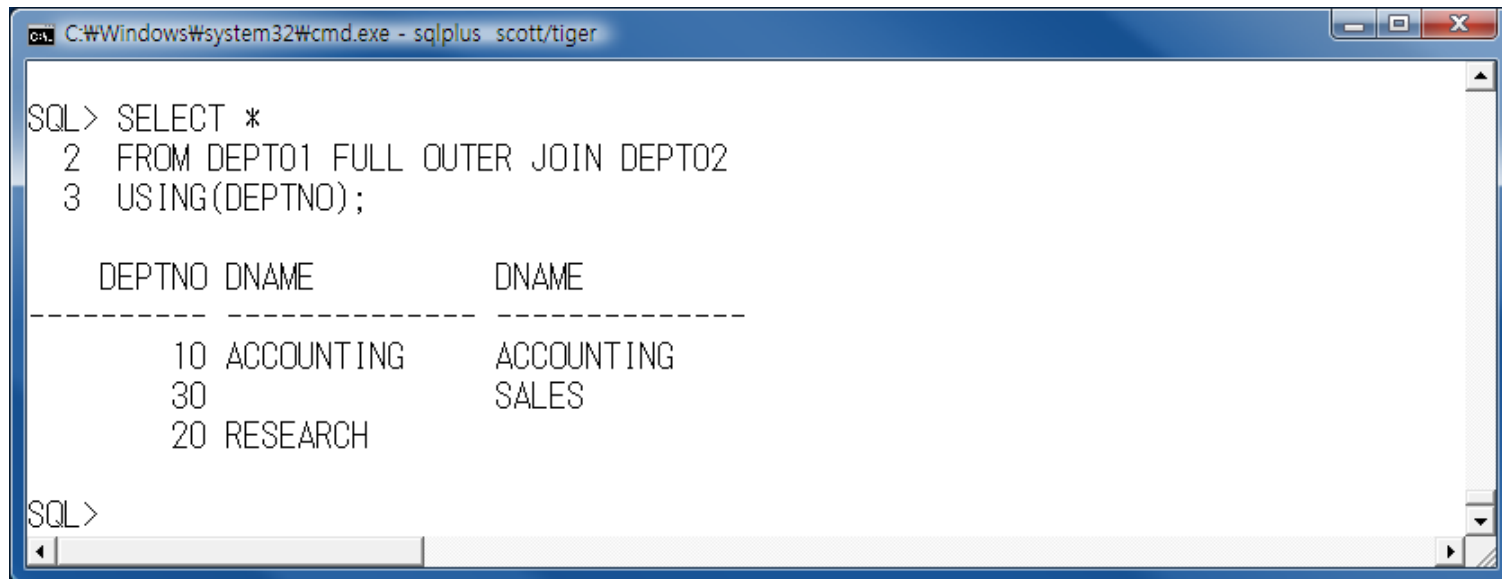
DEPTNO	DNAME	DNAME
10	ACCOUNTING	ACCOUNTING
30		SALES

The command prompt shows the prompt "SQL>" at the bottom left, indicating the query has been executed.

FULL OUTER JOIN

- ❖ FULL OUTER JOIN은 LEFT OUTER JOIN, RIGHT OUTER JOIN 을 합한 형태

```
SELECT *  
FROM DEPT01 FULL OUTER JOIN DEPT02  
USING(DEPTNO);
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the following SQL query is entered and executed:

```
SQL> SELECT *  
2 FROM DEPT01 FULL OUTER JOIN DEPT02  
3 USING(DEPTNO);
```

The output of the query is displayed as a table with three columns: DEPTNO, DNAME, and DNAME. The results are as follows:

DEPTNO	DNAME	DNAME
10	ACCOUNTING	ACCOUNTING
30		SALES
20	RESEARCH	

The command prompt shows the prompt "SQL>" at the bottom, indicating the query has been executed.

SET OPERATOR

- ❖ 하나 이상의 테이블로부터 자료를 검색하는 또 다른 방법은 SET연산자를 이용하는 방법이 있음
- ❖ SET연산자를 이용하여 여러 개의 SELECT문장을 연결하여 작성

- ❖ Syntax

```
SELECT                * | column1[, column2, column3, . . . .]
```

```
FROM                  table1
```

```
. . . . .
```

```
SET operator
```

```
SELECT                * | column1[, column2, column3, . . . .]
```

```
FROM                  table2
```

```
. . . . .
```

```
[ORDER BY            column | expression];
```



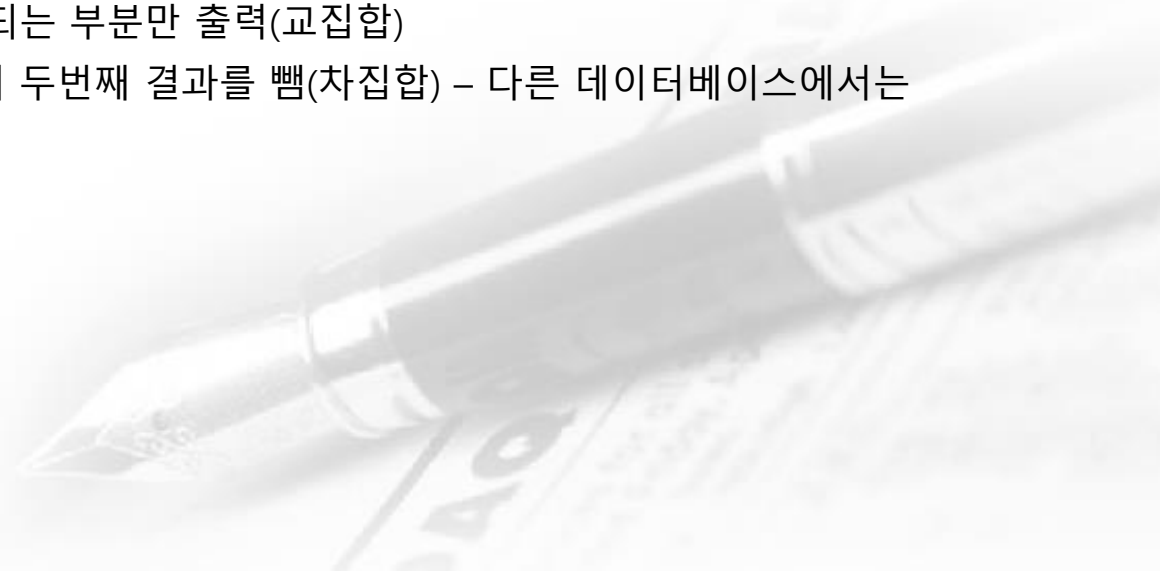
SET OPERATOR

❖ Guidelines

- ✓ 첫번째 SELECT 구문에서 기술된 열과 두번째 SELECT 구문에서 기술된 열들은 좌측부터 1대 1 대응하며 그 개수와 타입이 일치해야 함
- ✓ FROM절 뒤에 기술되는 테이블은 같을 수도 있고 다를 수도 있음
- ✓ 출력되는 HARDING은 첫번째 SELECT구문에서 기술된 열이 출력
- ✓ ORDER BY는 한번만 기술 가능하고 SELECT 구문의 마지막에 기술
- ✓ BLOB, CLOB, BFILE, LONG 형 컬럼에는 사용할 수 없음

❖ SET 연산자의 종류

- ✓ UNION :각 결과의 합(합집합:중복되는 값은 한번 출력)
- ✓ UNION ALL :각 결과의 합(합집합)
- ✓ INTERSECT :각 결과의 중복되는 부분만 출력(교집합)
- ✓ MINUS :첫번째 결과에서 두번째 결과를 뺀(차집합) – 다른 데이터베이스에서는 EXCEPT



UNION

- ❖ UNION과 UNION ALL의 차이
- ❖ 양쪽에서 검색된 결과를 모두 출력

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

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



INTERSECT

- ❖ 양쪽에서 검색된 자료만 출력

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

DEPTNO

10

20

30



MINUS

- ❖ 두번째 SELECT문장에서 검색되지 않았던 값을 첫번째 SELECT문장에서 출력
- ❖ 첫번째 SELECT문장에서 두번째 SELECT문장에서의 값을 뺀 것을 출력

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

DEPTNO

40



Sub Query

❖ Sub Query(하위 질의)

- ✓ Sub Query는 하나의 SQL 문장의 절 안에 포함된 또 하나의 SQL 문장
- ✓ Sub Query를 포함하고 있는 쿼리문을 메인 쿼리, 포함된 쿼리가 Sub Query
- ✓ Sub Query는 연산자의 오른쪽에 기술해야 하고 반드시 괄호로 감싸야 함
- ✓ Sub Query는 메인 쿼리가 실행되기 이전에 한번만 실행됨
- ✓ 단일 행 서브쿼리
 - 단일 행(Single Row) Sub Query는 수행 결과가 오직 하나의 데이터(행, row)만을 반환하는 Sub Query
 - 단일 행 Sub Query문에서는 오직 하나의 데이터(행, row)를 메인 쿼리에 보내게 되는데 메인 쿼리의 WHERE 절에서는 단일 행 비교 연산자인 =, >, >=, <, <=, <>를 사용하는 것이 가능
- ✓ 다중 행 서브쿼리
 - 다중 행 Sub Query는 Sub Query에서 반환되는 결과가 하나 이상의 행일 때 사용하는 Sub Query
 - 다중 행 Sub Query는 반드시 다중 행 연산자(Multiple Row Operator)와 함께 사용
 - IN, ANY, SOME, ALL, EXISTS(존재 여부만 확인)

단일 행 Sub Query

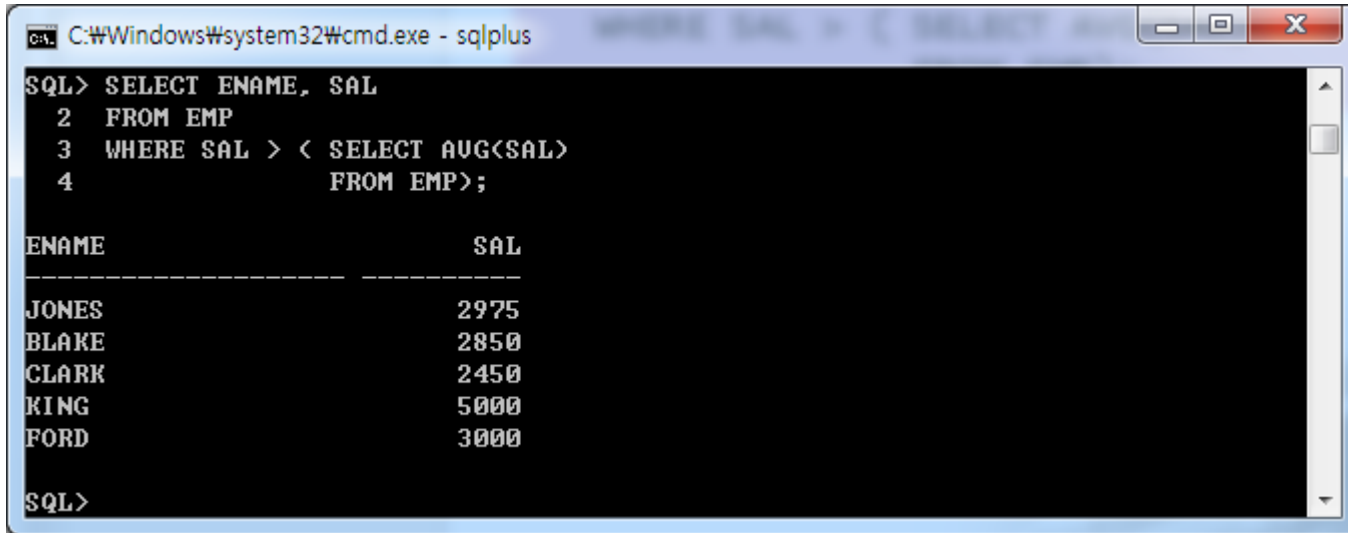
- ❖ 단일 행(Single Row) Sub Query는 수행 결과가 오직 하나의 데이터(행, row)만을 반환하는 Sub Query
- ❖ 단일 행 Sub Query문에서는 오직 하나의 데이터(행, row)를 메인 쿼리에 보내게 되는데 메인 쿼리의 WHERE 절에서는 단일 행 비교 연산자인 $=$, $>$, $>=$, $<$, $<=$, $<>$ 를 사용하는 것이 가능



Sub Query에서 그룹 함수

- ❖ 평균 급여를 구하는 쿼리문을 Sub Query로 사용하여 평균 급여보다 더 많은 급여를 받는 사원을 검색하는 문장

```
SELECT ENAME, SAL  
FROM EMP  
WHERE SAL > ( SELECT AVG(SAL)  
              FROM EMP);
```



```
SQL> SELECT ENAME, SAL  
2 FROM EMP  
3 WHERE SAL > ( SELECT AVG(SAL)  
4              FROM EMP);  
  
ENAME          SAL  
-----  
JONES          2975  
BLAKE          2850  
CLARK          2450  
KING           5000  
FORD           3000  
  
SQL>
```

다중 행 Sub Query

- ❖ 다중 행 Sub Query는 Sub Query에서 반환되는 결과가 하나 이상의 행일 때 사용하는 Sub Query
- ❖ 다중 행 Sub Query는 반드시 다중 행 연산자(Multiple Row Operator)와 함께 사용

종류	의미
IN	메인 쿼리의 비교 조건('=' 연산자로 비교할 경우)이 Sub Query의 결과 중에서 하나라도 일치하면 참
ANY, SOME	메인 쿼리의 비교 조건이 Sub Query의 검색 결과와 하나 이상이 일치하면 참
ALL	메인 쿼리의 비교 조건이 Sub Query의 검색 결과와 모든 값이 일치하면 참
EXIST	메인 쿼리의 비교 조건이 Sub Query의 결과 중에서 만족하는 값이 하나라도 존재하면 참

IN 연산자

- ❖ IN은 반환되는 여러 개의 행 중에서 하나만 참이 되어도 참이 되는 연산
- ❖ EMP 테이블에서 부서(DEPTNO)별로 가장 급여를 많이 받는 직원들과 동일한 급여를 받는 직원 번호(EMPNO), 직원이름(ENAME), 급여(SAL), 부서번호(DEPTNO)를 출력

```
SELECT EMPNO, ENAME, SAL, DEPTNO  
FROM EMP  
WHERE SAL = (SELECT MAX(SAL)  
              FROM EMP  
              GROUP BY DEPTNO);
```



SQL Error [1427] [21000]: ORA-01427: 단일 행 하위 질의에 2개 이상의 행이 리턴되었습니다.

세부사항(D) >>



IN 연산자

- ❖ EMP 테이블에서 부서(DEPTNO)별로 가장 급여를 많이 받는 직원들과 동일한 급여를 받는 직원 번호(EMPNO), 직원 이름(ENAME), 급여(SAL), 부서번호(DEPTNO)를 출력(IN 연산자 이용)

```
SELECT EMPNO, ENAME, SAL, DEPTNO  
FROM EMP  
WHERE SAL IN (SELECT MAX(SAL)  
FROM EMP  
GROUP BY DEPTNO);
```

	EMPNO	ENAME	SAL	DEPTNO
1	7,698	BLAKE	2,850	30
2	7,788	SCOTT	3,000	20
3	7,839	KING	5,000	10
4	7,902	FORD	3,000	20

IN 연산자

- ❖ 결과가 2개 이상 구해지는 쿼리문을 Sub Query로 기술할 경우에는 다중 행 연산자와 함께 사용
- ❖ EMP 테이블에서 SAL이 3000 이상 받는 사원이 소속된 부서(10번, 20번)와 동일한 부서에서 근무하는 사원의 ENAME, SAL, DEPTNO 를 검색

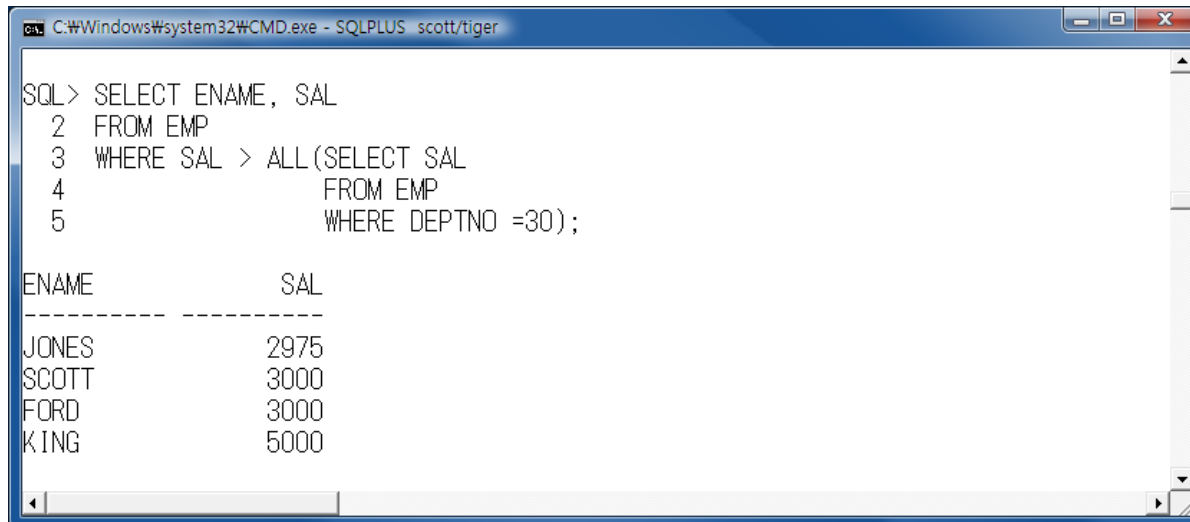
```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE DEPTNO IN ( SELECT DISTINCT DEPTNO
                  FROM EMP
                  WHERE SAL >= 3000 );
```

	ENAME	SAL	DEPTNO
1	SMITH	800	20
2	JONES	2,975	20
3	SCOTT	3,000	20
4	ADAMS	1,100	20
5	FORD	3,000	20
6	CLARK	2,450	10
7	KING	5,000	10
8	MILLER	1,300	10

ALL 연산자

- ❖ ALL 조건은 메인 쿼리의 비교 조건이 Sub Query의 검색 결과와 모든 값이 일치하면 참
- ❖ 찾아진 값에 대해서 AND 연산을 해서 모두 참이면 참
- ❖ > ALL 은 "모든 비교값 보다 크냐"고 묻는 것이 되므로 최대값보다 더 크면 참
- ❖ DEPTNO가 30번인 소속 사원들 중에서 급여를 가장 많이 받는 사원보다 더 많은 급여를 받는 사람의 이름, 급여를 출력하는 쿼리문을 작성

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ALL(SELECT SAL
                  FROM EMP
                  WHERE DEPTNO =30);
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\CMD.exe - SQLPLUS scott/tiger". The user has entered the following SQL query:

```
SQL> SELECT ENAME, SAL
2 FROM EMP
3 WHERE SAL > ALL(SELECT SAL
4                 FROM EMP
5                 WHERE DEPTNO =30);
```

The output of the query is displayed as a table with two columns: ENAME and SAL. The results are as follows:

ENAME	SAL
JONES	2975
SCOTT	3000
FORD	3000
KING	5000

ALL 연산자

- ❖ DEPTNO가 30번인 소속 직원들 중에서 급여를 가장 많이 받는 직원보다 더 많은 급여를 받는 사람의 이름, 급여를 출력하는 쿼리문을 작성

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > (SELECT MAX(SAL)
              FROM EMP
              WHERE DEPTNO =30);
```

	ENAME	SAL
1	JONES	2,975
2	SCOTT	3,000
3	KING	5,000
4	FORD	3,000




ANY 연산자

- ❖ ANY 조건은 메인 쿼리의 비교 조건이 Sub Query의 검색 결과와 하나 이상만 일치하면 참
- ❖ > ANY는 찾아진 값에 대해서 하나라도 크면 참
- ❖ 찾아진 값 중에서 가장 작은 값 즉, 최소값 보다 크면 참
- ❖ 다음은 EMP 테이블에서 DEPTNO가 30번인 직원들의 SAL 중 가장 작은 값(950)보다 많은 급여를 받는 직원의 이름, 급여를 출력하는 예제

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ANY ( SELECT SAL
                   FROM EMP
                   WHERE DEPTNO = 30 );
```

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ( SELECT MIN(SAL)
              FROM EMP
              WHERE DEPTNO = 30 );
```



ANY 연산자

- ❖ 다음은 EMP 테이블에서 DEPTNO가 30번인 직원들의 SAL 중 가장 작은 값(950)보다 많은 급여를 받는 직원의 이름, 급여를 출력하는 예제

	EMP ENAME	SAL
1	ALLEN	1,600
2	WARD	1,250
3	JONES	2,975
4	MARTIN	1,250
5	BLAKE	2,850
6	CLARK	2,450
7	SCOTT	3,000
8	KING	5,000
9	TURNER	1,500
10	ADAMS	1,100
11	FORD	3,000
12	MILLER	1,300

EXISTS 연산자

- ❖ EXISTS는 Sub Query로 어떤 데이터 존재 여부를 확인하는 연산자
- ❖ EXISTS의 결과는 참 과 거짓이 반환
- ❖ EMP 테이블에서 SAL 이 2000 이 넘는 사원이 있으면 ENAME, DNAME, SAL을 조회
SELECT ENAME, DNAME, SAL
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
AND EXISTS (SELECT 1 FROM EMP WHERE SAL > 2000);

	EMP ENAME	EMP DNAME	EMP SAL
1	CLARK	ACCOUNTING	2,450
2	KING	ACCOUNTING	5,000
3	MILLER	ACCOUNTING	1,300
4	JONES	RESEARCH	2,975
5	FORD	RESEARCH	3,000
6	ADAMS	RESEARCH	1,100
7	SMITH	RESEARCH	800
8	SCOTT	RESEARCH	3,000
9	WARD	SALES	1,250
10	TURNER	SALES	1,500
11	ALLEN	SALES	1,600
12	JAMES	SALES	950
13	BLAKE	SALES	2,850
14	MARTIN	SALES	1,250

프로시저

❖ PROCEDURE

- ✓ 절차형 SQL을 활용하여 특정 기능을 수행하는 일종의 트랜잭션 언어로 호출을 통해 실행되어 미리 저장해 놓은 SQL 작업을 수행
- ✓ 프로시저는 데이터베이스에 저장되어 수행되기 때문에 스토어드(Stored) 프로시저라고도 함
- ✓ 프로시저는 시스템의 일일 마감 작업, 일괄(Batch) 작업 등에 주로 사용
- ✓ 프로시저 생성

```
CREATE [OR REPLACE] PROCEDURE 프로시저명(파라미터) [지역변수 선언]
BEGIN
```

```
    프로시저 BODY;
```

```
END;
```

- ✓ 프로시저 실행

```
EXECUTE 프로시저명;
EXEC 프로시저명;
CALL 프로시저명;
```
- ✓ 프로시저 제거

```
DROP PROCEDURE 프로시저명;
```
- ✓ 프로시저 에러 확인

```
SHOW ERRORS;
```



커서(Cursor)

- ❖ 쿼리문의 처리 결과가 저장되어 있는 메모리 공간을 가리키는 포인터(Pointer)
- ❖ 커서는 내부에서 자동으로 생성되어 사용되는 묵시적 커서와, 사용자가 직접 정의해서 사용하는 명시적 커서가 있다.
- ❖ 커서의 수행은 열기(Open), 패치(Fetch), 닫기(Close)의 세 단계로 진행
- ❖ 묵시적 커서(Implicit Cursor)
 - ✓ DBMS 자체적으로 열고(Open) 패치(Fetch)되어 사용이 끝나면 닫히지만(Close) 커서의 속성을 조회하여 사용된 쿼리 정보를 열람하는 것이 가능하다.
 - ✓ 커서 속성
 - SQL%FOUND: 쿼리 수행의 결과로 패치(Fetch)된 튜플 수가 1개 이상이면 TRUE
 - SQL%NOTFOUND: 쿼리 수행의 결과로 패치(Fetch)된 튜플 수가 0개이면 TRUE
 - SQL%ROWCOUNT: 쿼리 수행의 결과로 패치(Fetch)된 튜플 수를 반환
 - SQL%ISOPEN: 커서가 열린(Open) 상태이면 TRUE •묵시적 커서는 자동으로 생성된 후 자동으로 닫히기 때문에 항상 FALSE
- ❖ 명시적 커서(Explicit Cursor)
 - ✓ 사용자가 직접 정의해서 사용하는 커서로 주로 절차형 SQL에서 SELECT문의 결과로 반환되는 여러 튜플들을 제어하기 위해 사용
 - ✓ 커서는 기본적으로 '열기(Open) - 패치(Fetch) - 닫기 (Close)' 순으로 이루어지며, 명시적 커서로 사용하기 위해서는 열기 전에 선언(Declare)을 해야 한다.

트리거

❖ 트리거(Trigger)

- ✓ 데이터베이스 시스템에서 데이터의 삽입(Insert), 갱신(Update), 삭제>Delete) 등의 이벤트(Event)가 발생할 때 마다 관련 작업이 자동으로 수행되는 절차형 SQL
- ✓ 트리거는 데이터베이스에 저장되며, 데이터 변경 및 무결성 유지, 로그 메시지 출력 등의 목적으로 사용
- ✓ 트리거 생성

```
CREATE [OR REPLACE] TRIGGER 트리거명 [동작시기 옵션] [동작 옵션]
ON 테이블명
REFERENCING [NEW | OLD] AS 테이블명
FOR EACH ROW
    [WHEN 조건식]
BEGIN
    트리거 BODY;
END;
```

- ✓ 트리거 제거

```
DROP TRIGGER 트리거명;
```



함수

❖ 사용자 정의 함수

- ✓ 프로시저와 유사하게 SQL을 사용하여 일련의 작업을 연속적으로 처리하며 종료 시 처리 결과를 단일 값으로 반환하는 절차형 SQL
- ✓ 사용자 정의 함수는 데이터베이스에 저장되어 SELECT, INSERT, DELETE, UPDATE 등 DML문의 호출에 의해 실행된다.
- ✓ 사용자 정의 함수는 예약어 RETURN을 통해 값을 반환하기 때문에 출력 파라미터가 없다.

✓ 함수 생성

CREATE [OR REPLACE] FUNCTION 사용자 정의 함수명(파라미터)

[지역변수 선언]

BEGIN

 사용자 정의 함수 BODY;

 RETURN 반환값;

END;

✓ 함수 호출

SELECT 사용자 정의 함수명 FROM 테이블명;

INSERT INTO 테이블명(속성명) VALUES (사용자 정의 함수명);

DELETE FROM 테이블명 WHERE 속성명 = 사용자 정의 함수명;

UPDATE 테이블명 SET 속성명 = 사용자 정의 함수명;

✓ 함수 삭제: DROP FUNCTION 사용자 정의 함수명;

Sequence

- ❖ 기본 키가 유일한 값을 갖도록 사용자가 직접 값을 생성해내려면 부담이 큼
- ❖ Sequence는 테이블 내의 유일한 숫자를 자동으로 생성하는 자동 번호 발생기로 Sequence를 기본키로 사용하게 되면 사용자의 부담을 줄일 수 있음 - 다른 데이터베이스에서는 auto_increment

- ❖ Sequence를 생성하기 위한 기본 형식

CREATE SEQUENCE sequence_name

[START WITH n] ①

[INCREMENT BY n] ②

[{MAXVALUE n | NOMAXVALUE}] ③

[{MINVALUE n | NOMINVALUE}] ④

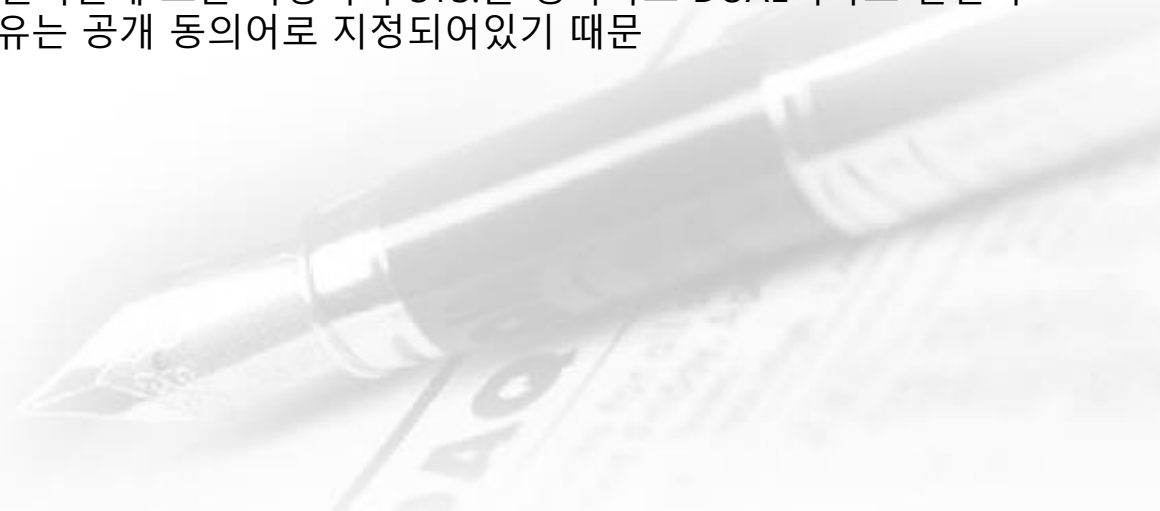
[{CYCLE | NOCYCLE}] ⑤

[{CACHE n | NOCACHE}] ⑥



SYNONYM

- ❖ 데이터베이스의 객체에 대한 소유권은 해당 객체를 생성한 사용자에게 있기 때문에 다른 사용자가 객체에 접근하기 위해서는 소유자로부터 접근 권한을 부여받아야 하며 또한 다른 사용자가 소유한 객체에 접근하기 위해서는 소유자의 이름을 객체 앞에 지정해야 함
- ❖ 객체를 조회할 때마다 일일이 객체의 소유자를 지정하는 것이 번거로울 경우 동의어를 정의하면 긴 이름대신 간단한 이름으로 접근할 수 있음
- ❖ 동의어는 개별 사용자를 대상으로 하는 비공개 동의어와 전체 사용자를 대상으로 한 공개 동의어가 있음
 - ✓ 비공개 동의어 - 객체에 대한 접근 권한을 부여받은 사용자가 정의한 동의어로 해당 사용자만 사용할 수 있음
 - ✓ 공개 동의어 - 권한을 주는 사용자가 정의한 동의어로 누구나 사용할 수 있으며 공개 동의어는 DBA 권한을 가진 사용자만이 생성할 수 있는데 SYNONYM 앞에 PUBLIC를 붙여서 정의
- ❖ DUAL은 원래 SYS가 소유하는 테이블 명이므로 다른 사용자가 DUAL 테이블에 접근하려면 SYS.DUAL로 표현해야 하는 것이 원칙인데 모든 사용자가 SYS.을 생략하고 DUAL이라고 간단하게 사용했는데 그럴 수 있었던 이유는 공개 동의어로 지정되어있기 때문



데이터 사전

❖ 개념

- ✓ 데이터 사전(Data Dictionary)에는 데이터베이스의 데이터를 제외한 모든 정보가 있다.
- ✓ 데이터 사전의 내용을 변경하는 권한은 시스템이 가지며, 사용자에게는 읽기 전용 테이블 형태로 제공되므로 단순 조회만 가능하다.
- ✓ 데이터를 제외한(데이터를 구성하는) 모든 정보라는 것은 데이터의 데이터를 의미하며 데이터 사전은 메타데이터(Meta data)로 구성되어 있다고 할 수 있다.

❖ 데이터 사전 내용

- ✓ 데이터 사전 안에 존재하는 메타데이터의 유형은 다음과 같다.
 - 사용자 정보(아이디, 패스워드 및 권한 등)
 - 데이터베이스 객체 정보(테이블, 뷰, 인덱스 등)
 - 무결성 제약 정보
 - 함수, 프로시저 및 트리거 등
- ✓ 데이터 사전 내용이 메타데이터라는 것은 모든 DBMS 제품에 공통이지만 데이터 사전을 구현하는 방법, 관리하는 방법 등의 차이로 메타데이터의 구체적인 내용은 제품마다 다르다.

❖ 데이터 사전 용도

- ✓ 사용자에게 데이터 사전은 단순 조회의 대상일 뿐이지만 데이터베이스 엔진을 이루는 컴파일러, 옵티마이저 등과 같은 구성 요소에 데이터 사전은 작업을 수행하는 데 필요한 참조 정보일 뿐만 아니라 작업의 대상

옵티마이저

❖ Query Optimizer

- ✓ 쿼리 최적화는 많은 관계형 데이터베이스 관리 시스템 및 그래프 데이터베이스와 같은 기타 데이터베이스의 기능
- ✓ 쿼리 최적화 프로그램은 가능한 쿼리 계획을 고려하여 주어진 쿼리를 실행하는 가장 효율적인 방법을 결정
- ✓ 실행 계획
 - DBMS의 옵티마이저가 수립한 SQL 코드의 실행 절차와 방법을 의미
 - EXPLAIN 명령어를 통해 확인이 가능하며, 그래픽 또는 텍스트로 표현
 - 요구사항들을 처리하기 위한 연산 순서가 적혀있으며, 연산에는 조인, 테이블 검색, 필터, 정렬 등이 있음
- ✓ RBO: 규칙 기반이라고 하며 여러 개의 규칙을 정해 놓고 부합되는 규칙 중 가장 비용이 적은 규칙을 선택하여 실행계획을 생성하는 것인데 그 비용이라는 것이 실제 SQL을 실행하여 발생하는 비용이 아닌 규칙 자체의 비용이므로 유연하지 못함
- ✓ CBO: RBO의 단점을 보완하고자 통계정보를 사용하여 SQL을 실제 실행했을 때 가장 비용이 적은 실행계획을 만들어 내는 방식으로 CBO가 무조건 RBO보다 좋은 것은 아님

