

# 4장. SELECT



# SQL 실습 준비

- 실습스크립트.txt를 열어 전체 선택한 후 복사하여 SQL Developer의 워크시트 창에 붙여 넣습니다.
- 그리고 전체 선택한 후 상단 ▶를 눌러 명령문 실행을 수행합니다.

The screenshot shows the SQL Developer interface with a script named 'Study.sql' open. The script contains the following SQL commands:

```
41 commit;
42
43 drop table salgrade;
44
45 create table salgrade
46 ( grade number(10),
47   losal number(10),
48   hisal number(10) );
49
50 insert into salgrade values(1,700,1200);
51 insert into salgrade values(2,1201,1400);
52 insert into salgrade values(3,1401,2000);
53 insert into salgrade values(4,2001,3000);
54 insert into salgrade values(5,3001,9999);
55
56 commit;
57
```

The 'Run' button (green play icon) is highlighted with a red box in the top toolbar. The toolbar also shows other icons for file operations, editing, and execution, along with a timer showing '0.125초'.



# SQL 실습 준비

- `select * from emp;` 를 입력하고 `ctrl + Enter`를 눌러 실행하면 `emp` 테이블의 결과가 출력됩니다.

```
42
43 drop table salgrade;
44
45 create table salgrade
46 ( grade number(10),
47   losal number(10),
48   hisal number(10) );
49
50 insert into salgrade values(1,700,1200);
51 insert into salgrade values(2,1201,1400);
52 insert into salgrade values(3,1401,2000);
53 insert into salgrade values(4,2001,3000);
54 insert into salgrade values(5,3001,9999);
55
56 commit;
57
58 select * from emp;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0,012초)

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7839	KING	PRESIDENT	(null)	81/11/17	5000	(null)	10
2	7698	BLAKE	MANAGER	7839	81/05/01	2850	(null)	30
3	7782	CLARK	MANAGER	7839	81/05/09	2450	(null)	10
4	7566	JONES	MANAGER	7839	81/04/01	2975	(null)	20



## 테이블에서 특정 열(Column) 선택하기

- 사원 테이블에서 사원 번호와 이름과 월급을 출력해 보겠습니다.

```
1 SELECT empno, ename, sal
2 FROM emp;
```

- 위의 SQL은 “EMPNO(사원 번호), ENAME(사원 이름), SAL(월급)을 EMP 테이블로부터 선택해서 화면에 출력하라”는 SQL입니다.
- SELECT는 선택하라는 뜻의 SQL입니다.
- SELECT절에는 테이블로부터 출력하고 싶은 열(column) 이름을 콤마(,)로 구분하여 작성합니다.
- FROM절 다음에는 데이터를 저장하고 있는 테이블명을 작성합니다.
- FROM EMP는 EMP 테이블로부터 데이터를 가져오라는 뜻입니다.
  - 1행 : empno(사원 번호), ename(사원 이름), sal(월급)을 선택해서 출력합니다.
  - 2행 : emp 테이블로부터 데이터를 가져옵니다.



## 테이블에서 특정 열(Column) 선택하기

- SQL 문장을 작성할 때는 대문자, 소문자 상관없이 수행됩니다.
- 가독성을 높이기 위해 SQL은 대문자로 작성하고 컬럼명과 테이블명은 소문자로 작성하기를 권장합니다.
- SELECT절과 FROM절을 한 줄로 작성해도 되고 여러 줄로 나누어 작성해도 수행됩니다.
- 그런데 SQL이 점점 길어지게 되므로 가급적 SELECT절과 FROM절을 각각 별도의 라인에 작성해주기를 권장합니다.

한 줄로 작성	여러 줄로 나눠서 작성
<code>SELECT empno, ename ,sal FROM emp;</code>	<code>SELECT empno, ename, sal FROM emp;</code>

- SELECT절 다음 라인에 FROM절을 작성할 때는 2칸에서 3칸 정도 공백을 주어 들여쓰기를 해주면 가독성이 좋아지므로 권장합니다.

들여쓰기 안 했을 때	들여쓰기 했을 때
<code>SELECT empno, ename, sal, deptno FROM emp;</code>	<code>SELECT empno, ename, sal, deptno FROM emp;</code>

# SQL 테이블에서 모든 열(COLUMN) 출력하기

- 사원 테이블을 모든 열(column)들을 전부 출력해 보겠습니다.

```
1 SELECT *  
2 FROM emp;
```

- 사원 테이블의 모든 열(column)과 모든 행(row)을 출력하는 SQL입니다.
- 테이블의 전체 열(column)들을 모두 결과 화면에 표시하고 싶을 때가 있습니다.
- 이럴 때는 SELECT 바로 다음에 \*(별표)를 작성하면 전체 열(column)들을 전부 조회할 수 있습니다.
  - 1행 : 모든 열을 검색합니다.
  - 2행 : emp 테이블로부터 데이터를 가져옵니다.
- 문장 맨 끝에 세미콜론( ; )은 SQL을 종료하고 실행하겠다는 표시입니다.
- 만약 \*(별표)를 사용하지 않는다면 다음의 쿼리와 같이 emp 테이블의 전체 열(column)들을 SELECT절에 일일이 나열해줘야 합니다.

```
1 SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno  
2 FROM emp;
```

# SQL 테이블에서 모든 열(COLUMN) 출력하기

- emp 테이블의 모든 컬럼을 작성하지 않고 \*(별표)를 사용하여 출력하면 간편하게 모든 컬럼을 조회할 수 있습니다.
- emp 테이블의 모든 컬럼을 출력하고 맨 끝에 다시 한번 특정 컬럼을 한번 더 출력해야 하는 경우가 있습니다.
- 이 경우 \*(별표) 앞에 '테이블명. 을 붙여 주어 작성하고 그 다음 한번 더 출력하고자 하는 컬럼명을 작성합니다.

```
1 SELECT dept.*, deptno
2 FROM dept;
```



# SQL

## 컬럼 별칭을 사용하여 출력되는 컬럼명 변경하기

- 사원 테이블의 사원 번호와 이름과 월급을 출력하는데 컬럼명을 한글로 '사원 번호', '사원 이름' 으로 출력해 보겠습니다.

```

1 SELECT empno as "사원 번호", ename as "사원 이름", sal as "Salary"
2 FROM emp;

```

사원 번호	이름	Salary
7839	KING	5000
7698	BLAKE	2850
7782	CLARK	2450
7566	JONES	2975
:	:	:

- 사원 번호와 이름, 월급, 부서 번호를 출력하는데 컬럼 별칭을 주어 출력되는 컬럼명을 변경하는 SQL 입니다.
- 출력되는 컬럼명을 변경하고자 할 때는 컬럼명 다음에 as를 작성하고 출력하고 싶은 컬럼명을 기술하면 됩니다.
- 이것을 컬럼 별칭(column alias)이라고 합니다.
- 컬럼명 empno는 한글로 '사원 번호'로 출력되고, ename은 '사원 이름'으로 출력되었습니다.
- Sal은 'Salary'로 대소문자가 구분되어 출력되었습니다.
- 대소문자를 구분하여 컬럼 별칭을 출력하고자 할 때는 컬럼 별칭 양쪽에 더블 쿼테이션 마크를 감싸 줘야합니다.





## 컬럼 별칭을 사용하여 출력되는 컬럼명 변경하기

- 컬럼 별칭에 더블 쿼테이션 마크를 감싸줘야 하는 경우는 다음과 같습니다.
    - 대소문자를 구분하여 출력할 때
    - 공백문자를 출력할 때
    - 특수문자를 출력할 때(\$, \_, #만 가능)
  - 1행 : emp 테이블로부터 컬럼 empno, ename, sal을 선택합니다. 컬럼명 empno는 한글로 '사원 번호'로 출력됩니다. 컬럼명 ename은 한글로 "사원 이름"으로 출력됩니다. 컬럼명 sal은 대소문자가 구분된 Salary로 출력됩니다.
  - 2행 : emp 테이블로부터 데이터를 가져옵니다.
- 
- 다음의 쿼리와 같이 수식을 사용하여 결과를 출력할 때 컬럼 별칭이 유용합니다.

```
1 SELECT ename, sal * (12 + 3000)
2 FROM emp;
```



# SQL

## 컬럼 별칭을 사용하여 출력되는 컬럼명 변경하기

- 수식을 사용할 경우 출력되는 컬럼명도 수식으로 출력됩니다.
- 그런데 수식명이 아닌 한글 '월급'으로 컬럼명을 출력하고 싶다면 수식 뒤에 as를 작성하고 컬럼 별칭을 사용하면 됩니다.

```
1 SELECT ename, sal * (12 + 3000) as 월급
2 FROM emp;
```

- 수식에 컬럼 별칭을 사용하면 ORDER BY절을 사용할 때 유용합니다.
- ORDER BY절에 수식명을 작성하지 않고 컬럼 별칭만 사용하면 되기 때문에 SQL 작성이 간편해집니다.



- 사원 테이블의 이름과 월급을 서로 붙여서 출력해 보겠습니다.

```
1 SELECT ename || sal
2 FROM emp;
```

ENAME  SAL
KING5000
BLAKE2850
CLARK2450
JONES2975
:

- 연결 연산자(concatenation operator)를 이용하면 컬럼과 컬럼을 서로 연결해서 출력할 수 있습니다.



## SQL 연결 연산자 사용하기(II)

- 다음의 쿼리와 같이 문자열과 연결해서 출력하면 연결 연산자를 사용한 의미가 있어집니다.

```
1 SELECT ename || '의 월급은 ' || sal || '입니다' as 월급정보
2 FROM emp;
```

### 월급정보

KING의 월급은 5000입니다.

BLAKE의 월급은 2850입니다.

CLARK의 월급은 2450입니다.

JONES의 월급은 2975입니다.

:

- 연결 연산자(concatenation operator)를 이용하면 컬럼과 컬럼을 연결해서 출력할 수 있고 위와 같이 컬럼과 문자열을 연결해서 출력할 수도 있습니다.



## SQL 연결 연산자 사용하기(II)

- 연결 연산자를 이용하여 컬럼들을 서로 연결하였다면 컬럼 별칭은 맨 마지막에 사용해야 합니다.

```
1 SELECT ename || '의 월급은 ' || job || '입니다' as 직업정보
2 FROM emp;
```

### 직업정보

KING 의 직업은 PRESIDENT 입니다

BLAKE 의 직업은 MANAGER 입니다

CLARK 의 직업은 MANAGER 입니다

JONES 의 직업은 MANAGER 입니다

MARTIN 의 직업은 SALESMAN 입니다

ALLEN 의 직업은 SALESMAN 입니다

TURNER 의 직업은 SALESMAN 입니다

JAMES 의 직업은 CLERK 입니다

:



# SQL

## 중복된 데이터를 제거해서 출력하기 (DISTINCT)

- 사원 테이블에서 직업을 출력하는데 중복된 데이터를 제외하고 출력해 보겠습니다.

```
1 SELECT DISTINCT job
2 FROM emp;
```

JOB
SALESMAN
CLERK
ANALYST
MANAGER
PRESIDENT

- 사원 테이블의 데이터 중에 직업과 부서 번호는 데이터가 중복되어 있습니다.
- 이런 컬럼의 데이터를 출력할 때 중복된 데이터를 제거하고 출력하려면 DISTINCT 키워드를 이용하면 됩니다.
- 컬럼명 앞에 DISTINCT를 작성하고 실행하면 중복행이 제거되고 UNIQUE한 값만 출력됩니다.



SQL

## 중복된 데이터를 제거해서 출력하기 (DISTINCT)

- DISTINCT 대신 UNIQUE를 사용해도 됩니다.

```
1 SELECT UNIQUE job
2 FROM emp;
```

# SQL 데이터를 정렬해서 출력하기 (ORDER BY)

- 이름과 월급을 출력하는데 월급이 낮은 직원부터 출력해 보겠습니다.

```
1 SELECT ename, sal
2 FROM emp
3 ORDER BY sal asc;
```

- 직원 테이블에서 이름과 월급을 출력하는데 월급이 낮은 직원부터 높은 직원 순으로 출력하는 SQL입니다.
- 데이터를 정렬해서 출력하려면 ORDER BY절을 사용하면 됩니다.
- ORDER BY절 다음에 정렬하고자 하는 데이터의 컬럼명을 기술합니다.
- 그리고 내림차순으로 정렬할지 오름차순으로 정렬할지 정렬 방식에 대한 옵션을 컬럼명 다음에 작성합니다.

정렬 방식	정렬 옵션	축약
오름차순	ASCENDING	ASC
내림차순	DESCENDING	DESC



# SQL 데이터를 정렬해서 출력하기 (ORDER BY)


- ORDER BY절에 ORDER BY sal ascending이라고 해도 실행되고, ascending을 ORDER BY sal asc로 축약된 키워드로 작성해도 실행이 됩니다.
- 월급이 큰 수부터 작은 수 순으로 출력하고 싶다면 ORDER BY sal desc로 작성하고 실행합니다.
  - 1~2행 : EMP 테이블로부터 이름과 월급을 선택합니다.
  - 3행 : 월급을 낮은 값부터 높은 값 순으로 정렬합니다.
- ORDER BY절은 SQL 작성 시에도 맨 마지막에 작성하고 오라클이 실행할 때도 맨 마지막에 실행합니다.

코딩 순서	SQL	실행 순서	SQL
1	SELECT ename, sal	2	SELECT ename, sal
2	FROM emp	1	FROM emp
3	ORDER BY sal asc;	3	ORDER BY sal asc;

# SQL 데이터 정렬해서 출력하기 (ORDER BY)

- ORDER BY절은 맨 마지막에 실행되기 때문에 SELECT절에 사용한 컬럼 별칭을 ORDER BY 절에 사용할 수 있습니다.

```
1 SELECT ename, sal as 월급
2 FROM emp
3 ORDER BY 월급 asc;
```



- ORDER BY절에는 다음과 같이 컬럼을 여러 개 작성할 수도 있습니다.

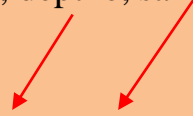
```
1 SELECT ename, deptno, sal
2 FROM emp
3 ORDER BY deptno asc, sal desc;
```

- 부서 번호를 먼저 ascending하게 정렬하고 부서 번호를 ascending하게 정렬된 것을 기준으로 월급을 descending하게 정렬하고 있습니다.

# SQL 데이터 정렬해서 출력하기 (ORDER BY)

- ORDER BY절에 컬럼명 대신 숫자를 적어줄 수도 있습니다.
- ORDER BY절에 작성한 숫자는 SELECT절 컬럼의 순서입니다.

```
1 SELECT ename, deptno, sal
2 FROM emp
3 ORDER BY 2 asc, 3 desc;
```



# SQL WHERE절 배우기 ① ( 숫자 데이터 검색 )

- 월급이 3000인 직원들의 이름, 월급, 직업을 출력해 보겠습니다.

```
1 SELECT ename, sal, job
2 FROM emp
3 WHERE sal = 3000;
```

- 월급이 3000인 직원의 이름과 월급, 직업을 출력하는 SQL입니다.
- 검색하기 원하는 조건을 WHERE절에 작성하여 데이터를 검색합니다.
- WHERE절은 FROM절 다음에 작성합니다.
  - 1~2행 : emp 테이블로부터 이름과 월급과 직업 컬럼을 선택합니다.
  - 3행 : 월급이 3000인 직원들의 데이터로만 행을 제한합니다.
- WHERE절의 검색 조건으로 사용하는 비교 연산자는 다음과 같습니다.

연산자	의미	연산자	의미	연산자	의미
>	크다	<=	작거나 같다	^=	같지 않다
<	작다	=	같다	<>	같지 않다
>=	크거나 같다	!=	같지 않다		

# SQL WHERE절 배우기 ① ( 숫자 데이터 검색 )

- WHERE절의 검색 조건으로 사용하는 비교 연산자는 다음과 같습니다.
  - 기타 비교 연산자

연산자	의미
BETWEEN AND	~ 사이에 있는
LIKE	일치하는 문자 패턴 검색
IS NULL	NULL 값인지 여부
IN	값 리스트 중 일치하는 값 검색

- 위의 비교 연산자를 사용하여 월급이 3000 이상인 직원들의 이름과 월급을 출력하는 SQL 문을 작성하면 다음과 같습니다.

```
1 SELECT ename as 이름, sal as 월급
2 FROM emp
3 WHERE sal >= 3000;
```

- WHERE절에서 `sal >= 3000`이라는 검색 조건을 주어 월급이 3000 이상인 직원들의 데이터만 검색해서 출력되었습니다.
- 위의 SQL에서 SELECT절에 컬럼 별칭을 사용했습니다.
- 그리고 출력 결과로 컬럼명을 한글로 '이름', 한글로 '월급'으로 출력되었습니다.

# SQL WHERE절 배우기 ① ( 숫자 데이터 검색 )

- 다음과 같이 컬럼 별칭인 월급을 WHERE절 검색 조건에 사용하면 실행되는지 확인해 보겠습니다.

```
1 SELECT ename as 이름, sal as 월급
2   FROM emp
3  WHERE 월급 >= 3000;
```

3행 오류:

ORA-00904: "월급": 부적합한 식별자

- 실행해보면 위와 같이 월급이 부적합한 식별자라면서 에러가 발생합니다.
- 이유는 오라클이 SQL을 실행하는 실행 순서 때문입니다.

코딩 순서	SQL	실행 순서	SQL
1	SELECT ename as 이름 sal as 월급	3	SELECT ename as 이름, sal as 월급
2	FROM emp	1	FROM emp
3	WHERE 월급 >= 3000;	2	WHERE 월급 >= 3000;

## SQL WHERE절 배우기 ① ( 숫자 데이터 검색 )

- 오라클은 FROM절을 실행하고 나서 WHERE절을 실행하기 때문에 에러가 발생합니다.
- FROM절을 실행하면서 데이터베이스에서 emp 테이블을 가져오고 WHERE절을 실행하면서 emp 테이블에서 한글로 된 '월급' 컬럼을 찾아보았지만 없기 때문에 부적합한 식별자라고 수행되면서 에러가 발생했습니다.

## SQL WHERE절 배우기 ② ( 문자와 날짜 검색 )

- 이름이 SCOTT인 사원의 이름, 월급, 직업, 입사일, 부서 번호를 출력해 보겠습니다.

```
1 SELECT ename, sal, job, hiredate, deptno
2 FROM emp
3 WHERE ename = 'SCOTT';
```

- 이름이 SCOTT인 사원의 이름과 월급, 직업, 입사일, 부서 번호를 출력하는 SQL입니다.
- 숫자와는 다르게 문자를 검색할 때는 문자 양쪽에 싱글 쿼테이션 마크를 둘러 감싸 주어야 합니다.
- 싱글 쿼테이션 마크 안에 있는 것이 숫자가 아니라 문자라고 오라클에게 알려주는 것입니다.
  - 1~2행 : emp 테이블로부터 이름과 월급, 직업, 입사일, 부서 번호 컬럼을 선택합니다.
  - 3행 : 이름이 SCOTT인 사원의 데이터로만 행을 제한합니다.



## SQL WHERE절 배우기 ② ( 문자와 날짜 검색 )

- 다음의 쿼리는 81년 11월 17일에 입사한 사원의 이름과 입사일을 출력합니다.

```
1 SELECT ename, hiredate
2 FROM emp
3 WHERE hiredate = '81/11/17';
```

- 문자뿐만 아니라 날짜도 양쪽에 싱글 쿼테이션 마크를 감싸 주어야 합니다.
- 날짜 데이터 검색의 경우 현재 접속한 세션(session)의 날짜 형식에 맞춰 작성해 주어야 합니다.
- 81/11/17은 '연도/월/일'로 된 날짜 형식입니다.
- 우리나라와는 다르게 미국이나 영국의 날짜 형식은 '일/월/연도'입니다.
- 그럼 17/11/81로 날짜를 검색해야 합니다.
- 나라마다 날짜 형식이 다르기 때문에 날짜 검색을 하기 전에 현재 접속한 세션(session)의 날짜 형식을 확인하는 것이 필요합니다.

# SQL WHERE절 배우기 ② ( 문자와 날짜 검색 )

- 현재 접속한 세션(session)의 날짜 형식은 NLS\_SESSION\_PARAMETERS를 조회하여 확인합니다.

```
1 SELECT *
2 FROM NLS_SESSION_PARAMETERS
3 WHERE PARAMETER = 'NLS_DATE_FORMAT';
```

- RR/MM/DD에서 RR은 년도이고 MM은 달이며 DD는 일입니다.
- 날짜 형식에 대한 정의는 다음과 같습니다.

형식	정의	형식	정의
YYYY	연도 4자리	HH24	시간 (0~24)
YY 또는 RR	연도 2자리	MI	분(0~59)
MM	달의 2자리값	SS	초(0~59)
MON	달의 영문 약어	WW	연의 주
DD	숫자 형식의 일	IW	ISO 표준에 따른 연의 주
DAY	요일	W	월의 주
DY	요일 약어	YEAR	영어 철자로 표기된 년도
D	요일의 숫자	MONTH	영어 철자로 표기된 달

## SQL WHERE절 배우기 ② ( 문자와 날짜 검색 )

- 연도 2자리 형식인 RR과 YY는 서로 다릅니다.
- 만약 현재 세션의 날짜 형식을 YY/MM/DD로 변경하고 81년도 11월 17일에 입사한 사원을 조회한다면 조회가 되지 않을 것입니다.

```
1 ALTER SESSION SET NLS_DATE_FORMAT='YY/MM/DD';
```

```
1 SELECT ename, sal
2 FROM emp
3 WHERE hiredate='81/11/17';
```

- 선택된 레코드가 없다고 나오는 이유는 YY로 검색을 하게 되면 81년을 2081년으로 인식하고 검색하기 때문입니다.
- RR은 81년을 1981년도로 검색합니다.
- RR은 현재 세기를 기준으로 81년도를 이전 세기로 인식하고 1981년도로 인식합니다.
- 그러나 YY는 81년도를 현재 세기의 연도로 인식해서 2081년도로 인식을 합니다.

## SQL WHERE절 배우기 ② ( 문자와 날짜 검색 )

- ALTER SESSION SET 명령어는 현재 내가 접속한 세션의 파라미터를 변경하는 명령어입니다.
- 세션이란 데이터베이스 유저로 로그인해서 로그아웃할 때까지의 한 단위를 말합니다.
- SCOTT 유저로 접속해서 SCOTT 유저로 로그아웃할 때까지 하나의 단위가 세션입니다.
- 바로 이 세션의 파라미터인 NLS\_DATE\_FORMAT은 현재 세션에서만 유효한 파라미터입니다.
- SCOTT으로 접속한 상태라면 다시 로그아웃을 했다가 접속하면 NLS\_DATE\_FORMAT은 RR/MM/DD인 기본값으로 복귀되어 세팅됩니다.
- 다시 날짜 형식을 RR/MM/DD로 변경합니다.

```
1 ALTER SESSION SET NLS_DATE_FORMAT='RR/MM/DD';
```



# 산술 연산자 배우기(\*, /, +, -)

- 연봉이 36000 이상인 직원들의 이름과 연봉을 출력해 보겠습니다.

```
1 SELECT ename, sal*12 as 연봉
2 FROM emp
3 WHERE sal*12 >= 36000;
```

- sal\*12가 36000 이상인 직원들의 이름과 연봉을 출력하는 SQL입니다.
- sal\*12는 컬럼 별칭을 주어 연봉으로 출력했습니다.
- 산술 연산자에는 우선순위가 있어서 곱하기와 더하기가 같은 식에 있으면 곱하기부터 실행합니다.
- 만약 더하기부터 실행하고 싶다면 괄호를 사용하면 됩니다.

	곱하기부터 실행됨	더하기부터 실행됨
SQL	SELECT 300 + 200 * 2 FROM DUAL;	SELECT (300 + 200) * 2 FROM DUAL;
결과	700	10000



## 산술 연산자 배우기(\*, /, +, -)

- 다음의 쿼리는 부서 번호가 10번인 직원들의 이름, 월급, 커미션, 월급 + 커미션을 출력하는 SQL 입니다.

```
1 SELECT ename, sal, comm, sal + comm
2 FROM emp
3 WHERE deptno = 10;
```

- 커미션에 데이터가 없습니다.
- 이 상태를 NULL 값이라고 합니다.
- NULL 값은 데이터가 없는 상태, 즉 데이터가 할당되지 않은 상태 또는 알 수 없는 값이라고 합니다.
- 그래서 커미션이 NULL인 직원들은 월급+커미션의 결과도 NULL 값이 출력되었습니다.
- NULL이 알 수 없는 값이므로 5000 더하기 NULL은 알 수 없는 값인 NULL이어서 NULL로 출력되는 것입니다.
- 산술식의 컬럼값이 NULL인 경우 결과도 NULL입니다.
- 이 경우는 NULL이 아닌 숫자로 변경해 주어야 월급+커미션 값이 출력될 수 있습니다.



## 산술 연산자 배우기(\*, /, +, -)

- NVL 함수는 NULL 데이터를 처리하는 함수입니다.
- NVL(comm, 0)은 comm을 출력할 때 comm이 NULL이면 0으로 출력하는 함수입니다.
- NVL을 사용하지 않았을 때는 5000 + NULL로 연산되었습니다.
- NULL은 알 수 없는 값(UNKNOWN)입니다.
- 5000 더하기 알 수 없는 값은 알 수 없는 값이어서 NULL로 출력됩니다.
- NVL 함수를 이용해서 NULL을 0으로 변경하면 5000 + 0으로 수행되어 5000이 결과로 출력될 수 있게 됩니다.

	nvl 함수를 사용하지 않았을 때	nvl 함수를 사용했을 때
SQL	SELECT sal + comm FROM emp WHERE ename='KING';	SELECT sal + NVL(comm,0) FROM emp WHERE ename='KING';
결과	NULL이 출력됨	5000



# SQL 비교 연산자 배우기 ①

- 월급이 1200 이하인 직원들의 이름과 월급, 직업, 부서 번호를 출력해 보겠습니다.

```
1 SELECT ename, sal, job, deptno
2 FROM emp
3 WHERE sal <= 1200;
```

- 1~2행 : emp 테이블로부터 이름과 월급, 직업, 부서 번호 컬럼을 선택합니다.
- 3행 : 월급이 1200 이하인 직원들의 데이터로만 행을 제한합니다.

ENAME	SAL	JOB	DEPTNO
JAMES	950	CLERK	30
SMITH	800	CLERK	20
ADAMS	100	CLERK	20





## SQL 비교 연산자 배우기 ②

- 월급이 1000에서 3000 사이인 직원들의 이름과 월급을 출력해 보겠습니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE sal BETWEEN 1000 AND 3000;
```

- 위 SQL은 다음의 SQL과 동일합니다.
- 월급 1000과 3000이 포함됩니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE (sal >= 1000 AND sal <= 3000);
```

- BETWEEN AND 사용 시 주의사항은 "BETWEEN 하한값 AND 상한값"순으로 작성해야 검색이 됩니다.
- "BETWENN 상한값 AND 하한값"으로 작성하면 검색되지 않습니다.



## SQL 비교 연산자 배우기 ②

- 다음의 쿼리는 월급이 1000에서 3000 사이가 아닌 직원들의 이름과 월급을 조회합니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE sal NOT BETWEEN 1000 AND 3000;
```

- 위의 예제는 1000과 3000을 포함하지 않는 다음의 SQL과 동일합니다.
- 이퀄(=)이 붙지 않습니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE (sal < 1000 OR sal > 3000);
```

- BETWEEN .. AND 연산자를 사용하는 것이 훨씬 가독성이 있고 코드가 심플합니다.
- 다음의 예제는 1982년도에 입사한 직원들의 이름과 입사일을 조회하는 쿼리입니다.

```
1 SELECT ename, hiredate
2 FROM emp
3 WHERE hiredate BETWEEN '1982/01/01' AND '1982/12/31';
```



## SQL 비교 연산자 배우기 ③

- 이름의 첫 글자가 S로 시작하는 직원들의 이름과 월급을 출력해 보겠습니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE ename LIKE 'S%';
```

- %는 와일드 카드(Wild Card)라고 합니다.
- 와일드 카드는 이 자리에 어떠한 철자가 와도 상관없고 철자의 개수가 몇 개가 되든 관계 없다는 뜻입니다.
- 첫 철자가 S로 시작하면 검색 대상이 됩니다.
- WHERE ename='S%'는 첫 번째 철자가 S이고 두 번째 철자가 %인 데이터를 검색하겠다는 것입니다.
- %가 특수문자 퍼센트가 아니라 와일드 카드가 되려면 이퀄 연산자(=)가 아닌 LIKE 연산자를 사용해야 합니다.



## SQL 비교 연산자 배우기 ③

- 다음의 쿼리는 이름의 두 번째 철자가 M인 사원의 이름을 출력합니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE ename LIKE '_M%';
```

- LIKE와 같이 쓰이는 기호는 와일드 카드(%)와 언더바(\_)가 있습니다.
- 와일드 카드(%)는 와일드 카드가 있는 자리에 어떠한 철자가 와도 관계없고 그 철자의 개수가 몇 개이든 관계없다는 것인 반면, 언더바(\_)는 어떠한 철자가 와도 관계없으나 자리수는 한 자리여야 된다는 의미입니다.
- 다음의 쿼리는 이름의 끝 글자가 T로 끝나는 사원들의 이름을 출력합니다.
- 와일드 카드를 앞에 기술하고 알파벳 T를 맨 뒤에 작성합니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE ename LIKE '%T';
```



## SQL 비교 연산자 배우기 ③

- 다음의 쿼리는 이름에 A를 포함하고 있는 직원들의 이름을 출력합니다.
- 와일드카드를 알파벳 A를 양쪽으로 기술하게 되면 이름이 A가 포함된 직원들을 전부 검색하게 됩니다.

```
1 SELECT ename, sal
2   FROM emp
3  WHERE ename LIKE '%A%';
```



## SQL 비교 연산자 배우기 ④

- 커미션이 NULL인 직원들의 이름과 커미션을 출력해 보겠습니다.

```
1 SELECT ename, comm
2 FROM emp
3 WHERE comm is null;
```

- NULL 값은 데이터가 할당되지 않은 상태라고도 하고 알 수 없는 값이라고도 합니다.
- 알 수 없는 값이기 때문에 이퀄 연산자(=)로는 비교할 수 없습니다.
- NULL 값을 검색하기 위해서는 is null 연산자를 사용해야 합니다.
- NULL이 아닌 데이터를 검색할 때도 COMM != NULL을 사용하여 검색할 수 없습니다.
- NULL은 데이터가 없는 상태이고 알 수 없는 값이기 때문입니다.



## SQL 비교 연산자 배우기 ⑤

- 직업이 SALESMAN, ANALYST, MANAGER인 직원들의 이름, 월급, 직업을 출력해 보겠습니다.

```
1 SELECT ename, sal, job
2   FROM emp
3  WHERE job in ('SALESMAN', 'ANALYST', 'MANAGER');
```

- in 연산자를 이용하여 직업이 SALESMAN, ANALYST, MANAGER인 직원 한 번에 조회합니다.
- 이퀄 연산자(=)는 하나의 값만 조회할 수 있는 반면 IN 연산자는 여러 리스트의 값을 조회할 수 있습니다.
- 다음의 쿼리는 위 예제와 동일한 결과를 출력하는 SQL입니다.

```
1 SELECT ename, sal, job
2   FROM emp
3  WHERE (job = 'SALESMAN' or job = 'ANALYST' or job = 'MANAGER');
```



## SQL 비교 연산자 배우기 ⑤

- 다음의 쿼리는 직업이 SALESMAN, ANALYST, MANAGER가 아닌 직원들의 이름과 월급과 직업을 검색합니다.

```
1 SELECT ename, sal, job
2   FROM emp
3  WHERE job NOT in ('SALESMAN', 'ANALYST', 'MANAGER');
```

- 위의 쿼리는 다음의 쿼리와 동일한 결과를 반환합니다.

```
1 SELECT ename, sal, job
2   FROM emp
3  WHERE (job != 'SALESMAN' and job != 'ANALYST' and job != 'MANAGER');
```





## SQL 논리 연산자 배우기

- 직업이 SALESMAN이고 월급이 1200 이상인 직원들의 이름, 월급, 직업을 출력해 보겠습니다.

```
1 SELECT ename, sal, job
2 FROM emp
3 WHERE job = 'SALESMAN' AND sal >= 1200;
```

- 직업 SALESMAN이 검색되어 조건이 TRUE이고, 월급이 1200 이상인 데이터도 EMP 테이블에서 검색이 되는 맞는 조건입니다.
- WHERE TRUE 조건 AND TRUE 조건이어서 데이터가 잘 검색되어 출력되었습니다.
- 둘 중에 하나라도 조건이 FALSE이면 데이터는 반환되지 않습니다.

```
1 SELECT ename, sal, job
2 FROM emp
3 WHERE job='ABCDEFGF' AND sal >= 1200;
```



## SQL 논리 연산자 배우기

---

- TRUE AND TRUE는 결과가 TRUE입니다.
- TRUE AND FALSE는 결과가 FALSE입니다.
- AND는 둘 다 TRUE여야 TRUE가 반환됩니다.
- 반면 OR는 둘 중에 하나만 TRUE여도 TRUE를 반환합니다.
- TRUE AND NULL은 NULL이 반환됩니다.
- NULL은 알 수 없는 값이기 때문에 TRUE인지 FALSE인지 알 수 없습니다.
- 만약 NULL이 TRUE라면 TRUE AND TRUE는 TRUE가 반환되고 FALSE라면 TRUE AND FALSE는 FALSE가 반환되므로 TRUE인지 FALSE인지 알 수 없으므로 NULL이 반환됩니다.



# 대소문자 변환 함수 배우기

- 사원 테이블의 이름을 출력하는데 첫 번째 컬럼은 이름을 대문자로 출력하고 두 번째 컬럼은 이름을 소문자로 출력하고 세 번째 컬럼은 이름의 첫 번째 철자는 대문자로 하고 나머지는 소문자로 출력해 보겠습니다.

```
1 SELECT UPPER(ename), LOWER(ename), INITCAP(ename)
2 FROM emp;
```

- upper 함수는 대문자로 출력, lower 함수는 소문자로 출력, initcap 함수는 첫 번째 철자만 대문자로 출력 하고 나머지는 소문자로 출력하는 함수입니다.
- 함수(function)는 다양한 데이터 검색을 위해 필요한 기능입니다.
- 예를 들어, 사원 테이블의 월급 평균값은 얼마인지, 사원 테이블에서 평균 월급 이상을 받는 사원들은 누구인지 출력하려면 함수를 알아야 합니다.

함수의 종류		설명
단일행 함수	정의	하나의 행을 입력받아 하나의 행을 반환하는 함수
	종류	문자함수, 숫자함수, 날짜함수, 변환 함수, 일반함수
다중 행 함수	정의	여러 개의 행을 입력받아 하나의 행을 반환하는 함수
	종류	그룹 함수



## 대소문자 변환 함수 배우기

- upper 함수와 lower 함수는 테이블 내 특정 문자 데이터를 검색하고자 할 때 데이터가 대문자 인지 소문자로 저장되어 있는지 확실하지 않을 때 정확한 데이터 검색을 위해 필요합니다.
- 다음의 쿼리는 이름이 scott인 사원의 이름과 월급을 조회하는 쿼리입니다.

```
1 SELECT ename, sal, job
2     FROM emp
3     WHERE job = 'SALESMAN' AND sal >= 1200;
```

- LOWER(ename)을 사용했기 때문에 대문자였던 사원 이름이 모두 소문자로 변환되어 소문자 scott을 확실하게 검색할 수 있게 됩니다.
- 우리는 사원 테이블의 사원 이름이 모두 대문자로 되어 있다는 것을 이미 알고 있습니다.
- 그런데 사원 테이블이 1억건이 넘는 대용량 테이블이라고 가정해보고 사원 이름이 대문자로 되어 있는지 소문자로 되어 있는지 모른다고 하면 LOWER(ename)=scott' 또는 UPPER(ename)='SCOTT이라고 작성해야 데이터를 확실하게 반환할 수 있게 됩니다.



# 문자에서 특정 철자 추출하기

- 영어 단어 SMITH에서 SMI만 잘라내서 출력해 보겠습니다.

```
1 SELECT SUBSTR('SMITH',1,3)
2 FROM emp;
```

- SUBSTR 함수는 문자에서 특정 위치의 문자열을 추출합니다.
- SUBSTR('SMITH', 1,3)에서 1은 추출할 철자의 시작 위치 번호입니다.
- 3은 시작 위치로부터 몇 개의 철자를 추출할지를 정하는 숫자입니다.
- 1의 위치는 S입니다.
- S 철자부터 3개를 추출하여 SMI가 출력됩니다.

S	M	I	T	H
1	2	3	4	5



- SELECT SUBSTR('SMITH',2,2) 쿼리는 SMITH 문자에서 두 번째 철자인 M부터 두 개의 철자를 추출하여 MI를 반환합니다.
- SELECT SUBSTR('SMITH',-2,2) 쿼리는 SMITH 문자에서 -2 자리인 T부터 두 개의 철자를 추출하여 TH를 반환합니다.
- SELECT SUBSTR('SMITH',2) 쿼리는 SMITH 문자 M부터 끝까지 추출하여 MITH를 반환합니다.



## 문자열의 길이를 출력하기

- 이름을 출력하고 그 옆에 이름의 철자 개수를 출력해 보겠습니다.

```
1 SELECT ename, LENGTH(ename)
2 FROM emp;
```

- LENGTH 함수는 문자열의 길이를 출력하는 함수입니다. LENGTH(ename)는 이름 철자의 개수가 출력됩니다.
- 한글도 마찬가지로 문자의 길이가 출력됩니다.

```
1 SELECT LENGTH('가나다라마')
2 FROM DUAL;
```

- 다음의 쿼리는 숫자 15를 반환합니다.
- LENGTHB는 바이트의 길이를 반환합니다.
- 한글이 한 글자에 3바이트이므로 15를 반환합니다.

```
1 SELECT LENGTHB('가나다라마')
2 FROM DUAL;
```



## 문자에서 특정 철자의 위치 출력하기

- 사원 이름 SMITH에서 알파벳 철자 M이 몇 번째 자리에 있는지 출력해 보겠습니다.

```
1 SELECT INSTR('SMITH', 'M')
2 FROM DUAL;
```

- INSTR 함수는 문자에서 특정 철자의 위치를 출력하는 함수입니다.
- SMITH에서 M은 두 번째 철자이므로 2를 출력합니다.
- abcdefgh@naver.com 이메일에서 naver.com만 추출하고 싶다면 INSTR과 SUBSTR을 이용하면 추출할 수 있습니다.
- 먼저 @의 위치를 INSTR로 추출합니다.

```
1 SELECT INSTR('abcdefg@naver.com', '@')
2 FROM DUAL;
```



# 문자에서 특정 철자의 위치 출력하기

- SUBSTR에 INSTR 함수로 추출한 @의 자리 숫자 다음 철자를 시작 숫자로 사용하여 다음과 같이 쿼리로 NAVER.COM을 추출합니다.

a	b	c	d	e	f	g	@	n	a	v	e	r	.	c	o	m
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

```
1 SELECT SUBSTR('abcdedfgh@naver.com', INSTR('abcdedfgh@naver.com', '@') + 1)
2 FROM DUAL;
```

- 이번에는 오른쪽에 .com을 잘라내고 naver만 출력해보겠습니다.

n	a	v	e	r	.	c	o	m
9	10	11	12	13	14	15	16	17

```
1 SELECT RTRIM(SUBSTR('abcdedfgh@naver.com', INSTR('abcdedfgh@naver.com', '@') + 1), '.com')
2 FROM DUAL;
```





## 특정 철자를 다른 철자로 변경하기

- 이름과 월급을 출력하는데, 월급을 출력할 때 숫자 0을 \*(별표)로 출력해 보겠습니다.

```
1 SELECT SUBSTR('abcdedfgh@naver.com', INSTR('abcdedfgh@naver.com', '@') + 1)
2 FROM DUAL;
```

- REPLACE 함수는 특정 철자를 다른 철자로 변경하는 문자 함수입니다.
- 이름과 월급을 출력하는데 월급의 숫자 0을 \*로 변경해서 출력합니다.
- 다음의 쿼리는 월급의 숫자 0~3까지를 \*로 출력합니다.

```
1 SELECT RTRIM(SUBSTR('abcdedfgh@naver.com', INSTR('abcdedfgh@naver.com', '@') + 1), '.com')
2 FROM DUAL;
```

- REGEXP\_REPLACE 함수는 정규식(Regular Expression) 함수입니다.
- 정규식 함수는 일반함수보다 더 복잡한 형태의 검색패턴으로 데이터를 조회할 수 있게 해주는 함수입니다.



# 특정 철자를 다른 철자로 변경하기

- 예제를 위해 다음의 테이블 생성 스크립트를 실행합니다.

```
1 CREATE TABLE TEST_ENAME(ENAME VARCHAR2(10));
2
3 INSERT INTO TEST_ENAME VALUES('김인호');
4 INSERT INTO TEST_ENAME VALUES('안상수');
5 INSERT INTO TEST_ENAME VALUES('최영희');
6 COMMIT;
```

- 다음의 쿼리는 이름의 두 번째 자리의 한글을 \*로 출력합니다.

```
1 SELECT REPLACE(ename,SUBSTR(ename, 2, 1), '*') as "전광판 이름"
2 FROM test_ename;
```



# 특정 철자를 N개 만큼 채우기

- 이름과 월급을 출력하는데 월급 컬럼의 자릿수를 10자리로 하고, 월급을 출력하고 남은 나머지 자리에 별표(\*)를 채워서 출력해 보겠습니다.

```
1 SELECT ename, LPAD(sal, 10, '*') as salary1, RPAD(sal, 10, '*') as salary2
2 FROM test_ename;
```

- KING의 월급을 보면 5000은 4자리여서 나머지 6자리에 별표(\*)를 채워서 출력하고 있습니다.
- LPAD의 뜻이 왼쪽(Left)으로 채워 넣다(PAD)여서 왼쪽으로 별표(\*)를 채우고 있습니다.

*	*	*	*	*	*	5	0	0	0
---	---	---	---	---	---	---	---	---	---

- RPAD(Sal,10,\*)는 오른쪽으로 별표(\*)를 채워서 출력합니다.

5	0	0	0	*	*	*	*	*	*
---	---	---	---	---	---	---	---	---	---



## 특정 철자를 N개 만큼 채우기

- LPAD나 RPAD를 이용하면 SQL로 데이터를 시각화하기 유용합니다.
- 다음의 쿼리는 이름과 월급을 출력하는데 월급 100을 네모(■) 하나로 출력하는 예제입니다.

```
1 SELECT ename, sal, lpad('■', round(sal/100), '■') as bar_chart
2 FROM emp;
```

- 막대 그래프처럼 사원 테이블의 월급을 시각화하여 출력하였습니다.
- lpad('■', round(sal/100), '■')에서 round(sal/100)이 16이라면 전체 16자리를 확보합니다.
- 16자리 중 ■를 하나 출력하고 나머지 15자리에 ■를 채워서 출력하여 ■가 16개 출력됩니다.



## 특정 철자 잘라내기

- 첫 번째 컬럼은 영어 단어 smith 철자를 출력하고, 두 번째 컬럼은 영어 단어 smith에서 s를 잘라서 출력하고, 세 번째 컬럼은 영어 단어 smith에서 o를 잘라서 출력하고, 네 번째 컬럼은 영어 단어 smiths의 양쪽에 s를 잘라서 출력해 보겠습니다.

```
1 SELECT 'smith', LTRIM('smith', 's'), RTRIM('smith', 'h'), TRIM('s' from 'smith')
2 FROM dual;
```

- 1~2행 : LTRIM('smith', 's')는 smith를 출력하는데 왼쪽 철자인 s를 잘라서 출력합니다.  
RTRIM('smith', 'h')는 smith를 출력하는데 오른쪽 철자인 h를 잘라서 출력합니다. TRIM('s' from 'smiths')는 smiths를 출력하는데 양쪽의 s를 잘라서 출력합니다.
- INSERT 문장으로 사원(EMP) 테이블에 사원 데이터를 입력합니다.
- JACK 사원 데이터를 입력할 때 JACK 오른쪽에 공백을 하나 넣어서 입력합니다.

```
1 INSERT INTO emp(empno, ename, sal, job, deptno) values(8291, 'JACK ', 3000, 'SALESMAN', 30);
2 Commit;
```



## 특정 철자 잘라내기

- 다음의 쿼리는 이름이 JACK인 사원의 이름과 월급을 조회하는 쿼리입니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE ename = 'JACK';
```

- 위의 쿼리의 결과로 아무것도 출력되지 않습니다.
- 아무것도 출력되지 않는 이유는 JACK 데이터를 입력할 때 오른쪽에 공백을 추가해서 입력했는데 WHERE절의 ename='JACK'으로 비교할 때 공백없이 비교했기 때문입니다.
- 우리는 공백을 하나만 입력하고 저장했습니다.
- 그래서 공백이 하나만 있다는 것을 알고 있지만, 공백을 하나만 입력했는지 여러 개를 입력했는지 모른다고 가정하면 JACK 데이터를 검색하기가 어렵습니다.
- 이럴 때는 다음과 같이 쿼리를 작성합니다.

```
1 SELECT ename, sal
2 FROM emp
3 WHERE RTRIM(ename) = 'JACK';
```



## SQL 특정 철자 잘라내기

- `RTRIM(ename)='JACK'`으로 JACK 데이터를 검색할 때 사원 이름(ename) 오른쪽에 있는 공백을 제거하고 검색합니다.
- 다음 실습을 위하여 입력한 데이터를 삭제합니다.

```
1 DELETE FROM emp WHERE TRIM(ename) = 'JACK';
```



# 반올림해서 출력하기

- 876.567 숫자를 출력하는데 소수점 두 번째 자리인 6에서 반올림해서 출력해 보겠습니다.

```
1 SELECT '876.567' as 숫자, ROUND(876.567, 1)
2 FROM dual;
```

- 1행 : ROUND(876.567,1)는 876.567을 출력할 때 소수점 이후 두 번째 자리에서 반올림합니다.

숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3


숫자	8	7	6	.	6
자리	-3	-2	-1	0	1





# SQL 반올림해서 출력하기

- `ROUND(876.567,2)`는 876.567을 출력할 때 소수점 이후 세 번째 자리에서 반올림합니다.




숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3

숫자	8	7	6	.	5	7	
자리	-3	-2	-1	0	1	2	

- `ROUND(876.567,-1)`는 876.567을 출력할 때 소수점 이전 일의 자리에서 바로 반올림합니다.



숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3


  

숫자	8	8	0				
자리	-3	-2	-1				



# SQL 반올림해서 출력하기

- ROUND(876.567,-2)는 876.567을 출력할 때 소수점 이전 십의 자리에서 바로 반올림합니다.

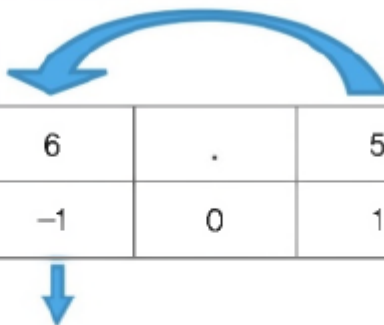


숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3

숫자	9	0	0
자리	-3	-2	-1

- ROUND(876.567,0)과 ROUND(876.567)의 결과는 동일합니다.
- 0의 자리는 소수점 자리고 0의 자리를 기준으로 두고 소수점 이후 첫 번째 자리에서 반올림합니다.



숫자	9	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3

숫자	8	7	7
자리	-3	-2	-1



# 숫자를 버리고 출력하기

- 876.567 숫자를 출력하는데 소수점 두 번째 자리인 6과 그 이후의 숫자들을 모두 버리고 출력해 보겠습니다.

```
1 SELECT '876.567' as 숫자, TRUNC(876.567, 1)
2 FROM dual;
```

- TURNC(876.567, 1)는 876.567을 출력할 때 소수점 이후 첫 번째 자리 이후 두 번째 자리부터 버리고 출력합니다.

숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3



숫자	8	7	6	.	5
자리	-3	-2	-1	0	1



## SQL 숫자를 버리고 출력하기

- `TRUNC(876.567,2)`는 876.567을 출력할 때 소수점 이후 두 번째 자리 이후 세 번째 자리부터 버리고 출력합니다.

숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3



숫자	8	7	6	.	5	6	
자리	-3	-2	-1	0	1	2	

- `TRUNC(876.567,-1)`는 876.567을 출력할 때 소수점 이전 일의 자리부터 바로 버리고 출력합니다.

숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3



숫자	8	7	0				
자리	-3	-2	-1				



## 숫자를 버리고 출력하기

- `TRUNC(876.567,-2)`는 876.567을 출력할 때 소수점 이전 십의 자리에서 바로 버리고 출력합니다.

숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3



숫자	8	0	0
자리	-3	-2	-1

- `TRUNC(876.567,0)`과 `TRUNC(876.567)`의 결과는 동일합니다.
- 0의 자리는 소수점 자리고 0의 자리를 기준으로 두고 소수점 이후를 전부 버리고 출력합니다.

숫자	8	7	6	.	5	6	7
자리	-3	-2	-1	0	1	2	3



숫자	8	7	6
자리	-3	-2	-1



## 나눈 나머지 값 출력하기

- 숫자 10을 3으로 나눈 나머지 값이 어떻게 되는지 출력해 보겠습니다.

```
1 SELECT MOD(10, 3)
2 FROM dual;
```

- 1행 : 10을 3으로 나눈 나머지 값인 1이 출력됩니다.

- 다음의 쿼리는 사원 번호와 사원 번호가 홀수이면 1, 짝수이면 0을 출력하는 쿼리입니다.

```
1 SELECT empno, MOD(empno, 2)
2 FROM emp;
```

- 다음의 쿼리는 사원 번호가 짝수인 사원들의 사원 번호와 이름을 출력하는 쿼리입니다.

```
1 SELECT empno, ename
2 FROM emp
3 WHERE MOD(empno, 2) = 0;
```



## SQL 나눈 나머지 값 출력하기

- 다음의 쿼리는 10을 3으로 나눈 몫을 출력하는 쿼리입니다.

```
1 SELECT FLOOR(10/3)
2 FROM dual;
```

- 1행 : 10을 3으로 나눈 몫은 3.33333333으로 출력됩니다. FLOOR는 3과 4 사이에서의 제일 바닥에 해당하는 값인 3을 출력합니다.



## SQL 날짜 간 개월 수 출력하기

- 이름을 출력하고 입사한 날짜부터 오늘까지 총 몇 달을 근무했는지 출력해 보겠습니다.

```
1 SELECT ename, MONTHS_BETWEEN(SYSDATE, hiredate)
2 FROM dual;
```

- 1행 : SYSDATE는 오늘 날짜를 확인하는 함수로 2019년 5월 30일 기준으로 출력된 결과입니다.  
MONTHS\_BETWEEN 함수는 날짜를 다루는 함수입니다. 날짜 값을 입력받아 숫자 값을 출력합니다.  
MONTHS\_BETWEEN 함수에 날짜 값을 입력할 때는 MONTHS\_BETWEEN(최신날짜, 예전 날짜)로 입력해야 합니다.
- MONTHS\_BETWEEN** 함수를 이용하지 않고 날짜만 가지고 연산을 해야 한다면 다음과 같이 날짜와 산술 연산만을 이용해서 산술식을 작성해야 합니다.
- 다음의 쿼리는 2018년 10월 1일에서 2019년 6월 1일 사이의 총 일수를 출력합니다.

```
1 SELECT TO_DATE('2019-06-01', 'RRRR-MM-DD') - TO_DATE('2018-10-01', 'RRRR-MM-DD')
2 FROM dual;
```





## SQL 날짜 간 개월 수 출력하기

- TO\_DATE 함수는 '2019-06-01'에서 2019년의 연도(RRRR), 06은 달(MM), 01은 일(DD)이라고 명시해줍니다.
- 다음의 쿼리는 2018년 10월 1일에서 2019년 6월 1일 사이의 총 주(week)수를 출력합니다.

```
1 SELECT ROUND((TO_DATE('2019-06-01', 'RRRR-MM-DD') - TO_DATE('2018-10-01', 'RRRR-MM-DD'))  
  / 7) as "총 주수"  
2 FROM dual;
```

- 35주의 결과가 출력됩니다.
- 그럼 2018년 10월 1일부터 2019년 6월 1일까지의 총 개월 수를 출력하려면 어떻게 해야 할까요?
- 주(WEEK)는 7로 나누었지만 달(MONTH)은 30으로 나누어야 할지 31로 나누어야 할지 정하기 어렵습니다.
- 이럴 때 MONTHS\_BETWEEN 함수를 사용합니다.
- MONTHS\_BETWEEN 함수는 날짜와 날짜 사이의 개월 수를 정확하게 계산해 줍니다.



## SQL 개월 수 더한 날짜 출력하기

- 2019년 5월 1일부터 100달 뒤의 날짜는 어떻게 되는지 출력해 보겠습니다.

```
1 SELECT ADD_MONTHS(TO_DATE('2019-05-01', 'RRRR-MM-DD'), 100)
2 FROM dual;
```

- 1행 : 2019년 5월 1 일에서 100달을 더하기 위해서 ADD\_MONTHS를 사용했습니다.

- 다음의 예제는 2019년 5월 1 일부터 100일 후에 돌아오는 날짜를 출력하는 쿼리문입니다.

```
1 SELECT TO_DATE('2019-05-01', 'RRRR-MM-DD') + 100
2 FROM dual;
```

- 그런데 일(DD)이 아니라 달이 되면 달의 기준을 30일로 해야 할지 31로 해야 할지 정하기 어렵습니다.
- 그래서 ADD\_MONTHS 함수를 이용하거나 다음과 같이 쿼리를 작성해도 됩니다.

```
1 SELECT TO_DATE('2019-05-01', 'RRRR-MM-DD') + interval '100' month
2 FROM dual;
```



# SQL 개월 수 더한 날짜 출력하기

- interval 함수를 활용하면 좀더 섬세하게 날짜 산술 연산을 구현할 수 있습니다.
- 예를 들면 2019년 5월 1일부터 1년 3개월 후의 날짜를 출력하는 쿼리는 다음과 같습니다.

```
1 SELECT TO_DATE('2019-05-01', 'RRRR-MM-DD') + interval '1-3' year(1) to month
2 FROM dual;
```

- 연도(year), 달(month), 일(day), 시간(hour), 분(minute), 초(second)까지 다양하게 지정할 수 있습니다.

INTERVAL 표현식	설명
INTERVAL '4' YEAR	An interval of 4 years 0 months
INTERVAL '123' YEAR(3)	An interval of 123 years 0 months
INTERVAL '6' MONTHS	An interval of 6 months
INTERVAL '600' MONTHS(3)	An interval of 600 months
INTERVAL '400' DAY(3)	400 days



## SQL 개월 수 더한 날짜 출력하기

- 연도(year), 달(month), 일(day), 시간(hour), 분(minute), 초(second)까지 다양하게 지정할 수 있습니다.

INTERVAL 표현식	설명
INTERVAL '10' HOUR	10 hours
INTERVAL '10' MINUTE	10 minutes
INTERVAL '4' DAY	4 days
INTERVAL '25' HOUR	25 hours
INTERVAL '40' MINUTE	40 minutes
INTERVAL '120' HOUR(3)	120 hours

- INTERVAL 사용 시 연도가 한 자리 인 경우는 YEAR를 사용하고 연도가 3자리 인 경우는 YEAR(3)을 사용합니다.



## SQL 개월 수 더한 날짜 출력하기

- 다음의 쿼리는 2019년 5월 01일에서 3년 후의 날짜를 반환합니다.

```
1 SELECT TO_DATE('2019-05-01', 'RRRR-MM-DD') + interval '3' year
2 FROM dual;
```

- TO\_YMINTERVAL 함수를 이용하면 2019년 5월 1일부터 3년 5개월 후의 날짜를 출력할 수 있습니다.

```
1 SELECT TO_DATE('2019-05-01', 'RRRR-MM-DD') + TO_YMINTERVAL('03-05') as 날짜
2 FROM dual;
```



# 특정 날짜 뒤에 오는 요일 날짜 출력하기

- 2019년 5월 22일부터 바로 돌아올 월요일의 날짜가 어떻게 되는지 출력해 보겠습니다.

```
1 SELECT '2019/05/22' as 날짜, NEXT_DAY('2019/05/22', '월요일')
2 FROM dual;
```

- 5월 22일 이후에 바로 돌아오는 월요일은 5월 27일입니다.

2019년 5월 달력

일	월	화	수	목	금	토
19	20	21	22	23	24	25
26	27	28	29	30	31	

- 다음의 쿼리는 오늘 날짜를 출력하는 쿼리입니다.

```
1 SELECT SYSDATE as "오늘 날짜"
2 FROM dual;
```



## 특정 날짜 뒤에 오는 요일 날짜 출력하기

- 오늘부터 앞으로 돌아올 화요일의 날짜를 출력하는 쿼리는 다음과 같습니다.

```
1 SELECT NEXT_DAY(SYSDATE, '화요일') as "다음 날짜"
2 FROM dual;
```

- 다음의 쿼리는 2019년 5월 22일부터 100달 뒤에 돌아오는 화요일의 날짜를 출력합니다.

```
1 SELECT NEXT_DAY(ADD_MONTHS('2019/05/22', 100), '화요일') as "다음 날짜"
2 FROM dual;
```

- 위와 같이 함수를 중첩해서 사용할 수 있습니다.
- 다음의 예제는 오늘부터 100달 뒤에 돌아오는 월요일의 날짜를 출력하는 쿼리입니다.

```
1 SELECT NEXT_DAY(ADD_MONTHS(sysdate, 100), '월요일') as "다음 날짜"
2 FROM dual;
```

# SQL 특정 날짜가 있는 달의 마지막 날짜 출력하기

- 2019년 5월 22일 해당 달의 마지막 날짜가 어떻게 되는지 출력해 보겠습니다.

```
1 SELECT '2019/05/22' as 날짜, LAST_DAY('2019/05/22') as "마지막 날짜"
2 FROM dual;
```

- LAST\_DAY 함수에 2019년 5월 22일 날짜를 지정하면 2019년 5월의 말일 날짜인 2019/05/31 일을 출력합니다.

2019년 5월 달력

일	월	화	수	목	금	토
19	20	21	22	23	24	25
26	27	28	29	30	31	

- 다음의 쿼리는 오늘부터 이번 달 말일까지 총 며칠 남았는지 출력하는 쿼리입니다.

```
1 SELECT LAST_DAY(sysdate) - sysdate as "남은 날짜"
2 FROM dual;
```



## 특정 날짜가 있는 달의 마지막 날짜 출력하기

- 만약 오늘 날짜가 2019년 5월 22일이라면 2019년 5월 31 일에서 2019년 5월 22일을 뺀 숫자인 9가 출력됩니다.
- 수행하는 오늘 날짜가 다르므로 결과 값이 다를 수 있습니다.
- 다음은 이름이 KING인 사원의 이름, 입사일, 입사한 달의 마지막 날짜를 출력하는 쿼리입니다.

```
1  SELECT ename, hiredate, LAST_DAY(hiredate)
2      FROM emp
3      WHERE ename = 'KING';
```

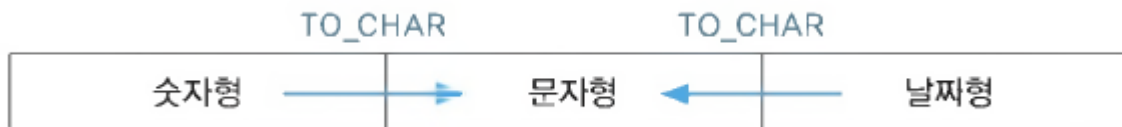


## 문자형으로 데이터 유형 변환하기

- 이름이 SCOTT인 사원의 이름과 입사한 요일을 출력하고 SCOTT의 월급에 천 단위를 구분할 수 있는 콤마(,)를 붙여 출력해 보겠습니다.

```
1 SELECT ename, TO_CHAR(hiredate, 'DAY') as 요일, TO_CHAR(sal, '999,999') as 월급
2 FROM emp
3 WHERE ename = 'SCOTT';
```

- 1행 : TO\_CHAR(hiredate, 'DAY')는 입사일을 요일로 출력합니다.
  - TO\_CHAR(sal, '999,999')는 월급을 출력할 때 천 단위를 표시하여 출력합니다.
- 숫자형 데이터 유형을 문자형으로 변환하거나 날짜형 데이터 유형을 문자형으로 변환할 때 TO\_CHAR 함수를 사용합니다.





# 문자형으로 데이터 유형 변환하기

- 날짜를 문자로 변환해서 출력하면 날짜에서 년, 월, 일, 요일 등을 추출해서 출력할 수 있습니다.

```

1  SELECT hiredate, TO_CHAR(hiredate, 'RRRR') as 연도, TO_CHAR(hiredate, 'MM') as 달,
   TO_CHAR(hiredate, 'DD') as 일, TO_CHAR(hiredate, 'DAY') as 요일
2  FROM emp
3  WHERE ename = 'KING';
  
```

- 날짜를 문자로 출력할 때 사용할 수 있는 날짜 포맷은 다음과 같습니다.

연도	RRRR, YYYY, RR, YY	주	WW, IW, W
월	MM, MON	시간	HH, HH24
일	DD	분	MI
요일	DAY, DY	초	SS

- 다음은 1981년도에 입사한 사원의 이름과 입사일을 출력하는 쿼리입니다.

```

1  SELECT ename, hiredate
2  FROM emp
3  WHERE TO_CHAR(hiredate, 'RRRR') = '1981';
  
```



## 문자형으로 데이터 유형 변환하기

- 날짜 컬럼에서 연도/월/일/시간/분초를 추출하기 위해 EXTRACT 함수를 사용해도 됩니다.

```
1 SELECT ename as 이름, EXTRACT(year FROM hiredate) as 연도, EXTRACT(month FROM hiredate) as 달, EXTRACT(day FROM hiredate) as 요일
2 FROM emp;
```

- 다음의 쿼리는 이름과 월급을 출력하는데, 월급을 출력할 때 천 단위를 표시해서 출력합니다.

```
1 SELECT ename as 이름, TO_CHAR(sal, '999,999') as 월급
2 FROM emp;
```

- 숫자 9는 자릿수이고 이 자리에 0~9까지 어떠한 숫자가 와도 관계없다는 뜻입니다.
- 쉼표(,)는 1천 단위를 나타내는 표시입니다.



## SQL 문자형으로 데이터 유형 변환하기

- 다음의 쿼리는 천 단위와 백만 단위를 표시하는 예제입니다.

```
1 SELECT ename as 이름, TO_CHAR(sal*200, '999,999') as 월급
2 FROM emp;
```

- 알파벳 L을 사용하면 화폐 단위 W(원화)를 붙여 출력할 수 있습니다.

```
1 SELECT ename as 이름, TO_CHAR(sal*200, 'L999,999,999') as 월급
2 FROM emp;
```



- 81년 11월 17일에 입사한 사원의 이름과 입사일을 출력해 보겠습니다.

```
1 SELECT ename, hiredate
2 FROM emp
3 WHERE hiredate = TO_DATE('81/11/17', 'RR/MM/DD');
```

- 3행 : TO\_DATE 함수를 이용하여 81 은 연도, 11은 달, 17은 일이라고 지정합니다.
- 날짜 데이터를 검색할 때는 접속한 세션의 날짜 형식을 확인해야 에러(Error) 없이 검색할 수 있습니다.
- 현재 접속한 세션의 날짜 형식을 확인하는 쿼리는 다음과 같습니다.

```
1 SELECT *
2 FROM NLS_SESSION_PARAMETERS
3 WHERE parameter = 'NLS_DATE_FORMAT';
```

- 결과가 RR/MM/DD라고 한다면 날짜를 검색할 때 다음과 같이 해야 합니다.

```
1 SELECT ename, hiredate
2 FROM NLS_SESSION_PARAMETERS
3 WHERE hiredate = '81/11/17';
```



- 날짜 형식이 DD/MM/RR이라면 다음과 같이 검색해야 합니다.

```
1 SELECT ename, hiredate
2 FROM emp
3 WHERE hiredate = '17/11/81';
```

- 여기서 17은 연도가 아니라 일(DD)입니다.
- 현재 내가 접속한 세션의 날짜 형식을 DD/MM/RR로 변경해보고 검색을 해봅니다.

```
1 ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/RR';
2
3 SELECT ename, hiredate
   FROM emp
  WHERE hiredate = '17/11/81';
```

- 현재 내가 접속한 세션의 날짜 포맷을 DD/MM/RR 형식으로 변경하는 명령어입니다.
- 세션이란 지금 SCOTT으로 오라클에 접속한 창을 의미합니다.



## 날짜형으로 데이터 유형 변환하기

- ALTER SESSION 명령어로 NLS\_DATE\_FORMAT 파라미터 설정을 변경했습니다.
- 이 ALTER SESSION 명령어로 변경한 파라미터 설정은 지금 접속한 세션에서만 유효합니다.
- SCOTT 창에서 로그아웃하고 다시 새로 접속하게 되면 ALTER SESSION으로 설정한 파라미터 설정 값은 사라지게 됩니다.
- 그런데 나라마다 또는 접속하는 세션마다 날짜 형식이 다를 수 있으므로 일관되게 날짜를 검색할 수 있는 방법이 필요합니다.
- 그래서 다음과 같이 TO\_DATE를 이용해서 81은 연도(RR)이고 11은 달(MM)이고 17은 일(DD)이라고 명시해 검색하면 접속한 세션의 날짜 형식과 관계없이 검색할 수 있습니다.

```
1 SELECT ename, hiredate
2     FROM emp
3     WHERE hiredate = TO_DATE('81/11/17', 'RR/MM/DD');
```

- 다시 날짜 포맷을 RR/MM/DD 형식으로 변환합니다.

```
1 ALTER SESSION SET NLS_DATE_FORMAT = 'RR/MM/DD';
```





# SQL 암시적 형 변환 이해하기

```
1 SELECT ename, sal
2 FROM emp
3 WHERE sal = '3000';
```

- 3행 : sal은 숫자형 데이터 컬럼인데 "3000"을 문자형으로 비교하고 있습니다.
- 그런데 '숫자형=문자형' 비교하여 데이터 검색이 안 될 것 같은데 결과가 나옵니다.
- SAL=3000'으로 잘못 비교해서 실행했지만 오라클이 알아서 '숫자형=숫자형'으로 암시적으로 형 변환을 하기 때문에 에러가 발생하지 않고 검색됩니다.
- 오라클은 문자형과 숫자형 두 개를 비교할 때는 문자형을 숫자형으로 변환합니다.
- 다음의 스크립트는 SAL을 일부러 문자형으로 만들어 생성하는 스크립트입니다.

```
1 CREATE TABLE EMP32 (ename VARCHAR2(10), sal VARCHAR2(10));
2
3 INSERT INTO emp32 VALUES('scott', '3000');
4 INSERT INTO emp32 VALUES('smith', '1200');
```



# SQL 암시적 형 변환 이해하기

- EMP32 테이블의 SAL 데이터는 문자형으로 만들어 저장했습니다.
- 그러면 다음과 같이 WHERE절에서 SAL = '3000'으로 쿼리를 작성하고 실행해야 됩니다.

```
1 SELECT ename, sal
2     FROM emp32
3     WHERE sal = '3000';
```

- 그런데 다음과 같이 3000을 숫자형으로 비교하면 검색되는지 확인해 봅니다.

```
1 SELECT ename, sal
2     FROM emp32
3     WHERE sal = 3000;
```

- 검색이 잘 됩니다. 검색이 잘되는 이유는 오라클이 내부적으로 숫자형=숫자형으로 비교해서 데이터를 검색했기 때문입니다.
- 오라클은 사용자가 수행한 SQL과는 달리 다음과 같이 쿼리를 변환해서 실행합니다.

```
1 SELECT ename, sal
2     FROM emp32
3     WHERE TO_NUMBER(sal) = 3000;
```



## SQL 암시적 형 변환 이해하기

- 오라클이 SAL을 TO\_NUMBER(SAL)로 변환하였습니다.
- 오라클이 내부적으로 실행하는 SQL을 확인하려면 다음과 같이 수행합니다.
- SET AUTOT ON 명령어는 SQL을 실행할 때 출력되는 결과와 SQL을 실행하는 실행 계획을 한 번에 보여 달라는 SQLPLUS 명령어입니다.
- 실행 계획이란, 오라클이 SQL을 실행할 때 어떠한 방법으로 데이터를 검색하겠다는 계획서입니다.
- 실행 계획 아래쪽에 내부적으로 실행한 SQL이 나옵니다.

1	SET AUTOT ON
2	
3	SELECT ename, sal
4	FROM emp32
5	WHERE sal = 3000;

- 위와 같이 오라클이 암시적으로 문자형을 숫자형으로 형 변환하였습니다.
- 위의 결과는 결과 창의 스크립트 출력 탭에서 확인할 수 있습니다.



## SQL NULL 값 대신 다른 데이터 출력하기

- 이름과 커미션을 출력하는데, 커미션이 NULL인 직원들은 0으로 출력해 보겠습니다.

```
1 SELECT ename, comm, NVL(comm, 0)
2 FROM emp;
```

- 1행 : 커미션이 NULL이 아닌 직원들은 자신의 커미션이 출력되고 커미션이 NULL인 직원들은 NULL 대신 0이 출력됩니다.
- 실제로 데이터가 0으로 변경되는 것은 아니고 출력되는 쿼리에서만 0으로 출력되어 보이는 것입니다.
- 다음은 이름, 월급, 커미션, 월급 + 커미션을 출력하는 쿼리입니다.

```
1 SELECT ename, sal, comm, sal + comm
2 FROM emp
3 WHERE job IN('SALESMAN', 'ANALYST');
```

- NULL은 알 수 없는 값이므로 커미션이 NULL인 직원들은 월급+커미션도 NULL로 출력되었습니다.



## SQL NULL 값 대신 다른 데이터 출력하기

- 다음의 쿼리는 커미션 NULL을 0으로 치환하여 월급+커미션을 출력하는 쿼리입니다.

```
1 SELECT ename, sal, comm, NVL(comm, 0), sal+NVL(comm, 0)
2 FROM emp
3 WHERE job IN('SALESMAN', 'ANALYST');
```

- 커미션 NULL이 0으로 치환되어 FORD와 SCOTT의 커미션이 NULL 대신에 0으로 출력되었습니다.
- 그리고  $3000 + 0$ 이 연산되어 3000으로 출력되었습니다.
- 다음의 쿼리는 NVL2 함수를 이용하여 커미션이 NULL이 아닌 직원들은  $sal+comm$ 을 출력하고, NULL인 직원들은  $sal$ 을 출력하는 예제입니다.

```
1 SELECT ename, sal, comm, NVL2(comm, sal+comm, sal)
2 FROM emp
3 WHERE job IN('SALESMAN', 'ANALYST');
```