[3주차] 3. SQL 연산 & 4. 함수

제출일 : 2022년 11월 18일 금요일

작성자 : 박민영

---

1. 집합 연산자

1.1 설명

집합 연산자

- 두 개의 질의 결과에 대한 연산자
- 피연산자로 항상 SELECT 문의 결과를 받는다.
-
- 집합 연산자는 연산 결과 0개 이상의 로우로 구성된 하나의 테이블을 반환한다.

연산자	연산 결과
UNION	- 두 질의의 결과를 하나로 합친다. (중복 허용 X) - 질의의 결과에 중복되어 포함된 로우에 대해서는 연산 결과에 한 번만 포함한다. - 중복을 제거한 결과의 합을 검색
UNION ALL	- 두 질의의 결과를 하나로 합친다. (중복 허용 O) - 중복을 포함한 결과의 합을 검색
INTERSECTION	- 두 질의의 결과에 공통적으로 포함된 로우를 추출한다. (중복 허용 X) - 양쪽 모두에서 포함된 행을 검색
MINUS	- 앞 질의의 결과로부터 뒤 질의의 결과에 포함된 모든 로우를 제거한다. (중복 허용 X) - 첫 번째 검색 결과에서 두 번째 검색 결과를 제외한 나머지를 검색

1.2 시나리오 수행

시나리오 내용
1. 두 개의 테이블 조회
2. UNION, UNION ALL을 이용한 합집합
3. INTERSECT를 이용한 교집합
4. MINUS를 이용한 차집합

1.3 시나리오 수행 결과

1. 두 개의 테이블 조회

```
-- 1. 두 개의 테이블 조회
SELECT * FROM izone;
SELECT * FROM ive;
```

```
SQL> -- 1. 두 개의 테이블 조회
SQL> SELECT * FROM izone;

NAME       DEBUT_ORDER BIRTH_YEAR
---------- ----------- ----------
eunbi                7       1995
sakura               2       1998
hyewon               8       1999
yena                 4       1999
chaeyeon            12       2000
chaewon             10       2000
minju               11       2001
nako                 6       2001
hitomi               9       2001
yuri                 3       2001
yujin                5       2003
wonyoung             1       2004

12 rows selected.
```

```
SQL> SELECT * FROM ive;

NAME       BIRTH_YEAR
---------- ----------
yujin            2003
gaeul            2002
rei              2004
wonyoung         2004
liz              2004
leeseo           2007

6 rows selected.
```

2. UNION, UNION ALL을 이용한 합집합

```
-- 2. UNION, UNION ALL을 이용한 합집합
---- 2.1 UNION (중복 허용 x)
SELECT name, birth_year FROM ive UNION SELECT name, birth_year FROM izone ORDER BY birth_year;
---- 2.2 UNION ALL (중복 허용 O)
SELECT name, birth_year FROM ive UNION ALL SELECT name, birth_year FROM izone ORDER BY birth_year;
```

```
SQL> -- 2. UNION, UNION ALL을 이용한 합집합
SQL> ---- 2.1 UNION (중복 허용 x)
SQL> SELECT name, birth_year FROM ive UNION SELECT name, birth_yea
r FROM izone ORDER BY birth_year;

NAME       BIRTH_YEAR
---------- ----------
eunbi            1995
sakura           1998
yena             1999
hyewon           1999
chaeyeon         2000
chaewon          2000
nako             2001
minju            2001
yuri             2001
hitomi           2001
gaeul            2002
yujin            2003
rei              2004
wonyoung         2004
liz              2004
leeseo           2007

16 rows selected.
```

UNION은 중복을 허용하지 않기 때문에, izone테이블과 ive 테이블에 중복으로 존재하는 데이터 wonyoung과 yujin이 한 번씩만 등장하는 것을 확인할 수 있다.

```
SQL> ---- 2.2 UNION ALL (중복 허용 O)
SQL> SELECT name, birth_year FROM ive UNION ALL SELECT name, birth
_year FROM izone ORDER BY birth_year;

NAME       BIRTH_YEAR
---------- ----------
eunbi            1995
sakura           1998
hyewon           1999
yena             1999
chaeyeon         2000
chaewon          2000
yuri             2001
hitomi           2001
nako             2001
minju            2001
gaeul            2002
yujin            2003
yujin            2003
liz              2004
wonyoung         2004
rei              2004
wonyoung         2004
leeseo           2007

18 rows selected.
```

UNION ALL은 중복을 허용하기 때문에, wonyoung과 yujin이 두 번 등장하는 것을 확인할 수 있다.

3. INTERSECT를 이용한 교집합

```
-- 3. INTERSECT를 이용한 교집합
SELECT name, birth_year FROM ive INTERSECT SELECT name, birth_year FROM izone;
```

```
SQL> -- 3. INTERSECT를 이용한 교집합
SQL> SELECT name, birth_year FROM ive INTERSECT SELECT name, birth
_year FROM izone;

NAME       BIRTH_YEAR
---------- ----------
wonyoung         2004
yujin            2003

2 rows selected.
```

izone테이블과 ive 테이블에 중복으로 존재하는 데이터 wonyoung과 yujin이 결과로 반환된다.

4. MINUS를 이용한 차집합

```
-- 4. MINUS를 이용한 차집합
SELECT name, birth_year FROM ive MINUS SELECT name, birth_year FROM izone;
```

```
SQL> -- 4. MINUS를 이용한 차집합
SQL> SELECT name, birth_year FROM ive MINUS SELECT name, birth_yea
r FROM izone;

NAME       BIRTH_YEAR
---------- ----------
gaeul            2002
leeseo           2007
liz              2004
rei              2004

4 rows selected.
```

ive 테이블을 기준으로, izone테이블과 ive 테이블에 중복으로 존재하는 데이터 wonyoung과 yujin을 제외한 데이터들이 결과로 반환되는 것을 확인할 수 있다.

1.sh

```
SET ECHO ON

-- 1. 두 개의 테이블 조회
SELECT * FROM izone;
SELECT * FROM ive;

-- 2. UNION, UNION ALL을 이용한 합집합
---- 2.1 UNION (중복 허용 x)
SELECT name, birth_year FROM ive UNION SELECT name, birth_year FROM izone ORDER BY birth_year;
---- 2.2 UNION ALL (중복 허용 O)
SELECT name, birth_year FROM ive UNION ALL SELECT name, birth_year FROM izone ORDER BY birth_year;


-- 3. INTERSECT를 이용한 교집합
SELECT name, birth_year FROM ive INTERSECT SELECT name, birth_year FROM izone;

-- 4. MINUS를 이용한 차집합
SELECT name, birth_year FROM ive MINUS SELECT name, birth_year FROM izone;

EXIT
```

2. 연산식의 변환

2.1 설명

연산식 내의 피연산자의 데이터 타입이 연산자가 요구하는 타입이 아닌 경우 Tibero에서는 가능하면 데이터 타입의 변환을 수행한다.

- 모든 데이터 타입 간의 변환이 가능한 것은 아니며, 연산식의 계산 과정에서 데이터 타입의 변환이 불가능한 경우에는 에러를 반환한다.
- 데이터 타입의 변환이 가능하더라도 실제 값에 따라 에러가 발생할 수 있다.

```
'30' + 50 = 80
```

덧셈(+) 연산자의 피연산자는 NUMBER 타입이어야 하나 문자열 '30'이 온 경우 이를 먼저 NUMBER 타입으로 변환한 후에 계산하게 된다.

```
'YEAR' || 2004 = 'YEAR2004'
```

접합(||) 연산자의 피연산자는 문자열 타입이어야 하나 NUMBER 타입의 2004가 온 경우 이를 먼저 문자열로 변환한 후에 연산을 수행한다.

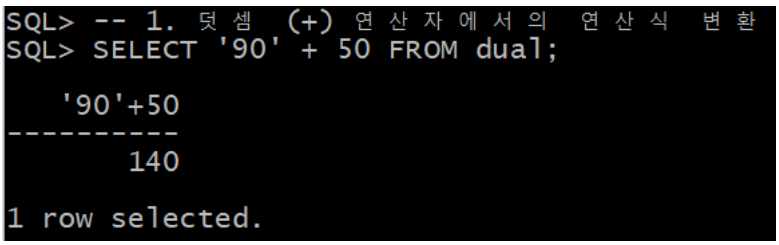
2.2 시나리오 수행

시나리오 내용
1. 덧셈 (+) 연산자에서의 연산식 변환
2. 접합 (||) 연산자에서의 연산식 변환
3. 접합 (||) 후 덧셈 (+) 연산자에서의 연산식 변환

2.3 시나리오 수행 결과

1. 덧셈 (+) 연산자에서의 연산식 변환

```
-- 1. 덧셈 (+) 연산자에서의 연산식 변환
SELECT '90' + 50 FROM dual;
```



2. 접합 (||) 연산자에서의 연산식 변환

```
-- 2. 접합 (||) 연산자에서의 연산식 변환
SELECT 'YEAR' || 2022 FROM dual;
```

```
SQL> -- 2. 접 합  (||) 연 산 자 에 서 의  연 산 식  변 환
SQL> SELECT 'YEAR' || 2022 FROM dual;

'YEAR'||2022
------------
YEAR2022

1 row selected.
```

3. 접합 (||) 후 덧셈 (+) 연산자에서의 연산식 변환

```
-- 3. 접합 (||) 후 덧셈 (+) 연산자에서의 연산식 변환
SELECT debut_order, birth_year, (debut_order || ' ') + birth_year "sum", name FROM loona ORDER BY debut_order;
```

```
SQL> -- 3. 접 합  (||) 후  덧 셈  (+) 연 산 자 에 서 의  연 산 식  변 환
SQL> SELECT debut_order, birth_year, (debut_order || ' ') + birth_year "sum"
, name FROM loona ORDER BY debut_order;

DEBUT_ORDER BIRTH_YEAR        sum NAME
----------- ---------- ---------- ----------
          1       2000       2001 Heejin
          2       2000       2002 Hyunjin
          3       1997       2000 Haseul
          4       2002       2006 Yeojin
          5       1996       2001 Bibi
          6       1997       2003 Kimlip
          7       1997       2004 Jinsol
          8       2001       2009 Choiri
          9       1997       2006 Eve
         10       1999       2009 Chuu
         11       2000       2011 Gowon
         12       2001       2013 Olivia Hye

12 rows selected.
```

2.sh

```
SET ECHO ON

-- 1. 덧셈 (+) 연산자에서의 연산식 변환
SELECT '90' + 50 FROM dual;
-- 2. 접합 (||) 연산자에서의 연산식 변환
SELECT 'YEAR' || 2022 FROM dual;
-- 3. 접합 (||) 후 덧셈 (+) 연산자에서의 연산식 변환
SELECT debut_order, birth_year, (debut_order || ' ') + birth_year "sum", name FROM loona ORDER BY debut_order;

EXIT
```

---

3. CASE 연산식

3.1 설명

CASE 연산식은 SQL 문장에서 IF... THEN ... ELSE 로직을 표현한다.

```
SELECT ...
CASE WHEN (조건1) THEN (나타낼 값1)
     WHEN (조건2) THEN (나타낼 값2)
...
ELSE (나타낼 값N)
END AS (새로운 열 이름)
```

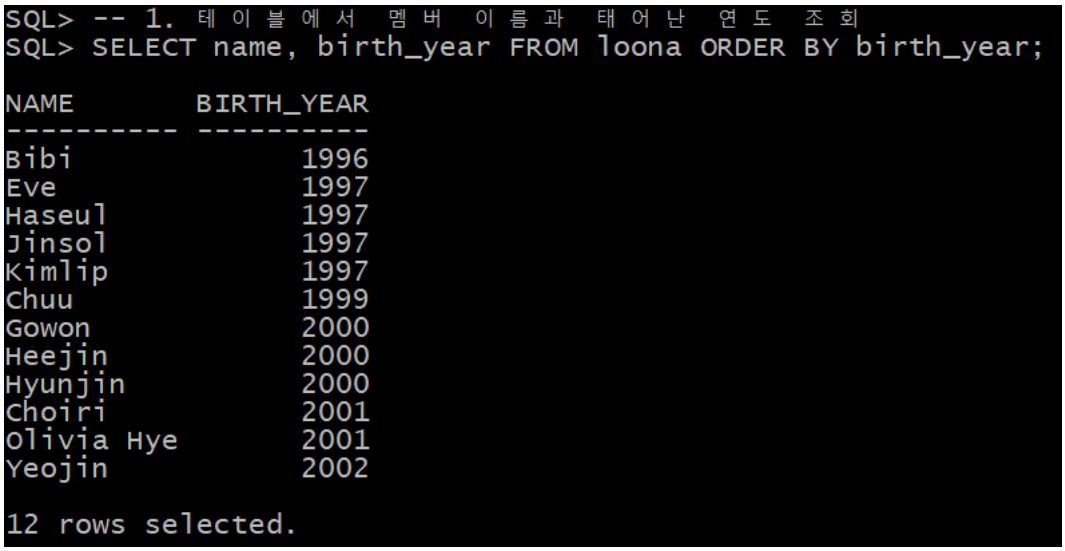
3.2 시나리오 수행

시나리오 내용
1. 테이블에서 멤버 이름과 태어난 연도 조회
2. 1999년 이전에 태어난 멤버들과 2000년 이후에 태어난 멤버들을 분류하여 조회

3.3 시나리오 수행 결과

1. 테이블에서 멤버 이름과 태어난 연도 조회

```
-- 1. 테이블에서 멤버 이름과 태어난 연도 조회
SELECT name, birth_year FROM loona ORDER BY birth_year;
```



2. 1999년 이전에 태어난 멤버들과 2000년 이후에 태어난 멤버들을 분류하여 조회

```
-- 2. 1999년 이전에 태어난 멤버들과 2000년 이후에 태어난 멤버들을 분류하여 조회
SELECT NAME,
  CASE WHEN birth_year BETWEEN 1996 AND 1999 THEN '~1999'
       WHEN birth_year BETWEEN 2000 AND 2002 THEN '2000~'
  END AS before2000
FROM loona
ORDER BY birth_year;
```

```
SQL> -- 2. 1999년 이전에 태어난 멤버들과 2000년 이후에 태어난 멤버들을 분류
하여 조회
SQL> SELECT NAME, CASE WHEN birth_year BETWEEN 1996 AND 1999 THEN '~1999' WH
EN birth_year BETWEEN 2000 AND 2002 THEN '2000~' END AS before2000 FROM loon
a ORDER BY birth_year;

NAME       BEFORE2000
---------- ----------
Bibi       ~1999
Eve        ~1999
Haseul     ~1999
Jinsol     ~1999
Kimlip     ~1999
Chuu       ~1999
Gowon      2000~
Heejin     2000~
Hyunjin    2000~
Choiri     2000~
Olivia Hye 2000~
Yeojin     2000~

12 rows selected.
```

Bibi 부터 Chuu 까지는 1999년 이전에 태어났기 때문에 '~1999'로 나타나고,

Gowon부터 Yeojin까지는 2000년 이후에 태어났기 때문에 '2000~'로 나타난 것을 알 수 있다.

3.sh

```
SET ECHO ON

-- 1. 테이블에서 멤버 이름과 태어난 연도 조회
SELECT name, birth_year FROM loona ORDER BY birth_year;
-- 2. 1999년 이전에 태어난 멤버들과 2000년 이후에 태어난 멤버들을 분류하여 조회
SELECT NAME, CASE WHEN birth_year BETWEEN 1996 AND 1999 THEN '~1999' WHEN birth_year BETWEEN 2000 AND 2002 THEN '2000~' END AS before2

EXIT
```

---

4. 변수

4.1 설명

변수를 사용하고 싶으면 변수명 앞에 :을 사용해서 선언하면 변수처리가 가능해진다.

- 변수선언 하기

```
VARIABLE :(변수명) (타입);

ex)
VARIABLE :X NUMBER;
```

- 변수에 값 넣기

```
EXECUTE :(변수명) := (값);

ex)
execute :X := 1997;
```

4.2 시나리오 수행

시나리오 내용

1. 변수1 선언 및 테이블 조회
2. 변수1 값 재할당 및 변수 2 선언 및 테이블 조회

4.3 시나리오 수행 결과

1. 변수1 선언 및 테이블 조회

```
-- 1. 변수1 선언 및 테이블 조회
VARIABLE :X NUMBER;
execute :X := 1997;
SELECT name FROM loona WHERE birth_year = :X;
```

```
SQL> -- 1. 변수1 선언 및 테이블 조회
SQL> VARIABLE :X NUMBER;
SQL> execute :X := 1997;

PSM completed.

SQL> SELECT name, birth_year FROM loona WHERE birth_year = :X;

NAME       BIRTH_YEAR
---------- ----------
Eve              1997
Haseul           1997
Jinsol           1997
Kimlip           1997

4 rows selected.
```

X에 1997을 넣고 X값과 일치하는 연도에 태어난 멤버를 조회하면 1997년에 태어난 멤버들이 출력된다.

2. 변수1 값 재할당 및 변수 2 선언 및 테이블 조회

```
-- 2. 변수1 값 재할당 및 변수 2 선언 및 테이블 조회
execute :X := 1999;
VARIABLE :Y NUMBER;
execute :Y := 2000;
SELECT name, birth_year FROM loona WHERE birth_year >= :X and birth_year <= :Y ORDER BY birth_year;
```

```
SQL> -- 2. 변수1 값 재할당 및 변수 2 선언 및 테이블 조회
SQL> execute :X := 1999;

PSM completed.

SQL> VARIABLE :Y NUMBER;
SQL> execute :Y := 2000;

PSM completed.

SQL> SELECT name, birth_year FROM loona WHERE birth_year >= :X and
birth_year <= :Y ORDER BY birth_year;

NAME       BIRTH_YEAR
---------- ----------
Chuu             1999
Gowon            2000
Heejin           2000
Hyunjin          2000

4 rows selected.
```

X에 1999 값을 새로 넣고, 새로운 변수 Y를 선언하여 2000을 넣는다. X값 이상 Y년 이하 연도에 태어난 멤버를 조회하면 1999, 2000년에 태어난 멤버들이 출력된다.

4.sh

```
-- 1. 변수1 선언 및 테이블 조회
VARIABLE :X NUMBER;
execute :X := 1997;
SELECT name, birth_year FROM loona WHERE birth_year = :X;

-- 2. 변수1 값 재할당 및 변수 2 선언 및 테이블 조회
execute :X := 1999;
VARIABLE :Y NUMBER;
execute :Y := 2000;
SELECT name, birth_year FROM loona WHERE birth_year >= :X and birth_year <= :Y ORDER BY birth_year;
```

---

5. 그룹 조건식 ANY, SOME, ALL

5.1 설명

ANY	- 왼쪽의 데이터 값이 오른쪽의 리스트 내의 값 중 **최소한 하나만 단순 비교 연산자를 만족**하면 그룹 조건식은 TRUE를 반환한다.
SOME	- 왼쪽의 데이터 값이 오른쪽의 리스트 내의 값 중 **최소한 하나만 단순 비교 연산자를 만족**하면 그룹 조건식은 TRUE를 반환한다. - ANY와 똑같다.
ALL	- 왼쪽의 데이터 값이 오른쪽의 리스트 내의 **모든 값에 대해 단순 비교 연산자를 만족**해야 그룹 조건식이 TRUE를 반환한다.

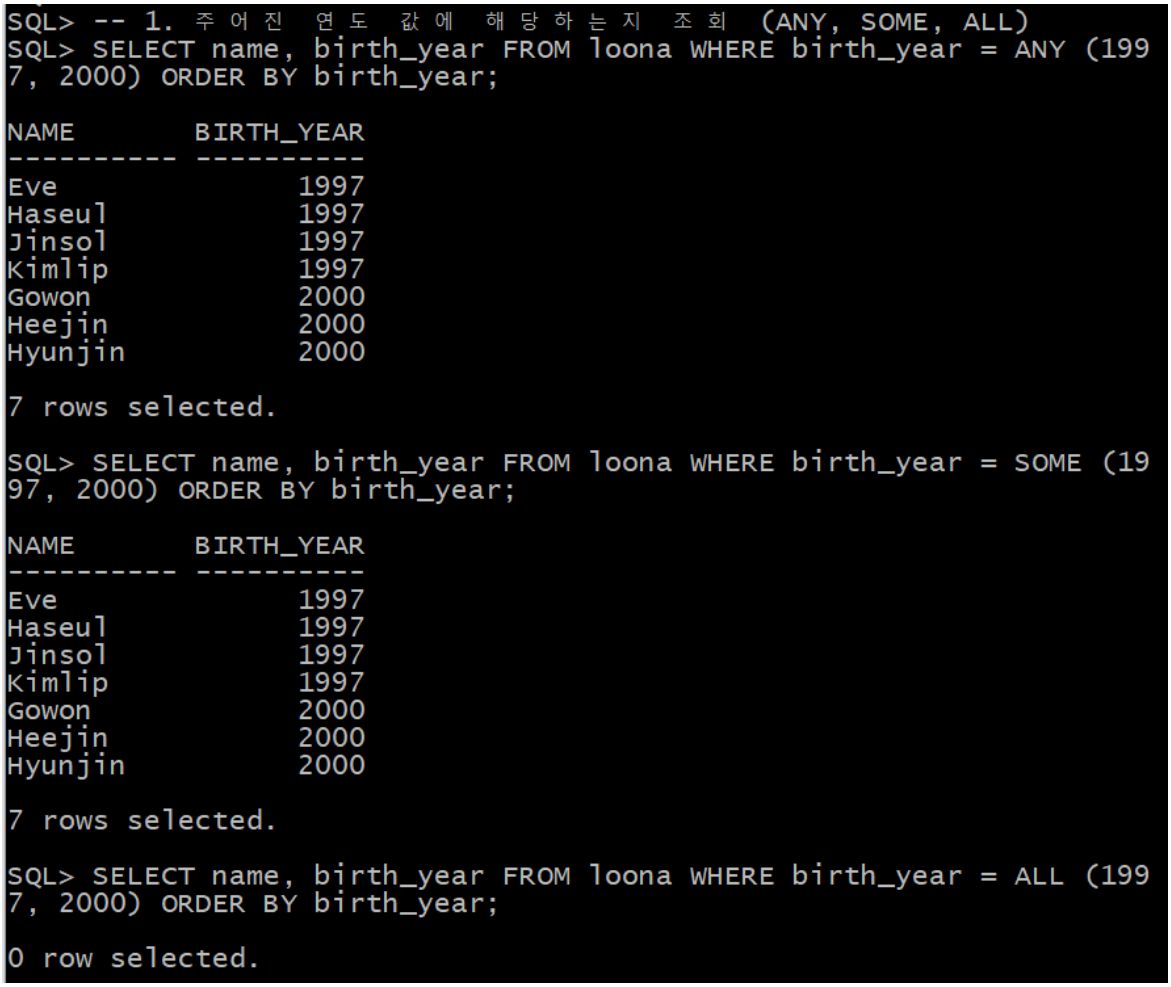
5.2 시나리오 수행

시나리오 내용
1. 주어진 연도 값에 해당하는지 조회 (ANY, SOME, ALL)
2. 주어진 연도 값보다 크거나 같은지 조회 (ANY, SOME, ALL)

5.3 시나리오 수행 결과

1. 주어진 연도 값에 해당하는지 조회 (ANY, SOME, ALL)

```
-- 1. 주어진 연도 값에 해당하는지 조회 (ANY, SOME, ALL)
SELECT name, birth_year FROM loona WHERE birth_year = ANY (1997, 2000) ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year = SOME (1997, 2000) ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year = ALL (1997, 2000) ORDER BY birth_year;
```

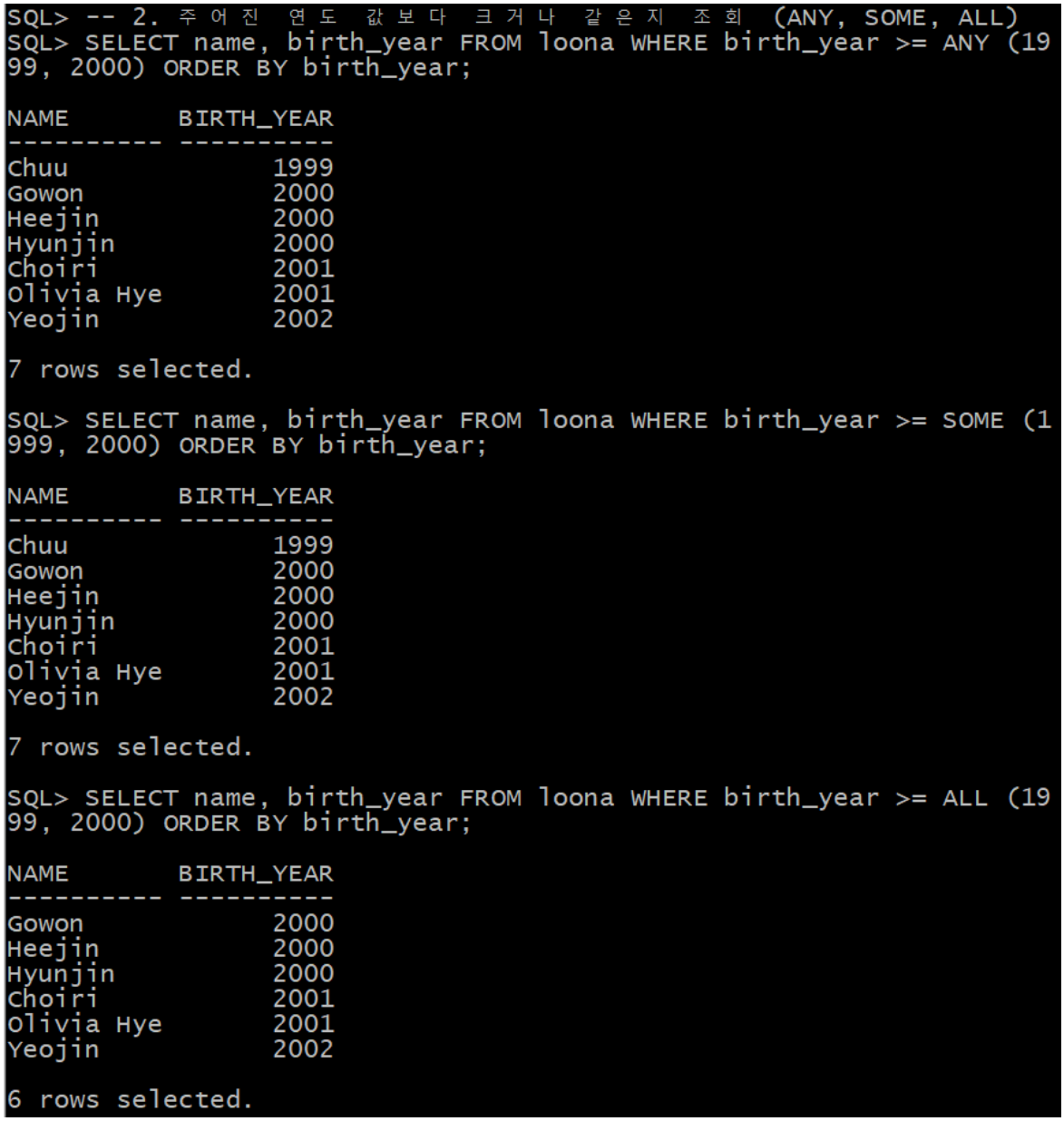


```
SQL> -- 1. 주어진 연도 값에 해당하는지 조회 (ANY, SOME, ALL)
SQL> SELECT name, birth_year FROM loona WHERE birth_year = ANY (199
7, 2000) ORDER BY birth_year;

NAME       BIRTH_YEAR
---------- ----------
Eve              1997
Haseul           1997
Jinsol           1997
Kimlip           1997
Gowon            2000
Heejin           2000
Hyunjin          2000

7 rows selected.

SQL> SELECT name, birth_year FROM loona WHERE birth_year = SOME (19
97, 2000) ORDER BY birth_year;

NAME       BIRTH_YEAR
---------- ----------
Eve              1997
Haseul           1997
Jinsol           1997
Kimlip           1997
Gowon            2000
Heejin           2000
Hyunjin          2000

7 rows selected.

SQL> SELECT name, birth_year FROM loona WHERE birth_year = ALL (199
7, 2000) ORDER BY birth_year;

0 row selected.
```

- birth_year = ANY (1997, 2000) 과 birth_year = SOME (1997, 2000)
  - 1997년과 2000년 둘 중 하나에 해당되기만 하면 select 된다.
- birth_year = ALL (1997, 2000)
  - 1997년과 2000년 둘 다 해당되어야 select 된다. 위의 예제에서는 해당되는 경우가 없기 때문에 0건이 조회된다.

2. 주어진 연도 값보다 크거나 같은지 조회 (ANY, SOME, ALL)

```
-- 2. 주어진 연도 값보다 크거나 같은지 조회 (ANY, SOME, ALL)
SELECT name, birth_year FROM loona WHERE birth_year >= ANY (1999, 2000) ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year >= SOME (1999, 2000) ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year >= ALL (1999, 2000) ORDER BY birth_year;
```



- birth_year >= ANY (1999, 2000) 과 birth_year >= SOME (1999, 2000)
  - 1999년과 2000년 둘 중 하나보다 크거나 같은 값이면 select 된다. 따라서 1999년생인 Chuu도 함께 조회되는 것을 확인할 수 있다.
- birth_year >= ALL (1999, 2000)
  - 1999년과 2000년 둘 보다 크거나 같은 값이면 select 된다. 따라서 1999년생인 Chuu는 2000년보다 크거나 같은 값이 아니기 때문에 함께 조회되지 않는다.

5.sh

```
SET ECHO ON

-- 1. 주어진 연도 값에 해당하는지 조회 (ANY, SOME, ALL)
SELECT name, birth_year FROM loona WHERE birth_year = ANY (1997, 2000) ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year = SOME (1997, 2000) ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year = ALL (1997, 2000) ORDER BY birth_year;

-- 2. 주어진 연도 값보다 크거나 같은지 조회 (ANY, SOME, ALL)
SELECT name, birth_year FROM loona WHERE birth_year >= ANY (1999, 2000) ORDER BY birth_year;
```

```
SELECT name, birth_year FROM loona WHERE birth_year >= SOME (1999, 2000) ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year >= ALL (1999, 2000) ORDER BY birth_year;

EXIT
```

---

6. BETWEEN, EXISTS, IN 조건식

6.1 설명

BETWEEN ... AND

- 왼쪽의 수치 값이 오른쪽의 두 수치 값 **사이에 존재하는지 비교**하는 연산자
- BETWEEN 예약어 앞에 NOT이 오면 BETWEEN 연산자 결과에 NOT 연산을 수행한 결과를 반환

EXISTS

- EXISTS 연산자의 오른쪽 subquery를 실행한 결과가

  하나 이상의 로우를 반환하면 TRUE, 그렇지 않으면 FALSE를 반환

IN

- IN 연산자는 왼쪽의 데이터 값이 **오른쪽의 리스트 내에 포함되어 있는지 비교**하는 연산자

6.2 시나리오 수행

시나리오 내용
1. BETWEEN ... AND 조건식
2. EXISTS 조건식
3. IN 조건식

6.3 시나리오 수행 결과

1. BETWEEN ... AND 조건식

```
-- 1. BETWEEN ... AND 조건식
SELECT name, birth_year FROM loona WHERE birth_year BETWEEN 1999 AND 2000 ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year NOT BETWEEN 1999 AND 2000 ORDER BY birth_year;
```

```
SQL> -- 1. BETWEEN ... AND 조 건 식
SQL> SELECT name, birth_year FROM loona WHERE birth_year BETWEEN 1999 AN
D 2000 ORDER BY birth_year;

NAME       BIRTH_YEAR
---------- ----------
Chuu             1999
Gowon            2000
Heejin           2000
Hyunjin          2000

4 rows selected.

SQL> SELECT name, birth_year FROM loona WHERE birth_year NOT BETWEEN 199
9 AND 2000 ORDER BY birth_year;

NAME       BIRTH_YEAR
---------- ----------
Bibi             1996
Eve              1997
Haseul           1997
Jinsol           1997
Kimlip           1997
Choiri           2001
Olivia Hye       2001
Yeojin           2002

8 rows selected.
```

첫번째 쿼리 - 태어난 연도가 1999 이상 2000 이하인 값에 해당하는 멤버들만 출력되었다.

두번째 쿼리 - 태어난 연도가 1999 이상 2000 이하인 값에 해당하지 않는 멤버들만 출력되었다.

2. EXISTS 조건식

```
-- 2. EXISTS 조건식
SELECT name, birth_year, debut_order FROM loona WHERE birth_year >= 2000 AND EXISTS (SELECT * FROM loona WHERE name = 'Eve') ORDER
SELECT name, birth_year, debut_order FROM loona WHERE birth_year >= 2000 AND EXISTS (SELECT * FROM loona WHERE name = 'Test') ORDE
```

```
SQL> -- 2. EXISTS 조 건 식
SQL> SELECT name, birth_year, debut_order FROM loona WHERE birth_year >=
 2000 AND EXISTS (SELECT * FROM loona WHERE name = 'Eve') ORDER BY birth
_year;

NAME       BIRTH_YEAR DEBUT_ORDER
---------- ---------- -----------
Gowon            2000          11
Heejin           2000           1
Hyunjin          2000           2
Choiri           2001           8
Olivia Hye       2001          12
Yeojin           2002           4

6 rows selected.

SQL> SELECT name, birth_year, debut_order FROM loona WHERE birth_year >=
 2000 AND EXISTS (SELECT * FROM loona WHERE name = 'Test') ORDER BY birt
h_year;

0 row selected.
```

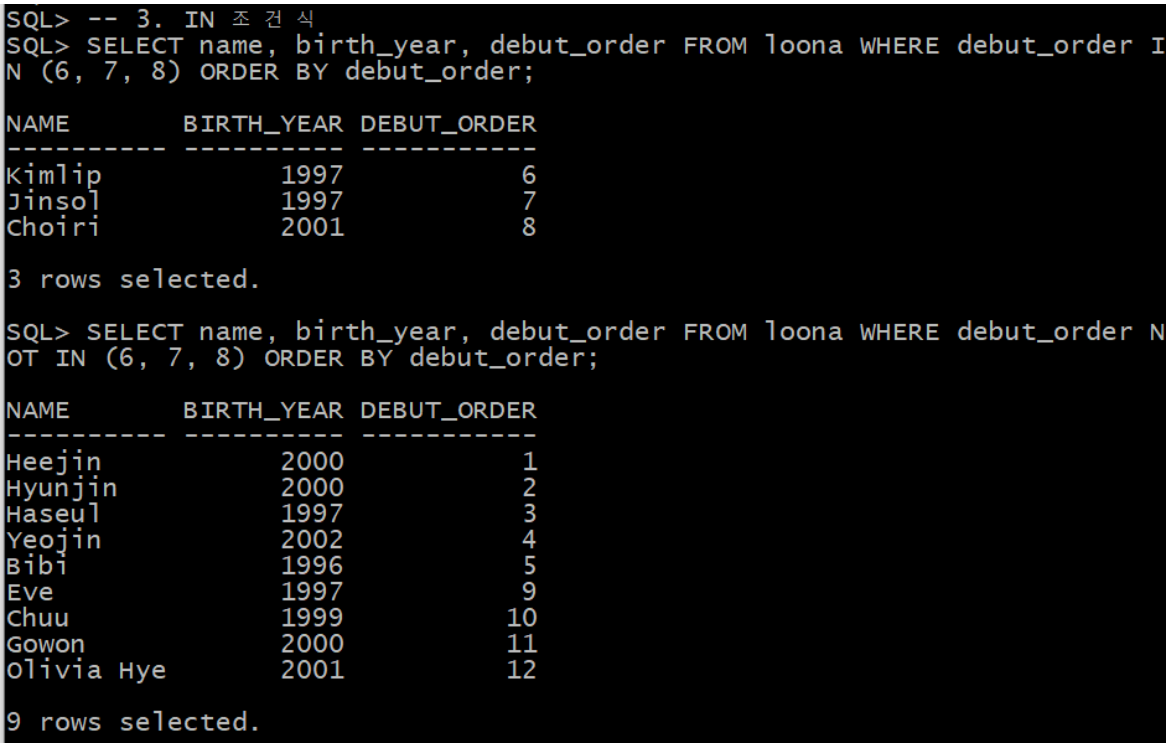
EXISTS 조건식은 서브 쿼리에 일치하는 결과가 한 건이라도 있으면 쿼리를 더 이상 수행하지 않는다.

첫번째 쿼리 - 서브 쿼리의 결과가 참이기 때문에 WHERE 절에 True가 반환되어서 정상적으로 조회할 수 있다.

두번째 쿼리 - 서브 쿼리의 결과가 거짓이기 때문에 WHERE 절에 False가 반환되어서 정상적으로 조회할 수 없다.

3. IN 조건식

```
-- 3. IN 조건식
SELECT name, birth_year, debut_order FROM loona WHERE debut_order IN (6, 7, 8) ORDER BY debut_order;
SELECT name, birth_year, debut_order FROM loona WHERE debut_order NOT IN (6, 7, 8) ORDER BY debut_order;
```



```
SQL> -- 3. IN 조 건 식
SQL> SELECT name, birth_year, debut_order FROM loona WHERE debut_order I
N (6, 7, 8) ORDER BY debut_order;

NAME       BIRTH_YEAR DEBUT_ORDER
---------- ---------- -----------
Kimlip           1997           6
Jinsol           1997           7
Choiri           2001           8

3 rows selected.

SQL> SELECT name, birth_year, debut_order FROM loona WHERE debut_order N
OT IN (6, 7, 8) ORDER BY debut_order;

NAME       BIRTH_YEAR DEBUT_ORDER
---------- ---------- -----------
Heejin           2000           1
Hyunjin          2000           2
Haseul           1997           3
Yeojin           2002           4
Bibi             1996           5
Eve              1997           9
Chuu             1999          10
Gowon            2000          11
Olivia Hye       2001          12

9 rows selected.
```

첫번째 쿼리 - 데뷔 순서가 6,7,8 중에 하나에 해당하는 멤버들을 출력한다.

두번째 쿼리 - 데뷔 순서가 6,7,8 중에 하나에 속하지 않는 멤버들을 출력한다.

6.sh

```
SET ECHO ON

-- 1. BETWEEN ... AND 조건식
SELECT name, birth_year FROM loona WHERE birth_year BETWEEN 1999 AND 2000 ORDER BY birth_year;
SELECT name, birth_year FROM loona WHERE birth_year NOT BETWEEN 1999 AND 2000 ORDER BY birth_year;

-- 2. EXISTS 조건식
SELECT name, birth_year, debut_order FROM loona WHERE birth_year >= 2000 AND EXISTS (SELECT * FROM loona WHERE name = 'Eve') ORDER BY
SELECT name, birth_year, debut_order FROM loona WHERE birth_year >= 2000 AND EXISTS (SELECT * FROM loona WHERE name = 'Test') ORDER BY

-- 3. IN 조건식
SELECT name, birth_year, debut_order FROM loona WHERE debut_order IN (6, 7, 8) ORDER BY debut_order;
SELECT name, birth_year, debut_order FROM loona WHERE debut_order NOT IN (6, 7, 8) ORDER BY debut_order;

EXIT
```

---

7. LIKE 조건식

7.1 설명

LIKE

- 문자열 데이터 간의 패턴을 비교하는 연산자이다.
- 왼쪽에 오는 문자열이 오른쪽에 오는 문자열 패턴에 일치하면 TRUE를 반환한다.

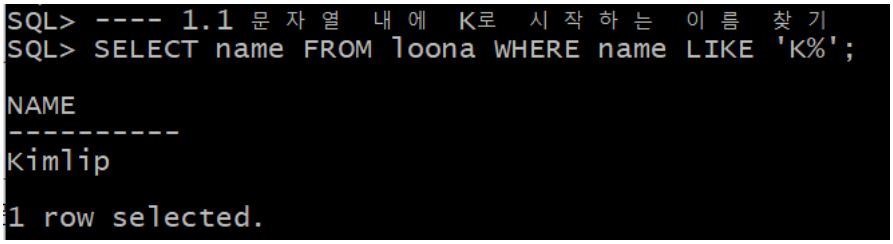
7.2 시나리오 수행

시나리오 내용
1. LIKE 조건식 ---- 1.1 문자열 내에 K로 시작하는 이름 찾기 ---- 1.2 문자열 내에 i로 끝나는 이름 찾기 ---- 1.3 문자열 내에 e가 들어 있는 이름 찾기 ---- 1.4 문자열 내에 e로 시작하고 한 문자 건너뛰고, 다음 문자가 e로 끝나는 이름 찾기

7.3 시나리오 수행 결과

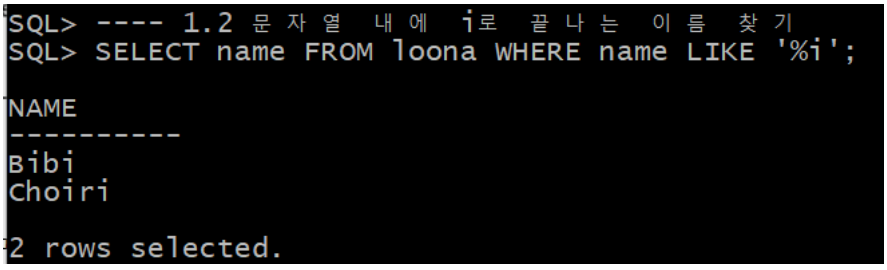
1. LIKE 조건식
---- 1.1 문자열 내에 K로 시작하는 이름 찾기

```
---- 1.1 문자열 내에 K로 시작하는 이름 찾기
SELECT name FROM loona WHERE name LIKE 'K%';
```



---- 1.2 문자열 내에 i로 끝나는 이름 찾기

```
---- 1.2 문자열 내에 i로 끝나는 이름 찾기
SELECT name FROM loona WHERE name LIKE '%i';
```



---- 1.3 문자열 내에 e가 들어 있는 이름 찾기

```
---- 1.3 문자열 내에 e가 들어 있는 이름 찾기
SELECT name FROM loona WHERE name LIKE '%e%';
```

```
SQL> ---- 1.3 문 자 열  내 에  e가  들 어  있 는  이 름  찾 기
SQL> SELECT name FROM loona WHERE name LIKE '%e%';

NAME
----------
Eve
Haseul
Heejin
Olivia Hye
Yeojin

5 rows selected.
```

---- 1.4 문자열 내에 e로 시작하고 한 문자 건너뛰고, 다음 문자가 e로 끝나는 이름 찾기

```
---- 1.4 문자열 내에 e로 시작하고 한 문자 건너뛰고, 다음 문자가 e로 끝나는 이름 찾기
SELECT name FROM loona WHERE name LIKE 'E_e';
```

```
SQL> ---- 1.4 문 자 열  내 에  e로  시 작 하 고  한  문 자  건 너 뛰 고 , 다 음  문 자 가  e로
 끝 나 는  이 름  찾 기
SQL> SELECT name FROM loona WHERE name LIKE 'E_e';

NAME
----------
Eve

1 row selected.
```

7.sh

```
SET ECHO ON

-- 1. LIKE 조건식
---- 1.1 문자열 내에 K로 시작하는 이름 찾기
SELECT name FROM loona WHERE name LIKE 'K%';
---- 1.2 문자열 내에 i로 끝나는 이름 찾기
SELECT name FROM loona WHERE name LIKE '%i';
---- 1.3 문자열 내에 e가 들어 있는 이름 찾기
SELECT name FROM loona WHERE name LIKE '%e%';
---- 1.4 문자열 내에 e로 시작하고 한 문자 건너뛰고, 다음 문자가 e로 끝나는 이름 찾기
SELECT name FROM loona WHERE name LIKE 'E_e';

EXIT
```

---

8. REGEXP_LIKE 조건식

8.1 설명

REGEXP_LIKE

- 정규표현식으로 된 문자열 패턴을 비교한다는 점을 제외하고는 LIKE와 동일

8.2 시나리오 수행

시나리오 내용

1. REGEXP_LIKE 조건식 ---- 1.1 문자열 내에 특정문자(r, e) 문자가 들어가 있는 이름 찾기 ---- 1.2 특정문자(n) 으로 끝나는 이름 찾기 ---- 1.3 특정문자(Y)로 시작하는 이름 찾기 ---- 1.4 첫문자가 A~H 까지의 문자로 시작해서, 한문자 건너뛰고, 다음문자가 u 인 이름 찾기 ---- 1.5 u 문자가 2번 반복되는 이름 찾기

8.3 시나리오 수행 결과

1. REGEXP_LIKE 조건식
---- 1.1 문자열 내에 특정문자(r, e) 문자가 들어가 있는 이름 찾기

```
---- 1.1 문자열 내에 특정문자(r, e) 문자가 들어가 있는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, 'r|e');
```

```
SQL> ---- 1.1 문자열 내에 특정문자(r, e) 문자가 들어가 있는 이름 찾기
SQL> SELECT name FROM loona WHERE REGEXP_LIKE (name, 'r|e');

NAME
----------
Choiri
Eve
Haseul
Heejin
Olivia Hye
Yeojin

6 rows selected.
```

---- 1.2 특정문자(n) 으로 끝나는 이름 찾기

```
---- 1.2 특정문자(n) 으로 끝나는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, 'n$');
```

```
SQL> ---- 1.2 특정문자(n) 으로 끝나는 이름 찾기
SQL> SELECT name FROM loona WHERE REGEXP_LIKE (name, 'n$');

NAME
----------
Gowon
Heejin
Hyunjin
Yeojin

4 rows selected.
```

---- 1.3 특정문자(Y)로 시작하는 이름 찾기

```
---- 1.3 특정문자(Y)로 시작하는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, '^Y');
```

```
SQL> ---- 1.3 특정문자(Y)로 시작하는 이름 찾기
SQL> SELECT name FROM loona WHERE REGEXP_LIKE (name, '^Y');

NAME
----------
Yeojin

1 row selected.
```

---- 1.4 첫문자가 A~H 까지의 문자로 시작해서, 한문자 건너뛰고, 다음문자가 u 인 이름 찾기

```
---- 1.4 첫문자가 A~H 까지의 문자로 시작해서, 한문자 건너뛰고, 다음문자가 u 인 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, '^[A-H].u');
```

```
SQL> ---- 1.4 첫 문 자 가  A~H 까 지 의  문 자 로  시 작 해 서 , 한 문 자  건 너 뛰 고 , 다 음
문 자 가  u 인  이 름  찾 기
SQL> SELECT name FROM loona WHERE REGEXP_LIKE (name, '^[A-H].u');

NAME
----------
Chuu
Hyunjin

2 rows selected.
```

---- 1.5 u 문자가 2번 반복되는 이름 찾기

```
---- 1.5 u 문자가 2번 반복되는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, 'u{2}');
```

```
SQL> ---- 1.5 u 문 자 가  2번  반 복 되 는  이 름  찾 기
SQL> SELECT name FROM loona WHERE REGEXP_LIKE (name, 'u{2}');

NAME
----------
Chuu

1 row selected.
```

8.sh

```
SET ECHO ON

-- 1. REGEXP_LIKE 조건식
---- 1.1 문자열 내에 특정문자(r, e) 문자가 들어가 있는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, 'r|e');
---- 1.2 특정문자(n) 으로 끝나는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, 'n$');
---- 1.3 특정문자(Y)로 시작하는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, '^Y');
---- 1.4 첫문자가 A~H 까지의 문자로 시작해서, 한문자 건너뛰고, 다음문자가 u 인 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, '^[A-H].u');
---- 1.5 u 문자가 2번 반복되는 이름 찾기
SELECT name FROM loona WHERE REGEXP_LIKE (name, 'u{2}');

EXIT
```

---

9. 문자함수

9.1 설명

명령어	설명
LOWER/UPPER	문자를 모두 소문자로 변경 / 대문자로 변경

명령어	설명
INITCAP	첫번째 글자만 대문자로 변경
CONCAT	두 문자열을 결합한 값을 반환
INSTR	문자 내의 특정 스트링의 위치를 구함
LENGTH	문자열의 길이를 구함
REPLACE	특정 문자열을 대신
LPAD/RPAD	왼쪽/오른쪽 문자 자리 채움
SUBSTR	문자열 중 특정 문자 또는 문자열의 일부분을 선택
TRIM	왼쪽 또는 오른쪽 문자를 자름

9.2 시나리오 수행

시나리오 내용
1. LOWER : 문자를 모두 소문자로 변경
2. UPPER : 문자를 모두 대문자로 변경
3. CONCAT : 두 문자열을 결합한 값을 반환
4. INSTR : 문자 내의 특정 스트링의 위치를 구함
5. LENGTH : 문자열의 길이를 구함
6. REPLACE : 특정 문자열을 대신
7. LPAD : 왼쪽 문자 자리 채움

9.3 시나리오 수행 결과

1. LOWER : 문자를 모두 소문자로 변경

```
-- 1. LOWER
SELECT LOWER('LOONA') FROM dual;
```

```
SQL> -- 1. LOWER
SQL> SELECT LOWER('LOONA') FROM dual;

LOWER('LOONA')
--------------
loona

1 row selected.
```

2. UPPER : 문자를 모두 대문자로 변경

```
-- 2. UPPER
SELECT UPPER('loona') FROM dual;
```

```
SQL> -- 2. UPPER
SQL> SELECT UPPER('loona') FROM dual;

UPPER('LOONA')
--------------
LOONA

1 row selected.
```

3. CONCAT : 두 문자열을 결합한 값을 반환

```
-- 3. CONCAT
SELECT CONCAT('loona ', 'orbit') FROM dual;
```

```
SQL> -- 4. INSTR
SQL> SELECT INSTR('loonatheworld','t') FROM dual;

INSTR('LOONATHEWORLD','T')
--------------------------
                         6

1 row selected.
```

4. INSTR : 문자 내의 특정 스트링의 위치를 구함

```
-- 4. INSTR
SELECT INSTR('loonatheworld','t') FROM dual;
```

```
SQL> -- 4. INSTR
SQL> SELECT INSTR('loonatheworld','t') FROM dual;

INSTR('LOONATHEWORLD','T')
--------------------------
                         6

1 row selected.
```

5. LENGTH : 문자열의 길이를 구함

```
-- 5. LENGTH
SELECT LENGTH('Loonatheworld') FROM dual;
```

```
SQL> -- 5. LENGTH
SQL> SELECT LENGTH('Loonatheworld') FROM dual;

LENGTH('LOONATHEWORLD')
-----------------------
                     13

1 row selected.
```

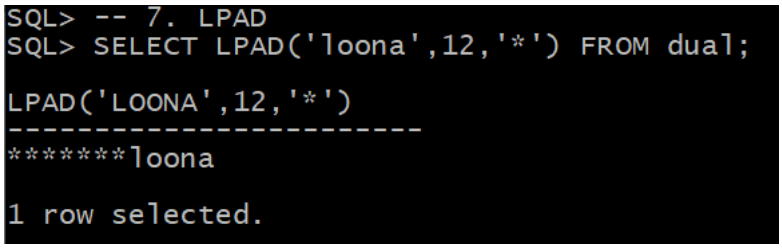
6. REPLACE : 특정 문자열을 대신

```
-- 6. REPLACE
SELECT REPLACE('loone', 'e','a') FROM dual;
```

```
SQL> -- 6. REPLACE
SQL> SELECT REPLACE('loone', 'e','a') FROM dual;

REPLACE('LOONE','E','A')
------------------------
loona

1 row selected.
```

7. LPAD : 왼쪽 문자 자리 채움

```
-- 7. LPAD
SELECT LPAD('loona',12,'*') FROM dual;
```



12글자에서 'loona'를 제외한 왼쪽부분 7칸을 *으로 채운다.

9.sh

```
SET ECHO ON

-- 1. LOWER
SELECT LOWER('LOONA') FROM dual;

-- 2. UPPER
SELECT UPPER('loona') FROM dual;

-- 3. CONCAT
SELECT CONCAT('loona ', 'orbit') FROM dual;

-- 4. INSTR
SELECT INSTR('loonatheworld','t') FROM dual;

-- 5. LENGTH
SELECT LENGTH('Loonatheworld') FROM dual;

-- 6. REPLACE
SELECT REPLACE('loone', 'e','a') FROM dual;

-- 7. LPAD
SELECT LPAD('loona',12,'*') FROM dual;

EXIT
```

---

10. 숫자 함수

10.1 설명

명령어	설명
ABS(n)	절대값
CEIL(n)	주어진 값보다 크지만 가장 근접하는 최소값을 구하는 함수
MOD(m,n)	m을 n으로 나누어 남는 값을 구하는 함수임
ROUND(n,[m])	n값의 반올림을 하는 함수로 m은 소수점 아래 자릿수를 표기함
TRUNC(n,[m])	n값을 버림하는 함수로 m은 소수점 아래 자릿수를 표기함
POWER	거듭제곱
SQRT	제곱근
SIGN	양수, 음수, 0인지를 구분

명령어	설명
CHR	ASCII값에 해당하는 문자를 구함

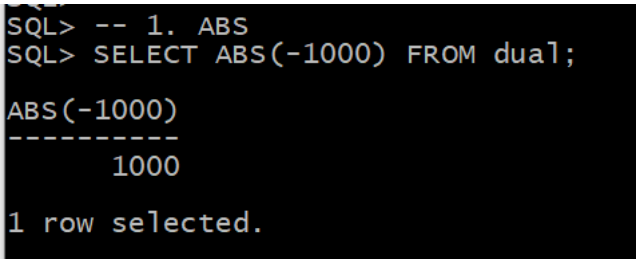
10.2 시나리오 수행

시나리오 내용
1. ABS(n) : 절대값
2. CEIL(n) : 주어진 값보다 크지만 가장 근접하는 최소값을 구하는 함수
3. MOD(m,n) : m을 n으로 나누어 남는 값을 구하는 함수임
4. ROUND(n,[m]) : n값의 반올림을 하는 함수로 m은 소수점 아래 자릿수를 표기함
5. TRUNC(n,[m]) : n값을 버림하는 함수로 m은 소수점 아래 자릿수를 표기함
6. POWER : 거듭제곱
7. SQRT : 제곱근
8. SIGN : 양수, 음수, 0인지를 구분
9. CHR : ASCII값에 해당하는 문자를 구함

10.3 시나리오 수행 결과

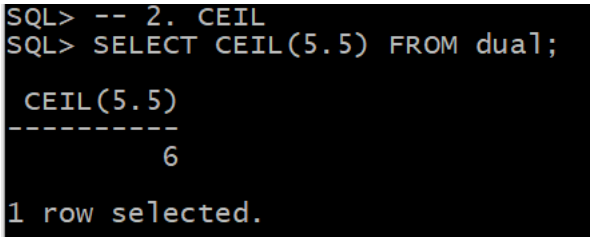
1. ABS(n) : 절대값

```
-- 1. ABS
SELECT ABS(-100) FROM dual;
```



2. CEIL(n) : 주어진 값보다 크지만 가장 근접하는 최소값을 구하는 함수

```
-- 2. CEIL
SELECT CEIL(11.1) FROM dual;
```



3. MOD(m,n) : m을 n으로 나누어 남는 값을 구하는 함수임

```
-- 3. MOD
SELECT MOD(10,4) FROM dual;
```

```
SQL> -- 3. MOD
SQL> SELECT MOD(10,3) FROM dual;

 MOD(10,3)
----------
         1

1 row selected.
```

4. ROUND(n,[m]) : n값의 반올림을 하는 함수로 m은 소수점 아래 자릿수를 표기함

```
-- 4. ROUND
SELECT ROUND(11.985, 2) FROM dual;
```

```
SQL> -- 4. ROUND
SQL> SELECT ROUND(11.985, 2) FROM dual;

ROUND(11.985,2)
---------------
          11.99

1 row selected.
```

5. TRUNC(n,[m]) : n값을 버림하는 함수로 m은 소수점 아래 자릿수를 표기함

```
-- 5. TRUNC
SELECT TRUNC(11.985, 1) FROM dual;
```

```
SQL> -- 5. TRUNC
SQL> SELECT TRUNC(11.985, 1) FROM dual;

TRUNC(11.985,1)
---------------
           11.9

1 row selected.
```

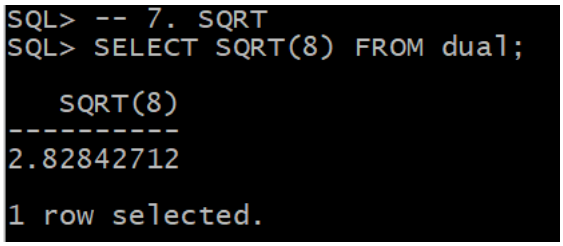
6. POWER : 거듭제곱

```
-- 6. POWER
SELECT POWER(4,2) FROM dual;
```

```
SQL> -- 6. POWER
SQL> SELECT POWER(4,2) FROM dual;

POWER(4,2)
----------
        16

1 row selected.
```

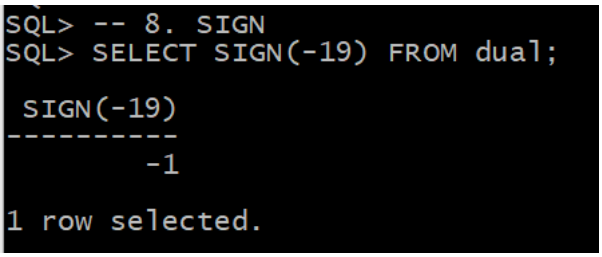
7. SQRT : 제곱근

```
-- 7. SQRT
SELECT SQRT(8) FROM dual;
```



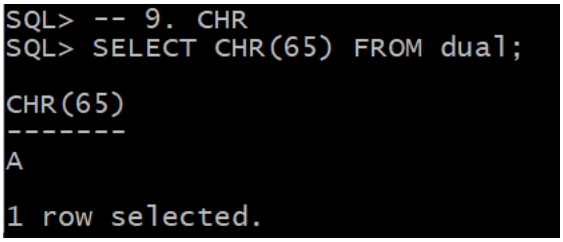
8. SIGN : 양수, 음수, 0인지를 구분

```
-- 8. SIGN
SELECT SIGN(-19) FROM dual;
```



9. CHR : ASCII값에 해당하는 문자를 구함

```
-- 9. CHR
SELECT CHR(65) FROM dual;
```



10.sh

```
SET ECHO ON

-- 1. ABS
SELECT ABS(-1000) FROM dual;

-- 2. CEIL
SELECT CEIL(5.5) FROM dual;

-- 3. MOD
SELECT MOD(10,3) FROM dual;

-- 4. ROUND
SELECT ROUND(11.985, 2) FROM dual;

-- 5. TRUNC
SELECT TRUNC(11.985, 1) FROM dual;
```

```
-- 6. POWER
SELECT POWER(4,2) FROM dual;

-- 7. SQRT
SELECT SQRT(8) FROM dual;

-- 8. SIGN
SELECT SIGN(-19) FROM dual;

-- 9. CHR
SELECT CHR(65) FROM dual;

EXIT
```

---

11. 날짜 함수

11.1 설명

명령어	설명
MONTHS_BETWEEN	두 날짜 사이의 월수를 계산
ADD_MONTHS	날짜에 월을 더함
NEXT_DAY	명시된 날짜로부터 다음 요일에 대한 날짜
LAST_DAY	월의 마지막날을 계산
ROUND	날짜를 반올림
TRUNC	날짜를 절삭

11.2 시나리오 수행

시나리오 내용
1. ADD_MONTHS : 날짜에 월을 더함
2. ROUND : 날짜를 반올림
3. TRUNC : 날짜를 절삭
4. LAST_DAY : 월의 마지막날을 계산
5. MONTHS_BETWEEN : 두 날짜 사이의 월수를 계산

11.3 시나리오 수행 결과

1. ADD_MONTHS : 날짜에 월을 더함

```
-- ADD_MONTHS
SELECT ADD_MONTHS(TO_DATE('2022/11/18','YYYY/MM/DD'),1) "Add_months" FROM dual;
```

```
SQL> -- ADD_MONTHS
SQL> SELECT ADD_MONTHS(TO_DATE('2022/11/18','YYYY/MM/DD'),1) "Add_months
" FROM dual;

Add_months
---------------------------------------------------------------
2022/12/18

1 row selected.
```

2. ROUND : 날짜를 반올림

```
-- ROUND
SELECT ROUND(TO_DATE('2022/11/18','YYYY/MM/DD'),'YEAR') "Round" FROM dual;
```

```
SQL> -- ROUND
SQL> SELECT ROUND(TO_DATE('2022/11/18','YYYY/MM/DD'),'YEAR') "Round" FRO
M dual;

Round
---------------------------------------------------------------------
2023/01/01

1 row selected.
```

3. TRUNC : 날짜를 절삭

```
-- TRUNC
SELECT TRUNC(TO_DATE('2022/11/18','YYYY/MM/DD'),'YEAR') "Trunc" FROM dual;
```

```
SQL> -- TRUNC
SQL> SELECT TRUNC(TO_DATE('2022/11/18','YYYY/MM/DD'),'YEAR') "Trunc" FRO
M dual;

Trunc
---------------------------------------------------------------------
2022/01/01

1 row selected.
```

4. LAST_DAY : 월의 마지막날을 계산

```
-- LAST_DAY
SELECT LAST_DAY(SYSDATE) FROM dual;
```

```
SQL> -- LAST_DAY
SQL> SELECT LAST_DAY(SYSDATE) FROM dual;

LAST_DAY(SYSDATE)
---------------------------------------------
2022/11/30

1 row selected.
```

5. MONTHS_BETWEEN : 두 날짜 사이의 월수를 계산

```
-- MONTHS_BETWEEN
SELECT MONTHS_BETWEEN(LAST_DAY(SYSDATE),SYSDATE) FROM dual;
```

```
SQL> -- MONTHS_BETWEEN
SQL> SELECT MONTHS_BETWEEN(LAST_DAY(SYSDATE),SYSDATE) FROM dual;

MONTHS_BETWEEN(LAST_DAY(SYSDATE),SYSDATE)
-----------------------------------------
                                .387096774

1 row selected.
```

11.sh

```
SET ECHO ON

-- ADD_MONTHS
SELECT ADD_MONTHS(TO_DATE('2022/11/18','YYYY/MM/DD'),1) "Add_months" FROM dual;

-- ROUND
SELECT ROUND(TO_DATE('2022/11/18','YYYY/MM/DD'),'YEAR') "Round" FROM dual;

-- TRUNC
SELECT TRUNC(TO_DATE('2022/11/18','YYYY/MM/DD'),'YEAR') "Trunc" FROM dual;

-- LAST_DAY
SELECT LAST_DAY(SYSDATE) FROM dual;

-- MONTHS_BETWEEN
SELECT MONTHS_BETWEEN(LAST_DAY(SYSDATE),SYSDATE) FROM dual;

EXIT
```

12. 날짜 함수

12.1 설명

명령어	설명
TO_CHAR	주어진 date type, number type의 값을 형식에 따라 문자열로 변환
TO_DATE	주어진 string type값을 format에 따라 date type값으로 변환
TO_NUMBER	주어진 string type값을 숫자형식으로 변환
TO_TIME	주어진 string을 형식에 따라 시간 값으로 변환
TO_TIMESTAMP	주어진 string을 형식에 따라 timestamp 타입 값으로 변환

12.2 시나리오 수행

시나리오 내용
1. TO_CHAR : 주어진 date type, number type의 값을 형식에 따라 문자열로 변환
2. TO_DATE : 주어진 string type값을 format에 따라 date type값으로 변환
3. TO_NUMBER : 주어진 string type값을 숫자형식으로 변환
4. TO_TIME : 주어진 string을 형식에 따라 시간 값으로 변환
5. TO_TIMESTAMP : 주어진 string을 형식에 따라 timestamp 타입 값으로 변환

12.3 시나리오 수행 결과

1. TO_CHAR : 주어진 date type, number type의 값을 형식에 따라 문자열로 변환

```
-- TO_CHAR
SQL> SELECT TO_CHAR(sysdate) FROM dual;
```

```
SQL> -- TO_CHAR
SQL> SELECT TO_CHAR(sysdate) FROM dual;

TO_CHAR(SYSDATE)
--------------------------------
2022/11/18

1 row selected.
```

2. TO_DATE : 주어진 string type값을 format에 따라 date type값으로 변환

```
-- TO_DATE
SQL> SELECT TO_DATE('18/11/2022','DD/MM/YYYY') FROM dual;
```

```
SQL> -- TO_DATE
SQL> SELECT TO_DATE('18/11/2022','DD/MM/YYYY') FROM dual;

TO_DATE('18/11/2022','DD/MM/YYYY')
---------------------------------------------------------
2022/11/18

1 row selected.
```

3. TO_NUMBER : 주어진 string type값을 숫자형식으로 변환

```
-- TO_NUMBER
SQL> SELECT TO_NUMBER('$35,000.00','$99,999.99') FROM dual;
```

```
SQL> -- TO_NUMBER
SQL> SELECT TO_NUMBER('$35,000.00','$99,999.99') FROM dual;

TO_NUMBER('$35,000.00','$99,999.99')
------------------------------------
                               35000

1 row selected.
```

4. TO_TIME : 주어진 string을 형식에 따라 시간 값으로 변환

```
-- TO_TIME
SQL> SELECT TO_TIME('18:05:31','HH24:MI:SS') FROM dual;
```

```
SQL> -- TO_TIME
SQL> SELECT TO_TIME('18:05:31','HH24:MI:SS') FROM dual;

TO_TIME('18:05:31','HH24:MI:SS')
--------------------------------
18:05:31.000000

1 row selected.
```

5. TO_TIMESTAMP : 주어진 string을 형식에 따라 timestamp 타입 값으로 변환

```
-- TO_TIMESTAMP
SQL> SELECT TO_TIMESTAMP('18:05:31','HH24:MI:SS.FF') FROM dual;
```

```
SQL> -- TO_TIMESTAMP
SQL> SELECT TO_TIMESTAMP('18:05:31','HH24:MI:SS.FF') FROM dual;

TO_TIMESTAMP('18:05:31','HH24:MI:SS.FF')
---------------------------------------------------------------
2000/01/01 18:05:31.000000000

1 row selected.
```

12.sh

```
SET ECHO ON

-- TO_CHAR
SELECT TO_CHAR(sysdate) FROM dual;

-- TO_DATE
SELECT TO_DATE('18/11/2022','DD/MM/YYYY') FROM dual;

-- TO_NUMBER
SELECT TO_NUMBER('$35,000.00','$99,999.99') FROM dual;

-- TO_TIME
SELECT TO_TIME('18:05:31','HH24:MI:SS') FROM dual;

-- TO_TIMESTAMP
SELECT TO_TIMESTAMP('18:05:31','HH24:MI:SS.FF') FROM dual;

EXIT
```