Chapter 4

Lecture (Lec 5 & 6) for Week 3

4.1 Built-in Functions

1. **LOWER and UPPER:** select City, LOWER(City), LOWER('City') from WEATHER;
2. **Concatenation** ||

```
select City||Country from LOCATION;

select CONCAT(City, Country) from LOCATION;

  2 are equivalent
```

3. **RPAD and LPAD:**

   The syntax:

```
LPAD( string1, paddedlength, [ pad_string ] )
```

   Parameters or Arguments:

   string1 is the string to pad characters to (the left-hand side).

   paddedlength is the number of characters to return. If the paddedlength is smaller than the original string, the LPAD function will *truncate the string to the size of paddedlength.*

   padstring is optional. This is the string that will be padded to the left-hand side of string1. If this parameter is *omitted*, the LPAD function will *pad spaces* to the left-side of string1.

   Example:

```
   LPAD('tech', 7);
Result: '   tech'

LPAD('tech', 2);
Result: 'te'

LPAD('tech', 8, '0');
Result: '0000tech'

LPAD('tech on the net', 15, 'z');
Result: 'tech on the net'

LPAD('tech on the net', 16, 'z');
Result: 'ztech on the net'
```

4. **LTRIM, RTRIM, and TRIM:** They trim off (removes) all specified characters from the left and right ends of strings.

   The syntax:

```
LTRIM( string1, [ trimstring ] )
```

   Parameters or Arguments:

   string1 is the string to trim the characters from the left-hand side.

   trimstring is the string that will be removed from the left-hand side of string1. If this parameter is *omitted*, the LTRIM function will remove *all leading spaces* from string1.

   Example:

```
  LTRIM('   tech')
Result: 'tech'

LTRIM('   tech', ' ')
Result: 'tech'

LTRIM('000123', '0')
Result: '123'

LTRIM('123123Tech', '123')
Result: 'Tech'
```

```
LTRIM('123123Tech123', '123')
Result: 'Tech123'

LTRIM('xyxzyyyTech', 'xyz')
Result: 'Tech'

LTRIM('6372Tech', '0123456789')
Result: 'Tech'
```

Last Example: In this example, every number combination from 0 to 9 has been listed in the trimstring parameter. By doing this, it does not matter the order that the numbers appear in string1, all leading numbers will be removed by the LTRIM function.

Combining: LTRIM and RTRIM

```
  (RTRIM('*#Hello*','*')

  LTRIM('*#Hello*','*#')

  Now combine them:

  LTRIM(RTRIM('*#Hello*','*'),'*#')
```

Using the TRIM Function: The preceding example showed how to combine two functionsa useful skill when dealing with string manipulation. *If you are trimming the exact same data from both the beginning and the end of the string, you can use the TRIM function in place of an LTRIM/RTRIM combination.*

```
  TRIM('   tech   ')
Result: 'tech'

Note: If you do not specify trim_character, the default value is
a blank space.


TRIM(' '  FROM  '   tech   ')
Result: 'tech'
```

```
TRIM(LEADING '0' FROM '000123')
Result: '123'

Note: If you specify LEADING, Oracle removes any leading characters equal
to trim_character.

TRIM(TRAILING '1' FROM 'Tech1')
Result: 'Tech'

Note: If you specify TRAILING, Oracle removes any trailing characters equal
to trim_character.


TRIM(BOTH '1' FROM '123Tech111')
Result: '23Tech'

Note: If you specify BOTH or none of the three, Oracle removes
leading and trailing characters equal to trim_character.
```

5. **LENGTH** LENGTH tells you how long a string ishow many characters it has in it, including letters, spaces, and anything else.
6. **SUBSTR:**

   The syntax:

   ```
   SUBSTR( string, start_position, [ length ] )
   ```

   Parameters or Arguments:

   string is the source string.

   startposition is the position for extraction. The first position in the string is always 1.

   length is optional. It is the number of characters to extract. If this parameter is omitted, the SUBSTR function will return the entire string.

   Note: If startposition is a positive number, then the SUBSTR function starts from the beginning of the string.

   If startposition is a negative number, then the SUBSTR function starts from the end of the string and counts backwards.

   Example:

```
  SUBSTR('This is a test', 6, 2)
Result: 'is'

SUBSTR('This is a test', 6)
Result: 'is a test'

SUBSTR('TechOnTheNet', 1, 4)
Result: 'Tech'

SUBSTR('TechOnTheNet', -3, 3)
Result: 'Net'

SUBSTR('TechOnTheNet', -6, 3)
Result: 'The'

SUBSTR('TechOnTheNet', -8, 2)
Result: 'On'
```

7. **INSTR:**

Syntax:

```
INSTR( string, substring [, start_position [,  nth_appearance ] ] )
```

Parameters or Arguments:

string is the string to search.

substring is the substring to search for in string.

startposition is the position in string where the search will start. This argument is optional. If omitted, it defaults to 1. The first position in the string is 1. If the startposition is *negative*, the INSTR function counts back startposition number of characters from the end of string and then searches towards the beginning of string.

nthappearance is the nth appearance of substring. This is optional. If omitted, it defaults to 1.

Example:

```
  INSTR('Tech on the net', 'e')
Result: 2   (the first occurrence of 'e')

INSTR('Tech on the net', 'e', 1, 1)
Result: 2   (the first occurrence of 'e')
```

```
INSTR('Tech on the net', 'e', 1, 2)
Result: 11  (the second occurrence of 'e')

INSTR('Tech on the net', 'e', 1, 3)
Result: 14  (the third occurrence of 'e')
```

8. **ASCII and CHR:**

   Example:

```
select CHR(70)||CHR(83)||CHR(79)||CHR(85)||CHR(71) R
as ChrValues
from DUAL;

OUTPUT:

R
-----
FSOUG
```

   The ASCII function performs the reverse operationbut if you pass it a string, only the first character of the string will be acted upon:

```
 select ASCII('FSOUG') from DUAL;
ASCII('FSOUG')
--------------
70
```

4.2 Regular Expression

Mainly used for advanced searching. Oracle 10g introduced support support for regular expressions in SQL and PL/SQL.

Example 1 : REGEXP_SUBSTR

```
 DROP TABLE t1;
CREATE TABLE t1 (
  data VARCHAR2(50)
);

INSERT INTO t1 VALUES ('FALL 2014');
INSERT INTO t1 VALUES ('2014 CODE-B');
INSERT INTO t1 VALUES ('CODE-A 2014 CODE-D');
```

```
INSERT INTO t1 VALUES ('ADSHLHSALK');
INSERT INTO t1 VALUES ('FALL 2004');
COMMIT;

SELECT * FROM t1;

DATA
--------------------------------------------------------------------------------
FALL 2014
2014 CODE-B
CODE-A 2014 CODE-D
ADSHLHSALK
FALL 2004

5 rows selected.

SQL>
```

Objective: If we needed to return rows containing a specific year we could use the LIKE operator (WHERE data LIKE '%2014%'), but how do we return rows using a comparison (¡, ¡=, ¿, ¿=, ¡¿)?

One Solution is: to pull out the 4 figure year and convert it to a number, so we don't accidentally do an ASCII comparison.

That is pretty easy using regular expressions.

Solution using Regular Expression: We can identify digits using the `"\d"` or `"[0-9]"` operators. We want a group of four of them, which is represented by the `"{4}"` operator. So our regular expression will be `"\d{4}"` or "[0-9]4". The `REGEXP_SUBSTR` function returns the string matching the regular expression, so that can be used to extract the text of interest. We then just need to convert it to a number and perform our comparison.

Solution Query:

```
 SELECT *
FROM   t1
WHERE  TO_NUMBER(REGEXP_SUBSTR(data, '\d{4}')) >= 2014;

DATA
----------------------------------------------------------
FALL 2014
2014 CODE-B
CODE-A 2014 CODE-D

3 rows selected.

SQL>
```