**UIT UNIVERSITY**

**Department of Engineering Technology**

**SET211 Database Systems/ CET224 Database Applications**

Lab#3

**Objective:**

* **Single-row and multiple-row functions in SQL**

.

**Name of Student: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Sec. \_\_\_\_\_\_\_\_\_\_\_**

**Date of Experiment: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Marks Obtained/Remarks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**THEORY**

**SQL functions**

Functions are a very powerful feature of SQL and can be used to do the following tasks:

* Perform calculations on data
* Modify individual data items
* Manipulate output for groups of rows
* Format dates and numbers for display
* Convert column datatypes

SQL functions may accept arguments and always return a value as illustrated in figure 4.1.

Output

Input

Function

a

r

g

1

ar

g

2

ar

g

3

Result Value

**Figure 4.1: Functions accept arguments and return a value**

**Types of SQL functions**

There are two distinct types of functions:

* Single-row
* Multiple-row

**Single-Row functions**

These functions operate on single rows only and return one result per row. There are different types of single-row functions. This session covers the following ones:

* Character
* Number
* Date
* Conversion

**Multiple-Row functions**

These functions manipulate groups of rows to give one result per group of rows.

**Single-Row Functions**

**Character Functions**

Single-row character functions accept character data as input and can return both character and number values. Character functions can be divided into the following: -

* Case conversion functions
* Character manipulation functions

Case Conversion Functions

Convert case for character strings

|  |  |
| --- | --- |
| **Function** | **Result** |
| LOWER(‘SQL Course’) | sql course |
| UPPER(‘SQL Course’) | SQL COURSE |
| INITCAP(‘SQL Course’) | Sql Course |

**Table 4.1**

**Examples**

1. To display the employee number, name (in upper case) and department number for employee *Blake*.

SELECT empno, UPPER(ename), deptno

FROM emp

WHERE LOWER(ename) = ‘blake’;

**Note**: Since the actual case of the letters in the employee name column may not be known, so it is necessary for comparison to convert the name to either uppercase or lowercase.

Character manipulation functions

Manipulate character strings

|  |  |
| --- | --- |
| **Function** | **Result** |
| CONCAT(‘Good’, ‘String’) | GoodString |
| SUBSTR(‘String’, 2, 4) | trin |
| LENGTH(‘String’) | 6 |
| INSTR(‘String’, ‘r’) | 3 |
| LPAD(sal, 10, ‘\*’) | \*\*\*\*\*\*5000 |

**Table 4.2**

**Example**

To display employee name and job joined together, length of employee name, and the numeric position of letter A in the employee name, for all employees who are in *sales*.

SELECT empno, CONCAT(ename, job), LENGTH(ename), INSTR(ename, ‘A’)

FROM emp

WHERE SUBSTR(job, 1, 5) = ‘SALES’;

**Number Functions**

Number functions accept numeric input and return numeric values.

***ROUND(column|expression, n)*** Rounds the column, expression or value to **n** decimal places or if **n** is omitted, no decimal places (If n is negative, numbers to left of decimal point are rounded)

***TRUNC(column|expression, n)*** Truncates the column, expression or value to *n* decimal places or if *n* is omitted, no decimal places (If *n* is negative, numbers to left of decimal point are truncated)

***MOD(m, n)*** Returns the remainder of *m* divided by *n*

|  |  |
| --- | --- |
| **Function** | **Result** |
| ROUND(45.927, 2) | 45.93 |
| ROUND(45.927) | 46 |
| ROUND(45.927, -1) | 50 |
| TRUNC(45.927, 2) | 45.92 |
| TRUNC(45.927) | 45 |
| MOD(20, 3) | 2 |

**Table 4.3**

**Examples**

1. SELECT ROUND(45.923, 2), ROUND(45.923, 0), ROUND(45.923, -1) FROM DUAL;

The DUAL is a dummy table with one column and one row.

1. SELECT TRUNC(45.923, 2), TRUNC(45.923), TRUNC(45.923, -1)

FROM DUAL;

1. To calculate the remainder of the ratio of salary to commission for all employees whose job title is salesman.

SELECT ename, sal, comm., MOD(sal, comm.)

FROM emp

WHERE UPPER(job) = ‘SALESMAN’;

**Date Functions**

SYSDATE is a date function that returns the current date and time. The current date can be displayed by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL.

The DUAL table is owned by the user SYS and can be accessed by all users. It contains one column, DUMMY, and one row with the value X. It is useful for returning a value once only – for instance, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data.

For example, to display the current date using the DUAL table as

SELECT SYSDATE

FROM DUAL;

**Arithmetic with Dates**

We can add or subtract a number to or from a date for a resultant date value. For example, to display the name and the number of weeks employed for all employees in department 10.

SELECT ename, (SYSDATE – HIREDATE) / 7 “Number of Weeks”

FROM emp

WHERE deptno = 10;

**SELECT ename, (SYSDATE - HIREDATE) / 7 "Numder of weeks"**

**FROM emp\_snew**

**WHERE deptno = 10;**

Date functions operate on Oracle dates. All date functions return a value of DATE datatype except MONTHS\_BETWEEN, which returns a numeric value.

|  |  |  |
| --- | --- | --- |
| **Function** | **Result** | **Description** |
| MONTHS\_BETWEEN(’01-SEP-95’, ’11-JAN-94’) | 19.6774194 | Number of months between two dates |
| ADD\_MONTHS(’11-JAN-94’, 6) | ’11-JUL-94’ | Add calendar months to dates |
| NEXT\_DAY(’01-SEP-95’, ‘FRIDAY’) | ’08-SEP-95’ | Next day of the date specified |
| LAST\_DAY(’01-SEP-95’) | ’30-SEP-95’ | Last day of the month |
| ROUND(TO\_DATE(’25-JUL-95’, ‘DD-MON-YY’), ‘MONTH’) | 01-AUG-95 | Round date |
| ROUND(TO\_DATE(’25-JUL-95’, ‘DD-MON-YY’), ‘YEAR’) | 01-JAN-96 | Round date |
| TRUNC(TO\_DATE(’25-JUL-95’, ‘DD-MON-YY’), ‘MONTH’) | 01-JUL-95 | Truncate date |
| TRUNC(TO\_DATE(’25-JUL-95’, ‘DD-MON-YY’), ‘YEAR’) | 01-JAN-95 | Truncate date |

**Table 4.4**

**Examples**

i. For all employees employed for fewer than 200 months, display the employee number, hiredate, number of months employed, six-month review date, first Friday after hiredate and last day of the month hired.

SELECT empno, hiredate, MONTHS\_BETWEEN(SYSDATE, hiredate) TENURE,

ADD\_MONTHS(hiredate, 6) REVIEW, NEXT\_DAY(hiredate, ‘FRIDAY’), LAST\_DAY(hiredate) FROM emp

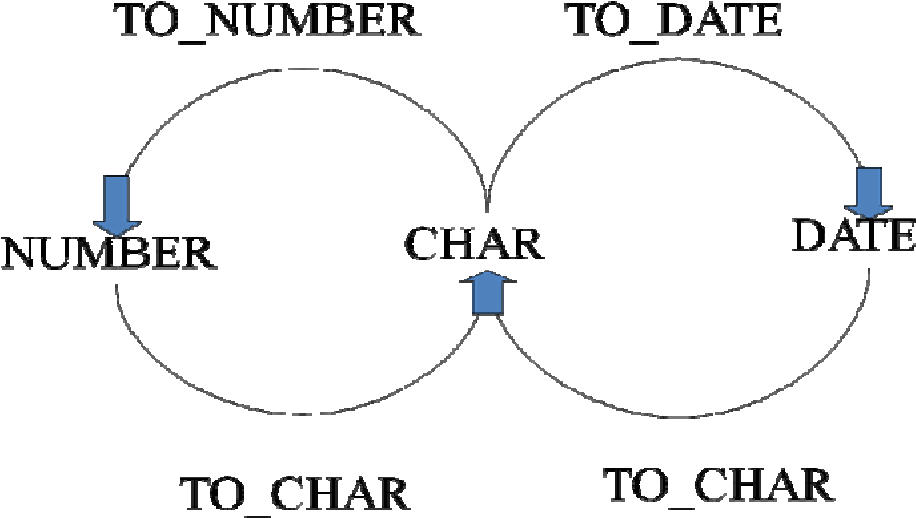
WHERE MONTHS\_BETWEEN(SYSDATE, hiredate) < 200;

ii. Comparing the hire dates for all employees who started in 1982, display the employee number, hiredate, and month started using the ROUND and TRUNC functions. SELECT empno, hiredate, ROUND(hiredate, ‘MONTH’),TRUNC(hiredate,

‘MONTH’) FROM emp

WHERE hiredate like ‘%82’;

**Conversion Functions**



**Figure 4.1**

SQL provides three functions to convert a value from one data type to another.

TO\_CHAR

TO\_NUMBER

TO\_DATE

**TO\_CHAR function with Dates**

i. To display the employee number, the month number and year of hiring

SELECT empno, TO\_CHAR(hiredate, ‘MM/YY’) Month\_Hired

FROM emp

WHERE ename = ‘BLAKE’;

**SELECT empno, TO\_CHAR(hiredate, 'MM/YY') Month\_Hired**

**FROM emp\_snew**

**WHERE ename = 'BLAKE';**

The second argument of TO\_CHAR is called *format model*, is in single quotation marks and is case sensitive. ii. To display the employee name and hiredate for all employees. The hiredate appears as 17 November, 1981.

SELECT ename, TO\_CHAR(hiredate, 'fmDD Month YYYY') HIREDATE FROM emp;

The *fm* element is used to remove padded blanks or suppress leading zeros.

iii. To print the employee name and time of joining in format HH:MI:SS (Assuming that hiredate column were used for storing joining time)

SELECT ename, TO\_CHAR(hiredate, ‘HH:MI:SS’) HIREDATE FROM emp;

**TO\_CHAR function with Numbers**

It is used to display a number value as a character string. This technique is especially useful for concatenating a numeric value to a character string.

i. To display the salary of employee SCOTT with $ sign preceded

SELECT TO\_CHAR(sal, ‘$99,999’) SALARY

FROM emp

WHERE ename = ‘SCOTT’;

The oracle server displays a string of pound signs (#) in place of a whole number whose digits exceed the number of digits provided in the format model. The oracle server rounds the stored decimal value to the number of decimal places provided in the format model.

**TO\_NUMBER function**

Convert a character string to a number format using the TO\_NUMBER function. TO\_DATE function

Converts a character string to a date format using the TO\_DATE function.

i. To display the names and hire dates of all the employees who joined on February 22, 1981

SELECT ename, hiredate

FROM emp

WHERE hiredate = TO\_DATE(‘February 22, 1981’, ‘Month dd, YYYY’);

**SELECT ename, hiredate**

**FROM emp\_snew**

**WHERE hiredate = TO\_DATE('February 22, 1981', 'Month dd, YYYY');**

**DECODE Function**

Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement. It decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes *expression* after comparing it to each *search* value. If the expression is the same as *search*, *result* is returned.

If the default value is omitted, a null value is returned where a search value does not match any of the result values.

DECODE (*col/expression*, *search1*, *result1* [,*search2*, *result2*, ….,] [, *default*]);

**Examples**

i. To print job, salary and revised salary depending on the job.

SELECT job, sal, DECODE (job, ‘ANALYST’, SAL\*1.1, ‘CLERK’, SAL\*1.15,

‘MANAGER’, SAL\*1.20, SAL) REVISED\_SALARY FROM emp;

The same statement can be written as an IF-THEN-ELSE statement:

IF job = ‘ANALYST’ THEN sal = sal \* 1.1

IF job = ‘CLERK’ THEN sal = sal \* 1.15 IF job = ‘MANAGER’ THEN sal = sal \* 1.20

ELSE sal = sal ii. To display the applicable tax rate for each employee in department 30

SELECT ename, sal, DECODE(TRUNC(sal/1000, 0),

0, 0.00,

1, 0.09,

2, 0.20,

3, 0.30,

4, 0.40,

5, 0.42,

6, 0.44,

0.45) TAX\_RATE

FROM emp

WHERE deptno = 30;

**Nesting Functions**

Single-row functions can be nested to any level. For example, the following query displays the head of a company who has no manager.

SELECT ENAME, NVL(TO\_CHAR(MGR), ‘No Manager’)

FROM EMP

WHERE MGR IS NULL;

**Multiple\_Row Functions**

Group functions operate on sets of rows to give one result per group. The following table identifies the options that can be used in the syntax:-

Following is the syntax of using group functions:- SELECT [*column*, ] *group\_function(column)*

FROM *table*

[WHERE *condition*]

[GROUP BY *column*]

[ORDER BY *column*];

**Applying multiple-row functions to all rows in a table**

**Examples**

1. To show the average salary, minimum salary, maximum salary and count of employees in the organization

SELECT AVG (SAL), MIN(SAL), MAX(SAL), COUNT(\*)

FROM EMP;

1. To show the minimum and maximum hiredate for employees

SELECT MIN (hiredate), MAX(hiredate)

FROM emp;

1. To return the number of rows in a table

SELECT COUNT(\*)

FROM emp

WHERE deptno = 30; iv. To return the number of nonnull rows in a table

SELECT COUNT(comm)

FROM emp

WHERE deptno = 30;

v. The group function like AVG do not include null rows. The NVL function forces group functions to include null values.

SELECT AVG(NVL(comm, 0))

FROM emp;

**Applying Multiple-row functions to groups of rows in a table**

**Examples**

1. To show the department-wise average salary,

SELECT deptno, AVG(sal) AVERAGE\_SALARY

FROM emp

GROUP BY deptno;

Note that all columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

1. To show the job-wise total salary for each department

SELECT deptno, job, sum(sal)

FROM emp

GROUP BY deptno, job;

**Excluding groups result**

In the same way that we use the WHERE clause to restrict the rows that we select, the HAVING clause is used to restrict groups. First the group function is applied and the groups matching the HAVING clause are displayed.

The syntax of the SELECT statement showing the HAVING clause along with the GROUP BY clause is shown below:-

SELECT *column, group\_function*

FROM *table*

[WHERE condition]

[GROUP BY group\_by\_expression]

[HAVING group\_condition]

[ORDER BY column];

The HAVING clause can precede the GROUP BY clause but it is recommended that the GROUP BY clause come first because it is more logical.

**Examples**

1. To show the department-wise average and maximum salary, in the descending order of average salary, for all departments having average salary higher than 4500.

SELECT DEPTNO, AVG(SAL), MAX(SAL)

FROM EMP

GROUP BY DEPTNO

HAVING AVG(SAL) > 2000

ORDER BY AVG(SAL)

1. To display the job title and total monthly salary for each job title with a total payroll exceeding 5000.

SELECT JOB, SUM(SAL) PAYROLL

FROM EMP

WHERE JOB NOT LIKE 'SALES%'

GROUP BY JOB

HAVING SUM(SAL) > 5000

ORDER BY SUM(SAL);

**Nesting Group Functions**

i. To display the maximum average salary by nesting group functions

SELECT max(avg(sal))

FROM emp

GROUP BY deptno;

**EXERCISES**

1. Write down the result of the function calls in the first column to the second column.

|  |  |
| --- | --- |
| **Function Call** | **Result** |
| SUBSTR(CONCAT(‘HIGH’, ‘SALARY’), 4, 6) |  |
| CONCAT(SUBSTR(‘INFORMATION’, 3, 4), ‘TECH’) |  |
| INSTR(CONCAT(‘GET’, ‘ING’), ‘TIN’) |  |
| ROUND(69.476, 1) |  |
| TRUNC(’13-MAR-90’, ‘MONTH’) |  |
| TRUNC(’13-MAR-90’, ‘YEAR’) |  |
| MOD(90, LENGTH(‘SEVENTY’)) |  |
| MONTHS\_BETWEEN(’14-AUG-96’, ’23-MAR-95’) |  |

1. Write down SQL queries to perform following functions:-
2. To show the current date. Label the column *Current* *Date*.

1. To display the employee number, name, salary, salary increase by 15% expressed as a whole number (labeled as *New Salary*), the difference between old salary and new salary (labeled as *Increment*).

1. To display the employee name and calculate the number of months between today and the date the employee was hired (Labeled as *Months\_Worked*). Order the results by the number of months employed and round the number of months up to the closest whole number.

1. Write a query that produces the following for each employee:

*<employee name>* earns *<salary>* monthly

1. To display the employee’s name (labeled *name*) with the first letter capitalized and all other letters lowercase and the length of their name (labeled *length*), for all employees whose name starts with J, A or M.

1. To list the name, hiredate, and day of the week (labeled *DAY*) on which job was started. Order the result by day of week starting with Monday.

1. To display the job-wise count of employees in each department as follows:-

**DEPTNO JOB NUM\_EMP**

1. To display the department name, location name, number of employees and the average salary for all employees in that department. Label the columns DNAME, LOC, NUMBER OF PEOPLE and SALARY, respectively. Round the average salary to two decimal places.

1. To display the employee name, department number and job title for all employees whose department location is *Dallas*.

1. To display the difference between the highest and lowest salaries (Labeled as

*DIFFERENCE*)

1. To show the manager name, MANAGER, and the number of employees, NUM, working under him.