COMP 3311
Database Management Systems

Lab 3

SQL Functions and Subqueries

Lab Objectives

- After this lab you should be able to use SQL
  - string functions.
  - number functions.
  - date functions.
  - aggregate functions.
  - subqueries.

SQL String Functions

- String functions take strings as input and output either strings or numerical values.

Function	Purpose
lower(*string*)	converts *string* to lowercase
upper(*string*)	converts *string* to uppercase
initcap(*string*)	sets first character of each word to uppercase
substr(*string*, *position*, *length*)	returns a *length* substring of *string* starting at *position*
concat(*string1*, *string2*)	concatenates *string1* and *string2*
instr(*string1*, *string2*)	returns location of *string2* in *string1*
length(*string*)	returns length of *string*
lpad(*string1*, *length*, *string2*)	pads *string1* with *string2* to the left to *length*
rpad(*string1*, *length*, *string2*)	pads *string1* with *string2* to the right to *length*
ltrim(*string*)	removes all spaces from the left of *string*
rtrim(*string*)	removes all spaces from the right of *string*

SQL String Functions (1)

- lower(*string*) – converts all the characters in *string* to lowercase.

```
select lower(lastName)
from Student;
```

- upper(*string*) – converts all the characters in *string* to uppercase.

```
select upper(lastName)
from Student;
```

- initcap(*string*) – sets the first character of each word in *string* to uppercase.

```
select initcap(courseName)
from Course;
```

SQL String Functions (2)

- substr(*string*, *position*, *length*) – returns a particular portion of *string* starting at *position* and of size *length*.

```
select substr(firstName, 2, 3)
from Student;
```

- concat(*string1*, *string2*) – concatenates *string1* and *string2*.
  Note: || can concatenate more than two strings.

```
select concat(lastName, firstName)
from Student;
```

- instr(*string1*, *string2*) – returns the location of *string2* in *string1*.

```
select instr(lastName, 'ea')
from Student;
```

SQL String Functions (3)

- length(*string*) – returns the length of *string*.

```
select length(lastName)
from Student;
```

- lpad(*string1*, *length*, *string2*) – pads *string1* to the left with *string2* so that the new string's length is equal to *length*.

```
select lpad('a', 10, 'b')
from dual;*
```

- rpad(*string1*, *length*, string2) – pads *string1* to the right with *string2* so that the new string's length is equal to *length*.

```
select rpad('a', 10, 'b')
from dual;*
```

* The table dual is an Oracle built-in relation for SQL queries that do not logically have table names.

SQL String Functions (4)

- ltrim(*string*) – removes all the spaces from the left of *string*.

```
select ltrim('    a  ')
from dual;
```

returns 'a '

- rtrim(*string*) – removes all the spaces from the right of *string*.

```
select rtrim('    a  ')
from dual;
```

returns '   a'

SQL Numeric Functions

- Numeric functions accept numeric inputs and output numeric values.

Function	Purpose
mod(*number1*, *number2*)	returns *number1* mod *number2*
power(*number1*, *number2*)	returns (*number1*)^{number2}
round(*number1*, *integer_number2*)	returns *number1* rounded to *integer_number2* places
trunc(*number1*, *integer_number2*)	truncates *number1* to *integer_number2* decimal places

SQL Date Functions

- The Oracle default date format is 'DD-MON-YY'. March 7, 2018 is therefore '07-MAR-18'.

Function	Purpose
add_months(*date*, *number*)	adds *number* of months to *date*
next_day(*date*, *weekday*)	returns the date of the first *weekday* that is later than *date*
last_day(*date*)	returns the date of the last day in the month of *date*
current_date	returns the current date
to_date(*string*, *date_format_string*)	convert *string* to the corresponding date according to *date_format_string*
to_char(*date*, *format_mask*)	convert *date* to a string according to *format_mask*

SQL Date Functions (1)

- add_months(*date*, *number*) – adds *number* of months to *date*.

```
select add_months('07-MAR-18', 2)
from dual;
```

```
ADD_MONTHS
--------------------
07-MAY-18
```

- next_day(*date*, *weekday*) – returns the *date* of the first *weekday* that is later than *date*.

The possible values for weekday are: 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'

```
select next_day('05-OCT-18', 'Saturday')
from dual;
```

```
NEXT_DAY
---------------
06-OCT-18
```

- last_day(*date*) – returns the date of *date*'s month's last day.

```
select last_day('07-MAR-18')
from dual;
```

```
LAST_DAY
---------------
31-MAR-18
```

SQL Date Functions (2)

- current_date – returns the current date.

```
select current_date
from dual;
```

```
CURRENT_DATE
-----------
27-SEP-18
```

- to_date(*string*, *date_format_string*) – converts *string* to the corresponding Oracle date format according to *date_format_string*.

```
select to_date('2018-03-23', 'yyyy-mm-dd')
from dual;
```

```
TO_DATE
---------
23-MAR-18
```

- to_char(*date*, *format_mask*) – converts *date* to a string according to *format_mask*. The format masks can be:
  - 'yyyy' : 4-digit year
  - 'mm' : 2-digit month
  - 'month' : 'January', 'February', etc.

SQL Aggregate Functions

- An aggregate function performs a calculation on a collection of input data and returns a single value for the data.

Function	Purpose
avg (*attribute_name*)	returns the average value
count(*attribute_name*)	returns the number of records
max(*attribute_name*)	returns the maximum value
min(*attribute_name*)	returns the minimum value
stddev(*attribute_name*)	returns the standard deviation
sum(*attribute_name*)	returns the total

ALL aggregate functions (except for **count**(*)) ignore NULL values (i.e., they do not include them in the calculation).

SQL Aggregate Function Examples (1)

- avg(*attribute_name*) – returns the average value in the *attribute_name* column.

```
select avg(cga)
from Student;
```

- count(*attribute_name*) – returns the number of records according to the *attribute_name* column.

```
select count(cga)
from Student;
```

- max(*attribute_name*) – returns the maximum value in the *attribute_name* column.

```
select max(cga)
from Student;
```

SQL Aggregate Function Examples (2)

- min(*attribute_name*) – returns the minimum value for the values in the *attribute_name* column.

```
select min(cga)
from Student;
```

- stddev(*attribute_name*) – returns the sample standard deviation for the values in the *attribute_name* column.

```
select stddev(cga)
from Student;
```

- sum(*attribute_name*) – returns the total of the values in the *attribute_name* column.

```
select sum(cga)
from Student;
```

GROUP BY And HAVING Clause

- **GROUP BY** groups data by one or more attributes, so that aggregate functions can be applied.
- The **HAVING** clause is applied to the groups formed by the **GROUP BY** clause and specifies the condition(s) for including a group in the results.
- A **WHERE** clause, if present, filters the records before groups are formed; the groups are then further filtered by the **HAVING** clause.

GROUP BY Clause

- The GROUP BY clause groups the data by one or more attributes, so that aggregate functions can be applied.

Query: Find the number of students in each department.

```
select departmentId, count(*)
from Student
group by departmentId;
```

Notes:

1. The non-aggregation attributes in the SELECT clause must be a subset of the attributes in the GROUP BY clause.
2. Oracle does not allow a column alias to be used in the GROUP BY clause (i.e., you cannot change lastName to ln in the GROUP BY clause).

GROUP BY With HAVING Clause (1)

- The HAVING clause is applied to the groups formed by the GROUP BY clause to specify the condition(s) under which the group should be included in the results.

  Query: Find the departments whose maximum cga is greater than 3.5.

```
select departmentId, max(cga)
from Student
group by departmentId
having max(cga)>3.5;
```

GROUP BY With HAVING Clause (2)

- An SQL statement can also contain the WHERE clause to specify the condition(s) for selecting the records; these records are then filtered by the condition(s) specified by the HAVING clause.

  Query: For the COMP and ELEC departments, determine whether their maximum cga is greater than 2.5 or less than 1.5.

```
select departmentId, max(cga)
from Student
where departmentId='COMP' or departmentId='ELEC'
group by departmentId
having max(cga)>2.5 or max(cga)<1.5;
```

Subqueries

- A subquery in the **WHERE** clause works as part of the row selection process.
- Use a subquery in the **WHERE** or **HAVING** clause when the criteria depends on the results from another table.

Example Subqueries (1)

Student(studentId, firstName, lastName, cga, departmentId)

Query: Find students whose CGA equals the minimum CGA.

```
select firstName, lastName, cga
from Student
where cga=(select min(cga)
           from Student);
```

Query: Find departments and their average CGA where the average department CGA is greater than the average CGA of all Students.

```
select departmentId, trunc(avg(cga), 2) as "avgCGA"
from Student
group by departmentId
having avg(cga)>(select avg(cga)
                 from Student);
```

Example Subqueries (2)

Student(studentId, firstName, lastName, cga, departmentId)

- The same query as the second query on the previous slide, but this query utilizes two temporary tables to store the result of the two subqueries.

```
select DeptAvgCga.departmentId, trunc(DeptAvgCga.avgCga, 2) as "avgCGA"
from (select departmentId, avg(cga) as avgcga
      from Student
      group by departmentId ) DeptAvgCga
where DeptAvgCga.avgcga>(select OverallAvgCga.overallAvgCga
                         from (select avg(cga) as overallAvgCga
                               from Student ) OverallAvgCga);
```

DeptAvgCga contains the average CGA of *each* department.

OverallAvgCga contains the average CGA of *all* students.

Summary

- We covered the following SQL topics in this lab:
  - string functions.
  - number functions.
  - date functions.
  - aggregate functions.
  - subqueries.

Lab Exercise

- You must complete the lab exercise and upload the result to Canvas by **11:59 p.m. on Friday**.

Ask for help if you need it!

IMPORTANT NOTE

If you want to save your modified SQL script files, copy them to the M drive or to a USB drive as any personal files on the lab computers will be automatically deleted periodically.