COMP 3311
Database Management Systems

Lab 4

SQL Functions and Subqueries

Lab Topics

- SQL functions
  - string
  - numeric
  - date
  - aggregate
- GROUP BY and HAVING clauses
- Subqueries

To see the result of the example queries in these lab notes, execute them in SQL Developer against the database created by the Lab4DB.sql script file.

SQL Functions

String
lower(*string*)
upper(*string*)
initcap(*string*)
substr(*string*, *position*, *length*)
concat(*string1*, *string2*)
instr(*string1*, *string2*)
length(*string*)
lpad(*string1*, *length*, *string2*)
rpad(*string1*, *length*, *string2*)
ltrim(*string*)
rtrim(*string*)

Numeric
mod(*number1*, *number2*)
power(*number1*, *number2*)
round(*number1*, *integer_number2*)
trunc(*number1*, *integer_number2*)
Date
add_months(*date*, *number*)
next_day(*date*, *weekday*)
last_day(*date*)
current_date
to_date(*string*, *date_format_string*)
to_char(*date*, *format_mask*)

Aggregate
avg(*attribute_name*)
count(*attribute_name*)
max(*attribute_name*)
min(*attribute_name*)
stddev(*attribute_name*)
sum(*attribute_name*)

SQL String Functions (1)

- 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 (2)

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

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

- upper(*string*) – converts *string* to all 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 (3)

- 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, ' ')
from Student;
```

SQL String Functions (4)

- 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;
```

SQL String Functions (5)

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

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

Query result: 'a '

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

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

Query result: '   a'

Note: Since attribute values of type char are always padded with trailing spaces to the length of the character string by Oracle Database, the rtrim function can be used to remove these trailing spaces. This is needed when comparing attribute values of type char to a regular expression.

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 (1)

- Date functions either return specific dates or convert strings to dates or dates to strings.

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*

The default date format is 'DD-MON-YY' (i.e., March 7, 2020 is '07-MAR-20').

SQL Date Functions (2)

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

```
select add_months('07-MAR-20', 2)
from dual;
```

```
ADD_MONTHS
---------
07-MAY-20
```

- 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-20', 'Saturday')
from dual;
```

```
NEXT_DAY
---------
10-OCT-20
```

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

```
select last_day('07-MAR-20')
from dual;
```

```
LAST_DAY
---------
31-MAR-20
```

SQL Date Functions (3)

- current_date – returns the current date.

```
select current_date
from dual;
```

```
CURRENT_DATE
-----------
27-SEP-20
```

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

```
select to_date('2020-03-23', 'yyyy-mm-dd')
from dual;
```

```
TO_DATE
---------
23-MAR-20
```

- 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

- Aggregate functions perform a calculation on a collection of input data and return 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 Clause

- The **GROUP BY** clause groups the data by one or more attributes, so that aggregate functions (e.g., count, sum, etc.) can be applied.

Query: Find the number of students in each department.

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

DEPARTMENTID	COUNT(*)
BUS	4
COMP	9
ELEC	4
MATH	4

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 rename departmentId to id in the select clause and then use id in the GROUP BY clause).

GROUP BY With HAVING Clause

- 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 maximum cga of each department.

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

DEPARTMENTID	MAX(CGA)
BUS	3.42
COMP	3.64
ELEC	3.37
MATH	3.56

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;
```

DEPARTMENTID	MAX(CGA)
COMP	3.64
MATH	3.56

GROUP BY With HAVING And WHERE Clause

- A WHERE clause, if present, filters the records before groups are formed; the groups are then further filtered by the HAVING clause.

Query: For the COMP and ELEC departments, determine whether their maximum cga is greater than 3.5 or less than 1.5.

```
select departmentId, max(cga)
from Student
where departmentId='COMP' or departmentId= 'ELEC'
group by departmentId
having max(cga)>3.5 or max(cga)<1.5;
```

DEPARTMENTID	MAX(CGA)
COMP	3.64

Subqueries

- A subquery in the WHERE clause works as part of the row selection process.
- Use a subquery in a 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);
```

The minimum cga of *all* students.

FIRSTNAME	LASTNAME	CGA
Donald	Trump	1.49

Example Subqueries (2)

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

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);
```

The average cga of *all* students.

DEPARTMENTID	avgCGA
COMP	3.01
MATH	3.11

Example Subqueries (3)

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.

DEPARTMENTID	avgCGA
COMP	3.01
MATH	3.11

DEPARTMENTID	AVGCGA
BUS	2.45
COMP	3.01
ELEC	2.7225
MATH	3.1175

OVERALLAVGCGA
2.862