

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, run them in [SQL Developer](#) against the database created by the Lab4DB.sql script file.

SQL Functions

String
<code>lower(<i>string</i>)</code>
<code>upper(<i>string</i>)</code>
<code>initcap(<i>string</i>)</code>
<code>substr(<i>string</i>, <i>position</i>, <i>length</i>)</code>
<code>concat(<i>string1</i>, <i>string2</i>)</code>
<code>instr(<i>string1</i>, <i>string2</i>)</code>
<code>length(<i>string</i>)</code>
<code>lpad(<i>string1</i>, <i>length</i>, <i>string2</i>)</code>
<code>rpadd(<i>string1</i>, <i>length</i>, <i>string2</i>)</code>
<code>ltrim(<i>string</i>)</code>
<code>rtrim(<i>string</i>)</code>

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

Aggregate
<code>avg(<i>attribute_name</i>)</code>
<code>count(<i>attribute_name</i>)</code>
<code>max(<i>attribute_name</i>)</code>
<code>min(<i>attribute_name</i>)</code>
<code>stddev(<i>attribute_name</i>)</code>
<code>sum(<i>attribute_name</i>)</code>

SQL String Functions (1)

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

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

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;
```

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

```
select rpadd('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'

SQL Numeric Functions

- ❑ Numeric functions accept numeric inputs and output numeric values.

Function	Purpose
<code>mod(<i>number1</i>, <i>number2</i>)</code>	returns <i>number1</i> mod <i>number2</i>
<code>power(<i>number1</i>, <i>number2</i>)</code>	returns (<i>number1</i>) ^{<i>number2</i>}
<code>round(<i>number1</i>, <i>integer_number2</i>)</code>	returns <i>number1</i> rounded to <i>integer_number2</i> places
<code>trunc(<i>number1</i>, <i>integer_number2</i>)</code>	truncates <i>number1</i> to <i>integer_number2</i> decimal places

SQL Date Functions (1)

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

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

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*.

<code>select add_months('07-MAR-20', 2)</code>	ADD_MONTHS
<code>from dual;</code>	-----
	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'

<code>select next_day('05-OCT-20', 'Saturday')</code>	NEXT_DAY
<code>from dual;</code>	-----
	10-OCT-20

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

<code>select last_day('07-MAR-20')</code>	LAST_DAY
<code>from dual;</code>	-----
	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
<code>avg(attribute_name)</code>	returns the average value
<code>count(attribute_name)</code>	returns the number of records
<code>max(attribute_name)</code>	returns the maximum value
<code>min(attribute_name)</code>	returns the minimum value
<code>stddev(attribute_name)</code>	returns the standard deviation
<code>sum(attribute_name)</code>	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 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 rename `lastName` to `ln` 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 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 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 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 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);
```

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.