Agenda for IS475/675 Lab Class: 02/10/2015

- **Work together in class**
  - Present format of SQL statements.
  - Discuss order of SQL SELECT statement execution
  - Present a few sample SQL SELECT statements

- **Work individually**
  - Go through this handout typing and exploring SQL SELECT statements
  - Handout is for your information – nothing will be turned in from this lab class
  - Ask questions/get answers about this handout or HW#3

Start of Class

1) Login to the COBA lab computer

2) Login to SQL Server Management Studio

3) Open a file located on the k: drive\IS475\LabFiles\create-emp1-s15.sql. If you can't see the k: drive, the file is available on WebCampus in the weekly handouts for today's class date.

4) Execute the file called 'create-emp1-s15.sql'. The file will first try and drop a table called 'emp1', then it will create the table and populate it with 15 rows of data. You should first see an error message, and then the table should build correctly. To validate whether the table has been built, look at it in Object Explorer.

General Syntax for accessing data from a single table

```
SELECT      [all or distinct] (what columns)
FROM        (table or tables)
WHERE       (condition for each row)
GROUP BY    (grouping fields)
HAVING      (condition for the group)
ORDER BY    (sort fields)
```

Referred to as the 'SELECT LIST"

Order of Actual Execution of a SQL SELECT Statement:

1) FROM
2) WHERE
3) GROUP BY
4) HAVING
5) SELECT
6) ORDER BY

When a SELECT statement is executed, the result is referred to as a "**result table**". It is a memory-based table; it can't be stored on disk from a standard SQL SELECT statement. Normally, the SELECT statement is called from another program, executed by the DBMS, and then the result table is passed back to that other program.

Syntax to access all data about all employees from an employee table called 'emp1'

```
SELECT      *
FROM        emp1;
```

	EmpNo	Ename	Job	Manager	HireDate	Phone	Salary	Commission	DeptNo
1	7839	KING, MARGARET	President	NULL	2013-11-17 00:00:00.000	7757845611	9000.00	NULL	10
2	8698	NG, JUNE	Clerk	7839	2015-01-15 00:00:00.000	7754562501	3120.00	NULL	10
3	7782	CLARK, ROBERT	MANAGER	7839	1999-06-09 00:00:00.000	7759810021	2450.00	3230.00	10
4	7566	JONES, MARTIN	MANAGER	7839	2006-04-02 00:00:00.000	8056719332	2975.00	NULL	20
5	7654	MARTIN, WILLIAM	SALESMAN	7698	2014-01-28 00:00:00.000	8586712300	5250.00	3400.00	30
6	7499	ALLEN, BERTRAM	SALESMAN	7698	2014-02-20 00:00:00.000	8586723441	3600.00	3300.00	30
7	7844	TURNER, ELIZABETH	SALESMAN	7698	2015-02-08 00:00:00.000	7754519002	6000.00	7500.00	33
8	7900	JAMES, KATHERINE	CLERK	7698	2013-12-03 00:00:00.000	7875623456	2950.00	NULL	30
9	7902	WONG, BRADFORD	ANALYST	7566	2014-09-07 00:00:00.000	9098337788	3500.00	NULL	20
10	7521	WARD, ROBERT	SALESMAN	7698	2012-02-22 00:00:00.000	8056671223	3250.00	5000.00	30
11	9015	JOHNSON, JAMES	SALESMAN	7698	2007-12-15 00:00:00.000	8058912334	2900.00	5000.00	30
12	8015	MARTINEZ, CONSUELO	ANALYST	7566	2013-12-15 00:00:00.000	8058938924	3900.00	5000.00	20
13	6743	QUESTA, MARIA	SALESMAN	7566	2012-05-10 00:00:00.000	7758891111	3824.00	1200.00	30
14	6011	MARQUEZ, JOTEN	CLERK	7698	2008-02-05 00:00:00.000	7756611255	2150.00	NULL	10
15	7788	CHENG, SUN-LIN	ANALYST	7566	2013-09-23 00:00:00.000	7756772990	3900.00	NULL	20

Guidelines for Writing SQL

- SQL statements start with a command, and then include few or many modifiers/extensions for the command.
- SQL statements are not case sensitive.
- The data stored in a data base ARE case sensitive.
- SQL statements can span more than one physical line; it is a free form language.
- SQL keywords cannot be abbreviated or split across lines.
- Keywords and/or main clauses are typically placed on separate lines.
- Tabs and indentation are used to enhance readability.
- Keywords are typically aligned in the first column.
- Keywords are usually capitalized.
- Data are usually in lowercase or a combination of uppercase and lowercase.
- Comments are included sparingly, but usefully. Comments can be included by a double dash in front of a line if you want to comment out a whole line, or with a /* in the area where you wish to place a comment with a */ to end the commented area.

Syntax to access the name, salary, hiredate, and department number for all employees

```
SELECT     ename, salary, hiredate, deptno
FROM       emp1;
```

Syntax to access the name, salary, hiredate, and department number and give them a columnar alias

```
SELECT    ename      AS  'employee name',
          salary     AS  'Current Salary',
          hiredate   AS  'Date Hired',
          deptno     AS  'department Number'
FROM      emp1;
```

Syntax to do the same as above, but sort the data by department number, and remove the optional AS clause

```
SELECT    ename      'employee name',
          salary     'Current Salary',
          hiredate   'Date Hired',
          deptno     'department Number'
FROM      emp1
ORDER BY   deptno;
```

Output from the above SQL statement – this is the 'standard' date format. Dates are not stored as they are displayed, so when a field is declared as a datetime data type, you can display it in many different formats.

Syntax to show the use of a function in the SELECT list

```
SELECT      LOWER(ename)      'employee name',
            salary            'Current Salary',
            hiredate          'Date Hired',
            deptno            'department Number'
FROM       emp1
ORDER BY    deptno;
```

Syntax to access the Name, Salary, Commission and calculate total remuneration

```
SELECT      ename                   'Employee Name',
            Salary,
             Commission,
            Salary + Commission 'Total Remuneration'
FROM        emp1
ORDER BY    ename;
```

Here is the syntax to handle null values in the SELECT list

```
SELECT      ename                         'Employee Name',
            Salary,
            ISNULL(Commission,0)          'Commission',
            Salary + ISNULL(Commission, 0) 'Total Remuneration'
FROM        emp1
ORDER BY ename;
```

Syntax to demonstrate calculations

```
SELECT      ename             'Employee Name',
            Salary,
            Commission,
            Salary + isnull(Commission, 0) 'Current remuneration',
            (Salary + isnull(Commission, 0)) * 1.20 'New remuneration'
FROM        emp1
ORDER BY    ename;
```

Syntax to demonstrate calculations and data type conversion

```
SELECT ename           'Employee Name',
       Salary,
        Commission,
       Salary + isnull(Commission, 0) 'Current remuneration',
       CAST(((Salary + isnull(Commission, 0)) * 1.20) as MONEY)
                                  'New remuneration'
FROM        emp1
ORDER BY    ename;
```

Syntax to view only some of the rows in the table, based on a condition

```
SELECT    ename     'employee name',
          salary    'Current Salary',
          hiredate  'Date Hired',
          deptno    'department Number'
FROM      emp1
WHERE     deptno = 20
ORDER BY   ename;
```

Syntax to expand the condition to see other rows in the table. The two statements below accomplish produce the same results.

```
SELECT    ename     'employee name',
          salary    'Current Salary',
          hiredate  'Date Hired',
          deptno    'department Number'
FROM      emp1
WHERE     deptno = 20 or deptno = 30
ORDER BY   ename;
```

```
SELECT    ename     'employee name',
          salary    'Current Salary',
          hiredate  'Date Hired',
          deptno    'department Number'
FROM      emp1
WHERE     deptno in (20, 30)
ORDER BY   ename;
```

Now it is time to work on your own!! The rest of this document asks you to type certain commands and see how they work. The goal is to become familiar with some of the functions and syntax of the SQL SELECT statement. Feel free to ask questions – raise your hand, and I will help you.

Syntaxes for other date displays – try them and look at the results

```
SELECT    ename      'employee name',
          salary     'Current Salary',
          hiredate   'Date Hired',
          deptno     'department Number'
FROM      emp1
ORDER BY   deptno;


SELECT     ename     'Employee Name',
           Salary,
           Deptno    'Department Number',
           CAST(Hiredate AS VARCHAR)     'Date Hired'
FROM       emp1
ORDER BY   ename;


SELECT     ename     'Employee Name',
           Salary
           Deptno    'Department Number',
           CONVERT(VARCHAR, Hiredate, 107)'Date Hired'
FROM       emp1

ORDER BY   ename;



SELECT     ename     'Employee Name',
           Salary
           Deptno    'Department Number',
           CONVERT(VARCHAR, Hiredate, 101)'Date Hired'
FROM       emp1

ORDER BY   ename;
```

Codes for SQL CONVERT with dates see pg. 255 of the class SQL text

Doing calculations with dates using the DATEDIFF function and getting the current date via GETDATE() function

```
SELECT   empno          'Employee Number',
         ename          'Employee Name',
         hiredate       'Date Hired',
         DATEDIFF(day, hiredate, getdate())
                        'Number of Days Employed',
         DATEDIFF(month, hiredate, getdate())
                        'Number of Months Employed',
         DATEADD(day, 90, hiredate)
                        'Date 90 days After Hire Date'
FROM     emp1
ORDER BY empno;
```

Here is the result table from the query above (this result table was produced on 2/9/2015, so your results will be slightly different):

	Employee Number	Employee Name	Date Hired	Number of Days Employed	Number of Months Employed	Date 90 days After Hire Date
1	6011	MARQUEZ, JOTEN	2008-02-05 00:00:00.000	2561	84	2008-05-05 00:00:00.000
2	6743	QUESTA, MARIA	2012-05-10 00:00:00.000	1005	33	2012-08-08 00:00:00.000
3	7499	ALLEN, BERTRAM	2014-02-20 00:00:00.000	354	12	2014-05-21 00:00:00.000
4	7521	WARD, ROBERT	2012-02-22 00:00:00.000	1083	36	2012-05-22 00:00:00.000
5	7566	JONES, MARTIN	2006-04-02 00:00:00.000	3235	106	2006-07-01 00:00:00.000
6	7654	MARTIN, WILLIAM	2014-01-28 00:00:00.000	377	13	2014-04-28 00:00:00.000
7	7782	CLARK, ROBERT	1999-06-09 00:00:00.000	5724	188	1999-09-07 00:00:00.000
8	7788	CHENG, SUN-LIN	2013-09-23 00:00:00.000	504	17	2013-12-22 00:00:00.000
9	7839	KING, MARGARET	2013-11-17 00:00:00.000	449	15	2014-02-15 00:00:00.000
10	7844	TURNER, ELIZABETH	2015-02-08 00:00:00.000	1	0	2015-05-09 00:00:00.000
11	7900	JAMES, KATHERINE	2013-12-03 00:00:00.000	433	14	2014-03-03 00:00:00.000
12	7902	WONG, BRADFORD	2014-09-07 00:00:00.000	155	5	2014-12-06 00:00:00.000
13	8015	MARTINEZ, CONSUELO	2013-12-15 00:00:00.000	421	14	2014-03-15 00:00:00.000
14	8698	NG, JUNE	2015-01-15 00:00:00.000	25	1	2015-04-15 00:00:00.000
15	9015	JOHNSON, JAMES	2007-12-15 00:00:00.000	2613	86	2008-03-14 00:00:00.000

Date functions are on pgs. 272-283 of SQL text. Date calculations are critical for business applications – I recommend that you familiarize yourself with the available functions.

Syntax to show how to concatenate the name, Salary, and department number and give the entire results an alias

```
SELECT    ename + ' earns ' +
          CAST(Salary as VARCHAR) +
          ' in the number ' +
          deptno + ' department' 'employee information'
FROM      emp1;
```

Output from the above SQL statement

employee information
KING, MARGARET earns 9000.00 in the number 10 department
NG, JUNE earns 3120.00 in the number 10 department
CLARK, ROBERT earns 2450.00 in the number 10 department
JONES, MARTIN earns 2975.00 in the number 20 department
MARTIN, WILLIAM earns 5250.00 in the number 30 department
ALLEN, BERTRAM earns 3600.00 in the number 30 department
TURNER, ELIZABETH earns 6000.00 in the number 33 department
JAMES, KATHERINE earns 2950.00 in the number 30 department
WONG, BRADFORD earns 3500.00 in the number 20 department
WARD, ROBERT earns 3250.00 in the number 30 department
JOHNSON, JAMES earns 2900.00 in the number 30 department
MARTINEZ, CONSUELO earns 3900.00 in the number 20 department
QUESTA, MARIA earns 3824.00 in the number 30 department
MARQUEZ, JOTEN earns 2150.00 in the number 10 department
CHENG, SUN-LIN earns 3900.00 in the number 20 department

Learning how to parse data – separating the first and last names into separate columns. The goal is to create the result table as shown below.

	Employee Last Name	Employee First Name
1	KING	MARGARET
2	NG	JUNE
3	CLARK	ROBERT
4	JONES	MARTIN
5	MARTIN	WILLIAM
6	ALLEN	BERTRAM
7	TURNER	ELIZABETH
8	JAMES	KATHERINE
9	WONG	BRADFORD
10	WARD	ROBERT
11	JOHNSON	JAMES
12	MARTINEZ	CONSUELO
13	QUESTA	MARIA
14	MARQUEZ	JOTEN
15	CHENG	SUN-LIN

The substring function is used to 'sub-divide' a string data type (char or varchar) attribute. The syntax is: substring(nameofattribute, position to start, number of characters to display). SQL is a 1-based language, so there is no zero position. Type the following to see how it works:

```
SELECT SUBSTRING(ename,1,6) 'Employee Last Name'
FROM   emp1;
```

That command will display the first six characters of the field ename. You should see this result table:

	Employee Last Name
1	KING,
2	NG, JU
3	CLARK,
4	JONES,
5	MARTIN
6	ALLEN,
7	TURNER
8	JAMES,
9	WONG,
10	WARD,
11	JOHNSO
12	MARTIN
13	QUESTA
14	MARQUE
15	CHENG,

To figure out how many characters the comma is located from the first position in ename requires the use of the CHARINDEX function. Type this statement in to understand how CHARINDEX works:

```
SELECT CHARINDEX(',', ename)
FROM EMP1
```

You should see this result table because CHARINDEX gives the character location (starting at position one) of whatever you are looking for in a particular string/character field.

	(No column name)
1	5
2	3
3	6
4	6
5	7
6	6
7	7
8	6
9	5
10	5
11	8
12	9
13	7
14	8
15	6

Now combine the two functions, SUBSTRING and CHARINDEX to locate all the characters in ename up to and including the comma:

```
SELECT SUBSTRING(ename,1,CHARINDEX(',' , ename))
      'Employee Last Name'
FROM    emp1;
```

	Employee Last Name
1	KING,
2	NG,
3	CLARK,
4	JONES,
5	MARTIN,
6	ALLEN,
7	TURNER,
8	JAMES,
9	WONG,
10	WARD,
11	JOHNSON,
12	MARTINEZ,
13	QUESTA,
14	MARQUEZ,
15	CHENG,

To get rid of the comma requires that you subtract 1 from the result of the CHARINDEX function:

```
SELECT SUBSTRING(ename,1,CHARINDEX(',' , ename)-1)
      'Employee Last Name'
FROM   emp1;
```

	Employee Last Name
1	KING
2	NG
3	CLARK
4	JONES
5	MARTIN
6	ALLEN
7	TURNER
8	JAMES
9	WONG
10	WARD
11	JOHNSON
12	MARTINEZ
13	QUESTA
14	MARQUEZ
15	CHENG

*** Give this a try: Try and figure out how you would get the first name to appear in a separate column using very similar syntax so that you could produce the result table shown here:

	Employee Last Name	Employee First Name
1	KING	MARGARET
2	NG	JUNE
3	CLARK	ROBERT
4	JONES	MARTIN
5	MARTIN	WILLIAM
6	ALLEN	BERTRAM
7	TURNER	ELIZABETH
8	JAMES	KATHERINE
9	WONG	BRADFORD
10	WARD	ROBERT
11	JOHNSON	JAMES
12	MARTINEZ	CONSUELO
13	QUESTA	MARIA
14	MARQUEZ	JOTEN
15	CHENG	SUN-LIN

*** Now that you know how to concatenate data (page 8 of this handout) and how to use SUBSTRING, give this a try: Write a query that would format the telephone number so that it will display in the following structure: (775)784-5611. I'd like you to give this task a try with the SUBSTRING function and concatenate your results into a single column.

This same task can also be accomplished with the SQL Server 2012 FORMAT function, but that function isn't described in your book. You are welcome to look it up on the web: https://msdn.microsoft.com/en-us/library/ee634924.aspx

If you sort the output by ename, here is the result table:

	Employee Name	Phone Number
1	ALLEN, BERTRAM	(858) 672-3441
2	CHENG, SUN-LIN	(775) 677-2990
3	CLARK, ROBERT	(775) 981-0021
4	JAMES, KATHERINE	(787) 562-3456
5	JOHNSON, JAMES	(805) 891-2334
6	JONES, MARTIN	(805) 671-9332
7	KING, MARGARET	(775) 784-5611
8	MARQUEZ, JOTEN	(775) 661-1255
9	MARTIN, WILLIAM	(858) 671-2300
10	MARTINEZ, CONSUELO	(805) 893-8924
11	NG, JUNE	(775) 456-2501
12	QUESTA, MARIA	(775) 889-1111
13	TURNER, ELIZABETH	(775) 451-9002
14	WARD, ROBERT	(805) 667-1223
15	WONG, BRADFORD	(909) 833-7788

Learning about the CASE Statement

The CASE statement evaluates a condition, or a list of conditions, and returns a value. For example, let's say you want to create a new column called 'SalaryRating'. You want the column to say 'Big Salary' if a person's salary is greater than 3000 and 'Little Salary' if a person's salary is less than or equal to 3000.

```
SELECT      ename,
            salary,
            CASE
                  WHEN salary > 3000
                  THEN 'Big Salary'
                  ELSE 'Little Salary'
                  END   SalaryRating
FROM        emp1
```

	ename	salary	SalaryRating
1	KING, MARGARET	9000.00	Big Salary
2	NG, JUNE	3120.00	Big Salary
3	CLARK, ROBERT	2450.00	Little Salary
4	JONES, MARTIN	2975.00	Little Salary
5	MARTIN, WILLIAM	5250.00	Big Salary
6	ALLEN, BERTRAM	3600.00	Big Salary
7	TURNER, ELIZABETH	6000.00	Big Salary
8	JAMES, KATHERINE	2950.00	Little Salary
9	WONG, BRADFORD	3500.00	Big Salary
10	WARD, ROBERT	3250.00	Big Salary
11	JOHNSON, JAMES	2900.00	Little Salary
12	MARTINEZ, CONSUELO	3900.00	Big Salary
13	QUESTA, MARIA	3824.00	Big Salary
14	MARQUEZ, JOTEN	2150.00	Little Salary
15	CHENG, SUN-LIN	3900.00	Big Salary

A CASE statement can have more than one WHEN clause. If you want to try it – enhance the query on the prior page so that 'SalaryRating' has four possibilities. A salary greater than 5000 should be 'Really Big Salary', between 3500 to 5000 should be 'Big Salary,' between 2000 to 3499.99 should be 'Mediocre Salary,' and less than 2000 should be 'Pittance Pay'. Here is the result table – you write the query:

	ename	salary	SalaryRating
1	KING, MARGARET	9000.00	Really Big Salary
2	NG, JUNE	3120.00	Mediocre Salary
3	CLARK, ROBERT	2450.00	Mediocre Salary
4	JONES, MARTIN	2975.00	Mediocre Salary
5	MARTIN, WILLIAM	5250.00	Really Big Salary
6	ALLEN, BERTRAM	3600.00	Big Salary
7	TURNER, ELIZABETH	6000.00	Really Big Salary
8	JAMES, KATHERINE	2950.00	Mediocre Salary
9	WONG, BRADFORD	3500.00	Big Salary
10	WARD, ROBERT	3250.00	Mediocre Salary
11	JOHNSON, JAMES	2900.00	Mediocre Salary
12	MARTINEZ, CONSUELO	3900.00	Big Salary
13	QUESTA, MARIA	3824.00	Big Salary
14	MARQUEZ, JOTEN	2150.00	Mediocre Salary
15	CHENG, SUN-LIN	3900.00	Big Salary

Back to just typing examples…

Syntax to access only the unique department numbers in the employee table

```
SELECT DISTINCT   deptno
FROM              emp1;
```

Syntax to access only the unique Salaries in the employee table

```
SELECT DISTINCT   Salary
FROM              emp1;
```

Most of the prior queries display all the rows in the underlying table (emp1). Now it is time to use the WHERE clause to limit the number of rows displayed on the output. FYI - the 'criteria' on the design grid in MS Access generates a SQL 'WHERE' clause.

Syntax to access the name, Salary, and department number for all employees with a Salary greater than 2000

```
SELECT      ename, Salary, deptno
FROM        emp1
WHERE       Salary > 2000;
```

Syntax to access the name, Salary, and department number for all employees with a Salary greater than 2000 and a dept number of 10

```
SELECT      ename, Salary, deptno
FROM        emp1
WHERE       Salary > 2000
AND         deptno = 10;
```

Options to access a range: These two queries below accomplish the same result table:

	ename	Salary	deptno
1	MARTIN, WILLIAM	5250.00	30
2	ALLEN, BERTRAM	3600.00	30
3	WONG, BRADFORD	3500.00	20

```
SELECT      ename, Salary, deptno
FROM        emp1
WHERE       hiredate BETWEEN ('01-jan-2014') AND
                  ('31-dec-2014');
```

```
SELECT      ename, Salary, deptno
FROM        emp1
WHERE       hiredate >= ('01-jan-2014') AND
            hiredate <= ('31-dec-2014');
```

Options to access a value from a group: These two queries below accomplish the same goal

```
SELECT      ename, Salary, deptno
FROM        emp1
WHERE       deptno IN (10, 20);

SELECT      ename, Salary, deptno
FROM        emp1
WHERE       deptno = 10 OR deptno = 20;
```

Using the current date in a conditional statement: These two queries accomplish the same result table:

	ename	Salary	hiredate
1	NG, JUNE	3120.00	2015-01-15 00:00:00.000
2	TURNER, ELIZABETH	6000.00	2015-02-08 00:00:00.000

```
SELECT      ename,
            Salary,
            hiredate
FROM        emp1
WHERE       DATEPART(yyyy, hiredate) =
            DATEPART(yyyy, GETDATE());

OR

SELECT      ename,
            Salary,
            hiredate
FROM        emp1
WHERE       year(hiredate) =
            year(GETDATE())
```

Check for both the current year and current month:

```
SELECT      ename,
            Salary,
            hiredate
FROM        emp1
WHERE       YEAR(hiredate) =  YEAR(GETDATE())
AND         MONTH(GETDATE()) = MONTH(hiredate)
```

Use the current date in a WHERE clause with a calculation. What if you want to look at all the employees who were hired two years ago? Here is the result table when running the query in 2015, so you write the query.

	ename	Salary	hiredate
1	KING, MARGARET	9000.00	2013-11-17 00:00:00.000
2	JAMES, KATHERINE	2950.00	2013-12-03 00:00:00.000
3	MARTINEZ, CONSUELO	3900.00	2013-12-15 00:00:00.000
4	CHENG, SUN-LIN	3900.00	2013-09-23 00:00:00.000

You are allowed only one WHERE clause in a SQL SELECT, but you can use as many AND's and OR's with that WHERE clause as you want.

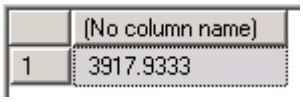
Finally, here is the syntax for wildcard searches in the WHERE clause

```
SELECT      ename, Salary, deptno
FROM        emp1
WHERE       ename LIKE 'SM%';


SELECT      ename, Salary, deptno
FROM        emp1
WHERE       ename LIKE '_M%';


SELECT      ename, Salary, deptno
FROM        emp1
WHERE       SOUNDEX(ename) = SOUNDEX('marten')
```

Learning about Aggregate Functions

An 'aggregate function' is a way to summarize data and provide more meaningful and informative output from the database. Aggregate queries are also referred to as 'summary queries'. Aggregate queries are intended to produce fewer rows than non-aggregate queries. In MS Access, you click on the 'totals' button and choose functions from a lovely drop down box. In SQL, you have to write the query…

Calculate the mean average of all salaries:

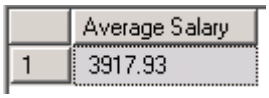


	(No column name)
1	3917.9333

```
SELECT      AVG(salary)
FROM        emp1;
```

Note that when you use a function, there is no column name unless you declare an alias for the column.

Round the result and add a column alias:



	Average Salary
1	3917.93

```
SELECT      ROUND(AVG(salary),2) 'Average Salary'
FROM        emp1;
```

Calculate the rounded average for only one department:

```
SELECT      ROUND(AVG(salary),2)
FROM        emp1
WHERE       deptno = 10;
```

Count rows:

```
SELECT      COUNT(*)
FROM        emp1;


SELECT      COUNT(*)
FROM        emp1
WHERE       deptno = 10;

SELECT      COUNT(*)
FROM        emp1
WHERE       salary > 2000 and deptno = 10;


SELECT      COUNT(DISTINCT deptno)
FROM        emp1;
```

Find the minimum and maximum values:

```
SELECT      MIN(hiredate)
FROM        emp1;


SELECT      MAX(hiredate)
FROM        emp1;

SELECT      MIN(ename)
FROM        emp1;


SELECT      MAX(hiredate)
FROM        emp1
WHERE       deptno = 10;
```

Combine aggregate functions into a single line result table:

```
SELECT      COUNT(salary) CountSalaryRows,
            SUM(salary) TotalSalary,
            MIN(salary) SmallestSalary
FROM        emp1
WHERE       deptno = 10 and salary < 4000;
```

Result table from above statement:

	CountSalaryRows	TotalSalary	SmallestSalary
1	3	7720.00	2150.00

Use aggregate functions in calculations and with other functions:

```
SELECT      MAX(salary + ISNULL(commission,0))
FROM        emp1;

SELECT      MAX(DATEDIFF(mm, hiredate, GETDATE()))
FROM        emp1;
```