

A TUTORIAL ON ORACLE/SQL*PLUS

by Junping Sun

Graduate School of Computer and Information Sciences
Nova Southeastern University
3301 College Avenue
Fort Lauderdale, Florida 33314 USA
E-mail: jps@nova.edu

©2004

What is ORACLE?

There are several major components as referred to the ORACLE DBMS software package in the following:

- ORACLE DBMS Server
The major DBMS engine.
- ORACLE SQL*PLUS
The user-friendly interactive SQL (structural query language) interface to access databases.
- ORACLE PL/SQL
Combination of procedural language and structural query language.
- ORACLE SQL*FORMS
The graphic front-end application development tool.
- ORACLE SQL*REPORTWRITER
Report generator.
- ORACLE SQL*NET
Oracle client/server database connect tool.
- ORACLE Open Client Adapter for ODBC (Open Database Connectivity)
Heterogeneous client/server database connect tool.

In this document, we will concentrate on the ORACLE SQL*PLUS because it is not only the fundamental component in the ORACLE DBMS environment, but also a user-friendly interactive interface from which you can start to learn about ORACLE DBMS.

Starting and Stopping SQL*PLUS:

This document describes how to use SQL*PLUS from the beginning. You will be able to follow the examples in SQL*PLUS given here and to observe the results of executions.

The computer on which the ORACLE DBMS resides is called `delphi` at Nova Southeastern University.

There are three steps to access ORACLE8 SQL*PLUS at NSU:

1. Login on `scis` unix:

After you login on `scis` successfully, the `scis` will prompt you as the following:

```
usercode@scis>
```

where the `usercode` is your login usercode and `scis` is the name of the `scis` computer.

2. Then type the following command to login on ORACLE8 SQL*PLUS at the prompt of `scis`:

```
usercode@scis> oracle8
```

As soon as you type in the above command line, the ORACLE DBMS will prompt you to input your ORACLE usercode/password as follows:

```
+-----+
|                                     |
|           Welcome to Nova Southeastern University           |
|           Office of Information and Technology (OIT)         |
|           Oracle Database Server                           |
|           Running Oracle Enterprise Edition 8.1.5.1.0       |
|           Please login with syntax:  username/password@delphi8 |
|                                     |
+-----+
```

Executing SQL*PLUS...

SQL*Plus: Release 8.1.5.0.0 - Production on Mon Sep 23 11:40:38 2002

(c) Copyright 1999 Oracle Corporation. All rights reserved.

Enter user-name:

When you see the prompt: Enter user-name:

enter your **usercode** on scis unix, and the password **mmis6301** as the follows:

usercode/password@delphi8

- a.) Please note both the usercode and the password should be typed at the same line separated by the slash character '/
- b.) The password is followed by the connecting string **@delphi8** that will connect to the server **delphi** where ORACLE DBMS engine resides.

Connected to:

```
Oracle8i Enterprise Edition Release 8.1.5.1.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.1.0 - Production
```

SQL>

When it prompts SQL>, the system is ready for you to enter either a SQL statement or a SQL*PLUS command.

A SQL statement is the statement that complies with the syntax rule of SQL. A list of SQL statements such as create, delete, insert, select, and update will be frequently used.

The SQL*PLUS commands are the auxiliary set of commands that may be only recognized by ORACLE SQL*PLUS. Frequently used commands in ORACLE SQL*PLUS interface are clear, describe, edit, get, help, list, save, spool, and run. These SQL*PLUS commands will be explained later.

To quit and terminate the SQL*PLUS session, you could use either the `exit` or the `quit` SQL*PLUS command after the prompt of `SQL>`.

In the next, we illustrate both how to use SQL statements to access a database and how to use the auxiliary set of interface commands (SQL*PLUS commands). Please note the difference between SQL statements and SQL*PLUS commands.

To Retrieve Data from a Database Table:

To print out the data from a database table, for example, the `customer` table owned by the user `jps2`, simply type the following statement at the prompt `SQL>`.

```
select      *      from      jps2.customer;
```

The prefix `jps2` is the owner of the table `customer`. The owner of a database table is the user who creates the database table and has all the privileges such as updating the table and granting other users the access to the table. In order for anyone other than the owner to access the table, the owner of the table should grant the access to the table.

After you type the statement followed by either the `return` or `enter` key, the system will automatically execute the SQL statement and display the result from the execution on your screen.

In the above statement, there are two clauses as follows:

The select clause: `select *`

The from clause: `from jps2.customer;`

The `select` clause will include all the columns from the `jps2.customer` table in the result table, and the `from` clause indicates the `jps2.customer` table from which the data records or the tuples will be retrieved. Here, the symbol `'*'` in the `select` clause implies all the column names in the `jps2.customer` table, i.e., the values of all the columns in the table will be retrieved and displayed as the result from the execution of the `select` SQL statement.

In order to improve the readability, you could enter the `select` statement in several lines such that each line contains one clause as follows:

```
SQL> select      *
      2  from      jps2.customer;
```

First, after the prompt `SQL>`, you can type the `select` clause, `select *`, followed by either the `return` or `enter` key at the end of the current line. The system will prompt the line number, 2, with the cursor at the beginning of line 2.

Second, you can enter the `from` clause, `from jps2.customer`, in line 2. Also please note that there is a semicolon at the end of the `from` clause.

- The semicolon at the end of the second clause means the end of the whole `select` statement.
- After a SQL statement is entered, the entered SQL statement will be kept in a specific place called SQL*PLUS buffer or memory buffer, but the SQL*PLUS command will not. There is also a corresponding file named `afiedt.buf` that is used to save the current content in the buffer.
- From the point view of SQL syntax rules, you need to put a semicolon at the end of each SQL statement to indicate the end of a SQL statement, especially, there are multiple SQL statements for the execution in the buffer. In some situation, the semicolon of the last SQL statement can be omitted. This also applies to the situation where only one SQL statement is executed at one time.

If you key in the statement correctly, the system will display the data in the `customer` table owned by the user `jps2`.

CUS	LAST	FIRST	STREET	CITY	ST	ZIP	BALANCE	CREDIT	SL
124	Adams	Sally	481 Oak	Lansing	MI	49224	818.75	1000	03
256	Samuels	Ann	215 Pete	Grant	MI	49129	21.5	1500	06
311	Charles	Don	48 College	Ira	MI	49034	825.75	1000	12
315	Daniels	Tom	914 Cherry	Kent	MI	48391	770.75	750	06
405	Williams	Al	519 Watson	Grant	MI	49219	402.75	1500	12
412	Adams	Sally	16 Elm	Lansing	MI	49224	1817.5	2000	03
522	Nelson	Mary	108 Pine	Ada	MI	49441	98.75	1500	12
567	Dinh	Tran	808 Ridge	Harper	MI	48421	402.4	750	06
587	Galvez	Mara	512 Pine	Ada	MI	49441	114.6	1000	06
622	Martin	Dan	419 Chip	Grant	MI	49219	1045.75	1000	03

10 rows selected.

What is printed out on your screen is the data in the `jps2.customer` table owned by the user `jps2`.

If you do not succeed in the execution, then there might be some typos when you enter the SQL statement. At this time, it is better to use the command `clear buffer` to remove the old statement in the SQL*PLUS buffer before you key in another new statement. Later on, we will discuss how to use editor to correct the errors in the SQL*PLUS buffer.

To clear the SQL*PLUS buffer, type the command `clear` after the prompt `SQL>`.

```
SQL> clear buffer
```

It is also a good practice to clear the SQL*PLUS buffer before you start entering another new SQL statement.

Check the Relations or Tables in a Database:

There are two types of relations or tables in a database. One is the set of tables created by yourself, and another is the set of tables created by some other users (owners) as we just see. In order to access the tables created by others, you need to be granted the access privilege by either the DBA (database administrator) or the original user (the owner of these table) who creates these tables. Next, we will learn how to find out not only a list of tables created by yourself, but also a list of tables that are created by other users and granted for you to access.

To display a list of tables created by yourself, type the following line after the prompt SQL>:

```
SQL> select      *
      2  from      tab
      3  where     tabtype = 'TABLE';
```

no rows selected

If you key in everything correctly, the system will display the message such that "no rows selected," i.e., there is nothing retrieved from the execution of the select statement. That is true since you have not created any table yet. Please note that the constant string 'TABLE' is different from the constant 'table' in the where clause because the string constant enclosed within both the left single quote and the right single quote are case-sensitive. Since the internal representation of upper case letters in computer systems is different from that of the lower case letters.

Although you have not created any table, you may have been granted the access to other tables created by some other users. To find it out if you are granted the access to the tables created by some other user, for example, jps2, type the following select statement:

```
SQL> select      object_name, object_type
      2  from      all_objects
      3  where     owner = 'JPS2' and object_type = 'TABLE';
```

OBJECT_NAME	OBJECT_TYPE
CUSTOMER	TABLE
ORDERS	TABLE
ORDER_LINE	TABLE
PART	TABLE
SALES_REP	TABLE

Please note that the constant string JPS2 is capitalized in the where clause,

```
where owner = 'JPS2'
```

You need to type the following command after the prompt SQL> in order to find out the list of attributes (column names) and their corresponding types in the table jps2.customer owned by jps2,

```
SQL> describe jps2.customer;
```

Name	Null?	Type
CUST_NO		CHAR(3)
LAST		CHAR(8)
FIRST		CHAR(6)
STREET		CHAR(12)
CITY		CHAR(9)
STATE		CHAR(2)
ZIP		CHAR(5)
BALANCE		NUMBER(7,2)
CREDIT		NUMBER(5)
SLSREP_NO		CHAR(2)

The describe command displays not only the names of all the columns but also the corresponding data types of all the columns in a table.

Please try the following commands one by one after the prompt SQL>,

```
describe jps2.customer;
```

```
describe jps2.sales_rep;
```

```
describe jps2.orders;
```

```
describe jps2.order_line;
```

```
describe jps2.part;
```

After you finish the above commands, type in the following SQL statements one by one to print out the data in each table.

```
select *  
from jps2.customer;
```

```
select *  
from jps2.sales_rep;
```

```
select *  
from jps2.orders;
```

```
select *  
from jps2.order_line;
```

```
select *  
from jps2.part;
```

Up to now we have learned how to print out both the description of a table structure (relation schema) and the data in a table (relation). We will learn more about the SQL*PLUS commands before we learn more about the SQL (Structural Query Language).

The Proprietary SQL*PLUS commands:

In general, the set of SQL*PLUS commands is independent of the ANSI-SQL standard, i.e., it is the set of proprietary commands in ORACLE SQL*PLUS interface. There is no guarantee that the set of proprietary commands in ORACLE SQL*PLUS will be supported by other relational DBMS products.

All the SQL*PLUS commands **must** be issued after the SQL> prompt instead of the line number prompt. Some of the SQL*PLUS commands require the argument(s) and some of them do not.

If the current prompt is the line number instead of the prompt SQL>, just hit either the `return` or `enter` key to switch from the line number prompt to the SQL> prompt.

The following is the list of frequently used SQL*PLUS commands.

Clear, describe, edit, get, help, list, run, save, spool, and start.

clear buffer

The `clear` command removes all the contents in the SQL*PLUS buffer and reset the buffer. Whenever you encounter a syntax error during SQL*PLUS session, it is a good practice to clear the buffer before entering a new SQL statement.

describe tablename

The `describe` command displays all the attribute names and the attribute types in the table **tablename**.

Example:

```
describe jps2.part;
```

It will display all the attribute names and attribute types in the table `jps2.part`.

edit

The `edit` command will invoke the default system editor. If you find the errors in a SQL statement in an interactive session, you could use the `edit` command to correct these errors in the SQL statement. Before exiting the `edit` session, you need to save the changes that you make during the editing session.

The current default editor in our system is `pico`. You could redefine the default editor by using the `define_editor` command.

To redefine the default system editor as the `vi` editor, just type the followings after the prompt `SQL>`,

```
define_editor = "vi"
```

get filename

The `get` command will load the file **filename** under your current working directory to the SQL*PLUS buffer. After the file is loaded successfully by `get` command, the content in the file will be listed and displayed on the screen.

help

The `help` command will display the on-line help menu in the SQL*PLUS. The on-line help menu is self-contained and very easy to read.

list

The `list` command will display on the screen the current contents in the SQL*PLUS buffer.

run

The `run` command will `list` the current contents in the SQL*PLUS buffer and execute it.

There are three ways to execute a SQL statement in the SQL*PLUS buffer:

1. Put the semicolon at the end of a SQL statement.

The current SQL statement in the SQL*PLUS buffer will be automatically executed after you put the semicolon at the end of the SQL statement with following either `return` or `enter` key. This will apply to the situation when you just type in a new SQL statement and want to execute it immediately.

Example:

```
SQL> select  *
      2  from    jps2.customer;
```

2. If you do not put the semicolon at the end of a SQL statement and the current cursor is at the beginning of the next line below the SQL statement, put the forward slash `/`. This will apply to the situation where there is no semicolon at the end of a SQL statement and the cursor has been moved to the next line below the SQL statement.

Example:

```
SQL> select  *
      2  from    jps2.customer
      3  /
```

3. You can use either `run` or forward slash `/` command to execute the SQL statement in the SQL*PLUS buffer after you finish editing the SQL statement with the default system editor.

- The `run` command must be issued after the prompt `SQL>`, i.e., you must switch from the line number prompt to the `SQL>` prompt by hitting either the `return` or `enter` key before issuing the `run` command to execute a SQL statement in the buffer. Please see the following example:

```
SQL> select      *
  2  from        jps2.customer
  3  <enter key> or <return key>
SQL> run
```

- The forward slash `/` command is different from the `run` command in that the forward slash `/` will not list the current contents in the SQL*PLUS buffer before the execution.
- The forward slash `/` command can be used after either the line number prompt or the `SQL>` prompt.

save filename

The **save** command will save the current contents of the SQL*PLUS buffer into the file **filename** in your current working directory.

save filename replace

The **save** command with the keyword **replace** at the end of the command line will overwrite the current content in the file **filename** with the current content in the buffer.

When you save a SQL statement in the SQL*PLUS buffer into a file, it is a good practice to save the file with the extension `.sql`. Otherwise, the SQL*PLUS will not be able to load the saved file later by using the SQL*PLUS commands such as `start` and `get`.

spool

The **spool** command can be used to save a SQL*PLUS session into a file.

To start the spool session, simple type **spool filename.lst** after the prompt `SQL>`.

To end the spool session, simple type **spool off** after the prompt `SQL>`.

Any session information such as both SQL statement entered from the keyboard and its execution result between the command line **spool filename.lst** and the command line **spool off** will be saved in the file **filename.lst**. If the file extension `.lst` is omitted, the system will automatically append it for you.

Example:

```
SQL> spool log.lst

SQL> select      *
SQL> from        jps2.customer;

SQL> spool off
```

In this example, the SQL statement and its execution result will be spooled or saved into the file with the name `log.lst`.

start filename.sql

The **start** command can be used to load one or more SQL statements stored in a file under your current working directory and to execute the SQL statement(s) immediately after loading. You should put a semicolon at the end of each SQL statement except the last SQL statement in the sequence if there are several SQL statements in the loaded file. The name of the file **filename** which contains SQL statement(s) should have the extension `.sql`.

Operations in the Relational Database Model:

In the traditional relational database model, there are eight operations such as select, project, join (theta-join), union, intersect, minus, product, and division. There is no commercial relational DBMS product that directly supports the division operation. The indirect implementation method in the SQL can support the division operation defined in the relational database model. The ORACLE DBMS supports outer join, and transitive closure operations besides select, project, join (theta-join), union, intersect, minus, and product operations. Both outer join and transitive closure operations have found many applications in data processing.

SELECT Operation:

The select operation in terms of the relational database model is a unary operation to retrieve records or tuples in a table based on some select condition. The select operation is used to retrieve one or more records (tuples) from a database table and the select condition(s) can be specified in the where clause of a SQL select statement.

The following SQL statement is to find the names of all the customers who have a credit limit of at least \$1500.

```
SQL> select c.last, c.first
       2  from   jps2.customer c
       3  where  c.credit >= 1500;
```

```
LAST      FIRST
```

```
-----  
Samuels Ann  
Williams Al  
Adams Sally  
Nelson Mary
```

Please note 'SQL>' in the above is the prompt of SQL*PLUS. The 'c' is the aliasing of the table `jps2.customer`. The advantage of using aliasing 'c' is that you do not have to type the whole string 'jps2.customer' whenever you make reference to the table 'jps2.customer.' The aliasing 'c' can be used to replace 'jps2.customer' whenever the table 'jps2.customer' is referenced.

The result table from querying `jps2.customer` table with the select condition `c.credit >= 1500` lists all the customers who have a credit limit of at least \$1500. The symbol '>=' is the comparison operator which means greater than and equal to, and the comparison expression `c.credit >= 1500` specifies the select condition for the query. This select statement is a typical SQL query statement which involves relational operations such as select and project. The select operation is specified in the where clause and the project operation is specified in the select clause. Please note the semantics of the select clause in the SQL is different from that of the select operation defined in the relational model.

Comparison Operators:

The following list contains the comparison operators available in ORACLE SQL*PLUS:

```
=          (equal)  
!=         (not equal)  
<>        (not equal)  
>         (greater than)  
>=        (greater than or equal)  
<         (less than)  
<=        (less than or equal)
```

The above list of comparison operators can be used for any comparison expression in a where clause. The comparison expression can be used to specify either the select condition for the select operation or the join condition for the join operation in terms of relational database operations. If the comparison expression contains one item that is a column name in a table and one constant of the same type as the column name, then the comparison expression specifies a select condition for a query. If the comparison expression contains the items such as column names on the both sides of the comparison operator, then the comparison expression defines a join condition. The join operation will be discussed later.

Logical Operators:

There are three logical operators that can be used to connect multiple comparison expressions in a where clause.

and
or
not

The following SQL statement uses the logical operator and, and will give the order number of those orders placed by customer 124 on September 5, 1998.

```
SQL> select o.order_no
      2  from   jps2.orders o
      3  where  o.cust_no = 124
      4  and   o.order_date = '9/05/98'
```

```
ORDER_NO
-----
12500
```

PROJECT operation:

The project operation in the relational database model is to retrieve some of column data in a table but not necessary all of the columns. The two above SQL statements are the examples of the PROJECT operation since these SQL statements only retrieve some of the columns values from the tables. The select clause in a SQL statement defines the PROJECT operation.

The following SQL statement is an example of PROJECT operation.

```
SQL> select c.cust_no, c.last, c.first, c.balance
      2  from   jps2.customer c
```

CUS	LAST	FIRST	BALANCE
124	Adams	Sally	818.75
256	Samuels	Ann	21.5
311	Charles	Don	825.75
315	Daniels	Tom	770.75
405	Williams	Al	402.75
412	Adams	Sally	1817.5
522	Nelson	Mary	98.75
567	Dinh	Tran	402.4
587	Galvez	Mara	114.6
622	Martin	Dan	1045.75

The above query statement in SQL retrieves the customer numbers, customer names, and their balances from the customer table.

JOIN operation:

The join operation in the relational database model is used to combine the related data from two or more relational tables. For example, the query such as: For each order placed on September 5, 1998, list the order number, order date, customer number, and customer name. The query involves the data items from two tables: orders and customer. The order number and order date items are from the table orders, and the customer number and customer name items are from the table customer. In order to combine these data items from two different tables, the join operation will be needed to link the corresponding data items from these two tables. The join column or the link between the table orders and the table customer is the column customer number (cust_no). The column customer number (cust_no) in the customer is the primary key for the table customer. The column customer number (cust_no) in the table orders is a foreign key.

```
SQL> select      o.order_no, o.order_date, o.cust_no,
 2              c.last, c.first
 3 from          jps2.orders o, jps2.customer c
 4 where         o.cust_no = c.cust_no
 5 and          o.order_date = '9/05/98'
```

ORDER_NO	ORDER_DATE	CUST_NO	LAST	FIRST
12500	9/05/98	124	Adams	Sally
12498	9/05/98	522	Nelson	Mary
12504	9/05/98	522	Nelson	Mary

The condition defined in the where clause such as o.cust_no = c.cust_no is the join condition, and o.order_date = '9/05/98' is the select condition.

The join operation is inevitable in many relational database queries since it is the implementation of 1:1, 1:M, and M:N relationships.

UNION

The union of two tables is a table containing every row that is either in the first table or in the second table. For example, to retrieve the names from either customer table (customer) or sales representative table (sales_rep).

The following query statement retrieves the last and first names from the sales representative table (sales_rep).

```
SQL> select s.last, s.first
 2 from      jps2.sales_rep s
```

LAST	FIRST
Jones	Mary

Smith William
Diaz Miguel

The following query statement retrieves the last and first names from the customer table (customer).

```
SQL> select c.last, c.first  
2 from jps2.customer c
```

LAST	FIRST
Adams	Sally
Samuels	Ann
Charles	Don
Daniels	Tom
Williams	Al
Adams	Sally
Nelson	Mary
Dinh	Tran
Galvez	Mara
Martin	Dan

The following query statement is corresponding to the union of query results from two above query statements.

```
SQL> select s.last, s.first  
2 from jps2.sales_rep s  
3 union  
4 select c.last, c.first  
5 from jps2.customer c
```

LAST	FIRST
Adams	Sally
Charles	Don
Daniels	Tom
Diaz	Miguel
Dinh	Tran
Galvez	Mara
Jones	Mary
Martin	Dan
Nelson	Mary
Samuels	Ann
Smith	William
Williams	Al

The result table from the above union statement is a list of last and first names which appear in both customer and sales_rep tables.

INTERSECT

The INTERSECT operation retrieves some records with certain common properties. The intersection of two tables is a table containing very rows which appear in both two tables. For example, to retrieve the first names which appear in both sales representative table (`sales_rep`) and customer table (`customer`).

The following query retrieves the first names from the `customer` table.

```
SQL> select c.first
      2  from   jps2.customer c
```

```
FIRST
-----
Sally
Ann
Don
Tom
Al
Sally
Mary
Tran
Mara
Dan
```

The following query retrieves the first names from the sales representative (`sales_rep`) table.

```
SQL> select s.first
      2  from   jps2.sales_rep s
```

```
FIRST
-----
Mary
William
Miguel
```

The following query statement in SQL retrieves the first name(s) which appear in both sale representative table (`sales_rep`) and customer table (`customer`).

```
SQL> select s.first
      2  from   jps2.sales_rep s
      3  intersect
      4  select c.first
      5  from   jps2.customer c
```

```
FIRST
-----
Mary
```

Since the first name 'Mary' is the only one which appears in both `sales_rep` and `customer` table, the result table from the above query statement contains only one item 'Mary.'

MINUS (Difference)

The difference (minus) of two tables is a table containing every row that is in the first table but not in the second table. For example, to retrieve the first names which appear only in the sales representative table (`sales_rep`) but not in the customer table (`customer`).

```
SQL> select s.first
 2  from   jps2.sales_rep s
 3  minus
 4  select c.first
 5  from   jps2.customer c
```

```
FIRST
-----
Miguel
William
```

The first names such as 'Miguel' and 'William' are the ones which appear only in the `sales_rep` table, but not in the `customer` table.

The following query statement retrieves the first names which appear only in the customer table (`customer`) but not in the sales representative table (`sales_rep`).

```
SQL> select c.first
 2  from   jps2.customer c
 3  minus
 4  select s.first
 5  from   jps2.sales_rep s
```

```
FIRST
-----
Al
Ann
Dan
Don
Mara
Sally
Tom
Tran
```

The above result is a list of first names which only appear in the `customer` table, but not in the `sales_rep` table. From these examples, it is hard to find the difference (minus) operation is not commutative.

PRODUCT Operation:

The product of two tables is a table containing the rows which are all the combination of the rows from two tables. The result table from the product operation contains all the columns from both tables.

The following query statement generates the result of product of orders and order_line tables.

```
SQL> select *  
      2 from jps2.orders, jps2.order_line
```

ORDER_NO	ORDER_DATE	CUST_NO	ORDER_NO	PART_NO	NUMBER_ORDERED	QUOTED_PRICE
12489	9/02/98	124	12489	AX12	11	21.95
12491	9/02/98	311	12489	AX12	11	21.95
12494	9/04/98	315	12489	AX12	11	21.95
12495	9/04/98	256	12489	AX12	11	21.95
12498	9/05/98	522	12489	AX12	11	21.95
12500	9/05/98	124	12489	AX12	11	21.95
12504	9/05/98	522	12489	AX12	11	21.95
12489	9/02/98	124	12491	BT04	1	149.99
12491	9/02/98	311	12491	BT04	1	149.99
12494	9/04/98	315	12491	BT04	1	149.99
12495	9/04/98	256	12491	BT04	1	149.99
12498	9/05/98	522	12491	BT04	1	149.99
12500	9/05/98	124	12491	BT04	1	149.99
12504	9/05/98	522	12491	BT04	1	149.99
12489	9/02/98	124	12491	BZ66	1	399.99
12491	9/02/98	311	12491	BZ66	1	399.99
12494	9/04/98	315	12491	BZ66	1	399.99
12495	9/04/98	256	12491	BZ66	1	399.99
12498	9/05/98	522	12491	BZ66	1	399.99
12500	9/05/98	124	12491	BZ66	1	399.99
12504	9/05/98	522	12491	BZ66	1	399.99
12489	9/02/98	124	12494	CB03	4	279.99
12491	9/02/98	311	12494	CB03	4	279.99
12494	9/04/98	315	12494	CB03	4	279.99
12495	9/04/98	256	12494	CB03	4	279.99
12498	9/05/98	522	12494	CB03	4	279.99
12500	9/05/98	124	12494	CB03	4	279.99
12504	9/05/98	522	12494	CB03	4	279.99
12489	9/02/98	124	12495	CX11	2	22.95
12491	9/02/98	311	12495	CX11	2	22.95
12494	9/04/98	315	12495	CX11	2	22.95
12495	9/04/98	256	12495	CX11	2	22.95
12498	9/05/98	522	12495	CX11	2	22.95
12500	9/05/98	124	12495	CX11	2	22.95
12504	9/05/98	522	12495	CX11	2	22.95
12489	9/02/98	124	12498	AZ52	2	12.95
12491	9/02/98	311	12498	AZ52	2	12.95
12494	9/04/98	315	12498	AZ52	2	12.95
12495	9/04/98	256	12498	AZ52	2	12.95
12498	9/05/98	522	12498	AZ52	2	12.95
12500	9/05/98	124	12498	AZ52	2	12.95
12504	9/05/98	522	12498	AZ52	2	12.95
12489	9/02/98	124	12498	BA74	4	24.95
12491	9/02/98	311	12498	BA74	4	24.95

12494	9/04/98	315	12498	BA74	4	24.95
12495	9/04/98	256	12498	BA74	4	24.95
12498	9/05/98	522	12498	BA74	4	24.95
12500	9/05/98	124	12498	BA74	4	24.95
12504	9/05/98	522	12498	BA74	4	24.95
12489	9/02/98	124	12500	BT04	1	149.99
12491	9/02/98	311	12500	BT04	1	149.99
12494	9/04/98	315	12500	BT04	1	149.99
12495	9/04/98	256	12500	BT04	1	149.99
12498	9/05/98	522	12500	BT04	1	149.99
12500	9/05/98	124	12500	BT04	1	149.99
12504	9/05/98	522	12500	BT04	1	149.99
12489	9/02/98	124	12504	CZ81	2	325.99
12491	9/02/98	311	12504	CZ81	2	325.99
12494	9/04/98	315	12504	CZ81	2	325.99
12495	9/04/98	256	12504	CZ81	2	325.99
12498	9/05/98	522	12504	CZ81	2	325.99
12500	9/05/98	124	12504	CZ81	2	325.99
12504	9/05/98	522	12504	CZ81	2	325.99

Aggregation Functions:

The SQL provides a list of aggregation functions such as `count()`, `avg()`, `sum()`, `min()`, `max()`, etc. The `count()` function computes the number of rows in a table or the number of items in a column. The functions such as `avg()` and `sum()` compute the average value and the total value of a numerical column in a table respectively. The functions such `max()` and `min()` compute the maximal value and the minimal value of a column in a table respectively.

For example, the following query retrieves the number of customers, the average, total, maximal, and minimal balance of all the customers in the customer table.

```
SQL> select      count(*), avg(balance), sum(balance),
                max(balance), min(balance)
  2 from        jps2.customer;
```

```
COUNT(*)      AVG(BALANCE)  SUM(BALANCE)  MAX(BALANCE)  MIN(BALANCE)
-----
      10          631.85      6318.5        1817.5         21.5
```

Group By:

The group by clause in SQL select statement allows to group data in a column or more in particular order, then calculate the statistics by using aggregation functions if desired.

For example, to retrieve the total quotation for each order from the `order_line` table since each order may have more than one order line.

```
SQL> select l.order_no, sum(l.quoted_price)
  2 from    jps2.order_line l
```

```
3 group by l.order_no
```

ORDER_NO	SUM(L.QUOTED_PRICE)
12489	21.95
12491	549.98
12494	279.99
12495	22.95
12498	37.9
12500	149.99
12504	325.99

The `group by` clause groups rows in the `order_line` table based on the column `order_no`, and the `sum()` function is applied to each individual group. The `sum()` function compute the total quotation for each group (`order_no`). Please notice the `order_no` groups with the order number 12491 and 12498.

Having Clause:

The select condition or constraint in the where clause is applied to each individual record or tuple, whereas the constraint specified in the having clause is applied to the each individual group obtained from `group by` clause.

For example, to retrieve the total quotation for those orders which contains more than one order line from the `order_line` table.

```
SQL> select l.order_no, sum(l.quoted_price)
2 from jps2.order_line l
3 group by l.order_no
4 having count(l.order_no) > 1
```

ORDER_NO	SUM(L.QUOTED_PRICE)
12491	549.98
12498	37.9

The orders with the order number 12491 and 12498 are those which have more than one order line.

Please type in the following SQL statements to retrieve the answers to these questions in reading assignment #1.

1. Find the names of all the customers who have a credit limit of at least \$1500.

```
SQL> select c.last, c.first
2 from jps2.customer c
3 where c.credit >= 1500
```

2. Give the order numbers of those orders placed by customer 124 on September 5, 1998.

```
SQL> select o.order_no
2 from jps2.orders o
3 where o.cust_no = 124 and o.order_date = '9/05/98'
```

3. Give the part number, description, and on-hand value (units on hand * price) for each part in item class AP.

```
SQL> select p.part_no, p.part_description,
2 p.units_on_hand * p.unit_price as
3 units_on_handrxunit_price
4 from jps2.part p
5 where p.item_class = 'AP'
```

4. Find the number and name of each customer whose last name is Nelson.

```
SQL> select c.cust_no, c.last, c.first
2 from jps2.customer c
3 where c.last = 'Nelson'
```

5. How many customers have a credit limit of \$1000?

```
SQL> select count(*)
2 from jps2.customer c
3 where c.credit = 1000
```

6. Find the total balance for all the customers represented by sales rep 12.

```
SQL> select sum(c.balance)
2 from jps2.customer c
3 where c.slsrep_no = '12'
```

7. For each order, list the order number, order date, customer number, and customer name.

```
SQL> select o.order_no, o.order_date, o.cust_no, c.last, c.first
2 from jps2.orders o, jps2.customer c
3 where o.cust_no = c.cust_no
```

8. For each order placed on September 5, 1998, list the order number, order date, customer number, and customer name.

```
SQL> select o.order_no, o.order_date, o.cust_no, c.last, c.first
2 from jps2.orders o, jps2.customer c
3 where o.cust_no = c.cust_no
4 and o.order_date = '9/05/98'
```

9. Find the number and name of each sales rep who represents any customers with a credit limit of \$1000.

```
SQL> select  s.slsrep_no, s.last, s.first
2  from      jps2.sales_rep s, jps2.customer c
3  where     s.slsrep_no = c.slsrep_no
4  and       c.credit = 1000
```

10. For each order, list the order number, order date, customer number, customer name, along with the number and name of the sales rep who represents the customer.

```
SQL> select  o.order_no,      o.order_date,
2           c.cust_no,        c.last, c.first,
3           s.slsrep_no,     s.last, s.first
4  from      jps2.orders o, jps2.customer c, jps2.sales_rep s
5  where     o.cust_no = c.cust_no
6  and       c.slsrep_no = s.slsrep_no
```

Any comments and suggestions will be appreciated. Please send your comments to the following address:

Dr. Junping Sun
Graduate School of Computer and Information Sciences
Nova Southeastern University
6100 Griffin Road
Fort Lauderdale, FL 33314-4416 USA

Phone: (954) 262-2082
Fax: (954) 262-3915
Internet: jps@nova.edu