Appendix B:

Oracle SQL*Plus

I have used the following sources:

- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Michael Gertz: Oracle/SQL Tutorial, 1999.
  `http://www.db.cs.ucdavis.edu/teaching/sqltutorial/`
- Stefan Brass: Skript zur Vorlesung Informationssysteme (in German), Universität Hildesheim, 1996/97.
- SQL*Plus User's Guide and Reference, Release 8.0, Oracle, 1997, Part No A53718-01.
- SQL*Plus Quick Reference, Release 8.0, Oracle, 1997, Part No A53718-01.

Objectives

- Being able to start SQL*Plus in the UNIX environment and enter SQL queries and SQL*Plus commands.
- Knowing the difference between SQL and SQL*Plus.
- Being able to do simple output formatting, e.g. to change the width of columns when the rows do not fit on the screen.
- Being able to do more advanced output formatting, e.g. to create master/detail reports.
- Being able to write SQL*Plus scripts and to assess the possibility to solve a given task with such a script.
- Note that the material of appendices is not relevant to the exams, but it is relevant to the homeworks.

Overview

1. Getting Started
2. More SQL*Plus Commands
3. Output Formatting
4. SQL*Plus Scripts

About SQL*Plus (1)

- SQL*Plus is one user interface to the Oracle DBMS.
  It is mainly used for “ad hoc” queries, which are not supported by special application programs or forms.
- Mainly, SQL*Plus does the following:
  - You enter a command in the database language SQL.
  - SQL*Plus sends this to the database server.
  - SQL*Plus gets the result back from the server.
  - SQL*Plus prints the result on the screen (or in a file).
- It is intended for experienced users.
  It is a text-based command-style interface. No graphics, no menus, but the full power of SQL (or even PL/SQL).

About SQL*Plus (2)

- SQL*Plus has a simple built-in editor for SQL commands.

  This is mainly needed for changing the last command if it was incorrect. Any operating system editor can be called.
- SQL*Plus has many commands for formatting the output tables (width of columns, page breaks, etc.)

  The output can also be written to files. In this way, "reports" of medium complexity can be generated.
- SQL*Plus can also execute a file with a sequence of commands (SQL*Plus script).

  So simple programs (without if, while, etc.) can be written in SQL*Plus. In fact, since SQL is powerful, more tasks than you might think can be done with such scripts.

Starting SQL*Plus (1)

- You must log in one of the SIS Solaris Computers.

  E.g. "`paradox.sis.pitt.edu`". The SPARC systems in room SIS 819 should work. The old SPARC systems ("...lis.pitt.edu") do not have Oracle installed.
- You can log in also from home over the internet via "telnet".
- These computers run a version of the UNIX operating system ("Solaris" from SUN Microsystems).

  You need a UNIX account with password to log in.
  Oracle runs on many different hardware platforms.
  You can get it also for PCs under Windows 95/98/NT.
- See Appendix A for more information about Solaris.

Starting SQL*Plus (2)

- Many environment variables, which are necessary for Oracle, are set automatically for you. (So you can ignore this page.)
  See also Tables 2-2&3 in "Administering Oracle8 on Solaris 2.x".
  - Identification of the database (there can be multiple databases) and network connection information:

    `ORACLE_SID=sis, TWO_TASK=sis`
  - Place of Oracle Data in the file system:

    `ORACLE_HOME=/opt/local/sources/packages/oracle`
    `/product/8.0.4`
  - Find Oracle programs and dynamic libraries:

    `LD_LIBRARY_PATH=...:$ORACLE_HOME/lib`
    `PATH=...:$ORACLE_HOME/bin`

Starting SQL*Plus (3)

- Enter the command "`sqlplus`".

  You have to enter a user-name and a password. Oracle has its own user list. A UNIX account is not sufficient for using Oracle, you need a database account in addition.

  Oracle can be configured to trust the identification of the operating system, but in our case you have to enter user-name and password.

  Often a guest account "scott/tiger" is created.
- The command "`SELECT * FROM CAT;`" shows all your database objects. `CAT` (catalog) is a data dictionary table.

  The result will probably be empty at the moment.

Starting SQL*Plus (4)

- SQL*Plus requires that all SQL commands must be terminated with a semicolon “;”.

  The semicolon is not part of SQL, it only helps SQL*Plus to detect the end of the SQL command.

  SQL commands can be split over multiple lines: If you start a new line without a semicolon, SQL*Plus will print a line number as a continuation prompt.
- In contrast, SQL*Plus control commands (i.e. everything except SQL and PL/SQL commands) end at the end of line.

  E.g. “quit” suffices, you don’t have to put a semicolon after it. If you want to continue an SQL*Plus command, you must put a hyphen as the last character of the line.

Starting SQL*Plus (5)

- Even without own tables, you can access the data dictionary tables. E.g. try

  `SELECT * FROM ALL_USERS;`
- All accessible tables are listed with

  `SELECT TABLE_NAME FROM ALL_TABLES;`
- “describe ⟨Table⟩” lists the columns of a table.
- There is a “help” command, e.g. “help quit”.
- You can leave SQL*Plus with “quit” or “exit”.
- You can interrupt a running query with ⟨Control⟩+C.

Reading Oracle Documentation

- You can use Netscape on the SIS Solaris systems (only).

  file:///opt/local/sources/packages/oracle/
  product/8.0.4/doc/server.804/index.htm
- If you have an X-Server running on another machine, you can redirect the Netscape-Window to that machine.

  E.g. your machine is 136.142.116.25. You start an X-server and enter “xhost +paradox.sis.pitt.edu” and log into paradox via telnet. Then you enter “DISPLAY=136.142.116.25:0.0” and “export DISPLAY” on paradox, and start Netscape.
- The Documentation is also available in PDF Format.

  E.g. “acroread $ORACLE_HOME/doc/index.pdf” or “acroread $ORACLE_HOME/doc/server.804/a53717.pdf for SQL*Plus (a53718.pdf: Quick Reference).

Building Demonstration Tables (1)

- Enter the command “demobld” from the UNIX prompt.

  So you must leave SQL*Plus first. The error message “default username feature not supported” of demobld is normal. It simply means that you must input username and password explicitly.
- Now “SELECT * FROM CAT;’’ should show five tables:
  - EMP: Information about employees.
  - DEPT: Information about departments.
  - SALGRADE: Salary intervals for assigning grade.
  - BONUS: Commission information.
  - DUMMY: Dummy table, only one row, one column.

Building Demonstration Tables (2)

- You can also copy the file

  /home/sbrass/public_html/db/sql/presidents.sql

  and then execute it under SQL*Plus with the command "@presidents" (This is an example of an SQL*Plus script).
- It creates the following tables:
  - PRESIDENT(PRES_NAME, BIRTH_YEAR, YEARS_SERV, DEATH_AGE, PARTY, STATE_BORN)
  - PRES_MARRIAGE(PRES_NAME, SPOUSE_NAME, PR_AGE, SP_AGE, NR_CHILDREN, MAR_YEAR)
  - PRES_HOBBY(PRES_NAME, HOBBY)

  (Continued on next page)

Building Demonstration Tables (3)

- Tables of the "US Presidents Database" (continued):
  - STATE(STATE_NAME, ADMIN_ENTERED, YEAR_ENTERED)
  - ELECTION(ELECTION_YEAR, CANDIDATE, VOTES, WINNER_LOSER_INDIC)
  - ADMINISTRATION(ADMIN_NR, PRES_NAME, YEAR_INAUGURATED)
  - ADMIN_PR_VP(ADMIN_NR, PRES_NAME, VICE_PRES_NAME)

  (Unfortunately, the data is not complete.)

Important Notes

- Please note that you must enter `demobld` outside of SQL*Plus (it is a UNIX command).
- On the other hand, `@presidents` is an SQL*Plus command.
- Although the data of all users are stored in one database, Oracle gives every user his/her own database schema.
- `demobld`/`@presidents` create your own copy of the tables.

  If you like, you can change them without affecting other users. Normally, other users cannot access your tables.
- The data is stored on the server, not in your home directory. So `ls` will not list your tables. You can access them only via Oracle (but you can export them from the DB into a file).

Overview

Correcting the Last Command (1)

- The last SQL command is still stored in a buffer. You can change and re-execute it (e.g. in case of a syntax error).
- `l[ist]`: Show current buffer contents.
  The current line is marked with an asterisc "`*`".
  After a syntax error, the current line is the line in which the error was detected. But "`l[ist]`" sets it to the last line.
- `l[ist]` ⟨`N`⟩: Set current line to ⟨`N`⟩.
- `c[hange]` `/`⟨`X`⟩`/`⟨`Y`⟩: Change ⟨`X`⟩ to ⟨`Y`⟩ in current line.
  The first occurrence of ⟨`X`⟩ is changed (even if it has a different capitalization). The pattern "`...`" in ⟨`X`⟩ matches arbitrary many arbitrary characters. You can use any non-alphanumeric character instead of /.

Correcting the Last Command (2)

- `a[ppend]` ⟨`X`⟩: Adds ⟨X⟩ to the current line.
  Only the first space after the command is regarded as a separation. Any other spaces are part of the added text.
- `i[nput]` ⟨`X`⟩: Adds ⟨X⟩ as a new line after the current one.
  Without arguments, `i[nput]` reads multiple input lines until you enter an empty line. All are added after the current line.
- ⟨`N`⟩ ⟨`X`⟩: Replace line ⟨`N`⟩ by ⟨`X`⟩.
  If you use 0 as ⟨`N`⟩, the line is inserted before all others.
- `del`: Deletes current line.
  You can use "`del` ⟨`N`⟩" to delete line ⟨N⟩,
  and "`del` ⟨`N`⟩ ⟨`M`⟩" to delete all lines from ⟨N⟩ to ⟨M⟩.

Correcting the Last Command (3)

- `r[un]`: Execute buffer (and print it first).
- `/`: Execute buffer (without listing the command).
- `get` ⟨`File`⟩: Read contents of file into buffer.
  The extension "`.sql`" is automatically added to ⟨File⟩.
- `save` ⟨`File`⟩: Writes buffer contents into file.
- `ed[it]`: Call operating system editor on the buffer contents.
  This gives you more comfortable editing possibilities.
  You can e.g., select the editor "`dtpad`" by using the command "`define _editor = dtpad`". You must close the editor window before SQL*Plus continues. You can also start `dtpad` outside SQL*Plus and use `@file` (see below).

More SQL*Plus Commands (1)

- `desc[ribe]` ⟨`Table`⟩: List column names and datatypes.
- `ho[st]` ⟨`Cmd`⟩: Execute an operating system command.
  E.g. "`host ls -l *.sql`". In the UNIX implementation of SQL*Plus, you can also use a "`!`" instead of "`host`".
- `spool` ⟨`File`⟩: Write a record of the session into a file.
  All following user input & system output is logged to "⟨`File`⟩`.lst`" (until "`spool off`" or "`spool out`").
- `spool off`: This stops the writing of the spool file.
- `spool out`: Stop writing spool file and print it.
  The name of the printer is defined by the environment variable "`LPDEST`" ("`lw8b`" by default).

More SQL*Plus Commands (2)

- passw[ord]: Change Oracle password.
  You have two different passwords for Oracle and for Solaris.
- conn[ect] ⟨User⟩: Log into Oracle as another user.
- set ⟨System Variable⟩ ⟨Value⟩: Set system option.
  SQL*Plus has over 50 system variables. The most important ones are mentioned below. See also "help set" and "SQL*Plus User's Guide and Reference", page 7-79.
- show ⟨Variable⟩: Show current value of system variable.
  "show all" shows the values of all system variables.
- store set ⟨File⟩: Save settings of system variables.
  They are stored in "⟨File⟩.sql".

More SQL*Plus Commands (3)

- set autocommit on: End a transaction after every update.
  If autocommit is off, this transaction is continued until you enter explicitly "commit" (or leave SQL*Plus). So if some error occurs, and the transaction has to be rolled back, you lose all your work. Of course, with autocommit on, you cannot undo your changes later with "rollback". So it might be safer to leave autocommit off, but say "commit" from time to time.

  set flagger entry: Print an error message whenever an SQL-command violates SQL92 standard (entry level).
  The default is "off", allowing Oracle extensions.
  Other standard levels are "intermediate" and "full".

User Variables (1)

- You can define user variables (also called "substitution variables") in the following form:

  define ⟨Variable⟩ = ⟨Value⟩
- E.g. define EMPLOYEE = SMITH
  define EMPLOYEE = 'SMITH' has exactly the same effect.
  Quotes are only needed if the value contains spaces, they do not become part of the variable value.
- define ⟨Variable⟩: Show current value of ⟨Variable⟩.
- define: Shows the values of all defined variables.
- undefine ⟨Variable⟩: Delete variable.

User Variables (2)

- SQL*Plus substitutes everywhere "&⟨Variable⟩" by the current value of the variable.
  In SQL statements and in SQL*Plus statements. The substitution takes place even inside string constants.
- ```
  select EMPNO, ENAME
  from   EMP
  where  ENAME = '&EMPLOYEE'
  ```
  You can use substitution variables also to define table or attribute names at runtime, or a complete condition.
  E.g. "select * from EMP where &COND;" is possible.
- If a letter or digit immediately follows the variable name, put a "." behind it, e.g. "&X.000" is substituted by 3000 if X=3.
  ```

User Variables (3)

- If you use “&X” without having defined “X”, SQL*Plus will ask for a value at runtime.

  So this is a simple input mechanism for SQL*Plus scripts (see below).
- The variable is not defined in this way, SQL*Plus will ask again the next time it has to evaluate “&X”.

  If you use the notation “&&X”, SQL*Plus defines the variable after it has asked the user for a value.
- If you want to enter a “&” literally, you must first define an escape character (“`set escape \`”) and then write “\&”.

Overview

1. Getting Started
2. More SQL*Plus Commands
3. Output Formatting
4. SQL*Plus Scripts

Table Output Format (1)

- Tables are printed in this way (here with `pagesize=6`):

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO

    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATIONS     BOSTON
```


Table Output Format (2)

- Tables are printed as a sequences of “pages”. The column names are repeated on each page.

  “`set pagesize` ⟨n⟩” sets the number of lines per page. The default is 14. If you set `pagesize` to 0, all headings are suppressed. The column headings are also suppressed with “`set headings off`”.
- Each page begins with a number of empty lines.

  This is controlled by the system variable `newpage`. The default is 1. If you set it to 0, a form feed (⟨Control⟩+L) is printed at the beginning of each page (in the spool file and on some terminals). “`set newpage none`” suppresses all empty lines at the beginning of a page.

Table Output Format (3)

- From a certain definable threshold, the number of selected rows is printed at the end of the table.

  “set feedback ⟨n⟩” sets this threshold. The default is 6. “set feedback on” always prints the number of selected rows and “set feedback off” suppresses this output.

- SQL*Plus can wait before every page for the user to press ⟨Return⟩.

  set pause on or set pause '⟨Prompt⟩' achieves this. You must press ⟨Return⟩ also before the first page.

- You can change the character used for underlining the attribute names (set underline), the string printed for null values (set null) and many more parameters.

Table Output Format (4)

- If a table row is wider than the screen width (defineable with set linesize), it is folded:

```
     EMPNO ENAME      JOB              MGR HIREDATE
---------- ---------- --------- ---------- ---------
       SAL       COMM     DEPTNO
---------- ---------- ----------
      7499 ALLEN      SALESMAN        7698 20-FEB-81
      1600        300         30

      7521 WARD       SALESMAN        7698 22-FEB-81
      1250        500         30
```

This format is quite unclear. You should try to make columns smaller to such that the record fits into one row.

COLUMN Command (1)

- How a column is printed can be influenced by the column command of SQL*Plus:

  col[umn] ⟨Attribute⟩ ⟨Option 1⟩ ⟨Option 2⟩ ...

  - One kind of option has the form “format ⟨Format⟩”.
  - E.g. “column MGR format 9999”: Use 4 digits.

    The field width is 5, one character is reserved for a possible minus. There are a lot of options for formatting numbers (especially for money). E.g. $999,990.99MI puts the dollar symbol, comma and decimal point in the specified places and a possible minus after the number. The “0” means that from this position leading zeros should be displayed.

COLUMN Command (2)

- Options for the column command, continued:
  - E.g. “column ENAME format A6”: Output field width 6.

    This is only for string valued attributes.

  - E.g. “column JOB format A6 truncated”

    This specifies that if the real value is longer than the field width, the rest should be ignored (not printed). The default is “wrapped” which means the output is continued in the next line under this field. If you want to break the string only at word boundaries, use “word_wrapped”.

COLUMN Command (3)

- Options for the `column` command, continued:
  - The "`heading`" option allows to define the column heading if you do not want the attribute name: E.g. "`column EMPNO format 9999 heading ID`".

    This is especially useful for computed columns, e.g.:

    ```
    select ENAME, SAL+COMM from EMP
    where  COMM is not null
    ```

    In this case, SQL*Plus would use `SAL+COMM` as column heading. But you can define some other heading, e.g. "`column SAL+COMM heading TOTAL`".
  - By adding "`justify left`" or "`justify right`" you can position the column heading.

COLUMN Command (4)

- Options for the `column` command, continued:
  - The options "`fold_before`"/"`fold_after`" can be used to specify where records should be broken between lines.
  - You can specify how a null value is represented, e.g. `column COMM null 'no comm.'`
  - "`column hiredate noprint`" suppresses the output of this column (e.g. you can order by columns not shown).
- You can remove a previous column definition (i.e. use the default output format) with "`column COMM clear`".
- `clear columns`: remove all column definitions.

Important Note

- Note that you must execute the `COLUMN` command (as well as `BREAK`, `TTITLE`, see below) before the query.
- SQL*Plus remembers these format specifications and applies them to the output of all future queries.
- Once you leave SQL*Plus, all such settings will be forgotten.

  In contrast, changes to tables are persistent.
- Normally, one would put the format definition together with the query into an SQL*Plus script (see below).
- Note that `COLUMN`, `BREAK`, `TTITLE` etc. are commands of Oracle SQL*Plus only. They are not clauses of SQL.

  Other database vendors might use different commands.

TTITLE Command (1)

- The `ttitle` ("top title) command defines a header to be printed at the top of each output page.
- It has the form "`ttitle` ⟨`Element 1`⟩ ⟨`Element 2`⟩ `...`":

```
ttitle center 'DATABASE CORP.' skip 1 -
       left 'Employee List' -
       right 'PAGE: ' format 99 SQL.PNO -
       skip 2
```

This centers "`DATABASE CORP.`" on the first line of the page header, then "`skip 1`" moves to the next line, in which "`Employee List`" is left justified and "`PAGE: `" plus the page number (two characters) is right justified. Finally, it moves two lines down for the query result to start.

TTITLE Command (2)

```
                     DATABASE CORP.
Employee List                                PAGE:   1

     EMPNO ENAME      JOB              SAL        MGR
---------- ---------- --------- ---------- ----------
      7369 SMITH      CLERK            800       7902
      7499 ALLEN      SALESMAN        1600       7698
      7521 WARD       SALESMAN        1250       7698

                     DATABASE CORP.
Employee List                                PAGE:   2

     EMPNO ENAME      JOB              SAL        MGR
---------- ---------- --------- ---------- ----------
      7566 JONES      MANAGER         2975       7839
```


TTITLE Command (3)

- `left`: Left justify the following text.
  `right`, `center`: analogously.
- `col` ⟨n⟩: Position following text at screen column ⟨n⟩.
  This may move the cursor backward.
  `tab` ⟨n⟩ moves the cursor forward ⟨n⟩ character positions.
- `skip` ⟨n⟩: Move down ⟨n⟩ lines.
- The output can contain user variables. The value of the variables can be formatted with a `format` clause.
- `ttitle off`: Switches printing of headlines off.
- `btitle` defines a footline (“bottom title”).

BREAK Command (1)

- Suppose we want to create a list of employees by department:

```
DEPTNO ENAME      EMPNO JOB          SAL
------ ---------- ----- --------- ------
    10 CLARK       7782 MANAGER     2450
    10 KING        7839 PRESIDENT   5000
    10 MILLER      7934 CLERK       1300
    20 ADAMS       7876 CLERK       1100
    20 FORD        7902 ANALYST     3000
    20 JONES       7566 MANAGER     2975
    20 SCOTT       7788 ANALYST     3000
    20 SMITH       7369 CLERK        800
    30 ALLEN       7499 SALESMAN    1600
    30 BLAKE       7698 MANAGER     2850
    30 JAMES       7900 CLERK        950
```


BREAK Command (2)

- It looks nicer if the `DEPTNO` is only printed when it changes (with respect to the previous row):

```
DEPTNO ENAME      EMPNO JOB          SAL
------ ---------- ----- --------- ------
    10 CLARK       7782 MANAGER     2450
       KING        7839 PRESIDENT   5000
       MILLER      7934 CLERK       1300
    20 ADAMS       7876 CLERK       1100
       FORD        7902 ANALYST     3000
       JONES       7566 MANAGER     2975
       SCOTT       7788 ANALYST     3000
       SMITH       7369 CLERK        800
    30 ALLEN       7499 SALESMAN    1600
       BLAKE       7698 MANAGER     2850
       JAMES       7900 CLERK        950
```

BREAK Command (3)

- With the "break" command, you can react on equal or changed column values.

  The "break" command makes only sense if you sort the output rows on this column (you can sort over multiple columns with different priorities, though).

- break on DEPTNO: Print "DEPTNO" only when changed.

  Of course, if SQL*Plus starts a new output page, the value for "DEPTNO" is always printed in the first row of the page.

- break on DEPTNO skip 2: Print two empty lines if "DEPTNO" changes. With "page" a new page is started.

  This also suppresses repeated values. To print "DEPTNO" in each row, use "break on DEPTNO skip 2 duplicates".

BREAK Command (4)

- You can have nested break definitions, e.g. an outer break over "DEPTNO" and an inner break over "JOB".

  Then you must use "order by DEPTNO, JOB" in the SQL-query to order the output over "DEPTNO", and for equal department numbers over "JOB".

- break on DEPTNO skip page on JOB skip 1

  The outer break must be listed first.

- You can have an outermost break "on report".

  This is only interesting for the "compute" command.

- break: Show all "break" definitions in effect.
- clear breaks: Delete all break definitions.

COMPUTE Command (1)

- The "compute" command prints summary lines at breaks:

```
DEPTNO ENAME      EMPNO JOB          SAL
------- ---------- ----- --------- ------
     10 CLARK       7782 MANAGER     2450
        KING        7839 PRESIDENT   5000
        MILLER      7934 CLERK       1300
*******                            ------
sum                                  8750

     20 ADAMS       7876 CLERK       1100
        FORD        7902 ANALYST     3000
        JONES       7566 MANAGER     2975
        SCOTT       7788 ANALYST     3000
*******                            ------
sum                                 10075
```


COMPUTE Command (2)

- compute sum of SAL on DEPTNO: Produces the above output.

  You need to have a "break" command first, e.g. "break on DEPTNO skip 1". The column after "on" must appear in the most recent "break" command.

- Instead of "sum", you can also choose: avg, count, maximum, minimum, number, std (standard derivation), variance.

  count ignores null values, number counts all rows.

- compute sum label 'Budget:' of SAL on DEPTNO

  This prints the string "Budget:" instead of "sum" in the summary lines. The break column must be wide enough.

COMPUTE Command (3)

- There can be more than one “compute” definition in effect. Especially, you can print a total summary at the very end in addition to the summaries for the single departments:

  ```
  compute sum of SAL on report
  ```

  Of course, there must be corresponding nested break definitions, e.g. “break on report on DEPTNO skip 2”.
- compute: list all “compute” definitions.
- clear computes: Removes all “compute” definitions.

Creating a Master/Detail Report (1)

- Suppose you want the department number in the page title, but not in the actual table:

```
Employees by Department       Department:  10

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7782 CLARK      MANAGER         2450
      7839 KING       PRESIDENT       5000
      7934 MILLER     CLERK           1300

Employees by Department       Department:  20

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7902 FORD       ANALYST         3000
      7876 ADAMS      CLERK           1100
```


Creating a Master/Detail Report (2)

- `column DEPTNO new_value DEPTVAR noprint`

  The clause “new_value DEPTVAR” assigns the current value of the column to the user variable “DEPTVAR”. There must be a “break” definition with the “skip page” action for the column. There is a corresponding “old_value” clause if you want to use the variable in a “btitle” footline.
- ```
  ttitle left 'Employees by Department' -
      right 'Department:  '  format 99 DEPTVAR -
      skip 1
  ```
- `break on deptno skip page`
- ```
  select DEPTNO, EMPNO, ENAME, JOB, SAL from EMP
  order by DEPTNO, EMPNO;
  ```


Setting a Variable to the Current Date

- Sometimes, the current date should be printed in the title of reports. Getting a variable which contains the current date is actually quite clumsy.
- `break on TODAY`
- `column TODAY new_value CURR_DATE`
- ```
  select to_char(SYSDATE, 'fmMonth DD, YYYY')
         TODAY from DUAL;
  ```

  “DUAL” is a system table with a single column and a single row. It is mentioned because SQL requires a “from” clause. Since the table has a single row, the “select” part is evaluated only once.

  “SYSDATE” returns the current date. “to_char” is used to format the date. The column of the resulting table is named “TODAY”.
  ```

Overview

1. Getting Started
2. More SQL*Plus Commands
3. Output Formatting
4. SQL*Plus Scripts

SQL*Plus Scripts (1)

- You can write SQL commands and SQL*Plus commands (e.g. for formatting the output) into a file.
  - Usually, such files have the extension “`.sql`”.
- From SQL*Plus, you can execute the commands in “⟨`File`⟩`.sql`” with “`@`⟨`File`⟩” (or “`start` ⟨`File`⟩”).
  - The environment variable “`ORACLE_PATH`” specifies a list of directories, in which SQL*Plus searches for the file.
- You can pass arguments: “`start` ⟨`File`⟩ ⟨`Arg 1`⟩ `...`”
  - You can access the arguments in the script as `&1`, `&2`, etc.
- The file “`login.sql`” is automatically executed whenever SQL*Plus starts.

SQL*Plus Scripts (2)

- You can start an SQL*Plus script from UNIX as

  `sqlplus` ⟨`User`⟩ `@`⟨`File`⟩ ⟨`Arg1`⟩ ⟨`Arg2`⟩ `...`

  In this syntax, SQL*Plus will prompt for the password.

  You can put the password on the command line (in the form ⟨`User`⟩`/`⟨`Password`⟩), but then other users can see it when they list processes with the UNIX command “`ps -ef`”.

  You can also put ⟨`User`⟩`/`⟨`Password`⟩ as the first line of the command file.

  Note that you must put a `quit`-command into the file, or the user will remain logged into SQL*Plus after the script is executed.

SQL*Plus Scripts (3)

- `rem`[`ark`]: The rest of the line is ignored.
  - You can put comments into the SQL*Plus script in this way.
  - The “`remark`” command must start in the first column.
- `prompt` ⟨`Text`⟩: This will print ⟨Text⟩.
- `pause` ⟨`Text`⟩: Wait until the user presses ⟨`Return`⟩.
  - ⟨`Text`⟩ is printed as a prompt.
- `accept` ⟨`Variable`⟩ ⟨`Type`⟩ `prompt` ⟨`Text`⟩ [`hide`]: Get input value for variable from user.
  - ⟨`Type`⟩ is “`number`” or “`char`”, “`hide`” is for passwords.
  - E.g.: `accept X number prompt 'Please enter X: '`

Writing Query Results to Files (1)

- You use the `spool`-command (see above), but in order to log only the pure table data, you must set the following system variables:
  - `set pagesize 0`: No headlines.

    Without this setting, SQL*Plus assumes that a new page starts on every 14 lines and prints the column names again. You can define the number of lines between such intermediate headlines with this variable. The value 0 is special since it suppresses all headlines.
  - `set feedback off`

    Do not print a summary like “`14 rows selected`”.

Writing Query Results to Files (2)

- System variable settings for pure table data, continued:
  - `set echo off`: No SQL-commands into spool-file.

    Normally, user input is also logged into the spool-file. This can be suppressed with “`set echo off`”. However, for some unclear reason, this works only from inside a SQL*Plus script, not for interactively entered commands.
  - `set termout off`: Do not print the query result on the screen (only in the spool file).

Writing Query Results to Files (3)

- `spool` ⟨`File`⟩: Log query results.
- `select ...`: The SQL query.
- `spool off`: Close spool file.
- You may want to use “`store set SAVESET`” and “`@SAVESET`” around these commands to restore the system variables to their previous values.

  You must add “`replace`” to the “`store`” command if the file might already exist. Alternatively, you may want to delete it at the end with “`host rm SAVESET.sql`”.
- If you want delimiting characters between the columns, you can use the string concatenation operator “`||`” of SQL.

Writing Query Results to Files (4)

- The SQL*Plus script shown on the next page saves the data in the “`DEPT`” table as a series of SQL “`insert`” commands.
- After executing this script, the file “`DEPT_BAK.sql`” contains:

```
insert into DEPT values (10, 'ACCOUNTING', 'NEW YORK');
insert into DEPT values (20, 'RESEARCH', 'DALLAS');
insert into DEPT values (30, 'SALES', 'CHICAGO');
insert into DEPT values (40, 'OPERATIONS', 'BOSTON');
```

- SQL*Plus scripts which generate (and possible execute) other SQL*Plus scripts are actually quite powerful.

  SQL*Plus has no “`if`” and “`while`” statements, but with this technique you have the full power of SQL to construct the commands you want to execute.

Writing Query Results to Files (5)

```
prompt Saving the DEPT table in DEPT_BAK.sql ...
store set SAVESET replace
set pagesize 0
set feedback off
set echo off
set termout off
spool DEPT_BAK.sql
remark Remember that a quote inside a string literal is doubled:
select 'insert into DEPT values (' ||
       DEPTNO || ', ''' || DNAME || ''', ''' ||
       LOC || ''');'
       from DEPT;
spool off
@SAVESET
host rm SAVESET.sql
```