Introduction to Oracle

Wolfgang May
may@informatik.uni-freiburg.de
Institute for Computer Science,
University Freiburg

October 2004

1 General

1.1 Administrative stuff

The training is carried out in groups of three until four students each. The participants log on to Oracle via their own UNIX accounts. Thus, in addition to the UNIX account for accessing the institute's workstations, also an **Oracle account** is required to connect to the database. Each participant has an own ORACLE-Account.
Group-directory
For each group there exists an additional directory `/home/dbis/db-prakt/grp`*n* where *all* group members (and the supervisors) are allowed to read and write. There, SQL-scripts of the group should be placed.
Communication with the supervisors is done by the mailing list

- `sql-user` : supervisors and participants

Slides, lecture notes, exercise sheets etc. can be found on the Web:

`http://dbis.informatik.uni-freiburg.de/index.php?course=SS06/`
`Praktikum/Programmierung+in+SQL/index.html`

1.2 Group Directory

For each group, there is a directory at
`/home/dbis/db-prakt/grp`*x*
where *x* is the group number. Additionally, each group is administered as a UNIX group, called `oragrp`*x*.
For the supervision process you have to put the SQL scripts into the directory to make it accessible for the supervisor some days before the discussion. For this, you should create directories `sheet1`–`sheet7` in the group directory, and then put your code for the individual exercises into the directories as `ex1.sql, ex2.sql, ...` The code should be

- correct, i.e., must be executable as an SQL query or program,
- suitably indented and
- contains comments that make it understandable.

The supervisor will have a look at your programs before the discussion (and will perhaps require you to enhance the code).
For organizing your directory in this way, you need the following UNIX commands:

- `cd /home/dbis/db-prakt/grp`*x*
- `mkdir sheet1` Then, the directory is generated, but only accessible for you. Make it usable for your group members by

- `chmod 2770 sheet1`
- `chgrp oragrp`*x* `sheet1`
- `cd sheet1`
- write a file `ex1.sql`, and make it also accessible for the group members by
- `chmod 660 ex1.sql`
- `chgrp oragrp`*x* `ex1.sql`

Alternatively you can use the shell scripts `oracheck` and `orafix` to check and correct the permissions. The scripts are available at the documents web page.
Also you can use the group directory for a CVS repository. Then you have version control and can setup email notifications in case of changes and additions.

1.3 Modifications of the UNIX environment

Each participant has to modify his system environment accordingly:

adapt .bashrc/ .login

- The environment variables for ORACLE are set by `setup databases/oracle101`. This command may either be called manually on-demand, or it can be added to the file `.bashrc` (for bash shell) or `.login` (for tcsh[1]).

  set ORACLE_WORK

  This file should also contain the line

  ```
  setenv WORK /home/dbis/db-prakt/grpn
  ```

  (where *n* is the number of the group where the participant belongs to), to be able to change to this directory simply with `cd $WORK`.

- create ~/oracle/

  Additionally, a directory `oracle/` should be created in the private home directory where the ORACLE system places its own files.

- login.sql

  The output from ORACLE is controlled by ORACLE Variables. These are set by the file `oracle/login.sql`. E.g., for interrupting the output after 50 lines a time, this file should contain the lines

  ```
  set pause '- continue -'
  set pause on
  set pagesize 50
  ```

  Similarly, the length of output lines can be constrained e.g. by `set linesize 200` (The semantics of these (and other) variables can be checked from the SQL*Plus environment (which is explained below) by calling `help set`).

- Language

  The language of ORACLE messages is set by the environment variable `NLS_LANG`.

  `setenv NLS_LANG american_america.we8iso8859p1` selects English, and
  `setenv NLS_LANG german_germany.we8iso8859p1` selects German.

- The `ORACLE_PATH` variable tells SQL*Plus where to search for its startup file "login.sql", and for SQL scripts. It is recommended to set `ORACLE_PATH` in `.login` to

  ```
  setenv ORACLE_PATH .:/home/dbis/db-prakt/MondialDB/oracle:$HOME/oracle
  ```

  since `/home/dbis/db-prakt/MondialDB/oracle` contains the scripts for generating the database.

- specify Editor

  For editing SQL scripts from SQL*Plus, an editor must be specified. This editor must be

[1]student account usually run under tcsh.

specified in the environment variable `EDITOR` (e.g., emacs oder xemacs, pico, oder vi), by adding the line

```
setenv EDITOR 'emacs -nw'
```

to `.login`.

If any of these files is changed, the changes must be activated by calling `source <filename>`.

Further Oracle Environment Variables. A description of all ORACLE environment variables can be found in Appendix B of the "Administrator's Reference Guide". Two of them are described below:

ORACLE_HOME: The installation directory of ORACLE.

ORACLE_ADMIN: describes the directory where local administrative scripts are placed (e.g., definitions of user roles for creating ORACLE accounts). The global startup file 'glogin.sql' for SQL*Plus can also be found here.

2 The User Interface SQL*Plus

SQL*Plus is a simple interactive user interface (also called "SQL*Plus-Editor"). It is called by he command `sqlplus` in a shell. By providing the user name (of the ORACLE account!) and a password, the connection to the database is opened. Authorization can also be done by using `/` instead of a user name – in this case, no password is required since the user is connected to the database account that is associated with his UNIX account. Then, the invocation of SQL*Plus can simply be done by `sqlplus /`. Having different user name and password is useful when a user has access to several ORACLE accounts (thus, not in the SQL training). After displaying some messages, the prompt `SQL>` appears in SQL*Plus. SQL*Plus is terminated by `quit`. start SQL

SQL*Plus commands extend the common SQL commands, allowing for more comfortable database queries and displays. SQL and SQL*Plus commands may be issued interactively or by SQL-Scripts.

2.1 Interactively

Commands are directly issued in the SQL*Plus command line. Interactively, only single SQL statements can be executed. An SQL statement may consist of several lines. Lines are enumerated automatically for an easier editing. After typing a semicolon at the end of a statement, the statement is executed.

```
SQL>  select * (Return)
   2  from <table>; (Return)
```

Instead of typing the commands directly, they can be copied from an editor window.

SQLPlus in emacs. Using SQL*Plus as described above in a shell, the interface is fairly uncomfortable (no history etc). A more comfortable use is possible by running SQL*Plus in an emacs: after starting a shell inside emacs by `Meta-x shell`, SQL*Plus may be called as usual by `sqlplus /`. Then, all emacs functionality, e.g., Meta-p/Meta-n (history), Ctrl-a/Ctrl-e (start/end of line) and cursor keys is available.

Changing the Database Note that changes to the database via SQL commands are visible only for the user who has executed this command. For all other users, the changes are only visible after executing `commit` (see description of transactions). Moreover, other users cannot also write the same table at the same time.

2.2 SQL*Plus by using SQL-Scripts

SQL scripts are files which contain SQL- and SQL*Plus statements, in the same way as these can be entered directly with SQL*Plus. Such files should have the extension *.sql (and do not require an x-flag since they are not executed, but only read by SQL*Plus).
Using scripts, a sequence of commands can be sent to the database without having to enter each command individually by SQL*Plus. Thus, complex transactions are possible.

Editing in SQL*Plus and an editor.
Form the SQL*Plus prompt, the specified editor is called by `edit <filename.sql>`. A new window opens for editing the given file. After leaving the editor, the user is back in the SQL*plus environment.

edit Scripts

Scripts are called in SQL*Plus by `start <filename>` (or `@<filename>` as a short key). An extension `.sql` can be omitted.

execute scripts

```
SQL>  edit name.sql   Invoke Editor
  ⋮
SQL>  start name      Execute Script
```

Some people may prefer to have SQL*Plus in a shell, (x)emacs as a window, and copy statements from one window to the other.

Useful SQL*plus commands:

ed(it) <file>: starts editor with `<file>`

sta(rt) <file>: executes the SQL script given by `<file>`. The extension `.sql` is required only if the filename itself contains dots (thus, it is recommended to use names such as `lsg1_2.sql` instead of `lsg1.2.sql`).

@<file>: equivalent to `start <file>`.

desc(ribe) <table>: shows the table structure of `<table>`.

show <xy>: shows the contents of the SQL*Plus variable `<xy>`.

show all: shows the contents of all SQL*Plus variables.

help: activates the help functionality.

help <keyword>: activates the help functionality to the given keyword.

ho(st) <cmd>: executes the Unix command `<cmd>`.

! <cmd>: analogous to `host`.

exit or quit: Leave SQL*Plus. Then, `commit` is executed automatically. If the most recent modifications to the database should not be materialized, an explicit `rollback` is required. If SQL*Plus variables have been changed, these changes persist. If this is not desired, these variables must be specified in `login.sql` which is executed each time when SQL*Plus is called.

SQL*Plus scripts may also be called by

```
sqlplus -S / @<SQL-Script>
```

directly from the UNIX shell. Then, SQL*Plus is invoked and the given script is started. The `-S` option suppresses the messages from SQL*Plus (copyright, prompt,...). Further output can be controlled by setting variables in the script (e.g. `set feedback off`, `set verify off`, ...). The final command of the script must be `exit`, otherwise SQL*Plus is not left. It is recommended for handling potential errors to have the command `whenever sqlerror exit sql.sqlcode` at the beginning of the script. This command leaves SQL*Plus if an error occurs.

3 How to get started – the first session

3.1 Generation of the Database

Log on to the ORACLE account via the SQL*Plus interface by calling `sqlplus /`. After its creation, the account is completely empty – i.e., an empty "Tablespace" in which a database must still be generated.[2]
The MONDIAL database is then created by calling

create database

```
@create.sql;
```

(calls the SQL script `/home/dbis/db-prakt/MondialDB/oracle/create.sql`). This script recursively calls other scripts which generate the whole database in the user's account. With each later login, the database is found in the same state as it has been left after the previous session. In case that unintendedly something is deleted/changed in the database, the original database can be restored by calling `@create.sql`.
Individual tables can also be copied (from the reference database in the "dbis" account) by calling the script `@consult` after deleting all tuples from the user's table:

```
delete from <table>;
@consult;
Tabelle: <table>
```

Recall that every SQL command must be finished by a **Semicolon** ... otherwise, nothing happens!

3.2 Querying the Data Dictionary.

The Data Dictionary stores data *about* the database schema (called *metadata*). It consists of several tables which can be queried as usual by `SELECT ... FROM`. Calling

```
SELECT * FROM user_objects;
```

yields an overview of all things that are stored in the database. E.g.,

```
SELECT object_name,object_type FROM user_objects;
```

returns the names of all stored objects and shows their data-object-type (e.g., table, procedure, function, user). Especially, the names of all tables can be listed by

```
SELECT object_name FROM user_objects WHERE object_type='TABLE' ;
```

Then,

```
DESCRIBE <table>;
```

shows the complete definition of the table `<table>`. The table itself can then be output by

```
SELECT * FROM <table>;
```

[2]note that "database" simultaneously means the whole system (in early papers also called a "data bank"), and the stored data itself.

First exercise: Look around in the MONDIAL database. It represents the world in the state of 1996.

3.3 Transactions and Persistent Changes.

Changes which have been issued interactively from SQL*Plus can either be made a) persistent by calling `commit`, or they can b) be undone by calling `rollback`. Leaving SQL*Plus automatically executes `commit`. The following commands illustrate this functionality:

`select * from city;`	shows the complete relation.
`delete from city;`	deletes all tuples from the relation, as can be checked by
`select * from city;`	
`rollback;`	undoes this change, as shown by
`select * from city;`	
`select * from country;`	shows this relation.
`delete from country;`	deletes it, that is made persistent by
`commit;`	and can be checked by:
`select * from country;`	Then, also a
`rollback;`	cannot help:
`select * from country;`	After
`delete from sea;`	and (unintended)
`quit;`	also this table is lost, as is shown when calling
`sqlplus /`	and
`select * from sea;`	Now, it is time to execute
`@create.sql`	to obtain again the original database state.

3.4 Daily Work

For solving the exercises, it is recommended to change to the group directory by calling `cd $WORK`. There, SQL*Plus, (x)emacs, pico etc. are invoked. Each group should use a consistent naming of scripts.

3.5 Help!

ORACLE/SQL*Plus provides an online help which is invoked by `help` and explains itself then. Especially, `help commands` yields a list of all commands of SQL*Plus, SQL and PL/SQL.
For SQL commands, the ORACLE *Server SQL Language* is the recommended reference, for general database concepts, the ORACLE *Server Concepts Manual* is useful. The Web page of the course also contains a link to a click-able Manual.