Oracle SQL Exercise

1. Executing SQL Commands

SQL commands in Oracle are NOT case sensitive. The command may extend over several lines and can be broken anywhere there is a space. SQL commands must end with a semicolon. Try these commands:

```
SQL> create table cardlist
        (name          char(20),
        address                char(20),
        phone          char(10));

SQL> insert into cardlist values
        ('John Smith',
        '1234 Up Street',
        '222-2222');

SQL> insert into cardlist values
        ('Mary Brown',
        '2567 Down Street',
        '333-3333');

SQL> insert into cardlist values
        ('Ed Lin',
        '123 Old Dos Way',
        '444-4444');
```

Now try an SQL*Plus command. SQL*Plus is an Oracle tool that provides a user interface and executes SQL commands. It provides some data formatting and allows interaction with the operating system. No semicolon is required for SQL*Plus commands.

```
SQL> DESCRIBE cardlist
```

What does it do?

2. SQL*PLUS Command Buffer

The last SQL command issued is automatically stored in a command buffer and can be recalled, edited and re-run.

```
SQL> List              (or just L)
```

Only the last SQL command is stored, not SQL*Plus command.

List *n* - displays the nth line of the command

```
SQL> List 2
```

Run - lists and executes the last command

/ - runs the last command without listing

SQL> Run

3. Editing Commands in the Buffer

The command in the buffer can be edited using the following commands. Only the current line is affected.

c(hange)	/oldstring/newstring/	- replaces oldstring with newstring
i(nput)	text	- adds text after the current line
d(el)		- deletes the current line
a(ppend)	text	- appends text to the end of the current line

Use these commands to edit your last INSERT command and add 5 new records to the cardlist table.

4. Saving and Retrieving Commands

A command can be saved to a file using the SAVE command:

SQL> SAVE filename (CREATE | REPLACE | APPEND)

CREATE - creates a new file or warns you if the file already exists

REPLACE - replaces an existing file

APPEND - appends the command to the end of an existing file

Save your last INSERT command in a file.

A file can be retrieved into the command buffer using the GET command. The START (@) command can be used to execute the statements in a file. Note that you cannot load a bunch of statements into your buffer and then RUN it. You need to use START or @ to directly execute the commands in a file.

SQL> GET filename (LIST | NOLIST)

LIST - lists the file as it is retrieved

NOLIST - does not list the file

GET does not automatically execute the command. You must use the RUN or / command (for only one statement in the buffer).

5. Using Variables in Command Scripts

When faced with using a command repetitively, it is often desirable to use variable names in the script. This enables the system to prompt you for data each time the script is run. Retype the INSERT command as follows:

```
SQL> Insert into cardlist values (&1, &2, &3);

The system will respond:
enter value for 1
SQL> 'Jack Jones'
enter value for 2
SQL> '1234 Cambie'
enter value for 3
SQL> '555-5555'
1 row created

SQL> save test

SQL> @test
```

6. Disconnect from Oracle

Use Exit or Quit to exit SQL*Plus.

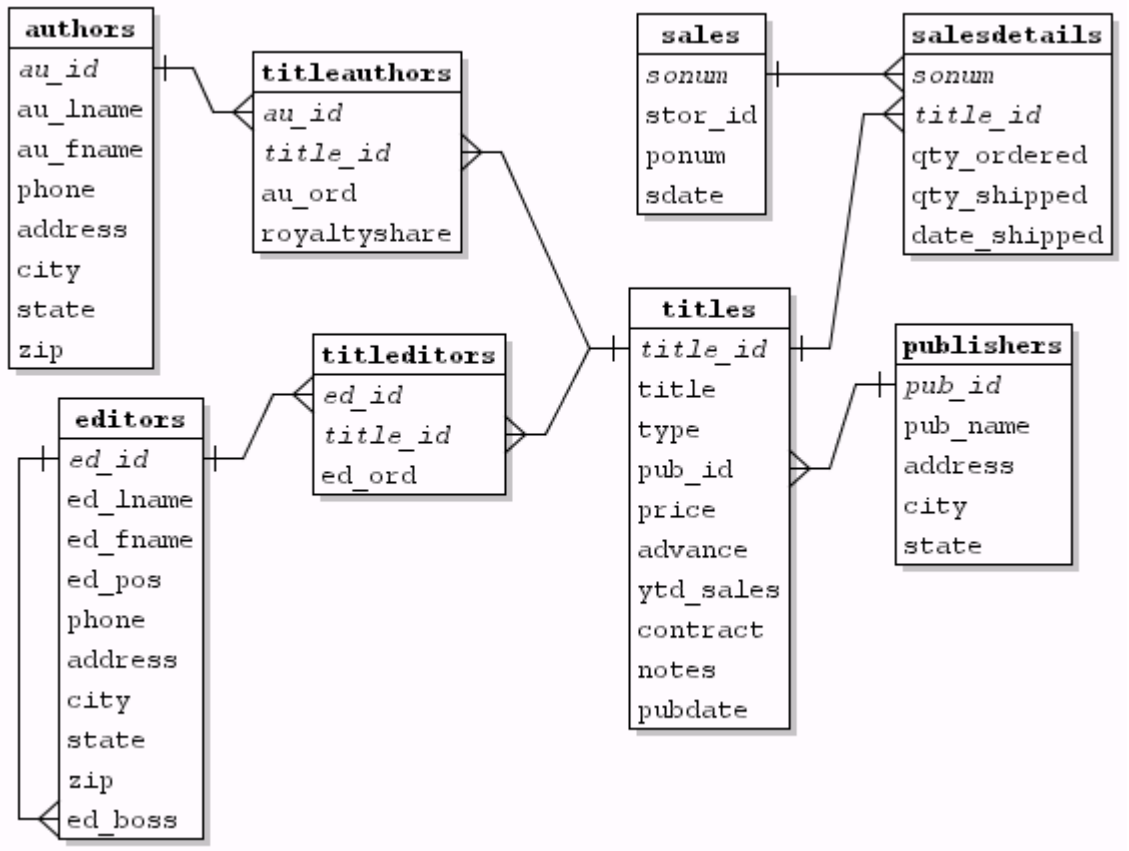
You can change to another account using the connect command.

```
SQL> connect username / password
```

7. Load Tables

Download the bookbiz.zip file in your space and unzip it. (You may need to create a directory, like tutorial2, save the file there, and make it your current directory). You will find a `bookbiz.sql` file with the SQL commands you need for this part.
Execute the file `bookbiz.sql` to create the tables for the following questions.
.

Browse through the tables and try to understand the structures. Draw a Table diagram for this database. A table diagram has a box for each table and shows the name of the table and its attributes. For each foreign key tha diagram shows a link (line) between the foreign key and the key referenced by that.



8. Create SQL Queries for each of the following:
(Don't bother with the Relational Algebra. Ask them to use only SQL)

a.. Show all columns and rows in the authors table. You should get all 23 rows.

Answer:
SELECT *
FROM authors;

Relational Algebra:
$\pi_{au\_id,\ au\_lname,\ au\_fname\ \ldots}$ (authors)

b. Show the first and last names of all authors. You will get all 23 rows.

Answer:
SELECT au_fname, au_lname
FROM authors;

Relational Algebra:
$\pi_{\ au\_fname,\ au\_lname\ \ldots}$ (authors)

c. Show the first and last names of all authors, but this time make the columns labels at the top "FirstName" and "LastName" (no blanks). Use column aliases for this. You will get 23 rows.

Answer:
```
SELECT au_fname AS FirstName, au_lname AS LastName
FROM authors;
```

Relational Algebra:
$\pi_{\text{au\_fname, au\_lname ...}}$ (authors)

d. Show the first and last names of all authors, but this time make the column labels at the top "First" and "Last" (no blanks). Use column aliases for this. Hint: The difference between this problem and the previous is that the names collide with SQL keywords. How do you avoid that? You will get 23 rows.

Answer:
```
SELECT au_fname AS "First", au_lname AS "Last"
FROM authors;
```

Relational Algebra:
$\pi_{\text{au\_fname, au\_lname ...}}$ (authors)

e. Which authors live in Walnut Creek*?* Show all columns. You should get 1 row (Akiko Yakomoto).

Answer:
```
SELECT *
FROM authors
WHERE city='Walnut Creek';
```

Relational Algebra:
$\pi_{\text{au\_id, au\_lname, au\_fname ...}}$ ($\sigma_{\text{city='Walnut Creek'}}$ authors)

f. Which orders are incomplete*?* List the title_ids and the number of titles that still have to be shipped to complete orders. Show only rows where not as many titles have been shipped as ordered. Use the salesdetails table for this. There should be 5 rows in your result.

Answer:
```
SELECT title_id,  qty_ordered-qty_shipped
FROM salesdetails
WHERE qty_shipped < qty_ordered;
```

Relational Algebra:
$\pi_{\text{title\_id, qty\_ordered – qty\_shipped}}$ ($\sigma_{\text{qty\_shipped < qty\_ordered}}$ salesdetails)

g. Which editors don't have a boss*?* List all editors (first and last names) without a boss (NULL in ed_boss field). There will be three rows in the result.

Answer:
```
SELECT ed_fname, ed_lname
FROM editors
WHERE ed_boss IS NULL;
```

Relational Algebra:

$\pi_{\text{ed\_fname, ed\_lname}} (\sigma_{\text{ed\_boss IS NULL}} \text{editors})$

h. Which editors do NOT have 993-86-0420 as a boss*?* List all editors (last names) and their bosses. HINT: This may not be quite as easy as it looks because of NULLs. If you did it right, you'll have a 7 row result.

Answer:
```
SELECT ed_lname, ed_boss
FROM editors
WHERE ed_boss IS NULL
OR ed_boss <> '993-86-0420';
```

Relational Algebra:

$\pi_{\text{ed\_lname, ed\_boss}} (\sigma_{\text{ed\_boss IS NULL} \vee \text{ed\_boss} <> \text{'993-86-0420'}} \text{editors})$

i. Which non-business books cost between $20 and $30*?* Give the title, type, and price. There are 6 rows in the result.

Answer:
```
SELECT title, type, price
FROM titles
WHERE type <> 'business' AND price BETWEEN 20 AND 30;
```

Relational Algebra:

$\pi_{\text{title, tpe, price}} (\sigma_{\text{type} <> \text{'business'} \wedge \text{price} >= 20 \wedge \text{price} <= 30} \text{titles})$

j. List the last names of all authors who have a letter 'k' in their last name*?* There should be three matching rows.

Answer:
```
SELECT au_lname
FROM authors
WHERE LOWER(au_lname) LIKE '%k%';
```

Relational Algebra:

$\pi_{\text{au\_lname}} (\sigma_{\text{au\_lname like '\%k\%'}} \text{authors})$

k. List all titles (title) followed by their publishers (pub_name). There should be 18 rows.

Answer:
```
SELECT title, pub_name
  FROM titles, publishers
  WHERE publishers.pub_id = titles.pub_id
```

Relational Algebra:

$\pi_{\text{title, pub\_name}}$ (titles $\bowtie$ publishers)