

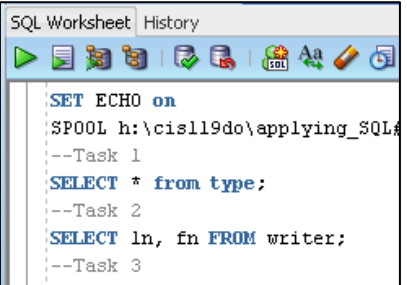
# Applying SQL #2

(30 points • est. time 120 minutes)

## Background

This exercise has you apply material we've seen in chapters 1-4 of our *Oracle SQL* text and the class notes. You will find it helpful to refer to the Issue25 and Student schema diagrams.

By the end of the assignment, you will have developed a script that resembles the illustration. The top two lines will ensure your commands and results are written to a text file (`applying_SQL#2.txt`) that you'll print and turn in for grading. The remaining lines are pairs – a comment to identify the task number and the statement which accomplishes the task.



```
SQL Worksheet History
SET ECHO on
SPOOL h:\cis119do\applying_SQL#2.txt
--Task 1
SELECT * from type;
--Task 2
SELECT ln, fn FROM writer;
--Task 3
```

## Preliminaries

1. Launch *SQL Developer* and connect to our *cis119do* database.
2. Save your script (worksheet) as **applying\_SQL#2.sql** to the `h:\cis119do` folder (if connected via mySCC) or to `c:\cis119do` (if connected via your own PC).
3. Enter the top two lines (**ECHO** and **SPOOL**) and the comment for Task 1.

## Tasks

Develop a statement for each task below. Build one statement at a time and run it (F9) to test that it works correctly. If not, tweak, run, and test the statement until it works correctly. If you get stuck, comment it out and move on to the next task. *Save your script frequently.*

1. Show the course number, description, and cost for courses that have no prerequisite and whose course number is less than 100. Sort the results in descending order of cost.
2. Show the last name, first name, and contact code for each writer. When no contact is available, display the text **No contact available** instead. Have the column heading for the contact values say **CONTACT CODE**.

3. Show the full name (eg: Lawton, Pat) and amount for only the freelancers. Also include a column of calculated results that shows what the amount would be if each freelancer were given a 27.3% raise, with the calculated amount, *always rounded up* to the next whole dollar.
4. Show the title and type of any BUS or ADV or FMK article. Use the **DECODE** function to display the text Business, Advertising, or Financial Market instead of the 3-character type abbreviation. Your output should resemble:

	TITLE	Article Type
1	Trans-Alaskan Oil Pipeline Opening	Business
2	AT&T Antitrust Settlement	Business
3	Building Trade Outlook	Business
4	Cola Advertising War	Advertising
5	Stock Market's Black Monday	Financial Market
6	Advertising Over the Past 25 Years	Advertising
7	The Economy Under Sub-Zero Population Growth	Business

5. Show the title, type, issue and length for articles whose title is longer than 30 characters and whose length is less than 2,000 words.
6. Display the description and cost for any course having **Unix** anywhere in its title. Use *string function(s)* to locate the courses. Do not use wildcard(s).
7. Use string functions to display a *nonduplicated* list of student salutations, with periods removed (eg: Mr. should display as Mr). Do not use **DECODE**.

8. Show each article's title and a descriptive word indicating the article's length. For articles up to 1,000 words in length, display the word **short**; for those longer than 1,000 and up to 2,500 words, display the word **medium**; for those longer than 2,500 and up to 4,000 words, display the word **long**. Any article longer than 4,000 words should be labeled **novel**. Your output should resemble the following. Note that the rows are sorted by title.

ID	TITLE	LENGTH
1	\$100 Billion AOL Time Warner Merger Approved	Medium
2	25% Tax Cut Bill Approved	Medium
3	AT&T Antitrust Settlement	Medium
4	Advertising Over the Past 25 Years	Long
5	Alternative Energy Sources	Medium
6	Bingham Family Feud	Medium
7	Building Trade Outlook	Medium
8	Bush Finally Wins!	Novel
9	Chrysler Asks U.S. For \$1 Billion	Short
10	Cola Advertising War	Medium

9. My schema includes a table named `pledge` that stores data for a bowling fundraiser in which people pledge either a flat amount (such as \$5.00), an amount per pin knocked down (eg: \$0.50), or a combination of both a flat amount and a per pin amount. List its structure (`descrttrollen.pledge`) and then develop a query that calculates `TOTAL PLEDGE`, as illustrated below. Note that the rows are arranged in `TOTAL PLEDGE` sequence.

ID	PLEDGEID	SCORE	FLAT_AMOUNT	PER_PIN_AMOUNT	TOTAL PLEDGE
1	7	100	5	5	505
2	4	171	(null)	2	342
3	5	255	10	1	265
4	2	200	5	1	205
5	1	100	50	(null)	50
6	3	90	(null)	0.5	45
7	6	134	(null)	0.1	13.4
8	8	300	(null)	(null)	0

## Wrap Up

1. Save your final script.
2. Run your final script.
3. Open your spooled text file in a word processor. Type your name in the first line. Change the font size to 8 or 10 points and use the Courier New font.
4. Save and **print** your spooled text file.
5. **Print** your final script.