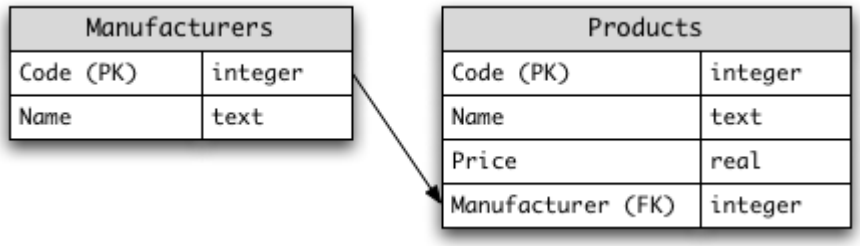


1.
2. Select the names of all the products in the store.
3. Select the names and the prices of all the products in the store.
4. Select the name of the products with a price less than or equal to $200.
5. Select all the products with a price between $60 and $120.
6. Select the name and price in cents (i.e., the price must be multiplied by 100).
7. Compute the average price of all the products.
8. Compute the average price of all products with manufacturer code equal to 2.
9. Compute the number of products with a price larger than or equal to $180.
10. Select the name and price of all products with a price larger than or equal to $180, and sort first by price (in descending order), and then by name (in ascending order).
11. Select all the data from the products, including all the data for each product's manufacturer.
12. Select the product name, price, and manufacturer name of all the products.
13. Select the average price of each manufacturer's products, showing only the manufacturer's code.
14. Select the average price of each manufacturer's products, showing the manufacturer's name.
15. Select the names of manufacturer whose products have an average price larger than or equal to $150.
16. Select the name and price of the cheapest product.
17. Select the name of each manufacturer along with the name and price of its most expensive product.
18. Add a new product: Loudspeakers, $70, manufacturer 2.
19. Update the name of product 8 to "Laser Printer".
20. Apply a 10% discount to all products.
21. Apply a 10% discount to all products with a price larger than or equal to $120.

22.

Query all attributes of every Japanese city in the **CITY** table. The **COUNTRYCODE** for Japan is `JPN`.

The **CITY** table is described as follows:

CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

Query the **NAME** field for all American cities in the **CITY** table with populations larger than `120000`. The CountryCode for America is `USA`.

The **CITY** table is described as follows:

CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

23.

24.

Query the names of all the Japanese cities in the **CITY** table. The **COUNTRYCODE** for Japan is JPN.

The **CITY** table is described as follows:

CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from **STATION**. Your result cannot contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

25. where LAT_N is the northern latitude and LONG_W is the western longitude.

Find the difference between the total number of **CITY** entries in the table and the number of distinct **CITY** entries in the table.

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where **LAT_N** is the northern latitude and **LONG_W** is the western longitude.

For example, if there are three records in the table with **CITY** values 'New York', 'New York', 'Bengalaru', there are 2 different city names: 'New York' and 'Bengalaru'. The query returns 1, because $\text{total number of records} - \text{number of unique city names} = 3 - 2 = 1$.

Query the list of CITY names from **STATION** which have vowels (i.e., a, e, i, o, and u) as both their first and last characters. Your result cannot contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

26. where LAT_N is the northern latitude and LONG_W is the western longitude.

Query the list of CITY names from **STATION** that do not start with vowels. Your result cannot contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

Query the list of CITY names from **STATION** that do not end with vowels. Your result cannot contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

27.

Query the list of CITY names from **STATION** that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

28. where LAT_N is the northern latitude and LONG_W is the western longitude.

Query the list of CITY names from **STATION** that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.

Input Format

The **STATION** table is described as follows:

STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT_N is the northern latitude and LONG_W is the western longitude.

29.

Query the Name of any student in **STUDENTS** who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

Input Format

The **STUDENTS** table is described as follows:

Column	*Type*
ID	*Integer*
Name	*String*
Marks	*Integer*

The Name column only contains uppercase

30. (A-Z) and lowercase (a-z) letters.

Write a query that prints a list of employee names (i.e.: the name attribute) from the **Employee** table in alphabetical order.

Input Format

The **Employee** table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number

31. of months they've been working for the company, and salary is their monthly salary.

Write a query that prints a list of employee names (i.e.: the name attribute) for employees in **Employee** having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee_id.

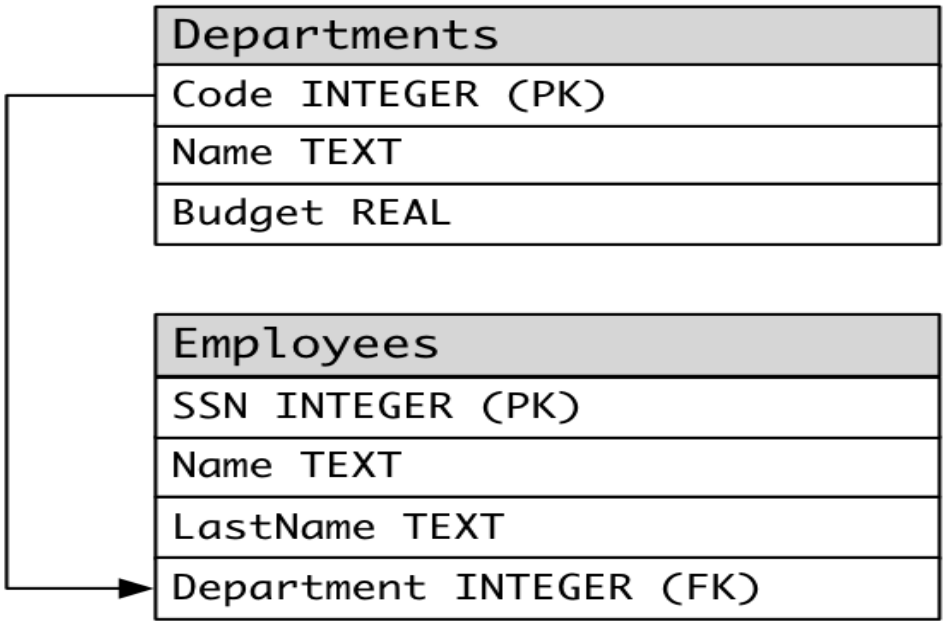
Input Format

The **Employee** table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

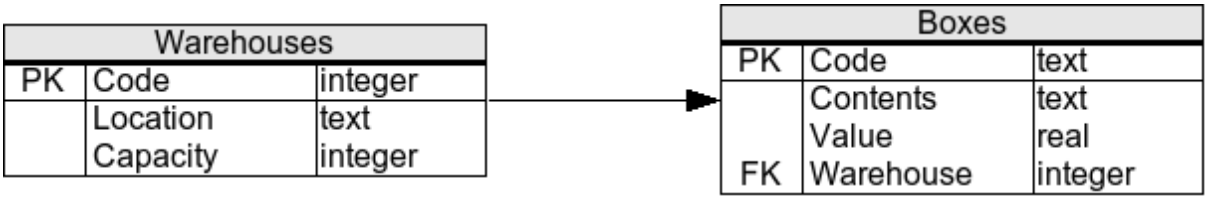
where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

32.

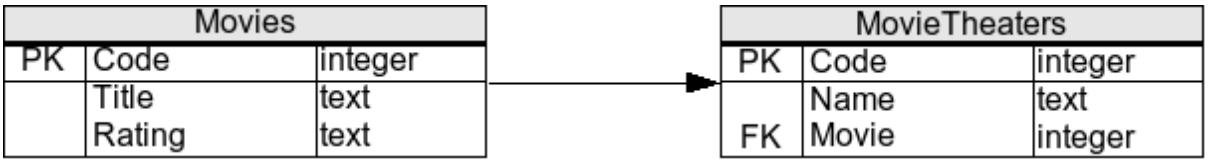


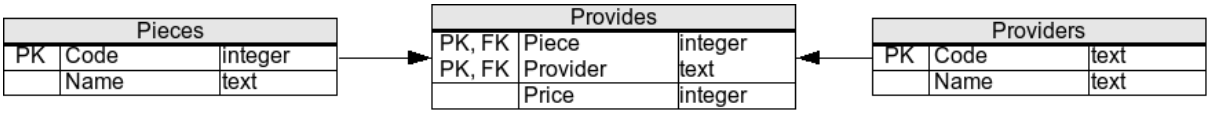
33.

34. Select the last name of all employees.
35. Select the last name of all employees, without duplicates.
36. -- 2.3 Select all the data of employees whose last name is "Smith".
37. -- 2.4 Select all the data of employees whose last name is "Smith" or "Doe".
38. -- 2.5 Select all the data of employees that work in department 14.
39. -- 2.6 Select all the data of employees that work in department 37 or department 77.
40. -- 2.7 Select all the data of employees whose last name begins with an "S".

41. Select the sum of all the departments' budgets.
42. Select the number of employees in each department (you only need to show the department code and the number of employees).
43. Select all the data of employees, including each employee's department's data.
44. Select the name and last name of each employee, along with the name and budget of the employee's department.
45. Select the name and last name of employees working for departments with a budget greater than $60,000.
46. Select the departments with a budget larger than the average budget of all the departments.
47. Select the names of departments with more than two employees.
48. Very Important - Select the name and last name of employees working for departments with second lowest budget.
49. Add a new department called "Quality Assurance", with a budget of $40,000 and departmental code 11.
50. Add an employee called "Mary Moore" in that department, with SSN 847-21-9811.
51. Reduce the budget of all departments by 10%.
52. Reassign all employees from the Research department (code 77) to the IT department (code 14).
53. Delete from the table all employees in the IT department (code 14).
54. Delete from the table all employees who work in departments with a budget greater than or equal to $60,000.
55. Delete from the table all employees.
56. 

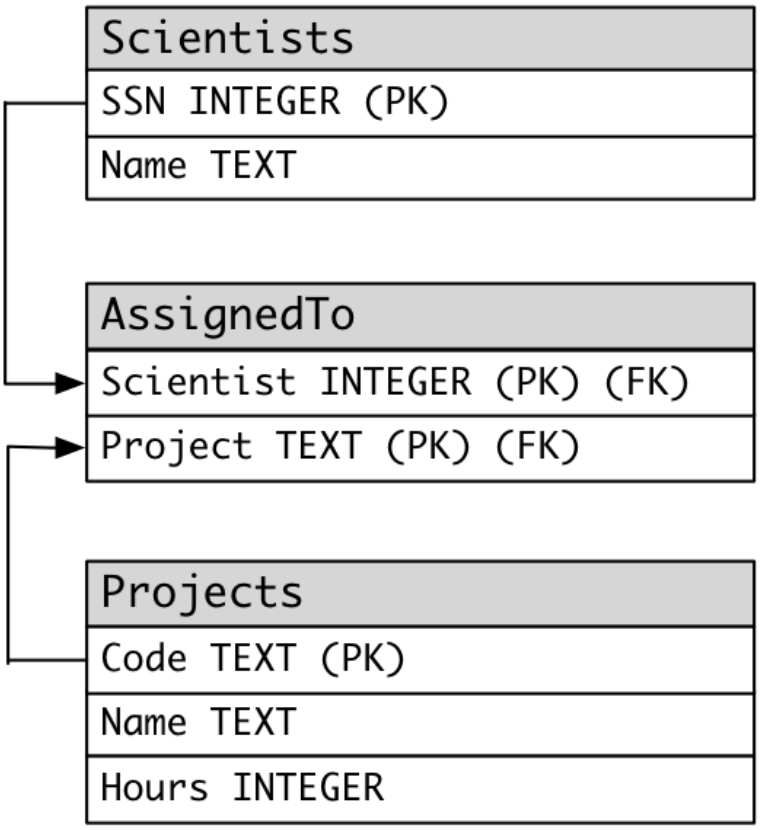
57. --3.1 Select all warehouses.
58. --3.2 Select all boxes with a value larger than $150.
59. --3.3 Select all distinct contents in all the boxes.
60. --3.4 Select the average value of all the boxes.
61. --3.5 Select the warehouse code and the average value of the boxes in each warehouse.
62. --3.6 Same as previous exercise, but select only those warehouses where the average value of the boxes is greater than 150.
63. --3.7 Select the code of each box, along with the name of the city the box is located in.
64. --3.8 Select the warehouse codes, along with the number of boxes in each warehouse.
65. -- Optionally, take into account that some warehouses are empty (i.e., the box count should show up as zero, instead of omitting the warehouse from the result).
66. --3.9 Select the codes of all warehouses that are saturated (a warehouse is saturated if the number of boxes in it is larger than the warehouse's capacity).
67. --3.10 Select the codes of all the boxes located in Chicago.
68. --3.11 Create a new warehouse in New York with a capacity for 3 boxes.

69. --3.12 Create a new box, with code "H5RT", containing "Papers" with a value of $200, and located in warehouse 2.
70. --3.13 Reduce the value of all boxes by 15%.
71. --3.14 Remove all boxes with a value lower than $100.
72. -- 3.15 Remove all boxes from saturated warehouses.
73. -- 3.16 Add Index for column "Warehouse" in table "boxes"
74. -- !!!NOTE!!!: index should NOT be used on small tables in practice
75. -- 3.17 Print all the existing indexes
76. -- !!!NOTE!!!: index should NOT be used on small tables in practice
77. -- 3.18 Remove (drop) the index you added just
78. -- !!!NOTE!!!: index should NOT be used on small tables in practice
79. 

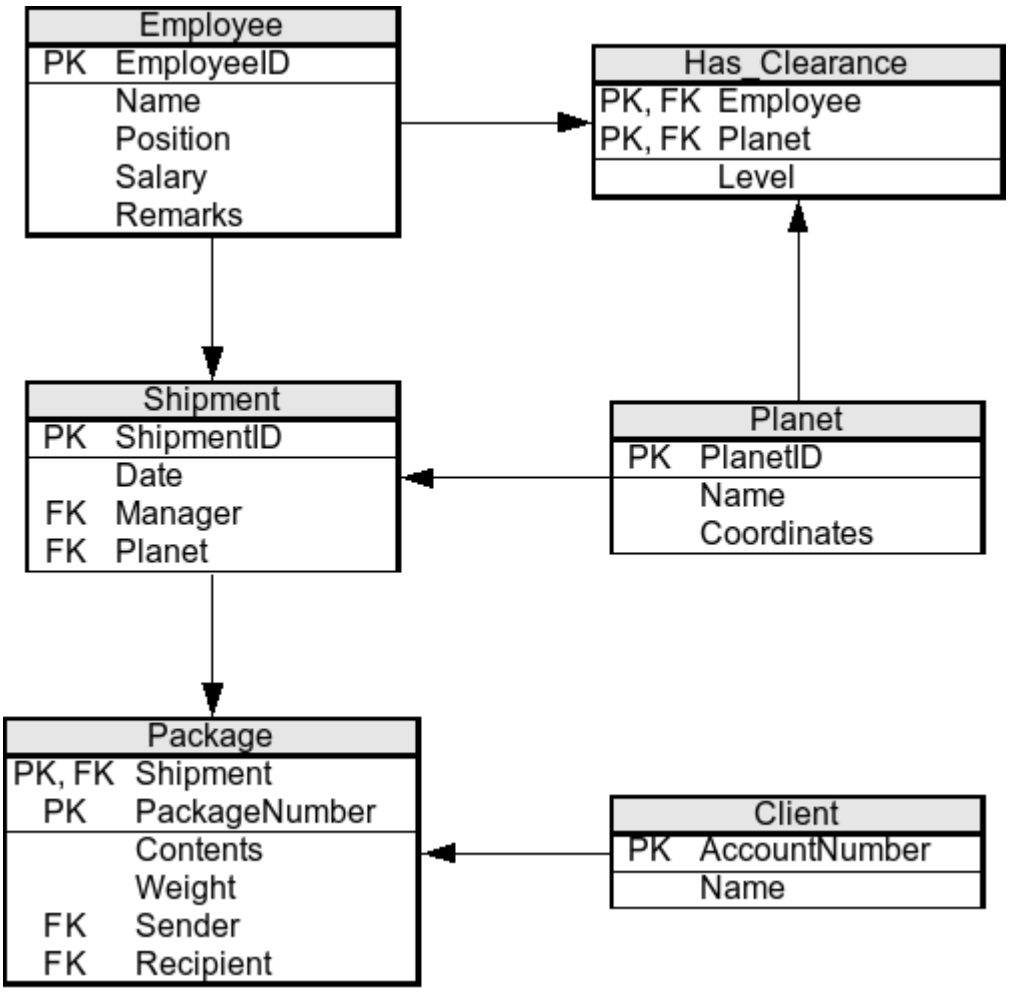
80. -- 4.1 Select the title of all movies.
81. -- 4.2 Show all the distinct ratings in the database.
82. -- 4.3 Show all unrated movies.
83. -- 4.4 Select all movie theaters that are not currently showing a movie.
84. -- 4.5 Select all data from all movie theaters
85. -- and, additionally, the data from the movie that is being shown in the theater (if one is being shown).
86. -- 4.6 Select all data from all movies and, if that movie is being shown in a theater, show the data from the theater.
87. -- 4.7 Show the titles of movies not currently being shown in any theaters.
88. -- 4.8 Add the unrated movie "One, Two, Three".
89. -- 4.9 Set the rating of all unrated movies to "G".
90. -- 4.10 Remove movie theaters projecting movies rated "NC-17".
91. 

92. -- 5.1 Select the name of all the pieces.
93. -- 5.2 Select all the providers' data.
94. -- 5.3 Obtain the average price of each piece (show only the piece code and the average price).
95. -- 5.4 Obtain the names of all providers who supply piece 1.
96. -- 5.5 Select the name of pieces provided by provider with code "HAL".
97. -- 5.6
98. -- ----------
99. -- !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
100. -- Interesting and important one.
101. -- For each piece, find the most expensive offering of that piece and include the piece name, provider name, and price
102. -- (note that there could be two providers who supply the same piece at the most expensive price).
103. -- ----------

104. -- 5.7 Add an entry to the database to indicate that "Skellington Supplies" (code "TNBC") will provide sprockets (code "1") for 7 cents each.
105. -- 5.8 Increase all prices by one cent.
106. -- 5.9 Update the database to reflect that "Susan Calvin Corp." (code "RBT") will not supply bolts (code 4).
107. -- 5.10 Update the database to reflect that "Susan Calvin Corp." (code "RBT") will not supply any pieces
108. -- (the provider should still remain in the database).



109.
110. -- 6.1 List all the scientists' names, their projects' names,
111. -- and the hours worked by that scientist on each project,
112. -- in alphabetical order of project name, then scientist name.
113. -- 6.2 Select the project names which are not assigned yet

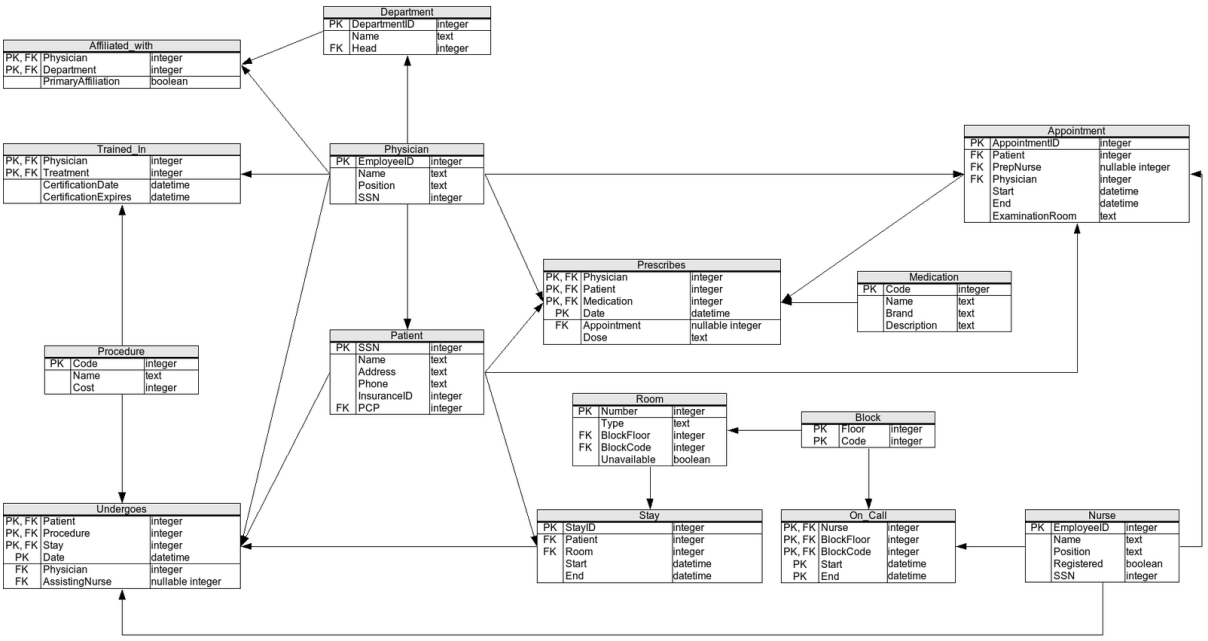


114.

115. -- 7.1 Who receieved a 1.5kg package?

116. -- The result is "Al Gore's Head".

117. -- 7.2 What is the total weight of all the packages that he sent?



118.

119. -- 8.1 Obtain the names of all physicians that have performed a medical procedure they have never been certified to perform.

120. -- 8.2 Same as the previous query, but include the following information in the results: Physician name, name of procedure, date when the procedure was carried out, name of the patient the procedure was carried out on.
121. -- 8.3 Obtain the names of all physicians that have performed a medical procedure that they are certified to perform, but such that the procedure was done at a date (Undergoes.Date) after the physician's certification expired (Trained_In.CertificationExpires).
122. -- 8.4 Same as the previous query, but include the following information in the results: Physician name, name of procedure, date when the procedure was carried out, name of the patient the procedure was carried out on, and date when the certification expired.
123. -- 8.5 Obtain the information for appointments where a patient met with a physician other than his/her primary care physician. Show the following information: Patient name, physician name, nurse name (if any), start and end time of appointment, examination room, and the name of the patient's primary care physician.
124. -- 8.6 The Patient field in Undergoes is redundant, since we can obtain it from the Stay table. There are no constraints in force to prevent inconsistencies between these two tables. More specifically, the Undergoes table may include a row where the patient ID does not match the one we would obtain from the Stay table through the Undergoes.Stay foreign key. Select all rows from Undergoes that exhibit this inconsistency.
125. -- 8.7 Obtain the names of all the nurses who have ever been on call for room 123.
126. -- 8.8 The hospital has several examination rooms where appointments take place. Obtain the number of appointments that have taken place in each examination room.
127. -- 8.9 Obtain the names of all patients (also include, for each patient, the name of the patient's primary care physician), such that \emph{all} the following are true:
128. -- The patient has been prescribed some medication by his/her primary care physician.
129. -- The patient has undergone a procedure with a cost larger that $5,000
130. -- The patient has had at least two appointment where the nurse who prepped the appointment was a registered nurse.
131. -- The patient's primary care physician is not the head of any department.
132. -- 10.1 Join table PEOPLE and ADDRESS, but keep only one address information for each person (we don't mind which record we take for each person).
133. -- i.e., the joined table should have the same number of rows as table PEOPLE
134. -- 10.2 Join table PEOPLE and ADDRESS, but ONLY keep the LATEST address information for each person.
135. -- i.e., the joined table should have the same number of rows as table PEOPLE

136.

Write a query identifying the type of each record in the **TRIANGLES** table using its three side lengths. Output one of the following statements for each record in the table:

- **Equilateral**: It's a triangle with 3 sides of equal length.
- **Isosceles**: It's a triangle with 2 sides of equal length.
- **Scalene**: It's a triangle with 3 sides of differing lengths.
- **Not A Triangle**: The given values of A, B, and C don't form a triangle.

Input Format

The **TRIANGLES** table is described as follows:

Column	*Type*
A	*Integer*
B	*Integer*
C	*Integer*

Generate the following two result sets:

1. Query an alphabetically ordered list of all names in **OCCUPATIONS**, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: `AnActorName(A)`, `ADoctorName(D)`, `AProfessorName(P)`, and `ASingerName(S)`.
2. Query the number of ocurrences of each occupation in **OCCUPATIONS**. Sort the occurrences in ascending order, and output them in the following format:

   ```
   There are a total of [occupation_count] [occupation]s.
   ```

   where `[occupation_count]` is the number of occurrences of an occupation in **OCCUPATIONS** and `[occupation]` is the lowercase occupation name. If more than one Occupation has the same `[occupation_count]`, they should be ordered alphabetically.

Note: There will be at least two entries in the table for each type of occupation.

Input Format

The **OCCUPATIONS** table is described as follows:

Column	*Type*
Name	*String*
Occupation	*String*

Occupation will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

137.