396 Top Dog

The Problem

You are a top-secret, government-employed software engineer assigned to TOP-DOG, the latest military intelligence program. TOP-DOG handles everything from mapping out enemy territory and position to parsing highly encrypted messages from the Commander in Chief.

The operator of TOP-DOG needs to be able to transfer information from remote computer workstations onto diskette in case of hardware problems or well-placed enemy fire. All TOP-DOG information is stored in the Oracattle database, but the only way to access the database is through the Dispatcher, an infamous and powerful software layer that only allows access to the database on a "need-to-know" basis. You are in charge of writing a piece of software which will get the desired information from the Oracattle database and store it on a "flat" file which will later be sent out to disk.

Input and Output

The input consists of one or more descriptions with a blank line between two consecutive ones. Each of them begin with a line containing the number of the description (1, 2, ...), and then three sets of lines (*pseudofiles*) separated with a blank line.

The desired tables to be exported will be stored in the first set. This *pseudofile* will simply contain the names of each table to be exported on separate lines. There will be no blank lines, and each table name will be unique. An example is shown in the second set of lines in the sample input section.

For each table, you must search in the second *pseudofile* which will contain all column names, types, and sizes (if VARCHAR) for each table in the Oracattle database. This is the only information which the Dispatcher allows you access to without begging. An example file can be found in the sample input section.

Note that, for security reasons, not every table in the first *pseudofile* will always be available in the second one. Under each table name is the column name (one unique word), data type, and size (for VARCHAR), each separated by one space. The `#` sign indicates there are no more columns for the table. There will never be consecutive `#` signs immediately following each other, and each table will contain at least one column name with the size. The only four data types are VARCHAR, INT, DATE, and LONGINT.

An Oracattle SQL statement must be built in order to query the Dispatcher for the desired table data. The Oracattle SQL statement must be precisely built in order to keep the Dispatcher happy (we wouldn't want the Dispatcher to be confused). The statement begins with "`SELECT`", followed by each column name and generic data type in parenthesis, separated with commas, terminated with "`FROM`", the table name, and a semicolon. The generic data type is CHAR for VARCHAR and DATE, and NUM for INT and LONGINT. The generic name must be used because the Dispatcher only understands data as CHAR or NUM (it may be powerful, but it's not extremely intelligent). These `SELECT` statements must be put into the first *pseudofile* (set of lines) in the output file, just after the number of description. Look at the sample output section, for the exact format.

If a table name cannot be found in the second *pseudofile*, a "`$TABLE NOT FOUND>`" statement must be substituted for the `SELECT` statement in the output file. No blank lines are to be in this set of lines, and only single spaces are to separate `SELECT`, column names, `FROM`, and the table name in the `SELECT` statements. The entire `SELECT` statement must be on one line.

Since you currently do not have access to the Dispatcher, we will assume that you have correctly built the first output *pseudofile* and that the Dispatcher has processed it and created the table information in the third input *pseudofile*. This set of lines contains the table name followed by the data from each row in the Oracattle Database table. "`<NULL>`" is returned for rows with empty fields. An example of this third set of lines is shown in the sample input section.

Note that each line there may contain any number of spaces between words unless it was declared as a NUM. Also, all data is returned by row and table name in the same order it was presented to the Dispatcher. If no data exists in the table, a `#` sign immediately follows the table name (as in `GROUPSPI` in the example data).

You must now finally integrate all the information you have received from the Dispatcher into a second output *pseudofile* (separated from the first one with a blank line). This set of lines will contain all data needed to describe the database tables. This *pseudofile* will later be imported using the Oracattle SqlImporter (OSI), a text to database utility. Lucky for you, all you need to do is get it into the OSI format. This can be a little tricky. The first argument to be supplied is the table name, followed by the number of columns in parentheses, followed by the number of records (rows) in parentheses (no spaces on this line). Next comes the column name and then the data in quotation marks (a single space should separate the column name and it's data). When the maximum length of the column data is greater than 64, the size must also be supplied in parenthesis immediately following the column name (no space in-between). This is so OSI can allocate more memory for large data. An example is shown in the sample output section.

No blank lines are to exist into this set of lines, and all data must remain on the same line as the column name (no end-of-line characters in-between quotation marks). Once this file has been created, you are all done!

There will be a maximum of 100 columns in a single table, but there may exist any number of rows in a single table. The maximum column name and table name length is 25, and the maximum data length is 100. All input *pseudofiles* will always contain data in an the expected format (as described in these specifications), so there is no need for error checking. Remember, case is iMpOrTaNt- `<NULL>` is not the same as `<null>`.

Sample Input

```
1
INTELSYS
GROUPSPI
SYSINTEL
DEPLOYREG

GROUPSPI
GRCODE INT
GRSUBNET VARCHAR 20
GRREGION VARCHAR 25
GRACTION VARCHAR 100
GRREF VARCHAR 100
#
INTELSYS
ISDATE DATE
```

```
ISNUM INT
ISGEN LONGINT
ISSUBGEN VARCHAR 25
#
SYSINTEL
SITRANS INT
SISUBLET LONGINT
SINUM INT
SIGEN VARCHAR 10
SIACTION VARCHAR 50
SINOTES VARCHAR 100
#
QUICKFI
QFDATE DATE
QFDATA VARCHAR 100
#

INTELSYS
122922T DEC 94
1
2
<NULL>
111111Z DEC 01
3
4
CONFIRMED
010101Z DEC 02
5
6
<NOT CONFIRMED>
020202Z DEC 03
7
8
CAN´T SAY
#
GROUPSPI
#
SYSINTEL
342
3498938
000
SCOUTA
PURGE DATABASE
UNABLE TO COMPLY WITH A2DD UNDER GENERAL BURK´S COMMAND
#
```

Sample Output

```
1
SELECT (CHAR) ISDATE, (NUM) ISNUM, (NUM) ISGEN, (CHAR) ISSUBGEN FROM INTELSYS;
SELECT (NUM) GRCODE, (CHAR) GRSUBNET, (CHAR) GRREGION, (CHAR) GRACTION, (CHAR) GRREF FROM GROUPSPI;
SELECT (NUM) SITRANS, (NUM) SISUBLET, (NUM) SINUM, (CHAR) SIGEN, (CHAR) SIACTION, (CHAR) SINOTES FROM SYSINTEL;
<TABLE NOT FOUND>

INTELSYS(4)(4)
ISDATE "122922T DEC 94"
ISNUM "1"
ISGEN "2"
ISSUBGEN ""
ISDATE "111111Z DEC 01"
ISNUM "3"
ISGEN "4"
ISSUBGEN "CONFIRMED"
ISDATE "010101Z DEC 02"
ISNUM "5"
ISGEN "6"
ISSUBGEN "<NOT CONFIRMED>"
ISDATE "020202Z DEC 03"
ISNUM "7"
ISGEN "8"
ISSUBGEN "CAN´T SAY"
GROUPSPI(5)(0)
SYSINTEL(6)(1)
SITRANS "342"
SISUBLET "3498938"
SINUM "000"
SIGEN "SCOUTA"
SIACTION "PURGE DATABASE"
SINOTES(300) "UNABLE TO COMPLY WITH A2DD UNDER GENERAL BURK´S COMMAND"
```