Hochschule Fulda
University of Applied Sciences

FACHBEREICH
ANGEWANDTE
INFORMATIK

Prof. Dr. Prof. h.c. Peter Peinl

Verteilte Datenbanken
Distributed Databases

Assignment 2

In this assignment, a simple table has to be loaded with synthetic data. Synthetic means that an arbitrary number of tuples can generated by means of a program. Benchmarking makes very often use of such tables containing synthetic data.

The aim of this small program is to present and use techniques to generate massive amounts of synthetic data that do not seem artificial to a human observer. The degree of naturalness of the data considerably depends on the expenditure involved in the programming in the production. In order to limit the effort, only some of the possibilities will be shown. The ideas and techniques outlined can, however, be extended as desired.

The overall task of the assignment comprises the following steps (subtasks) :

1. Create a table (definition) using a text file (for example using Oracle SQLDeveloper) nd test it.

2. Write a program that loads the table according to the following specification. The name of the table, at first, will be R1K. Eventually your program should allow to build bigger versions of the table, named R10K, R100K, etc.

3. The table has several columns, a numeric primary key, a candidate key and a foreign key. Moreover, to check for the correctness and facilitate debugging of your program, you may formulate integrity constraints. They define the domains (boundaries) of the values contained in the table's columns and may help when testing the program.

   Name the table R1K (the 1K in the name stands for 1000 tuples) and define columns with the following names and types:

   - PK integer (primary key)
   - CK1 integer (candidate key)
   - FK integer (foreign key referencing PK)
   - FIBO integer
   - GV10 integer
   - GV10000 integer
   - UV30 integer
   - LV1000 integer
   - STADT100 varchar2(256)
   - DAT100 date

   Write a program, that is capable of inserting an arbitrary given number of tuples into the table.

4. The values and to be stored in the columns and the restrictions to be respected by the program are as follows.

- **PK** contains consecutive (gapless) integer values starting with 1. This column is the primary key.

- **CK1** is a candidate key of the table. The value CK1 is always three times the value of the respective PK. However, CK1 has to be set to the SQL null value, if it would be 0 modulo 5.

- **FK** is of type integer. The values contained in column FK have to comply with the foreign key relationship to column PK. Hence, FK will always be the last decimal digit of PK. In other words, FK is PK modulo 10. If the remainder is 0, no actual value will be given to FK, i.e. FK will be set to the SQL null value..

  Examples:

PK	**FK**	Correct
1	1	Yes
2	2	Yes
7	7	Yes
8	4	No
10	Null	Yes
10	0	No
73	3	Yes
93	9	No
234	4	Yes

- **FIBO** stores the numbers of the Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, ..,
  i.e.. $fibo_{i+2} = fibo_{i+1} + fibo_i$.
  Those values rapidly exceed the boundaries of an integer value in Java. Whenever (before) that happens, restart with a value of 0 and repeat the series.

- **GV100** contains integers (equally distributed) between 1 and 100 (use the Java random number generator).

- **GV10000** contains integers (equally distributed) between 1 and 10000 (use the Java random number generator).

- **UV30** contains a skewed (unequal) distribution of the number from 1 to 29. The use of the random number generator and the algorithm described below will achieve this.

  Fetch a number (equally distributed between 1 and 1000) from the random number.
  1. If the random number lies between 1 and 899, the value of UV30 will be the third last digit of the random number plus 1 (example: 004 becomes 1, 075 becomes 1, 321 becomes 4, 899 becomes 9.
  2. If the random number lies between 900 and 989, the value of UV30 will be the second last digit of the random number plus 11 (example: 902 wird 11, 956 becomes 16, 989 becomes 19.
  3. If the random number lies between 990 und 1000, the value of UV30 will be the last digit of the random number plus 20 (example: 990 becomes 20, 905 becomes 25, 999 becomes 29 and 1000 becomes 20.

  Please note, that the rules mentioned above imply, that UV30 will never evaluate to 30.

- **LV1000** has to be loaded with integers between 1 and 1000. First, generate a random number between 1 and 100. In case the number divided by 2, 3, 5, or 7 has no remainder, an SQL null value is to be stored, otherwise die random number generated in the first step.

- **STADT100** contains character string values, selected from 20 predefined character strings. For this purpose, define an array of string constants in the program. The value of a tuple will be chosen randomly from the array. An equal distribution of values will result from that procedure. Make the names of 20 different cities the constants of the array.

- **DAT100** contains dates equally distributed (random number generator) from today to 99 days from now. In Oracle you may use the SYSDATE function. Tomorrow's date, for example, is given bySYSDATE+1-

5. Define all the constraints for the tables that you think are useful. This facilitates the verifiability of your program and ensures a systematic program design.

6. After completion of the loading process the result has to be documented by one or several SQLqueries, listing
   - count and count unique for all columns,
   - maximum and minimum values of all numerical columns and
   - the number of columns contained in the table.

7. Run the program several times (with 1000, 10000, 100000 tuples) and name the tables R1K, R10K, R100K, respectively and measure the time for inserting all lines printed at the end.