DB2

Forum: https://forum-db.informatik.uni-tuebingen.de/c/ss20-db2

Assignment 10 (7.7.2020)

Submission: Tuesday, 14.7.2020, 10:00 AM

Relevant videos: up to `DB2 - Chapter 12 - Video #71`.

https://tinyurl.com/DB2-2020

1. [15 Points] **Replacement Sort**

   Replacement Sort is an algorithm that can create longer runs in External Merge Sort's initial *Pass* 0. Your task is to implement *Replacement Sort* in C, based on Knuth's algorithm (see below).

   For simplicity, we assume that each buffer page can only hold a single *record*, consisting of a single natural number. The program's input is a file containing an unsorted, line-separated sequence of singleton `integer` values, each representing a record (see file `input.seq` for an example). Its output is a bunch of sorted `.run`-files with the same structure.

   File `rpsort.c` provides you with a framework of the program. You have to complete it by implementing function:

```
/* Implementation of replacement sort ala Knuth, TAoCP, volume 3 */
void replacement_sort(slot *buffer, int buffer_size, FILE *input_seq, FILE
    *output_run) {
    /*** YOUR CODE HERE ***/
}
```

   (a) Implement the algorithm as given below. Stick to the libraries imported and functions provided by `rpsort.c`.[1] Note that **solutions that do not compile will be graded with zero points**. Also, your solution should process the given example file `input.seq` without unexpected errors.

   Replacement Sort Algorithm ala Knuth:
   Operates on a block of `buffer_size` memory *slots*. Each slot can store a single *record* value and has a *state* that is either `ON` or `OFF`.

   Step 1: The `buffer_size` slots are filled with records from the input to be sorted.

   Step 2: All slots are put into the `ON` state.

   Step 3: Select the slot which has the smallest value of all `ON` slots.

   Step 4: Transfer the contents of the selected slot to the output (call its value *min*).

   Step 5: Replace the contents of the selected slot by the next input record:

   - If new record value $>$ *min*, go to **Step 3**.
   - If new record value $=$ *min*, go to **Step 4**.
   - If new record value $<$ *min*, go to **Step 6**.

   Step 6: Turn the selected slot `OFF`.

   - If all slots are now `OFF`, a sorted run is completed: Start a new run and go to **Step 2**.
   - Else, go to **Step 3**.

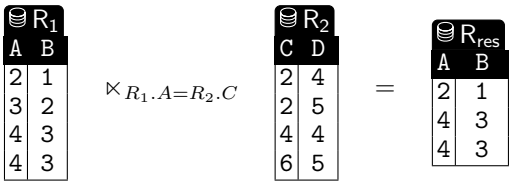
   (b) **Experiment:** Use your implementation to create the output runs for input `input.seq` (100000 records) and `buffer_size`s of 100, 1000 and 10000 slots. For each `buffer_size`, what is the average *run size*?

[1] Of course you are still allowed to define additional functions to support your implementation of `replacement_sort()`.

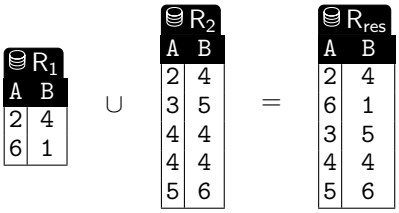
2. [15 Points] **Hash and Sorting-based Implementation of Relational Operators**

The following examples outline the semantics of the relational algebra operators *Left Semi Join* ($\ltimes_{a=b}$) and *Relational Union* ($\cup$):

- *Left Semi Join* ($\ltimes_{a=b}$): The result contains every row of R_1 that finds **at least one** join partner in R_2:



- *Union* ($\cup$): The result contains all rows of R_1 and R_2 with **all duplicates removed** (even if one of the input table contains duplicates):



For *both* operators, formulate a *hash-* and a *sorting-based* algorithm in pseudocode:

(a) `HashSemiJoin(`R_1`, `R_2`, `c_l`, `c_r`)`,

(b) `MergeSemiJoin(`R_1`, `R_2`, `c_l`, `c_r`)`,

(c) `HashUnion(`R_1`, `R_2`)`,

(d) `MergeUnion(`R_1`, `R_2`)`

Use pseudocode syntax based on the sample code found on Slide 15 (`MergeJoin`) and Slide 23 (`HashJoin`) of Chapter 12 (Joins). Do not use unusual operators in your pseudocode and include a short note if you assume input tables to be sorted.