Chair of Applied Computer Science III

University of Mannheim

Prof. Dr. Guido Moerkotte
Email: moerkotte@uni-mannheim.de

Daniel Flachs
Email: daniel.flachs@uni-mannheim.de

Database Systems II
Spring Semester 2019

Solution to Exercise Sheet 7

Created April 10, 2019

Exercise 1

In this exercise, you should familiarize yourself with lambda expressions in C++. They are used for the implementation of certain SQL queries in the subsequent exercise.

Exercise 1 a)

A lambda expression is an ad-hoc, locally scoped functor, i. e., an anonymous class that overrides the `operator()` member function. It behaves more or less like an anonymous inline function. Inform yourself about lambdas by reading the following blog article: `https://blog.feabhas.com/2014/03/demystifying-c-lambdas/`.

Exercise 1 b)

You are given a `std::vector<int>` called `vec`, i. e., a vector of integers, which you would like to print element-wise.

With `std::for_each` (included in the `algorithm` header), you can apply a function to all entries of a vector. It has the following signature:

```
Function for_each(InputIterator first, InputIterator last, Function fn);
```

where `first` and `last` are iterators of the first element and the element beyond the last element to be processed. `fn` is a unary function that accepts an element in the range as an argument. This can either be a function pointer or a function object (like a lambda). Its return value, if any, is ignored.

Complete the following code snippet such that it prints

```
-7, 5, 13, 8, -19, -1, -8, 128, 9, 4,
```

```
std::vector<int> vec = { -7, 5, 13, 8, -19, -1, -8, 128, 9, 4 };
std::for_each( vec.cbegin(), vec.cend(), /* Put your lambda here */ );
```

Solution

```
std::vector<int> vec = { -7, 5, 13, 8, -19, -1, -8, 128, 9, 4 };
std::for_each(vec.cbegin(), vec.cend(),
              [] (const int& elem) -> void { std::cout << elem << ", "; });
```

Exercise 1 c)

As before, you are given a `std::vector<int>` called `vec`, i. e., a vector of integers, but this time, you would like to modify it.

Use `std::for_each` and a lambda expression to modify each value of `vec` such that it is replaced by its absolute value, e. g., $-7 \to 7$, and $11 \to 11$. Use your 'print lambda' from the previous subtask to print the now modified vector.

Solution

```
std::vector<int> vec = { -7, 5, 13, 8, -19, -1, -8, 128, 9, 4 };
std::for_each(vec.begin(), vec.end(),
              [] (int& elem) -> void { elem = std::abs(elem); });
```

Exercise 1 d)

With so-called *captures*, variables outside the lambda expression can be passed to the lambda's body either by value or by reference, without the need to supply them as parameters. All variables that should be passed to the lambda need to be listed in the *capture list* at the beginning of the lambda definition: `[<comma-separated list of variables>]`. Putting just the variable name passes the variable by constant value, adding a `&` symbol passes the respective variable by reference.

Implement a lambda function that calculates the sum of all elements in the vector.

Solution

```
int sum = 0;
std::for_each(vec.cbegin(), vec.cend(),
              [&sum] (const int& elem) -> void { sum += elem; });
```

Exercise 2

This exercise builds upon the physical algebra implementation from exercise sheet 6 and uses the same simple main-memory database implementation. Sheet 6 was concerned with table scans, selections and projections, whereas this sheet deals with different implementations for joins.

Exercise 2 a)

Recap how the *Nested Loop Join (NLJ)* and the *Hash Join (HJ)* algorithm work[1]. Write down their pseudocode.

[1] For instance, read `https://en.wikipedia.org/wiki/Nested_loop_join` and `https://en.wikipedia.org/wiki/Hash_join`.

Solution

The following join algorithms compute the join $R \bowtie_p S$. For the Hash Join, the join predicate p is restricted to equality. By convention, for $R \bowtie^{\text{hj}} S$, the right relation S denotes the build relation (usually the smaller relation), the left relation R the probe relation. For two tuples x, y, $x \circ y$ denotes tuple concatenation.

```
NestedLoopJoin
   Input: two relations $R$ and $S$; a join predicate $p$
   Output: the tuples in $R \times S$ that satisfy $p$
1  for each tuple $r \in R$
2      for each tuple $s \in S$
3          if $p(r, s)$
4              output the tuple $r \circ s$
```

```
HashJoin
   Input: a build relation $S$, a probe relation $R$;
       a join predicate $p$; a hash function $h$
   Output: the tuples in $R \times S$ that satisfy $p$
1  Initialize $H$ to be an empty hash table with $h$ as a hash function.
2  for each tuple $s \in S$                // Build
3      Insert $s$ into $H[h(s)]$.
4  for each tuple $r \in R$                // Probe
5      for each $t \in H[h(r)]$
6          if $p(r, t)$
7              Output the tuple $r \circ t$.
```

Exercise 2 b)

List the pros and cons of both the Nested Loop Join and the Hash Join.

Solution

	Pros	**Cons**
NLJ	Can handle arbitrary join predicates. No materialization of intermediate results (tuples) necessary.	Slow runtime: $O(\lvert R\rvert \cdot \lvert S\rvert)$
HJ	Fast runtime: $O(\lvert R\rvert + \lvert S\rvert)$	In general, only equi-joins are possible. Pipeline breaker: Tuples must be materialized in hash table.

Exercise 2 c)

Download the `zip` archive from the website. Compared to the previous exercise sheet, only two classes for the Nested Loop Join and two classes for the Hash Join were added to `PhysAlgebra.hh`.

(i) Complete the `step()` member function of the `NestedLoopJoinInner` operator of the physical algebra.

(ii) Complete the `step()` member function of the `HashJoinProbe` operator of the physical algebra.

Solution

See code.

Exercise 2 d)

In the `database` subdirectory, you can find the two relations `persondb_persons` and `persondb_hometowns` with both their schema and their data. The former stores a person with an ID and a name, the latter assigns 0, 1 or more than 1 hometowns (rather: cities of residence) to a person using a foreign-key on the persons' IDs.

Implement the following query, which lists the hometowns of each person, once using a nested loop join and once using a hash join.

```
SELECT *
FROM persondb_persons AS p, persondb_hometowns AS h
WHERE p.ID == h.PERSON_ID;
```

Solution

See code.