# Project 2 – Advanced Databases

## Problem statement

We want to execute the following query:

(?a)---follows--->(?b)---friendOf--->(?c)---likes--->(?d)---hasReview--->(?e)

Over the given datasets that are described in the section “Dataset description”

In order to solve this query we need to be able to join two tables.

The required join algorithms are further described in the section Algorithm description

## Algorithm description

### Hash-join

Input two tables / list of rows /triples with one value for each table on which we join.

Join on last value of table1 and first value of table2

Use table1 to build dict with join attribute as key and list of rows that have it as values

Iterate over rows of table2 use join attribute to get all rows of table1 with value, join them together.

Return them

### Sort-merge-join

Input two tables / list of rows /triples with one value for each table on which we join.

Join on last value of table1 and first value of table2

Sort both tables on join key

Iterate over table1 and table2 by increasing the indices

If same values try to first increase indicee of table2 till not same value anymore then go back to previous indice of table2 + 1

Otherwise if value table1 smaller value table2 increase value table1

Otherwise if value table2 smaller value table1 increase value table2

### Skip sort merge join

Same as sort merge join but trying to increase by more than one

If one of the values in the table is smaller try to increase by more than 1 if it was too much increase by 1 instead.

### Join multiple tables

Takes care of the 3 required joins by calling the selected algorithm three times.

## Dataset description

The dataset consists of two files:

Each of them contains a triple with subject, property, object in each line.  
The size of the file and the number of relevant triples is as follows

File1 “100k.txt” with 109’0310 triples.

Num friendOf triples: 45712

Num follows triples: 31887

Num likes triples: 1032

Num hasReview triples: 1453

Upper limit join: 2’185’693’176’253’824

File2 “watdiv.10M.nt” with 10’916’458 triples.

Num friendOf triples: 4’491’142

Num follows triples: 3’289’307

Num likes triples: 112’401

Num hasReview triples: 149’634

Upper limit join: 248’462’961’060’947’778’084’996 ≈ 2,5 \* 10^23

In the data preprocessing step the triples get converted into a dictionary that has the properties as key with a table of the corresponding subject object table.

## Experiment and analysis

In order to join the 4 triples/tables we have to perform three joins.  
Below I listed the time required for each join as well as the required total time for each join algorithm.

### 100k.txt

#### Hash join

1. Join needed 0.73
2. Join needed 6.63
3. Join needed 76.08

Hash join query needed: 83.91 seconds to run the procedure

#### Sort merge join

### Join needed 1.96

### Join needed 5.06

### Join needed 17.51

Sort merge join query needed: 24.53 seconds to run the procedure

#### Sort merge join skip

1. Join needed 1.96
2. Join needed 4.09
3. Join needed 16.39

Sort merge join skip query needed: 22.80 seconds to run the procedure

In my experiment sort-merge join with skipping was the fastest

sort merge join is second

hash join is last

It is interesting to note that hash join was the fastest on the first join but much worse later on.  
This is most likely due to saving and retrieving ever growing lists when saving a new value in the dictionary.

Sort merge join and sort merge join skip are unsurprisingly similar in runtime and skipping values only really gives a benefit once the tables get bigger.

It would be interesting to look into a sort hash algorithm that gets rid of often retrieving and storing the lists in the dictionaries while not having to compare many values to get to the right one.

For the second part of the experiment on the “watdiv.10M.nt” file I had to split up the data since my Computer had a MemoryError after using about 16GB RAM and 40-50GB in the committed storage.  
  
This might affect the results but I don’t see any other way of doing it.

n\*log(n) != 100 \* (n/100 \* log(n/100)) = n \* log(n^100/100^100)  
In my case of 100 splits this would mean that the left side is about 100 times faster for n = 1000.  
  
This being said below the results follow.

#### Sort merge join

Sort merge join query needed: 3594.74 seconds to run the procedure

(Pretty much exactly one hour)

## Conclusion

Use sort merge join with skipping if you want the fastest join algorithm of the three selected