Chair of Data Science and Engineering
Department of Informatics
Technical University of Munich

Ecorrection
Place student sticker here

Note:

- During the attendance check a sticker containing a unique code will be put on this exam.
- This code contains a unique number that associates this exam with your registration number.
- This number is printed both next to the code and to the signature field in the attendance check list.

Foundations in Data Engineering

Exam:	IN2326 / Retake	**Date:**	Friday 9th April, 2021
Examiner:	Prof. Dr. Thomas Neumann	**Time:**	14:15 – 15:45

	P 1	P 2	P 3	P 4	P 5	P 6	P 7	P 8
I								
II								

Working instructions

- This exam consists of **16 pages** with a total of **8 problems**.
  Please make sure now that you received a complete copy of the exam.
- The total amount of achievable credits in this exam is 90 credits.
- Detaching pages from the exam is prohibited.
- Allowed resources:
  - **Any official lecture or exercise material from this winter term** (this exam is open-book)
  - A calculator or calculator website/tool/app.
- You can print, edit, scan and upload this document or fill out the document digitally. In this case, **do not** use the annotation or comment function of your PDF program, but use the text input fields provided in the exam PDF.
- **Answers are only accepted if the solution approach is documented.** Give a reason for each answer unless explicitly stated otherwise in the respective subproblem.
- Subproblems marked by * can be solved without results of previous subproblems.
- Do not write with red or green colors nor use pencils.
- **Do not** write into the credit boxes (☐☐).
- You **must not communicate with other students** in any way during the exam.
- We reserve the right for oral follow-up examination.

Left room from _____ to _____ / Early submission at _____

Problem 1 Unix Command Line Tools (8 credits)

The following questions cover *Unix command line tools*. You may use gnu coreutils and other programs that are commonly contained in Linux distributions.

0 1 2 3

a)*
For which type of data are the gnu coreutils suited for and what are their limitations in terms of data format? Explain in at most 3 sentences and name one format they are suited for and one format they cannot handle.

The gnu coreutils work on line-based text data. They can process raw texts and define delimiters, enabling them to understand the concept of records/tuples/words. However, they are not able to understand nested data structures like XML or JSON.
Supported formats: e.g., txt (raw texts), csv
Unsupported formats: e.g., xml, json

- 1p: work on line based data
- 1p: explain limitation
- 0.5p: supported format
- 0.5p: unsupported format

0 1 2

b)*
You are given a text file `students.txt`. Write a Bash command to change the name of student "Timo Brown" for all occurrences in the file to "Timo Gray".

```
sed -i 's/Timo Brown/Timo Gray/g' students.txt
```

- 1p: 0.5p replace file/in place (-i), 0.5p 'g'
- 1p: `sed`

0 1 2 3

c)*
The file `results.txt` stores the results of one alpine skiing race. Each row contains the following data: `race completion time`, `athlete id`, `athlete name` in this order. These are some entries:

```
54.91 37 Lena Duerr
53.88 90 Mikaela Shiffrin
54.37 39 Katharina Liensberger
54.23 45 Wendy Holdener
54.36 60 Petra Vlhova
```

Write a Bash command to determine the 3 fastest athletes and only output their id and name. You can assume that there are not duplicate race completion times.

```
sort -k 1 -n results.txt | cut -f 2-4 | head -3
```

- 1p: sort
- 1p: cut
- 1p: head

Problem 2 Performance Spectrum (11 credits)

The following questions test your understanding of the performance spectrum of modern computers.

a)* Program *P* compresses files stored on different positions on disk and writes the compressed files sequentially to a PCIe SSD. Program *P* needs 19s to process all files.
Estimate the size of the uncompressed data and show your calculation steps. You can assume that the program compresses each file by 50%. Ignore CPU costs for compression and assume the average file size is 12MB.

0 1 2 3 4 5

$numbImages * (0.01s \text{ seektime latency} + 12MB/200MB/s + 0.5 * 12MB/2Gb/s) = 19s$
$numbImages = 19s/(0.01s \text{ seektime latency} + 12MB/200MB/s + 0.5 * 12MB/2Gb/s)$
$totalDatasize = 260 * 12MB$

- 1p: using seektime (needed due to random accesses on disk)
- 1p: reading the files from disk
- 1p: writing the compresses files to PCIe SSD
- 1p: determine number of images
- 1p: calculate total datasize

b)*
When data processing tasks exceed your current machine, there are two options: *scale-up* or *scale-out*. Explain both approaches, each in one sentence.

0 1 2

Scale-up: Upgrade to a machine with more computation power (e.g., faster or larger processor) and faster hardware (PCIe SSD, persistent-memory, more DRAM...) -> largest upgrade is a supercomputer.
Scale-out: Distribute work over multiple machines (nodes) and add additional nodes to scale -> distributed system, cluster/cloud computing.

- 1p: Definition scale-up
- 1p: Definition scale-out

c)* Both approaches, scale-up, and scale-out, face challenges when scaling. Discuss for both approaches if they can scale indefinitely while being efficient. Explain for each approach why it can scale without limitation or discuss the limitation, e.g., naming the bottleneck.

0 1 2 3 4

Scale-up cannot scale indefinitely. Hardware costs and available maximum size of supercomputers is the limitation. At some point, the hardware costs are not efficiently anymore (not economic), or data size is too large, even for the biggest supercomputer's capacity.

Scale-out's limitation depends on the algorithms. Communication costs over the network are the performance bottleneck and increase with higher number of machines. At some point, communication is too expensive for most algorithms. Theoretically, for some highly independent algorithms, we could scale nearly without limitation.

- 2p: Scale-up
- 2p: Scale-out

Problem 3 SQL (I) (5 credits)

The questions of *Problem 3-5* test your knowledge of SQL. The tasks are split over three problems for reasons of correction.

The *FDE Institute* (FDEI) maintains a database containing data of Covid-19 tests. The database has the following schema and contains no `NULL` values.

```
CREATE TABLE People(
  id        int PRIMARY KEY,
  firstname text,
  lastname  text,
  sex       char(1),
  zipCode   int FOREIGN KEY
            REFERENCES
            City(zipCode)
);
```

```
CREATE TABLE City(
  zipCode    int PRIMARY KEY,
  name       text,
  population integer
);
```

```
CREATE TABLE PCRTests(
  personId      int FOREIGN
                KEY REFERENCES
                People(id),
  testDate      date,
  testPositive  bool,
  contactPerson int FOREIGN
                KEY REFERENCES
                People(id),
  PRIMARY KEY(personId,
              testDate)
);
```

```
CREATE TABLE RapidTests(
  personId      int FOREIGN
                KEY REFERENCES
                People(id),
  testDate      date,
  testPositive  bool,
  contactPerson int FOREIGN
                KEY REFERENCES
                People(id),
  PRIMARY KEY(personId,
              testDate)
);
```

In the tasks of *Problem 3-5*, we will query this database to analyze the data.

0 1 2 3 4 5

Determine the total number of people with a *positive PCR test* between July and October 2020 for each postal code area (`zip code`). You can write dates as follows `2020-2-30` (`Year-Month-Date`). Return the `zipCode` and number of positive PCR tests results.

```
Select zipCode, count(*)
From People p, PCRTests t
Where p.id = t.personId and
      t.testPositive = True and
      t.testDate between 2020-7-1 and 2020-10-30
Group by zipCode;
```

- 1p: for correct select and from
- 1p: count()
- 2p: where 0.5p per predicate (1p for correct between)
- 1p: group by

Problem 4 SQL (II) (13 credits)

Continuation of problem 3 using the same database.

Next, we want to use the data to trace infection chains.

1. Determine the entire infection chain caused by the person with id `165` (test date `2020-7-3`). Return first name, last name, and chaining degree for all people of the infection chain.
2. Also, *explain* whether your query contains cycles or terminates.

An infection chain works as follows: Person *A* is tested and has a positive *PCR or Rapid test* result. To trace the infection chain, all people *A* has met have to do a test and they name *A* as their *contact person* for the test. Assume *B* met *A*, person *B* now has `chaining degree` 1. If *B* tests positive, all people *B* met have to be tested (with `chaining degree` 2) and name *B* as *contact person*. This continues until no one tests positive. Only *positive* tested people count to *A*'s infection chain since negative tested people were not infected.

```
with positiveTests (personId, testDate, contactPerson) as (
        select *
        from PCRTests
        where testPositive = True
   union
        select *
        from RapidTests
        where testPositive = True
),
recursive infectionChain (personId, testDate, degree) as (
        select personId, testDate, 1
        from positiveTests t
        where contactPerson = 165
          and t.testDate > 2020-7-3
  union
        select personId, testDate, rec.degree + 1
        from positiveTests t, infectionChain rec
        where rec.personId = t.contactPerson
          and t.testDate > rec.testDate
)
select p.firstname, p.lastname, c.degree
from infectionChain c, people p
where p.id = c.personId;
```

- 2p: for positiveTests table (1p union, 1p both tables correct)
- 3p: recursion start (1p select with degree 1, 1p where contactPerson, 1p testdate)
- 1p: union
- 3p: recursion step (0.5p select with next degree, 0.5p from clause, 1p personID, 1p testDate)
- 1p: output select & from
- 1p: where p.id = c.personId

Second part:

- 1p: Due to the timestamps there is a total order and thus no cycles in the data.
- 1p: The data is finite due to the current date, thus the query has no cycles and will end.

Problem 5 SQL (III) (12 credits)

Continuation of problem 3 using the same database.

Eventually, the FDEI wants to analyze cities with high 7-day incidence.

Therefore, return *for each day* all cities with 7-day incidence higher than 75. For simplicity, you can assume each city only has one zip code. Return the date, city name, zip code, and the 7-day incidence value. The 7-day incidence is calculated as follows:

sum of positive PCR test over 7 days/(*city population*/100,000)

You can ignore corner cases for the first and last days of the data.

```
select i.day, c.name, c.zipCode,
       (i.cases / (c.population/100000.0)) as incidence
from City c,
        (select distinct p.zipCode, t.testDate as day,
                count(*) over (partition by p.zipCode
                               order by testDate
                               range between 6 preceding and current row
                              ) as cases
         from people p, pcrTests t
         where t.personId = p.id
           and t.testPositive = True) as i
where c.zipCode = i.zipCode
  and (i.cases / (c.population/100000.0)) > 75;

-- Alternative solution --

select c.name, c.zipCode,
       (i.cases / (c.population/100000.0))
from City c,
        (select i.zipCode, i.date,
                        sum(cases) over (partition by i.zipCode
                                         order by i.testDate
                                         range between 3 preceding and
                                                       3 following
                                        ) as total_cases
        from (select p.zipCode as zipCode, t.testDate as date,
                     count(*) as cases
              from people p, pcrTests t
              where t.personId = p.id
                and t.testPositive = True
              group by p.zipCode, t.testDate) as i
        ) as i
where c.zipCode = i.zipCode
  and (i.cases / (c.population/100000.0)) > 75;
```


- 1p: output select (day, city name, city zipCode, incidence)
- 1p: calculation with double value
- 2p: where clause (1p join zipCode, 1p >50)
- 2p: subquery select (1p distinct, 1p attributes)
- 4p: window function (1p each: sum, partition by, order by, range between)
- 2p: subquery where clause (1p id, 1p infected)

Problem 6 MapReduce (13 credits)

The following questions test your command of the Map-Reduce concept.

The recent blockade of the Suez canal by Ever Given has caused financial damage to many companies. In this task, we will query a container ship data set using Map-Reduce.
The *data set S* contains one entry for each trip of a container ship with the following columns:

Attribute	Description	Data Type
ShipID	Unique id of each ship.	integer
DepartureScheduled	Scheduled date of the ship's departure.	date
Departure	Actual date of the ship's departure.	date
ArrivalScheduled	Scheduled date of the ship's arrival.	date
Arrival	Actual date of the ship's arrival.	date
PortDepartureID	Unique id of the departure port of the trip.	integer
PortDestinationID	Unique id of the destination port of the trip.	integer
CargoValue	Value of the ship's cargo.	double
ShipCompany	Unique name of the ship company.	text

The date in *DepartureScheduled*, *Departure*, *Arrival* and *ArrivalScheduled* is stored in the format $(YYYY-MM-DD)$. You can use $>, <, =, != $ for date comparison and determine a duration by subtracting two dates.

You may also use the following functions:

- `days()`: converts a duration to days.
- `count()`: counts the number of input values.
- `max()`: returns the maximum of its input values.
- `min()`: returns the minimum of its input values.
- `sum()`: produces the sum of its input values.
- `asSet()`: returns the set of its input values.

Further, the port IDs are partitioned by continent:

Continent	ID-Range
Africa	[0-1140]
Asia	[1141-2380]
Australia	[2381-3130]
Europe	[3131-4020]
North America	[4021-5310]
South America	[5311-6940]

Write *Map-Reduce pseudo code* to answer the following questions. Use a *main function* to call all map and reduce functions.

a)*
Determine the average number of trips per container ship for each ship company.

```
def map(s):
        emit(s.ShipCompany, s.ShipID)

def reduce(shipCompany, shipID):
        numbTrips = count(shipID)
        numbShips = count(asSet(shipID))
        emit(shipCompany, numbTrips/numbShips)

def main():
        m = mapAll(S, map)
        return reduceAll(m, reduce)
```

- 0.5p: map emit
- 0.5p: count numbTrips
- 1p: numbShips (0.5p asSet + 0.5p count)
- 0.5p: average (numbTrips/numbShip)
- 0.5p: reduce emit
- 1p: for main

0 1 2 3 4

b)*
Due to the Suez canal incident a lot of ships were delayed, costing ship companies a lot of money.

Determine the financial damage of each ship company per destination port caused by Ever Given's accident.

A ship was delayed by the Suez canal incident if its arrival was scheduled between *March 24th* and *March 30th*, it was delayed by *more than 7 days*, and the ship traveled from *Europe to Asia* or *vice versa*. For simplicity, assume the costs of a delayed ship are *50%* of its cargo's value.
Only return ship companies that had total costs *higher than 300000$* by the incident.

Return for each company the company's name, the destination port, the company's total costs inflicted by the Suez canal incident, and share of costs for the destination port (i.e., costs of the company's ships traveling to this destination port / company's total costs).

```
def map1(s):
        if (days(s.ArrivalScheduled - s.Arrival) > 7 and
                s.ArrivalScheduled between 2021-03-23 and 2021-03-30 and
                ((s.PortDepartureID between 3131 and 4020 and
                  s.PortDestinationID between 1141 and 2380) or
                 (s.PortDepartureID between 1141 and 2380 and
                  s.PortDestinationID between 3131 and 4020))
           )
           emit(s.shipCompany, (s.PortDestinationID, s.CargoValue))

def reduce1(shipCompany, (destinationPort, cargoValue)):
        sumCosts = 0.5*sum(cargoValue)
        if(sumCosts > 300000$)
                emit(shipCompany, (destinationPort, cargoValue, sumCosts))

def map2(shipCompany, (destinationPort, cargoValue, sumCosts)):
        emit((shipCompany, destinationPort), (cargoValue, sumCosts))

def reduce2((shipCompany, destinationPort), (cargoValue, sumCosts)):
        emit((shipCompany, destinationPort),
             (sum(cargoValue)/sumCosts, sumCosts))

def main():
        mFiltered = mapAll(S, map1)
        rFiltered = reduceAll(mFiltered, reduce1)
        mPerPort = mapAll(rFiltered, map2)
        return reduceAll(mPerPort, reduce2)
```

- 4.5p map1: 0.5p if, 0.5p > 7 days, 1p time interval, 1p departure port, 1p arrival port, 0.5p correct emit
- 1.5p reduce1: 0.5p sumCosts, 0.5p filter, 0.5p correct emit
- 0.5p map2: correct emit (shipCompany, arrivalPort)
- 1.5p reduce2: 0.5p correct emit, 0.5p sum per port, 0.5p share per port
- 1p main: correct calls of all map and reduce functions + return

Problem 7 NeoJoin (11 credits)

The following questions test your knowledge of the NeoJoin.

The Relations *A* and *B* are distributed over three nodes, as shown in Table 7.1 and Table 7.2.

Node	x	hash(x)
1	18	0
1	19	2
1	21	0
2	15	0
2	22	1
3	18	0
3	22	1
3	17	2
3	1	1

Table 7.1: Distribution of Relation *A*.

Node	y	hash(y)
1	18	0
1	22	1
2	15	0
2	19	2
3	17	2
3	1	1

Table 7.2: Distribution of Relation *B*.

We want to join *A.x* with *B.y*. The data are already partitioned with the hash function *hash*(), as shown in the tables.
All nodes are connected by a star-shaped network with one central switch. Thus, the nodes are only connected to the switch, have the same bandwidth, and can send and receive one tuple per time slot.

a)*
Your task is to determine the *partition to nodes* assignment that *balances the amount of work on all nodes* and create a *minimal schedule* to move the tuples to their partition node. Use a notation like this for the schedule: **Time slot 1:** Node ? sends *x* = ... to Node ?, Node ? sends *x* = ... to Node ?,...

Partitioning:

Partition	0	1	2
Node 1	**3**	1	1
Node 2	2	1	**1**
Node 3	1	**3**	2

Partition to node assignment: **Node 1**: partition 0, **Node 2**: partition 2, **Node3**: partition 1

Time slot 1:

- Node 1: sends *x* = 19 to Node 2
- Node 2: sends *x* = 22 to Node 3
- Node 3: sends *x* = 18 to Node 1

Time slot 2:

- Node 1: sends *y* = 22 to Node 3
- Node 2: sends *x* = 15 to Node 1
- Node 3: sends *x* = 17 to Node 2

Time slot 3:

- Node 1: does nothing
- Node 2: sends *y* = 15 to Node 1
- Node 3: sends *y* = 17 to Node 2

- 1.5p: partition to nodes assignment (0.5p per assignment)
- 4.5p: schedule (0.5p per transferred tuple)
- 1p: minimal schedule

0 1 2 3 4 5 6 7

0
1
2
3
4

b)*
Instead of the *balanced amount of work on all nodes* strategy, we can also apply the *minimize network traffic* variant. Compare both approaches by explaining one advantage and one disadvantage of each strategy.

Balanced amount of work on all nodes:
Advantage: all nodes are utilized as much as possible to distribute computation (minimize idle node time) -> shorter computation phase.
Disadvantage: moves more tuples over the network as data locality is not utilized to the maximum -> longer data movement phase.

Minimize network traffic:
Advantage: completely utilizes data locality -> shorted data movement phase.
Disadvantage: computation is not distributed equally, thus nodes might be idle for a longer time or not used at all -> longer computation phase.

- 2p: balanced amount of work (1p advantage, 1p disadvantage)
- 2p: minimize network traffic (1p advantage, 1p disadvantage)

Problem 8 XML: XPath and XQuery (17 credits)

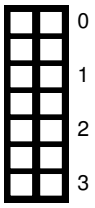
The following questions test your knowledge of XPath and XQuery.

The FDE university uses an XML file "uni" to store its data, and the file looks as follows (excerpt from the file):

```
<lectures>
  <lecture ID="IN2326">
    <titel>Foundations in Data Engineering</titel>
    <spw>2</spw>
  </lecture>
  <lecture ID="MA4800">
    <titel>Foundations in Data Analysis</titel>
    <spw>4</spw>
  </lecture>
  ...
</lectures>
<students>
  <student ID="M1234">
    <name>Alex</name>
    <semester>1</semester>
    <attends lectures="IN2326"/>
  </student>
  <student ID="M1235">
    <name>Timo</name>
    <semester>4</semester>
    <attends lectures="IN2326;MA4800;IN2219;IN2118"/>
  </student>
  ...
</students>
```

You can use XPath and XQuery to solve the following tasks.

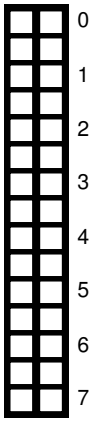
a)*
Return the name of the student with ID `M1424`.




```
doc('uni')//student[@id=M1424]/name
```

1p: navigate to student, **1p:** filter for correct id via attribute, **1p:** return the name

b)*
Write a XQuery query that returns the *sum* of *semester periods per week* (spw) for each student. Return each student's name and spw sum.
You can use functions like `min()`, `max()`, `sum()` for aggregations and `tokenize(data, 'separator')` to convert a string to an array/sequence using the separator.




```
for $s in doc('uni')//student
  let $spw_sum := sum(
    for $lID in tokenize($s/attends/@lectures,";")
      return doc('uni'//lecture/[@ID=$lID]/spw)
  )
  return <student>{$s/name}<sum>{$spw_sum}</sum></student>
```

- 1p: for loop over students (0.5p for, 0.5p path)
- 1p: variable for spw_sum
- 0.5p: sum aggregation
- 1.5p: nested for loop over lectures (0.5p for, 0.5p path, 0.5p tokenize)
- 1p: filter for id
- 1p: return spw (0.5p return, 0.5p path)
- 1p: return name (0.5p) and sum (0.5p)

c)*

We get the following XQuery query:

```
for $x in (1,2)
   return <spw semester="{$x}"> {
      for $s in doc('unknown.xml')/uni/st
         return <student name = "{$s/name/text()}"> {
            for $v in tokenize($s/plan/sem[@semNr=$x]/@lectures," ")
               return sum(doc('unknown.xml')/uni/lectures/lecture[@LID=$v]/@SPW)
         } </person>
   } </spw>
```

This query produces the following output for a unknown XML document *unknown.xml*:

```
<spw semester="1">
   <student name="Timo"> 2 <student/>
   <student name="Moritz"> 3 <student/>
</spw>
<spw semester="2">
   <student name="Timo"> 5 <student/>
   <student name="Moritz"> 2 <student/>
</spw>
```

Create a XML document that fits the output of the query.

```
<uni>
        <st>
                <name> Timo </name>
                <plan>
                        <sem semNr="1" lectures="V1"/>
                        <sem semNr="2" lectures="V2"/>
                </plan>
        </st>
        <st>
                <name> Moritz </name>
                <plan>
                        <sem semNr="1" lectures="V1 V2"/>
                        <sem semNr="2" lectures="V3"/ >
                </plan>
        </st>
        <lectures>
                <lecture LID="V1" SPW="2"/>
                <lecture LID="V2" SPW="3"/>
                <lecture LID="V3" SPW="2"/>
        </lectures>
</uni>
```

- 2p: first student (0.5p name, 0.5p xml structure, 0.5p semNr=1, 0.5p semNr=2)
- 2p: second student (0.5p name, 0.5p xml structure, 0.5p semNr=1, 0.5p semNr=2)
- 1.5p: correct lectures (0.5p structure, 0.5p LID, 0.5p SPW)
- 0.5p: correct amount of lectures
- 1p: correct XML syntax

Additional space for solutions–clearly mark the (sub)problem your answers are related to and strike out invalid solutions.