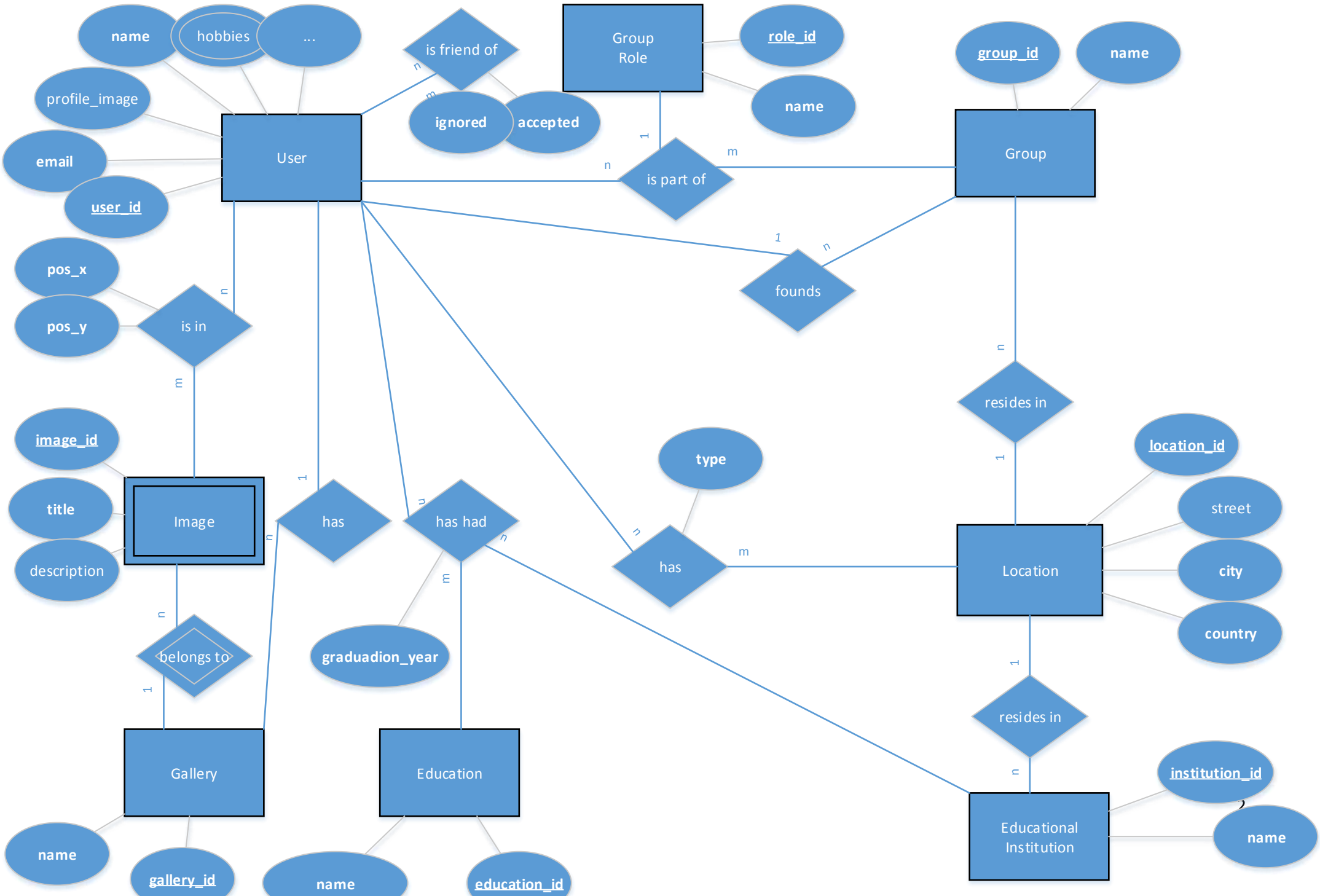
Übungsblatt 1

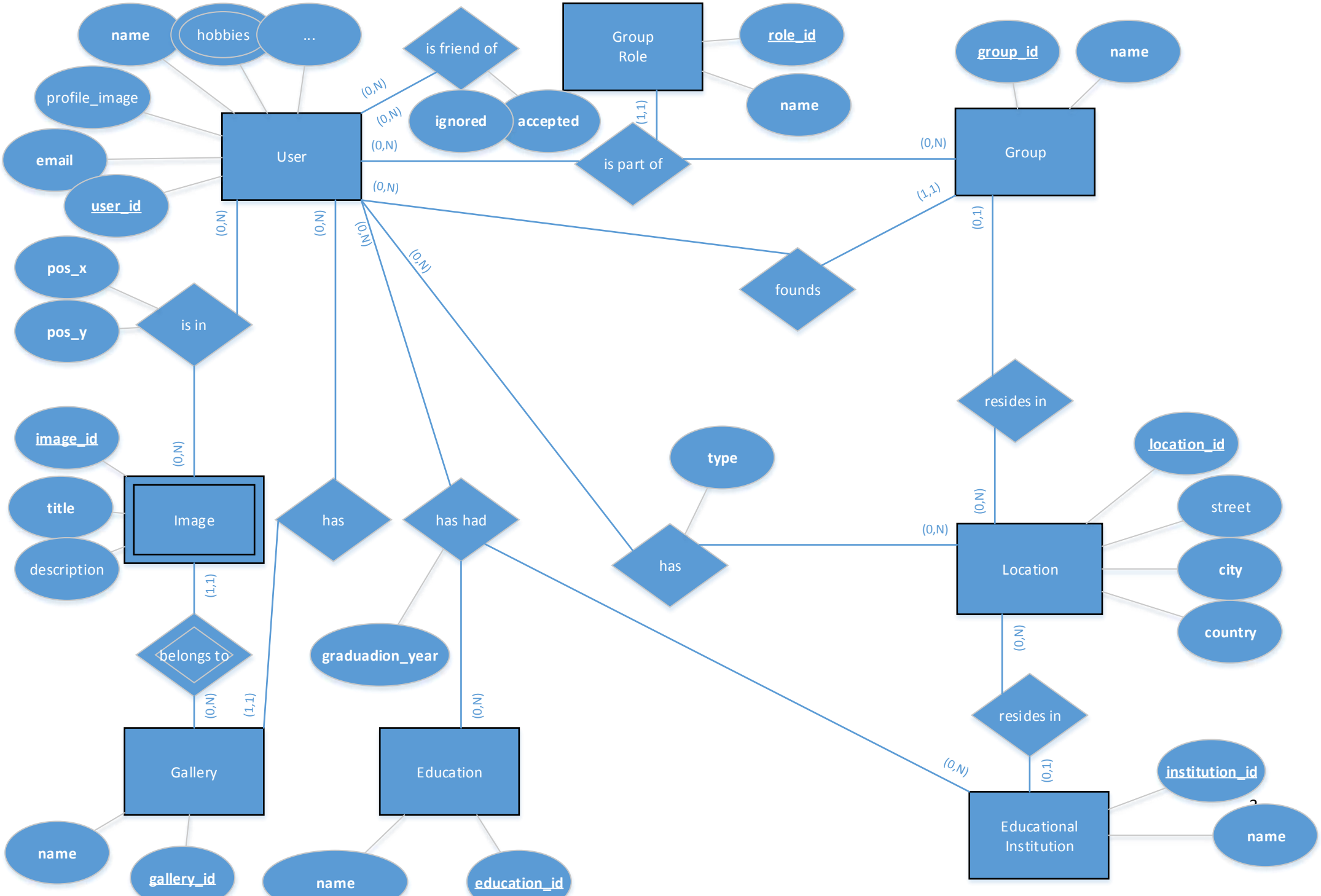
1. Modellierung & Mapping

a. Chen & Min-Max Diagramme

Die Diagramme befinden sich auf den nächsten beiden Seiten (zuerst Chen, dann Min-Max).

Bitte beachten: User hat zur Relation „is friend of“ nur einen Strich, obwohl hier eigentlich zwei sein müssten. Ich habe das aus Platz- und Formatierungsgründen so gemacht, damit mir die beiden Linien nicht zusammenfallen. Die Kardinalität ist n:m (Chen) bzw. (0,N)(0,N) (Min-Max) und befindet sich über und unter der Linie um sie auszudrücken.


name
hobbies
...
profile_image
email
user_id
User
is friend of
ignored
accepted
Group Role
role_id
name
is part of
Group
group_id
name
pos_x
pos_y
is in
founds
resides in
image_id
title
description
Image
has
has had
graduadion_year
type
has
Location
location_id
street
city
country
belongs to
Gallery
Education
resides in
Educational Institution
institution_id
name
name
gallery_id
name
education_id


name
hobbies
...
profile_image
email
user_id
User
is friend of
ignored
accepted
Group Role
role_id
name
is part of
group_id
name
Group
pos_x
pos_y
is in
founds
resides in
image_id
title
description
Image
has
has had
type
has
location_id
street
city
country
Location
belongs to
graduadion_year
Gallery
Education
resides in
Educational Institution
institution_id
name
name
gallery_id
name
education_id
(0,N)
(1,1)
(0,1)

b. Verfeinerung

Ich habe das Modell in Aufgabe 1a nach bestem Wissen und Gewissen erstellt und weiß nicht genau, was ich hier genau tun soll. Meiner Meinung nach habe ich dabei die Regeln für gute E/R-Modellierung eingehalten und beachtet.

c. Relationales Schema

Relationen direkt aus den Entitäten, 1:n-Beziehungen bereits aufgelöst

```
Location: {[
            location_id: int,
            street: string,
            city: string,
            country: string
]}
```

Schlüsselkandidaten: {location_id}, {street, city, country}

Primärschlüssel: {location_id}

Sekundärschlüssel: {country, city}

```
User: {[
            user_id: int,
            email: string,
            profile_image: binary,
            name: string,
            hobbies: string
]}
```

Schlüsselkandidaten: {user_id}, {email}, {name}

Primärschlüssel: {user_id}

Sekundärschlüssel: {name}, {email}

Das Feld „hobbies“ ist mehrwertig und müsste durch eine eigene Tabelle ausgedrückt werden.

Group: {[group_id: int, name: string, user_id, location_id]}

Schlüsselkandidaten: {group_id}, {name, user_id}

Primärschlüssel: {group_id}

Sekundärschlüssel: {name}

GroupRole: {[role_id: int, name: string]}

Schlüsselkandidaten: {role_id}, {name}

Primärschlüssel: {role_id}

Sekundärschlüssel: {name}

EducationalInstitution: {[institution_id: int, name: string, location_id]}

Primärschlüssel: {institution_id}, {name,location_id}

Sekundärschlüssel: {name}

Education: {[education_id: int, name: string]}

Schlüsselkandidaten: {education_id}, {name}

Primärschlüssel: {education_id}

Sekundärschlüssel: {name}

Gallery: {[gallery_id: int, name: string, user_id]}

Primärschlüssel: {gallery_id}, {user_id, name}

Sekundärschlüssel: {user_id, name}

Image: {[image_id: int, title: string, description: string, gallery_id]}

Schlüsselkandidaten: {image_id,gallery_id}

Primärschlüssel: {image_id, gallery_id}

Sekundärschlüssel: {title}

Relationen aus den Beziehungen

`UserGroupRole: {[user_id, group_id, role_id]}`

Primärschlüssel: `{user_id, group_id, role_id}`

Sekundärschlüssel: {}

`UserLocations: {[user_id, location_id, type: string]}`

Schlüsselkandidaten: `{user_id, location_id}`

Primärschlüssel: `{user_id, location_id}`

Sekundärschlüssel: `{type}`

Der Typ der UserLocation müsste auch über eine Tabelle aufgelöst werden, da sonst in der Datenbank die Typen nicht validiert werden können. Ansonsten muss das in der Applikation geschehen, was aber unsauber ist.

`UserEducations: {[user_id, institution_id, education_id, graduation_year: date]}`

Primärschlüssel: `{user_id, institution_id, education_id}`

Sekundärschlüssel: {}

`UserTaggedImages: {[user_id, gallery_id, image_id, pos_x: int, pos_y: int]}`

Primärschlüssel: `{user_id, gallery_id, image_id}`

Sekundärschlüssel: {}

2. SQL

a.

```
SELECT
  `fh1`.`name` AS `Von`,
  `fh2`.`name` AS `Nach`,
  DATE_FORMAT(`f`.`abflug`, '%k:%i') AS `Uhrzeit`
FROM
  `flug` AS `f`
  INNER JOIN `flughafen` AS `fh1`
    ON `f`.`von` = `fh1`.`flughafen_id`
  INNER JOIN `flughafen` AS `fh2`
    ON `f`.`nach` = `fh2`.`flughafen_id`
WHERE
  `fh1`.`iata` = 'TEE'
  AND
  `f`.`abflug` BETWEEN CAST('2011-08-01' AS DATETIME) AND CAST('2011-08-30'
AS DATETIME)
;
```

Hier gibt es keine Ergebnistupel weil vom Flughafen „TEE“ keine Flüge ausgehen.

b.

```
SELECT
  `p`.`vorname`,
  `p`.`nachname`,
  `pd`.`geburtsdatum`
FROM
  `passagier` AS `p`
  INNER JOIN `passagierdetails` AS `pd`
    ON `p`.`passagier_id` = `pd`.`passagier_id`
  INNER JOIN `buchung` AS `b`
    ON `b`.`passagier_id` = `p`.`passagier_id`
  INNER JOIN `flug` AS `f`
    ON `f`.`flug_id` = `b`.`flug_id`
  INNER JOIN `fluglinie` AS `fl`
    ON `fl`.`fluglinie_id` = `f`.`fluglinie_id`
WHERE
  `fl`.`firmenname` = 'Australia Airlines'
  AND
  `pd`.`geburtsdatum` BETWEEN CAST(CURDATE() - 250000 AS DATE) AND
CAST(CURDATE() - 200000 AS DATE)
;
```

c.

```
SELECT
  `p`.*
FROM
  `passagier` AS `p`
  INNER JOIN `buchung` AS `b`
    ON `b`.`passagier_id` = `p`.`passagier_id`
  INNER JOIN `flug` AS `f`
    ON `f`.`flug_id` = `b`.`flug_id`
  INNER JOIN `flugzeug` AS `fz`
    ON `fz`.`flugzeug_id` = `f`.`flugzeug_id`
  INNER JOIN `flugzeug_typ` `ft`
    ON `ft`.`typ_id` = `fz`.`typ_id`
WHERE
  `ft`.`bezeichnung` = 'Airbus 380'
;
```

Die Bezeichnung „Airbus 380“ stimmte in der Angabe nicht, da er in der Datenbank als „Airbus A380“ abgespeichert ist.

d.

```
SELECT
  *
FROM
  `flugzeug` AS `fz`
WHERE
  `fz`.`flugzeug_id` NOT IN (SELECT `flugzeug_id` FROM `flug`)
;
```

Korrelierte Queries sind nicht gut für die Performance, deswegen habe ich diese Aufgabe zunächst mit einem normalen Subquery gelöst. Hier nachfolgend noch eine Lösung mit einem korrelierten Query, die ich allerdings nicht testen konnte, da PHPMyadmin dabei out of sync geht.

```
SELECT
      *
FROM
      `flugzeug`
WHERE
      (
            SELECT
                  `flug`.`flugzeug_id`
            FROM
                  `flug`
            WHERE
                  `flug`.`flugzeug_id` = `flugzeug`.`flugzeug_id`
      ) IS NULL
;
```

e.

```
SELECT
  MAX(`fz`.`kapazitaet`)
FROM
  `flugzeug` AS `fz`
  INNER JOIN `flug` AS `f`
    ON `f`.`flugzeug_id` = `fz`.`flugzeug_id`
  INNER JOIN `fluglinie` `fl`
    ON `fl`.`fluglinie_id` = `f`.`fluglinie_id`
WHERE
  `fl`.`firmenname` = 'Thailand Airlines'
;
```

f.

```
SELECT DISTINCT
      `fh1`.`name` AS `Startflughafen`,
      `fh2`.`name` AS `Direkt erreichbar`,
      IFNULL(`fh3`.`name`, "") AS `Erreichbar über zwei Hops`,
      IFNULL(`fh4`.`name`, "") AS `Erreichbar über drei Hops`,
      IFNULL(`fh5`.`name`, "") AS `Erreichbar über vier Hops`
FROM
      `flughafen`       AS `fh1`
      INNER JOIN `flug`      AS `f1`  ON `f1`.`von`          =
`fh1`.`flughafen_id`
      INNER JOIN `flughafen`  AS `fh2` ON `fh2`.`flughafen_id` =
`f1`.`nach`
      LEFT JOIN `flug`       AS `f2`  ON `f2`.`von`          =
`fh2`.`flughafen_id`
      LEFT JOIN `flughafen`  AS `fh3` ON `fh3`.`flughafen_id` = `f2`.`nach`
      LEFT JOIN `flug`       AS `f3`  ON `f3`.`von`          =
`fh3`.`flughafen_id`
      LEFT JOIN `flughafen`  AS `fh4` ON `fh4`.`flughafen_id` = `f3`.`nach`
      LEFT JOIN `flug`       AS `f4`  ON `f4`.`von`          =
`fh4`.`flughafen_id`
      LEFT JOIN `flughafen`  AS `fh5` ON `fh5`.`flughafen_id` = `f4`.`nach`
WHERE
      `fh1`.`flughafen_id` = 2233
;

SELECT DISTINCT
      `fh1`.`name` AS `Startflughafen`,
      `fh2`.`name` AS `Direkt erreichbar`,
      IFNULL(`fh3`.`name`, "") AS `Erreichbar über zwei Hops`,
      IFNULL(`fh4`.`name`, "") AS `Erreichbar über drei Hops`,
      IFNULL(`fh5`.`name`, "") AS `Erreichbar über vier Hops`
FROM
      `flughafen` AS `fh1`
      INNER JOIN `flugplan`  AS `f1`  ON `f1`.`von`          =
`fh1`.`flughafen_id`
      INNER JOIN `flughafen` AS `fh2` ON `fh2`.`flughafen_id` = `f1`.`nach`
      LEFT JOIN `flugplan`   AS `f2`  ON `f2`.`von`          =
`fh2`.`flughafen_id`
      LEFT JOIN `flughafen`  AS `fh3` ON `fh3`.`flughafen_id` = `f2`.`nach`
      LEFT JOIN `flugplan`   AS `f3`  ON `f3`.`von`          =
`fh3`.`flughafen_id`
      LEFT JOIN `flughafen`  AS `fh4` ON `fh4`.`flughafen_id` = `f3`.`nach`
      LEFT JOIN `flugplan`   AS `f4`  ON `f4`.`von`          =
`fh4`.`flughafen_id`
      LEFT JOIN `flughafen`  AS `fh5` ON `fh5`.`flughafen_id` = `f4`.`nach`
WHERE
      `fh1`.`flughafen_id` = 2233
;
```

Wieder keine Ergebnisse, da vom Flughafen mit der ID 2233 keine Flüge ausgehen.

Es ist unklar ob im Flugplan das „nach” sich auf die Endstation oder den letzten Flughafen bezieht, selbiges gilt natürlich für Flug.

Die Tabelle „flughafen_erreichbar” sieht aus, als würde sie das Ergebnis eines „SELECT INTO”-Querys sein, auf jeden Fall hilft sie nicht die Aufgabenstellung zu lösen.

3. Normalformen

R1

R1(A,B,C,D) mit FD1 = {A →B, B →C, A →D}

Schlüsselkandidaten: {A}

R1{A, B, C, D}

1.NF: Ja – Alle Attributwerte sind atomar laut Angabe.

2.NF: Ja – kein Attribut ist abhängig von einer Schlüsselkandidatenteilmenge.

3.NF: Nein – C ist von A über B transitiv abhängig.

BCNF: Nein – Relation ist nicht in 3.NF.

R2(A,B,C,D) mit FD2 = {AB →C, B →D}

Schlüsselkandidaten: {A,B}

R2{A, B, C, D}

1.NF: Ja – Alle Attributwerte sind atomar laut Angabe.

2.NF: Nein – D ist abhängig von Schlüsselkandidatenteilmenge (B).

3.NF: Nein – Relation ist nicht in 2.NF.

BCNF: Nein – Relation ist nicht in 3.NF.

R3(A,B,C,D) mit FD3 = {AB →C, AC →D, AD →B}

Schlüsselkandidaten: {A,B,C,D}

R3{_A_, _B_, _C_, _D_}

1.NF: Ja – Alle Attributwerte sind atomar laut Angabe.

2.NF: Ja – Es gibt keine Nichtschlüsselattribute.

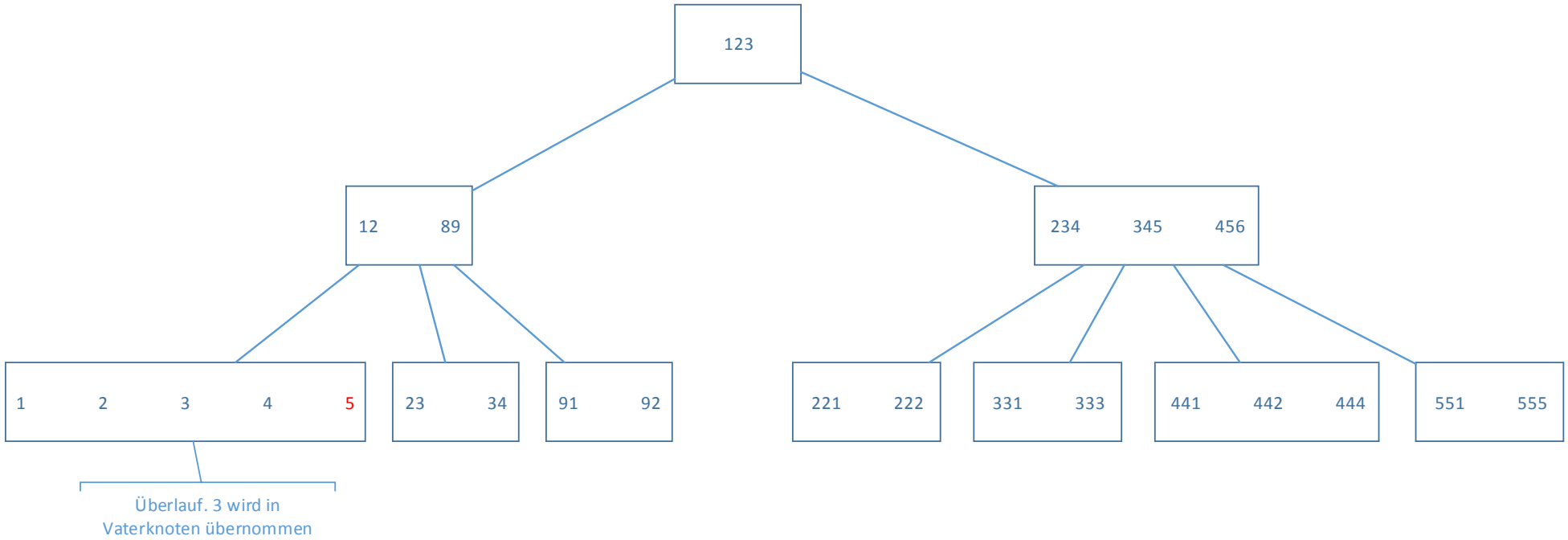
3.NF: Ja – Es gibt keine Nichtschlüsselattribute.

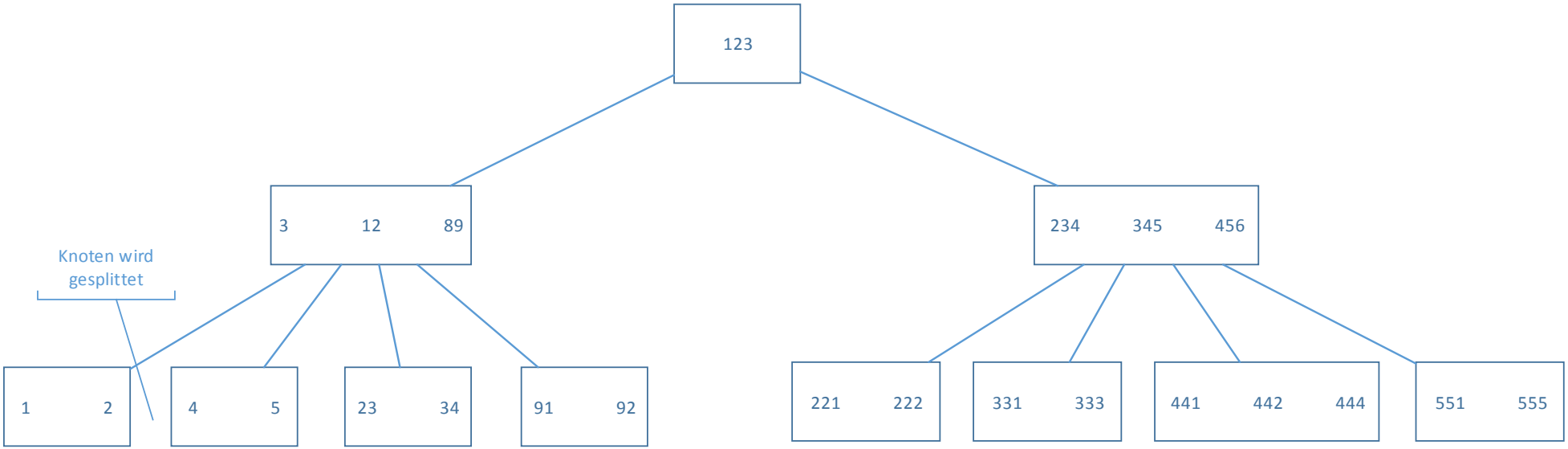
BCNF: Nein – keine Determinante ist Superschlüssel.

4. B-Baum

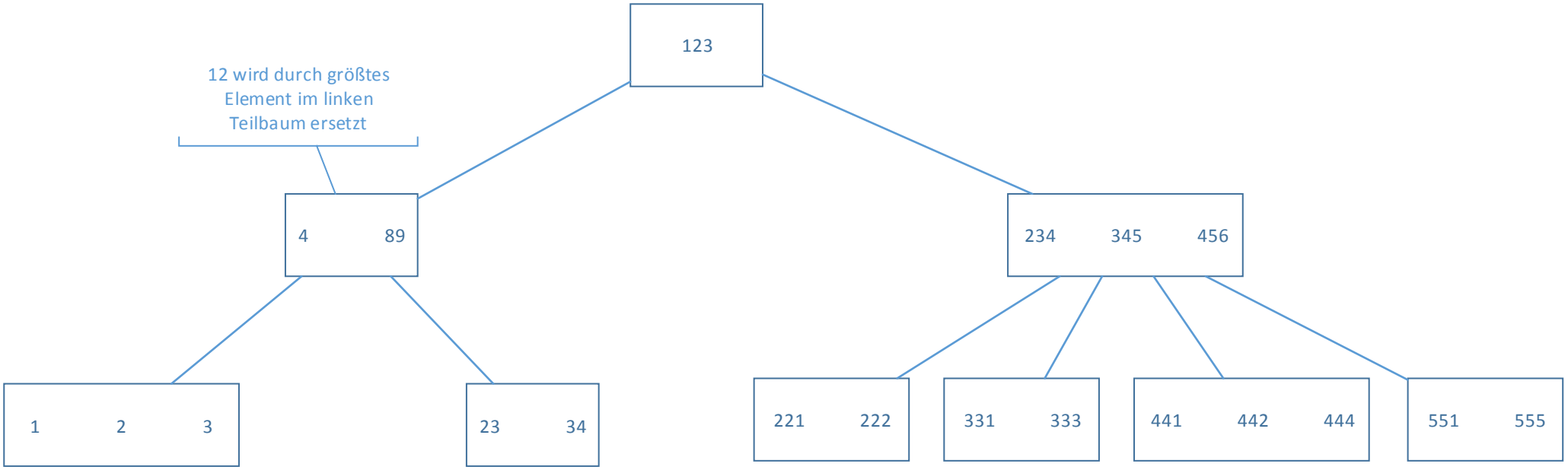
a.

5 einfügen

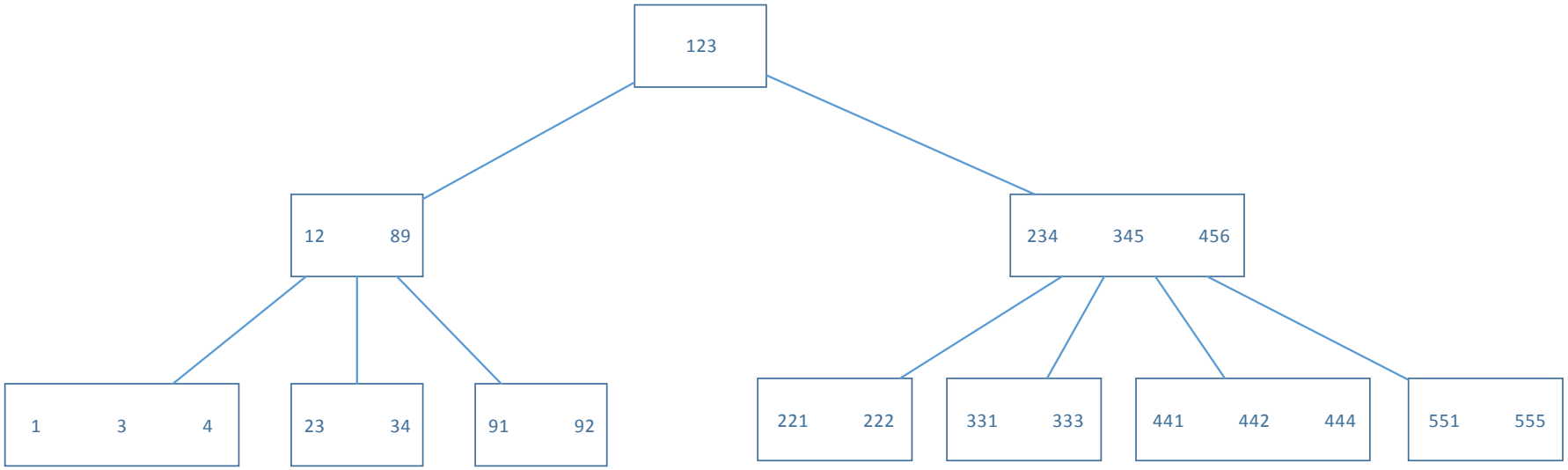



123
3 12 89
234 345 456
Knoten wird gesplittet
1 2
4 5
23 34
91 92
221 222
331 333
441 442 444
551 555

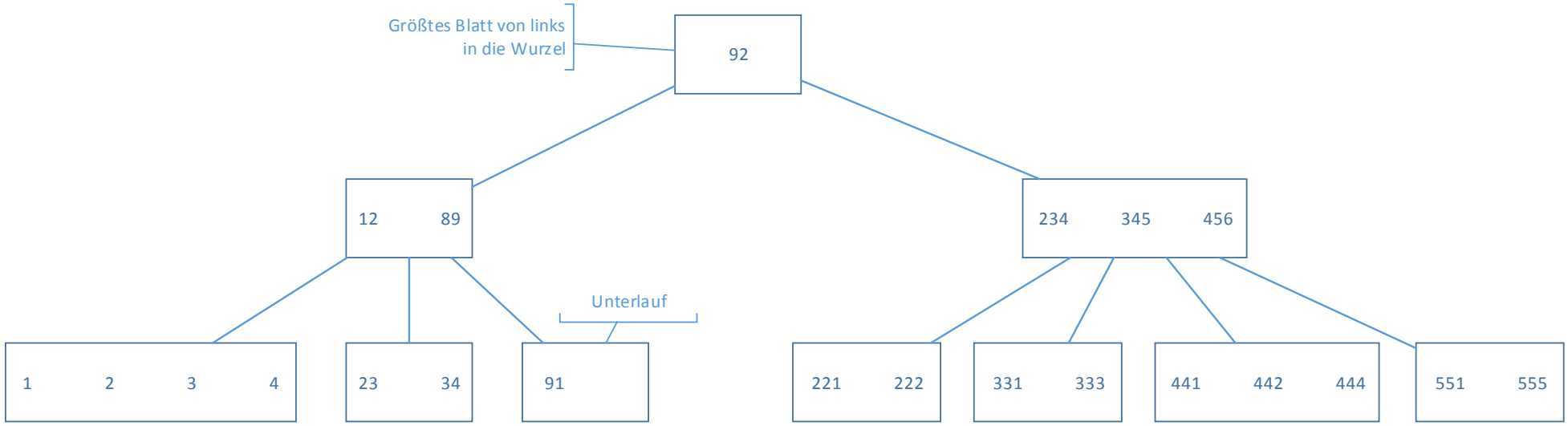
12 löschen

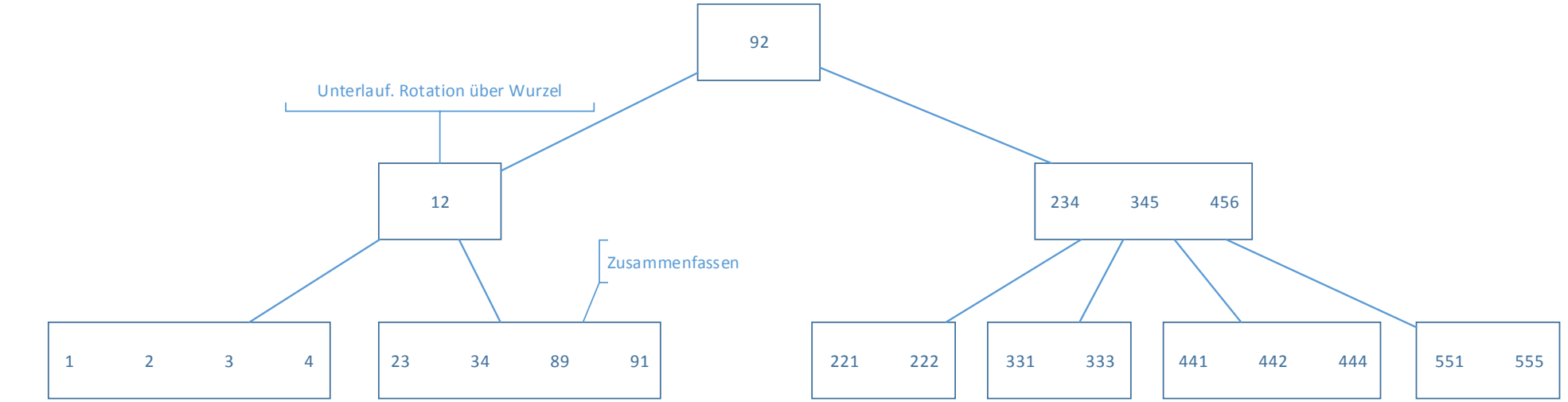

123
12 wird durch größtes Element im linken Teilbaum ersetzt
4 89
234 345 456
1 2 3
23 34
221 222
331 333
441 442 444
551 555

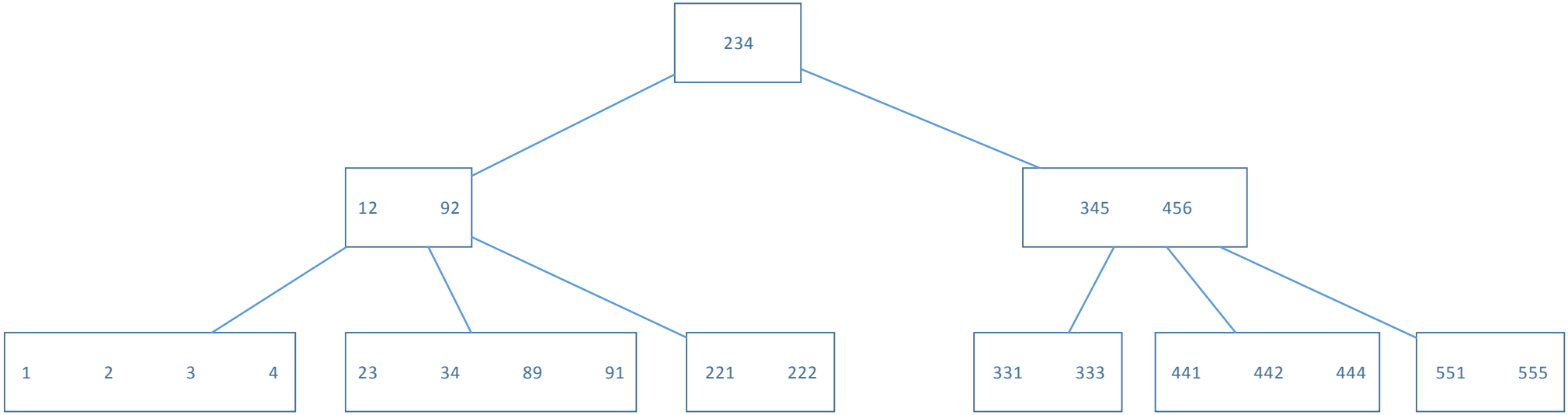
2 löschen


123
12 89
234 345 456
1 3 4
23 34
91 92
221 222
331 333
441 442 444
551 555

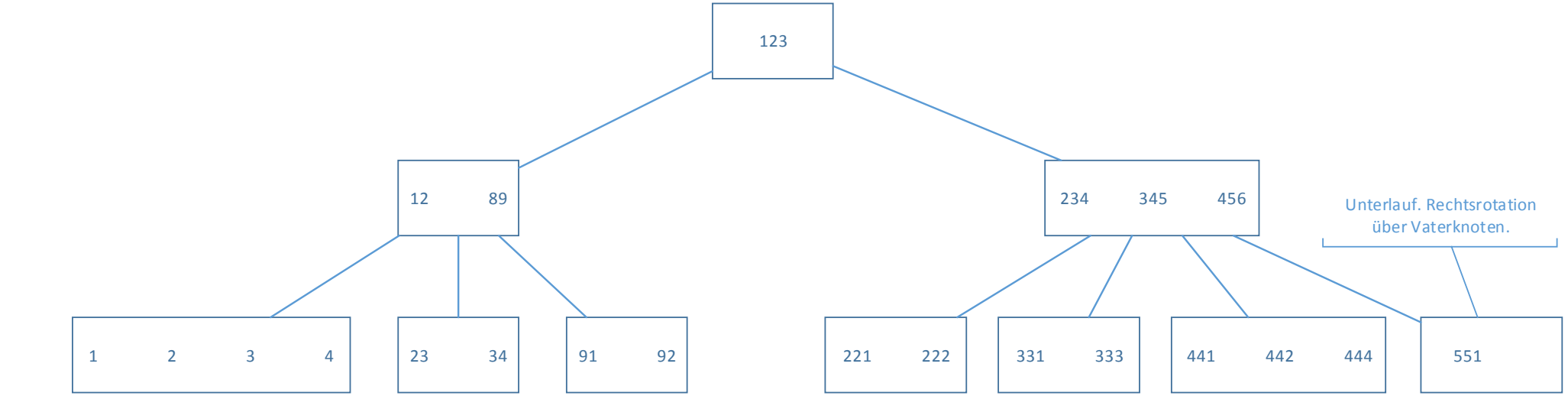
123 löschen

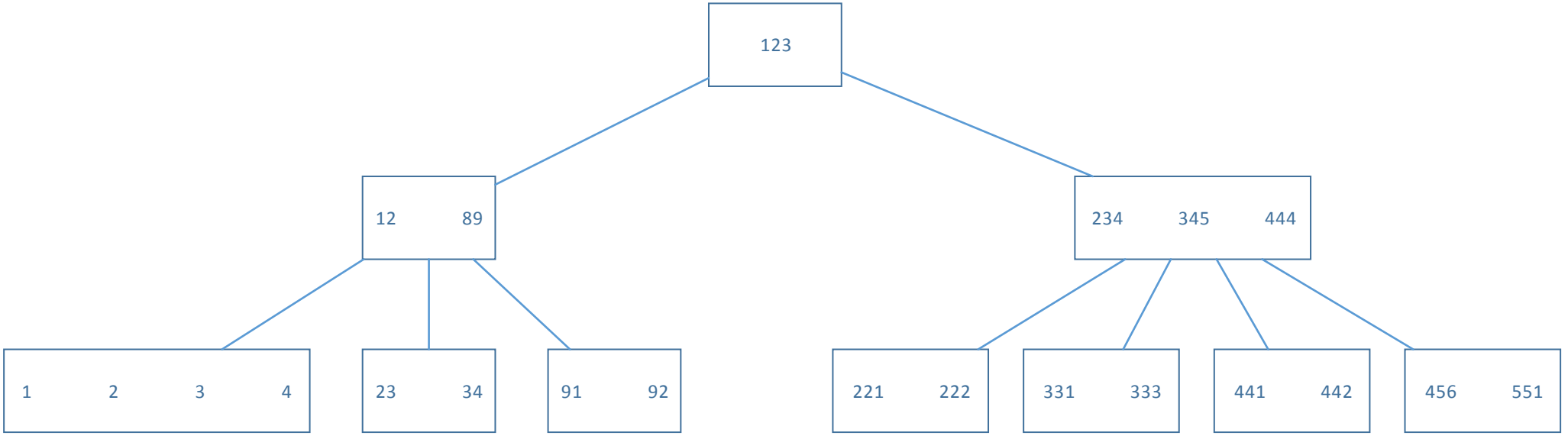

Größtes Blatt von links
in die Wurzel
92
12 89
234 345 456
Unterlauf
1 2 3 4
23 34
91
221 222
331 333
441 442 444
551 555


92
Unterlauf. Rotation über Wurzel
12
234 345 456
Zusammenfassen
1 2 3 4
23 34 89 91
221 222
331 333
441 442 444
551 555


234
12 92
345 456
1 2 3 4
23 34 89 91
221 222
331 333
441 442 444
551 555

555 löschen


123
12 89
234 345 456
Unterlauf. Rechtsrotation
über Vaterknoten.
1 2 3 4
23 34
91 92
221 222
331 333
441 442 444
551


123
12 89
234 345 444
1 2 3 4
23 34
91 92
221 222
331 333
441 442
456 551

b.

Eine Formel habe ich im Skriptum leider nicht gefunden dazu. Die folgende Formel habe ich aus einem Skriptum von Udo Kelter.

$$n = \left\lfloor \frac{b - t}{k + i + t} \right\rfloor$$
$$n = \left\lfloor \frac{2048 - 8}{20 + (20 - 8) + 8} \right\rfloor$$
$$n = 51$$

b Blockgröße in Bytes: 2048 B

t Platzbedarf für eine Medienadresse (Verweis auf Teilbaum): 8 B

k Platzbedarf für einen Schlüsselwert (nicht gegeben): 20 B

i Platzbedarf für Satzinhalt: 12 B

n Ordnung des Baumes: 51