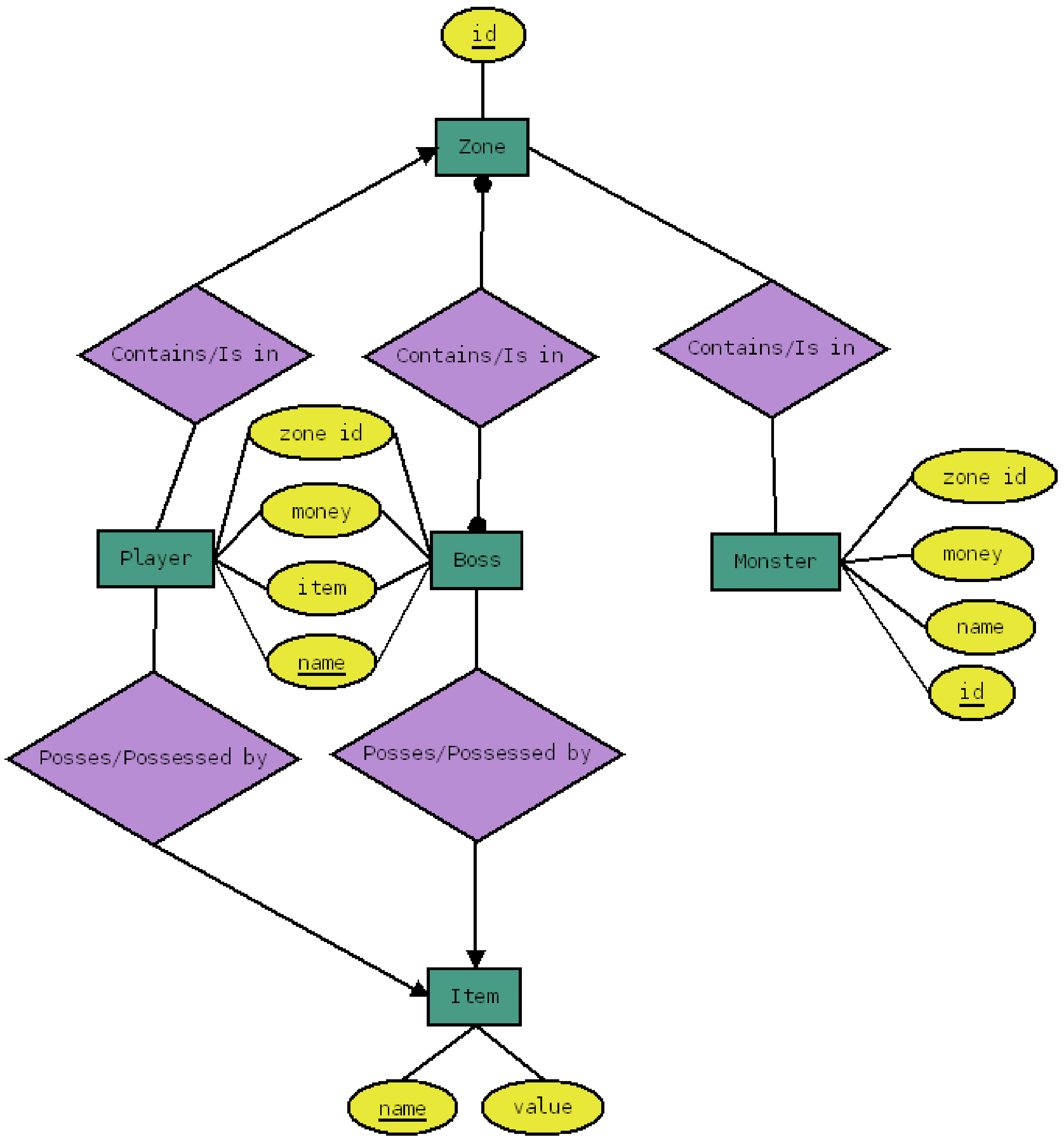
# DM505 – Database Design and Programming

Exam

Mads Petersen

## Task 1

### Task 1.a



**Player - Zone relation**, each player can be in 0 or 1 zone, since they can be offline or online, if they are offline they are in 0 zones, if they are online they are in max 1 zone. But each zone can have many players or no players.
**Boss - Zone relation**, Here it was decided that each boss is in exactly 1 zone and each zone has exactly 1 boss. From the described relationships each boss might be in more than 1 zone, but it makes sense to make each boss unique to its' zone to make the zone feel and the boss feel special to players.
**Monster - Zone relation**, As per the description each zone can have many monsters and each monster can be in many zones. Even though it isn't specifically mentioned it makes sense that some monsters might be in more than 1 zone, therefore it is decided to add an extra attribute to monster, id, name will then function more as a type of monster, and each monster will then have a unique id for tracking it. So e.g. if zone 1 has 4 zombies, it will be related to 4 monster objects with name "Zombie" but with different ids.
**Player - Item relation**, As per the description each player can have a max of 1 item, but each item can be on many players.
**Boss - Item relation**, Same as player - item.
**Monster - Item relation**, Even though the description states that a monster can have an item, the monster object does not have an item attribute, therefore it is decided that only bosses and players can have items.

## Task 1.b

Zone(
id:INTEGER
)

Player(
name:STRING,
zone id:INTEGER,
money: INTEGER,
item:STRING
)

Boss(
name:STRING,
zone id:INTEGER,
money: INTEGER,
item:STRING
)

Monster(
id:INTEGER,
zone id:INTEGER,
money: INTEGER,
name:STRING
)

Item(
name:STRING,
value: INTEGER,
)

# Task 2

## Task 2.a

For a set of attributes to be a key, they must functionally describe all other attributes in the relation and it must not be possible to remove any attribute from the set and still functionally describe all other attributes i.e. it must be minimal. Therefore it will be useful to investigate the closure of every attribute in the relation. But since there is only 1 FD that has a singleton left side, namely $E \rightarrow D$, it is only possibly to have a key consisting of a single attribute if that attribute is E. So it is only necessary to investigate the closure of E of all the single attribute possiblities. Doing so leads to:
X = {E}
Using $E \rightarrow D$, D is added to X
X = {E,D} No more rightsides can be used, therefore it is concluded that: $\{E\}^+ = \{E,D\}$ and therefore E by itself is not a key. Of the two attribute possibilities there are: AB, AC, AD, AE, BC, BD, BE, CD, CE, DE. Immediately it can be seen from the previous conclusion about E, that $\{DE\}^+ = \{E\}^+ = \{E,D\}$ so it is not a key. AC has no initial FD to expand by, neither does AD or BD, Therefore the closure of the remaining possiblities are determined, in the same way as above:
$\{A,B\}^+ = \{A,B,C,D\}$
$\{A,E\}^+ = \{A,E,D\}$
$\{B,C\}^+ = \{B,C,D,A\}$
$\{B,E\}^+ = \{B,E,D\}$
$\{C,D\}^+ = \{C,D,A\}$
$\{C,E\}^+ = \{C,E,D,A\}$
Therefore it is concluded that there is no set of 2 attributes that is a key for the relation either. Moving on to 3 attribute keys, we see that $\{A,B\}^+$ only lags E and $\{C,E\}^+$ only B and $\{B,C\}^+$ also only lags E. And since $B \rightarrow B$ is always true in any relation containing B, and the same for E. If we add them we will get {A,B,E} and {C,E,B} as keys since {C,E,B} = {B,C,E}. Of the other 3 attribute possiblities
{A,B,C}
{A,B,D}
{A,C,D}
{A,C,E}
{A,D,E}
{B,C,D}
{B,D,E}
{C,D,E}
{A,B,C} which is not a key since $\{A,B\}^+$ contains C and {A,B,D} is not a key for the same reason. {A,C,D} is not key since $\{C,D\}^+$ contains A, {A,C,E} is not key since $\{C,E\}^+$ contains A. And {A,D,E} is not key since $\{A,E\}^+$ contains D, {B,C,D} is not key since $\{B,C\}^+$ contains D, {B,D,E}is not key since $\{B,E\}^+$ contains D, {C,D,E} is not key since $\{C,E\}^+$ contains D. And since all 2 attribute possiblities have been tested {A,B,E} and {C,E,B} must be minimal.

## Task 2.b

To be in BCNF every left side of any FD must be a superkey. And since none of the left sides contains any of the keys determined above it is not in BCNF. To be in BCNF we have to ensure that each left side is a superkey. We can see that if we make a relation $R_1$(A,B,C,D) then AB will be a key for that relation, and thereby also a superkey. But CD, will not be a superkey. Therefore it is further decomposed into $R_1$(A,B,C), now AB is a key, and the FD $AB \rightarrow C$ still holds as the only one. From here we can see that if we make a relation containing just the left and right side of each FD we will get relations that are in BCNF, therefore we end up with:
$R_1$(A,B,C)
$R_2$(B,C,D)
$R_3$(C,D,A)
$R_4$(E,D)

In each of the decomposed relations only a single FD hold, in $R_1$ AB $\rightarrow$ C, in $R_2$ BC $\rightarrow$ D, in $R_3$ CD $\rightarrow$ A and in $R_4$ E $\rightarrow$ D. And the the relations consist of only the contents of every FD therefore the left side of every FD will be a key and also a superkey.

## Task 2.c

It can be seen that R is not in 3NF since to be in 3NF either the left side of every FD must be a superkey or the right side must be prime(i.e. be a member of some key). AB $\rightarrow$ C doesn't break 3NF since C is prime, but BC $\rightarrow$ D does since D is not prime. CD $\rightarrow$ A doesn't break 3NF either since A is prime but E $\rightarrow$ D also breaks since D is not prime. Since 3NF is an expansion on BCNF therefore any relation that is in BCNF will also be in 3NF, therefore the relations presented in the previous task will be in 3NF.

## Task 2.d

The primary kinds of anomalies are:

- Redundancy
- Update anomalies
- Deletion anomalies

**Redundancy** If the relation R(A,B,C,D,E) is considered from the above tasks. It can be seen that since (A,B,E) is a key for the relation, then every entry that has unique combination of A,B,E must be unique. Therefore if an entry has identical A and B entries but different E entries, the data from A and B is repeated. E.g.

| A | B | C | D | E |
|---|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ | $e_1$ |

Now if another entry has to be added it could look like this:

| A | B | C | D | E |
|---|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ | $e_1$ |
| $a_2$ | $b_1$ | $c_2$ | $d_2$ | $e_2$ |

And since the key attributes (A,B,E), (B,C,E) are different, that means it has to be two seperate entries even though the data in B is repeated. And of course the data in B could be repeated in evey column entered.

**Update anomaly** If we use the table from above, we can see that if $b_1$ has to be changed, then it has to be changed in two seperate rows. And since $b_1$ could be repeated many more times it could potentially have to be updated in many rows and this might mean that it is forgotten in some rows leading to inconsistency.

**Deletion anomaly** If we insert another entry in the relation above:

| A | B | C | D | E |
|---|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ | $e_1$ |
| $a_2$ | $b_1$ | $c_2$ | $d_2$ | $e_2$ |
| $a_1$ | $b_3$ | $c_1$ | $d_1$ | $e_1$ |

Again the redundancy anomaly can be seen, but also if $b_3$ had to be deleted the keys for this row would be the same as for the first row and the row would be lost.

# Task 3

## Task 3.a

```
CREATE TABLE "Mechanic"
(
EID integer PRIMARY KEY REFERENCES Employees(EID),
HourlyPrice REAL NOT NULL CHECK (HourlyPrice > 10)
);
```

## Task 3.b

```
SELECT
Employees.Firstname,
Employees.Lastname
FROM
Employees,
Mechanic,
Repairs
WHERE
Repairs.PartsCost + Repairs.Hours * Mechanic.HourlyPrice
=
(SELECT MAX(Repairs.PartsCost + Repairs.Hours * Mechanic.HourlyPrice) FROM Repairs);
```

## Task 3.c

```
SELECT
AVG(Repairs.PartsCost) AS AverageCost
FROM Repairs
INNER JOIN Cars
ON Repairs.License = Cars.License
WHERE Repairs.License = Cars.License AND Cars.Year = 2000;
```

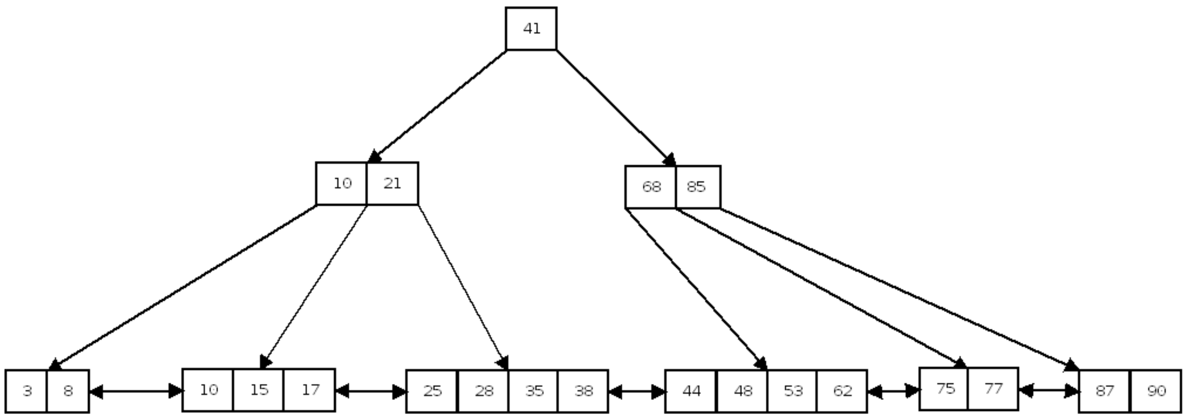
## Task 3.d

```
UPDATE Repairs
SET PartsCost = 100
WHERE License IN (SELECT License FROM Cars WHERE Brand = 'Mercedes')
```

## Task 4

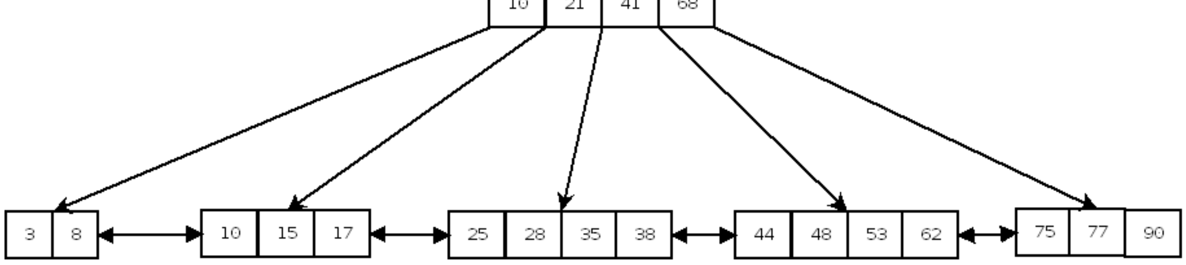
### Task 4.a

Since 17 is less than 21, it goes in the block that the pointer left of 21 points to. Because this block is full it must be split in 2, so it is split into the blocks (3,8) and (10,15), 17 is then inserted into the right most block to keep the left right ascending order resulting in the blocks (3,8) and (10,15,17). Since 10 is the new lowest reachable index in the new block, 10 is added to the parent block. Because this block is full, it is also split in two blocks (21,41) and (68,85) and 10 goes in the left block and 41 is moved up to the parent block and since there is none it becomes the new root. Resulting in the following tree:



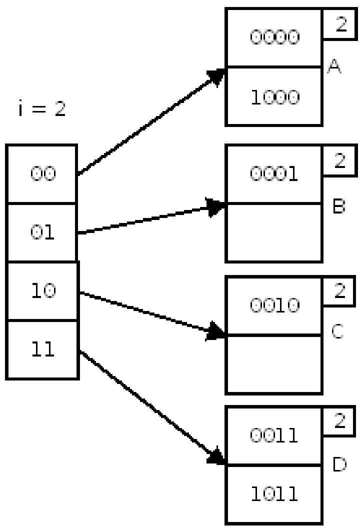
### Task 4.b

First 87 is found in the right most block of the leaf blocks, it can be observed that if 87 is deleted from its' block the block will have too few elements and therefore must be merged with the block to the left resulting in (75,77,90), now since a block has been deleted the parent block needs to be adjusted and 85 is deleted from it. resulting in just (68), this is not enough entries and the block must be merged with its' sibling resulting in (10,21,68), now the root is pulled down to level and the new root block is (10,21,41,68) and the final result looks like:



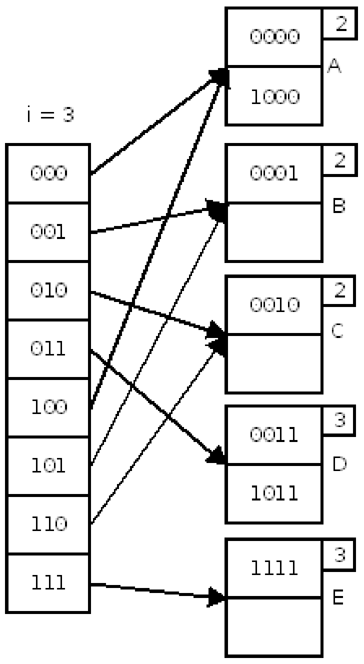
### Task 4.c

The key with hash 1000 hashes to bucket A and since A is full it is doubled and j is set to j+1. A bucket for hashes ending in 00 is created and a bucket for hashes ending in 10 is created. The already existing hashes 0000 and 0010 is split 1 in each. The new value is then put in the new bucket for hash values ending in 00. Resulting in the following:



## Task 4.d

Since this hash values goes into bucket D, and bucket D is full and i = j. The bucket has to be doubled and i has to be increased by one. The existing values stay in the bucket they are in and the new value hashes to the new bucket for hashes ending in 111. And the final result looks as follows:


i = 3
000
001
010
011
100
101
110
111
0000
1000
2
A
0001
2
B
0010
2
C
0011
1011
3
D
1111
3
E