Databaskonstruktion

1 Limitation of Diagram

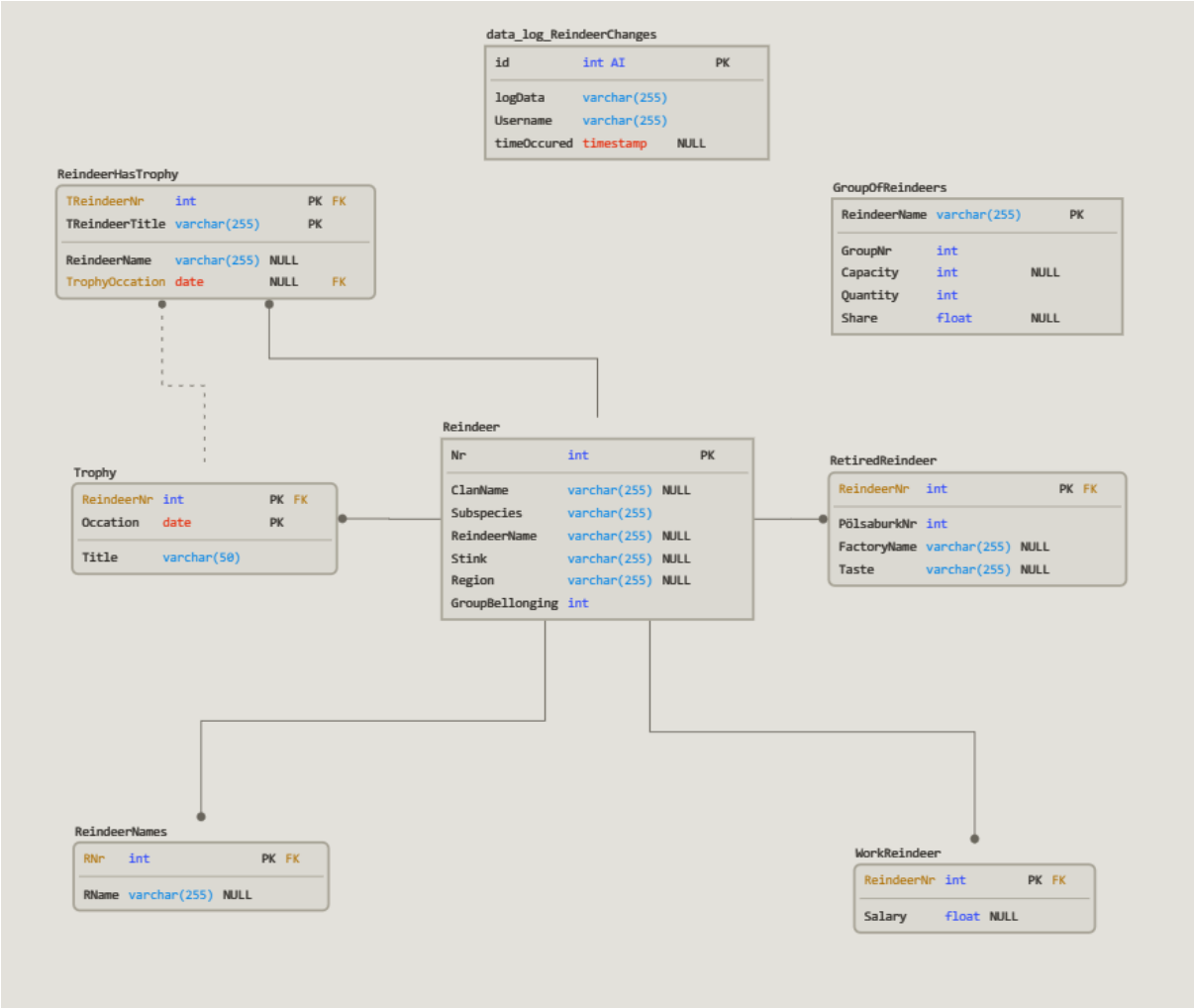


Figure 1. ER-diagram of Reindeer data cluster (*Main*).

The provided Entity-Relation diagram for the assignment has been restricted to the entities data clusters modeling the Reindeers with their respective children and dependencies.

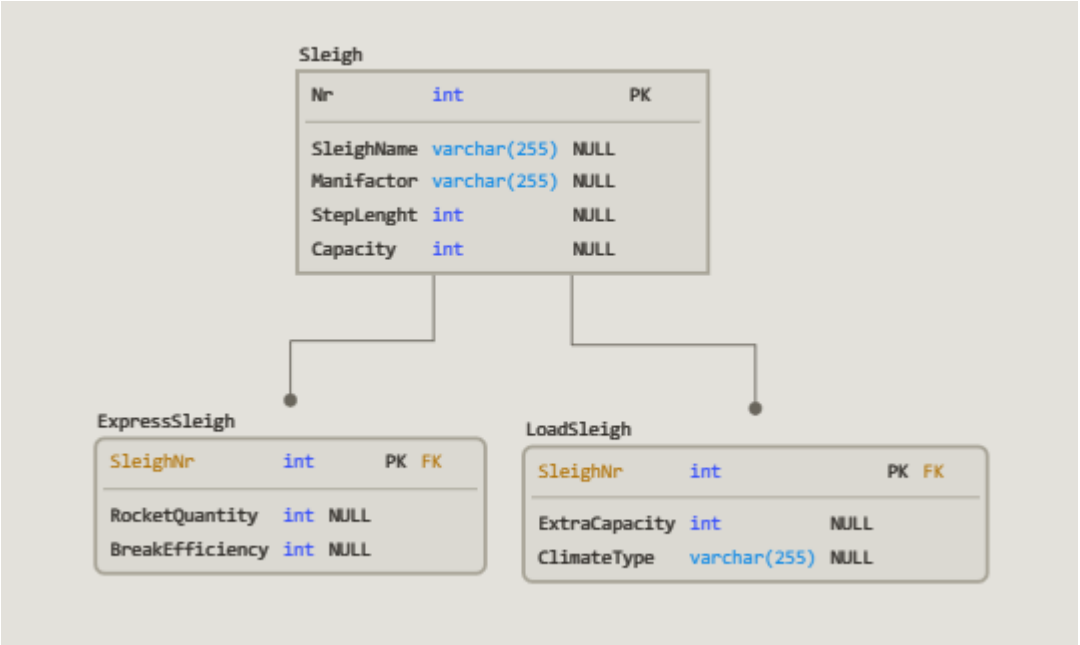


Figure 2. ER-diagram of Sleighs smaller data cluster.

With the logic in mind that reindeers also operate with sleighs, the smaller data cluster of Sleigh and its children are also included.
During the realization of the database, I found out that some new tables had to be included to implement some sort of additional functionalities.

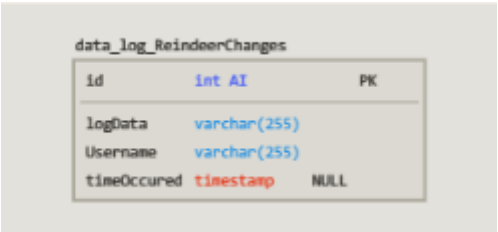


Figure 3. ER-diagram individual Log entity keeping track of activity (*Should be able to access and store logging data independently of class if implementing some form of abstract entity*).

Logging of activity is a fundamental part of database systems, having the ability to backtrack to where and who made a change is a cornerstone of maintaining and administering a database.
Lack of time has resulted that I could not create as many tables as I would like to log activity in the database. Therefore, the concept of having an abstract entity that has a blueprint of how the system should handle activity, will be further explored in the future.

2 Database Implementation

Varje större deluppgift skall helst ha ett eget kapitel.

Tänk på att när man diskuterar programkoden så skall man illustrera med små stycken programkod och med screenshots i de fall det är relevant.

Man väljer vilka stycken av programkod och vilka screenshots man tycker är viktiga för att diskutera i rapporten. Programkoden skall delas in i stycken och man skall inte göra ”appendix” där stora stycken programkod är inklippta utan förklarande text mellan varje stycke programkod.

Figurer med programkod skall inte överstiga en halv sida men kan vara så korta som en enda rad programkod. Figurer refereras med att man nämner figuren se figur 1. Figurerna skall blandas med text som ger en diskussion och förklaring av den programkod man har byggt. ***Programkod MÅSTE vara text som använder ett monotyp typsnitt (t ex Courier New) och använda sig av K&R indentering med obligatoriska måsvingar.***

```
function errorCallback(error)
{
      switch(errorCode){
            case: PERMISSION_DENIED:
                  // Geolocation API stöds inte; gör något
                  break;
            case: POSITION_UNAVAILABLE:
                  // Misslyckat positionsanrop
                  break;
            default:
                  // Okänt fel
                  break;
      }
}
```

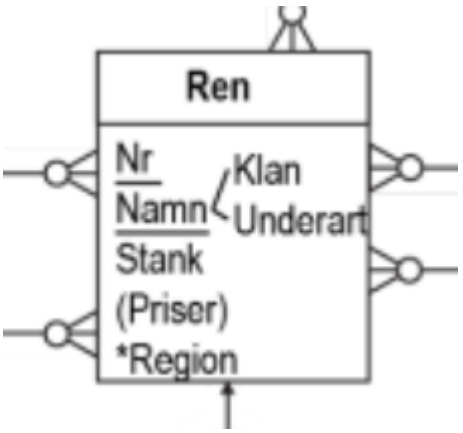
Figur 1 Exempel citerad programkod

2.1 Create Table

…

For the restricted part of the data clusters that I chose, present the data that are linked to Santa's reindeers. I began the realization of the database by creating the standard tables and defined the rows provided.

```
CREATE TABLE Reindeer(
    Nr int  unique not null,
    ClanName varchar(255),
    Subspecies varchar(255) not null,
    ReindeerName varchar(255),
    Stink varchar(255),
    Region varchar(255),
    GroupBellonging int not null,

    primary key (Nr)

)ENGINE=INNODB;
```



The table above represents the Reindeer entity. Note that design decisions were made that a reindeer was no longer defined by a unique primary key identified by their number AND name with clan and subspecies. This decision were because that I found that implementation of name containing these values is either redundant (because the number identifying should be enough just as a social security number, so it was denormalized one step backwards), or that reindeers could be inserted to the table containing no value for the clan as maybe clans are combined (and information on if two reindeers have a child from two different clans, what's the deciding factor for what clan should they be was not provided). Also we have learned in the past courses that names are not a great candidate key to choose because two individuals can have the same name, yet still be different individuals.
Note that another row is added which is *GroupBellonging* as this value stores the group that this particular reindeer belongs to.

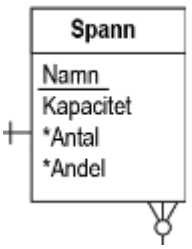
```sql
CREATE TABLE WorkReindeer(
    ReindeerNr int,
    Salary float,


    primary key(ReindeerNr),
    foreign key(ReindeerNr) REFERENCES Reindeer(Nr)

)ENGINE=INNODB;


CREATE TABLE RetiredReindeer(
    ReindeerNr int unique not null,
    PölsaburkNr int not null,
    FactoryName varchar(255),
    Taste varchar(255),

    primary key (ReindeerNr),
    foreign key (ReindeerNr) REFERENCES Reindeer(Nr)

)ENGINE=INNODB;
```

As seen in the provided ER-diagram we have a disjunction of inherited hierarchy. This means that a working reindeer can not also be a retired reindeer, as the specification says that once a reindeer becomes retired gets taken to the slaughterhouse and becomes pölsa.

```sql
CREATE TABLE Trophy(
    ReindeerNr int unique not null,
    -- YYYY-MM-DD --
    Occation DATE not null,
    Title varchar(50) not null,

    primary key (ReindeerNr, Occation),
    FOREIGN KEY (ReindeerNr) REFERENCES Reindeer(Nr)

)ENGINE=INNODB;
```

The table Trophy is created outside of the reindeer class, that is because the specification declared this as a multivalued attribute. Multivalued attributes need to be treated in a special way as a reindeer can have multiple trophies. E.g. a reindeer has the ability to win the same title multiple years in a row.

```
CREATE TABLE GroupOfReindeers(
    ReindeerNr int not null,
    ReindeerName varchar(255) not null,
    GroupNr int not NULL,
    Capacity int,
    Quantity int not null,
    Share float,


    primary key(ReindeerNr),
    foreign key(ReindeerNr) REFERENCES Reindeer(Nr)

)ENGINE=INNODB;
```



Group of reindeers is also a table related to the main reindeer table. Where a reindeer is unable to exist without also belonging to a group. No adjustments were made to modify the table.

```sql
CREATE TABLE Sleigh(
    Nr int unique not null,
    SleighName varchar(255),
    Manifactor varchar(255),
    StepLenght int,
    Capacity int,

    primary key (Nr)

)Engine=INNODB;

CREATE TABLE LoadSleigh(
    SleighNr int unique not null,
    ExtraCapacity int,
    ClimateType varchar(255),

    primary key(SleighNr),
    foreign key(SleighNr) REFERENCES Sleigh(Nr)

)ENGINE=INNODB;

CREATE TABLE ExpressSleigh(
    SleighNr int unique not null,
    RocketQuantity int,
    BreakEfficiency int,

    primary key(SleighNr),
    foreign key (SleighNr) REFERENCES Sleigh(Nr)
)ENGINE=INNODB;
```

The data cluster managing the sleigh for Santa, is kind of their own data tables as a design choice was made that groups of reindeers does not have a particular sleigh assigned to them. What if a new sleigh is made and upgrades group number 2s? And I think that E.g. group 1 can use group 2s express sleigh because of reparations.

```
CREATE TABLE ReindeerNames(
    RNr int unique not null,
    RName varchar(255),

    PRIMARY KEY (RNr),
    foreign key (RNr) REFERENCES Reindeer(Nr)


)ENGINE=INNODB;
```

A horizontal denormalization was made to split the reindeers names and a table was created to only store their reindeer number as well as name. This was made with the functionality in mind as when trying to select reindeer and you only want their number and name, it should be easier and less clustered table, as the reindeer entity can be a little too much.

```
CREATE TABLE ReindeerHasTrophy(
    TReindeerNr int unique not null,
    ReindeerName varchar(255),
    TReindeerTitle varchar(255),
    TrophyOccation DATE,

    FOREIGN KEY(TReindeerNr) REFERENCES Reindeer(Nr),
    FOREIGN KEY(TReindeerNr, TrophyOccation) REFERENCES Trophy(ReindeerNr, Occation),
    PRIMARY KEY(TReindeerNr, TReindeerTitle)


)ENGINE=INNODB;
```

ReindeerHasTrophy is a table made so that a functionality could be implemented so a procedure call can insert a trophy for a reindeer in the future.

```
CREATE TABLE data_log_ReindeerChanges(
    id int unsigned NOT NULL auto_increment,
    logData varchar(255) NOT NULL,
    Username varchar(255) NOT NULL,
    timeOccured TIMESTAMP,
    PRIMARY KEY(id)
)ENGINE=INNODB;
```

As aforementioned, a way to log information of any form of update to tables is fundamental to implement a secure system. Also as stated above, this table is not enough in my opinion and in the future should be implemented with abstraction in mind.
The bigger a database is, the bigger the necessity of a structured logging system is needed.

2.2 Triggers

...

```
DELIMITER $$
CREATE TRIGGER tr_invokeInsert_reindeerDupe
BEFORE INSERT ON WorkReindeer
FOR EACH ROW
    BEGIN
        IF ((SELECT RetiredReindeer.ReindeerNr FROM RetiredReindeer) = NEW.ReindeerNr) THEN
            SIGNAL sqlstate '45000' SET MESSAGE_TEXT = 'Reindeer is pensionized';
        END IF;
    END$$


CREATE TRIGGER tr_invokeUpdate_reindeerDupe
BEFORE UPDATE ON WorkReindeer
FOR EACH ROW
    BEGIN
        IF ((SELECT RetiredReindeer.ReindeerNr FROM RetiredReindeer) = NEW.ReindeerNr) THEN
            SIGNAL sqlstate '45000' SET MESSAGE_TEXT = 'Reindeer is pensionized';
        END IF;
    END$$
DELIMITER ;

DELIMITER $$
```

The assignment specifies that at least a couple of triggers/updates should be implemented. I choose to build triggers to fire and return a *sql message* to prompt the user that is inserting data to transfer a reindeer from retired to worker reindeer is invalid. As the entities are disjoint and a already retired reindeer is impossible to go back to work.

```
DELIMITER $$
CREATE TRIGGER tr_invokeError_sleighName
BEFORE INSERT ON Sleigh
FOR EACH ROW
    BEGIN
        IF (new.SleighName = "Brynolf") OR (new.SleighName = "Rudolf") THEN
            SIGNAL sqlstate '46000' SET MESSAGE_TEXT = 'Sleigh name cannot be Brynolf or Rudolf!';
        END IF;
    END$$
```

Another trigger that should be fired according to the specification is that a sleigh can never have the names ‘Brynolf’ or ‘Rudolf’. This was a very simple trigger to create.

```
CREATE TRIGGER tr_invokeInsert_reindeerHasTrophyDupe
BEFORE INSERT ON ReindeerHasTrophy
FOR EACH ROW
    BEGIN
        IF EXISTS (SELECT TReindeerTitle FROM ReindeerHasTrophy WHERE TReindeerTitle = NEW.TReindeerTitle
        AND TrophyOccation = NEW.TrophyOccation) THEN
        SIGNAL sqlstate '47000'
        SET MESSAGE_TEXT = 'That title has already been distributed that day!';
        END IF;
    END $$
```

For the VG part of the report additional triggers should be implemented. The first is a trigger to fire when an insert happens to a trophy table that has already been distributed for that year.

```
CREATE TRIGGER tr_invokeLogging_reindeerStatusChange
AFTER DELETE ON WorkReindeer
FOR EACH ROW
    BEGIN
        INSERT INTO data_log_ReindeerChanges(logData, Username, timeOccured)
        VALUES(CONCAT(('Deleted: ', ReindeerNr, @logData)), 'Santa', CURRENT_TIMESTAMP());
    END$$

DELIMITER ;
```

The second is for logging data when a reindeers status gets changed. This is the easiest way I found in the limited time that I had to realize the database to log data. The possibilities are endless when implementing this in a real system.

For the views part of the script I implemented some views to be used in procedure calls which is going to be discussed further later in this report.

```
CREATE VIEW ShowReindeersInGroups AS
SELECT Reindeer.Nr AS 'ReindeerNr', Reindeer.ReindeerName, GroupOfReindeers.GroupNr
FROM Reindeer INNER JOIN GroupOfReindeers
ON Reindeer.GroupBellonging = GroupOfReindeers.GroupNr
ORDER BY GroupOfReindeers.GroupNr ASC;
```

ShowReindeersInGroups is a view that is going to be used in a procedure call that is going to cast all the reindeers and in their corresponding groups. This is done by an inner join between Reindeer and GroupOfReindeers. parsing the data in row GroupNr in ascending order, increasing readability.

```
CREATE VIEW ListSalaries AS
SELECT  ReindeerNames.RNr AS 'Nr', ReindeerNames.RName AS 'Name', WorkReindeer.Salary
FROM ReindeerNames INNER JOIN WorkReindeer
ON ReindeerNames.RNr = WorkReindeer.ReindeerNr;
```

ListSalaries is a view of an inner join between Reindeer and WorkReindeer, to show the caller a table of salaries of all the salaries of the working reindeers as well as a ReindeerNames table making its appearance here to show the individual names as well. This increases the readability when called via a procedure call.

```
CREATE VIEW ViewGroups AS
SELECT ReindeerNames.RNr AS 'Nr', ReindeerNames.RName AS 'Name', GroupOfReindeers.GroupNr
FROM ReindeerNames INNER JOIN GroupOfReindeers
ON ReindeerNames.RNr = GroupOfReindeers.ReindeerNr;
```

Lastly, a view to view the groups. What distincts this view from ShowReindeersInGroups is that a procedure call called WhosInMyGroup will use this view to invoke the procedure to view the tables that correspond to the parameters of the procedure.

3 Discussions

The system allows for implementation of privileges to be asserted user profiles. The different profiles should be used to restrict access for the users that should not be able to do different jobs.
E.g. Santa, is at the top of the hierarchy and should be able to either view or manipulate data as he is pleased. According to practice lower responsibility users e.g. children should not have the possibility to manipulate or view any data regarding anything but their own wishlists.

```
-- Santa --
CREATE USER 'Santa'@'localhost' IDENTIFIED BY 'Northpole';

GRANT ALL PRIVILIGES ON a20muhal.* TO 'Santa'@'localhost';
```

Therefore, privileges should be granted to individuals or groups of individuals (e.g. all children should have the same rights, but some helpers might need additional functionality if they are middle managers).

```
-- Reindeer User --
CREATE USER 'Reindeer'@'localhost' IDENTIFIED BY 'Northpole';

GRANT EXECUTE ON a20muhal.WhosInMyGroup TO 'a20muhalReindeer'@'localhost';
```

Reindeer users were created to have the ability to show what group they belong to. This is just to show how privileges should be implemented in the future.

Referenser

Boulos, M.N.K., Warren, J., Gong, J. & Yue, P. (2010) Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping. *International journal of health geographics* 9, 14.

Shin, Y. & Wunsche, B.C. (2013) A smartphone-based golf simulation exercise game for supporting arthritis patients. *2013 28th International Conference of Image and Vision Computing New Zealand (IVCNZ), IEEE,* pp. 459–464.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2012) *Experimentation in Software Engineering*, Berlin, Heidelberg: Springer Berlin Heidelberg.