```
# Marcus Otterström
# Minhui Zhong

USE lab4;

# 1: Foreign keys
# Skriv queries (ALTER TABLE) för koppla ihop tabellerna med foreign_keys och ta
med lämpliga val för vad
# som ska hända vid updates och deletes på PK:
# departments-mangager till employees-id,
# project-supervisor till employees-id,
# projectmembers-e_id till employees-id
# projectmembers-p_id till projects-id

# departments-mangager till employees-id,
ALTER TABLE Departments
      ADD CONSTRAINT fk_departments_manager
    FOREIGN KEY (manager) REFERENCES Employees(id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT;

# project-supervisor till employees-id,
ALTER TABLE Projects
      ADD CONSTRAINT fk_projects_supervisor
      FOREIGN KEY (supervisor) REFERENCES Employees(id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT;

# projectmembers-e_id till employees-id
ALTER TABLE Project_members
      ADD CONSTRAINT fk_project_memebers_e_id
      FOREIGN KEY (e_id) REFERENCES Employees(id)
      ON UPDATE CASCADE
      ON DELETE CASCADE;

# projectmembers-p_id till projects-id
ALTER TABLE Project_members
      ADD CONSTRAINT fk_project_memebers_p_id
      FOREIGN KEY (p_id) REFERENCES Projects(id)
      ON UPDATE CASCADE
      ON DELETE CASCADE;

# Tar bort dessa constraints innan vi kör andra queries då det blir mycket
jobbigare att kompilera annars
ALTER TABLE Departments DROP FOREIGN KEY fk_departments_manager;
ALTER TABLE Projects DROP FOREIGN KEY fk_projects_supervisor;
ALTER TABLE Project_members DROP FOREIGN KEY fk_project_memebers_e_id;
ALTER TABLE Project_members DROP FOREIGN KEY fk_project_memebers_p_id;

# 2: Projects
# Skriv queries så att projects ändras (ALTER TABLE) så att varje project alltid
har en supervisor,
# så att två projektnamn inte kan vara samma, och så att det alltid måste finnas
ett projektnamn.

# varje project alltid har en supervisor
ALTER TABLE Projects MODIFY supervisor int NOT NULL;

# två projektnamn inte kan vara samma, alltid måste finnas ett projektnamn
```

```
ALTER TABLE Projects MODIFY name varchar(50) UNIQUE NOT NULL;


# 3: Departments
# Ändra i DB så att alla anställda alltid är knutna till en avdelning. Ändra så att
default för nya
# anställda och för anställda utan dept ska vara "Training". Det ska inte kunna
finnas anställda
# utan avdelningar. Lägg också in att departments måste ha ett namn och det namnet
måste vara unikt
# samt att varje department måste ha en manager.

# Det ska inte kunna finnas anställda utan avdelningar.
# Ändra så att default för nya anställda och för anställda utan dept ska vara
"Training"
ALTER TABLE Employees MODIFY department int NOT NULL DEFAULT 4;

# Ändra i DB så att alla anställda alltid är knutna till en avdelning.
ALTER TABLE Employees
    ADD CONSTRAINT fk_employees_department
      FOREIGN KEY (department) REFERENCES Departments(id)
    ON DELETE RESTRICT
      ON UPDATE CASCADE;

# departments måste ha ett namn och det namnet måste vara unikt
ALTER TABLE Departments MODIFY department varchar(50) UNIQUE NOT NULL;

# varje department måste ha en manager
ALTER TABLE Departments MODIFY manager int NOT NULL;

# Tar bort dessa constraints innan vi kör andra queries då det blir mycket
jobbigare att kompilera annars
ALTER TABLE Employees DROP FOREIGN KEY fk_employees_department;

# 4: Employees
# Vi (eller vår kund/beställare) vet att det kommer göras många sökningar och
sorteringar på efternamn.
# Lägg därför in INDEX där. Användarnamn (login) måste också spärras så inte två
kan få samma.

ALTER TABLE Employees ADD INDEX (last_name);
ALTER TABLE Employees MODIFY login varchar(50) UNIQUE;


# 5: Optimera datatyper
# Skriv queries som ändrar (ALTER TABLE)  ett par av datatyperna till något som är
effektivare:
# projects - name behöver inte vara VARCHAR(50). Ändra till något som är mer lagom.
# departments - id kommer aldrig vara större heltal än 50. Skriv query för att byta
till mer lagom datatyp än INT.
# employees - title - är just nu endast något av "dr", "mr", "ms", "mrs", "rev"
eller "honorable". Byt datatyp till ENUM (och ta eventuellt med något ytterligare
alternativ som ni tycker bör finnas med).
# employees - leta reda på ett par kolumner där vi alltid måste ha något värde och
ändra schemat så att vi inte längre kan lagra null-värden för dessa.
# employees - det får inte kunna lagras dubletter av epostadress. Skriv query som
ändrar tabellstrukturen och query som visar att det inte längre går att lagra
dubletter.
```

```
# projects - name behöver inte vara VARCHAR(50). Ändra till något som är mer lagom.
ALTER TABLE Projects MODIFY name VARCHAR(25) UNIQUE;

# departments - id kommer aldrig vara större heltal än 50. Skriv query för att byta
till mer lagom datatyp än INT.
ALTER TABLE Departments MODIFY id TINYINT AUTO_INCREMENT;
ALTER TABLE Employees MODIFY department TINYINT;
DESCRIBE Departments;

# employees - title - är just nu endast något av "dr", "mr", "ms", "mrs", "rev"
eller "honorable".
# Byt datatyp till ENUM (och ta eventuellt med något ytterligare alternativ som ni
tycker bör finnas med).
ALTER TABLE Employees MODIFY title ENUM
('dr','mr','ms','mrs','rev','honorable','mx');

# employees - leta reda på ett par kolumner där vi alltid måste ha något värde och
ändra
# schemat så att vi inte längre kan lagra null-värden för dessa.
ALTER TABLE Employees MODIFY birth_date date NOT NULL;

# employees - det får inte kunna lagras dubletter av epostadress.
ALTER TABLE Employees MODIFY email VARCHAR(50) UNIQUE;

# Skriv query som ändrar tabellstrukturen och query som visar att det inte längre
går att lagra dubletter.
UPDATE Employees SET email = 'my-unique-email@example.com' WHERE id = 1;
UPDATE Employees SET email = 'my-unique-email@example.com' WHERE id = 2;


# 6: Skapa vy: salary_data_dept
# Skapa en vy som ger avdelningsnamn, minsta lön, högsta lön, medellön och antal
anställda för varje avdelning.
DROP VIEW IF EXISTS salary_data_dept;
CREATE VIEW salary_data_dept AS
    SELECT
           departments.department,
           MIN(salary) AS lowest_salary,
           MAX(salary) AS highest_salary,
           AVG(salary) AS average_salary,
           COUNT(*) AS amount_of_employees
      FROM Employees
    JOIN Departments ON Employees.department = Departments.id
      GROUP BY department;

SELECT * FROM salary_data_dept;


# 7: Skapa funktion och vy: retirement_status
# Skapa en funktion som utifrån födelsedatum ger svaren "more than 5 years" för de
som har mer än 5 (hela) år kvar
# tills de fyller 65, för de som har mellan 5 och 1 år till 65 svarar funktionen
hur många år det är kvar
# ex "3 years left", för de som fyller inom år men ännu inte fyllt svarar
funktionen "Up for retirement" och för de
# som är äldre än 65 så svarar funktionen "Retired".

# Skapa sedan en vy som listar anställda med titel, förnamn, efternamn, ålder,
födelsedag och avdelning de jobbar på
```

```
# samt deras retirement_status. Sortera på avdelning och sedan födelsedatum. Visa
bara för de som är 55 år eller äldre.
DROP FUNCTION IF EXISTS retirement_status_function;

DELIMITER //

CREATE FUNCTION retirement_status_function(birth_date DATE)
RETURNS VARCHAR(20)
NO SQL
BEGIN
    DECLARE years_until_65 INT;
    SELECT 65 - TIMESTAMPDIFF(YEAR, birth_date, DATE(NOW())) INTO years_until_65;

      IF years_until_65 > 5 THEN
          RETURN 'more than 5 years';
      ELSEIF years_until_65 > 1 THEN
          RETURN CONCAT(years_until_65 - 1, ' whole year(s) left');
      ELSEIF years_until_65 > 0 THEN
          RETURN 'up for retirement';
      ELSE
        RETURN 'retired';
    END IF;
END //
DELIMITER ;

DROP VIEW IF EXISTS retirement_status;
CREATE VIEW retirement_status AS
    SELECT
            title,
            first_name,
            last_name,
            TIMESTAMPDIFF(YEAR, birth_date, DATE(NOW())) as age,
            birth_date,
            Departments.department,
            retirement_status_function(birth_date)
      FROM Employees
      JOIN Departments ON Departments.id = Employees.department
      WHERE TIMESTAMPDIFF(YEAR, birth_date, DATE(NOW())) > 55
      ORDER BY Departments.department, birth_date;

SELECT * FROM retirement_status;

# 8: Skapa tabell: total_salary
# Skapa en tabell för summan av löner, lägg in nuvarande värde (gör SELECT
SUM(salary)...)
# och lägg till triggers på anställdas löner så att summan av löner alltid stämmer
i nya tabellen.
# Tabellen ska också ha ett fält för last_update som visar när den senast
uppdaterades.
# Testa och visa med några UPDATE och INSERT  och DELETE av anställda att det
fungerar.
# (Detta skulle kunna spara resurser för DB om vi har många anställda och ofta
ställer frågor
# kring summan av löner. Med detta upplägg så behöver vi inte göra den potentiellt
tunga summeringen
# av alla löner utan det räcker med att göra en enkel uppslagning i den alltid
uppdaterade total_salary.
# Se även materialiserad vy (sid 312 i Databasteknik.))
```

```
# (Använd inte SUM i triggers - det är just den lite tyngre beräkningen att summera
alla raderna som vi
# vill undvika med denna övning.)

DROP TABLE IF EXISTS Total_salary;
CREATE TABLE Total_salary (
    value INT,
    last_edit TIMESTAMP NOT NULL DEFAULT NOW() ON UPDATE NOW()
);

INSERT INTO Total_salary (value)
VALUES ((SELECT SUM(salary) FROM Employees));

DROP PROCEDURE IF EXISTS update_total_salary;
DROP TRIGGER IF EXISTS update_total_salary_update;
DROP TRIGGER IF EXISTS update_total_salary_insert;
DROP TRIGGER IF EXISTS update_total_salary_delete;

DELIMITER //

CREATE PROCEDURE update_total_salary(old_salary INT, new_salary INT)
BEGIN
      IF old_salary != new_salary THEN
          UPDATE Total_salary SET value = value - old_salary + new_salary;
    END IF;
END //

CREATE TRIGGER update_total_salary_update
BEFORE UPDATE ON Employees FOR EACH ROW CALL update_total_salary(OLD.salary,
NEW.salary);

CREATE TRIGGER update_total_salary_insert
BEFORE INSERT ON Employees FOR EACH ROW CALL update_total_salary(0, NEW.salary);

CREATE TRIGGER update_total_salary_delete
BEFORE DELETE ON Employees FOR EACH ROW CALL update_total_salary(OLD.salary, 0);

DELIMITER ;

# Original value
SELECT * FROM Total_salary;

INSERT INTO Employees (id, birth_date, salary) VALUES (1234, '1990-01-01',
1000000);
# Total salary + 1 000 000
SELECT * FROM Total_salary;

UPDATE Employees SET salary = salary + 1000 WHERE id = 1234;
# Total salary + 1 000
SELECT * FROM Total_salary;

UPDATE Employees SET salary = salary - 2000 WHERE id = 1234;
# Total salary - 2 000
SELECT * FROM Total_salary;

DELETE FROM Employees WHERE id = 1234;
# Total salary - 999 000
SELECT * FROM Total_salary;
```

```
# 9: Flytta hr_notes
# Skriv queries för att skapa en tabell och flytta allt innehåll i hr_notes till
den nya tabellen
# samt koppla den nya tabellen till employees-id med foreign keys. Ta sedan bort
kolumnen för hr_notes
# från employees. (Spara era queries för redovisningen. Det räcker inte att visa
nya tabellerna utan det
# krävs att ni redovisar vilka queries som behöver köras för att skapa dem.)
# Skriv query som visar efternamn, förnamn, hr_notes för alla anställda utifrån den
nya tabellen.

DROP TABLE IF EXISTS Hr_notes;
CREATE TABLE Hr_notes (
    id INT PRIMARY KEY AUTO_INCREMENT,
      note TEXT,
    employee_id INT REFERENCES Employees(id)
);

# Migration can only run once, comment after running (or recreate database)
INSERT INTO Hr_notes (employee_id, note) SELECT id, hr_notes FROM Employees;
ALTER TABLE Employees DROP COLUMN hr_notes;

SELECT last_name, first_name, Hr_notes.note AS hr_notes
FROM Employees
JOIN Hr_notes ON Hr_notes.employee_id = Employees.id;


# 10: Begränsa löneändringar
# Skriv kod för trigger som gör att löner begränsas så att det inte går att sänka
lönen
# för en anställd och inte går att höja med mer än 10% i en update.
DROP TRIGGER IF EXISTS restrict_salary_change;
DELIMITER //
CREATE TRIGGER restrict_salary_change
BEFORE UPDATE ON Employees
    FOR EACH ROW
    IF OLD.salary != NEW.salary AND
       (NEW.salary > (OLD.salary + OLD.salary * 0.1) OR
        NEW.salary < (OLD.salary - OLD.salary * 0.1)) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Får inte höja eller sänka
lönen med mer än 10%.';
    END IF //

DELIMITER ;

INSERT INTO Employees (id, birth_date, salary) VALUES (9999, '1990-01-01', 10000);
SELECT salary FROM employees WHERE id = 9999;
UPDATE employees SET salary = 20000 WHERE id = 9999; # Not allowed, too high. 10100
is allowed.
SELECT salary FROM employees WHERE id = 9999;


# 11: Egen vy
# Skapa en egen vy som gör något som är meningsfullt och användbart med databasen.
# Hitta på något eget som kan passa och vara intressant. Skriv tydliga kommentarer
och queries som
# visar hur den används och fungerar. (Vanlig vy räcker. Behöver ej vara en
materialiserad vy med
```

```
# triggers etc.  Men för den intresserade som vill ha en liten utmaning går det
även att bygga en
# materialiserad vy och redovisa den.)

# Räknar ut vilka 20 som har högst lön sett till sin ålder (lön per år sen födsel)
DROP VIEW IF EXISTS richest_youngsters;
CREATE VIEW richest_youngsters AS
    SELECT
        first_name,
            last_name,
            salary,
            TIMESTAMPDIFF(YEAR, birth_date, DATE(NOW())) as age
      FROM Employees
      ORDER BY salary / age DESC
      LIMIT 20;

SELECT * FROM richest_youngsters;

# 12: Egen procedure eller function
# Skapa en egen procedure eller function som gör något som är meningsfullt och
användbart med databasen.
# Hitta på något eget som kan passa och vara intressant. Skriv tydliga kommentarer
och queries som visar hur den används och fungerar.
DROP PROCEDURE IF EXISTS birth_date_countdown;
DELIMITER //
CREATE PROCEDURE birth_date_countdown()
BEGIN
      # Räknar fram hur många dagar det är kvar till de anställda ska fira sin
födelsedag
    SELECT
            first_name,
            last_name,
            birth_date,
            DAYOFYEAR(birth_date) - DAYOFYEAR(NOW()) AS days_until_birthday
    FROM Employees
    WHERE DAYOFYEAR(birth_date) - DAYOFYEAR(NOW()) > 0
    ORDER BY days_until_birthday;
END //
DELIMITER ;

CALL birth_date_countdown();
```