ISE DI – CASUS_

COURSE DATABASE

Nabben

22 augustus 2023

HAN_UNIVERSITY OF APPLIED SCIENCES

INHOUDSOPGAVE

CASE DESCRIPTION

This case study concerns the implementation of a small company database system in which data about employees (including some historical data), their salaries, the departments where they work and company courses they take have to be managed.

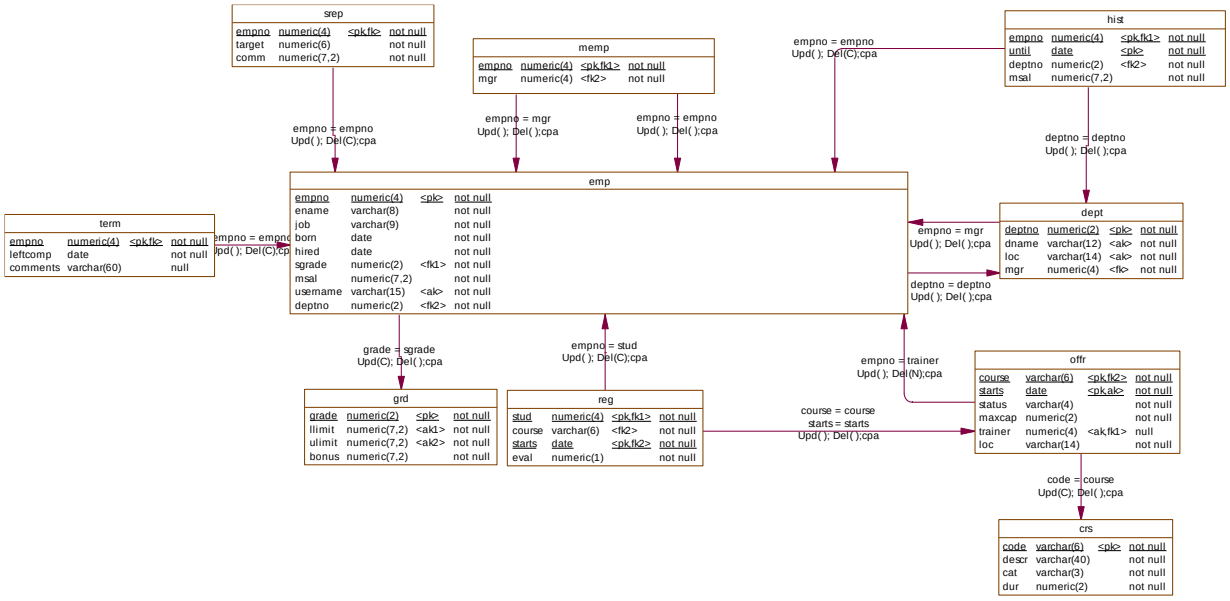
In the spirit of the database modeling and design course DMDD the ten tables involved are documented in predicate style:

Table **EMP**: The employee with employee number **EMPNO** has name **ENAME**, job **JOB,** was born on **BORN**, is hired on **HIRED,** has a monthly salary of **MSAL** dollars within **SGRADE** salary grade, is assigned to account **USERNAME**, and works for the department number **DEPTNO**.
Table **SREP:** The sales representative with employee number **EMPNO** has an annual sales target of **TARGET** dollars and a yearly commission of **COMM** dollars.
Table **MEMP**: The employee with employee number **EMPNO** is managed by the employee with employee number **MGR**.
Table **TERM**: The employee with employee number **EMPNO** has resigned or was fired on date **LEFTCOMP** due to reasons **COMMENTS** (initially empty).
Table **DEPT**: The department with department number **DEPTNO**, has name **DNAME**, is located at **LOC**, and is managed by the employee with employee number **MGR**.
Table **GRD**: The salary grade with id **GRADE** has a lower monthly salary limit of **LLIMIT** dollars, an upper monthly limit of **ULIMIT** dollars, and a maximum yearly bonus of **BONUS** dollars.
Table **CRS**: The course with code **CODE** has description **DESCR**, falls in category **CAT**, and has a duration of **DUR** days.
Table **OFFR**: The course offering for the course with code **COURSE** that starts on date **STARTS**, has status **STATUS**, has a maximum capacity of **MAXCAP** attendees, is offered at location **LOC**, and (unless **TRAINER** equals NULL) the offering has the employee with employee number **TRAINER** assigned as the trainer.
Table **REG**: The employee whose employee number is **STUD** has registered for a course with code **COURSE** that starts on **STARTS**, and (unless **EVAL** equals -1) has rated the course with an evaluation score of **EVAL**.
Table **HIST**: At date **UNITIL**, for the employee whose employee number is **EMPNO**, either department or the monthly salary (or both) have changed. Prior to the date **UNTIL**, the department for that employee was **DEPTNO** and the monthly salary was **MSAL** (table initially empty).

Note: whether you like it or not, this is the database you have to deal with (you have to deal with some design flaws in your code).

Relational schema

This schema is created with SAP PowerDesigner, shows the main structural- and integrity aspects of the database.



SCRIPTS

At OnderwijsOnline (https://onderwijsonline.han.nl) in the course folder ISE DI chapter 2.2, you'll find three sql scripts, a basic create table script (COURSE_cretab.sql), an insert script (COURSE_database_state.sql) and a constraints creation script (COURSE_constraints.sql). Run these scripts in the order as listed above and study these scripts.

ASSIGNMENT TASKS

Below you'll find tasks to implement the Course Offering Database, but before you start, read the next section meticulously.

The case study consists of two sets of assignments. One set of questions consists of 14 **feedback assignments** (in Dutch: formatieve feedback/beoordeling) organized per week. You hand in your answers to these feedback questions at the end of the week and receive feedback in the next week. The procedure of how to hand in/upload your work and where to look for feedback will be explained by your lecturer at startup of the course.

The second set of case study assignments consists of **assignments which will actually be graded** (in Dutch: summatieve feedback/beoordeling). Ideally you hand in answers to these assignments after you have received feedback on feedback assignments concerning the same theme.

At the begin of the first week a detail planning will be given when each task should be finished and uploaded for feedback. For example (the first week):

Lesson	Topics	Case study (bold=FEEDBACK) (underline=REPORT)
1.1	Herhaling SQL Advanced SQL till EXISTS/NOT EXISTS (slide 24)	Task **1** + **2.1**
1.2	Advanced SQL tm SET operatoren (slide 36)	Task 2.2
1.3	Advanced SQL tm Logical query processing (slide 55).	**Upload Task 1 + 2.1**

This way we try to spread the case study work load and force you to practice from the early start of the course.
In the end you deliver a report that should meet all these requirements:

- Deliver your report in Word format, **not in PDF format nor just as a .sql script file.**
  It is easier for us to add feedback to Word files.
- Structure your document per task corresponding this document. Add all relevant T-SQL code in this document. Add also the definition of the task.
- Add all tests of one constraint or information need in this report (see chapter 6). All other tests should be delivered as SQL script(s).
- Don't forget to index the document and add page numbers. It is advicable to use this case document as a template for your own report.
- The total file size should not exceed the limitations of iSAS.

1 SETUP DATABASE

Run the case study database scripts provided on OnderwijsOnline

1.1 Task 1 - Implement Foreign Keys and Cascading Rules

Check the constraint script, add or change the correct foreign key and cascading rule declarations as depicted in the PowerDesigner PDM.

The PDM schema uses some abbrevations:

- Upd() means no cascade
- Upd(C) means cascade
- Del(N) means set null
- cpa means change parent allowed. You can ignore this for this task.

Make sure you explicitly define the cascading rules. Don’t use default implementations.

2 INFORMATION NEEDS

Some of these information needs are also usefull for implementing constraints during the rest of this case. Give an SQL implementation for each of the information needs as mentioned below. Add some testdata if needed.

Create one View per information need that produces the correct result.

2.1 Task 2.1 – List the valid trainers

Give a list of all offered courses that are given by a valid trainer. A valid trainer is:

1) someone whose job is trainer and who is at least one year an employee in this organization,
2) or someone who has attended this course (as a participant).

Return the empno and ename of the trainer and the course code and course description of the applicable courses.
Use one or more set operators or exists operators.

The header of this view should be:
CREATE VIEW VW_ListTheValidTrainers

The output should by in this format:

code	descr	empno	ename
AM4DP	Applied Math for DB-Pros	1016	Mark
AM4DP	Applied Math for DB-Pros	1017	Marcel
AM4DP	Applied Math for DB-Pros	1018	Marco
AM4DPM	AM4DP for Managers	1016	Mark
AM4DPM	AM4DP for Managers	1017	Marcel
AM4DPM	AM4DP for Managers	1018	Marco
…	…	…	…

Here course AM4DP *can be taught by employee 1016, 1017 and 1018.*

2.2 Task 2.2 – List exact same set of course registrations

Give a list of all employees that are registered for exact the same set of courses started on the same date. Make sure to filter out the duplicates (pair A and B and B and A are duplicate pairs).
Use one or more set operators or exists operators.

The header of this view should be:
CREATE VIEW VW_ListExactSameSetOfCourseRegistrations

The output should by in this format:

Empno1	Empno2
…	…
1030	1032
1030	1033
1032	1033
…	…

Where as employee 1030, 1032 and 1033 are registered for exact the same set of courses started on the same date.

2.3 Task 2.3 – List total costs per salary grade per year

The cost per salary grade is calculated by the yearly salary of all employees of this grade. The yearly salary is based on the monthly salary plus the yearly bonus as defined per grade.
Give a list of all the salary grades, the total salary costs per salary grade (the bonus included) per year and also give a running total.
Use one or more windowing functions or CTE operators.

The header of this view should be:
CREATE VIEW VW_ListTotalCostsPerSalaryGradePerYear

The output should by in this format:

grade	costs	runningtotal
1	0.00	0.00
2	55600.00	55600.00
3	110700.00	166300.00
…	…	…

2.4 Task 2.4 – List salary changes per employee

The table hist stores the history of modifications in salary. If the salary of an employee changes the previous salary is stored in the history table together with the actual date (column until). The salary in the employee table is always to most recent salary.

Give a list of salary modifications per employee (so sorted on employee) that gives an overview of the salary someone had in a certain periode.

The output should by in this format:

empno	from	until	msal
…	…	…	…
1000	1992-01-01	1995-06-30	10000.00
1000	1995-06-30	2001-08-30	11100.00
1000	2001-08-30	2007-10-30	12200.00
1000	2007-10-30	2013-12-30	13300.00
1000	2013-12-30	2020-02-29	14400.00
1000	2020-02-29	NULL	15500.00
1004	2000-07-01	2004-06-30	1000.00
1004	2004-06-30	2011-12-30	1470.00
1004	2011-12-30	2019-06-30	1930.00
1004	2019-06-30	NULL	2400.00
1005	2000-07-01	NULL	7900.00
…	…	…	…

In this example empno 1005 has had no salary modification. The periode is from hire date until now (displayed as null). Empno 1000 has had 5 modifications. He started at 1992-01-01 (hired date) with a salary of 10000 and had his 1st salary increase of 1100 at 1995-06-30 and so on. Form 2020-02-29 till now he earns 15500 per month.

Use one or more windowing functions or CTE operators.

The header of this view should be:

CREATE VIEW VW_ListSalaryChangesPerEmployee

Note: The hisyt table has some data but you can create your own test data for testing your solution.

3 IMPLEMENT DECLARATIVE CONSTRAINTS

This chapter involves rather simple constraints which can be implemented using declarative constructs.

3.1 Task 3.1 – President earns more than $10.000

Implement and test the constraint '*the president of the company earns more than $10.000 monthly'*.

3.2 Task 3.2 – Company hires adult personnel only

Implement and test the constraint '*the company hires adult personnel only'.*

4 IMPLEMENT CONSTRAINTS & FUNCTIONALITY USING STORED PROCEDURES

4.1 Task 4.1 – Trainers cannot teach different courses simultaneously

Implement and test a stored procedure enforcing the business rule '*trainers cannot teach different courses simultaneously'*. The rule should be validated as soon as a new course offering is inserted.

Note: There is already an AK on the offer table that prevents trainers from giving another course on the same day. The rule 'Trainers cannot teach different courses simultaneously' is even stricter than that. A course has a length in number of days (see table crs column dur). A trainer can only give one course per day. So, if trainer X starts a course Y on 01-10-2021 that lasts for two days, he can only (at the earliest) give the next course on 03-10-2021. Courses cannot overlap for a trainer (otherwise he would have to split himself that day and that is not possible....).

4.2 Task 4.2 – Promote employee as a manager

In case an employee is promoted to MANAGER several database actions need to be executed;

- The employee role needs to be updated in 'MANAGER'
- The employees of the same department as their new manager will be managed by this employee. Therefore, the data in table MEMP need to be updated likewise.
- The salary grade of this new manager will be set to 9 and his salary amount will be set to the minimum of this grade plus an extra amount of 20.00 per employee that is managed by him/her. The salary should always be limited to the max of this grade.

For example: if employee 1015 (Antoinet) will become the new manager of department 15, she will manage 21 employees and have a salary of 8420 (8000 + 20.00 * 21).

Some extra checks need to be done on forehand;

- The new manager is already working for this department.
- There is no other employee in this department with the role 'MANAGER'

The header of this stored procedure should be:

CREATE PROCEDURE stp_SetManager (@empno numeric (4.0), @deptno numeric(2,0))

Note: Transactions and concurrency control is not an issue yet. In Task 7.2 you will add transactions and optimize this for avoiding concurrency problems.

5 IMPLEMENT CONSTRAINTS USING TRIGGERS

The following constraints need to be implemented using one or more triggers.

When you develop the constraint:

- Include in your report the original definition of the constraint,
- Describe which tables and columns are involved in this constraint
- Describe for which actions (I/U/D) this constraint should be validated

It is not allowed to implement a constraint with a combination of check constraints and user defined functions.

5.1 Task 5.1 – Only valid trainers allowed

Implement using a trigger the constraint *'You are allowed to teach a course only if: 1) your job type is trainer and you have been employed for at least one year or 2) you have attended the course yourself (as participant).'*

5.2 Task 5.2 – Administrator for department

Implement the following constraints using triggers *'A department that employs the president or a manager should also employ at least one administrator'.*

5.3 Task 5.3 – Salary grade limitations

Implement the following constraints using triggers '*The llimit of a salary grade must be higher than the llimit of the next lower salary grade and the ulimit of the salary grade must be higher than the ulimit of the next lower'.*

Note: the numbering of the grades can contain holes.

For example:

Suppose we have the following data (only the relevant columns are listed)

Table Grd

grade	***llimit***	***ulimit***	***…***
1	*500.00*	*1500.00*	*…*
2	*1000.00*	*2500.00*	*…*
6	*2000.00*	*3500.00*	*…*
7	*3000.00*	*4500.00*	*…*

In this case the llimit of grade 6 is higher than the llimit of the next lower salary grade (which is grade 2). If we want to insert grade 3 then the llimit should be between 1000 (llimit of grade 2) and 2000 (llimit of grade 6). Also the ullimit should be between 2500 and 3500.

5.4 Task 5.4 – Trainers can start only one course a time

Implement the following constraints using triggers 'T*he start date and known trainer uniquely identify course offerings.'*

Note: the constraint 'ofr_unq' is too strict, this does not allow multiple unknown trainers on the same start date, this unique constraint should therefore be dropped. Create a trigger to implement the constraint, the use of a filtered index is not allowed;

For example:
Suppose we have the following data (only the relevant columns are listed)

Table Offr

course	starts	…	trainer
AM4DP	2004-03-02	…	1016
AM4DP	2004-03-03	…	1017
AM4DP	2006-08-03	…	1016
APEX	2004-03-03	…	NULL
DBCENT	2004-03-03	…	NULL

In this situation the unique constraint on the columns starts and trainer would be violated (marked as red). That is too strict for this case. That's why you need to drop this unique constraint.
But, as soon as the trainer is known (so not null) the combination starts and trainer should be unique because a trainer can only teach one course at one day.Suppose we want to update the trainer for course DBCENT at 2004-03-03. In that case on it can only be updated to 1016 and not to 1017 because 1017 already teaches AM4DP at 2004-03-03.

5.5 Task 5.5 – Home based course offerings

Implement the following constraints using triggers *'At least half of the course offerings (measured by duration) taught by a trainer must be 'home based'.*

Note: 'Home based' means the course is offered at the same location where the employee is employed.

For example:
Suppose we have the following data (only the relevant columns are listed)

Table Emp

empno	**….**	**deptno**
1017	….	15

Table Dept

deptno	**…**	**loc**
15	…	SAN FRANCISCO

Table Crs

code	**…**	**dur**
RGARCH	…	4
RGDEV	…	3

Table Offr

course	**starts**	**…**	**trainer**	**loc**
RGARCH	2004-10-03	…	1017	SAN FRANCISCO
RGDEV	1997-08-28	…	1017	SAN FRANCISCO
RGDEV	2004-09-02	…	1017	HOUSTON

In this situation employee 1017 works at deptno 15 which is located in SAN FRANCISCO. Employee 1017 teaches 3 courses. The total duration of the courses in his home town (so called home based) SAN FANSISCO is 7 days (4 days RGARCH at 2004-10-03 + 3 days RGDEV at 2004-09-02). The total duration of the course which are not home based is 3 days (3 days RGDEV at 2004-09-02 in HOUSTON). So this employee teaches at least half of his courses home based.

6 IMPLEMENT UNIT TESTS

The constraints and functionality that you developed need to be tested. The tasks below describe what you nee to unit test as a minimum. Tests should be developed using the tSQLt framework.

6.1 Task 6.1 – Implement Unit tests for an information need

Implement unit tests for one of the information needs as mentioned in chapter 2. Choose one yourself.

6.2 Task 6.2 – Implement Unit tests for a declarative constraint

Implement unit tests for one of the implemented declarative constraints as mentioned in chapter 3. Choose one yourself.

6.3 Task 6.3 – Implement Unit tests for a stored procedure

Implement unit tests for one of the developed stored procedures as mentioned in chapter 4. Choose one yourself.

6.4 Task 6.4 – Implement Unit tests for constraint ‘Only valid trainers allowed’

Implement unit tests for the constraint as mentioned in 5.1.

6.5 Task 6.5 – Implement Unit tests for all other constraints

Implement unit tests for at least

- one implementation of a information need (view)
- one implementation of a declarative constraint (check constraint)
- one implementation of a trigger
- one implementation of a stored procedure

Do not use the same tests as in task 6.1 to 6.4!

Add all tests in one single sql script file and add a reference to the tests in your report.

7 ANALYZE CONCURRENCY CONTROL

In this part of the assignment you are going to analsye whether or the code is suited for mult-user aspects.

7.1 Task 7.1 – Analyze and solve concurrency problem stp_RegisterForCourse

Analyze the stored procedure printed hereafter. It implements the constraint *a trainer cannot register for a course offering taught by him- or herself*.

This implementation potentially suffers from concurrency problems when run in default isolation level mode READ COMMITTED. Illustrate where and why problems may arise by describing what kind of locks are acquired (e.g. s-locks and x-locks), when they are acquired, why they are acquired and for how long they lock a resource.

The code of the stored procedure is listed below:

```
create or alter procedure stp_RegisterForCourse
        @studentCode      numeric(4),
        @courseCode               varchar(6),
        @startDatum               date,
        @eval numeric(1) = -1
as
BEGIN
        DECLARE @savepoint varchar(128) = CAST(OBJECT_NAME(@@PROCID) as varchar(125)) +
        CAST(@@NESTLEVEL AS varchar(3))
        DECLARE @startTrancount int = @@TRANCOUNT;
        SET TRANSACTION ISOLATION LEVEL READ COMMITTED
        BEGIN TRY
                BEGIN TRANSACTION
                SAVE TRANSACTION @savepoint;
                -- Student of the course cannot be the same employee as the trainer of the course
                IF (select trainer from offr where course = @courseCode and starts = @startDatum) = @studentCode
                    RAISERROR('Person registering for this course is already provisioned to be the trainer
                        of the course', 16, 1)
                -- In case of no conflict insert new registration
                insert into reg values (@studentCode, @courseCode, @startDatum, @eval)
                COMMIT TRANSACTION
        END TRY
        BEGIN CATCH
                IF XACT_STATE() = -1 AND @startTrancount = 0
                BEGIN
                        ROLLBACK TRANSACTION
                END
                ELSE
                BEGIN
                        IF XACT_STATE() = 1
                        BEGIN
                                ROLLBACK TRANSACTION @savepoint;
                                COMMIT TRANSACTION;
                        END;
                END;
                THROW
        END CATCH;
END
```

Note: Suppose a second transaction changes the trainer of this course in an interleaved scenario. Explain your analysis with two transactions that illustrates how in this scenario the constraint could be violated. Add a success scenario with an isolation level that solves the problem (at least for this stored procedure).

7.2 Task 7.2 – Analyze and solve concurrency problem stp_SetManager

Analyze the stored procedure of task 4.2 and add transaction management with default isolation level to this stored procedure using the correct template.

This implementation potentially suffers from concurrency problems when run in default isolation level mode READ COMMITTED. Illustrate where and why problems may arise by describing what kind of locks are acquired (e.g. s-locks and x-locks), when they are acquired, why they are acquired and for how long they lock a resource.

Explain your analysis by giving interleaved scenarios with two transactions that illustrate why it can or can’t go wrong. Add a success scenario with an isolation level that solves the problem.

8 IMPLEMENT INDEXES

In this part of the assignment you are going to implement indexen in order to get a better queryplan and thus a better performance for your query.

8.1 Task 8.1 – Optimize information need 1

Find a query of Task 2 which may be optimize by using indexes.

Analyze the current queryplan and think of an index on one or more columns, either clustered or non-clustered, that may optimize the query performance.
Describe your solution and motivate your choice of columns and clustering well. Give the code for the indexes and the execution plans before and after adding the index.

Note: improvement might be hard to actually measure given this small dataset

8.2 Task 8.2 – Optimize information need 2

Find another query (different from task 8.1), which may be optimize by using indexes.

Analyze the currenct queryplan and think of an index on one or more columns, either clustered or non-clustered, that may optimize the query performance.
Describe your solution and motivate your choice of columns and clustering well. Give the code for the indexes and the execution plans before and after adding the index.

Add extra test data if the improvement is hard to measure given the small dataset!

9 IMPLEMENT HISTORY USING CODE GENERATION

The Course database implements a bit of history awareness like in for instance the HIST table (changes in employee's department and/or the salary amount are recorded by stacking the historical state in the HIST table). We want the history of all data changes to be automatically recorded in history tables belonging to the original tables using database triggers. Note: assume primary keys are immutable!

For every table in the Course database;

- a history version table needs to be developed with name HIST_<table name> so HIST_EMP, HIST_DEPT etc.
  These history tables versions need to have the same structure as the original tables, but with a different primary key consisting of the original primary key column(s) combined with a rowversion column (type rowversion). The primary key is the only constraint of these history tables.
- a trigger needs to be developed with name TRG_<table_name>_FILLHISTORY so TRG_EMP_FILLHISTORY, TRG_DEPT_FILLHISTORY etc. These trig-gers will add the inserted, updated and delete records to the history tables.

9.1 Task 9.1 – Generate history tables

The first part of this assignment is to create the history tales that will store the history data.
The following steps need te bo taken;

- Create a working instance for one history table based on the Course database.
- Define the template for the code that need to be generated
- Write a stored procedure that generates the 'create history table' script for a specific table (given a table name as the input parameter). Determine where you can find (using the Information Schema Views) the metadata and retrieve the parameter values you have to use in this template to produce the specific history table.
- Test the correct generation of one history table (similar as your working instance)

The header of this stored procedure should be:
CREATE PROCEDURE stp_CreateHistoryTable (@schema_name sysname, @table_name sysname)

Be sure you meet the naming convention requirement as mentioned above.

9.2 Task 9.2 – Generate history triggers

The second part of this assignment is to create the history triggers that store the data modifications in the history tables.

- Create a working instance for a trigger that populates a history table. Define the template for the code that need to be generated
- Write a stored procedure that generates the 'create trigger' script for a specific table (given a table name as the input parameter). Determine where you can find (using the Information Schema Views) the metadata and retrieve the parameter values you have to use in this template to produce the specific history table.
- Test the correct generation of one generated trigger (similar as your working instance)
- Test the automatically record of data changes in history tables.

The header of this stored procedure should be:
CREATE PROCEDURE stp_CreateHistoryTrigger (@schema_name sysname, @table_name sysname)

9.3 Task 9.3 – Generate history tables and triggers for all tables

Write a stored procedure that generates all history tables and all triggers for all tables in the Course database.

Test the automatically record of data changes in history tables.

The header of this stored procedure should be:
CREATE PROCEDURE stp_CreateHistoryTablesAndTriggersForAllTables

10 IMPLEMENT SECURITY

In this part you are going to implement some security aspects to implement these business requirements secribed in the tasks below.

Implement a suitable (minimal and maintainable) security policy in de database, no more no less. The way you do this may depend on the way you implemented the constraints listed in task.

Test your security regime. Create test users representing the user types employee and reporting service testing via “impersonation” the implementation of the security policy.

10.1 Task 10.1 – Implement Employee account

Employees can register themselves for courses on offer using an app provided by the Human Resources department. For this purpose you should create an **employee** account. The only data the employee have full access to are the data in the REG table. Of course read access to the EMP and OFFR tables are also needed (foreign key checks require the user to have access to the referenced data).

10.2 Task 10.2 – Implement Reporting account

For reporting purpose a specific reporting service account needs to be created allowing reporting tools full read access to all data. Be aware that this account also needs to see the output of the information needs of task 2.

OPEN UP
NEW HAN_UNIVERSITY OF APPLIED SCIENCES
HORIZONS.