# CASE DI

#### 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 **COURSE**: 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 -1) 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*

On the last page of this document you’ll find the relational schema of the database. This schema created with SAP PowerDesigner, shows the main structural- and integrity aspects of the database.

#### Scripts

At OnderwijsOnline (HAN University’s Electronic Learning Environment at url https://onderwijsonline.han.nl) in the course folder ISE DI chapter 2.3, 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.

#### Constraints to be implemented by you in the Course Offering Database in addition to the constraints the scripts already specify (note: even after implementation of these 11 constraints the set still is far from complete):

1. The president of the company earns more than $10.000 monthly.
2. A department that employs the president or a manager should also employ at least one administrator.
3. The company hires adult personnel only.
4. A salary grade overlaps with at most one lower salary grade. The llimit of a salary grade must be higher than the llimit of the next lower salary grade. The ulimit of the salary grade must be higher than the ulimit of the next lower salary grade.
5. The start date and known trainer uniquely identify course offerings. Note: the use of a filtered index is not allowed.
6. Trainers cannot teach different courses simultaneously.
7. An active employee cannot be managed by a terminated employee.
8. A trainer cannot register for a course offering taught by him- or herself.
9. 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.
10. Offerings with 6 or more registrations must have status confirmed.
11. You are allowed to teach a course only if:

your job type is trainer and  
- you have been employed for at least one year

* or you have attended the course yourself (as participant)

#### Communicatie_opdrachtAssignment tasks

#### Below you’ll find tasks to implement the Course Offering Database. You can start with tasks A in the first week of this course. For task B you’d better wait for the explanation of stored procedures and triggers. For tasks C, D, E and F the explanation of concurrency, indexing, code generation and security is necessary. In week 5 you have to hand in the code implementing constraints 1 to 5. Your lecturer will provide extensive feedback on the code written by you (note it concerns extra feedback in addition to the in-class code review feedback!). Create at minimum two stored procedure and two trigger code units for this extra feedback to be really useful. Add your test scripts too!

You have to pass every individual task to pass this assignment.

1. Implement the database according to the PowerDesigner PDM you find on the last page of this document. Once again you’ll find the scripts to get you started on OnderwijsOnline. Add to the constraint script the foreign key and the cascading rule declarations as depicted in the PowerDesigner PDM.
2. Implement constraints 1 through 11. In case a declarative implementation is possible, provide it.

Motivate your choice for (a combination of) stored procedures or triggers. Build at least 3 stored procedures and 3 triggers. If your choices result in less than 3 of each, create alternative solutions to in the end deliver at minimum 3 of each. Analyze what actions on which tables may cause the constraint to be violated (describe all possible scenarios). Implement the in your opinion most logical of the possibly many scenarios that may cause the constraint to be violated.

**FIRST** create the test cases you’ll need test your solution. Add tests with **multiple** rows if your solution is a trigger (this means at least tests with at minimum two allowed, two dis-allowed and a combination of rows). Only than start implementing the constraint.

So, every trigger should correctly handle multiple row SQL statements.

Minimize the use of variables. Use only one SQL-statement for a condition where ever possible.

Use correct error handling and transaction management in both Triggers and Stored Procedures like taught in the course.

Choose **two** of your procedural **constraints**. One which has no problems with non-repeatable read or phantoms, in a multi-user environment under the default isolation level (READ COMMITTED) and one which does.

Explain your choices by giving 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 problematic one.

For every scenario describe what kind of locks are acquired (e.g. s-locks and x-locks), when, why and for how long.

Find **two queries** to possibly optimize by adding indexes (discuss your ideas concerning the two queries to optimize with your lecturer). Use queries you wrote in task B. Now think of an index (per query) on one or more columns, either clustered or non-clustered, that may optimize the query performance. Describe your solution using plain text (do not just throw a bunch of code “over the wall”) 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)

1. 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 available with name HIST\_<table name> so HIST\_EMP, 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 timestamp column (type timestamp). The primary key is the only constraint of these history tables.

Assignment:  
  
- isolate the fixed boilerplate template T-SQL trigger code and determine where you can find (using the Information Schema Views) and retrieve the parameter values you have to generate into the fixed boilerplate template T-SQL code to produce the specific trigger;

- write at minimum two stored procedures that do the job, one generating the trigger code per table (on the basis of an input parameter), one calling this stored procedure as many times as there are non-history tables.

1. Employees can register themselves for courses on offer using an app provided by the Human Resources department. 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).   
     
   For reporting purposes a specific service account needs to be created allowing reporting tools full read access to all data.

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

**Product delivery 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.

Add all relevant T-SQL code. Don’t forget to index the document and add page numbers.

* The total file size should not exceed 10 MB
* Deliver all CREATE, ALTER and INSERT scripts and all test cases.

