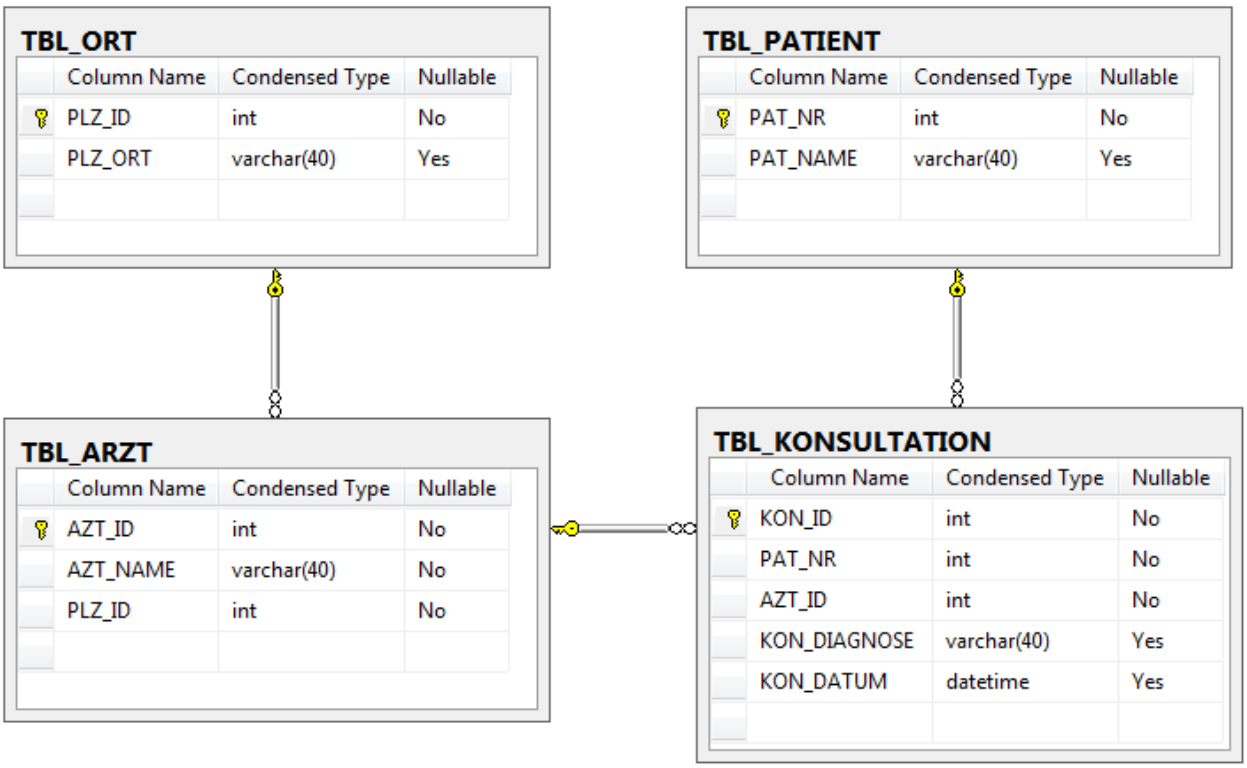
Aufgaben: Transact SQL “Trigger”

Datenmodell



Teil 1: Datenmodell implementieren (Voraussetzung)

Implementieren Sie das obige Datenmodel unter Ihrem Benutzer-Account in der SQL-Server Datenbank.

A1.1

Erstellen Sie die Textdatei CREATE_TABLES.SQL mit den CREATE TABLE Befehlen.

```
CREATE TABLE [user.]table ({column_element ¦ table_constraint}
            [,{column_element ¦ table_constraint} ] ...)
column_element      = column datatype [column constraint]
column constraint   = column [NULL] | [NOT NULL]
table constraint    = [{UNIQUE | PRIMARY KEY}]
```

A1.2

Erstellen Sie die Beziehungen zwischen den Tabellen (referenzielle Integrität).

```
ALTER TABLE
ADD CONSTRAINT name FOREIGN KEY (attr)
REFERENCES  tablename, (attr)
```

A1.3

Fügen Sie die Testdaten in die Tabellen ein. Prüfen Sie ob die referenzielle Integriät erfüllt wird.

```
INSERT INTO [user.]tabelle [ (column [,column] ...) ]
            VALUES (value [,value] ...)
```

Teil 2: Triggers

Syntax:

```
CREATE TRIGGER [Besitzer.]Triggername
ON [Besitzer.]Tabellenname | Sichtname
[FOR | AFTER | INSTEAD OF] {INSERT | UPDATE | DELETE}
[WITH ENCRYPTION]
AS Sql_Anweisungen
```

A3.1

Implementieren Sie die Anforderung, dass ein Datum (KON_DATUM) einer Konsultation nicht in der Zukunft liegen darf.

a) Lösung mit Check Constraint

```
ALTER TABLE tabelle ADD CONSTRAINT  ck_datum  CHECK (...)
```

```
ALTER TABLE [dbo].[TBL_KONSULTATION] ADD CONSTRAINT check_datum CHECK ([KON_DATUM] <= getdate())

SET DATEFORMAT dmy
go
insert TBL_KONSULTATION values(1234, 50001, 'Röteln', '09.11.2013 10:00:00')
```

b) Lösung mit INSTEAD Trigger

```
CREATE TRIGGER trigger_name ON tabelle INSTEAD OF INSERT
BEGIN ... END
```

```
create TRIGGER trInsert
    ON TBL_KONSULTATION
    INSTEAD OF INSERT
AS
BEGIN
        DECLARE @KonDatum datetime;

        SELECT @KonDatum = KON_DATUM FROM inserted;
        IF @KonDatum > GETDATE()
                RAISERROR ('Ungültiges Datum der Konsultation!', 16, 1);
        ELSE
                INSERT INTO TBL_KONSULTATION(PAT_NR, AZT_ID, KON_DIAGNOSE, KON_DATUM)
                        SELECT PAT_NR, AZT_ID, KON_DIAGNOSE, KON_DATUM FROM inserted;

END
```

A3.2

Stellen Sie sicher, dass ein Arzt pro Tag max. 5 Konsultationen ausführen darf.

a) Lösung mit AFTER INSERT

```
CREATE TRIGGER trigger_name ON tabelle AFTER INSERT AS
BEGIN ... END
```

```
create TRIGGER trInsert
    ON TBL_KONSULTATION
    AFTER INSERT
AS
BEGIN
        DECLARE @KonId int;
        DECLARE @AztId int;
        DECLARE @CountOf int;

        SELECT @KonId = KON_ID FROM inserted;
        SELECT @AztId = AZT_ID FROM inserted;

        SELECT @CountOf = count(*)
    FROM TBL_KONSULTATION
        WHERE AZT_ID = @AztId
        AND convert(date, KON_DATUM, 102) = convert(date, getdate(), 102);

        IF @CountOf > 5
        BEGIN
            DELETE FROM TBL_KONSULTATION
            WHERE KON_ID = @KonId;
            RAISERROR ('Zuviele Konsultationen!', 16, 1);
    END;
END;
```

b) Lösung mit INSTEAD Trigger

```
CREATE TRIGGER trigger_name ON tabelle INSTEAD OF INSERT
BEGIN ... END
```

```
create TRIGGER trInsert
    ON TBL_KONSULTATION
    INSTEAD OF INSERT
AS
BEGIN
        DECLARE @CountOf int;

        SELECT @CountOf = count(*)
        FROM TBL_KONSULTATION, inserted
        where TBL_KONSULTATION.AZT_ID = inserted.AZT_ID
        group by TBL_KONSULTATION.AZT_ID
        IF @CountOf > 5
                RAISERROR ('Zuviele Konsultationen!', 16, 1);
        ELSE
                INSERT INTO TBL_KONSULTATION(PAT_NR, AZT_ID, KON_DIAGNOSE, KON_DATUM)
                        SELECT PAT_NR, AZT_ID, KON_DIAGNOSE, KON_DATUM FROM inserted;
END
```

A3.3

Erstellen Sie ein Trigger welcher beim löschen eines Patienten automatisch alle zugehörigen Konsultationen mit löscht.

```
CREATE TRIGGER trigger_name ON tabelle AFTER [DELETE ¦ INSERT ¦ UPDATE] AS
BEGIN ... END
```

```
create trigger td_tbl_patient on TBL_PATIENT for delete as
begin
    declare
       @numrows  int,
       @errno    int

    select  @numrows = @@rowcount
    if @numrows = 0
       return

    /*  Delete all children in "TBL_KONSULTATION"  */
    delete TBL_KONSULTATION
    from   TBL_KONSULTATION t2, deleted t1
    where  t2.PAT_NR = t1.PAT_NR
    print 'Delete Trigger wurde ausgefuehrt'
    return
end
```

A3.4

Erstellen Sie ein Trigger welcher sämtliche Modifikationen in der Patiententabelle in einer Log-Tabelle protokolliert. Die Log-Tabelle muss vorgängig von Ihnen angelegt werden.
Attribute für Log-Tabelle (ID, Datum, Benutzername, Meldungstext)

```
CREATE TRIGGER trigger_name trigger ON tabelle AFTER UPDATE AS
BEGIN ... END
user_name( )
getdate()
```

```
create table TBL_LOG
(
    LOG_ID    numeric               identity,
    LOG_USER  nvarchar(20)          null    ,
    LOG_MSG   nvarchar(120)         null    ,
    LOG_TIME  datetime              null    ,
    constraint PK_TBL_LOG primary key (LOG_ID)
)
go

create trigger tu_tbl_patient on TBL_PATIENT for update as
begin
    declare
       @numrows  int,
       @errno    int

    select  @numrows = @@rowcount
    if @numrows = 0
       return

   insert into TBL_LOG (LOG_USER, LOG_TIME, LOG_MSG )
   select user_name(), getdate(), d.PAT_NAME  from inserted i, deleted d
 where d.PAT_NR = i.PAT_NR
       select 'Update Trigger wurde ausgefuehrt : ', @numrows
return
end
```

A3.5

Wenn der Anwender einen Datensatz in der Patiententabelle ändert, soll automatisch in einem Feld (z.B. letzte_bearbeitung (DateTime)) das Änderungsdatum gesetzt werden. Wie lässt sich diese Anforderung lösen?

```
CREATE TRIGGER trPatientUpdate
    ON dbo.TBL_PATIENT
    FOR UPDATE
AS
  IF (@@ROWCOUNT = 0)
    RETURN
  BEGIN
    UPDATE  TBL_PATIENT
        SET pat_last_modify = getdate()
        WHERE PAT_NR IN (SELECT PAT_NR FROM INSERTED)
  END
GO
```

Optional Erweiterung A3.4

```
go
create trigger tu_tbl_patient on TBL_PATIENT after
update,insert,delete
as
begin
    declare
       @numrows  int,
       @errno    int,
        @event_type varchar(20)

    select  @numrows = @@rowcount
    if @numrows = 0
       return

     IF EXISTS(SELECT * FROM inserted)
     begin
          IF EXISTS(SELECT * FROM deleted)
               SELECT @event_type = 'update'
          ELSE
               SELECT @event_type = 'insert'
     end
     ELSE
     begin
          IF EXISTS(SELECT * FROM deleted)
               SELECT @event_type = 'delete'
          ELSE
               --no rows affected - cannot determine event
```

```
                SELECT @event_type = 'unknown'
     end

     if @event_type = 'insert'
     begin
        insert into TBL_LOG (LOG_USER, LOG_TIME, LOG_MSG )
               select user_name(), getdate(), 'inserted - '
+ PAT_NAME  from inserted
        select 'Insert Trigger wurde ausgefuehrt : ',
@numrows
     end
     else if @event_type = 'update'
     begin
        insert into TBL_LOG (LOG_USER, LOG_TIME, LOG_MSG )
               select user_name(), getdate(), 'updated - ' +
d.PAT_NAME
                        from inserted i, deleted d
                         where d.PAT_NR = i.PAT_NR
        select 'Update Trigger wurde ausgefuehrt : ',
@numrows
     end
     else if @event_type = 'delete'
     begin
        insert into TBL_LOG (LOG_USER, LOG_TIME, LOG_MSG )
               select user_name(), getdate(), 'deleted - ' +
PAT_NAME  from deleted
        select 'Delete Trigger wurde ausgefuehrt : ',
@numrows
     end
return
end
```