## Laboration 5 och Teoriprov med kryssfrågor

**Ni redovisar labben genom att spela in en film där ni kör all er kod i Live SQL och visar att den fungerar enligt instruktioner i labben. Den avslutande teoridelen ska också vara med i er film.**

Under den här roliga labben kommer Du att få skapa ett litet banksystem åt Gagnefs sparbank. Den före detta potatisodlaren Thomas Kvist, som för övrigt är mycket indolent, har tillträtt posten som bankdirektör på den nya banken. Thomas har, genom en tuff akademisk verksamhetsanalys, arbetat fram något som kan liknas vid en systemspecifikation. Nedan hittar Du en översikt över vilka tabeller du måste skapa för att lösa uppgiften.

SQL> desc bankkund

Name Null? Type

----------------------------------------- -------- ------------

PNR NOT NULL VARCHAR2(11)

FNAMN NOT NULL VARCHAR2(25)

ENAMN NOT NULL VARCHAR2(25)

PASSWD NOT NULL VARCHAR2(16)

SQL> desc kontotyp

Name Null? Type

----------------------------------------- -------- ------------

KTNR NOT NULL NUMBER(6)

KTNAMN NOT NULL VARCHAR2(20)

RÄNTA NOT NULL NUMBER(5,2)

SQL> desc ränteändring

Name Null? Type

----------------------------------------- -------- ------------

RNR NOT NULL NUMBER(6)

KTNR NOT NULL NUMBER(6)

RÄNTA NOT NULL NUMBER(5,2)

RNR\_DATUM NOT NULL DATE

SQL> desc konto

Name Null? Type

----------------------------------------- -------- ------------

KNR NOT NULL NUMBER(8)

KTNR NOT NULL NUMBER(6)

REGDATUM NOT NULL DATE

SALDO NUMBER(10,2)

SQL> desc kontoägare

Name Null? Type

----------------------------------------- -------- ------------

RADNR NOT NULL NUMBER(9)

PNR NOT NULL VARCHAR2(11)

KNR NOT NULL NUMBER(8)

SQL> desc uttag

Name Null? Type

----------------------------------------- -------- ------------

RADNR NOT NULL NUMBER(9)

PNR NOT NULL VARCHAR2(11)

KNR NOT NULL NUMBER(8)

BELOPP NUMBER(10,2)

DATUM NOT NULL DATE

SQL> desc insättning

Name Null? Type

----------------------------------------- -------- ------------

RADNR NOT NULL NUMBER(9)

PNR NOT NULL VARCHAR2(11)

KNR NOT NULL NUMBER(8)

BELOPP NUMBER(10,2)

DATUM NOT NULL DATE

SQL> desc överföring

Name Null? Type

----------------------------------------- -------- ------------

RADNR NOT NULL NUMBER(9)

PNR NOT NULL VARCHAR2(11)

FRÅN\_KNR NOT NULL NUMBER(8)

TILL\_KNR NOT NULL NUMBER(8)

BELOPP NUMBER(10,2)

DATUM NOT NULL DATE

**OBS!** Flera uppgifter i laborationen ska testas/verifieras. Att testa/verifiera innebär att man först redovisar hur testet genomförts (dvs SQL-kod som använts för de olika alternativen) och därefter redovisar man output efter körning/exekvering av koden. Detta ska visas i er inspelning.

**Uppgift 1**

Skapa tabeller och constraints enligt ovanstående tabellbeskrivningar. Den kolumn som står överst i respektive definition är primary key för tabellen. **Referensintegritet** framgår tydligt via namngivningen av kolumnerna **utom i ett fall**, tabellen överförings två kolumner från\_knr och till\_knr refererar bägge till konto(knr). Det framgår att bankdirektören använt *action modeling* som metod. Detta då handlingarna **ta ut pengar**, **sätta in pengar** och **överföring av pengar mellan konton** representeras i varsin tabell med meningsfulla namn. Kolumnen pnr i tabellen bankkund är ett personnummer.

Implementera datamodellen med SQL-DDL i databasen. ( = skriv SQL-kod som skapar ovanstående tabeller och constraints och kör denna i SQL). I denna uppgift ska du redovisa din DDL-kod.

**Svar 1:**

-- Skapar table för bankkund--

**CREATE** **TABLE** bankkund **(**

PNR VARCHAR2**(**11**)** **NOT** **NULL,**

FNAMN VARCHAR2**(**25**)** **NOT** **NULL,**

ENAMN VARCHAR2**(**25**)** **NOT** **NULL,**

PASSWD VARCHAR2**(**16**)** **NOT** **NULL**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** bankkund **ADD** **CONSTRAINT** PK\_bankkund **PRIMARY** **KEY** **(**PNR**);**

-- Test/verifierring --

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'BANKKUND'**;**

-- Skapar table för bankkund--

**CREATE** **TABLE** kontotyp **(**

KTNR NUMBER**(**6**)** **NOT** **NULL,**

KTNAMN VARCHAR2**(**20**)** **NOT** **NULL,**

RÄNTA NUMBER**(**5**,**2**)** **NOT** **NULL**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** kontotyp **ADD** **CONSTRAINT** PK\_kontotyp **PRIMARY** **KEY** **(**KTNR**);**

-- Test/verifierring --

**SELECT** **\*** **from** kontotyp**;**

**DESC** kontotyp**;**

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'KONTOTYP'**;**

-- Skapar table för ränteändring--

**CREATE** **TABLE** ränteändring **(**

RNR NUMBER**(**6**)** **NOT** **NULL,**

KTNR NUMBER**(**6**)** **NOT** **NULL,**

RÄNTA NUMBER**(**5**,**2**)** **NOT** **NULL,**

RNR\_DATUM DATE **NOT** **NULL**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** ränteändring **ADD** **CONSTRAINT** PK\_ränteändring **PRIMARY** **KEY** **(**RNR**);**

-- Lägger till constraint och foreign key --

**ALTER** **TABLE** ränteändring

**ADD** **CONSTRAINT** FK\_ränteändring\_kontotyp **FOREIGN** **KEY** **(**KTNR**)** **REFERENCES** kontotyp**(**KTNR**);**

-- Test/verifierring --

**SELECT** **\*** **from** ränteändring**;**

**DESC** ränteändring**;**

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'RÄNTEÄNDRING'**;**

-- Skapar table för konto--

**CREATE** **TABLE** konto **(**

KNR NUMBER**(**8**)** **NOT** **NULL,**

KTNR NUMBER**(**6**)** **NOT** **NULL,**

REGDATUM DATE **NOT** **NULL,**

SALDO NUMBER**(**10**,**2**)**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** konto **ADD** **CONSTRAINT** PK\_konto **PRIMARY** **KEY** **(**KNR**);**

-- Lägger till constraint och foreign key --

**ALTER** **TABLE** konto

**ADD** **CONSTRAINT** FK\_konto\_kontotyp **FOREIGN** **KEY** **(**KTNR**)** **REFERENCES** kontotyp**(**KTNR**);**

-- Test/verifierring --

**SELECT** **\*** **from** konto**;**

**DESC** ränteändring**;**

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'KONTO'**;**

-- Skapar table för kontoägare--

**CREATE** **TABLE** kontoägare **(**

RADNR NUMBER**(**9**)** **NOT** **NULL,**

PNR VARCHAR2**(**11**)** **NOT** **NULL,**

KNR NUMBER**(**8**)** **NOT** **NULL**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** kontoägare **ADD** **CONSTRAINT** PK\_kontoägare **PRIMARY** **KEY** **(**RADNR**);**

-- Lägger till constraint och foreign key --

**ALTER** **TABLE** kontoägare

**ADD** **CONSTRAINT** FK\_kontoägare\_kund

**FOREIGN** **KEY** **(**PNR**)** **REFERENCES** BANKKUND**(**PNR**);**

**ALTER** **TABLE** kontoägare

**ADD** **CONSTRAINT** FK\_kontoägare\_konto

**FOREIGN** **KEY** **(**KNR**)** **REFERENCES** KONTO**(**KNR**);**

-- Test/verifierring --

**SELECT** **\*** **from** kontoägare**;**

**DESC** ränteändring**;**

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'KONTOÄGARE'**;**

-- Skapar table för uttag--

**CREATE** **TABLE** uttag **(**

RADNR NUMBER**(**9**)** **NOT** **NULL,**

PNR VARCHAR2**(**11**)** **NOT** **NULL,**

KNR NUMBER**(**8**)** **NOT** **NULL,**

BELOPP NUMBER**(**10**,** 2**),**

DATUM DATE **NOT** **NULL**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** uttag **ADD** **CONSTRAINT** PK\_uttag **PRIMARY** **KEY** **(**RADNR**);**

-- Lägger till constraint och foreign key --

**ALTER** **TABLE** uttag

**ADD** **CONSTRAINT** FK\_uttag\_kund **FOREIGN** **KEY** **(**PNR**)** **REFERENCES** bankkund**(**PNR**);**

**ALTER** **TABLE** uttag

**ADD** **CONSTRAINT** FK\_uttag\_konto **FOREIGN** **KEY** **(**KNR**)** **REFERENCES** konto**(**KNR**);**

-- Test/verifierring --

**SELECT** **\*** **from** uttag**;**

**DESC** uttag**;**

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'UTTAG'**;**

-- Skapar table för insättning--

**CREATE** **TABLE** insättning **(**

RADNR NUMBER**(**9**)** **NOT** **NULL,**

PNR VARCHAR2**(**11**)** **NOT** **NULL,**

KNR NUMBER**(**8**)** **NOT** **NULL,**

BELOPP NUMBER**(**10**,** 2**),**

DATUM DATE **NOT** **NULL**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** insättning **ADD** **CONSTRAINT** PK\_insättning **PRIMARY** **KEY** **(** RADNR **);**

-- Lägger till constraint och foreign key --

**ALTER** **TABLE** insättning

**ADD** **CONSTRAINT** FK\_insättning\_kund **FOREIGN** **KEY** **(** PNR **)**

**REFERENCES** BANKKUND **(** PNR **);**

**ALTER** **TABLE** insättning

**ADD** **CONSTRAINT** FK\_insättning\_konto **FOREIGN** **KEY** **(** KNR **)**

**REFERENCES** KONTO **(** KNR **);**

-- Test/verifierring --

**SELECT** **\*** **from** insättning**;**

**DESC** insättning**;**

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'INSÄTTNING'**;**

-- Skapar table för överföring--

**CREATE** **TABLE** överföring **(**

RADNR NUMBER**(**9**)** **NOT** **NULL,**

PNR VARCHAR2**(**11**)** **NOT** **NULL,**

FRÅN\_KNR NUMBER**(**8**)** **NOT** **NULL,**

TILL\_KNR NUMBER**(**8**)** **NOT** **NULL,**

BELOPP NUMBER**(**10**,**2**),**

DATUM DATE **NOT** **NULL**

**);**

-- Lägger till constraint och primary key --

**ALTER** **TABLE** överföring **ADD** **CONSTRAINT** PK\_överföring **PRIMARY** **KEY** **(**RADNR**);**

-- Lägger till constraint och foreign key --

**ALTER** **TABLE** överföring

**ADD** **CONSTRAINT** FK\_överföring\_kund **FOREIGN** **KEY** **(**PNR**)** **REFERENCES** bankkund**(**PNR**);**

**ALTER** **TABLE** överföring

**ADD** **CONSTRAINT** FK\_överföring\_från **FOREIGN** **KEY** **(**FRÅN\_KNR**)** **REFERENCES** konto**(**KNR**);**

**ALTER** **TABLE** överföring

**ADD** **CONSTRAINT** FK\_överföring\_till **FOREIGN** **KEY** **(**TILL\_KNR**)** **REFERENCES** konto**(**KNR**);**

-- Test/verifierring --

**SELECT** **\*** **from** överföring**;**

**DESC** överföring**;**

**SELECT** constraint\_name**,** constraint\_type

**FROM** user\_constraints

**WHERE** table\_name **=** 'ÖVERFÖRING'**;**

**Uppgift 2**

Nu är det dags att du ritar en datamodell över de tabeller som du skapade i uppgift 1.

Du klistrar in en bild på din datamodell i labbrapporten. Eller så kan du bifoga den som en jpg-bild.

**OBS! Fula oläsliga och röriga datamodeller underkänns direkt.** **Använd principer från föreläsningen för att gruppera tabeller enligt: Aktiva objekt, Komplexa objekt och passiva objekt.**

**Svar 2:**

En bild som visar text, diagram, svart och vit, Parallell

AI-genererat innehåll kan vara felaktigt.

**Uppgift 3**

Skapa en trigger med namnet biufer\_bankkund som startar **före** insert eller update av kolumnen passwd i tabellen bankkund. Triggern skall kontrollera att lösenordet är exakt sex tecken långt, varken mer eller mindre. Om inte detta krav uppfylls skall triggern stoppa transaktionen och meddela att detta var felet.

**Svar 3**

**CREATE** **OR** **REPLACE** **TRIGGER** biufer\_bankkund

**BEFORE** **INSERT** **OR** **UPDATE** **ON** bankkund

**FOR** **EACH** **ROW**

**WHEN** **(length(new.**passwd**)** **<>** 6**)**

**BEGIN**

RAISE\_APPLICATION\_ERROR**(**-20001**,** 'Ditt lösenord måste vara exakt 6 tecken långt.'**);**

**END;**

**/**

-- Test/verifierring --

**SELECT** **\***

**FROM** user\_triggers

**WHERE** trigger\_name **=** 'BIUFER\_BANKKUND'**;**

-- ORA-20001: Lösenordet måste vara exakt 6 tecken långt. Lång--

**insert** **into** bankkund**(**pnr**,** fnamn**,** enamn**,** passwd**)**

**VALUES(**'900101-0101'**,** 'HTML'**,** 'CSS'**,** 'javascript'**)**

--ORA-20001: Lösenordet måste vara exakt 6 tecken långt. Kort --

**insert** **into** bankkund**(**pnr**,** fnamn**,** enamn**,** passwd**)**

**VALUES(**'890101-0101'**,** 'Responsive'**,** 'Design'**,** 'react'**)**

---------------------------------------

--1 row inserted.--

**insert** **into** bankkund**(**pnr**,** fnamn**,** enamn**,** passwd**)**

**VALUES(**'880101-0101'**,** 'MY'**,** 'SQL'**,** 'oracle'**)**

---------------------------------------

--test-

**select** **\*** **from** BANKKUND**;**

-- tabort inmattade data from bakkund--

**DELETE** **FROM** BANKKUND **WHERE** fnamn**=**'MY'**;**

**Uppgift 4**

Skapa en procedur med namnet do\_bankkund. Proceduren skall användas för att lägga till nya kunder i tabellen bankkund. **Skapa inparametrarna i ordningen** pnr, fnamn, enamn, passwd.

**Svar 4**

-- Här skapas eller ersättas proceduren "do\_bankkund". Den används för att lägga till nya kunder i tabellen "bankkund"

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_bankkund **(**

-- Inparametrar:

p\_pnr **IN** bankkund**.**pnr**%TYPE,** -- Personnummer

p\_fnamn **IN** bankkund**.**fnamn**%TYPE,** -- Förnamn

p\_enamn **IN** bankkund**.**enamn**%TYPE,** -- Efternamn

p\_passwd **IN** bankkund**.**passwd**%TYPE** -- Lösenord

**)**

**IS**

**BEGIN**

-- Lägger till en ny rad i tabellen bankkund med värden från inparametrarna

**INSERT** **INTO** bankkund **(**pnr**,** fnamn**,** enamn**,** passwd**)**

**VALUES** **(**p\_pnr**,** p\_fnamn**,** p\_enamn**,** p\_passwd**);**

**END;**

**/**

-- Test/verifierring --

**BEGIN**

do\_bankkund**(**'870101-0101'**,** 'Data'**,** 'Bas'**,** 'höst24'**);**

**commit;**

**END;**

**/**

**select** **\***

**from** bankkund**;**

**SELECT** **object\_name,** status

**FROM** user\_objects

**WHERE** object\_type **=** 'PROCEDURE'

**AND** **object\_name** **=** 'DO\_BANKKUND'**;**

**Uppgift 5**

Lägg till fyra kunder med hjälp av proceduren do\_bankkund. Testa om triggern biufer\_bankkund fungerar genom att försöka stoppa in ett lösenord som är fel. Gör detta genom att kopiera och klistra in nedanstående: Kör **triggertest** före "Start copy..":

**Triggertest** = begin do\_bankkund('691124-4478','Bo','Ek','qwe'); end;

-------------------------Start copy and paste---------------------------------

BEGIN

do\_bankkund('540126-1111','Hans','Rosendahl','olle45');

do\_bankkund('560126-1111','Hans','Rosengårdh','olle85');

do\_bankkund('540126-1457','Lina','Karlsson','asdfgh');

do\_bankkund('691124-4478','Leena','Kvist','qwerty');

COMMIT;

END;

/

-------------------------End copy and paste---------------------------------

1. Verifiera trigger-test med för få tecken
2. Verifiera trigger-test med för många tecken
3. Verifiera därefter innehåll i tabellen bankkund efter ’copy paste’ vilket då utgör bevis för att triggern fungerar med rätt inparameter för lösenord

**Svar 5**

/\* Triggertest med nedan insert men den funger inte då lösenordet är mindre än 6 ord.

Får fel meddelandet ERROR at line 1:

ORA-20001: Lösenordet måste vara exakt 6 tecken långt.

ORA-06512: at "SQL\_DC0EVZB8UCJV124OU9JVZHDXEL.BIUFER\_BANKKUND", line 5

ORA-04088: error during execution of trigger 'SQL\_DC0EVZB8UCJV124OU9JVZHDXEL.BIUFER\_BANKKUND'

ORA-06512: at "SQL\_DC0EVZB8UCJV124OU9JVZHDXEL.DO\_BANKKUND", line 12

ORA-06512: at line 1.\*/

**begin**

do\_bankkund**(**'691124-4478'**,**'Bo'**,**'Ek'**,**'qwe'**);**

**end;**

**begin**

do\_bankkund**(**'691124-4478'**,**'Bo'**,**'Ek'**,**'qwe12356'**);**

**end;**

-- Dessa insert fungerar då dem uppfyller kriterierna--

**BEGIN**

do\_bankkund**(**'540126-1111'**,**'Hans'**,**'Rosendahl'**,**'olle45'**);**

do\_bankkund**(**'560126-1111'**,**'Hans'**,**'Rosengårdh'**,**'olle85'**);**

do\_bankkund**(**'540126-1457'**,**'Lina'**,**'Karlsson'**,**'asdfgh'**);**

do\_bankkund**(**'691124-4478'**,**'Leena'**,**'Kvist'**,**'qwerty'**);**

**COMMIT;**

**END;**

**/**

-- tester med ---

**select** **\* from** bankkund**;**

**Uppgift 6**

**Börja med** att skapa en **sekvens** med namnet radnr\_seq. Denna skall vi använda till att skapa primärnyckelvärden för alla kolumner med namnet radnr.

Lägg nu till rader i tabellerna kontotyp, konto och kontoägare genom att kopiera och klistra in nedanstående i SQL Developer.

-------------------------Start copy and paste---------------------------------

INSERT INTO kontotyp(ktnr,ktnamn,ränta)

VALUES(1,'bondkonto',3.4);

INSERT INTO kontotyp(ktnr,ktnamn,ränta)

VALUES(2,'potatiskonto',4.4);

INSERT INTO kontotyp(ktnr,ktnamn,ränta)

VALUES(3,'griskonto',2.4);

COMMIT;

INSERT INTO konto(knr,ktnr,regdatum,saldo)

VALUES(123,1,SYSDATE - 321,0);

INSERT INTO konto(knr,ktnr,regdatum,saldo)

VALUES(5899,2,SYSDATE - 2546,0);

INSERT INTO konto(knr,ktnr,regdatum,saldo)

VALUES(5587,3,SYSDATE - 10,0);

INSERT INTO konto(knr,ktnr,regdatum,saldo)

VALUES(8896,1,SYSDATE - 45,0);

COMMIT;

INSERT INTO kontoägare(radnr,pnr,knr)

VALUES(radnr\_seq.NEXTVAL,'540126-1111',123);

INSERT INTO kontoägare(radnr,pnr,knr)

VALUES(radnr\_seq.NEXTVAL,'691124-4478',123);

INSERT INTO kontoägare(radnr,pnr,knr)

VALUES(radnr\_seq.NEXTVAL,'540126-1111',5899);

INSERT INTO kontoägare(radnr,pnr,knr)

VALUES(radnr\_seq.NEXTVAL,'691124-4478',8896);

COMMIT;

-------------------------End copy and paste---------------------------------

**Svar 6**

-- skapar en sekvens med namnet radnr\_seq. Denna används till att skapa primärnyckelvärden för alla kolumner med namnet radnr. --

**CREATE** **SEQUENCE** radnr\_seq

**START** **WITH** 1

**INCREMENT** **BY** 1**;**

-- Test--

**SELECT** sequence\_name

**FROM** user\_sequences

**WHERE** sequence\_name **=** 'RADNR\_SEQ'**;**

-- Lägger nu till rader i tabellerna kontotyp, konto och kontoägare genom att kopiera och klistra in nedanstående i SQL Developer.--

**INSERT** **INTO** kontotyp**(**ktnr**,**ktnamn**,**ränta**)**

**VALUES(**1**,**'bondkonto'**,**3.4**);**

**INSERT** **INTO** kontotyp**(**ktnr**,**ktnamn**,**ränta**)**

**VALUES(**2**,**'potatiskonto'**,**4.4**);**

**INSERT** **INTO** kontotyp**(**ktnr**,**ktnamn**,**ränta**)**

**VALUES(**3**,**'griskonto'**,**2.4**);**

**COMMIT;**

**INSERT** **INTO** konto**(**knr**,**ktnr**,**regdatum**,**saldo**)**

**VALUES(**123**,**1**,SYSDATE** **-** 321**,**0**);**

**INSERT** **INTO** konto**(**knr**,**ktnr**,**regdatum**,**saldo**)**

**VALUES(**5899**,**2**,SYSDATE** **-** 2546**,**0**);**

**INSERT** **INTO** konto**(**knr**,**ktnr**,**regdatum**,**saldo**)**

**VALUES(**5587**,**3**,SYSDATE** **-** 10**,**0**);**

**INSERT** **INTO** konto**(**knr**,**ktnr**,**regdatum**,**saldo**)**

**VALUES(**8896**,**1**,SYSDATE** **-** 45**,**0**);**

**COMMIT;**

**INSERT** **INTO** kontoägare**(**radnr**,**pnr**,**knr**)**

**VALUES(**radnr\_seq**.NEXTVAL,**'540126-1111'**,**123**);**

**INSERT** **INTO** kontoägare**(**radnr**,**pnr**,**knr**)**

**VALUES(**radnr\_seq**.NEXTVAL,**'691124-4478'**,**123**);**

**INSERT** **INTO** kontoägare**(**radnr**,**pnr**,**knr**)**

**VALUES(**radnr\_seq**.NEXTVAL,**'540126-1111'**,**5899**);**

**INSERT** **INTO** kontoägare**(**radnr**,**pnr**,**knr**)**

**VALUES(**radnr\_seq**.NEXTVAL,**'691124-4478'**,**8896**);**

**COMMIT;**

-- Tester med ---

**select** **\***

**from** kontotyp**;**

**select** **\***

**from** kontoägare**;**

**select** **\***

**from** konto**;**

-- och även med hjälp av join kan jag testa samtliga tabellerna--

**SELECT**

bk**.**fnamn **||** ' ' **||** bk**.**enamn **AS** namn**,**

ka**.**pnr**,**

k**.**knr **AS** kontonummer**,**

k**.**regdatum**,**

k**.**saldo**,**

kt**.**ktnr**,**

kt**.**ktnamn**,**

kt**.**ränta

**FROM** konto k

**LEFT** **JOIN** kontoägare ka **ON** k**.**knr **=** ka**.**knr

**LEFT** **JOIN** bankkund bk **ON** ka**.**pnr **=** bk**.**pnr

**JOIN** kontotyp kt **ON** k**.**ktnr **=** kt**.**ktnr

**ORDER** **BY** k**.**knr**;**

/\* Visar alla konton i banken, oavsett om de har en ägare eller inte.

Jag använder LEFT JOIN så att även konton utan ägare fortfarande visas i resultatet.\*/

**Uppgift 7**

Skapa en funktion med namnet logga\_in. Denna skall returnera 1 eller 0 beroende av hur det gick. Inloggning sker med att en bankund lämnar sitt personnummer och lösenord, dvs kolumnerna pnr och passwd i tabellen bankkund. **Testa att funktionen fungerar** genom att göra:

SELECT logga\_in('pnr','lösenord')

FROM dual;

Verifiera att **logga\_in** ger 1 vid korrekt inloggning, och 0 vid felaktig genom att:

1. Testa med felaktigt lösenord
2. Testa med felaktigt personnummer
3. Testa med korrekta parametrar

**Svar 7**

**CREATE** **OR** **REPLACE** **FUNCTION** LOGGA\_IN **(**

P\_PNR **IN** BANKKUND**.**PNR**%TYPE,**

P\_PASSWD **IN** BANKKUND**.**PASSWD**%TYPE**

**)**

**RETURN** NUMBER **AS**

V\_PASSWD BANKKUND**.**PASSWD**%TYPE;**

**BEGIN**

**SELECT**

PASSWD

**INTO** V\_PASSWD

**FROM**

BANKKUND

**WHERE**

PNR **=** P\_PNR**;**

**IF** V\_PASSWD **=** P\_PASSWD **THEN**

**RETURN** 1**;**

**ELSE**

**RETURN** 0**;**

**END** **IF;**

**END;**

-- Testar att funktionen fungerar genom att göra: --

**SELECT** **OBJECT\_NAME,** OBJECT\_TYPE

**FROM** USER\_OBJECTS

**WHERE** OBJECT\_TYPE **=** 'FUNCTION'

**AND** **OBJECT\_NAME** **=** 'LOGGA\_IN'**;**

-- och --

-- 1. Testa med felaktigt lösenord--

**SELECT** LOGGA\_IN**(**'540126-1111'**,** 'felaktigtlösenord'**)** **FROM** dual**;**

-- 2. Testa med felaktigt personnummer--

**SELECT** LOGGA\_IN**(**'540126-1110'**,** 'olle45'**)** **FROM** dual**;**

-- 3. Testa med korrekta parametrar --

**SELECT** LOGGA\_IN**(**'540126-1111'**,** 'olle45'**)** **FROM** dual**;**

**Uppgift 8**

Skapa en funktion med namnet get\_saldo. Denna skall returnera aktuellt saldo för det kontonummer (knr) som skickas in till funktionen**. Testa att funktionen fungerar!**

Verifiera att **get\_saldo** returnerar korrekt saldo genom att:

1. Verifiera innehållet i tabellen KONTO
2. Testa med korrekt kontonummer
3. Testa med felaktigt kontonummer

**Svar 8**

**CREATE** **OR** **REPLACE** **FUNCTION** get\_saldo **(**

p\_knr **IN** konto**.**knr**%TYPE** -- Inparameter: kontonummer

**)**

**RETURN** NUMBER **AS**

v\_saldo konto**.**saldo**%TYPE;** -- Variabel för att lagra saldo

**BEGIN**

-- Hämta saldo för det givna kontonumret

**SELECT** saldo **INTO** v\_saldo

**FROM** konto

**WHERE** knr **=** p\_knr**;**

-- Returnera saldo

**RETURN** v\_saldo**;**

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

-- Om inget konto med det angivna kontonumret finns

RAISE\_APPLICATION\_ERROR**(**-20002**,** 'Kontonumret finns inte.'**);**

**END;**

**/**

-- test om funktionen finns --

**SELECT** **object\_name,** status

**FROM** user\_objects

**WHERE** object\_type **=** 'FUNCTION'

**AND** **object\_name** **=** 'GET\_SALDO'**;**

-- test om saldo--'

**select** get\_saldo **(**5899**)**

**from** dual

-- varifera med konto table--

**select** **\*** **from** konto**;**

-- se ett fel-kontonummer--

**SELECT** get\_saldo**(**13**)** **FROM** dual**;**

**Uppgift 9**

Skapa en funktion med namnet get\_behörighet. Denna tar två inparametrar, pnr och knr, och returnerar 1 (ett) om personen i fråga är ägare till belastningskontot eller 0 (noll) om personen **inte** är ägare. All data som behövs för detta finns i tabellen kontoägare. Med belastningskonto avses i denna kontext det konto som det görs uttag ifrån, dvs vid uttag och överföringar. Vid överföringar är belastningskontot kolumnen från\_konto. **Testa att funktionen fungerar!**

Verifiera att **get\_behörighet** fungerar genom att:

1. Verifiera innehållet i tabellen KONTOÄGARE
2. Testa med fel personnummer
3. Testa med fel kontonummer
4. Testa med rätt parametrar

**Svar 9**

-- Skapar eller ersätter en funktion som kontrollerar om en viss person (pnr)

-- har behörighet till ett visst konto (knr)

**CREATE** **OR** **REPLACE** **FUNCTION** get\_behörighet **(**

f\_pnr **IN** bankkund**.**pnr**%TYPE,** -- IN-parameter: personnummer

f\_knr **IN** kontoägare**.**knr**%TYPE** -- IN-parameter: kontonummer

**)**

**RETURN** NUMBER **AS**

v\_pnr bankkund**.**pnr**%TYPE;** -- Variabel för att lagra resultatet av SELECT-frågan

**BEGIN**

-- Hämtar personnummer från tabellen kontoägare där både pnr och knr matchar

**SELECT** pnr **INTO** v\_pnr

**FROM** kontoägare

**WHERE** pnr **=** f\_pnr **AND** knr **=** f\_knr**;**

-- Om vi fick ett resultat som matchar personnumret, returnera 1 (behörig)

**IF** v\_pnr **=** f\_pnr **THEN**

**RETURN** 1**;**

**ELSE**

-- Annars returnera 0 (ej behörig)

**RETURN** 0**;**

**END** **IF;**

-- Om SELECT-satsen inte hittar någon rad, hanteras det här:

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

-- Returnera 0 om ingen matchande rad hittades (ej behörig)

**RETURN** 0**;**

**END;**

**/**

-- Test och varjfera--

**SELECT** **object\_name,** status

**FROM** user\_objects

**WHERE** object\_type **=** 'FUNCTION'

**AND** **object\_name** **=** 'GET\_BEHÖRIGHET'**;**

**SELECT** **\*** **FROM** "KONTOÄGARE"**;**

-- 0, 2. Testa med fel personnummer--

**SELECT** get\_behörighet**(**'540126-1113'**,** 123**)** **AS** test\_fel\_pnr **FROM** dual**;**

-- 0, 3. Testa med fel kontonummer--

**SELECT** get\_behörighet**(**'540126-1111'**,** 9999**)** **AS** test\_fel\_knr **FROM** dual**;**

--1, 4. Testa med rätt parametrar--

**SELECT** get\_behörighet**(**'540126-1111'**,** 123**)** **AS** test\_korrekt **FROM** dual**;**

**Uppgift 10**

Skapa en trigger med namnet aifer\_insättning. Denna trigger skall se till att saldot blir rätt efter insättning på ett visst konto. Trigger skall starta **efter** insert på tabellen insättning.

**Svar 10**

-- Skapa eller ersätt triggern som uppdaterar saldot efter en insättning

**CREATE** **OR** **REPLACE** **TRIGGER** aifer\_insättning

**AFTER** **INSERT** **ON** insättning -- Triggern aktiveras efter varje insättning i tabellen "insättning"

**FOR** **EACH** **ROW** -- Triggern körs för varje ny rad som läggs till i tabellen "insättning"

**BEGIN**

-- Kontrollera om beloppet inte är NULL och konto finns

**IF** **:NEW.**belopp **IS** **NOT** **NULL** **AND** **:NEW.**knr **IS** **NOT** **NULL** **THEN**

-- Uppdaterar saldot i tabellen konto baserat på insättningen

**UPDATE** konto

**SET** saldo **=** saldo **+** **:NEW.**belopp -- Lägg till det nya beloppet som just har satts in på kontot

**WHERE** knr **=** **:NEW.**knr**;** -- Matchar kontonumret från den insatta raden (insättning) med tabellen "konto"

**END** **IF;**

**END;**

**/**

-- Test/verifierring --

**SELECT** **\***

**FROM** user\_triggers

**WHERE** trigger\_name **=** 'AIFER\_INSÄTTNING'**;**

-- inget saldo--

**SELECT** **\*** **from** konto**;**

-- insert saldo--

**INSERT** **INTO** insättning **(**radnr**,** pnr**,** knr**,** belopp**,** datum**)**

**VALUES** **(**radnr\_seq**.NEXTVAL,** '540126-1111'**,** '123'**,** 2500**,** **SYSDATE);**

-- ny saldo--

**SELECT** **\*** **from** konto**;**

**Uppgift 11**

Skapa en trigger med namnet bifer\_uttag. Denna trigger skall se till att man inte kan ta ut mer pengar än vad det finns på kontot. Triggern skall starta **före** insert på tabellen uttag. **OBS!** Triggern skall anropa funktionen get\_saldo för att utföra kontrollen.

**Svar 11**

**CREATE** **OR** **REPLACE** **TRIGGER** bifer\_uttag

**BEFORE** **INSERT** **ON** uttag -- Triggern aktiveras innan ett uttag görs i tabellen "uttag"

**FOR** **EACH** **ROW**

**DECLARE**

v\_saldo konto**.**saldo**%TYPE;** -- Variabel för att hålla det aktuella saldot från kontot

**BEGIN**

-- Anropa funktionen get\_saldo för att hämta saldot för det aktuella kontot (knr)

v\_saldo **:=** get\_saldo**(:NEW.**knr**);** -- :NEW.knr används för att referera till det kontonummer som ska användas i uttaget

-- Kontrollera om beloppet som ska tas ut är större än det aktuella saldot

**IF** **(**v\_saldo **-** **:new.**belopp**)** **<** 0 **THEN**

-- Om uttagsbeloppet är större än saldot, stoppa insättningen och ge ett felmeddelande

RAISE\_APPLICATION\_ERROR**(**-20001**,** 'Du har inet tillräckligt med pengar på kontot för detta uttag.'**);**

**END** **IF;**

**END;**

**/**

-- Test/verifierring --

**SELECT** **\***

**FROM** user\_triggers

**WHERE** trigger\_name **=** 'BIFER\_UTTAG'**;**

--- ORA-20001: Du har inet tillräckligt med pengar på kontot för detta uttag.--

**INSERT** **INTO** uttag **(**radnr**,** pnr**,** knr**,** belopp**,** datum**)**

**VALUES** **(**radnr\_seq**.NEXTVAL,** '540126-1111'**,** '123'**,** 8000**,** **SYSDATE);**

**SELECT** **\*** **from** konto**;**

**SELECT** **\*** **from** uttag**;**

**Uppgift 12**

Skapa en trigger med namnet aifer\_uttag. Triggern skall se till att saldot på kontot blir rätt efter uttag. Triggern skall starta **efter** insert på tabellen uttag.

**Svar 12**

**CREATE** **OR** **REPLACE** **TRIGGER** aifer\_uttag

**AFTER** **INSERT** **ON** uttag -- Triggern aktiveras efter varje insättning i tabellen "uttag"

**FOR** **EACH** **ROW**

**DECLARE**

v\_saldo konto**.**saldo**%TYPE;** -- Variabel för att hålla det aktuella saldot från kontot

**BEGIN**

-- Hämta det aktuella saldot för det kontonummer som är kopplat till uttaget

**SELECT** saldo **INTO** v\_saldo

**FROM** konto

**WHERE** knr **=** **:NEW.**knr**;** -- :NEW.knr refererar till det kontonummer som finns i den nyligen insatta raden i uttag

-- Uppdatera saldot genom att dra av uttagsbeloppet

**UPDATE** konto

**SET** saldo **=** v\_saldo **-** **:NEW.**belopp -- Dra av uttagsbeloppet från det aktuella saldot

**WHERE** knr **=** **:NEW.**knr**;** -- Matchar kontonumret med det som finns i den nya raden i uttag

**END;**

**/**

-- Test/verifierring --

**SELECT** **\***

**FROM** user\_triggers

**WHERE** trigger\_name **=** 'AIFER\_UTTAG'**;**

**INSERT** **INTO** uttag **(**radnr**,** pnr**,** knr**,** belopp**,** datum**)**

**VALUES** **(**radnr\_seq**.NEXTVAL,** '540126-1111'**,** '123'**,** 1000**,** **SYSDATE);**

**SELECT** **\*** **from** konto**;**

**SELECT** **\*** **from** uttag**;**

**SELECT** **\*** **from** insättning**;**

**Uppgift 13**

Skapa ytterligare en trigger. Denna gång med namnet bifer\_överföring. Triggern skall se till att det finns tillräckligt med pengar på belastningskontot vid överföring av pengar från ett konto till ett annat. **OBS!** Triggern skall anropa funktionen get\_saldo för att utföra kontrollen. Här är ett typexempel på hur vi kan återanvända vår SQL-kod!

**Svar 13**

-- uppgift 13--

**CREATE** **OR** **REPLACE** **TRIGGER** bifer\_överföring

**BEFORE** **INSERT** **ON** överföring

**FOR** **EACH** **ROW**

**DECLARE**

v\_saldo konto**.**saldo**%TYPE;**

**BEGIN**

-- Hämta saldot från belastningskontot med hjälp av funktionen get\_saldo

v\_saldo **:=** get\_saldo**(:NEW.**från\_knr**);**

-- Kontrollera om det finns tillräckligt med pengar för att genomföra överföringen

**IF** v\_saldo **<** **:NEW.**belopp **THEN**

RAISE\_APPLICATION\_ERROR**(**-20003**,** 'Otillräckligt saldo på från-kontot för överföring.'**);**

**END** **IF;**

**END;**

**/**

-- Test/verifierring --

**SELECT** **\***

**FROM** user\_triggers

**WHERE** trigger\_name **=** 'BIFER\_ÖVERFÖRING'**;**

-- Fungerar \*\*\*1000\*\*\*---

**INSERT** **INTO** överföring **(**radnr**,** pnr**,** från\_knr**,** till\_knr**,** belopp**,** datum**)**

**VALUES** **(**radnr\_seq**.NEXTVAL,** '540126-1111'**,** '123'**,** 5899**,** 1000**,** **SYSDATE);**

--fungerar inte ORA-20003: Otillräckligt saldo på från-kontot för överföring.\*\*3000\*\* ---

**INSERT** **INTO** överföring **(**radnr**,** pnr**,** från\_knr**,** till\_knr**,** belopp**,** datum**)**

**VALUES** **(**radnr\_seq**.NEXTVAL,** '540126-1111'**,** '123'**,** 5899**,** 3000**,** **SYSDATE);**

**Uppgift 14**

Skapa nu den sista triggern i denna labb. Triggern skall ha namnet aifer\_överföring. Den skall se till att saldot blir rätt på respektive konto som är inblandat i överföringen.

Triggern skall starta **efter** insert på tabellen överföring.

**Svar 14**

-- uppgift 14--

-- Triggern aktiveras efter att en ny överföring har registrerats

**CREATE** **OR** **REPLACE** **TRIGGER** aifer\_överföring

**AFTER** **INSERT** **ON** överföring

**FOR** **EACH** **ROW**

**BEGIN**

-- Minska saldot på från-kontot

**UPDATE** konto

**SET** saldo **=** saldo **-** **:NEW.**belopp

**WHERE** knr **=** **:NEW.**från\_knr**;**

-- Öka saldot på till-kontot

**UPDATE** konto

**SET** saldo **=** saldo **+** **:NEW.**belopp

**WHERE** knr **=** **:NEW.**till\_knr**;**

**END;**

**/**

-- Test/verifierring --

**SELECT** **\***

**FROM** user\_triggers

**WHERE** trigger\_name **=** 'AIFER\_ÖVERFÖRING'**;**

--Internöverföring--

**INSERT** **INTO** överföring **(**radnr**,** pnr**,** från\_knr**,** till\_knr**,** belopp**,** datum**)**

**VALUES** **(**radnr\_seq**.NEXTVAL,** '540126-1111'**,** '123'**,** 5899**,** 1000**,** **SYSDATE);**

**INSERT** **INTO** överföring **(**radnr**,** pnr**,** från\_knr**,** till\_knr**,** belopp**,** datum**)**

**VALUES** **(**radnr\_seq**.NEXTVAL,** '540126-1111'**,** '5899'**,** 123**,** 305**,** **SYSDATE);**

--Se konto--

**SELECT** **\*** **from** konto**;**

**Uppgift 15**

Skapa en procedur med namnet do\_insättning. Proceduren skapar en rad i tabellen insättning. Proceduren skall skriva ut ett meddelande om vad saldot är efter insättningen.

**Svar 15**

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_insättning **(**

p\_pnr **IN** insättning**.**pnr**%TYPE,** -- Personnummer

p\_knr **IN** insättning**.**knr**%TYPE,** -- Kontonummer

p\_belopp **IN** insättning**.**belopp**%TYPE** -- Belopp att sätta in

**)**

**AS**

v\_radnr NUMBER**(**9**);** -- Variabel för radnummer

v\_saldo konto**.**saldo**%TYPE;** -- Variabel för att lagra saldot efter insättningen

ogiltigt\_belopp **EXCEPTION;** -- Eget fel för negativt belopp

**BEGIN**

-- Kontrollera att beloppet inte är negativt

**IF** p\_belopp **<=** 0 **THEN**

**RAISE** ogiltigt\_belopp**;** -- Om beloppet är negativt eller 0, kasta ett fel

**END** **IF;**

-- Generera nästa radnummer från sekvens

**SELECT** radnr\_seq**.NEXTVAL** **INTO** v\_radnr **FROM** dual**;**

-- Lägg till insättningen

**INSERT** **INTO** insättning **(**radnr**,** pnr**,** knr**,** belopp**,** datum**)**

**VALUES** **(**v\_radnr**,** p\_pnr**,** p\_knr**,** p\_belopp**,** **SYSDATE);**

**COMMIT;** -- Bekräfta transaktionen (valfritt i procedurer, beroende på hur ni jobbar med transaktionshantering)

-- Hämta nya saldot

v\_saldo **:=** get\_saldo**(**p\_knr**);**

-- Skriv ut resultat

DBMS\_OUTPUT**.**PUT\_LINE**(**'Insättning genomförd. Aktuellt saldo: ' **||** v\_saldo **||** ' kr'**);**

**EXCEPTION**

**WHEN** ogiltigt\_belopp **THEN**

RAISE\_APPLICATION\_ERROR**(**-20020**,** 'Beloppet måste vara större än 0.'**);** -- Felmeddelande om beloppet är negativt eller 0

**WHEN** **OTHERS** **THEN**

-- Felhantering med feltext- TYPE fel personnummer, ELLER FELKONTO ---

DBMS\_OUTPUT**.**PUT\_LINE**(**'Något gick fel vid insättningen: ' **||** **SQLERRM);**

**END;**

**/**

-- test--

**SELECT** **object\_name,** status

**FROM** user\_objects

**WHERE** object\_type **=** 'PROCEDURE'

**AND** **object\_name** **=** 'DO\_INSÄTTNING'**;**

--Sättar in pengar--

**EXEC** do\_insättning**(**'540126-1111'**,** 123**,** 582**);**

--Se konto--

**SELECT** **\*** **from** konto**;**

**Uppgift 16**

Testa att proceduren do\_insättning fungerar.

Verifiera **do\_insättning** genom att:

1. Verifiera innehåll i tabellen KONTO **före insättning sker**
2. Verifiera innehåll i tabellen INSÄTTNING **före insättning sker**
3. Testa med fel personnummer
4. Testa med fel kontonummer
5. Testa med negativt belopp
6. Testa med rätt parametrar
7. Verifiera innehållet i tabellen INSÄTTNING **efter insättning skett**
8. Verifiera saldo i tabellen KONTO **efter insättning skett**

**Svar 16**

-- 1. Verifiera innehåll i tabellen KONTO före insättning sker--

**SELECT** **\*** **FROM** konto **WHERE** knr **=** 5899**;**

-- 2. Verifiera innehåll i tabellen INSÄTTNING före insättning sker --

**SELECT** **\*** **FROM** insättning **WHERE** knr **=** 5899**;**

-- 3. Testa med fel personnummer --

**EXEC** do\_insättning**(**'840101-1111'**,** 5899**,** 100**);**

-- 4. Testa med fel kontonummer --

**EXEC** do\_insättning**(**'540126-1111'**,** 1111**,** 4600**);**

-- 5. Testa med negativt belopp --

**EXEC** do\_insättning**(**'540126-1111'**,** 5899**,** -250**);**

-- 6. Testa med rätt parametrar --

**EXEC** do\_insättning**(**'540126-1111'**,** 5899**,** 1250**);**

-- 7. Verifiera innehållet i tabellen INSÄTTNING efter insättning skett --

**SELECT** **\*** **FROM** insättning **WHERE** knr **=** 5899 **ORDER** **BY** datum **DESC;**

-- 8. Verifiera saldo i tabellen KONTO efter insättning skett --

**SELECT** **\*** **FROM** konto **WHERE** knr **=** 5899**;**

**Uppgift 17**

Skapa en procedur som heter do\_uttag. Denna skapar en rad i tabelen uttag. Skapa ett eget deklarerat undantag med namnet obehörig. Detta kastas om en obehörig person försöker ta ut pengar från ett konto som denne **inte** är ägare till. Använd funktionen get\_behörighet för att utföra kontrollen. Om man är obehörig skall ett meddelande om detta skrivas ut och naturligtvis skall **inte** transaktionen då gå igenom. Om man är behörig skall proceduren skriva ut meddelande om aktuellt saldo efter uttaget.

**Svar 17**

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_uttag **(**

p\_pnr **IN** uttag**.**pnr**%TYPE,** -- Inparameter: Personnummer

p\_knr **IN** uttag**.**knr**%TYPE,** -- Inparameter: Kontonummer

p\_belopp **IN** uttag**.**belopp**%TYPE** -- Inparameter: Belopp att ta ut

**)**

**AS**

v\_radnr NUMBER**(**9**);** -- Variabel för att lagra nytt radnummer från sekvens

v\_behörighet NUMBER**(**1**);** -- Variabel som anger om personen är behörig (1) eller ej (0)

obehörig **EXCEPTION;** -- Egendefinierat undantag för obehörig åtkomst

ogiltigt\_belopp **EXCEPTION;** -- Undantag för ogiltigt (negativt eller noll) belopp

**BEGIN**

-- Kontrollera om beloppet är negativt eller 0

**IF** p\_belopp **<=** 0 **THEN**

**RAISE** ogiltigt\_belopp**;**

**END** **IF;**

-- Kontrollera behörighet med hjälp av funktionen get\_behörighet

v\_behörighet **:=** get\_behörighet**(**p\_pnr**,** p\_knr**);**

-- Om användaren är behörig:

**IF** v\_behörighet **=** 1 **THEN**

-- Hämta nästa värde från sekvens för radnummer

**SELECT** radnr\_seq**.NEXTVAL** **INTO** v\_radnr **FROM** dual**;**

-- Infoga ny rad i tabellen uttag med angivna parametrar

**INSERT** **INTO** uttag **(**radnr**,** pnr**,** knr**,** belopp**,** datum**)**

**VALUES** **(**v\_radnr**,** p\_pnr**,** p\_knr**,** p\_belopp**,** **SYSDATE);**

**ELSE**

-- Om inte behörig, kasta undantag

**RAISE** obehörig**;**

**END** **IF;**

**EXCEPTION**

-- Fångar det egendefinierade undantaget "obehörig"

**WHEN** obehörig **THEN**

RAISE\_APPLICATION\_ERROR**(**-20006**,** 'Behörighet saknas.'**);**

-- Fångar ogiltiga belopp (negativt eller 0)

**WHEN** ogiltigt\_belopp **THEN**

RAISE\_APPLICATION\_ERROR**(**-20008**,** 'Beloppet måste vara ett positivt värde större än 0.'**);**

**END;**

**/**

--test--

**SELECT** **object\_name,** status

**FROM** user\_objects

**WHERE** object\_type **=** 'PROCEDURE'

**AND** **object\_name** **=** 'DO\_UTTAG'**;**

--fungerar--

**EXEC** do\_uttag**(**'540126-1111'**,** 5899**,** 582**);**

-- ej fungerar fel pnr, ORA-20006: Behörighet saknas.--

**EXEC** do\_uttag**(**'540126-1110'**,** 123**,** 582**);**

-- ej fungerar fel knr, ORA-20006: Behörighet saknas.--

**EXEC** do\_uttag**(**'540126-1111'**,** 1111**,** 582**);**

**SELECT** **\*** **FROM** UTTAG**;**

**Uppgift 18**

Testa att proceduren do\_uttag fungerar. **Nu** är det även tillfälle att se om triggrarna sköter sitt jobb. Kontrollera om det går att ta ut mer än vad som finns på kontot. Kontrollera att saldot uppdateras som det ska.

Verifiera **do\_uttag** genom att:

1. Verifiera innehåll i tabellen KONTO **före uttag sker**
2. Verifiera innehåll i tabellen UTTAG **före uttag sker**
3. Testa med fel personnummer
4. Testa med fel kontonummer
5. Testa med för stort belopp
6. Testa med negativt belopp
7. Testa med rätt parametrar
8. Verifiera innehållet i tabellen UTTAG **efter uttag skett**
9. Verifiera innehållet i tabellen KONTO **efter uttag skett**

**Svar 18**

Verifiera do\_uttag genom att**:**

-- 1. Verifiera innehåll i tabellen KONTO före uttag sker --

**SELECT** **\*** **FROM** konto **WHERE** knr **=** 123**;**

-- 2. Verifiera innehåll i tabellen UTTAG före uttag sker--

**SELECT** **\*** **FROM** uttag **WHERE** knr **=** 123**;**

-- 3. Testa med fel personnummer \*\* ORA-20006: Behörighet saknas.\*\*---

**EXEC** do\_uttag**(**'19840101-1111'**,** 123**,** 500**);**

-- 4. Testa med fel kontonummer--

**EXEC** do\_uttag**(**'540126-1111'**,** 999**,** 690**);**

-- 5. Testa med för stort belopp, \*\* ORA-20001: Du har inte tillräckligt med pengar på kontot för detta uttag.\*\*---

**EXEC** do\_uttag**(**'540126-1111'**,** 123**,** 150000**);**

-- 6. Testa med negativt belopp \*\* ORA-20008: Beloppet måste vara ett positivt värde större än 0.\*\*--

**EXEC** do\_uttag**(**'540126-1111'**,** 123**,** -300**);**

-- 7. Testa med rätt parametrar \*\* PL/SQL procedure successfully completed.\*\*--

**EXEC** do\_uttag**(**'540126-1111'**,** 123**,** 50**);**

-- 8. Verifiera innehållet i tabellen UTTAG efter uttag skett--

**SELECT** **\*** **FROM** uttag **WHERE** pnr **=** '540126-1111' **AND** knr **=** 123**;**

-- 9. Verifiera innehållet i tabellen KONTO efter uttag skett--

**SELECT** saldo **FROM** konto **WHERE** knr **=** 123**;**

**Uppgift 19**

Skapa labbens sista objekt som är en procedur med namnet do\_överföring. Denna skapar en ny rad i tabellen överföring. Skapa även här ett eget deklarerat undantag med namnet obehörig. Om man är obehörig skall ett meddelande om detta skrivas ut och naturligtvis skall **inte** transaktionen då gå igenom. Om man är behörig skall proceduren skriva ut meddelande om aktuellt saldo **på bägge kontonumren** som är inblandade i överföringen.

**Svar 19**

**CREATE** **OR** **REPLACE** **PROCEDURE** do\_överföring **(**

p\_pnr **IN** överföring**.**pnr**%TYPE,**

p\_från\_knr **IN** överföring**.**från\_knr**%TYPE,**

p\_till\_knr **IN** överföring**.**till\_knr**%TYPE,**

p\_belopp **IN** överföring**.**belopp**%TYPE**

**)**

**AS**

v\_saldo\_fr konto**.**saldo**%TYPE;**

v\_saldo\_t konto**.**saldo**%TYPE;**

v\_radnr NUMBER**(**9**);**

v\_behörighet NUMBER**(**1**);**

v\_tmp konto**.**knr**%TYPE;**

obehörig **EXCEPTION;**

ogiltigt\_belopp **EXCEPTION;**

**BEGIN**

-- Kontroll: Beloppet måste vara positivt

**IF** p\_belopp **<=** 0 **THEN**

**RAISE** ogiltigt\_belopp**;**

**END** **IF;**

-- Kontroll: Från-konto måste finnas (annars NO\_DATA\_FOUND kastas)

**SELECT** knr **INTO** v\_tmp **FROM** konto **WHERE** knr **=** p\_från\_knr**;**

-- Kontroll: Till-konto måste finnas (annars NO\_DATA\_FOUND kastas)

**SELECT** knr **INTO** v\_tmp **FROM** konto **WHERE** knr **=** p\_till\_knr**;**

-- Kontrollera behörighet

v\_behörighet **:=** get\_behörighet**(**p\_pnr**,** p\_från\_knr**);**

**IF** v\_behörighet **=** 1 **THEN**

**SELECT** radnr\_seq**.NEXTVAL** **INTO** v\_radnr **FROM** dual**;**

**INSERT** **INTO** överföring **(**radnr**,** pnr**,** från\_knr**,** till\_knr**,** belopp**,** datum**)**

**VALUES** **(**v\_radnr**,** p\_pnr**,** p\_från\_knr**,** p\_till\_knr**,** p\_belopp**,** **SYSDATE);**

**COMMIT;**

v\_saldo\_fr **:=** get\_saldo**(**p\_från\_knr**);**

v\_saldo\_t **:=** get\_saldo**(**p\_till\_knr**);**

DBMS\_OUTPUT**.**PUT\_LINE**(**'Saldo efter genomförd överföring:'**);**

DBMS\_OUTPUT**.**PUT\_LINE**(**'Ditt konto: ' **||** v\_saldo\_fr **||** ' kr'**);**

DBMS\_OUTPUT**.**PUT\_LINE**(**'Mottagande konto: ' **||** v\_saldo\_t **||** ' kr'**);**

**ELSE**

**RAISE** obehörig**;**

**END** **IF;**

-- Felhantering

**EXCEPTION**

**WHEN** obehörig **THEN**

RAISE\_APPLICATION\_ERROR**(**-20007**,** 'Behörighet saknas.'**);**

**WHEN** ogiltigt\_belopp **THEN**

RAISE\_APPLICATION\_ERROR**(**-20009**,** 'Beloppet måste vara större än 0.'**);**

**WHEN** NO\_DATA\_FOUND **THEN**

RAISE\_APPLICATION\_ERROR**(**-20010**,** 'Det kontonummer du försöker använda finns inte.'**);**

**END;**

**/**

-- test---

**SELECT** **object\_name,** status

**FROM** user\_objects

**WHERE** object\_type **=** 'PROCEDURE'

**AND** **object\_name** **=** 'DO\_ÖVERFÖRING'**;**

**Uppgift 20**

Testa att flytta pengar mellan konton med proceduren do\_överföring. Testa även här att triggrarna fungerar.

Verifiera **do\_överföring** genom att:

1. Verifiera innehåll i tabellen KONTO **före överföring sker**
2. Verifiera innehåll i tabellen ÖVERFÖRING **före överföring sker**
3. Testa med fel personnummer
4. Testa med fel **från**-kontonummer
5. Testa med fel **till**-kontonummer
6. Testa med för stort belopp
7. Testa med negativt belopp
8. Testa med rätt parametrar
9. Verifiera innehållet i tabellen ÖVERFÖRING **efter överföring skett**
10. Verifiera innehållet i tabellen KONTO **efter överföring skett**

**Svar 20**

-- 1. Verifiera innehåll i tabellen KONTO före överföring sker--

**SELECT** **\*** **FROM** konto**;**

-- 2. Verifiera innehåll i tabellen ÖVERFÖRING före överföring sker--

**SELECT** **\*** **FROM** överföring**;**

-- 3. Testa med fel personnummer \*\* ORA-20007: Behörighet saknas.\*\*-

**EXEC** do\_överföring**(**'540126-1110'**,** 123**,** 5899**,** 100**);**

-- 4. Testa med fel från-kontonummer \* \*ORA-20007: Behörighet saknas.\*\*---

**EXEC** do\_överföring**(**'540126-1111'**,** 124**,** 5899**,** 100**);**

-- 5. Testa med fel till-kontonummer,\*\* ORA-20010: Det kontonummer du försöker använda finns inte.\*\*--

**EXEC** do\_överföring**(**'540126-1111'**,** 123**,** 899**,** 100**);**

-- 6. Testa med för stort belopp, \*\* ORA-20003: Otillräckligt saldo på från-kontot för överföring.\*\* --

**EXEC** do\_överföring**(**'540126-1111'**,** 123**,** 5899**,** 150000**);**

-- 7. Testa med negativt belopp, \*\* ORA-20009: Beloppet måste vara större än 0.\*\* ---

**EXEC** do\_överföring**(**'540126-1111'**,** 123**,** 5899**,** -120**);**

--8. Testa med rätt parametrar \*\*PL/SQL procedure successfully completed.\*\*--

**EXEC** do\_överföring**(**'540126-1111'**,** 123**,** 5899**,** 100**);**

-- 9. Verifiera innehållet i tabellen ÖVERFÖRING efter överföring skett --

**SELECT** **\*** **FROM** överföring **WHERE** från\_knr **=** 123 **AND** till\_knr **=** 5899**;**

-- 10. Verifiera innehållet i tabellen KONTO efter överföring skett -

**SELECT** **\*** **FROM** konto **WHERE** knr **IN** **(**123**,** 5899 **);**

**Uppgift 21 – Egna reflektioner!**

Skriv något om labbserien! Vad har du lärt dig? För svårt? För lätt? För mycket? För lite?

**Svar 21**

Vi tycker att labb fem var mycket givande och praktisk. Det tog en hel vecka, men varje sekund var värd det. Vi kunde procedurer, triggers och sekvenser i teorin, men i praktiken hade vi svårt att förstå hur de fungerade och varför de är viktiga. Denna labb gav oss djupare förståelse för dem. Labb fems databodel gav oss djupare insikt i olika datamodelleringstekniker och datanormalisering som enligt oss är en viktig grund för vår framtid i yrket i systemutveckling.

Tester och verifieringar var också mycket viktiga, och vi följde Pärs instruktion att skriva vår kod i delar och testa innan vi skriver ett stort kodblock. Därefter kunde vi debugga och använda den. En annan sak som var givande var att vi kunde återanvända vår kod, vilket effektiviserade grupparbetet.

Vi hade lite svårt med filminspelningen, men kraften i grupparbete löste det problemet också. Det som var mest roligt var att se och förstå ett banksystem på djupet, och att följa hela processen från uppgift 1 till 20 för att utveckla ett banksystem som vi kan ha nytta av i vårt framtida yrke. En app byggs inte på en dag, utan genom stegvisa uppdateringar och tillägg av nya funktioner.

Vi tycker att kursen och denna labb har varit mycket givande.

**På nästa sida hittar du teorifrågor som ska besvaras och redovisas i er film. Ni ska förklara och motivera varför ni väljer att svara som ni gör.**

**Teoriprov – här ska du motivera och förklara varför du har svarat på ett visst sätt.**

För betyget G måste alla uppgifterna vara rätt besvarade och motiverade.

### Uppgift 1) 1p

Utvecklarna Ahmed och Lotta har stött på en del problem. Ahmed arbetar som DBA och håller på med att fylla på en tabell som heter artikel med data. Tabellen ägs av schemat webshop. Lotta ropar till Ahmed att hon inte ser någon data om artiklarna i tabellen artikel. Lotta är inloggad mot samma schema (webshop) som Ahmed.

Ahmed sitter vid en dator och är inloggad mot databasen ORAWEB som finns på **oracle18.abc.se:1599 (port 1599)**. Han har precis exekverat följande SQL-kod:

INSERT INTO webshop.artikel(artnr,artnamn,pris)

VALUES(artnr.NEXTVAL,'spade',475);

När **Ahmed** skriver **SELECT \* FROM webshop.artikel;** erhåller han följande output:

artnr artnamn pris

--------------------------------------

1 spade 475

När Lotta kör samma selectsats från sin dator (en annan dator) mot servern **oracle18.abc.se:1599 (port 1599)** så får hon följande output:

**SELECT \* FROM webshop.artikel;**

no rows selected

**Din uppgift blir att lösa deras problem genom att besvara:**

1. **Varför ser inte Lotta samma data som Ahmed?**

Eftersom Ahmed inte har kört COMMIT efter att han gjorde sin INSERT. I en Oracle-databas blir data synlig för andra användare först när transaktionen har commitats. Det betyder att Ahmed fortfarande ser datan eftersom han är kvar i samma transaktion, men Lotta kan inte se den eftersom ändringen ännu inte är sparad permanent i databasen.

1. **Vad behöver Ahmed göra för att Lotta ska se samma data?**

Ahmed behöver köra kommandot COMMIT. Då sparas datan permanent i databasen och blir tillgänglig för alla användare, inklusive Lotta, som är inloggad mot samma databas.

### Uppgift 2) 1p

Nedanstående objektmodell ska transformeras till en informationsmodell. Detta innebär att alla M:M förhållanden måste bort. Modellen är under utveckling för ett byggföretag Kvasts Bygg i Gagnef AB. Modellen beskriver olika byggprojekt. Ett projekt kan vara att bygga ett uterum, en altan eller att bygga ett garage. Företaget har många olika projekt. Byggprojekten behöver olika arbetare för att kunna slutföras. Det kan behövas snickare, elektriker, plåtslagare eller rörmokare för att utföra alla delar i projektet.

Som vi ser på bilden kan "Ett projekt kan inkludera flera arbetare, en arbetare kan jobba i flera projekt". **Din uppgift blir att få bort M:M förhållandet genom att rita en ny informationsmodell.**

En bild som visar text, linje, diagram, kvitto

Automatiskt genererad beskrivning

**Min lösning i form av en ny modell:**

En bild som visar text, linje, nummer, skärmbild

AI-genererat innehåll kan vara felaktigt.

För att omvandla objektmodellen till en korrekt informationsmodell har vi tagit bort M:M-relationen mellan Projekt och Arbetare. Vi har infört en mellanliggande tabell ProjektMedarbetare som kopplar ihop dessa två entiteter med hjälp av främmande nycklar. Varje rad i ProjektMedarbetare representerar en arbetare som deltar i ett visst projekt.

### Uppgift 3) 1p

**Om vi ska lagra ett orderdatum i en databastabell, vad ska det vara för datatyp på kolumnen?**

1. date
2. number
3. varchar2

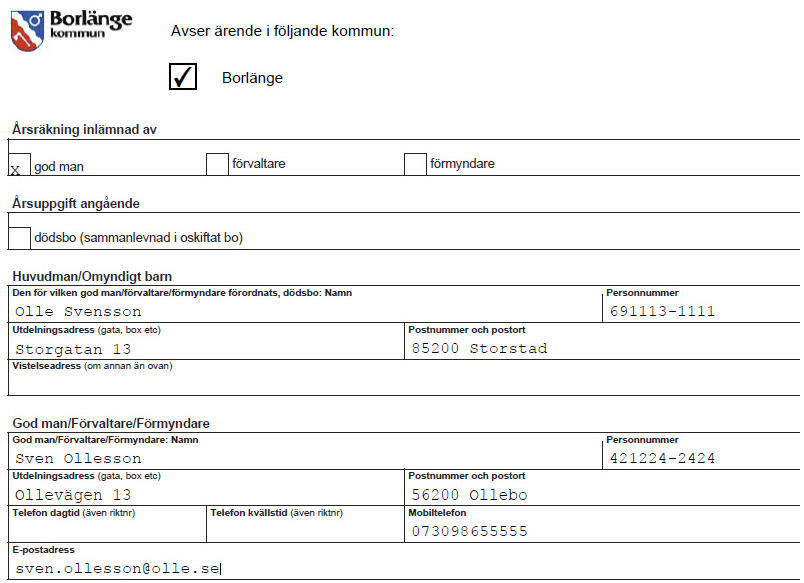
**Markera och motivera ditt svar.**

**Motivering:**  
Vi väljer DATE eftersom orderdatum är ett datumvärde, och därför ska kolumnen ha datatypen DATE. Med datatypen DATE kan man lagra både datum och tid på ett korrekt sätt. Det möjliggör att man kan jämföra, sortera och beräkna datum på ett smidigt sätt. Exempel på användning är: att hämta dagens datum (SYSDATE), beräkna antalet månader mellan två datum, eller se hur många dagar som har gått mellan två olika beställningar.

Datatyperna NUMBER och VARCHAR2 är inte lämpliga för att lagra datum, eftersom de inte kan hantera datumlogik. De tillåter inte datumfunktioner, vilket gör det svårt att filtrera, jämföra eller sortera datum korrekt.

### Uppgift 4) 1p

Studera nedanstående bild



**Vad representerar den ifyllda blanketten?**

1. Data
2. Metadata
3. Information

**Markera och motivera ditt svar.**

Blanketten räknas som information eftersom den innehåller färdig, strukturerad data med tydlig betydelse till exempel vem ärendet gäller och vad det handlar om. Det är alltså mer än bara data som kan bestå av enstaka siffror eller ord utan sammanhang och mer än metadata som enbart beskriver vad olika delar betyder som exempelvis "namn" eller "personnummer". Om blanketten hade varit tom och endast innehållit rubriker utan ifylld data skulle det snarare ha klassats som metadata. När den däremot är ifylld innehåller den både data och metadata vilket tillsammans utgör information.

### Uppgift 5) 1p

**PK, Primary key eller primärnyckel**

I det paper som Owen Eriksson skrev: *To Denominate and Characterise in the Context of Information Systems*, kommer Owen fram till några viktiga slutsatser. Är nedanstående sant eller falskt enligt dessa slutsatser?

**Påstående:** **Artikelnummer som används som PK bör innehålla en beskrivning av de objekt som de identifierar.**

1. sant
2. falskt

**Markera och motivera ditt svar.**

Enligt Owen Eriksson bör en primärnyckel som till exempel ett artikelnummer endast användas för att identifiera ett objekt inte för att beskriva det. Om man bygger in beskrivande information i nyckeln riskerar man att:

* Numren tar slut snabbare
* Identifieraren måste ändras vid organisationsförändringar
* Systemet blir svårare att underhålla

Därför ska en primärnyckel vara neutral, stabil och unik, inte bära på betydelse bara på rätt objekt.

### Uppgift 6) 1p

Om du studerar nedanstående bild och tabellen ränteändring. Är följande databasdesign en implementation av principen: Börje Langefors elementarmeddelande (e-message)?

### En bild som visar text, skärmbild, linje, diagram Automatiskt genererad beskrivning

1. ja
2. nej

**Markera och motivera ditt svar.**

Vi anser att svaret är ja. Tabellen för ränteändring visar tydligt vad ett elementarmeddelande är, precis som Börje Langefors beskriver det. Varje rad i tabellen representerar en specifik och avgränsad händelse i verkligheten, att räntan för en viss kontotyp har ändrats vid ett visst datum. Det handlar inte endast om generella data, utan om något mer konkret som faktiskt har hänt. Datamodellen fångar en tydlig informationshändelse, vilket är själva grunden i principen om e-meddelanden.

**Lämna in hela dokumentet med en länk till en film eller filmen som en mp4-fil. Gör inlämningen i kursrummet.**

**https://www.youtube.com/watch?v=gq2hDiwQKgg&t=2913s**