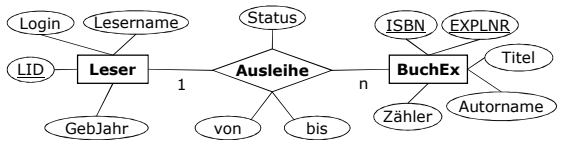

1

Aufgabe 1a

```
CREATE TABLE Leser (
    LID         INT             PRIMARY KEY,
    Login       VARCHAR(12)     UNIQUE,
    Lesername   VARCHAR(40)     NOT NULL,
    GebJahr     SMALLINT        CHECK
                  (GebJahr BETWEEN 1901
                   AND CURRENT_YEAR-1)
)

CREATE TABLE BuchEx (
    ISBN        CHAR(10),
    EXPLNR      SMALLINT,
    Titel       VARCHAR(90) NOT NULL,
    Autorname   VARCHAR(60),
    Zähler      INT NOT NULL DEFAULT 0,
    PRIMARY KEY (ISBN, EXPLNR)
)

CREATE TABLE Ausleihe (
    ISBN        CHAR(10)        NOT NULL,
    EXPLNR      SMALLINT        NOT NULL,
    von         DATE            NOT NULL,
    bis         DATE            NOT NULL,
    LeserID     INT             NOT NULL
      REFERENCES Leser(LID) ON DELETE OF Leser NO ACTION,
    Status      VARCHAR(40) CHECK (Status IN ('ausgeliehen', 'verlängert', 'verloren')),
    PRIMARY KEY (LeserID, ISBN, EXPLNR),
    FOREIGN KEY (ISBN, EXPLNR)
      REFERENCES BuchEx(ISBN, EXPLNR) ON DELETE OF BuchEx CASCADE,
    CHECK ( von < bis ),
    CHECK ( DAYS(bis) – DAYS(von) < 30 ),
    CHECK ( NOT EXISTS (
                SELECT Lesername FROM Leser, BuchEx
                WHERE LID=LeserID AND Lesername = Autorname ) )
)
```

2

Aufgaben 1b+c: Integritätsbedingungen

1. Eindeutiges Login — UNIQUE
   Satztyp

2. 1900<Geb-Jahr<aktuelles Jahr — CHECK (GebJahr BETWEEN 1901 AND CURRENT_YEAR-1)
   Attribut

3. Ausleih-Dauer nicht negativ — CHECK (bis – von >= 0)
   Satzausprägung

   Ausleih-Dauer <= 30 Tage — CHECK (DAYS(bis) – DAYS(von) < 30)
   Satzausprägung

4. Leser-Name <> Autor-Name — CHECK (NOT EXISTS (SELECT Name FROM Leser L, Buch B WHERE LID=L.LeserID AND Lesername = B.Autorname))
   mehrere Satztypen

5. *Status nur 3 definierte Werte* — CHECK (Status IN ('ausgeliehen', 'verlängert', 'verloren'))
   Attribut

3

Aufgabe 2a: Sicht-Konzept

1 **CREATE VIEW** JungLeser(LID, Login) **AS**
SELECT LID, Login
FROM Leser
WHERE CURRENT_YEAR – GebJahr < 20

Akualisierbar? Nein.
UPDATE auf LID und Login denkbar, jedoch kein INSERT wg. NOT NULL u. GebJahr BETWEEN …

2 **CREATE VIEW** AusleihBücher(ISBN, Titel) **AS**
SELECT DISTINCT B.ISBN, Titel
FROM BuchEx B **NATURAL JOIN** Ausleihe A
WHERE Status IN ('ausgeliehen', 'verlängert')

3 **CREATE VIEW** AusleihAnzahl(ISBN, Anzahl) **AS**
SELECT *BI.ISBN*, **COALESCE**(*Anzahl*,0)
FROM (**SELECT DISTINCT** *ISBN* **FROM** *BuchEx*) **AS** *BI*
NATURAL LEFT OUTER JOIN
(**SELECT** *ISBN*, **COUNT**(*) AS *Anzahl*
FROM *Ausleihe*
GROUP BY *ISBN*) **AS** *ANZ*

4

Aufgabe 2a+b: Sicht-Konzept, Rechte

4 **CREATE VIEW** AusleihData(*LID*, *Lesername*, *Titel*, *Autorname*) **AS**
SELECT *LID*, *Lesername*, *Titel*, *Autorname*
FROM (*Leser L* **JOIN** *Ausleihe A* **ON** *L.LID=A.LeserID)*
NATURAL JOIN BuchEx

5 **CREATE VIEW** UnausgeliehenData (ISBN, Autorname, ZählerInsg) **AS**
SELECT ISBN, Autorname, SUM(Zähler)
FROM BuchEx
WHERE ISBN **NOT IN** (SELECT ISBN FROM Ausleihe)
GROUP BY ISBN, Autorname

Sichten akualisierbar? Nein:
- Primärschlüssel unvollständig
- Aggregation
- Join über mehrere Tabellen

GRANT INSERT **ON** Leser **TO** Bibliothekar

REVOKE UPDATE, DELETE **ON** JungLeser **FROM** Bilbo

5

Aufgabe 2d: Sicht-Konzept

1 **SELECT** **COUNT**(*)
FROM JungLeser

2 **SELECT** ISBN, Titel
FROM AusleihAnzahl
WHERE Anzahl = (**SELECT MAX**(Anzahl) **FROM** AusleihAnzahl)

3 **SELECT** LID
FROM AusleihData
GROUP BY LID, Autorname
HAVING **COUNT**(*) > 2

GROUP BY **LID**
HAVING COUNT(*) > 2
→ Wer hat mehr als zwei Bücher ausgeliehen?

GROUP BY **LID, Autorname**
HAVING COUNT(*) > 2
→ Wer hat mehr als zwei Bücher *desselben Autors* ausgeliehen?

4 **SELECT** Autorname
FROM UnausgeliehenData
WHERE Zähler = 0

6

Aufgabe 2e: Sicht-Konzept

Vorteile:

- Benutzerfreundlichkeit
  - Für best. Benutzer bestimmte Daten sichtbar machen
  - Ausschnitte, Vorberechnung, Aggregatfunktionen
- Datenunabhängigkeit (verbesserte Schema-Evolution)
- Datenschutz/Zugriffskontrolle

Nachteile

- eingeschränkte Änderbarkeit der Daten
- erhöhter Speicherbedarf und Aktualisierungsaufwand bei materialisierten Sichten (dafür schnellerer Zugriff)
- Einschränkungen bei Definition der Sichten (Schachtelung von Aggregatfunktionen, Gruppenbildung)

Aufgabe 3: Trigger

1 max. 5 Bücher ausleihen

```
CREATE TRIGGER max5buecher
AFTER INSERT ON Ausleihe
REFERENCING NEW AS nrow NEW TABLE AS ntab
FOR EACH ROW
WHEN ( 5 < (   SELECT COUNT(*)
               FROM ntab
               WHERE LeserID=nrow.LeserID) )
ROLLBACK
```


Aufgabe 3: Trigger

2 Zähler in BuchEx bei Buchrückgabe hochzählen

```
CREATE TRIGGER hochzaehlen
AFTER DELETE ON Ausleihe
REFERENCING OLD AS orow
FOR EACH ROW
UPDATE BuchEx SET zaehler=zaehler+1
WHERE ISBN=orow.ISBN AND EXPLNR=orow.EXPLNR
```

Generell bei Trigger zu beachten (1-17):

- nur änderungsgesteuerter Aufruf
- keine verzögerte Auswertung (vgl. DEFERRED bei CHECK möglich)
- gegenseitige Aktivierung von Triggern
- dynamische Fehler schwer zu lokalisieren

Aufgabe 3c: Trigger

Assertion: tabellenunabhängige (satztypübergreifende) Integritätsbedingung basierend auf Bedingung mit Suchausdruck [Aussage, Zusicherung]

Realisierbar als Assertion ist nur Trigger 1.
Andere Bedingungen sind nicht als Bedingung über einen Tabellenzustand formulierbar.

```
CREATE ASSERTION AMax5Buecher
CHECK ( NOT EXISTS
        ( SELECT LeserID
          FROM   Ausleihe
          GROUP BY LeserID
          HAVING COUNT(*) > 5 ) )
```


Aufgabe 3d: Trigger

3 Status-Änderung prüfen (dynamische Integritätsbedingung)

```
CREATE TRIGGER uebergaenge
AFTER UPDATE OF Status ON Ausleihe
REFERENCING OLD AS orow NEW AS nrow
FOR EACH ROW
WHEN (  ( orow.status = 'verloren' AND nrow.status = 'ausgeliehen' )
     OR   ( orow.status = 'verloren' AND nrow.status = 'verlängert' ) )
     ROLLBACK
```

4

```
CREATE TRIGGER buchVerloren
AFTER UPDATE OF Status ON Ausleihe
REFERENCING OLD AS orow NEW AS nrow
FOR EACH ROW WHEN ( nrow.status = 'verloren' )
  DELETE FROM BuchEx WHERE
  ISBN=nrow.ISBN AND EXPLNR=nrow.EXPLNR
```


Aufgabe 3e: Trigger

3 Status-Änderung prüfen (dynamische Integritätsbedingung)

```
CREATE TABLE stati (
  alt      VARCHAR(40),
  neu      VARCHAR(40),
  PRIMARY KEY (alt, neu)
)
```

stati	
alt	**neu**
ausgeliehen	verlängert
ausgeliehen	verloren
verlängert	verloren

```
CREATE TRIGGER uebergaenge
AFTER UPDATE OF Status ON Ausleihe
REFERENCING OLD AS orow NEW AS nrow
FOR EACH ROW
 WHEN NOT EXISTS
( SELECT alt, neu FROM stati
  WHERE alt = orow.status AND neu = nrow.status )
ROLLBACK
```