4 Schema Definition with SQL / DDL (II)

4.3 SQL/DDL - Constraints

Different kinds of integrity constraints

Important

- on attributes, column constraints
  " 0 <= accountValue ", "must not be NULL"
- cardinalities
- "semantic" constraints ("business rules")
  e.g. "The percentage of movies not older than one year must be 25% or more"
  " After 100 loans a video tape (not DVD) has to be discarded"

– Column constraint
Specify constraint as part of column definition

– Table constraint More than one row involved, specify constraint after the column definitions

Constraints

Constraints may be violated when DB is changed (update, insert, delete)

⇨ Exception

```
ORA-02291: integrity constraint (SYS_C0067174)
  violated - parent key not found
```

Constraint name (optional)

```
CONSTRAINT <name> <def>
```

Advantage: error message shows violated constraint in a readable way

```
ORA-02291: integrity constraint
  (FKMovieOnTape.SYS_C0067174)  violated - parent
  key not found
```


4.3.1 Attribute and simple table constraints

- PRIMARY KEY
  – Only once per table
  – Not necessary, but omission is bad style
  – May be column constraint (single attribute) or table constraint

```
CREATE TABLE Tape(
  t_id          INTEGER PRIMARY KEY,
  m_id          INTEGER NOT NULL,
  format        CHAR(5),
  aquired             DATE )

CREATE TABLE People(
  name          VARCHAR(20),
  birth_date    DATE, ...,
  CONSTRAINT pk PRIMARY KEY (name, birth_date)
)
```


Not Null constraint

- NOT NULL

  Simplest constraint on attribute values, column constraint

```
CREATE TABLE ExtraCharge (
format   CHAR(5) NOT NULL,
extraCh  DECIMAL(3,2))
```

- Default values

```
<attributeName> <attributeType> DEFAULT <value>
```

- UNIQUE
  - Column contains only unique values
  - Left over from SQL-89 (no primary key constraint)
  - Should be used for candidate keys
  - Column constraint or table constraint

Unique semantics

Strange UNIQUE semantics:
a column with more than one NULL value does not violate the UNIQUE constraint

```
CREATE TABLE uniqueTest (
 x  INTEGER UNIQUE,
 y  VARCHAR(2)
 );
INSERT INTO uniqueTest
  VALUES (NULL,'a');
INSERT INTO uniqueTest
  VALUES (NULL,'a');

SELECT * FROM uniqueTest;

         X Y
---------- --
         1 a
           a
           a
```

Simple example

Therefore UNIQUE without NOT NULL does not really make sense.

SQL/DML statements, will be discussed later

UNIQUE + NOT NULL =PRIMARY KEY

CHECK clause

Predicates which must hold for each row

Enumeration:

```
CHECK (VALUES IN (' comedy', 'entertainment',
 'suspense', 'drama', 'action', 'sci-fi')).
```

Interval restriction:

```
CHECK (extraCh >= 0),
CHECK (extraCh < 10)
```

equivalent to

```
CHECK (extraCh >= 0 AND extraCh < 10)
```

Multicolumn constraint:

```
CREATE TABLE Accounts (
  ... amount DECIMAL(9,2),
      credit DECIMAL(7,2),...,
 CONSTRAINT accountIsPos
  CHECK amount + credit > 0 )
```


Use of table constraints

General constraint syntax

for column (except **NOT NULL**) and table constraints

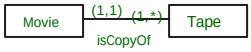
```
CREATE TABLE <tab> (< listOfColumnSpecs>
   [[CONSTRAINT <constraintName>]
  <constraint on columns or table>]0..n
)
```

Constraint may be **UNIQUE / PRIMARY KEY** or **CHECK** clause or **REFERENCES**

```
CREATE TABLE Customer(
  ...
  name VARCHAR(40),
  zipcode VARCHAR(6),
  ... ,
  CONSTRAINT customerUniqueness
    UNIQUE (name, zipcode, street, no)
  )
```


4.3.2 Enforcing cardinality constraints

"Hard code" cardinality constraints in the database schema




```
Tape(id, acqDate,format, m_Id )
Movie(m_id, title, category, price_day,
      director, year)
```

(1,..) ⇨ NOT NULL constraint sufficient ?

```
CREATE TABLE Tape (
 t_id      INTEGER PRIMARY KEY,
 aquired DATE,
 format  CHAR(5) NOT NULL,
 mId INTEGER NOT NULL)
```

Sufficient to enforce constraint only if *foreign key*

Foreign keys

- Constraint **NOT (m_ID IS NULL)**
  always true in example above for every row
  ... but the value of **m_Id** may not be among the **mId** values in table **Movie**
- **m_ID** is a foreign key in **Tape** table
  - Should exist, otherwise "missing entity" ~"dangling pointer"
  - "Find title of movie on tape with id=1175" would fail

```
CREATE TABLE Tape (
 ...,
 movieId INTEGER NOT NULL,
 CONSTRAINT movieExists
  FOREIGN KEY (movieID) REFERENCES Movie(mId)
 );
```


Foreign keys

Important concept

Let R be a relation with key k

fk ⊂ Σ(S) of relation S is foreign key if for all tuples s∈ S holds:

1. The attributes of s.fk contain only NULL values or only values ≠ NULL
2. If all attibutes of s.fk have value != NULL
   there exists a tuple r∈ R and the value of s.fk is the key of s

Preserving referential integrity

Referential integrity means:
Foreign key constraint holds

important

DBS has to prevent violations

- prevent execution of SQL statements which violate referential Integrity
- INSERT, UPDATE, DELETE statements modify state
- Update and Delete of a table may cause a violation
- Violations cause an exception
- Avoid violations by appropriate actions specified in a REFERENCES clause of the referencing table

Preserving referential integrity

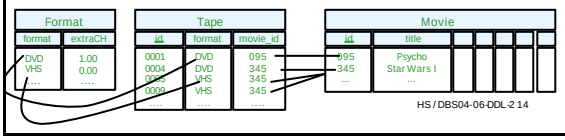
- Actions on referenced tables:
  - **ON DELETE CASCADE**
  - – delete all referenced tuples
    e.g. if a movie is deleted, all tapes with this movie are deleted from the database
  - **ON DELETE SET NULL**
  - **ON DELETE DEFAULT**
  - **ON UPDATE CASCADE**
  - – update key in referencing table
    e.g. movie gets a new key, update value used as foreign key in the same way
  - **ON UPDATE SET NULL**
  - **ON UPDATE SET DEFAULT**

Example: Referential Integrity in SQL/DDL

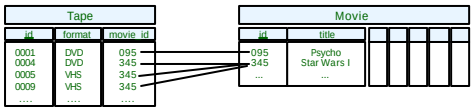
```
CREATE TABLE Tape(
  t_id       INTEGER PRIMARY KEY,
  format     VARCHAR(5) NOT NULL,
  m_id       INTEGER NOT NULL,
  CONSTRAINT tapeNotEmpty
  FOREIGN KEY (m_id) REFERENCES Movie(m_id)
    ON DELETE CASCADE,
  CONSTRAINT formatCheck
  FOREIGN KEY (format) REFERENCES Format(name)
    ON DELETE SET DEFAULT);
```

NOT NULL and FOREIGN KEY together guarantee constraint min=1

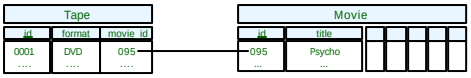
ON condition preserves constraint



Example (cont.)

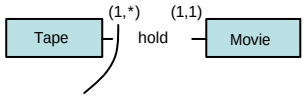



```
Delete from Movie
Where id = 345;
```



SQL / DDL: Integrity constraints

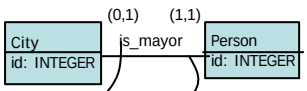
- Cardinality constraints



– NOT NULL ensures min = 1

```
CREATE TABLE Tape(
  id       INTEGER PRIMARY KEY,
  format   CHAR(10) NOT NULL,
  m_id INTEGER NOT NULL,
  CONSTRAINT tapeNotEmpty
  FOREIGN KEY (m_id) REFERENCES Movie(m_id),
  CONSTRAINT formatCheck
  FOREIGN KEY (format) REFERENCES Format(name));
```


Cardinality constraints



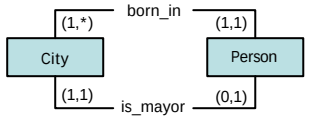
– NOT NULL ensures min = 1

– UNIQUE ensures max= 1

```
CREATE TABLE City(
  id       INTEGER PRIMARY KEY,
  mayor    INTEGER UNIQUE NOT NULL,
CONSTRAINT mayorFK
  FOREIGN KEY (mayor) REFERENCES Person(id));
```


Circular relationships

Example



- Table must be created in order to be referenced
- How to define “circular” constraints?
  - – Specify constraints after table definition

```
ALTER TABLE Person
  ADD CONSTRAINT birthPlaceReference
  FOREIGN KEY (birthplace)
  REFERENCES city(id);
ALTER TABLE Person
  MODIFY COLUMN( birthplace NOT NULL);
```

4.3.3 Deferred constraints

The Chicken-Egg problem

```
CREATE TABLE chicken(cID INT PRIMARY KEY,
                     eID INT);
CREATE TABLE egg(eID INT PRIMARY KEY,
                 cID INT);

    -- Example by J. Widom et al.
ALTER TABLE chicken
    ADD CONSTRAINT chickenREFegg
    FOREIGN KEY (eID) REFERENCES egg(eID);

ALTER TABLE egg
    ADD CONSTRAINT eggREFchicken
    FOREIGN KEY (cID) REFERENCES chicken(cID)  ;
```

What happens if an egg / chicken is inserted?

Deferred constraints

Insertion violates foreign key constraint

```
INSERT INTO chicken VALUES(1, 2);
ORA-02291: integrity constraint
  (chickenREFegg.SYS_C0067174)  violated - parent
  key not found
INSERT INTO egg VALUES(2, 1);
ORA-02291: integrity constraint
  (eggREFchicken.SYS_C0067174)  violated - parent
  key not found
```

Defer constraint checking

```
ALTER TABLE chicken
   ADD CONSTRAINT chickenREFegg
   FOREIGN KEY (eID) REFERENCES egg(eID)
   INITIALLY DEFERRED DEFERRABLE;
```


Deferred constraints and transactions

Deferred constraints checked at the end of a transaction
Transaction: unit of work consisting of one or more operations on the DB
"**COMMIT**" closes a transaction

```
INSERT INTO chicken VALUES(1, 2);
-- constraint not checked here
INSERT INTO egg VALUES(2, 1);
COMMIT;  -- but here
```

Variants

```
INITIALLY DEFERRED DEFERRABLE
INITIALLY IMMEDIATE DEFERRABLE
SET CONSTRAINT <name>
              [DEFERED|IMMEDIATE]
```

allow checking at arbitrary times

Complex CHECK clauses

CHECK clause may be an arbitrary predicate over the DB (SQL 99)

```
CREATE TABLE competitors_Price(title:..., price...)
CREATE TABLE movie (...
  CONSTRAINT price_Limit
   CHECK (" At least 4 * lowest price"))
```

Will be specified as SQL SELECT clause

CHECK clause defined as a predicate over one table only in most DBS implementations

4.3.4 Assertions and triggers

- Assertions
  - Integrity constraints defined independently from table definitions
  - Similar to CHECK constraints assigned to a table, when evaluated to FALSE: exception
  - semantic difference
    Table assigned constraints always hold for empty tables

```
CREATE ASSERTION never_noMovie
 CHECK (--"always at least one movie in
         -- Movie table"
         SELECT ...   )
```

  - Would be always TRUE as a table assigned check clause, but not as an assertion
  - Most current DBS do not support sophisticated constraints , e.g. table independent assertions

Assertions and triggers SQL / DDL

- Trigger

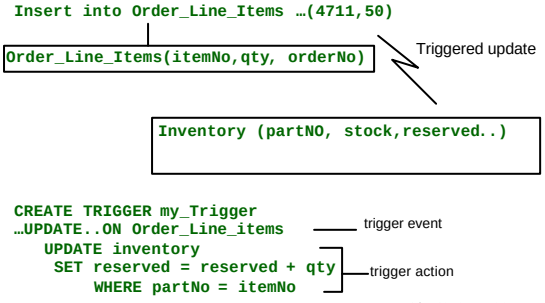
  (<predicate>, <action>)-rule

  Semantics:
  if <predicate> is true after DB state is changed
  <action> is performed
  - CHECK clause: exception if violated
  - TRIGGER: action is performed if condition holds
  - <action> may be
    - some operation on DB , e,g. cleanup

```
CREATE TRIGGER alwaysTape
   AFTER INSERT ON Movie
   REFERENCING NEW ROW AS m
   FOR EACH ROW WHEN
      NOT EXISTS (SELECT * FROM Tape
                  WHERE movie_id=m.id)
     <do something>; -- e.g. insert tape
```

TRIGGER

- Example for action on DB




```
CREATE TRIGGER my_Trigger
…UPDATE..ON Order_Line_items        ← trigger event
   UPDATE inventory
    SET reserved = reserved + qty   ← trigger action
        WHERE partNo = itemNo
```


Triggering applications

- Triggers very useful for triggering actions outside DB
  - Triggering audit trails
    If table is changed store an entry about this event in a special place (the audit trail)
  - Triggering an application program
    If a customer has rented a tape and she has a non-NULL email address, send her a mail

4.3.5 Case study

- Relation Schema Video-DB

```
CREATE TABLE Movie(
  m_id        INTEGER PRIMARY KEY,
  title       VARCHAR(60) NOT NULL,
  cat         CHAR(20),
  year         DATE,
  director    VARCHAR(40),
  price_Day   DECIMAL(4,2),
  length      INTEGER,
  CONSTRAINT plausible_year
   CHECK (year > TO_DATE('01.01.1900','DD.MM.YYYY')),
  CONSTRAINT allowedPrice
   CHECK ( (price_Day >= 0) AND (price_Day < 10.0))
 );
```


Example

```
CREATE TABLE Customer (
 mem_no       INTEGER PRIMARY KEY,
 name         VARCHAR) NOT NULL,
 first_name   VARCHAR(30),
 zip          CHAR (10),
 city         VARCHAR(30),
 );
```

```
CREATE TABLE Phones(
  phone    VARCHAR(40),
  customer INTEGER,
  CONSTRAINT pk_phone
    PRIMARY KEY(phone,customer),
  CONSTRAINT fk_phone
    FOREIGN KEY (customer)
    REFERENCES Customer (mem_no)
    ON DELETE CASCADE
);
```

```
CREATE TABLE Tape(
  t_id        INTEGER PRIMARY KEY,
  m_id        INTEGER NOT NULL,
  format      CHAR(5),
  aquired     DATE,
CONSTRAINT movieExists
       FOREIGN KEY (m_id) REFERENCES movie(m_id)
       ON DELETE CASCADE );
```

```
ALTER TABLE Tape
 ADD ( CONSTRAINT formatExists
       FOREIGN KEY (format) REFERENCES Format(format)
       ON DELETE SET NULL);
```

Example

```
CREATE TABLE Format(
  format   CHAR(5)PRIMARY KEY,
  extraCh  DECIMAL (3,2)
);
```

```
CREATE TABLE Rental(
 tape_id      INTEGER NOT NULL,
 mem_no       INTEGER,
 from_date    DATE NOT NULL,
 until_date   DATE,
 PRIMARY KEY (tape_id, from_date),
 CONSTRAINT fk_Tape
    FOREIGN KEY (tape_id) REFERENCES Tape(t_id)
    ON DELETE CASCADE,
 CONSTRAINT fk_Customer
    FOREIGN KEY (mem_no)
    REFERENCES Customer (mem_no)
    ON DELETE CASCADE);
```

```
CREATE TABLE Actor (
 stage_name VARCHAR(30)
 PRIMARY KEY,
 real_name  VARCHAR(30),
 birth_date DATE
);
```

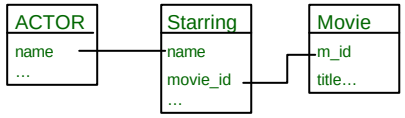
Referenced column must be unique

```
CREATE TABLE Starring (
  movie_id    INTEGER,
  actor_name  VARCHAR(30),
 CONSTRAINT pkStarr
       PRIMARY KEY (movie_id, actor_name),
 CONSTRAINT foreignKeyMovieID
       FOREIGN KEY (movie_id)
         REFERENCES Movie (m_id)
          ON DELETE CASCADE,
 CONSTRAINT foreignKeyStagename
       FOREIGN KEY (actor_name)
         REFERENCES Actor(stage_name)
          ON DELETE CASCADE
);
```

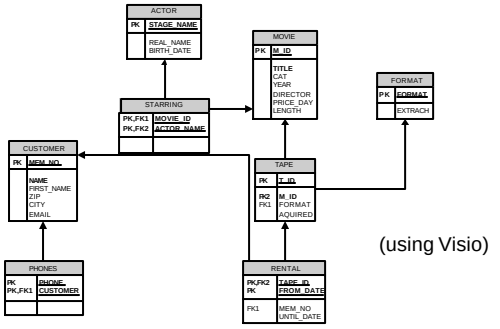
Note on Relational Database Design

Direct design of Relational DB

- some design tools do not support ER-M but use RDM directly
- Relationships defined by foreign keys
- Reengineering tools produce RDB schema with foreign keys



MovieDB reengineered



(using Visio)

4.3.6 Metadata management

Meta data

- All definitions and other "data on data" are called metadata
- Stored in system data structures
- Data structures for metadata called "the catalogue" or "data dictionary" in particular when used for more than one DB
- In most systems stored as tables
- Makes metadata "first class": may be queried und modified in the same way as the data tables

```
Select <Table_Name> from User_Tables;
```


Metadata management: example

Querying the catalog using SQL: a first example (ORACLE)

Attributes of the user_constraints table

```
SQL>  SELECT constraint_name,search_condition,
            delete_rule
      FROM  user_constraints
      WHERE table_name = 'MOVIE';
```

Result

```
CONSTRAINT_NAME  SEARCH_CONDITION                          DELETE_RULE
-----------------------------------------------------------------------

SYS_C001360     " TITLE" IS NOT NULL

PLAUSIBLE_YEAR   year > TO_DATE('01.01.1900','DD.MM.YYYY')

ALLOWEDPRICE     pricePDay >= 0) AND (pricePDay < 100.0)

SYS_C001363      TAPE_MOVIE                               CASCADE
```

No standard for metadata management! completely different in Postgres!

Virtual tables: views

More SQL Schema Definition Statements

- **CREATE TABLE** defines base tables
- View: virtual table, has a definition, but no extent, definition is executed when table is accessed

```
CREATE VIEW <name> AS  <SQL-select>
```

e.g.

```
CREATE VIEW C_T_List AS
     SELECT tapeId, cuNo FROM Rents;
```

May be used as ordinary tables for reads, updates more involved

To be discussed later…

4.3.7 Modifying and deleting definitions

ALTER TABLE

```
ALTER TABLE<tableName><redefinition>;
```

Add a column:

```
ALTER TABLE Tape
  ADD (AquiredBy CHAR(20));
```

Modify type:

```
ALTER TABLE Tape
  MODIFY (AquiredBy CHAR(30));
```

Many more variants of ALTER statement
see manual

Deletion of schema elements

Table delete

Delete table only if not referenced

```
DROP TABLE <tableName> restrict;
```

Delete table and reference

```
DROP TABLE <tableName> cascade;
```

Oracle:

```
DROP TABLE <tableName>
        cascade constraints;
```


Sequences

Sequences of numbers….

- useful for artificial keys: 1,2,3,….4711,…
- Where is the problem or
  How to count consistently?
- Persistence required!

- Pseudo columns

```
Create SEQUENCE tapeSeq; -- different
                         -- options
INSERT INTO Tape
(t_Id,m_Id,format,aquired)
 VALUES
(tapeSeq.nextval,4711,'DVD','2003-4-4');
```


Oracle and more

- Oracle
  - PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, REFERENCES, CHECK supported, uses sequence objects
- Postgres
  - very similar to ORACLE (SQL99), SERIAL type as an abbreviation of sequence objects
- MySQL (V3.x)
  - PRIMARY KEY, NOT NULL, UNIQUE supported
  - FOREIGN KEY, REFERENCES, CHECK accepted (for compatibility) but not supported
  - Simple but risky creation of unique artificial key values: `<attributName><attributeType> [<constraint>][AUTO_INCREMENT]`

Summary

- Standard Query Language (SQL)
  - Data definition language (DDL)
  - Data manipulation language (DML)
  - In almost all current DBMS
  - All SQL implementations differ from standard
  - core SQL99 is basically supported by high-end DBS
- Important terms and concepts
  - Data types
  - Create, change, delete tables
  - Referential integrity
  - Integrity constraints
  - TRIGGERS