Relational Databases

a) Mapping to relational database schema

RecordLabel = ({name, zip, city, street}, {Name})
Album = ({aid, title, recordLabelName, pubYear}, {aid})
SoloAlbum = ({aid, recordedBy}, {aid})
BandAlbum = ({aid, recordedBy}, {aid})
Artist = {{artid, name, telephone}, {artid}}
ArtistEmail = {{artid, email}, {{artid, email}}}
Band = {{bid, name, leader}, {bid}}
PartOf = {{artid, bid}, {{artid, bid}}}

Interrelational dependencies:
Album[recordLabelName] ⊆ RecordLabel[name]
SoloAlbum[aid] ≠ BandAlbum[aid]
SoloAlbum[recordedBy] ⊆ Artist[artid]
BandAlbum[recordedBy] ⊆ Band[bid]
ArtistEmail[artid] ⊆ Artist[artid]
Band[leader] ⊆ Artist[artid]
PartOf[artid] ⊆ Artist[artid]
PartOf[bid] ⊆ Band[bid]

b) Tupel calculus

{bl | (∃ b) (∃ sa) (∃ sa_a) (∃ rl)
  Artist(bl) ∧ Band(b) ∧ b[leader] = bl[artid] ∧
  SoloAlbum(sa) ∧ sa[recordedBy] = bl[artid] ∧
  Album(sa_a) ∧ sa_a[aid] = sa[aid] ∧
  RecordLabel(rl) ∧ sa_a[recordLabelName] = rl[name] ∧
  rl[city] = ‘Munich’
}

c) SQL

SELECT rl.name, b.name, count(*)
FROM Album a, BandAlbum ba, Band b, RecordLabel rl
WHERE rl.name = a.recordLabelName AND
  ba.aid = a.aid AND
  ba.recordedBy = b.bid AND
  a.pubYear < 1980 AND
  b.leader NOT IN (SELECT recordedBy FROM SoloAlbum)
GROUP BY rl.name, b.name

d) relational algebra

\[ \Pi_{\text{Name}}(\text{Band} \bowtie_{\text{recordedBy}} \text{BandAlbum} \bowtie_{\text{aid}} \sigma_{\text{recordLabelName} = \text{FunkyRecords}(\text{Album})} ) \]
- \[ \Pi_{\text{Name}}(\text{Band} \bowtie_{\text{bid}} \text{PartOf} \bowtie_{\text{artid}} \bowtie_{\text{recordedBy}} \text{SoloAlbum} ) \]
Algorithmic Schema Design
Consider a relation \( r = (U,F) \) with the attribute set \( U = \{ A, B, C, D, E, F, G, H, I, J \} \) and the set of functional dependencies \( F = \{ AB \rightarrow C, A \rightarrow DE, B \rightarrow F, B \rightarrow G, F \rightarrow G, D \rightarrow HIJ \} \).

a) Check for the following functional dependencies whether they hold or not:
- \( AF \rightarrow BI: AF \rightarrow ADEF \supseteq ADEF \supseteq BI \)
- \( AB \rightarrow DJ: AB \rightarrow ABC \rightarrow ABCDE \rightarrow ABCDEF \rightarrow ABCDEFG \rightarrow ABCDEFGHIJ \supseteq DJ \)

b) What are the key(s) of \( r \)?
   The only key is \( AB \):
   - super-key: see above
   - minimal: \( A \rightarrow ADHIJ, B \rightarrow BFG \)
   \( A \) and \( B \) do not occur on any right side of a FD. Therefore, they must be part of any key. This shows that \( AB \) is the only key.

c) In which normal form is \( r \)?
   1\(^{st}\) normal form: trivially true, since \( r \) is a relation!
   2\(^{nd}\) normal form: no, since e.g. \( A \rightarrow DE \) and not \( A \rightarrow U \)

d) Design a relational database schema using the decomposition algorithm. Does the resulting decomposition preserve all dependencies? If not, which is lost?

\( A \rightarrow DE \) violates BCNF:
\( R_1 = (ADE, \{A \rightarrow DE\}) \)
\( R_2 = (ABCFGHIJ, \{AB \rightarrow C, B \rightarrow F, B \rightarrow G, F \rightarrow G\}) \)

\( R_2 \) is not in BCNF, since \( B \rightarrow FG \) violates BCNF:
\( R_{21} = (BFG, \{B \rightarrow F, B \rightarrow G, F \rightarrow G\}) \)
\( R_{22} = (ABCHIJ, AB \rightarrow C) \)

\( R_{21} \) is not in BCNF, since \( F \rightarrow G \) violates BCNF.
\( R_{211} = (FG, \{F \rightarrow G\}) \)
\( R_{212} = (BF, \{B \rightarrow F\}) \)

decomposition \( \mathcal{R} = \{R_1, R_{211}, R_{212}, R_{22}\} \)
lost dependencies: \( D \rightarrow HIJ, B \rightarrow G \)

e) Design a relational database schema using the synthesis algorithm.

Basis calculation
- Make \( F \) \( r \)-minimal:
  \( B_1 = \{ AB \rightarrow C, A \rightarrow D, A \rightarrow E, B \rightarrow F, B \rightarrow G, F \rightarrow G, D \rightarrow H, D \rightarrow I, D \rightarrow J\} \)
- Make \( B_1 \) \( l \)-minimal
  \( B_2 = B_1 \)
- Remove redundant FD from \( B_2 \)
  \( B_3 = \{ AB \rightarrow C, A \rightarrow D, A \rightarrow E, B \rightarrow F, B \rightarrow G, F \rightarrow G, D \rightarrow H, D \rightarrow I, D \rightarrow J\} \)

Synthesize decomposition from this basis:
\( R_1 = (ABC, \{AB \rightarrow C\}) \)
\( R_2 = (ADE, \{A \rightarrow D, A \rightarrow E\}) \)
\( R_3 = (BF, \{B \rightarrow F\}) \)
\( R_4 = (FG, \{F \rightarrow G\}) \)
\( R_6 = (DHIJ, \{D \rightarrow HIJ\}) \)

no „key relation“ neccessary, since \( ABC \rightarrow U \)

\( \mathcal{R} = \{R_1, \ldots, R_6\} \)
Object Oriented Databases

a) Transfer the above ER-diagram into an UML structure diagram.

b) Let us assume for this subtask, that a car dealer can be in at most one purchasing association. Explain briefly the difference between the following types of object relationships. Give for each of these types two classes of your structure diagram, that are in a relationship which could be reasonably modelled by this type.

- association
- aggregation
- composition
c) Do not consider the bank account and customer entities for the following tasks:

### a. Map the ER diagram to ODL classes (including definitions of extents as entry points).

```java
class PurchasingAssoc
  (extent all_purchasingAssoc
    key name)
  {attribute name;
   relationship set<CarDealer> dealers inverse CarDealer::memberOfAssoc}

class LeasedCar extends Car
  (extent all_leasedCars
    key name)
  {attribute double leasingRate;}
```

### b. Formulate the following information demands as OQL queries against your database:

i. All car dealers who sell leased Mercedes cars together with the number of purchasing associations they belong to.

```sql
SELECT struct(dealer:d, purchasingAssoc: COUNT(p IN d.purchasingAssoc))
FROM d IN all_carDealers
WHERE d IN (SELECT lc.dealer FROM lc IN all_leasedCars WHERE lc.manufacturer='Mercedes')
```

ii. All car dealers with the average price of their purchased cars and the average leasing rate of the leased cars.

```sql
SELECT struct(dealer:d,
  avgPrice: avg(SELECT pc.price FROM pc IN all_purchasedCars WHERE pc.dealer=d),
  avgLeasing: avg(SELECT lc.leasingRate FROM lc IN all_leasedCars WHERE lc.dealer=d))
FROM d in all_carDealers
```

iii. All purchasing associations with a list of their members that sell BMW cars.

```sql
SELECT struct(purchasingAssoc: pa,
  dealer: (SELECT d FROM d IN pa.dealers
             WHERE d IN (SELECT c.dealer FROM c IN all_Cars
                           WHERE c.manufacturer='BMW')))
FROM pa in all_PurchasingAssoc;
```
XML and Semistructured Databases

a) Give an example XML document that is valid according to the XML schema.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <book isbn="132-42345-232">
    <title>The Big Nothing</title>
    <author>The Writer</author>
    <author>The CoWriter</author>
    <publicationYear>2002</publicationYear>
  </book>
  <journal>
    <title>The Nothing Journal</title>
    <editor>The Chief</editor>
    <editor>The Assistant</editor>
    <firstIssue>1</firstIssue>
  </journal>
</library>
```

b) Transform the XML schema into an equivalent DTD (with library as the root element) as far as possible. Which constraints cannot be expressed in a DTD?

```xml
<!DOCTYPE library [
<!ELEMENT library (book|journal)+>
<!ELEMENT book (title, author+, publicationYear)>
<!ATTLIST book isbn CDATA #REQUIRED>
<!ELEMENT journal (title, editor+, firstIssue, lastIssue?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publicationYear (#PCDATA)>
<!ELEMENT editor (#PCDATA)>
<!ELEMENT firstIssueAvailable (#PCDATA)>
<!ELEMENT lastIssueAvailable (#PCDATA)>
]> Constraints, that cannot be represented in DTD:
- datatype of publicationYear, firstIssueAvailable, lastIssueAvailable
- upper limit of repetition of author and editor elements (possible, but complex;
  all possible repetitions have to be enumerated; impracticable for larger upper limits)
  example for author in book:
    (author | (author, author) | (author, author, author))

c) Formulate an XQuery against a valid document of this type that lists for each author the books he/she has written and were published after 2002 and the number of journals he/she is an editor of in the following output format:

```xml
<result>
{for $a in distinct-values(doc("library.xml")/library/book/author)
  return
    <author name="{$a}'">
      {for $b in /library/book[author=$a] where $b/publicationYear>2002
        return
          <book title="{$b/title}" publicationYear="{$b/publicationYear}"/>
      }
      <journalEditor noJournals="{count(doc("library.xml")/library/journal[editor=$a])}'"/>
    </author>
} </result>
```
d) Consider the two XSLT scripts presented below. Decide for each XSLT script whether it fulfills these requirements. Identify and correct all mistakes you find.

Both scripts are incorrect.

Script a is wrong because of the following mistakes:

- The journals are not excluded from the transformation.

- The authors of the books are not excluded.
  Solution: Use two apply-templates constructs with select-attributes as presented in b)

- The title information is not embedded in title elements.
  Solution: Use copy-of instead of value-of or surround it with title tags as presented for the published element.

Script b contains one mistake:

- The title of a book is not stored inside a dedicated title element.
  `<xsl:apply-templates select="title"/>` selects the title element in the source document for further processing, but the default rules only copy the text content, not the element tags. Compare the treatment of the publicationYear element to see how to correct this mistake.

Questions:

- What is semi-structured data?
- What are differences between the Relational and the Semi-Structured Data Model? What are the advantages/disadvantages? When is which usually used?
- What are differences between semi-structured data models (such as OEM) and XML?
- What is RDF? What is the purpose of RDF?