Advance Database Models (Exercise 1)

1. Database: organized collection of data
2. Database Management System: kind of software used to manage the database
3. Database System: Database + DBMS = CRM plus
4. Database Model: set of concepts used to describe data, defines syntax, e.g., relational database model: tables, rows, keys
5. Data Model: data model for the DB. How data can be structured
6. Database Schema:

- DataModel: describes in an abstract way how data is structured in an information system
- Database Model:
- Database schema:
- DB:
- DB System:

- DBMS: software used to manage data used to manage data, independent of the application. E.g., Oracle 10g
- *DB = CRM@A, CRM@B, CRM@C
- *DB = application
- *DB System:

- *Database Schema: structure of data, we have no idea how it's related. Organization, contacts, addresses, phones etc.
- Database schema are specific to a certain universe of discourse, e.g., contacts, addresses etc are specific to CRM@A
- Logical data model (implementation)
- *Database Model: Object Relational Model (independent of application)
- *Data Models: Java 100, Hibernate (early design stage) (implementation)

Hibernate middleware: When data is required in Java as well as ORDBMS, we may need Java object and semantic ORDBMS objects. Mapping of objects into other objects is done by Hibernate (mapping between data models)

Java: data can be structured in the form of objects and classes
Different datamodels at different stages.

Graphical way describing what should be implemented

conceptual design: before implementation

Artists

- aid
- artistname
- country
- year

CD

- title
- label
- year

Tracks

- title
- duration

Band

- bid
- country

for optional relationships its better to put into a table coz it will give NULL value.

Logical Design: figure out tables, attributes, relationships etc.

CD (Cid, Title, Label, Year)

Artist (aid, artistname, country, year)

Band (bid, country)

Tracks (bid, songtitle, duration, cid)

CD-A (Cid, Aid)

CD-B (Cid, Bid)

Member (MID, name, BID)

Data Definition using SQL

Create Table CD

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cid</td>
<td>integer</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>Title</td>
<td>varchar</td>
<td>Title of the CD</td>
</tr>
<tr>
<td>Label</td>
<td>varchar</td>
<td>Label of the CD</td>
</tr>
<tr>
<td>Year</td>
<td>int</td>
<td>Year of the CD</td>
</tr>
</tbody>
</table>

Create Table Tracks

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bid</td>
<td>integer</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>songtitle</td>
<td>varchar</td>
<td>Name of the song</td>
</tr>
<tr>
<td>duration</td>
<td>varchar</td>
<td>Duration of the song</td>
</tr>
</tbody>
</table>

Create Table Artist

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>integer</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>artistname</td>
<td>varchar</td>
<td>Name of the artist</td>
</tr>
<tr>
<td>country</td>
<td>varchar</td>
<td>Country of the artist</td>
</tr>
<tr>
<td>year</td>
<td>int</td>
<td>Year of the artist</td>
</tr>
</tbody>
</table>

Create Table Band

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bid</td>
<td>integer</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>country</td>
<td>varchar</td>
<td>Country of the Band</td>
</tr>
</tbody>
</table>

Create Table Member

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mid</td>
<td>integer</td>
<td>Unique identifier</td>
</tr>
<tr>
<td>name</td>
<td>varchar</td>
<td>Name of the Member</td>
</tr>
<tr>
<td>bid</td>
<td>integer</td>
<td>Unique identifier</td>
</tr>
</tbody>
</table>

Foreign Key:

- `bid` references `mid`
- `Cid` references `mid`
Physical design: needed for performance benefits, optimization creation of indexes for lookups and joins.

Create Index 
ON Bands (Name)

**EXERCISE 3**

<table>
<thead>
<tr>
<th>ThreadID</th>
<th>Subject</th>
<th>Nickname</th>
<th>created by</th>
<th>written by</th>
<th>Role</th>
<th>GroupID</th>
<th>GroupName</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, *]</td>
<td>[1, *]</td>
<td>[1, *]</td>
<td>[1, *]</td>
<td>[1, *]</td>
<td>1:*</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Attribute in a table cannot hold multiple values

1) 1st Normal form: It is in first normal form (atomic values)
   - ThreadID, subject, user ID, nickname
   - messageID

2nd Normal form: If table is in 1NF and no non-prime key attribute is dependent on the proper subset of any candidate key of a table (partial dependency)
   - All non prime key attributes are fully functional dependent of any key of K

1st Normal form: users_interest

A B C D E
1 7 7 7
2 2 2 2
1 1 1 1
## Exercise 2

1. a) Integration - uniform, non-redundant data management
   b) Operations - insert, query, update, delete, select, calculation
   c) Catalog - access database description in the data
   d) Usability - different users/application must be able to have
   e) Integrity - ensure conformity of database contents with real world
   f) Security - prevention of unauthorized access
   g) Transaction - multiple DB operations as an atomic unit
   h) Synchronization - coordination of concurrent transaction
   i) Database - data recovery after system error

Database System = DB + DBMS

Management of real-world facts could be done by following systems:

<table>
<thead>
<tr>
<th>EXCEL</th>
<th>Network File System (File servers in a centralized array)</th>
<th>MySQL</th>
<th>DB2 Big (Relational DB System)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Integrity</td>
<td>-</td>
<td>-</td>
<td>RDBMS + constraints + triggers</td>
</tr>
<tr>
<td>b) Operations</td>
<td>+</td>
<td>only file operation</td>
<td>no database operation</td>
</tr>
<tr>
<td>c) Synchronization</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
</tr>
<tr>
<td>d) Integration</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
</tr>
<tr>
<td>e) Views</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>f) Transaction</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>g) Security</td>
<td>-/+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>h) Recovery</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>i) Catalog</td>
<td>-</td>
<td>-/+</td>
<td>+</td>
</tr>
</tbody>
</table>

Certain files like data are not always
which can be used, read never

* Integrity:* we can put constraints and triggers function in RDBMS
  strong of real world data correctly
  NFS - doesn't care of the content of the file, applications need to take care of it

* Operations:*
* Synchronization:* in excel everyone gets own copy
  in file system din. that during many people access same
data, quite restrictive.

In DB synchronization is reduced to the granularity of tuple

* Integration:* store thing once, avoid redundancy
  access network file system and put research restriction.
* Views - Using Excel Professional there is a way to represent views for specific audience
* Transaction - DB must be consistent before and after operation
  * follows ACID properties
* Security - can password protect Excel file and NFS but not fine grained
  * grant access, privileges, control of access

Ans: 2. Three Level Schema Architecture - describes why DB are used

- External Schema
- Conceptual Schema
- Internal Schema

Different Uses:

- Logical Data Independence
  (I don't need to know the data)
- Physical Data Independence
  (I don't need to know data in conceptual schema as nothing will interfere with my views (logical data independence))
  - I am also decoupled with the details of storage of my data (physical data independence)

- If I am only interested in certain data I just need relevant data. So database knows where are they stored where is it partitioned?
  - Typical query processing

- If I am only interested in certain data I just need relevant data. So database knows where are they stored where is it partitioned?
  - Typical query processing

Application running on LSF

- for Physical data independence
  - if there is a failure then there is no problem
  - if the application needs a student data you don't need to know where the data is stored

- for Logical data independence
  - different views for different people
  - Views are a subset of data for one system
1. Primary key: (matrNo) Student, lecture(id), attends(matrNo, id)
2. NOT NULL: all except - grade, lecturername, birthdate
3. Foreign Keys: Attends (matrNo), Attends (lecture id)
4. Range Constraints: Attends (grades) 1.0 ≤ grade ≤ 5.0
   can be value constraints
   pattern constraints: attends (semester) like wise case
5. Datatypes
   These are also constraints and a set of allowed values,
   int, varchar, decimal
(Conceptual Design Models)

Advance Database Models (Exercise 4)

ANS: 1 - Conceptual = High level, abstract, graphical

a) Basic ER

- **BrandName**
- **Home Power**
- **Car Model**
  - M:N with **Product Line**
- **Built by**
  - N:N with **Subsidiary**
  - 1:N with **Subsidiary**
- **Subsidiary**
  - **Country**
  - **Manpower**
- **Limosine**
  - **Company**
  - **Stationary**
  - **Car Type**
  - **Sportive**

*We can take ID's for passenger car for each type and store in a table for 'sportive', 'limousine', etc.*

b) ER with object-oriented notations

(Semantically Rich Model)

*Total Specialization = every instance of a supertype must be an instance of at least one subtype*

*Partial Specialization: an instance of a subtype can be an instance of a supertype or instance of a type itself*
Much more meaningful, more semantics

```
CarModels
  \rightarrow partial specialize
  (car models can have other models than Truck and PC)

  \rightarrow disjoint (PC \cap T = \emptyset)
  \quad PC \subset CM
  \quad T \subset CM

Passenger Car

Truck
```

```
Passenger Car (brandname, horsepower, # seats, # doors)

Trucks (brandname, horsepower, # axes, max freight)

CarModels (brandname, horsepower)
```

```
CM

  \rightarrow overlapping due to cabins and spews car

PC

Trucks

Cabin

SC

Lim

SW
```

```
C) UML

abstract car model

  Brandname

  HP

\rightarrow disjoint

PC

Tr

Partial and Total specialization could be expressed via abstract classes.

These classes don't have slots instances but used to express duel classes.

We can also use object constraint language where we can define the subset of classes etc.
```
Ans:\rightarrow Logical Design

* Artist must be singer or songwriter or both at the same time
* Artist may be influenced by other artist
* Singer songwriter becomes singer songwriter.

1. Approach: Store everything in one table
   - Artist (ID, name), I is singer, I songwriter, sex, genre)
   - Performance (Singer ID, Writer ID)
   - Influence (Artist ID, influence Artist ID)

2. Approach: Create tables for singer and songwriter:
   - Artist (ID, name)
     - Singer (Singer ID, sex,)
     - Songwriter (Songwriter ID, genre)
     - Singer Songwriter (ID) = (overlapping between singer, songwriter)
   - Performance (Singer ID, Asst ID)
   - Influence (Artist ID, Influential Artist ID)

3. Approach: Map all non abstract types to relations with all attributes
   - Artist is abstract and has total specialization

   Singer (ID, name, sex)
   Songwriter (ID, name, genre)
   Singer Songwriter (ID, name, eylet)
   Performance (Singer ID, songwriter ID)
   Influenced by (Not possible)}
EXERCISE 5

ANS: 1. NF2 relations or nested relations provide non-atomic attributes by allowing an attribute to be of a relationship type (schema) and value of such a type was nested relation.

<table>
<thead>
<tr>
<th>Department</th>
<th>SSN</th>
<th>Name</th>
<th>Telephone 1</th>
<th>Telephone 2</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>4711</td>
<td>Todd</td>
<td>038283-12230</td>
<td>03811-498---</td>
<td>6000</td>
</tr>
<tr>
<td></td>
<td>5588</td>
<td>Wight</td>
<td>0591-3567777</td>
<td>0191-3392---</td>
<td>6000</td>
</tr>
<tr>
<td>Mathematics</td>
<td>6884</td>
<td>Beast</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PNF nested relation forms a subclass of nested relation in general in which the set of atomic attributes at every level forms the key of the level. Parent Child

\[
P_1 \rightarrow C_1
\]

\[
P_2 \rightarrow C_2
\]

\[
P_3 \rightarrow C_3
\]

\[
P_4 \rightarrow C_4
\]

\[
P_5 \rightarrow C_5
\]

PNF requires a flat key on every nesting level of a nested relation.

Car Sales

Model     Manufacturer     Car

Build Year Km       LP     Interested Customer

Name          Last offer  Telephone
### Car Sales

<table>
<thead>
<tr>
<th>Model</th>
<th>Manufacturer</th>
<th>Car 1</th>
<th>Car 2</th>
<th>Z.C</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>A6</td>
<td>Audi</td>
<td>2008</td>
<td>2010</td>
<td>Stefan</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12000</td>
<td>10000</td>
<td>Eike</td>
<td>3000</td>
</tr>
<tr>
<td>S500</td>
<td>Mercedes</td>
<td>2010</td>
<td>2012</td>
<td>Andrea</td>
<td>8000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60000</td>
<td>40000</td>
<td>Mike</td>
<td>9000</td>
</tr>
</tbody>
</table>

---

**Partition Normal Form**: flat key on every nesting level.

1. Needs to be flat
2. Needs to be a key

---

For car nesting level with this data, it works but when we add more data, it is highly unlikely to be a key worth BY, LP, KM.

(same info represented in a redundant way)

---

If I do unnesting I don't get the same thing.

---

Would be meaningful in the application.

PNF — quality criteria to perform nesting and unnesting operations.
$NP^2$: A tuple consists of a set of tuples

- tuples may contain sets of lists, lists of lists

Relational scheme:

- Auction (aid, title, description, sp, sellerid)
- Members (mid, name, email)
- Bids (aid, mid, price)
- Ratings (mid, date, score, command)
- Images (aid, binary, pos)

For $ENP^2$

create array of images

- auction (aid, title, description, sp, sellerid, image[3])

- member (mid, ... ... , ratings MULTISET (date, score, ... ... , BAG)

- bid biddings SET (auction id, price)
b) Combine all values of inner nesting level with outer nesting level.

Unnesting needs to be done inside out.

<table>
<thead>
<tr>
<th>A</th>
<th>Audi</th>
<th>2008</th>
<th>12000</th>
<th>11000</th>
<th>Stefan</th>
<th>10700</th>
<th>0174</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Audi</td>
<td>2008</td>
<td>12000</td>
<td>11000</td>
<td>Eric</td>
<td>98000</td>
<td>0177</td>
</tr>
<tr>
<td>C</td>
<td>Audi</td>
<td>2010</td>
<td>10000</td>
<td>13000</td>
<td>Alice</td>
<td>85000</td>
<td>0173</td>
</tr>
</tbody>
</table>

Creating a flat table out of a nested table.

Unnesting of this relation will be a cross product.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

[Table diagram]

In the model:
- CarsMasterModel (id, model, manufacturer)
- CarDetails (id, BuildYear, kms, ListPrice, modelId)
- Customers (id, Name)
- CustomerNumbers (id, customerId, Telephone)
- InterestedCustomers (CarDetails, Customers & LOC)

d) i) Select d.ListPrice from CarDetails carMasterModel c.
    Where d.Id = c.Id
    And c.Manufacturer = "Citroen"

Select car.ListPrice
from Carsales, UNNEST Carsales.Car
where Carsales.Manufacturer = "Citroen"
\[ T_{\text{lastprice}} (J_{M='C'} (\text{Year (Carsales)})) \]

(In oracle we have Table operation in case of UNNEST)
Can also use FUSION $\rightarrow$ creates a set of last price

- SELECT Cust. last offer
  FROM Carsales, UNNEST Cars, UNNEST Interest CustName as Cust
  WHERE Model = "Beetle"
  and manufact = 'vw'

- SELECT Telephone. *
  FROM Carsales, UNNEST Cars, UNNEST IC as cust, UNNEST Dph as Telephone
  WHERE Model = "Ford Fiesta"
  and Cust.Name = "Stuart".