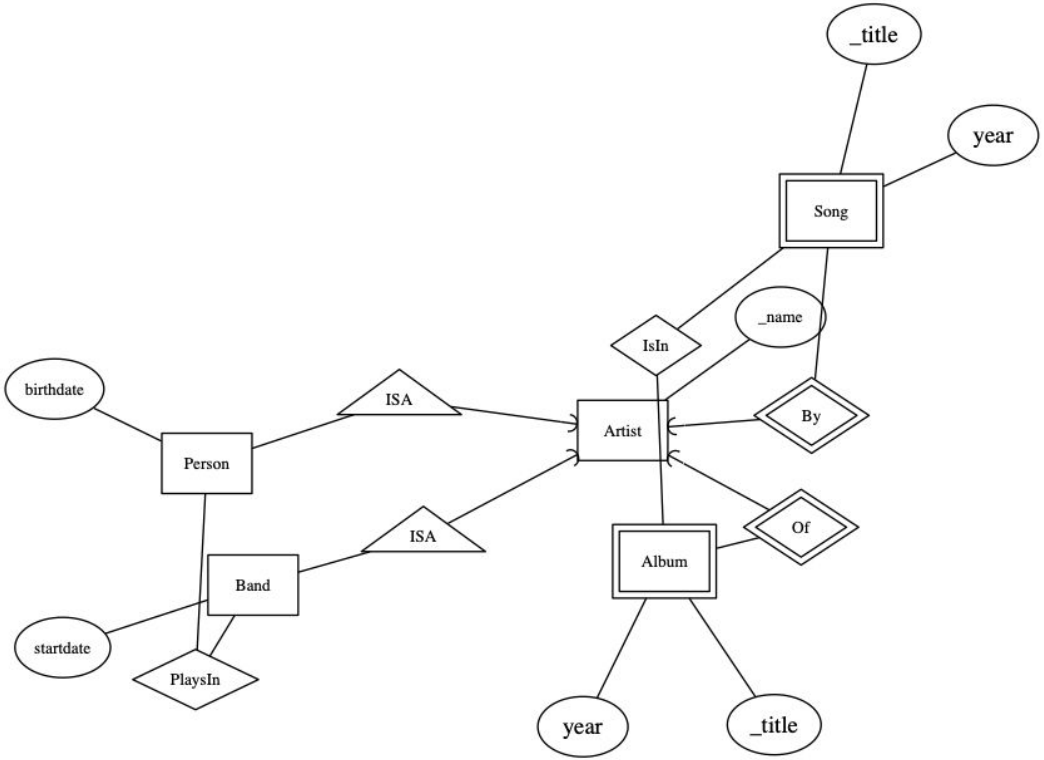
Database Exam Solutions 22 March 2019

Aarne Ranta
CSE, Chalmers & GU

Exam questions

1. ER modelling



Artist(_name)

Person(_name,birthdate)
  name -> Artist.name

Band(_name,startdate)
  name -> Artist.name

PlaysIn(_personName,_bandName)
  personName -> Person.name
  bandName -> Band.name

Album(_title,year,_artistName)
  artistName -> Artist.name

Song(_title,year,_artistName)
  artistName -> Artist.name

IsIn(_songTitle,_songArtistName,_albumTitle,_albumArtistName)
  (songTitle,songArtistName) -> Song.(title,artistName)
  (albumTitle,albumArtistName) -> Album.(title,artistName)

2. Functional dependencies

2.1
Attributes: A B C D E
Functional dependencies:
A -> B
C -> D
E -> A

Keys:
C E

All attributes belong to its closure:
A, because of E->A
B, because of E->A->B
C, because of C->C
D, because of C->D
E, because of E->E

BCNF violations (the first 3):
A -> B, C -> D, E -> A

2.2
BCNF decomposition on A -> B:

1. Attributes: A B, Keys: A
No violations

2. Attributes: E A C D, Keys: E C
Violations: E -> A

2.1 Attributes: E A, Keys: A
No violations

2.2 Attributes: E C D, Keys: EC
Violations: C -> D

2.2.1. Attributes: C D, Keys: C
No violations

2.2.2. Attributes: C E, Keys: C E
No violations

2.3
Example:

It helps to order the FDs in this way:

E -> A -> B C -> D

These are like two independent relations. But it can make sense to link them together. Here is an example:

Country -> Currency -> Value
Language -> LanguageFamily

Assuming

- Country determines currency, which determines value
- Language belongs to a definite family
- A country can have several languages from different families

3. SQL queries

3.1

```
SELECT title, year
FROM Albums
WHERE artist = 'Metallica' AND year >= 2000 ;
```

3.2

```
SELECT SUM(length)
FROM Songs,Tracks
WHERE album = 'Vespertine' AND artist = 'Björk' AND title = song ;
```

3.3

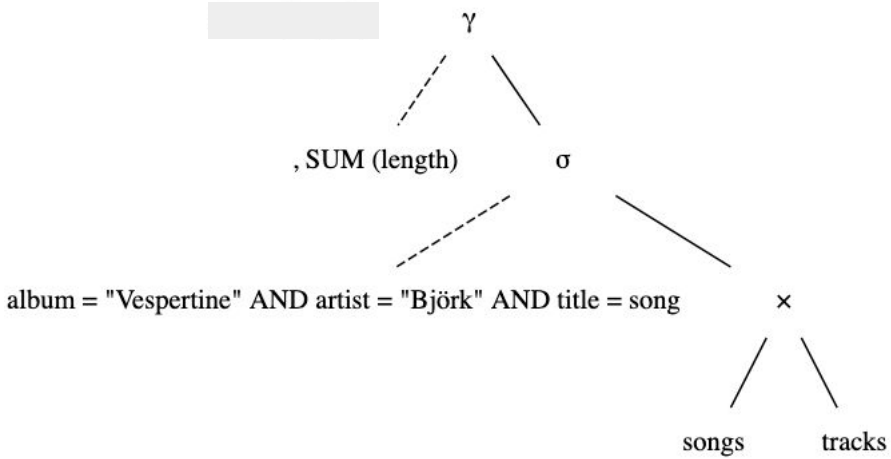
```
SELECT title
FROM Albums
WHERE artist IN (SELECT name FROM Persons) ;
```

3.4

```
SELECT name
FROM Members, Persons
WHERE birthdate = (
  SELECT MIN(birthdate)
  FROM Members,Persons
  WHERE band='Metallica' AND person = name
  ) AND band='Metallica' AND person = name ;

-- alternative:
WITH MetallicaMembers AS (
  SELECT person, birthdate
  FROM Persons, PlaysIn
  WHERE band='Metallica' AND person = name
  )
SELECT person
FROM MetallicaMembers
WHERE birthdate = (
  SELECT MIN(birthdate)
  FROM MetallicaMembers)
```

4. Algebra and theory




```
SELECT * FROM Albums NATURAL JOIN Songs

       title       |  artist   | year | length
-------------------+-----------+------+--------
 Master of Puppets | Metallica | 1986 |    513

SELECT * FROM Albums FULL OUTER JOIN Songs
  USING (title,artist)

       title       |  artist   | year | year | length
-------------------+-----------+------+------+--------
 Cocoon            | Björk     |      | 2001 |    264
 Hidden Place      | Björk     |      | 2001 |    264
 Master of Puppets | Metallica | 1986 | 1986 |    513
 Vespertine        | Björk     | 2001 |      |
```

Does this make sense? Not really. A join makes sense if the two tables are about the same objects, identified by the joining attributes, and just give different informations about these objects. But here, albums and songs are different objects, which just sometimes happen to have the same title.

5. Constraints and triggers

1. Every song in the playlist exists.
Yes: FOREIGN KEY (song,artist) REFERENCES …

2. All playlists of one and the same owner have different names.
No: the following would be OK
PL001, hottest hits, Joe
PL002, hottest hits, Joe

3. All positions in a given playlist are unique.
Yes: PRIMARY KEY (playlist,position)

4. The positions can be listed in order 1,2,3,... with no numbers missing
No: the following would be OK: 2,7,11,...

```
CREATE VIEW PlaylistM123 AS (
  SELECT position, song, Songs.artist, length
  FROM PlaylistSongs, Songs
  WHERE playlist = 'M123'
        AND song = Songs.title
        AND PlaylistSongs.artist = Songs.artist
  ORDER BY position
  ) ;
```

```
CREATE FUNCTION insertPlaylistFunction() RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.position > (SELECT MAX(position) FROM PlaylistSongs WHERE
playlist = 'M123'))
    THEN INSERT INTO PlaylistSongs VALUES
      ('M123',NEW.song,NEW.artist,
 1+(SELECT MAX(position) FROM PlaylistSongs WHERE playlist = 'M123')) ;
    ELSE
      UPDATE PlaylistSongs
        SET position = position + 1
        WHERE position >= NEW.position AND playlist = 'M123' ;
      INSERT INTO PlaylistSongs VALUES ('M123',NEW.song,NEW.artist,
NEW.position) ;
  END IF ;
  RETURN NULL ;
END
$$ LANGUAGE 'plpgsql' ;

CREATE TRIGGER insertPlaylist
  INSTEAD OF INSERT ON PlaylistM123
  FOR EACH ROW
  EXECUTE PROCEDURE insertPlaylistFunction() ;
```

6. JSON

6.1

A table listing all members of a band on a row would violate the first normal form: that all posts in a tuple have atomic values. Or the even more fundamental principle that all tuples have the same length.

6.2

```
{ "type" : "array",
  "items":{
     "type" :"object",
     "properties": {
        "band": {"type": "string"},
        "members": {
           "type": "array",
           "items": {
              "type":"object",
              "properties": {
                 "name" : {"type":"string"},
                 "year" : {"type": "integer"}
              }}}}}
```

6.3

```
[
  {"band": "The Beatles",
   "members": [
       {"name": "John Lennon", "year": 1960},
       {"name": "Paul McCartney", "year": 1960},
       {"name": "Ringo Starr", "year": 1962},
       {"name": "George Harrison", "year": 1960}
   ]
  },
  {"band": "Metallica",
   "members": [
       {"name": "James Hetfield", "year": 1981},
       {"name": "Lars Ulrich", "year": 1981},
       {"name": "Kirk Hammett", "year": 1983},
       {"name": "Dave Mustaine", "year": 1982},
       {"name": "Cliff Burton", "year": 1982}
   ]
  }
]
```

6.4

```
$.[*].[?(@.band=="The Beatles")].members[?(@.year>1960)]
```