i Forside

UNIVERSITY OF OSLO

The Faculty of Mathematics and Natural Sciences

Exam for IN2090/INF1300

Date: 6. December 2019
Time: 14:30–18:30
Place: Silurveien 2 (Sal 3B, 3C, 3D, 4C, 4D)

Examination support material:

- The book Elmasri & Navathe: Fundamentals of Database Systems, Global Edition, 7th Edition (book used in IN2090, from 2019).
- The book Halpin & Morgan: Information Modelling and Relational Databases, Second Edition (book used inn INF1300 and IN2090 until 2018).
- 4 handwritten A4-pages with notes (2 sheets if both sides are written on)

The exam consists of 3 parts (maximum points in parenthesis):

1. Modelling (40)
2. SQL (50)
3. Relational model (10)

In this exam, you are supposed to make drawings to answer the modelling exercises (task 1.1 and task 1.2). You are to use the sketching paper handed to you in the exam room. You can use more than one sketching sheet per task. See instructions for filling out sketching sheets in the link below the task bar. In these two exercises you can choose whether to solve them using the modelling language ER or ORM2.

You may NOT hand in sketching sheets for any other tasks than task 1.1 and task 1.2. You will NOT be given extra time to fill out the "general information" on the sketching sheets (task codes, candidate number etc.)

On the multiple choice exercises 4, 12 and 13, wrong answers give negative points. However, the minimum total number of points on each whole exercise is 0.

1.1 Modellering: Kandidater

In this (and the next) exercise you will make a model for a database that contain information about interview candidates (for software developers) and the interviews they participate in. You can choose whether to solve these exercises in ER (an overview of ER-notation is attached) or ORM2 (an overview of ORM2-notation is attached).

The models should be drawn on paper using Scantron. You can choose whether you want to make one large model for both exercises, or split them up into two models. If you choose to split them up, you only need to include the entities (and their keys) for those entities you need from exercise 1 in the model in exercise 2. Please feel free to add comments to your models. Please write and draw clearly.

The model we will make in this first exercise will model candidates:

1. For each candidate the database should be able to record the candidate’s personal number (unique among candidates), name, and email address (unique among candidates).
2. Furthermore, the database should record information about the current employer of each candidate. If the candidate is currently unemployed, no current employer should be recorded; if the candidate has more than one current employer (e.g., currently has two part-time jobs), for simplicity, we limit the information to be recorded only to one of them (i.e., each candidate has at most one current employer in our database). The job title the candidate has at its current employer should also be recorded (e.g. "product manager").
3. For each employer the database should record an id (unique), name, and contact information (composed of: phone, email, and address).

In this task you can hand in sketches. Use the sketching paper handed to you in the exam room for this. See instructions in the link below the task bar.

Maximum marks: 7

1.2 Modellering: Intervjuer

We will continue on our model of candidates from the previous exercise. In this exercise we will model interviews.

Just like the previous exercise, you can choose whether to use ER or ORM2. You can also choose whether to make one large model for both exercises, or split them up into two smaller models. If you choose to split them up, you only need to include the entites (and their keys) for the entities you need from exercise 1 in your model in this exercise.

The model you will make in this exercise should contain the following information about interviews:

1. For each candidate the database should be able to record the interviews the candidate participates in. For each interview, an interview number, date, and time are recorded. Interviews are uniquely identified by the combination of the candidate's personal number and the interview number. Examples of interviews include: the candidate with personal number 123 participates in interview with number #1 on Dec 18th 2019 at 2pm; the candidate with personal number 123 participates in interview with number #2 on Dec 21st 2019 at 3pm; the candidate with personal number 456 participates in interview with number #1 on Dec 21st 2019 at 3pm; etc.
2. Interviews are conducted by interviewers on certain topics (*hint*: use a ternary relation). Each interviewer has an employee id and phone numbers recorded. Each topic has a unique name (e.g. "data structures") and a set of tests associated with it (each test is simply represented as a string in our database).
3. The database should be able to record the fact that many interviews may be conducted by many interviewers on many topics. In other words, given an interview and an interviewer, we can have many topics; given an interview and a topic we can have many interviewers; and given an interviewer and a topic, we can have many interviews. For example: the candidate with personal number 123 participates in interview with number #1 conducted by interviewer with employee id 111 on topic "data structures"; the candidate with personal number 456 participates in interview with number #1 conducted by interviewer with employee id 111 on topic "Java programming", etc.
4. Furthermore, the database should be able to capture that every interviewer is certified on at least one topic (but can be certified on many). Every topic can have many certified interviewers. Certification happens on a particular date by a particular certification provider. E.g., the interviewer with employee id 111 is certified on the topic "data structures" and received the certification on Jan 19th 1980 from the University of Oslo.
5. In addition, each topic must be assigned to exactly one responsible interviewer (e.g., the interviewer responsible for designing the tests for a given topic). An interviewer may be responsible for at most one topic (but does not need to be responsible for a topic).

In this task you can hand in sketches. Use the sketching paper handed to you in the exam room for this. See instructions in the link below the task bar.

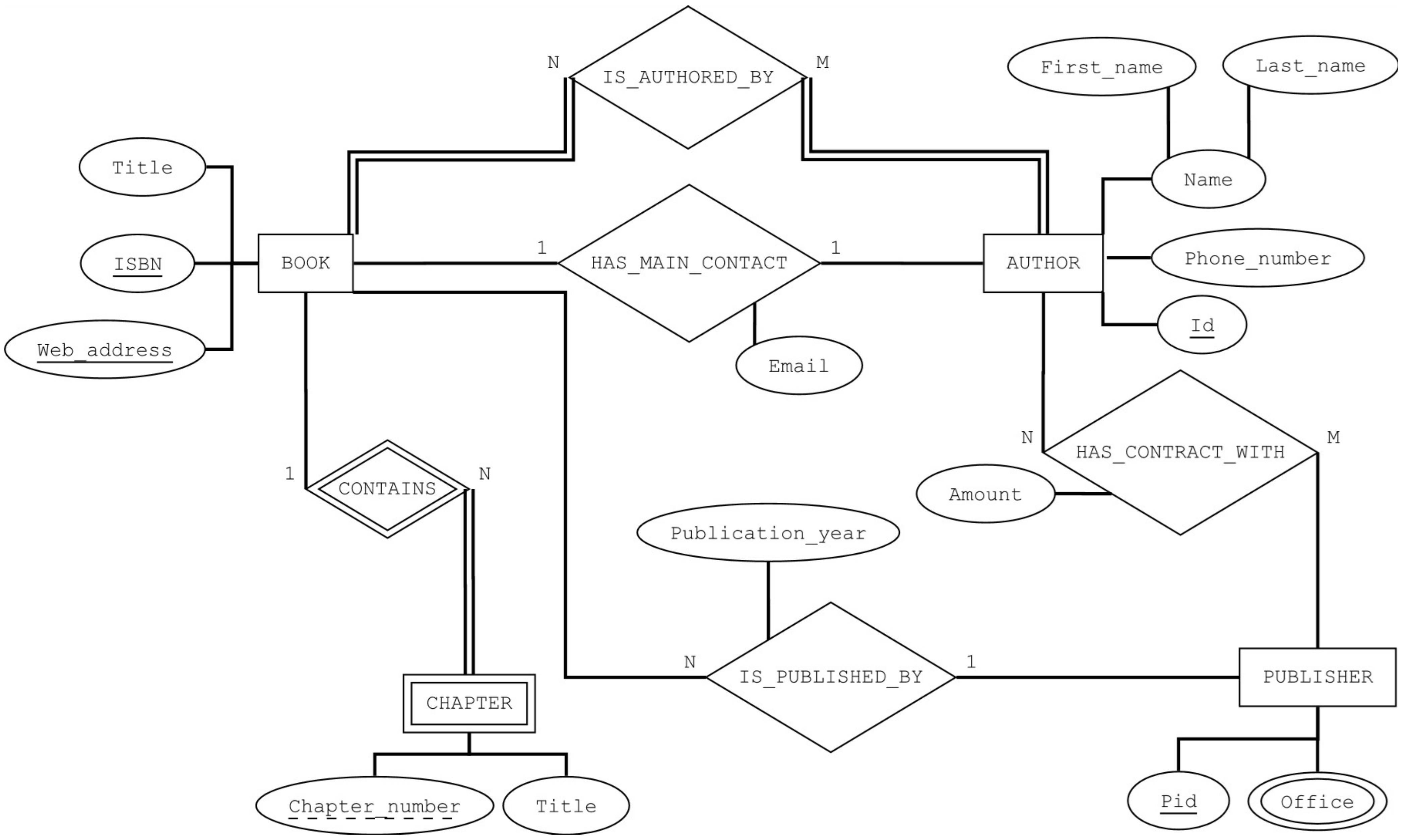
Maximum marks: 18

1.3 Realisering

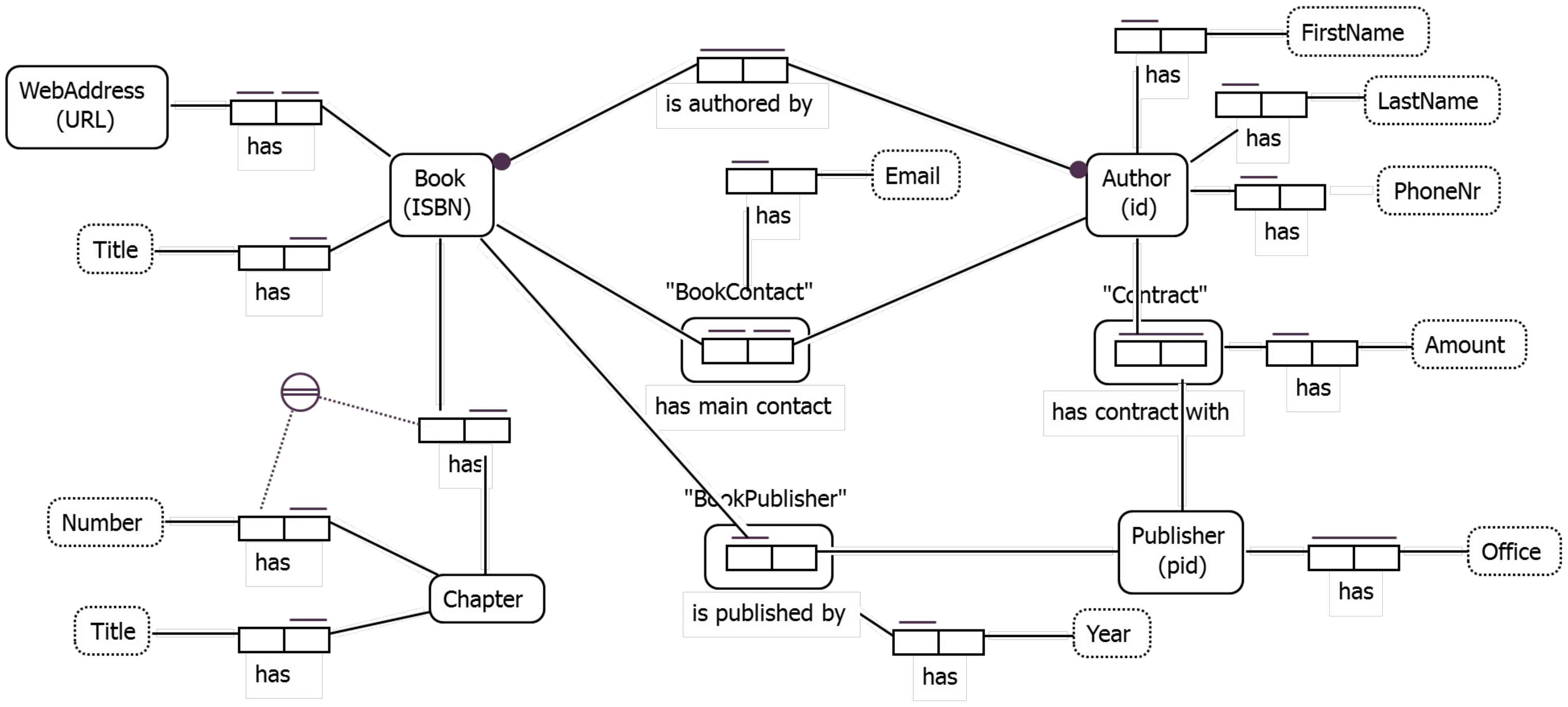
Map either the following ER model or ORM2 model to a relational database schema using the algorithm for realization. Please explain any decisions you make along the way.

Use underlines to mark candidate keys. In addition, use **bold font** to indicate the primary key for each relation (buttons for underlining and bold font you will find in the menu above). Also list all foreign keys on the form T(A) -> P(B) (here the attribute(s) A of T are foreign keys referencing P's B-attribute(s)).

ER-model:


N
IS_AUTHORED_BY
M
First_name
Last_name
Title
Name
BOOK
ISBN
1
HAS_MAIN_CONTACT
1
AUTHOR
Phone_number
Web_address
Id
Email
N
HAS_CONTRACT_WITH
M
1
CONTAINS
N
Amount
Publication_year
N
IS_PUBLISHED_BY
1
PUBLISHER
CHAPTER
Chapter_number
Title
Pid
Office

ORM2-model:


FirstName
has
is authored by
LastName
WebAddress (URL)
has
has
Book (ISBN)
Email
Author (id)
has
has
PhoneNr
Title
has
has
"BookContact"
"Contract"
Amount
has
has main contact
has contract with
has
"BookPublisher"
Number
Publisher (pid)
Office
has
has
is published by
Chapter
Title
Year
has
has

Fill in your answer here

Format

Words: 0

Maximum marks: 15

2.1 Skranker og SQL

Given a database made with the following SQL-script:

```
CREATE TABLE species (
  sid int PRIMARY KEY,
  name text NOT NULL,
  type text CHECK (type = 'Pattedyr' OR type = 'Fisk' OR type = 'Fugl')
);

CREATE TABLE animal (
  aid int PRIMARY KEY,
  name text NOT NULL,
  mother_of int UNIQUE REFERENCES animal(aid),
  sid int REFERENCES species(sid)
);

INSERT INTO species VALUES
(0, 'Katt', 'Pattedyr'),
(1, 'Hund', 'Pattedyr'),
(2, 'Gris', 'Pattedyr')
(3, 'Spurv', 'Fugl');

INSERT INTO animal VALUES
(0, 'Doglas', NULL, 1),
(1, 'Mons', NULL, 0),
(2, 'Plutina', 0, 1),
(3, 'Princess', NULL, 2),
(4, 'Caterine', 1, 0);
```

For each of the SQL-commands below, decide if they are legal according to the database schema of the database above (i.e. the command succeeds without errors) or illegal according to the database schema above (i.e. fails and gives an error).

	Legal	Illegal
DELETE FROM animal WHERE did = 3;	○	○
UPDATE animal SET mother_of = 3 WHERE did = 0;	○	○
INSERT INTO species VALUES (4, 'Flue', 'Innsekt');	○	○
INSERT INTO animal VALUES (7, 'Mona', 1, 0);	○	○
INSERT INTO animal VALUES (6, 'Timmy', NULL, 2);	○	○
DELETE FROM animal WHERE name = 'Mons';	○	○
UPDATE animal SET mother_of = 4 WHERE mother_of IS NULL;	○	○
DROP TABLE species CASCADE;	○	○

Maximum marks: 10

2.2 Band etter 2000

In this (and the next) exercise you will use the following database schema:

Genre(genreID, name)
Band(bandID, name, started, genreID)
Person(personID, name, born)
Member(personID, bandID)
Album(albumID, bandID, name, released)
Song(songID, name, playtime, albumID)

with the following foreign keys:

Band(genreID) references Genre(genreID)
Member(personID) references Person(personID)
Member(bandID) references Band(bandID)
Album(bandID) references Band(bandID)
Song(albumID) references Album(albumID)

A genre consists of a unique genreID, a name (e.g. 'pop' or 'metal'); a band has a unique bandID, a name, a date stating when the band was started and a genreID pointing to the genre the band plays in; a person has a unique personID, a name and a date of birth; persons can be member of bands and this is described by the Member relation (note that a person can be member of many bands, and a band can have many members); an album consists of a unique albumID, a name, a bandID pointing to the band who made the album, and a date for when it was released; a song has a unique songID, a name, a playtime in seconds, and an albumID pointing to the album this song is part of.

For example, the database could contain the following data:

Genre

genreID	Name
0	Pop
1	Rock
2	Metal

Band

bandID	name	started	genreID
0	Blue Floyd	1985-01-19	1
1	Bettany Swords	1991-12-01	0
2	Shallow Violett	1989-04-23	1

Person

personID	name	born
0	Peter Smith	1963-02-09
1	Mary Green	1978-08-16
2	Bettany Evans	1981-09-09

Member

personID	bandID
0	0
1	2
2	1
1	0

Album

albumID	bandID	name	released
0	0	Surfin	1986-02-07
1	1	Love	1999-11-09
2	0	Fog on the grass	1990-03-13

Song

Song

songID	name	playtime	albumID
0	Board	231	0
1	Miss you	126	1
2	Sharks around	322	0
3	Fog on the grass	401	2
4	Biking in the sun	209	2

Exercise: Write an SQL-query that finds all bands that either was started after year 2000 or contains the string 'King' in its name. Output the name of the band and the year it was started.

Fill in your answer here

Maximum marks: 5

2.3 Timer Pop-musikk fra 90s

In this (and the next) exercise you will use the following database schema:

Genre(genreID, name)
Band(bandID, name, started, genreID)
Person(personID, name, born)
Member(personID, bandID)
Album(albumID, bandID, name, released)
Song(songID, name, playtime, albumID)

with the following foreign keys:

Band(genreID) references Genre(genreID)
Member(personID) references Person(personID)
Member(bandID) references Band(bandID)
Album(bandID) references Band(bandID)
Song(albumID) references Album(albumID)

A genre consists of a unique genreID, a name (e.g. 'pop' or 'metal'); a band has a unique bandID, a name, a date stating when the band was started and a genreID pointing to the genre the band plays in; a person has a unique personID, a name and a date of birth; persons can be member of bands and this is described by the

Member relation (note that a person can be member of many bands, and a band can have many members); an album consists of a unique albumID, a name, a bandID pointing to the band who made the album, and a date for when it was released; a song has a unique songID, a name, a playtime in seconds, and an albumID pointing to the album this song is part of.

For example, the database could contain the following data:

Genre

genreID	Name
0	Pop
1	Rock
2	Metal

Band

bandID	name	started	genreID
0	Blue Floyd	1985-01-19	1
1	Bettany Swords	1991-12-01	0
2	Shallow Violett	1989-04-23	1

Person

personID	name	born
0	Peter Smith	1963-02-09
1	Mary Green	1978-08-16
2	Bettany Evans	1981-09-09

Member

personID	bandID
0	0
1	2
2	1
1	0

Album

albumID	bandID	name	released
0	0	Surfin	1986-02-07
1	1	Love	1999-11-09
2	0	Fog on the grass	1990-03-13

Song

songID	name	playtime	albumID
0	Board	231	0
1	Miss you	126	1
2	Sharks around	322	0
3	Fog on the grass	401	2
4	Biking in the sun	209	2

Exercise: Write a query that finds the number of hours of music from the genre 'Pop' made by bands startet between the year 1990 and the year 2000. Note: One hour is 3600 seconds.

Fill in your answer here

Maximum marks: 5

2.4 Personer født på interessant dato

In this (and the next) exercise you will use the following database schema:

Genre(genreID, name)
Band(bandID, name, started, genreID)
Person(personID, name, born)
Member(personID, bandID)
Album(albumID, bandID, name, released)
Song(songID, name, playtime, albumID)

with the following foreign keys:

Band(genreID) references Genre(genreID)
Member(personID) references Person(personID)
Member(bandID) references Band(bandID)
Album(bandID) references Band(bandID)
Song(albumID) references Album(albumID)

A genre consists of a unique genreID, a name (e.g. 'pop' or 'metal'); a band has a unique bandID, a name, a date stating when the band was started and a genreID pointing to the genre the band plays in; a person has a unique personID, a name and a date of birth; persons can be member of bands and this is described by the Member relation (note that a person can be member of many bands, and a band can have many members); an album consists of a unique albumID, a name, a bandID pointing to the band who made the album, and a date for when it was released; a song has a unique songID, a name, a playtime in seconds, and an albumID pointing to the album this song is part of.

For example, the database could contain the following data:

Genre

genreID	Name
0	Pop
1	Rock
2	Metal

Band

bandID	name	started	genreID
0	Blue Floyd	1985-01-19	1
1	Bettany Swords	1991-12-01	0
2	Shallow Violett	1989-04-23	1

Person

personID	name	born
0	Peter Smith	1963-02-09
1	Mary Green	1978-08-16
2	Bettany Evans	1981-09-09

Member

personID	bandID
0	0
1	2
2	1
1	0

Album

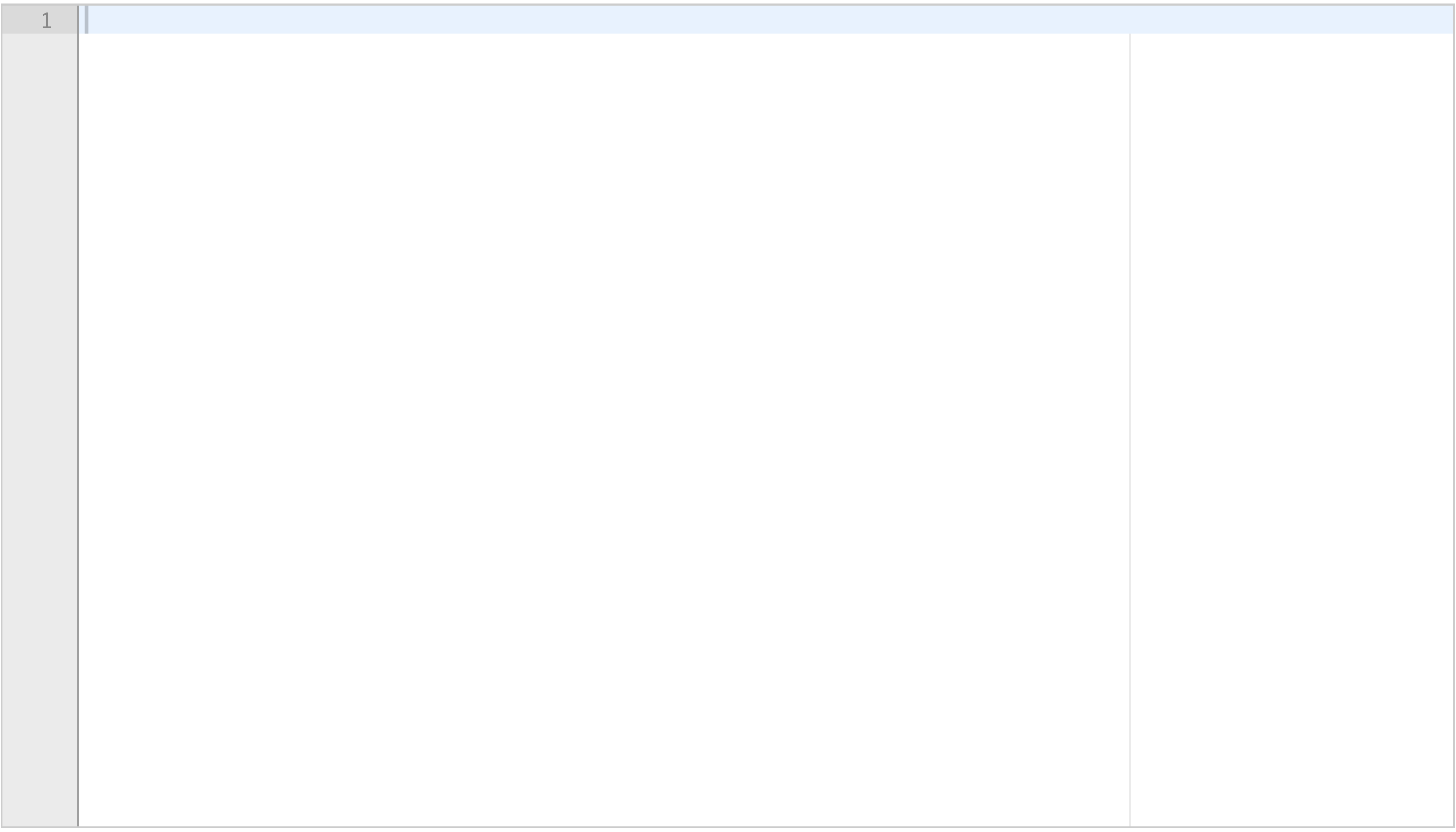
albumID	bandID	name	released
0	0	Surfin	1986-02-07
1	1	Love	1999-11-09
2	0	Fog on the grass	1990-03-13

Song

songID	name	playtime	albumID
0	Board	231	0
1	Miss you	126	1
2	Sharks around	322	0
3	Fog on the grass	401	2
4	Biking in the sun	209	2

Exercise: Write a query that finds the name of all persons born on a date for which either a band was started or an album was released.

Fill in your answer here



Maximum marks: 5

2.5 Sanger per band

In this (and the next) exercise you will use the following database schema:

Genre(genreID, name)
Band(bandID, name, started, genreID)
Person(personID, name, born)
Member(personID, bandID)
Album(albumID, bandID, name, released)
Song(songID, name, playtime, albumID)

with the following foreign keys:

Band(genreID) references Genre(genreID)
Member(personID) references Person(personID)
Member(bandID) references Band(bandID)
Album(bandID) references Band(bandID)
Song(albumID) references Album(albumID)

A genre consists of a unique genreID, a name (e.g. 'pop' or 'metal'); a band has a unique bandID, a name, a date stating when the band was started and a genreID pointing to the genre the band plays in; a person has a unique personID, a name and a date of birth; persons can be member of bands and this is described by the Member relation (note that a person can be member of many bands, and a band can have many members); an album consists of a unique albumID, a name, a bandID pointing to the band who made the album, and a date for when it was released; a song has a unique songID, a name, a playtime in seconds, and an albumID pointing to the album this song is part of.

For example, the database could contain the following data:

Genre

genreID	Name
0	Pop
1	Rock
2	Metal

Band

bandID	name	started	genreID
0	Blue Floyd	1985-01-19	1
1	Bettany Swords	1991-12-01	0
2	Shallow Violett	1989-04-23	1

Person

personID	name	born
0	Peter Smith	1963-02-09
1	Mary Green	1978-08-16
2	Bettany Evans	1981-09-09

Member

personID	bandID
0	0
1	2
2	1
1	0

Album

albumID	bandID	name	released
0	0	Surfin	1986-02-07
1	1	Love	1999-11-09
2	0	Fog on the grass	1990-03-13

Song

songID	name	playtime	albumID
0	Board	231	0
1	Miss you	126	1
2	Sharks around	322	0
3	Fog on the grass	401	2
4	Biking in the sun	209	2

Exercise: Write a query that finds the number of songs each band has made (i.e. the sum of the number of songs on all of their albums combined) for bands having made less than 3 songs. Write out the bandID, the name of the band and the number of songs the band has made.

Note: There might be bands that has not yet released any albums or albums containing no songs, these bands should also be included in the result with a number of songs equal to 0.

Fill in your answer here

Maximum marks: 5

2.6 Slett tomme album

In this (and the next) exercise you will use the following database schema:

Genre(genreID, name)
Band(bandID, name, started, genreID)
Person(personID, name, born)
Member(personID, bandID)
Album(albumID, bandID, name, released)
Song(songID, name, playtime, albumID)

with the following foreign keys:

Band(genreID) references Genre(genreID)
Member(personID) references Person(personID)
Member(bandID) references Band(bandID)
Album(bandID) references Band(bandID)
Song(albumID) references Album(albumID)

A genre consists of a unique genreID, a name (e.g. 'pop' or 'metal'); a band has a unique bandID, a name, a date stating when the band was started and a genreID pointing to the genre the band plays in; a person has a unique personID, a name and a date of birth; persons can be member of bands and this is described by the Member relation (note that a person can be member of many bands, and a band can have many members); an album consists of a unique albumID, a name, a bandID pointing to the band who made the album, and a date for when it was released; a song has a unique songID, a name, a playtime in seconds, and an albumID pointing to the album this song is part of.

For example, the database could contain the following data:

Genre

genreID	Name
0	Pop
1	Rock
2	Metal

Band

bandID	name	started	genreID
0	Blue Floyd	1985-01-19	1
1	Bettany Swords	1991-12-01	0
2	Shallow Violett	1989-04-23	1

Person

personID	name	born
0	Peter Smith	1963-02-09
1	Mary Green	1978-08-16
2	Bettany Evans	1981-09-09

Member

personID	bandID
0	0
1	2
2	1
1	0

Album

albumID	bandID	name	released
0	0	Surfin	1986-02-07
1	1	Love	1999-11-09
2	0	Fog on the grass	1990-03-13

Song

songID	name	playtime	albumID
0	Board	231	0
1	Miss you	126	1
2	Sharks around	322	0
3	Fog on the grass	401	2
4	Biking in the sun	209	2

Exercise: Write an SQL-command that deletes all albums not containing any songs.

Fill in your answer here

Maximum marks: 5

2.7 Nyeste album

In this (and the next) exercise you will use the following database schema:

Genre(genreID, name)
Band(bandID, name, started, genreID)
Person(personID, name, born)
Member(personID, bandID)
Album(albumID, bandID, name, released)
Song(songID, name, playtime, albumID)

with the following foreign keys:

Band(genreID) references Genre(genreID)
Member(personID) references Person(personID)
Member(bandID) references Band(bandID)
Album(bandID) references Band(bandID)
Song(albumID) references Album(albumID)

A genre consists of a unique genreID, a name (e.g. 'pop' or 'metal'); a band has a unique bandID, a name, a date stating when the band was started and a genreID pointing to the genre the band plays in; a person has a unique personID, a name and a date of birth; persons can be member of bands and this is described by the Member relation (note that a person can be member of many bands, and a band can have many members); an album consists of a unique albumID, a name, a bandID pointing to the band who made the album, and a date for when it was released; a song has a unique songID, a name, a playtime in seconds, and an albumID pointing to the album this song is part of.

For example, the database could contain the following data:

Genre

genreID	Name
0	Pop
1	Rock
2	Metal

Band

bandID	name	started	genreID
0	Blue Floyd	1985-01-19	1
1	Bettany Swords	1991-12-01	0
2	Shallow Violett	1989-04-23	1

Person

personID	name	born
0	Peter Smith	1963-02-09
1	Mary Green	1978-08-16
2	Bettany Evans	1981-09-09

Member

personID	bandID
0	0
1	2
2	1
1	0

Album

albumID	bandID	name	released
0	0	Surfin	1986-02-07
1	1	Love	1999-11-09
2	0	Fog on the grass	1990-03-13

Song

songID	name	playtime	albumID
0	Board	231	0
1	Miss you	126	1
2	Sharks around	322	0
3	Fog on the grass	401	2
4	Biking in the sun	209	2

Exercise: Write an SQL-command that makes a view with name "newest_albums" which contains the 10 newest albums. The view should for each album show its name, the name of the band that made the album, the date the album was released, and the total number of songs on the album. Order the view according to release date, with the newest first.

(Note: Here you can assume that every album contains at least one song.)

Fill in your answer here

Maximum marks: 5

2.8 Super-album

In this (and the next) exercise you will use the following database schema:

Genre(genreID, name)
Band(bandID, name, started, genreID)
Person(personID, name, born)
Member(personID, bandID)
Album(albumID, bandID, name, released)
Song(songID, name, playtime, albumID)

with the following foreign keys:

Band(genreID) references Genre(genreID)
Member(personID) references Person(personID)
Member(bandID) references Band(bandID)
Album(bandID) references Band(bandID)
Song(albumID) references Album(albumID)

A genre consists of a unique genreID, a name (e.g. 'pop' or 'metal'); a band has a unique bandID, a name, a date stating when the band was started and a genreID pointing to the genre the band plays in; a person has a unique personID, a name and a date of birth; persons can be member of bands and this is described by the Member relation (note that a person can be member of many bands, and a band can have many members); an album consists of a unique albumID, a name, a bandID pointing to the band who made the album, and a date for when it was released; a song has a unique songID, a name, a playtime in seconds, and an albumID pointing to the album this song is part of.

For example, the database could contain the following data:

Genre

genreID	Name
0	Pop
1	Rock
2	Metal

Band

bandID	name	started	genreID
0	Blue Floyd	1985-01-19	1
1	Bettany Swords	1991-12-01	0
2	Shallow Violett	1989-04-23	1

Person

personID	name	born
0	Peter Smith	1963-02-09
1	Mary Green	1978-08-16
2	Bettany Evans	1981-09-09

Member

personID	bandID
0	0
1	2
2	1
1	0

Album

albumID	bandID	name	released
0	0	Surfin	1986-02-07
1	1	Love	1999-11-09
2	0	Fog on the grass	1990-03-13

Song

songID	name	playtime	albumID
0	Board	231	0
1	Miss you	126	1
2	Sharks around	322	0
3	Fog on the grass	401	2
4	Biking in the sun	209	2

Exercise: Write a query that finds the name of all bands that has released a super-album. A super-album is an album that has a total playtime of more than an hour (3600 seconds). Output the bandID, the name of the band, and the number of super-albums the band has made.

Hint: It is a good idea to start by finding all "albumID"s for all super-albums in a separate query (e.g. by using WITH). Then one can find which bands made those, and then count.

Fill in your answer here

Maximum marks: 10

3.1 FDer

Given the following relation:

$$R(A, B, C, D, E, F, G)$$

and the following functional dependencies (FDs):

$$A \to C$$
$$C \to B, D$$
$$A, B \to D, F$$
$$B, E \to G$$

Check off each attribute that is contained in the closure of {A}.

□ F

□ E

□ B

□ A

□ G

□ C

□ D

Check off each attribute that is contained in the closure of {B}.

□ A

□ B

□ E

□ F

□ D

□ G

□ C

Check off the attributes that must be contained in every candidate key.

□ G

□ F

□ A

□ D

□ C

□ E

□ B

Maximum marks: 6

3.2 Normalformer

Given the following relation:

$$R(\underline{A, B}, C, D, E, F)$$

where the candidate key is underlined. (That is, we only have one candidate key, $\{A, B\}$)

For each of the FDs below, assume that the FD holds for the relation R above, and use the algorithm for normal forms to decide which normal form the FD (alone) implies that R is of.

	2NF	BCNF	1NF	3NF
B -> D	○	○	○	○
A -> C	○	○	○	○
A, D -> E	○	○	○	○
A, B -> F	○	○	○	○

Maximum marks: 4

Question 1

Attached

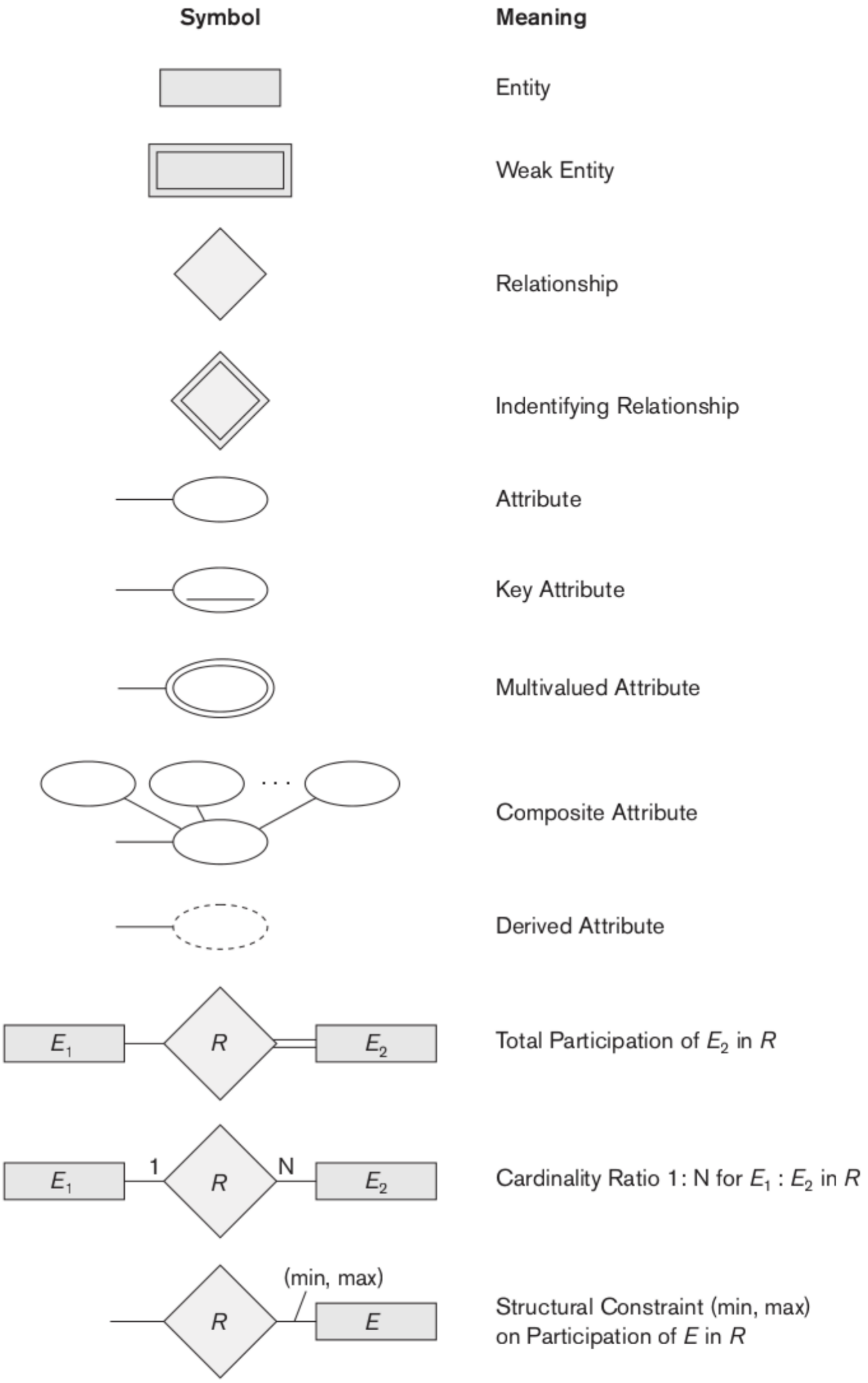
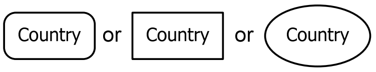
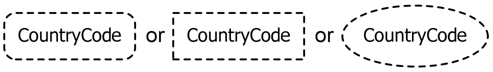
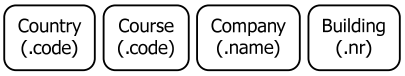
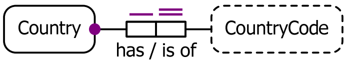
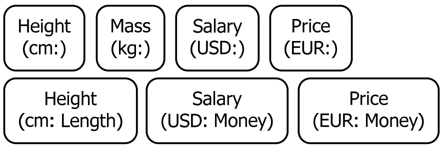
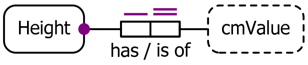
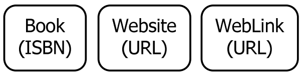
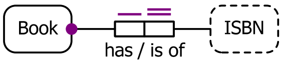
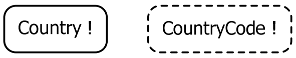
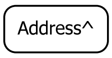
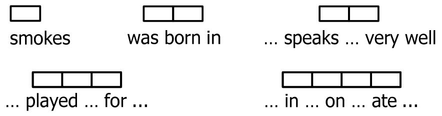
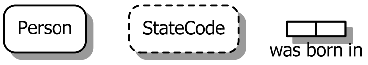
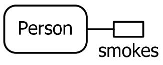
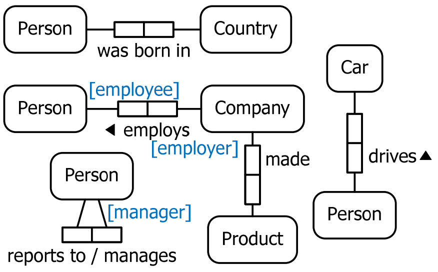
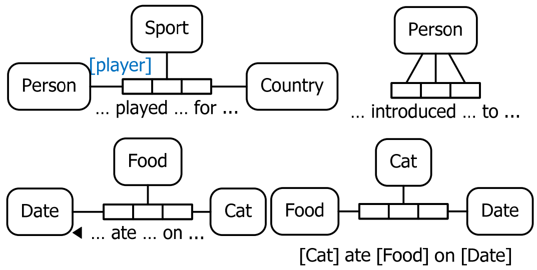
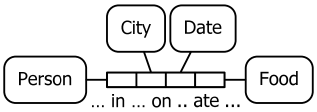
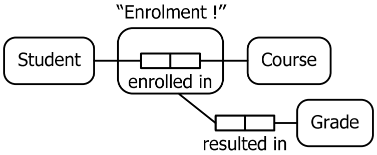
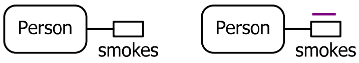
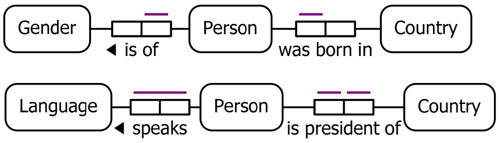
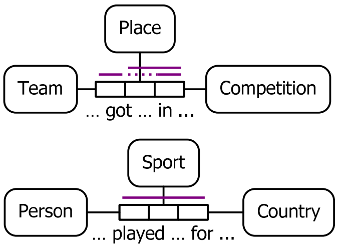
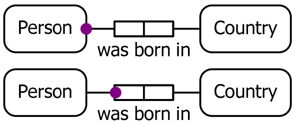
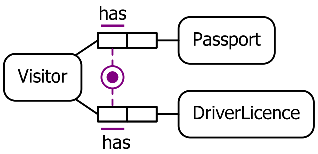
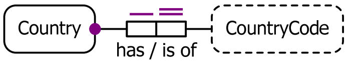

Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
. . .
Composite Attribute
Derived Attribute
E_1
R
E_2
Total Participation of E_2 in R
E_1
1
R
N
E_2
Cardinality Ratio 1: N for E_1 : E_2 in R
(min, max)
R
E
Structural Constraint (min, max) on Participation of E in R

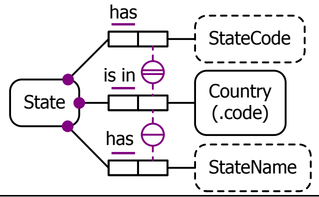
Figure 3.14
Summary of the notation for ER diagrams.

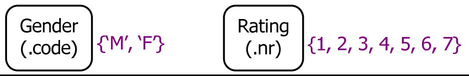
ORM 2 Graphical Notation

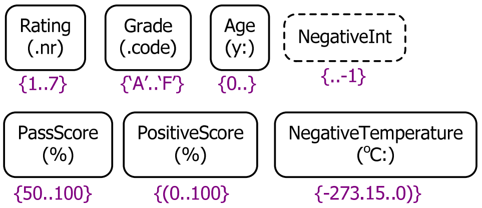
Terry Halpin

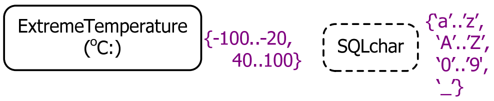
Construct	*Examples*	*Description/Notes*
Entity Type		Named soft rectangle, named hard rectangle, or named ellipse. The soft rectangle shape is the default.
Value Type		Named, dashed, soft rectangle (or hard rectangle or ellipse).
Entity type with popular reference mode	 Country (.code) Course (.code) Company (.name) Building (.nr)	Abbreviation for injective reference relationship to value type, e.g. 
Entity type with unit-based reference mode	 Height (cm:) Mass (kg:) Salary (USD:) Price (EUR:) Height (cm: Length) Salary (USD: Money) Price (EUR: Money)	Abbreviation for reference type, e.g.  Optionally, unit type may be displayed.
Entity type with general reference mode		Abbreviation for reference type, e.g. 
Independent Object Type		Instances of the type may exist, without playing any elementary fact roles
External Object Type		This notation is tentative (yet to be finalized)
Predicate (unary, binary, ternary, etc.)	 smokes was born in ... speaks ... very well ... played ... for ... ... in ... on ... ate ...	Ordered set of 1 or more role boxes with at least one predicate reading in mixfix notation. If shown, object placeholders are denoted by "...". If placeholders are not shown, unaries are in prefix and binaries are in infix notation.
Duplicate type or predicate shape		If an object type or predicate shape is displayed more than once (on the same page or different pages) it is shadowed.
Unary fact type		The smokes role may be played by instances of the Person object type
Binary fact type	 Person was born in Country Car Person [employee] Company ◄ employs [employer] made drives ▲ Person [manager] reports to / manages Product Person	By default, predicate readings (binary or longer) are read left-to-right or top-to-bottom. An arrow-tip is used to display a different reading direction. Role names may be displayed in square brackets beside their role. Forward and inverse readings for binaries may be shown together, separated by "/".

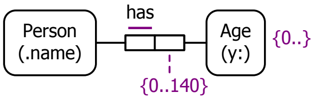
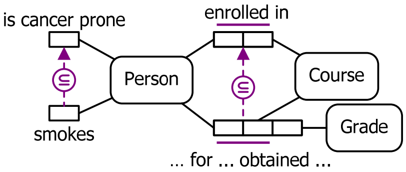
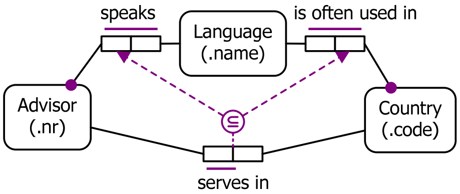
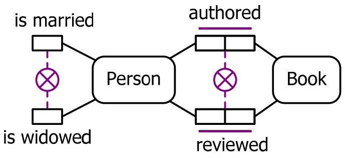
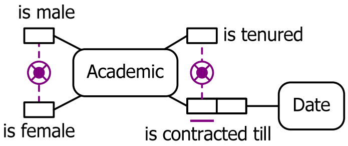
Construct	*Examples*	*Description/Notes*
Ternary fact type	 Sport [player] Person Country ... played ... for ... Person ... introduced ... to ... Food Date Cat ... ate ... on ... Cat Food Date [Cat] ate [Food] on [Date]	Role names may be added in square brackets. Arrow-tips are used to reverse the default left-right or top-down reading order. Reading orders other than forward and reverse are shown using named placeholders.
Quaternary fact type		The above notes for the ternary case apply here also. Fact types of higher arity (number of roles) are also permitted.
Objectification (a.k.a. nesting)	 "Enrolment !" Student enrolled in Course resulted in Grade	The enrolment fact type is objectified as an entity type whose instances can play roles. In this example, the objectification type is independent, so we can know about an enrolment before the grade is obtained.
Internal uniqueness constraint (UC) on unaries		These are equivalent (by default, predicates are assumed to be populated with sets, so no whole fact may be duplicated).
Internal UC on binaries	 Gender is of Person was born in Country Language speaks Person is president of Country	The examples show the 4 possible patterns: 1:*n* (one-to-many); *n*:1 (many-to-one); *m*:*n* (many-to-many); 1:1 (one-to-one0
Internal UC on ternaries. For *n*-aries (n > 1) each UC must span at least *n*-1 roles	 Place Team Competition ... got ... in ... Sport Person Country ... played ... for ...	The first example has two, 2-role UCs: the top UC forbids ties; the other UC ensures that each team gets only place per competition (a dotted line excludes its role from the UC). The second example has a spanning UC (many-to-many-to-many).
Simple mandatory role constraint	 Person Country was born in Person Country was born in	The example constraint means that each person was born in some country. The mandatory role dot may be placed at either end of the role connector.
Inclusive-or constraint (disjunctive mandatory role)		The constraint is displayed as a circled dot connected to the constrained roles. The example constraint means that each visitor referenced in the model must have a passport or a driver licence (or both).
Preferred internal UC		A double bar on a UC indicates it underlies the preferred reference scheme.

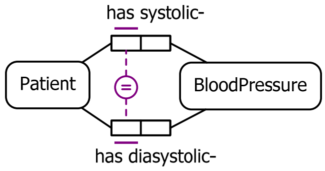
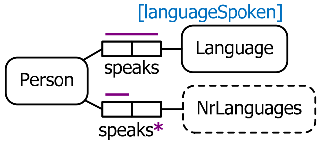
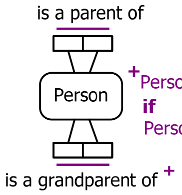
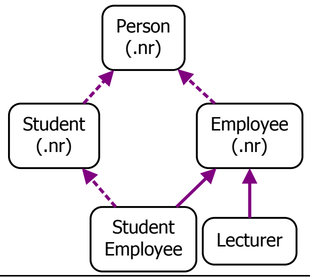
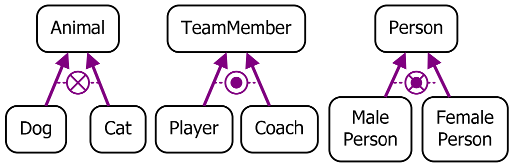
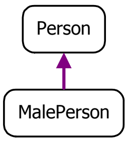
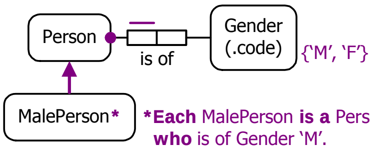
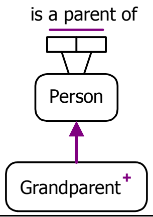
Construct	*Examples*	*Description/Notes*
External UC (double-bar indicates preferred identifier)	 has StateCode is in State Country (.code) has StateName	Here, each state is primarily identified by combining its country and state code. Each combination of country and state name also applies to only one state.

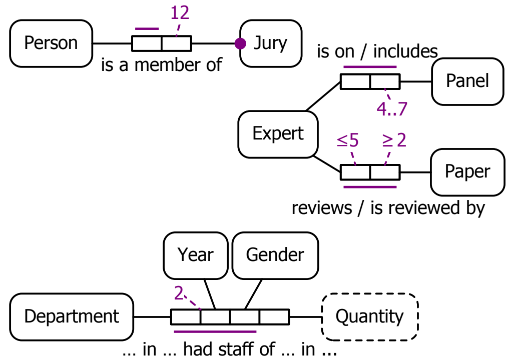
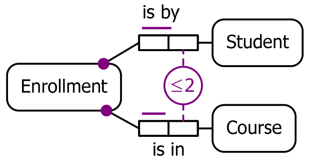
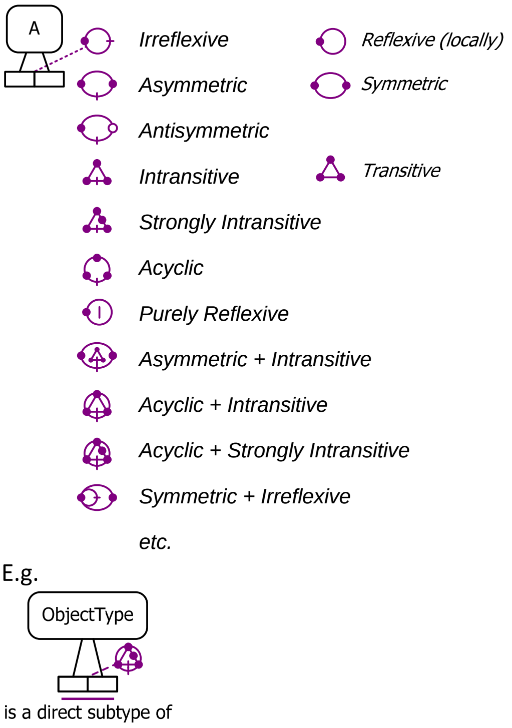
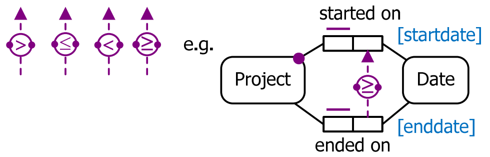
Object Type Value Constraint	 Gender (.code)
	{'M', 'F'}
	Rating (.nr)
	{1, 2, 3, 4, 5, 6, 7}
		Enumerations

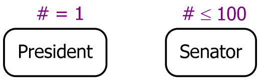
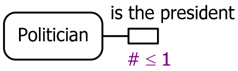
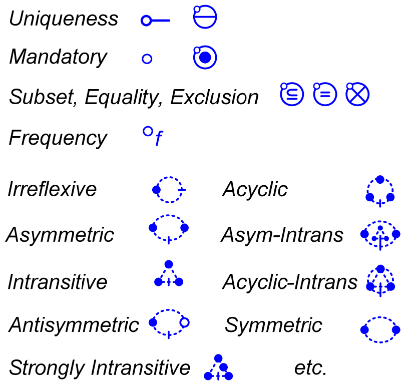
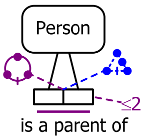
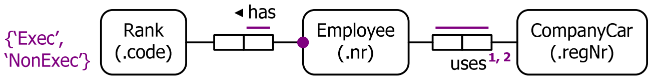
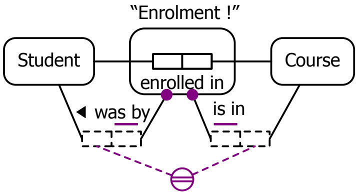
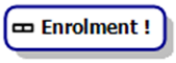
	 Rating (.nr)
	{1..7}
	Grade (.code)
	{'A'..'F'}
	Age (y:)
	{0..}
	NegativeInt
	{..-1}
	PassScore (%)
	{50..100}
	PositiveScore (%)
	{(0..100}
	NegativeTemperature (ºC:)
	{-273.15..0)}
		Ranges are inclusive of end values by default. Round brackets are used to exclude an end value. Square brackets may be added to explicitly declare inclusion, e.g. the constraint on PositiveScore may also be specified as {(0..100]}.

	 ExtremeTemperature (ºC:)
	{-100..-20, 40..100}
	SQLchar
	{'a'..'z', 'A'..'Z', '0'..'9', '_'}
		Multiple combinations are allowed.

Role value constraint		As for object type value constraints, but connected to the constrained role. Here, an age of a person must be at most 140 years.
Subset constraint	 is cancer prone smokes Person enrolled in Course Grade ... for ... obtained ...	The arrow points from the subset end to the superset end (e.g. if a person smokes then that person is cancer prone). The role sequences at both ends must be compatible. A connection to the junction of 2 roles constrains that role pair.
Join subset constraint	 speaks Language (.name) is often used in Advisor (.nr) Country (.code) serves in	The constrained role pair at the superset end is projected from a role path that involves a conceptual join on Language. The constraint declares that if an advisor serves in a country then that advisor must speak a language that is often used in that country.
Exclusion constraint	 is married is widowed Person authored reviewed Book	These constraints mean that no person is both married and widowed, and no person reviewed and authored the same book. Exclusion may apply between 2 or more compatible role sequences, possibly involving joins.
Exclusive-or constraint	 is male is female Academic is tenured is contracted till Date	An exclusive-or constraint is simply the conjunction of an inclusive-or constraint and an exclusion constraint. Also known as an xor constraint.

Construct	*Examples*	*Description/Notes*
Equality constraint	 has systolic- Patient = BloodPressure has diasystolic-	This constraint means that a patient's systolic BP is recorded if and only if his/her diastolic BP is recorded. An equality constraint may apply between 2 or more compatible role sequences, possibly involving joins.
Derived fact type, and derivation rule	 [languageSpoken] Person Language speaks NrLanguages speaks* ***For each** Person, nrLanguages = **count**(languageSpoken).	A fact type is either asserted, derived, or semiderived. A derived fact type is marked with an asterisk "*". A derivation rule is supplied. A double asterisk "**" indicates derived and stored (eager evaluation).
Semiderived fact type, and derivation rule	 +Person$_1$ is a grandparent of Person$_2$ **if** Person$_1$ is a parent of **some** Person$_3$ **who** is a parent of Person$_2$.	A fact type is semiderived if some of its instances may be derived, and some of its instances may be simply asserted. It is marked by "+" (half an asterisk). "++"indicates semiderived and stored (eager evaluation for derived instances).
Subtyping	 Person (.nr) Student (.nr) Employee (.nr) Student Employee Lecturer	All subtypes are proper subtypes. An arrow runs from subtype to supertype. A solid arrow indicates a path to the subtype's preferred identifier (e.g. here, student employees are primarily identified by their employee number). A dashed arrow indicates the supertype has a different preferred identifier.
Subtyping constraints	 Animal Dog Cat TeamMember Player Coach Person Male Person Female Person	A circled "X" indicates the subtypes are mutually exclusive. A circled dot indicates the supertype equals the union of the subtypes. The combination (xor constraint) indicates the subtypes partition the supertype (exclusive and exhaustive).
Subtype derivation status	  Person is of Gender (.code) {'M', 'F'} MalePerson* ***Each** MalePerson **is a** Person **who** is of Gender 'M'.  +**Each derived** Grandparent **is a** Person **who** is a parent of **some** Person **who** is a parent of **some** Person.	A subtype may be - asserted, - derived (denoted by "*"), - or semiderived (denoted by "+"). If the subtype is asserted, it has no mark appended and has no derivation rule. If the subtype derived or semiderived, a derivation rule is supplied.

Construct	*Examples*	*Description/Notes*
Internal frequency constraint	 Person 12 is a member of Jury is on / includes Panel 4..7 Expert ≤5 ≥2 Paper reviews / is reviewed by Year Gender 2 Department Quantity ... in ... had staff of ... in ...	This constrains the number of times an occurring instance of a role or role sequence may appear in each population. Here: each jury has exactly 12 members; each panel that includes an expert includes at least 4 and at most 7 experts; each expert reviews at most 5 papers; each paper that is reviewed is reviewed by at least 2 experts; and each department and year that has staff numbers recorded in the quaternary appears there twice (once for each gender).
External frequency constraint		The example constraint has the following meaning. In this context, each combination of student and course relates to at most two enrolments (i.e. a student may enroll at most twice in the same course)
Ring constraints	 A Irreflexive Reflexive (locally) Asymmetric Symmetric Antisymmetric Intransitive Transitive Strongly Intransitive Acyclic Purely Reflexive Asymmetric + Intransitive Acyclic + Intransitive Acyclic + Strongly Intransitive Symmetric + Irreflexive etc. E.g. ObjectType is a direct subtype of	A ring predicate *R* is locally reflexive if and only if, for all *x* and *y*, *xRy* implies *xRx*. E.g. "knows" is locally but not globally reflexive. Reflexive, symmetric and transitive properties may also be enforced using semiderivation rather than by constraining asserted fact types. The example constrains the subtyping relationship in ORM to be both acyclic (no cycles can be formed by a chain of subtyping connections) and strongly intransitive (no object type *A* can be both a direct subtype of another type *B* and an indirect subtype of *B*, where indirect subtyping means there is a chain of two or more subtyping relationships that lead from *A* to *B*). Ring constraints may be combined only if they are compatible, and one is not implied by the other. ORM tools ensure that only legal combinations are allowed.
Value-comparison constraints	 ≤ < ≥ e.g. started on [startdate] Project ≥ Date [enddate] ended on	The example constraint verbalizes as: **For each** Project, **existing** enddate >= startdate.

Construct	*Examples*	*Description/Notes*
Object cardinality constraint		The example constraints ensure there is exactly one president and at most 100 senators (at any given time),
Role cardinality constraint		The example constraint ensures that at most one politician plays the role of president (at any given time).
Deontic constraints	 *Uniqueness* *Mandatory* *Subset, Equality, Exclusion* *Frequency* o*f* *Irreflexive* *Acyclic* *Asymmetric* *Asym-Intrans* *Intransitive* *Acyclic-Intrans* *Antisymmetric* *Symmetric* *Strongly Intransitive* *etc.* e.g. 	Unlike alethic constraints, deontic constraint shapes are colored blue rather than violet. Most include "o" for "obligatory". Deontic ring constraints instead use dashed lines. In the parenthood example, the alethic frequency constraint ensures that each person has at most two parents, the alethic ring constraint ensures that parenthood is acyclic, and the deontic ring constraint makes it obligatory for parenthood to be strongly intransitive.
Textual constraints	 {'Exec', 'NonExec'} Rank (.code) ◂ has Employee (.nr) uses 1, 2 CompanyCar (.regNr) [1] **Each** Employee **who** has Rank 'NonExec' uses **at most one** CompanyCar. [2] **Each** Employee **who** has Rank 'Exec' uses **some** CompanyCar.	First-order constraints with no graphic notation may be expressed textually in the FORML 2 language. These examples use footnoting to capture a restricted uniqueness constraint and a restricted mandatory role constraint.
Objectification display options: link fact types, and compact display.	 "Enrolment !" Student enrolled in Course ◂ was by is in 	Internally, link fact types connect objectified associations to their component object types. By default, display of link fact types is suppressed. If displayed, link predicate shapes use dashed lines instead of solid lines. Objectification object types may also be displayed without their defining components, using an object type shape containing a small predicate shape, as shown in this Enrolment example.

Question 2

Attached

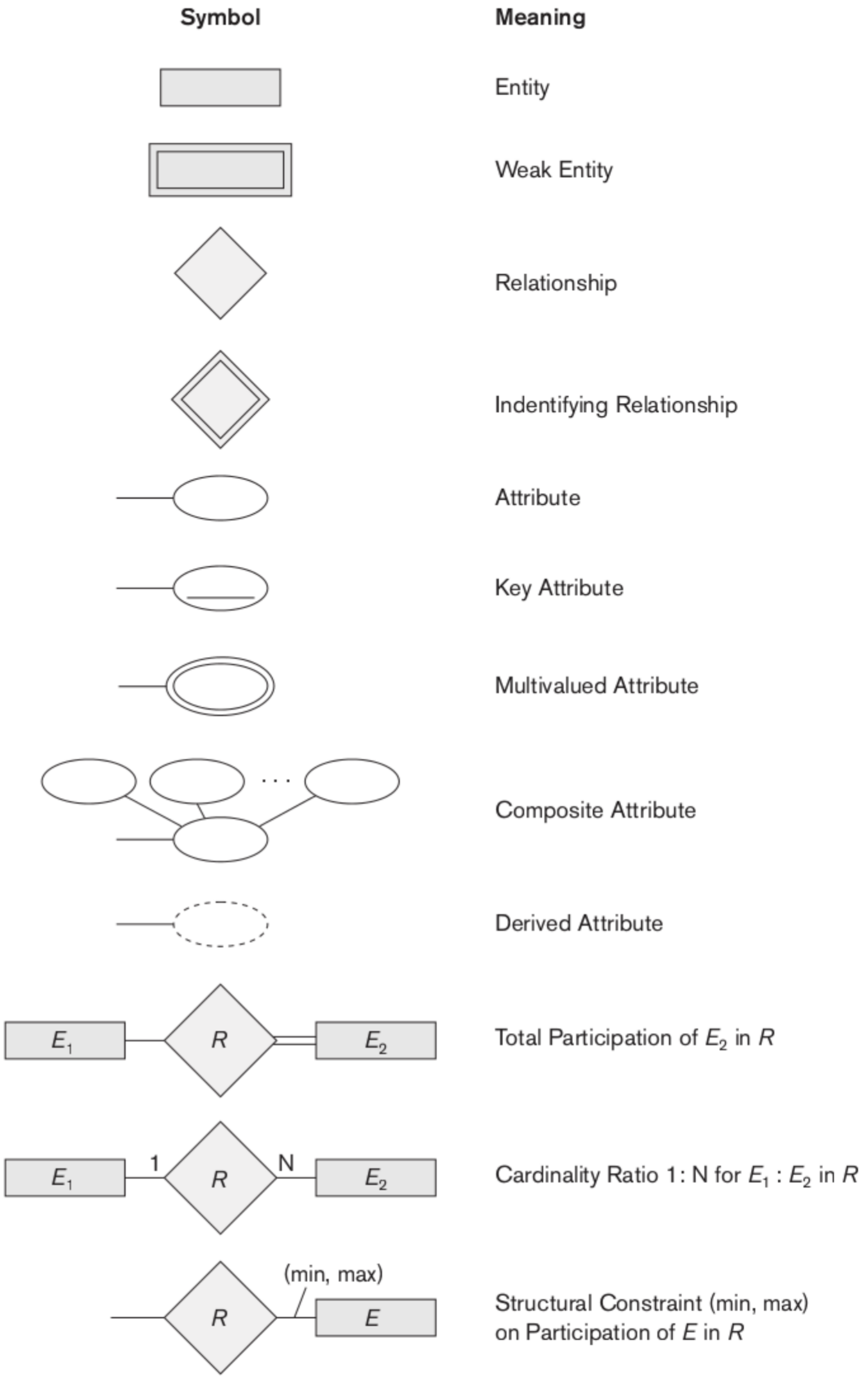
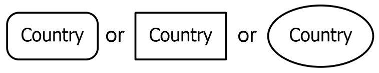
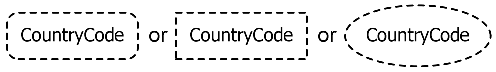
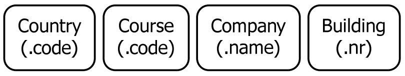
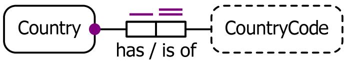
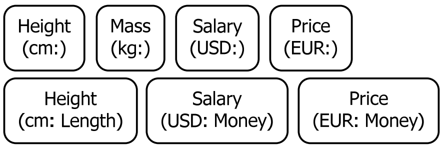
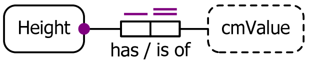
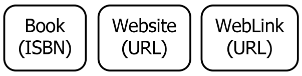
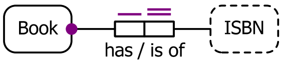
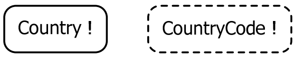
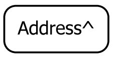
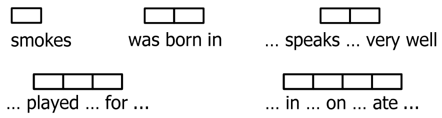
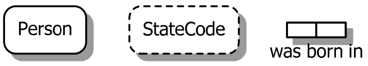
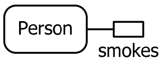
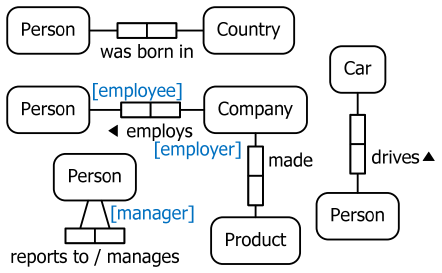
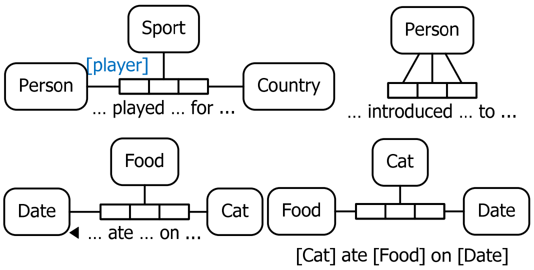
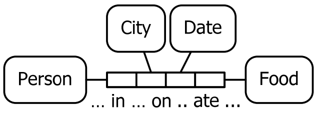
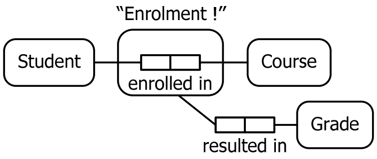
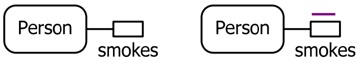
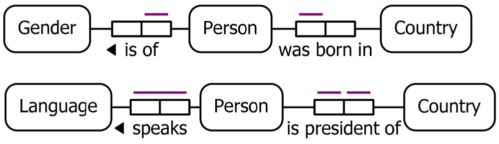
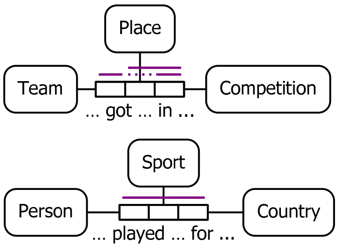
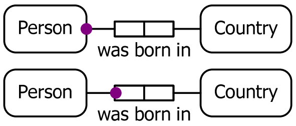
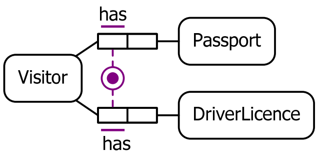
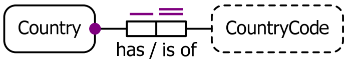

Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
. . .
Composite Attribute
Derived Attribute
E_1
R
E_2
Total Participation of E_2 in R
E_1
1
R
N
E_2
Cardinality Ratio 1: N for E_1 : E_2 in R
R
(min, max)
E
Structural Constraint (min, max) on Participation of E in R

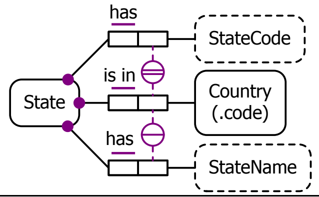
Figure 3.14
Summary of the notation for ER diagrams.

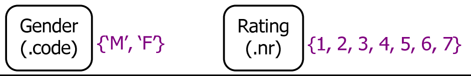
ORM 2 Graphical Notation

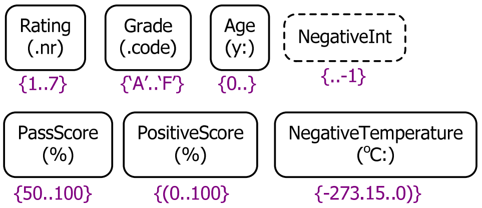
Terry Halpin

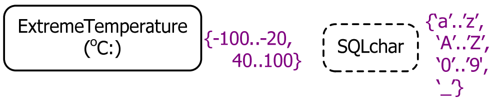
Construct	*Examples*	*Description/Notes*
Entity Type		Named soft rectangle, named hard rectangle, or named ellipse. The soft rectangle shape is the default.
Value Type		Named, dashed, soft rectangle (or hard rectangle or ellipse).
Entity type with popular reference mode	 Country (.code) Course (.code) Company (.name) Building (.nr)	Abbreviation for injective reference relationship to value type, e.g. 
Entity type with unit-based reference mode	 Height (cm:) Mass (kg:) Salary (USD:) Price (EUR:) Height (cm: Length) Salary (USD: Money) Price (EUR: Money)	Abbreviation for reference type, e.g.  Optionally, unit type may be displayed.
Entity type with general reference mode		Abbreviation for reference type, e.g. 
Independent Object Type		Instances of the type may exist, without playing any elementary fact roles
External Object Type		This notation is tentative (yet to be finalized)
Predicate (unary, binary, ternary, etc.)	 smokes was born in ... speaks ... very well ... played ... for ... ... in ... on ... ate ...	Ordered set of 1 or more role boxes with at least one predicate reading in mixfix notation. If shown, object placeholders are denoted by "…". If placeholders are not shown, unaries are in prefix and binaries are in infix notation.
Duplicate type or predicate shape		If an object type or predicate shape is displayed more than once (on the same page or different pages) it is shadowed.
Unary fact type		The smokes role may be played by instances of the Person object type
Binary fact type	 Person was born in Country Car Person [employee] Company ◂ employs [employer] made drives ▴ Person [manager] reports to / manages Product Person	By default, predicate readings (binary or longer) are read left-to-right or top-to-bottom. An arrow-tip is used to display a different reading direction. Role names may be displayed in square brackets beside their role. Forward and inverse readings for binaries may be shown together, separated by "/".

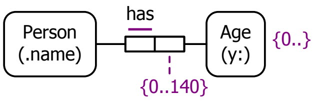
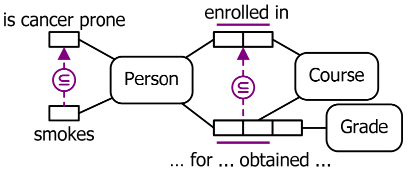
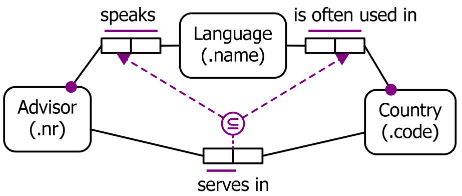
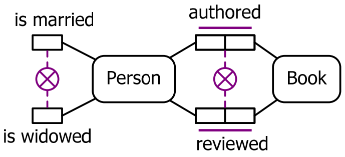
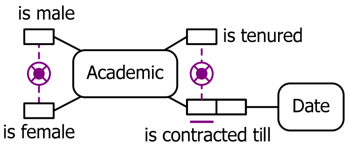
Construct	*Examples*	*Description/Notes*
Ternary fact type	 Sport [player] Person Country ... played ... for ... Person ... introduced ... to ... Food Date Cat ... ate ... on ... Cat Food Date [Cat] ate [Food] on [Date]	Role names may be added in square brackets. Arrow-tips are used to reverse the default left-right or top-down reading order. Reading orders other than forward and reverse are shown using named placeholders.
Quaternary fact type		The above notes for the ternary case apply here also. Fact types of higher arity (number of roles) are also permitted.
Objectification (a.k.a. nesting)	 "Enrolment !" Student enrolled in Course resulted in Grade	The enrolment fact type is objectified as an entity type whose instances can play roles. In this example, the objectification type is independent, so we can know about an enrolment before the grade is obtained.
Internal uniqueness constraint (UC) on unaries		These are equivalent (by default, predicates are assumed to be populated with sets, so no whole fact may be duplicated).
Internal UC on binaries	 Gender is of Person was born in Country Language speaks Person is president of Country	The examples show the 4 possible patterns: 1:*n* (one-to-many); *n*:1 (many-to-one); *m*:*n* (many-to-many); 1:1 (one-to-one0
Internal UC on ternaries. For *n*-aries (n > 1) each UC must span at least *n*-1 roles	 Place Team Competition ... got ... in ... Sport Person Country ... played ... for ...	The first example has two, 2-role UCs: the top UC forbids ties; the other UC ensures that each team gets only place per competition (a dotted line excludes its role from the UC). The second example has a spanning UC (many-to-many-to-many).
Simple mandatory role constraint	 Person was born in Country Person was born in Country	The example constraint means that each person was born in some country. The mandatory role dot may be placed at either end of the role connector.
Inclusive-or constraint (disjunctive mandatory role)		The constraint is displayed as a circled dot connected to the constrained roles. The example constraint means that each visitor referenced in the model must have a passport or a driver licence (or both).
Preferred internal UC		A double bar on a UC indicates it underlies the preferred reference scheme.

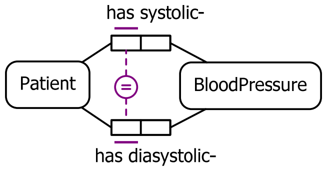
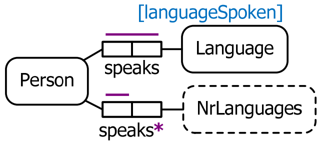
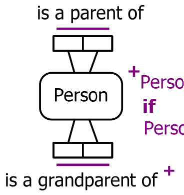
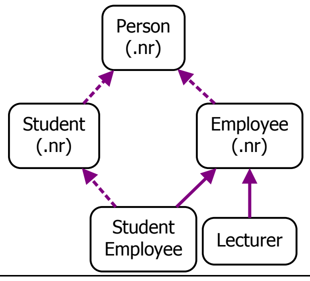
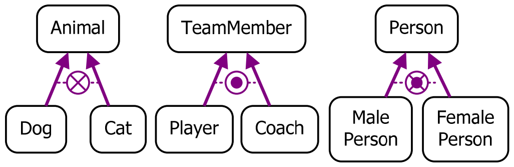
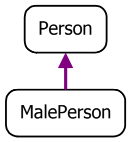
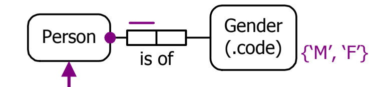
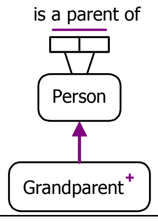
Construct	*Examples*	*Description/Notes*
External UC (double-bar indicates preferred identifier)	 has StateCode is in State Country (.code) has StateName	Here, each state is primarily identified by combining its country and state code. Each combination of country and state name also applies to only one state.

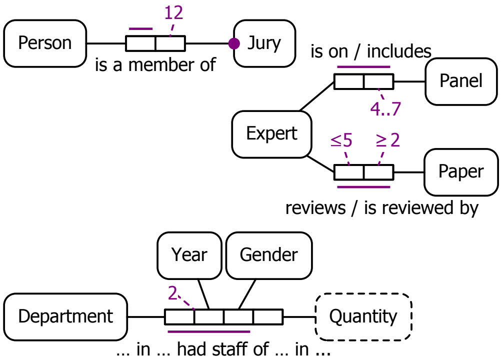
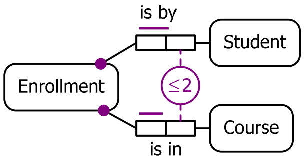
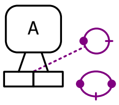
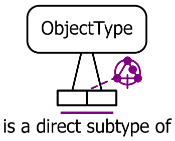
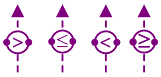
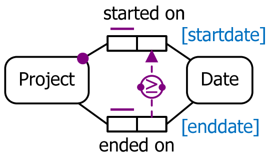
Object Type Value Constraint	 Gender (.code)
	{'M', 'F'}
	Rating (.nr)
	{1, 2, 3, 4, 5, 6, 7}
		Enumerations

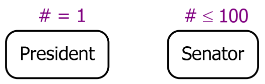
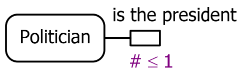
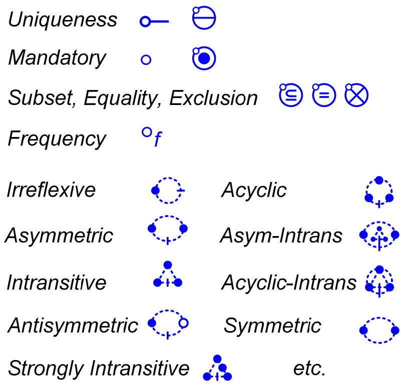
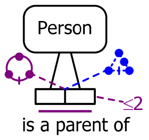
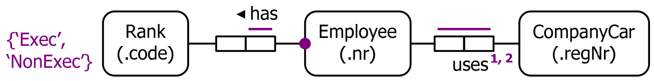
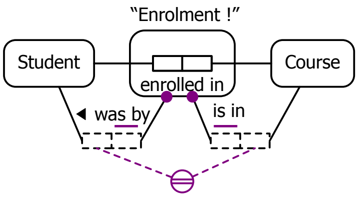
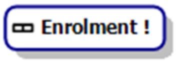
	 Rating (.nr)
	{1..7}
	Grade (.code)
	{'A'..'F'}
	Age (y:)
	{0..}
	NegativeInt
	{..-1}
	PassScore (%)
	{50..100}
	PositiveScore (%)
	{(0..100}
	NegativeTemperature (ºC:)
	{-273.15..0)}
		Ranges are inclusive of end values by default. Round brackets are used to exclude an end value. Square brackets may be added to explicitly declare inclusion, e.g. the constraint on PositiveScore may also be specified as {(0..100]}.

	 ExtremeTemperature (ºC:)
	{-100..-20, 40..100}
	SQLchar
	{'a'..'z', 'A'..'Z', '0'..'9', '_'}
		Multiple combinations are allowed.

Role value constraint		As for object type value constraints, but connected to the constrained role. Here, an age of a person must be at most 140 years.
Subset constraint	 is cancer prone enrolled in Person Course Grade smokes ... for ... obtained ...	The arrow points from the subset end to the superset end (e.g. if a person smokes then that person is cancer prone). The role sequences at both ends must be compatible. A connection to the junction of 2 roles constrains that role pair.
Join subset constraint	 speaks Language (.name) is often used in Advisor (.nr) Country (.code) serves in	The constrained role pair at the superset end is projected from a role path that involves a conceptual join on Language. The constraint declares that if an advisor serves in a country then that advisor must speak a language that is often used in that country.
Exclusion constraint	 is married authored Person Book is widowed reviewed	These constraints mean that no person is both married and widowed, and no person reviewed and authored the same book. Exclusion may apply between 2 or more compatible role sequences, possibly involving joins.
Exclusive-or constraint	 is male is tenured Academic Date is female is contracted till	An exclusive-or constraint is simply the conjunction of an inclusive-or constraint and an exclusion constraint. Also known as an xor constraint.

Construct	*Examples*	*Description/Notes*
Equality constraint	 has systolic- Patient = BloodPressure has diasystolic-	This constraint means that a patient’s systolic BP is recorded if and only if his/her diastolic BP is recorded. An equality constraint may apply between 2 or more compatible role sequences, possibly involving joins.
Derived fact type, and derivation rule	 [languageSpoken] Person speaks Language speaks* NrLanguages ***For each** Person, nrLanguages = **count**(languageSpoken).	A fact type is either asserted, derived, or semiderived. A derived fact type is marked with an asterisk “*”. A derivation rule is supplied. A double asterisk “**” indicates derived and stored (eager evaluation).
Semiderived fact type, and derivation rule	 +Person$_1$ is a grandparent of Person$_2$ **if** Person$_1$ is a parent of **some** Person$_3$ **who** is a parent of Person$_2$.	A fact type is semiderived if some of its instances may be derived, and some of its instances may be simply asserted. It is marked by “$^{+}$” (half an asterisk). “$^{++}$”indicates semiderived and stored (eager evaluation for derived instances).
Subtyping	 Person (.nr) Student (.nr) Employee (.nr) Student Employee Lecturer	All subtypes are proper subtypes. An arrow runs from subtype to supertype. A solid arrow indicates a path to the subtype’s preferred identifier (e.g. here, student employees are primarily identified by their employee number). A dashed arrow indicates the supertype has a different preferred identifier.
Subtyping constraints	 Animal Dog Cat TeamMember Player Coach Person Male Person Female Person	A circled “X” indicates the subtypes are mutually exclusive. A circled dot indicates the supertype equals the union of the subtypes. The combination (xor constraint) indicates the subtypes partition the supertype (exclusive and exhaustive).
Subtype derivation status	  Person is of Gender (.code) {‘M’, ‘F’} MalePerson* ***Each** MalePerson **is a** Person **who** is of Gender ‘M’.  +**Each derived** Grandparent **is a** Person **who** is a parent of **some** Person **who** is a parent of **some** Person.	A subtype may be • asserted, • derived (denoted by “*”), • or semiderived (denoted by “$^{+}$”). If the subtype is asserted, it has no mark appended and has no derivation rule. If the subtype derived or semiderived, a derivation rule is supplied.

Construct	*Examples*	*Description/Notes*
Internal frequency constraint	 Person 12 is a member of Jury is on / includes Panel 4..7 Expert ≤5 ≥2 Paper reviews / is reviewed by Year Gender 2 Department Quantity … in … had staff of … in …	This constrains the number of times an occurring instance of a role or role sequence may appear in each population. Here: each jury has exactly 12 members; each panel that includes an expert includes at least 4 and at most 7 experts; each expert reviews at most 5 papers; each paper that is reviewed is reviewed by at least 2 experts; and each department and year that has staff numbers recorded in the quaternary appears there twice (once for each gender).
External frequency constraint		The example constraint has the following meaning. In this context, each combination of student and course relates to at most two enrolments (i.e. a student may enroll at most twice in the same course)
Ring constraints	 *Irreflexive* *Reflexive (locally)* *Asymmetric* *Symmetric* *Antisymmetric* *Intransitive* *Transitive* *Strongly Intransitive* *Acyclic* *Purely Reflexive* *Asymmetric + Intransitive* *Acyclic + Intransitive* *Acyclic + Strongly Intransitive* *Symmetric + Irreflexive* *etc.* E.g. 	A ring predicate *R* is locally reflexive if and only if, for all *x* and *y*, *xRy* implies *xRx*. E.g. “knows” is locally but not globally reflexive. Reflexive, symmetric and transitive properties may also be enforced using semiderivation rather than by constraining asserted fact types. The example constrains the subtyping relationship in ORM to be both acyclic (no cycles can be formed by a chain of subtyping connections) and strongly intransitive (no object type *A* can be both a direct subtype of another type *B* and an indirect subtype of *B*, where indirect subtyping means there is a chain of two or more subtyping relationships that lead from *A* to *B*). Ring constraints may be combined only if they are compatible, and one is not implied by the other. ORM tools ensure that only legal combinations are allowed.
Value-comparison constraints	 e.g.  started on [startdate] Project ≥ Date [enddate] ended on	The example constraint verbalizes as: **For each** Project, **existing** enddate >= startdate.

Construct	*Examples*	*Description/Notes*
Object cardinality constraint	 	The example constraints ensure there is exactly one president and at most 100 senators (at any given time),
Role cardinality constraint	 	The example constraint ensures that at most one politician plays the role of president (at any given time).
Deontic constraints	 *Uniqueness* *Mandatory* *Subset, Equality, Exclusion* *Frequency* *f* *Irreflexive* *Acyclic* *Asymmetric* *Asym-Intrans* *Intransitive* *Acyclic-Intrans* *Antisymmetric* *Symmetric* *Strongly Intransitive* *etc.* e.g.  	Unlike alethic constraints, deontic constraint shapes are colored blue rather than violet. Most include “o” for “obligatory”. Deontic ring constraints instead use dashed lines. In the parenthood example, the alethic frequency constraint ensures that each person has at most two parents, the alethic ring constraint ensures that parenthood is acyclic, and the deontic ring constraint makes it obligatory for parenthood to be strongly intransitive.
Textual constraints	 {‘Exec’, ‘NonExec’} Rank (.code) ◂ has Employee (.nr) uses 1, 2 CompanyCar (.regNr) [1] **Each** Employee **who** has Rank ‘NonExec’ uses **at most one** CompanyCar. [2] **Each** Employee **who** has Rank ‘Exec’ uses **some** CompanyCar.	First-order constraints with no graphic notation may be expressed textually in the FORML 2 language. These examples use footnoting to capture a restricted uniqueness constraint and a restricted mandatory role constraint.
Objectification display options: link fact types, and compact display.	 “Enrolment !” Student enrolled in Course ◂ was by is in  	Internally, link fact types connect objectified associations to their component object types. By default, display of link fact types is suppressed. If displayed, link predicate shapes use dashed lines instead of solid lines. Objectification object types may also be displayed without their defining components, using an object type shape containing a small predicate shape, as shown in this Enrolment example.