Del A: Modellering og realisering

Exam in IN2090 - Databases and data modelling and INF1300 - Introduction to databases

6. December 2018 14:30 – 18:30 (4 hours)

Support material:

- Halpin & Morgan: Information Modelling and Relational Databases. Second Edition.
- 4 handwritten A4 pages of notes (2 sheets if there is writing on both sides).

A calculator is not allowed.

The exam is in two parts, part A (ORM modelling and relational mapping) and part B (SQL). Each part counts for 50% of the grade.

One of the lecturers will come to the exam room about 30 minutes to one hour after the exam starts. It is important that you have read through the whole set of questions by then, as you may ask the lecturer for clarifications if needed.

You should read through all of part A before you start modelling, and you should read through all of part B before solving any of the questions.

Feel free to include comments in the model. If you feel that something in the question text is unclear, you may leave comments to explain what assumptions you have made.

The answer to question 2 (modelling) must be drawn on paper (drawing paper) that you will be given.
Instructions on how to fill out the drawing paper are on your desk. Remember to write down the code and other information immediately; as you will not be given time to do this when the exam is over. No extra time will be given to fill out the headers on the drawing papers (codes, candidate numbers etc.). Please note that this question is marked as "oral" on the question list due to technical reasons.

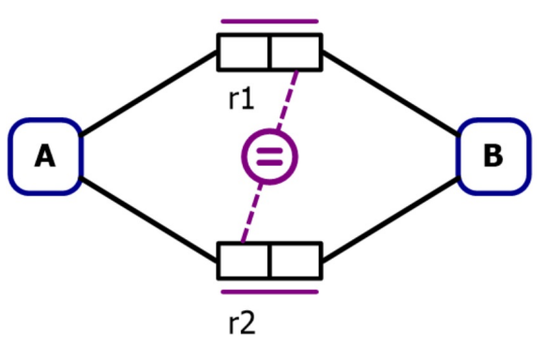
If you need to draw anything else on paper (for questions other than the modelling question), e.g. to show your thinking, please draw this after the answer to question 2, noting clearly on the drawing which question it belongs to. Then hand the paper in for question 2.

In principle, only question 2 requires the use of drawing paper. Draw clearly and understandably.

1 Eksterne skranker (5%)

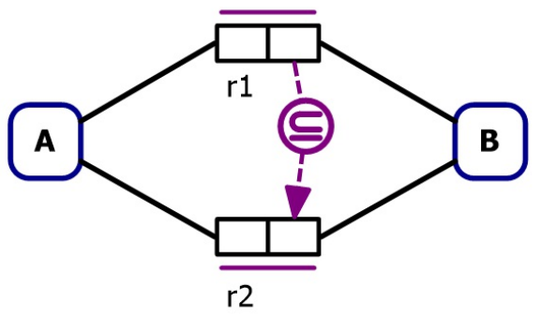
*The following give you **1** point for each correct answer. **-1** point for each wrong answer. **0** points if you do not answer.*

In the models below (ORM2) you are to assume that all entities have a unique representation.



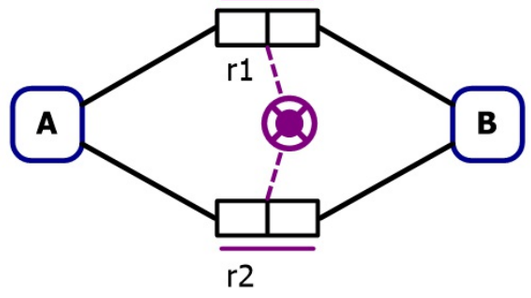
Is the placement of the external constraint shown above valid or invalid?

- Valid
- Invalid



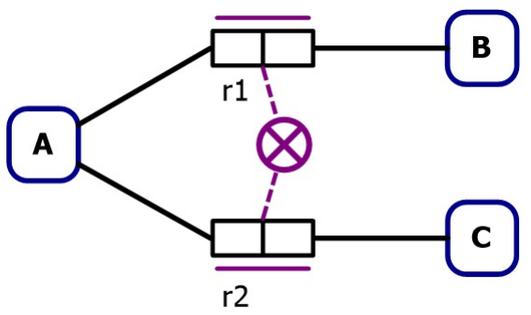
Is the placement of the external constraint shown above valid or invalid?

- Valid
- Invalid



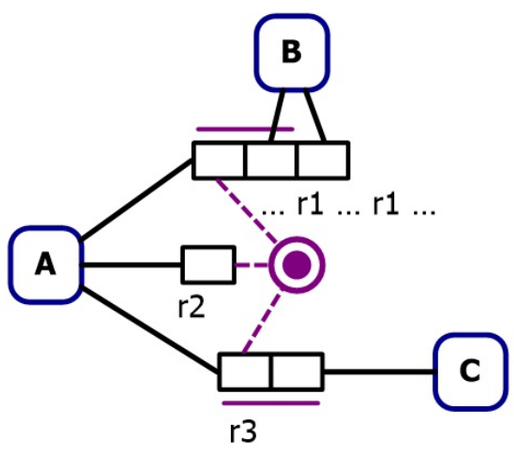
Is the placement of the external constraint shown above valid or invalid?

- Valid
- Invalid



Is the placement of the external constraint shown above valid or invalid?

- Valid
- Invalid



Is the placement of the external constraint shown above valid or invalid?

- Valid
- Invalid

Maximum marks: 5

2 Modellering i ORM (35%)

This assignment must be solved on drawing paper that you have been given. The instructions for filling out such paper are on your desk. The reference document ORM2 Graphical Notation is

attached.

In this problem you will model a course registration system for a university. The system will also can handle deliveries for assignments and exams. The model must satisfy the requirements listed below. To make the model clearer, you may split the ORM model across multiple pages. It's best to use one page for each of the two parts, but you may also draw the whole model as one coherent model.

Part 1: *Make an ORM2 model that contains the following information about users and courses:*

a. Each user is identified by a unique number (e.g. 483226). Additionally, each user must have a username (e.g. user 483226 has the username «olanor»). No users may share a username. We also want to register a full name for the users, but this may be missing for some users. Multiple users may have the same name.
b. Users can be registered for multiple courses. Courses are represented by a course code. Additionally, all courses must have a course name, and multiple courses may share the same name. There is no upper limit on how many users can be registered in a course.
c. Some users are tutors for courses, but users cannot be tutors for courses they are registered for. Users can be tutors for multiple courses, and each course may have many tutors.
d. We also want to register results from previous semesters, so that each user has one result per course in a given semester. For example, user 483226 may have received the grade "C" in the course "IN1010" in the semester "spring 2017". It must be possible to get a result for the same course in multiple semesters, e.g. user 483226 may also get the grade "B" in the course IN1010 spring 2018. A user may also get the same grade in several semesters, but a user may not get multiple results in one course in the same semester. Semesters are represented with a semester code (i.e. "spring 2018"). The result may be "passed", "failed", or a letter between "A" and "F".
e. All users who are tutors for a course, must have a result registered in that course.

Part 2: *Make an ORM2 model that contains additional information about assignments, deliveries and courses. In this part, we will not consider semesters or results (from part 1).*

a. In this system, we would like to add mandatory assignments ("obliger"). All assignments belong to one course, and all assignments have an assignment number. The assignment number may not be reused multiple times in a course, but the number can be used for different courses (e.g. IN1010 may only have one assignment 1, but IN2090 may also have an assignment 1).
b. Users can make deliveries. All deliveries belong to one or several users (i.e. they can make deliveries as groups). Deliveries can belong to one assignment, or it can be an exam delivery, in which case it belongs to a course (from part 1). All deliveries must either belong to an assignment, or be an exam delivery in a course, but cannot be both. Deliveries also need a suitable unique representation in the model.
c. Some courses can overlap with other courses with a certain number of credits. Each course may overlap with multiple courses, but a course cannot overlap with itself. The overlap goes both ways, i.e. if IN2090 overlaps by 10 credits with INF1300, then INF1300 also has overlaps by 10 credits with IN2090.

Maximum marks: 35

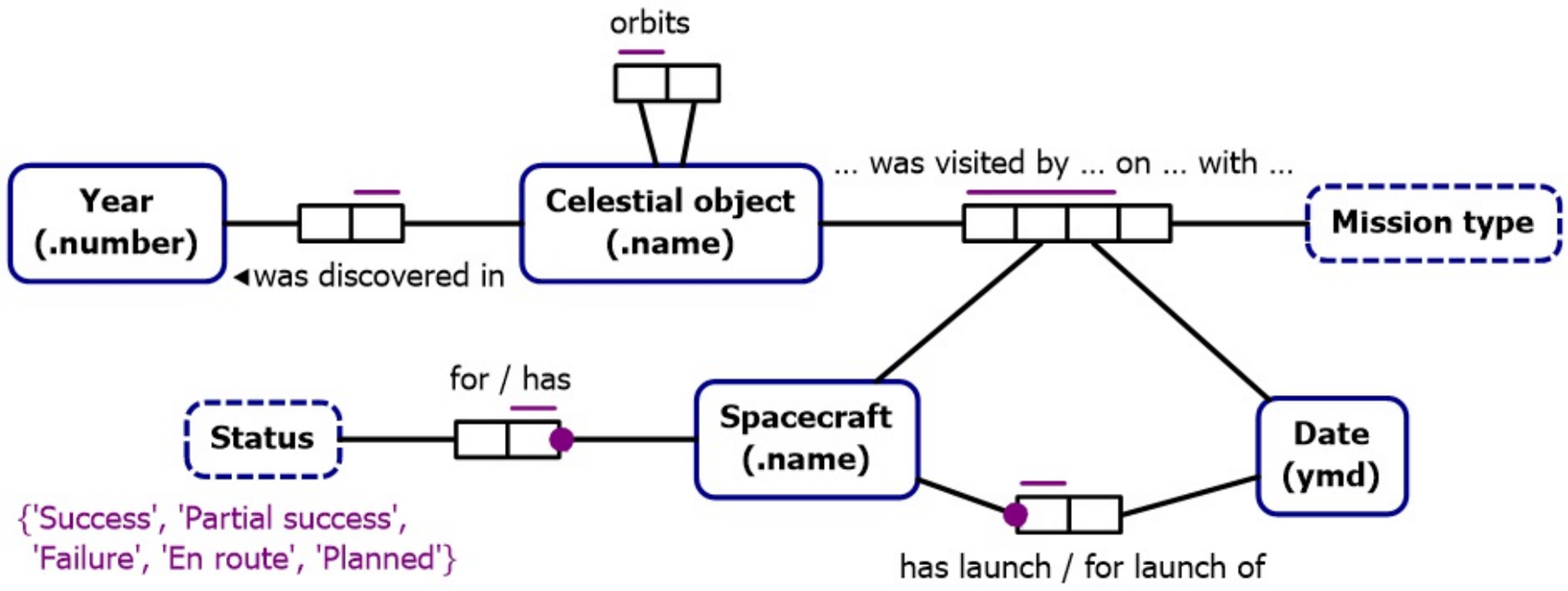
3 Realisering (10%)

Transform the following ORM2 model into a relational database schema.

The database schema must be correct (correspond to the ORM2 model), effective (avoid redundancy and limit the number of tables), and readable (easy to understand). Use the algorithm you have been taught to create such a schema.

For each relation, specify its name and the names of every attribute. You should not specify the data types/domains for the attributes, and no SQL should be used in this problem. Mark primary keys as underlined text. Mark other candidate keys with a **bold font**. Write down the foreign keys using the following form:

FromRelation(fromAttribute) → ToRelation(ToAttribute)



Fill in your answer here

Maximum marks: 10

Del B: SQL og relasjonsteori (50%)

In question 4-11 you will work with the tables described below. The description is copied on the page for each question. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)
orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products,* id is an int, and the other attributes are strings to store a name and description, respectively.

The table *customers* stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

4 SQL, create (5%)

In this part of the exam, you will work with the following schema for a small order database. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)
orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products*, id is an int, and the other attributes are strings to store a name and description, respectively.

The table *customers* stores information about customers. id and pnr (personal number) has the type

int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

Problem: Write a CREATE statement for the table *orders* with appropriate data types for the attributes, so that the created table is as described above, including primary and foreign keys.

Fill in your answer here

Maximum marks: 5

5 SQL, view (5%)

In this part of the exam, you will work with the following schema for a small order database. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)
orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products*, id is an int, and the other attributes are strings to store a name and description, respectively.

The table *customers* stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

Problem: Define a view delayed_orders(id, customer_id, time_entered) that contains all orders with the status 'delayed'.

Fill in your answer here

Maximum marks: 5

6 SQL, kunder og ordrer (5%)

In this part of the exam, you will work with the following schema for a small order database. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)

orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products*, id is an int, and the other attributes are strings to store a name and description, respectively.

The table *customers* stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

Problem: Write a query that returns all customers whose name begins with "Ola", together with their orders if they have any. Include customers without any orders. The result should contain the customer's name and address, as well as the order's id and status, sorted by registration time so that the most recently registered customer appear first.

Fill in your answer here

Maximum marks: 5

7 SQL, aggregering (5%)

In this part of the exam, you will work with the following schema for a small order database. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)
orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products*, id is an int, and the other attributes are strings to store a name and description, respectively.

The table customers stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

Problem: Write a query that finds the number of orders per customer for each different status. Print the customer id and name, as well as the number and status.

Fill in your answer here

Maximum marks: 5

8 SQL, kunder med flere ordrer (5%)

In this part of the exam, you will work with the following schema for a small order database. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)
orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products*, id is an int, and the other attributes are strings to store a name and description, respectively.

The table *customers* stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

Problem: Write a query that finds all customers who have at least two orders with different shipping_address. Print the customer id and navn.

Fill in your answer here

Maximum marks: 5

9 SQL, produkter bestilt oftest (10%)

In this part of the exam, you will work with the following schema for a small order database. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)
orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products*, id is an int, and the other attributes are strings to store a name and description, respectively.

The table *customers* stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

Problem: Write a query that returns all customers who have ordered the products that were ordered most often (i.e. the products found on the highest number of different orders). Print the customer ids.

Keep in mind that several products may have been ordered most often. E.g. if product A and product B both have been ordered 500 times and thus occur on the highest number of orders, all customers who have ordered either product A or product B should be printed.

Fill in your answer here

Maximum marks: 10

10 SQL, total kostnad (10%)

In this part of the exam, you will work with the following schema for a small order database. Primary keys are underlined.

products(id, name, description)
customers(id, pnr, fdate, name, address, country, time_registered)
orders(id, customer_id, shipping_addr, time_entered, status)
order_items(order_id, product_id, quantity, unit_price)

In the table *products*, id is an int, and the other attributes are strings to store a name and description, respectively.

The table *customers* stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

The table *orders* stores data about om individual orders. id is an int, customer_id is a foreign key to customers, shipping_addr is a string, time_entered a timestamp showing when an order was entered, and status is a status to store the order's status (e.g. "shipped" or "delayed").

The table *order_items* stores data about individual items on an order. order_id is a foreign key to orders, product_id is a foreign key to products, while quantity and unit_price have the type int, and store how many units of the given product was ordered and the price for one unit in this order.

Problem: Write a query that finds the number of units and total cost for all products sold to customers in the country "USA", among products that have sold more than 10 units. Print the product id and name, as well as the number of units and total cost.

An example is provided under the text box.

Fill in your answer here

Example: If we have the following data in *order_lines* for customers from "USA":

order_id	product_id	quantity	unit_price
1	1	9	100
1	2	15	200
5	3	5	25
12	1	1	200

Then the result may look like this (you may choose different column names):

product_id	name	number	cost
1	*name here*	10	1100
2	*name here*	15	3000

Maximum marks: 10

11 Relasionsteori (5%)

In this problem we will only consider the table *customers*. The description of the table is the same as in the other problems. The primary key is underlined.

customers(<u>id</u>, pnr, fdate, name, address, country, time_registered)

The table *customers* stores information about customers. id and pnr (personal number) has the type int, fdate (birth date) has the data type date. Name, address, and country are strings, while time_registered has the type timestamp. time_registered is a timestamp representing when the customer was originally registered in the database. All customers have a unique combination of pnr and fdate.

Problem:

a. Which attributes in the table *customers* are candidate keys?
b. In which normal form is the table *customers*? Justify your answer.

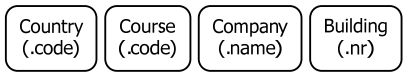
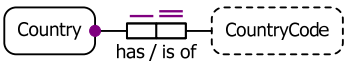
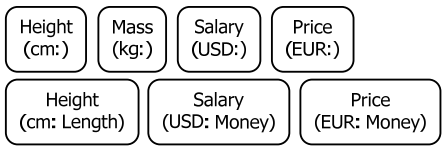
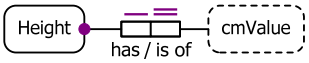
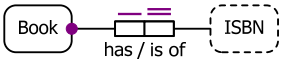
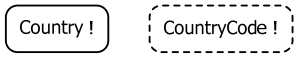
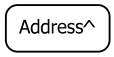
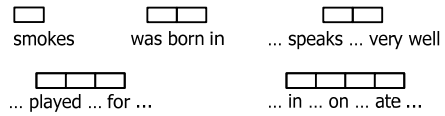
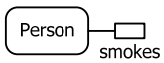
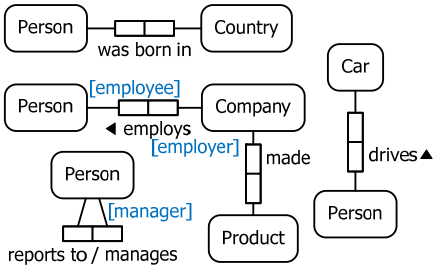
Fill in your answer here

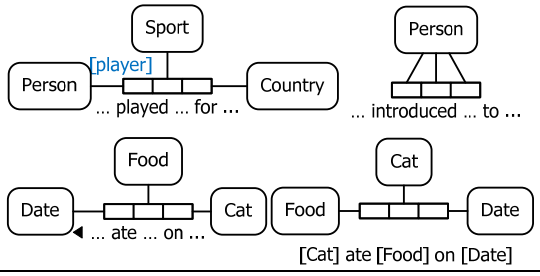
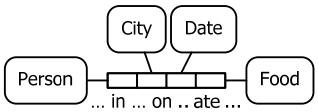
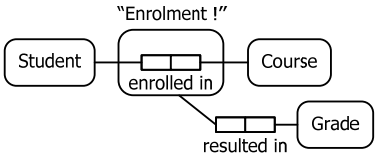
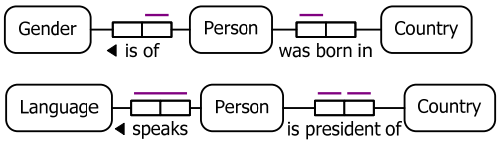
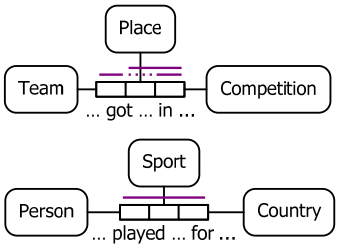
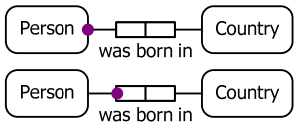
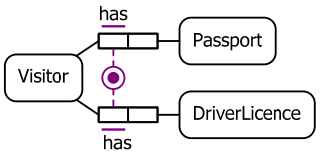
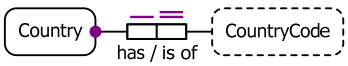
Maximum marks: 5

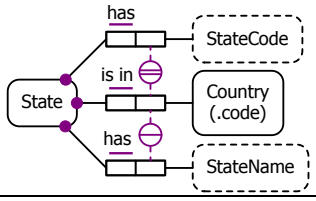
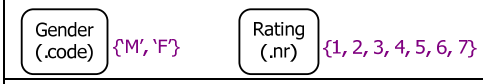
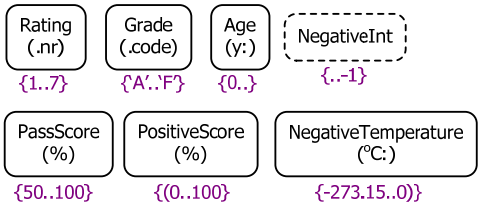
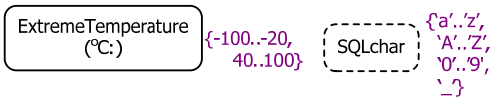
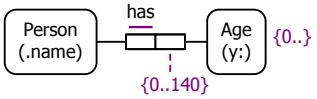
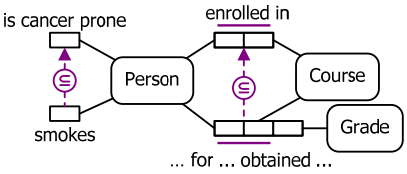
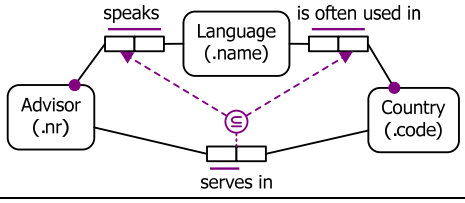
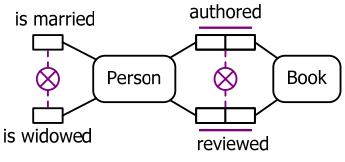
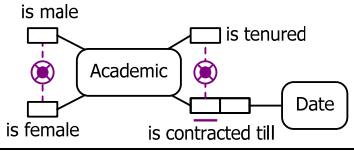
Question 2
Attached

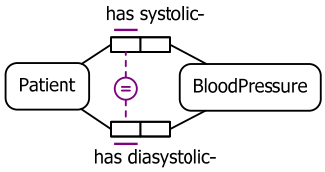
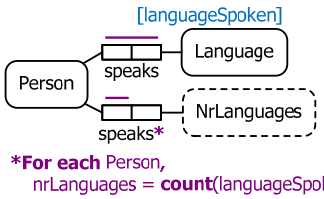
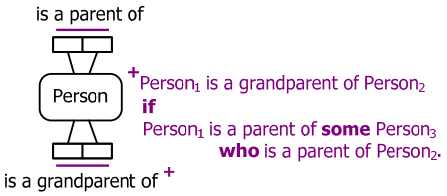
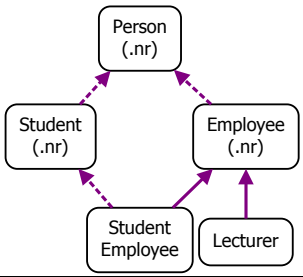
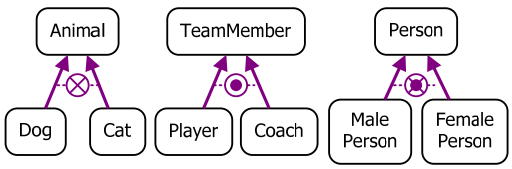
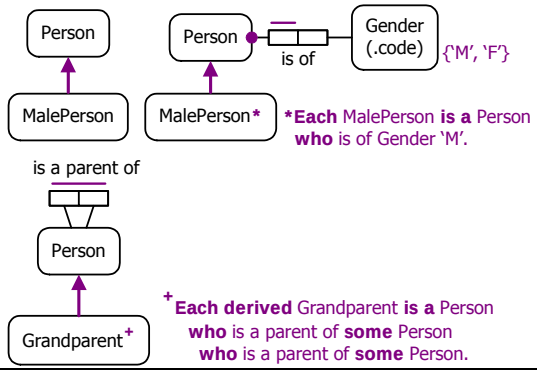
ORM 2 Graphical Notation

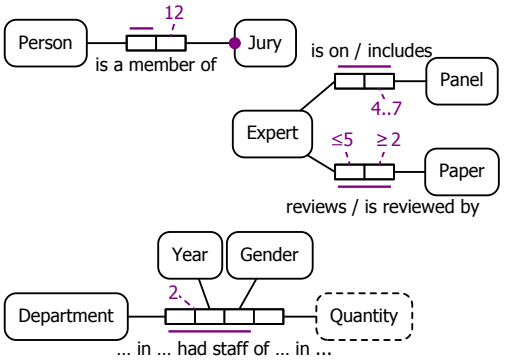
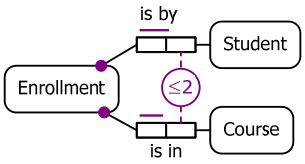
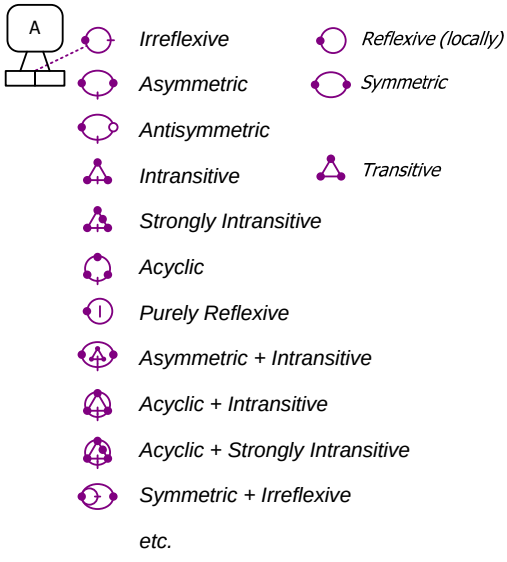
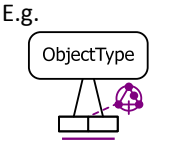
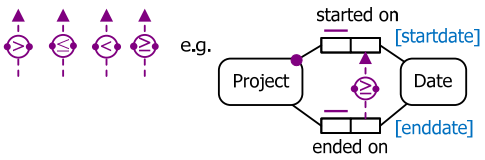
Terry Halpin

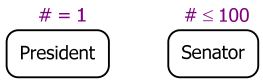
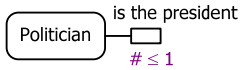
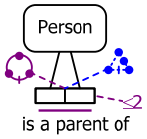
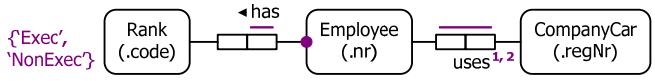
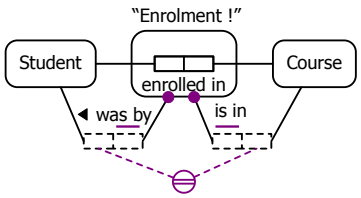
Construct	*Examples*	*Description/Notes*
Entity Type		Named soft rectangle, named hard rectangle, or named ellipse. The soft rectangle shape is the default.
Value Type		Named, dashed, soft rectangle (or hard rectangle or ellipse).
Entity type with popular reference mode		Abbreviation for injective reference relationship to value type, e.g. 
Entity type with unit-based reference mode		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.)		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		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		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)		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		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		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		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)		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		*Enumerations*
		*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]}.
		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		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		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		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		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		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		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		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		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		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		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		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	 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		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* ≤2 *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	 [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.	 	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.