INFS1200 — Week 5 Notes

Table of contents

Applied Class 4: Entity Relationship Mapping

Practice for 2026-03-16-er-to-relational-mapping[1]. See relational-mapping-notation[2] and er-diagram-notation[3] for reference.

Section A — Basic mapping

Question 1 — Select the correct mapping

A diagram combining several mapping rules at once: `H` (strong entity, key `i`, attribute `j`) and `A` (strong entity, key `b`, attributes `c`, `d`) participate in a ternary relationship `K` (roles: `N` for `H`, `M` for `A`, `1` for `P`, own attribute `l`) together with `P` (strong entity, key `q`, attribute `r`). `A` —1:N `R1`→ `E` (weak entity, partial key `f`, attribute `g`). `A` —1:1 `R2`— `P`. `P` —1:N `S`→ `M` (weak entity, partial key `o`).

[1] courses/infs1200/2026-03-16-er-to-relational-mapping.pdf
[2] courses/infs1200/relational-mapping-notation.pdf
[3] courses/infs1200/er-diagram-notation.pdf

Working

```
H [i, j]

A [b, c, d]

E [b, f, g]
E.b references A.b

P [q, b, r]
P.b references A.b

K [b, i, q, l]
K.b references A.b
K.i references H.i
K.q references P.q

M [q, o]
M.q references P.q
```

- `E` is a **weak entity** owned by `A` — its key is `(b, f)` (`A`'s key
  - its own partial key `f`).
- `R2` is a plain 1:1 relationship between `A` and `P` — merge it into one side rather than creating a separate relation for it; `P` is extended with `A`'s key as a foreign key here.
- `K` is a **ternary** relationship — its own relation gets a foreign key to *all three* participants, but only the foreign keys from the **many**-cardinality sides (`H`'s `N` and `A`'s `M`) form the primary key; `P`'s foreign key (`q`, the `1` side) is a plain attribute, not part of the key — same pattern as the lecture's N-ary "1:1:1 & N:1:1" worked cases.
- `M` is a second **weak entity**, this time owned by `P` — key `(q, o)`.

Question 2 — Subclasses and weak entities

`F` (key `a`, composite attribute `b` → `c`/`d`) has a recursive 1:N relationship `R` (attribute `r`), and disjointly (`d`) specialises into `G` (attributes `i`, `j`) and `H` (attribute `k`). `H` —1:N `S`→ `X` (weak entity, partial key `x`, attributes `y`, `z`).

(a) What's the relationship between `F`, `G`, `H`, and how are `G`/`H` mapped? (b) Interpret `S`/`X` with respect to `H`.

Working

(a) `G` and `H` are **disjoint subclasses** of `F` (an instance is at most one of `G`/`H`, never both). Standard subclass mapping — each subclass's primary key *is* the superclass's primary key, plus a foreign key back to it:

```
F [a, c, d, superF, r]
F.superF references F.a

G [a, i, j]
G.a references F.a

H [a, k]
H.a references F.a
```

(`F.superF`/`r` come from `R`, `F`'s own recursive 1:N relationship — mapped the same way as any binary 1:N relationship, just back onto `F` itself.)
(b) `X` is a **weak entity** owned by `H` via the identifying relationship `S` — each `X` belongs to exactly one `H`, but one `H` can own many `X`s. `X`'s real key is `(a, x)` (`H`'s key, inherited from `F`, plus `X`'s own partial key `x`):

```
X [a, x, y, z]
X.a references H.a
```

Question 3 — Olympics database

Map the ER diagram: *Athlete (id, sex, name, age, country)* —*M:N* *Participates (placement)*→ *Event (id, name, category)* —*N:1* *Venue (id, name, address).*

Working

```
Athlete [id, sex, name, age, country]

Venue [id, name, address]

Event [id, name, category, venue]
Event.venue references Venue.id

Participates [athleteID, eventID, placement]
Participates.athleteID references Athlete.id
Participates.eventID references Event.id
```

`Event`—`Venue` is 1:N, so `Event` (the N side) gets a plain foreign key. `Athlete`—`Event` is M:N, so `Participates` is its own new relation, carrying both foreign keys plus the relationship's own attribute `placement`.

Section B — Advanced mapping

Question 1 — Cinema

Map: `Movie` (`name`, `description`, `runningTime`) —1:N `Viewing` (`date`, `time`, multivalued `Snack`)←1:N `Customer` (`id`, `name`, `email`), each `Viewing` overseen by exactly one `Attendant` (`staffId`, `name`, `phone`).

i Working

```
Movie [name, description, runningTime]

Customer [id, name, email]

Attendant [staffID, name, phone]

Viewing [name, id, date, time, attendant]
Viewing.name references Movie.name
Viewing.id references Customer.id
Viewing.attendant references Attendant.staffID

ViewingSnack [name, id, date, time, snack]
ViewingSnack.{name, id, date, time} references Viewing.{name, id, date, time}
```

`Viewing`'s own key is the composite `(name, id, date, time)` — the movie + customer + timestamp jointly identify one viewing. `Snack` is multivalued, so it gets its own relation (`ViewingSnack`) with a composite foreign key back to the full `Viewing` key.

Question 2 — Student peer evaluation

Map: `Student` (`sid`, `firstName`, `secondName`) submits `Assessments` (`sid`+`number`); students peer-evaluate each other's assessment submissions, recording a `mark`.

Working

```
Student [sid, firstName, secondName]

Assessment [sid, number]
Assessment.sid references Student.sid

PeerEvaluation [studentSid, assessmentSid, number, evaluationId, sidGrades, mark]
PeerEvaluation.studentSid references Student.sid
PeerEvaluation.sidGrades references Student.sid
PeerEvaluation.{assessmentSid, number} references Assessment.{sid, number}
```

`PeerEvaluation` has **two separate foreign keys to `Student`** playing different roles — `studentSid` (the evaluator submitting the mark) and `sidGrades` (whose assessment is being marked) — plus a composite foreign key to the specific `Assessment` being evaluated.

Question 3 — Spot the mistakes

A junior developer mapped question 1's Cinema diagram as:

```
Movie [name, description, runningTime]

Customer [id, name, email]

Attendant [staffID, name, phone]

Viewing [name, id, date, time, attendant]
Viewing.name references Movie.name
Viewing.id references Customer.id
Viewing.attendant references Attendant.staffID

ViewingSnack [name, id, date, time, snack]
ViewingSnack.name references Movie.name
ViewingSnack.id references Viewing.id
ViewingSnack.date references Viewing.date
ViewingSnack.time references Viewing.time
```

Find the mistakes.

Working

`ViewingSnack` should reference `Viewing`'s composite key as **one** foreign key constraint (`ViewingSnack.{name, id, date, time} references Viewing.{name, id, date, time}`), not four *separate*, independently-named foreign key lines pointing at different tables (`Movie` for `name`, `Viewing` for `id`, and `Viewing` again for `date`/`time`, inconsistently) — splitting a composite foreign key up like this breaks the link between a `ViewingSnack` and *the specific* ***`Viewing`*** it belongs to (nothing actually forces all four columns to jointly match one real `Viewing` row), and pointing `name` back at `Movie` instead of `Viewing` is simply the wrong target relation for a foreign key that's supposed to identify *which viewing* the snack belongs to.

Section C — EER diagram mapping

Question 1 — Medical clinic

Map: ***`Doctor`*** *(****`licenceNo`****) splits into* ***`GP`****/****`Specialist`*** *(****`specialisationArea`****);* ***`Doctor`*** *—M:N* ***`WorksIn`***→ ***`Clinic`*** *(****`regNo`****);* ***`Clinic`*** *has a subclass* ***`SpecialistClinic`****, run by exactly one* ***`Specialist`*** *(****`Runs`****, 1:N);* ***`Patient`*** *(****`patientId`****) —M:N* ***`Appointment`*** *(****`dateTime`****,* ***`fee`****)→* ***`Doctor`****.*

Working

```
Doctor [licenceNo]

Patient [patientId]

Clinic [regNo]

GP [licenceNo]
GP.licenceNo references Doctor.licenceNo

Specialist [licenceNo, specialisationArea]
Specialist.licenceNo references Doctor.licenceNo

SpecialistClinic [regNo]
SpecialistClinic.regNo references Clinic.regNo

Appointment [licenceNo, dateTime, patientId, fee]
Appointment.licenceNo references Doctor.licenceNo
Appointment.patientId references Patient.patientId
```

```
WorksIn [licenceNo, regNo]
WorksIn.licenceNo references Doctor.licenceNo
WorksIn.regNo references Clinic.regNo

Runs [licenceNo, regNo]
Runs.licenceNo references Specialist.licenceNo
Runs.regNo references SpecialistClinic.regNo
```

`GP`/`Specialist` are standard disjoint-subclass mappings of `Doctor`. `SpecialistClinic` is a subclass of `Clinic` with no extra attributes of its own — just a foreign key back to `Clinic`. `Runs` is a 1:N relationship between `Specialist` and `SpecialistClinic` specifically (not the general `Doctor`/`Clinic` pair), so its foreign keys target the *subclass* relations, not the superclasses.

Reference material

ER Diagram Notation

Reference legend for the symbols used across every ER/EER diagram in this course — introduced across 2026-02-23-conceptual-database-design-and-the-er-model[4] and 2026-03-02-weak-entities-the-eer-model-and-design-choices[5].

Core symbols

Symbol	Meaning
Rectangle	**Entity type**
Double-line rectangle	**Weak entity type** (no key attribute of its own)
Oval	**Attribute**
Oval, name underlined	**Key attribute**
Oval, name underlined with a dotted line	**Partial key attribute** (weak entity's own distinguishing attribute)
Double-line oval	**Multivalued attribute** (can hold more than one value)
Dashed-line oval	**Derived attribute** (computed from another stored attribute, e.g. Age from BirthDate)

[4] courses/infs1200/2026-02-23-conceptual-database-design-and-the-er-model.pdf
[5] courses/infs1200/2026-03-02-weak-entities-the-eer-model-and-design-choices.pdf

Symbol	Meaning
Diamond	**Relationship type**
Double-line diamond	**Identifying relationship type** (connects a weak entity to its owner entity)

- **Composite attribute**: an attribute with its own sub-attributes attached (e.g. `Name` splitting into `FirstName`/`MiddleName`/`LastName`) — the parent attribute oval isn't underlined/dashed/doubled itself unless it's also a key/derived/multivalued attribute.
- **Value sets** (the domain of legal values for an attribute, e.g. "integers 21-65") are never shown on the diagram itself.

Relationship constraints

See relationship-constraints[6] for cardinality ratio (1:1, 1:N, M:N) and participation (total/partial) notation.

Subclasses / superclasses (EER)

Symbol	Meaning
between two entity types	The entity type on the narrow side is a **subclass** of the one on the wide side
A circle joining several subclasses to one superclass	Several entity types are all **subclasses of the same superclass**
`d` inside that circle	**Disjoint** specialization — an entity instance can belong to at most one of the subclasses
`o` inside that circle	**Overlapping** specialization — an entity instance can belong to more than one subclass at once
Single line from superclass to the subclass circle	**Partial** specialization — not every superclass instance needs to belong to a subclass
Double line from superclass to the subclass circle	**Total** specialization — every superclass instance must belong to at least one subclass

[6] courses/infs1200/relationship-constraints.pdf

Naming conventions (INFS1200/7900 style guide)

Course-specific naming standard for entities/relationships/attributes on ER and EER diagrams (distinct from — and in addition to — the mandatory notation above):

- **Entity names**: capitalised, no spaces, ideally one word (e.g. `STUDENT`, `POLICEOFFICER` — not `PoliceOfficer`).
- **Relationship names**: capitalised, no spaces, ideally one word, preferably a verb (e.g. `CREATES`, `ACTSIN` — not `LeadActor`).
- **Attribute names**: UpperCamelCase — first letter of each word capitalised, no spaces, acronym letters stay capitalised (e.g. `ComputerIP`, `DateOfBirth` — not `Date of Birth`).

How diagrams are drawn in these notes

Diagrams in this course's notes are drawn as Mermaid `flowchart` graphs (renders in both PDF and HTML), using shapes chosen to match the Chen notation above as closely as Mermaid's flowchart shapes allow:

Mermaid shape	Used for
`[Entity]` (rectangle)	Entity type
`[[WeakEntity]]` (subroutine, double vertical bars)	Weak entity type
`{Relationship}` (diamond/rhombus)	Relationship type
`([Attribute])` (stadium)	Attribute
`((d))` / `((o))` (small circle)	Disjoint/overlapping specialisation marker

Since Mermaid can't reliably underline/dash node text across renderers, attribute qualifiers are written as a plain-text suffix instead: `(key)`, `(partial key)`, `(derived)`, `(multi)`. Composite attributes are drawn as a parent attribute node connected to its component attribute nodes. Cardinality is a text label on the edge (`"1"`, `"N"`, `"M"`); **total** participation/identifying relationships use a thick edge (`===`/`==>`), **partial** participation uses a thin edge (`---`/`-->`).

Any diagram with more than ~6-8 nodes needs an explicit `%%| fig-width:` cell option (in inches) or it runs off the right edge of the PDF page - mermaid sizes itself from the diagram's own SVG bounding box, and there's no project-wide default that reaches it (a document/project `fig-width` only applies to executed-code-cell figures, e.g. matplotlib). `5.5` fits this vault's PDF page width (`scrartcl`, `DIV=11`, letter) comfortably:

````
```{mermaid}
%%| fig-width: 5.5
flowchart TD
    ...
````
````

```

## Relational Mapping Notation

Reference legend + course style guide for writing relational schemas — introduced across 2026-03-09-the-relational-model-and-integrity-constraints[7] and 2026-03-16-er-to-relational-mapping[8].

### Schema notation

- `Relation [attr1, attr2, ...]` — a relation schema; the (primary) key attribute(s) are underlined.
- A composite primary key gets a single continuous underline spanning all of its attributes, e.g. `Enrolment [studentId, courseCode, sem, year]`.
- `Relation.fk references OtherRelation.pk` — a foreign key constraint, listed directly under the relation it belongs to.
- `Relation.{fk1, fk2} references OtherRelation.{pk1, pk2}` — a composite foreign key referencing a composite primary key.

### Naming convention (INFS1200/7900 style guide)

- **Table names**: UpperCamelCase (first letter of each word capitalised, no spaces) — e.g. `Flight`, not `FLIGHT`.
- **Attribute names**: lowerCamelCase (first letter of each word from the *second* word onwards capitalised) — e.g. `departureTime`, not `Departure Time`. Acronym attribute names stay entirely lowercase (e.g. `eta`, not `ETA`).
- A space separates the table name from the opening bracket: `Flight [planeNumber, ...]`, not `Flight[planeNumber, ...]`.

### Layout convention

A table's foreign key constraint lines go directly underneath that table's own definition, with a blank line before the next table starts — not all grouped together under a separate "Foreign Keys:" heading at the end:

[7] courses/infs1200/2026-03-09-the-relational-model-and-integrity-constraints.pdf
[8] courses/infs1200/2026-03-16-er-to-relational-mapping.pdf
```

```
Employee [ssn, firstName, lastName, dob, manager]
Employee.manager references Employee.ssn

Dependant [ssn, name, dob]
Dependant.ssn references Employee.ssn

DependantPhoneNumber [ssn, name, phoneNumber]
DependantPhoneNumber.{ssn, name} references Dependant.{ssn, name}
```