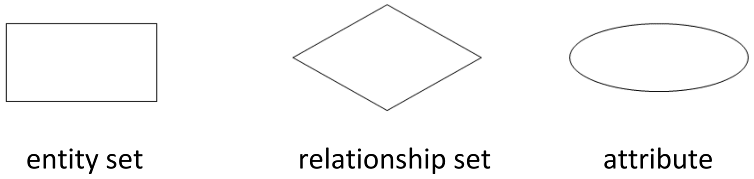
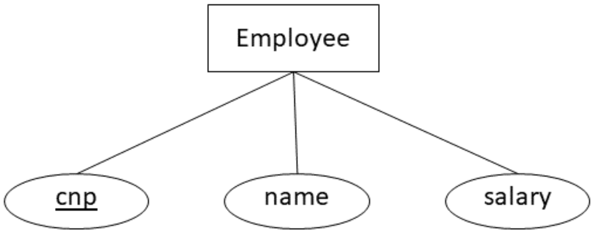
Databases
Seminar 1

Mapping Entity-Relationship Diagrams to Relational Schemas
SQL – Data Definition Language

Concepts in the entity-relationship (ER) model. The ER diagram



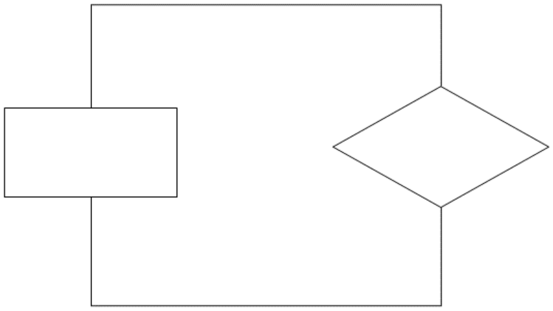
Example:

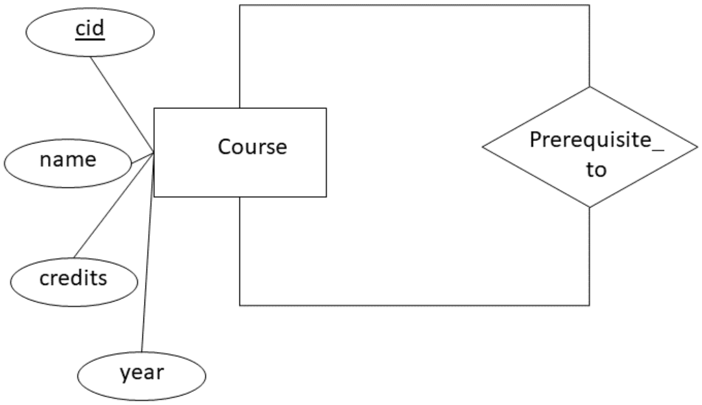


- *cnp* – primary key for the *Employee* entity set

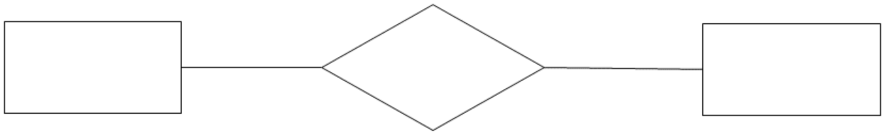
The degree of a relationship set – the number of entity sets that participate in the relationship set
Examples:

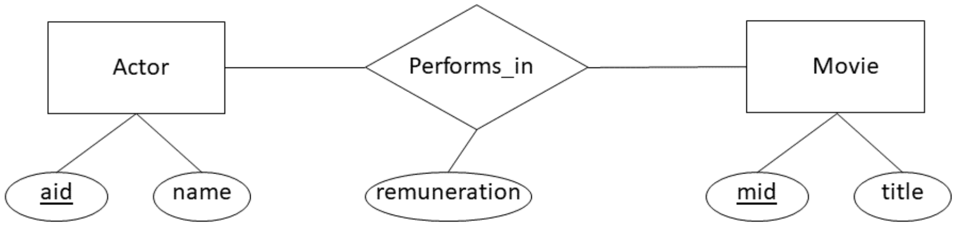
- Unary relationship set (degree = 1)





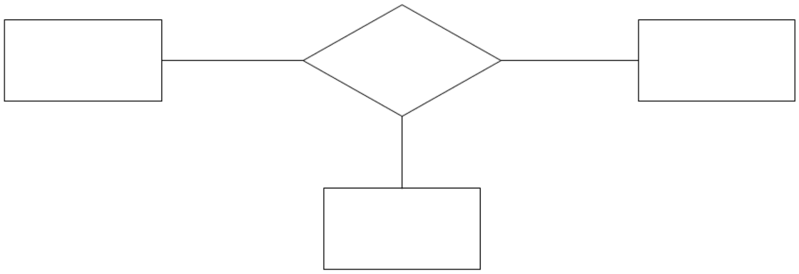
- Binary relationship set (degree = 2)

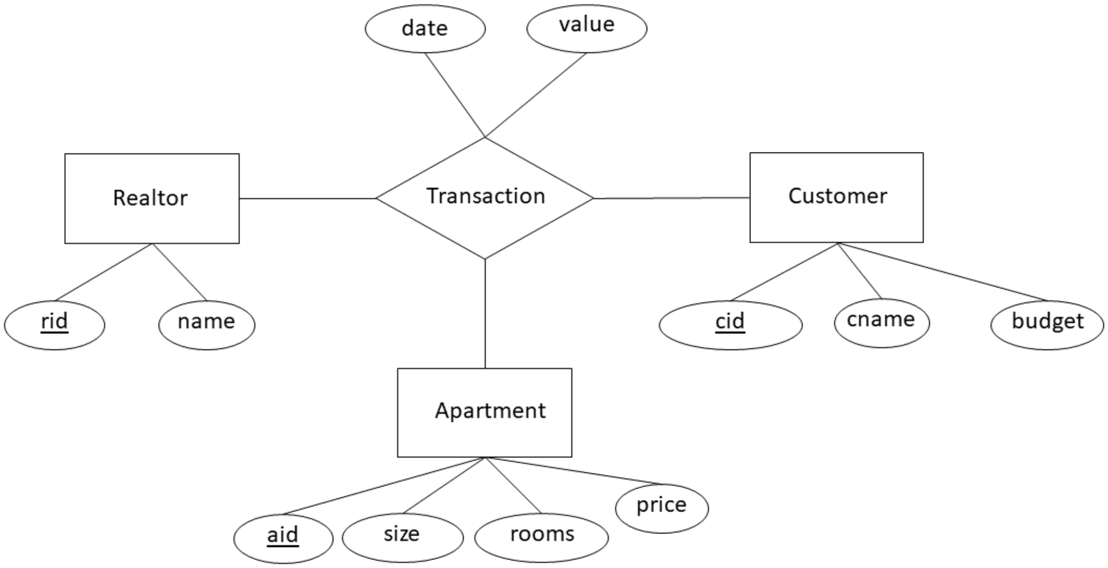




- *remuneration* – descriptive attribute for the *Performs_in* relationship set

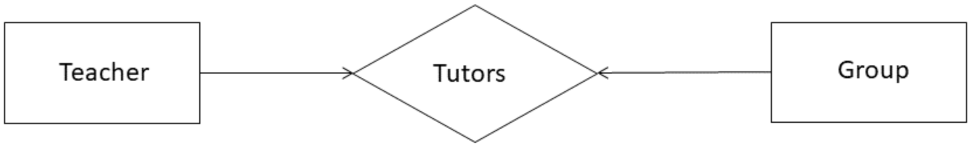
- Ternary relationship set (degree = 3)



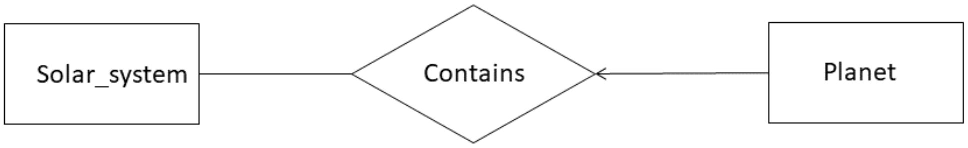


Mapping cardinalities - binary relationship sets

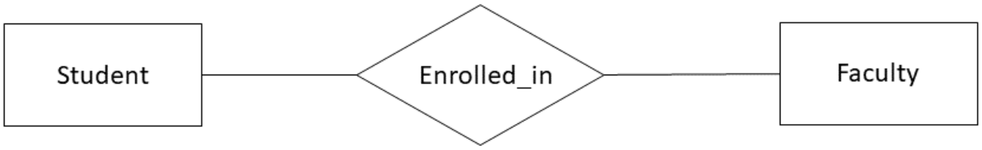
1:1



1:n

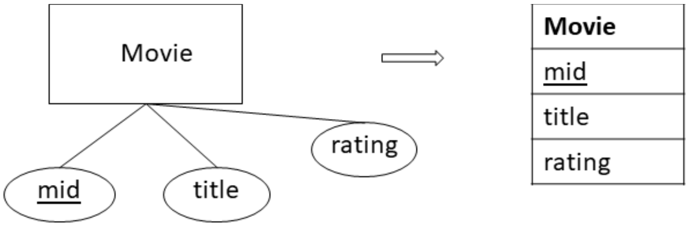


m:n



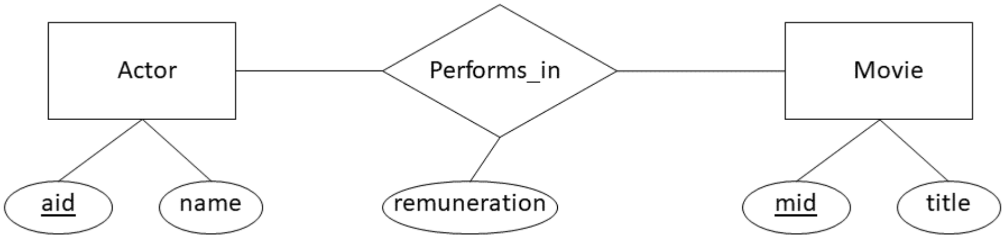
Translation to the Relational Model

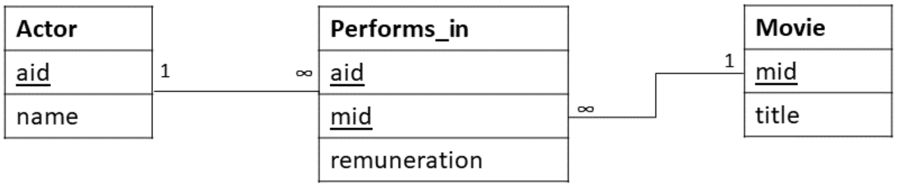
Entity set -> relation



- the name of the entity set becomes the name of the relation;
- the attributes of the entity set become attributes in the relation;
- the primary key of the entity set becomes the primary key of the relation.

m:n relationship set -> relation



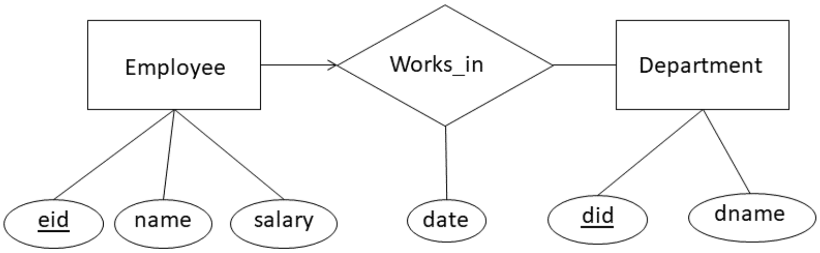


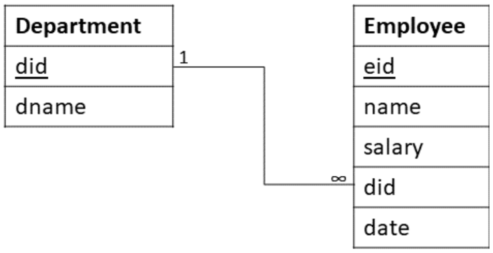
- the name of the relationship set becomes the name of the relation;
- the primary key attributes for each entity set that takes part in the relationship set:
    - become attributes in the relation;
    - are foreign keys in the relation;
    - can become the primary key of the relation;
- the relationship set descriptive attributes become attributes in the relation.

n:1 (or 1:n) relationship set

- one can avoid creating an additional relation.

Mapping Entity-Relationship Diagrams to Relational Schemas
SQL – Data Definition Language





- the *Employee* entity set is on the *n* (*many*) side of the *Works_in* relationship set, while the *Department* entity set is on the *1* side;
- the relationship set data is included in the *Employee* relation, corresponding to the entity set that lies on the *n* side of the relationship set; this relation will store every employee's department along with the date when he/she started working there;
- the key in the *Department* relation (corresponding to the entity set that lies on the *1* side of the relationship set) becomes a foreign key in the *Employee* relation.

Note: conceptual modeling is detailed in a next lecture. The current seminar aims to provide support for the first lab.

SQL – Data Definition Language

Statement that creates the *Movie* table (relation):

```
CREATE TABLE Movie
  (mid CHAR(10),
  title VARCHAR(70),
  year_of_release TINYINT,
  running_time INT,
  box_office DECIMAL(12, 2))
```

- the type (domain) of each field (attribute) is specified and enforced by the DBMS whenever tuples are added or modified.

Statement that creates the *MovieCast* table:

```
CREATE TABLE MovieCast
  (mid CHAR(10),
  aid CHAR(10),
  remuneration DECIMAL(12, 2))
```

- stored data - which actors perform in which movies, and their corresponding remuneration .

Statement that drops the *Movie* table:

```
DROP TABLE Movie
```

- both the schema information and the tuples in the table are removed.

Statement that alters the schema of the *Movie* table by adding a new field:

```
ALTER TABLE Movie
ADD synopsis VARCHAR(500)
```

- every tuple in the current instance is extended with the *null* value in the new field;
- this statement assumes a table named *Movie* exists.

Statement that alters the schema of the *Movie* table by removing a field:

```
ALTER TABLE Movie
DROP COLUMN running_time
```

Statement that alters the schema of the *Movie* table by changing the type of a field and adding a NOT NULL constraint:

```
ALTER TABLE Movie
ALTER COLUMN year_of_release SMALLINT NOT NULL
```

* careful when altering columns with associated constraints

The *MovieCast* table creation statement with the primary key declaration:

```
CREATE TABLE MovieCast
  (mid CHAR(10),
  aid CHAR(10),
  remuneration DECIMAL(12, 2),
  PRIMARY KEY(mid, aid))
```

- multiple candidate keys can be declared using UNIQUE; one of them is chosen as the primary key;
- the primary key *{mid, aid}* corresponds to the constraint "for a given actor and a given movie, there is a single remuneration"; there are no two tuples in the relation with identical values in both the *mid* and the *aid* fields.

The *MovieCast* table creation statement with unrealistic constraints:

```
CREATE TABLE MovieCast
  (mid CHAR(10),
  aid CHAR(10),
  remuneration DECIMAL(12, 2),
  PRIMARY KEY(aid),
  UNIQUE(mid, remuneration))
```

- this is an example of how not to define keys; designating *{aid}* as the primary key corresponds to the constraint "an actor can only perform in one movie", whereas choosing *{mid, remuneration}* as a candidate key corresponds to the constraint "no two actors can get the same remuneration for a given movie"; such constraints prevent the storage of database instances that can arise in practice.

Statement that adds a primary key to the *Movie* table:

```
ALTER TABLE Movie
ADD CONSTRAINT PK_Movie PRIMARY KEY(mid)
```

- *mid* needs to be NOT NULL.

The *MovieCast* table creation statement with a foreign key declaration:

```
CREATE TABLE MovieCast
  (mid CHAR(10),
  aid CHAR(10),
  remuneration DECIMAL(12, 2),
  PRIMARY KEY(mid, aid),
  FOREIGN KEY(mid) REFERENCES Movie(mid))
```

- declaring *mid* to be a foreign key like above corresponds to the constraint "the *MovieCast* table can store actors only for movies that appear in the *Movie* table".

The *MovieCast* table creation statement with foreign key actions:

```
CREATE TABLE MovieCast
```

```
  (mid CHAR(10),
  aid CHAR(10),
  remuneration DECIMAL(12, 2),
  PRIMARY KEY(mid, aid),
  FOREIGN KEY(mid) REFERENCES Movie(mid)
    ON DELETE CASCADE
    ON UPDATE NO ACTION
)
```

- in the case of update / delete operations, in order to enforce referential integrity constraints, the system can execute 4 actions:
  - NO ACTION – the update / delete is not allowed if it violates the specified integrity constraint (default option);
  - CASCADE – the update / delete is allowed on the parent table, but it also generates updates / deletes on the child table;
  - SET NULL – the foreign key column values are replaced with *null* (only if they are nullable);
  - SET DEFAULT – the foreign key column values are replaced with their default values (specified with DEFAULT); if a column is nullable and doesn't have a DEFAULT definition, *null* will be considered as the default value for the column.
- what happens if we replace the ON UPDATE action with SET NULL in the above statement?

Statement that adds a foreign key to the *MovieCast* table (table *Actor* is assumed to exist with *{aid}* as primary key):

```
ALTER TABLE MovieCast
ADD CONSTRAINT FK_MovieCast_Actor FOREIGN KEY(aid) REFERENCES Actor(aid)
```

Statement that removes a foreign key from the *MovieCast* table:

```
ALTER TABLE MovieCast
DROP FK_MovieCast_Actor
```

Statement that adds a DEFAULT definition to the *MovieCast* table:

```
ALTER TABLE MovieCast
ADD DEFAULT 0 FOR remuneration
```

The *Movie* table creation statement with a CHECK constraint:

```
CREATE TABLE Movie
  (mid CHAR(10),
  title VARCHAR(70),
  year_of_release SMALLINT,
  running_time INT,
  box_office DECIMAL(12, 2),
  CONSTRAINT PK_Movie PRIMARY KEY(mid),
  CONSTRAINT year_range
    CHECK(year_of_release >= 1905 AND year_of_release <= 2018))
```