Home
What's New ?
Getting Started
Online Server
Running
F.A.Q.
Download
Forum
Theme
Preprocessing
Standard Library
Hitchhiker's Guide
PDF Guide

Sequence Use Case Class Activity Component State Object Deployment Timing Rege

PlantUML > Language specification > Entity Relationship

Entity Relationship Diagrams (Chen's notation)

This page is for Chen's Entity Relationship notation, which is commonly used in teaching. *See also Information Engineering diagrams.*

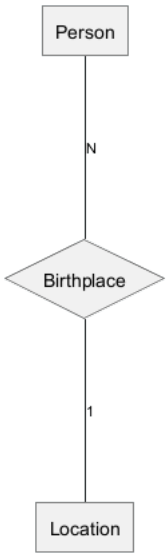
Entity Relationship (ER) diagrams are used to model databases at a conceptual level by describing entities, their attributes, and the relationships between them. In addition to basic relationships, PlantUML also supports subclasses and union types. This extended notation is sometimes referred to as Enhanced Entity Relationship (EER) or Extended Entity Relationship notation.

[Ref. GH-945 and GH-1718]

Minimal Example

Vertical *(by default)*

```
@startchen

entity Person {
}
entity Location {
}
relationship Birthplace {
}

Person -N- Birthplace
Birthplace -1- Location

@endchen
```



Horizontal

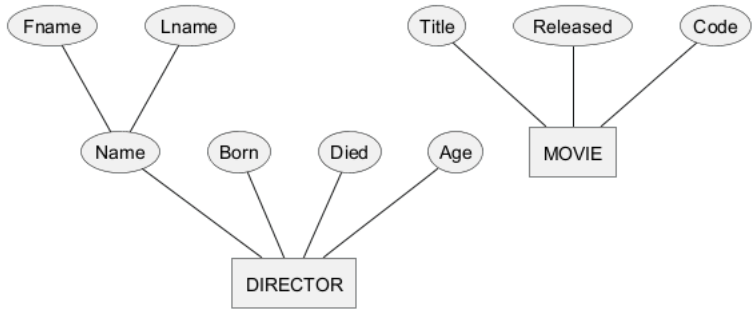
```
@startchen
left to right direction

entity Person {
}
entity Location {
}
relationship Birthplace {
}

Person -N- Birthplace
Birthplace -1- Location

@endchen
```

[Ref. PR-1740]

Entities and attributes

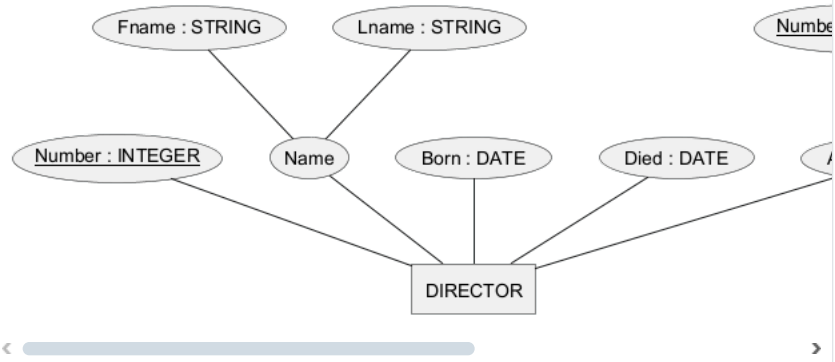
Home
What's New ?
Getting Started
Online Server
Running
F.A.Q.
Download
Forum
Theme
Preprocessing
Standard Library
Hitchhiker's Guide
PDF Guide


```
@startchen

entity DIRECTOR {
  Name {
    Fname
    Lname
  }
  Born
  Died
  Age
}

entity MOVIE {
  Title
  Released
  Code
}

@endchen
```



Attributes can be *keys*, meaning that their value is unique among entities of a given type, or they can be *derived*, meaning that their value is computed based on other attributes. Attributes may also be *multi-valued*, or have their *domain* (set of allowed values) defined.

```
@startchen

entity DIRECTOR {
  Number : INTEGER <<key>>
  Name {
    Fname : STRING
    Lname : STRING
  }
  Born : DATE
  Died : DATE
  Age : INTEGER
}

entity CUSTOMER {
  Number : INTEGER <<key>>
  Bonus : REAL <<derived>>
  Name : STRING <<multi>>
}

@endchen
```



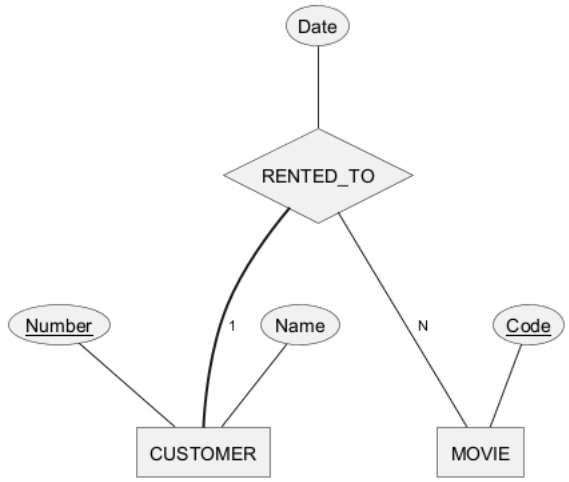
Relationships

Relationships describe how entities are related to each other. These can be *one-to-one*, *one-to-many*, or *many-to-many*. They can have *total participation* (mandatory) or *partial participation* (optional). Total

Home
What's New ?
Getting Started
Online Server
Running
F.A.Q.
Download
Forum
Theme
Preprocessing
Standard Library
Hitchhiker's Guide
PDF Guide

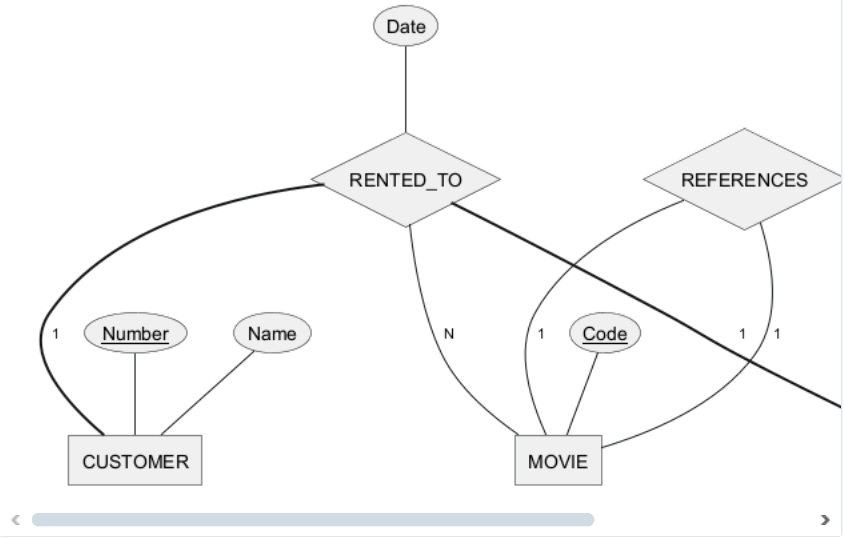
Sequence Use Case Class Activity Component State Object Deployment Timing Rege

```
@startchen

entity CUSTOMER {
  Number <<key>>
  Name
}

entity MOVIE {
  Code <<key>>
}

relationship RENTED_TO {
  Date
}

RENTED_TO =1= CUSTOMER
RENTED_TO -N- MOVIE

@endchen
```



Relationships are not limited to two entities.

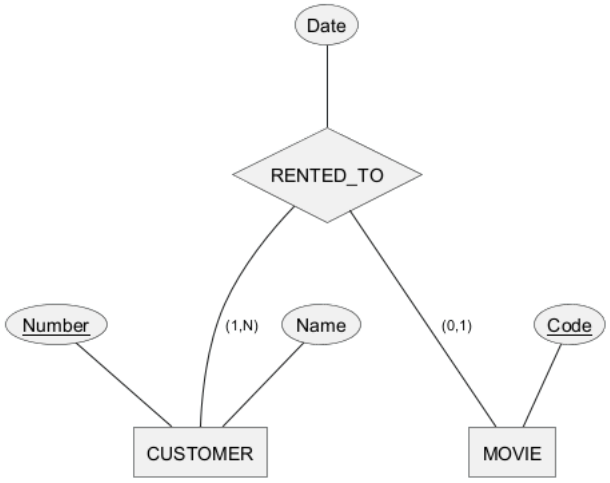
```
entity CUSTOMER {
  Number <<key>>
  Name
}

entity MOVIE {
  Code <<key>>
}

entity INVOICE {
  Number <<key>>
  Amount
}

relationship RENTED_TO {
  Date
}

RENTED_TO =1= CUSTOMER
RENTED_TO -N- MOVIE
RENTED_TO =1= INVOICE

relationship REFERENCES {
}

REFERENCES -1- MOVIE
REFERENCES -1- MOVIE

@endchen
```



Structural constraints

The cardinality of relationships can also be expressed as a range.

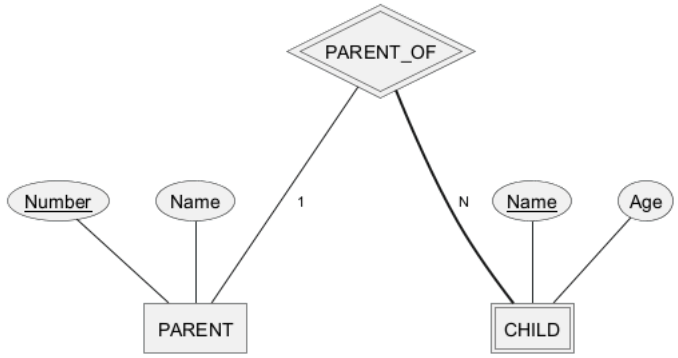
```
entity CUSTOMER {
  Number <<key>>
  Name
}

entity MOVIE {
  Code <<key>>
}

relationship RENTED_TO {
  Date
}

RENTED_TO -(1,N)- CUSTOMER
RENTED_TO -(0,1)- MOVIE

@endchen
```



Identifying relationships

A *weak* entity does not have a key attribute that uniquely identifies each instance of that entity. Instead, it is identified by the combination of a *partial key* on the weak entity itself and the key of another entity, which it is related to via an *identifying relationship*. A weak entity must have total participation in its identifying relationship.

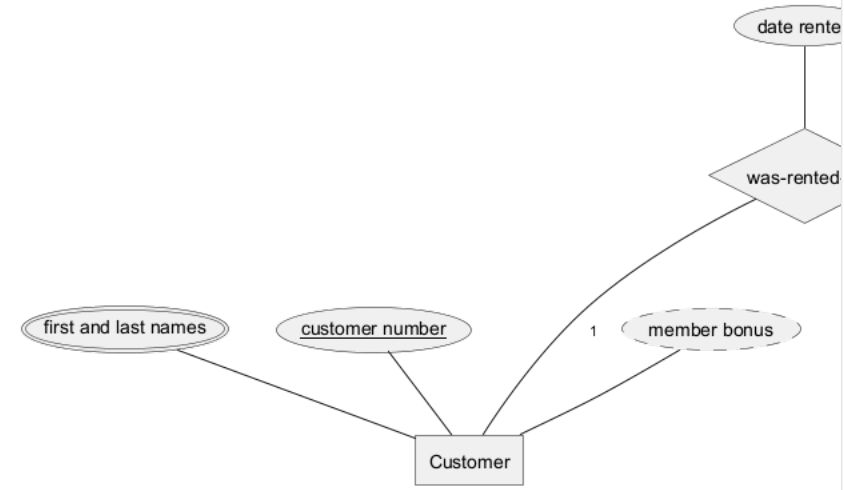
```
entity PARENT {
  Number <<key>>
  Name
}

entity CHILD <<weak>> {
  Name <<key>>
  Age
}

relationship PARENT_OF <<identifying>> {
}

PARENT_OF -1- PARENT
PARENT_OF =N= CHILD

@endchen
```



Aliases

Entities, attributes and relationships can be given aliases to make the diagram more readable.

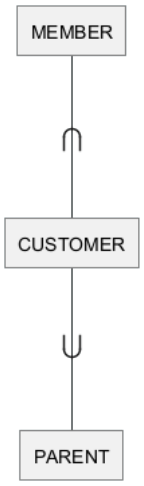
```
entity "Customer" as CUSTOMER {
  "customer number" as Number <<key>>
  "member bonus" as Bonus <<derived>>
  "first and last names" as Name <<multi>>
}

entity "Movie" as MOVIE {
  "barcode" as Code
}

relationship "was-rented-to" as RENTED_TO {
  "date rented" as Date
}

RENTED_TO -1- CUSTOMER
RENTED_TO -N- MOVIE

@endchen
```



Subclasses and categories

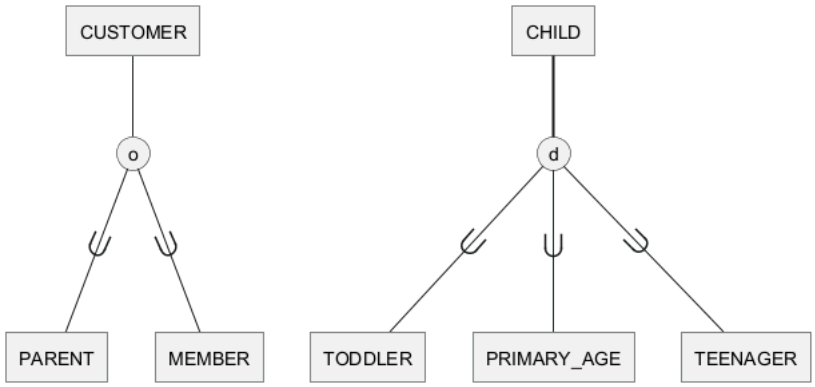
Entities can have *subclasses* and *superclasses*, much like in OOP, however a given subclass can have multiple superclasses. These are visually indicated using the subset symbol from set-theory.

```
@startchen

entity CUSTOMER {
}

entity PARENT {
}

entity MEMBER {
}

CUSTOMER ->- PARENT
MEMBER -<- CUSTOMER

@endchen
```



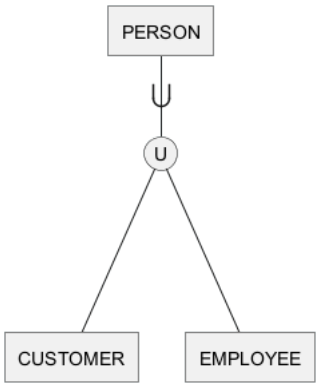
We can show how the different subclasses of a given entity are related by combining the associations. They can be either *disjoint* (one at a time) or *overlapping* (multiple at the same time).

```
entity CUSTOMER {
}

entity PARENT {
}

entity MEMBER {
}

CUSTOMER ->- o { PARENT, MEMBER }

entity CHILD {
}

entity TODDLER {
}

entity PRIMARY_AGE {
}

entity TEENAGER {
}

CHILD =>= d { TODDLER, PRIMARY_AGE, TEENAGER }

@endchen
```



Categories or *union types* are similar to subclasses and can be used to group together multiple related entities.

```
@startchen

entity CUSTOMER {
}

entity EMPLOYEE {
}

entity PERSON {
}

PERSON ->- U { CUSTOMER, EMPLOYEE }

@endchen
```



Complex Example

Home
What's New ?
Getting Started
Online Server
Running
F.A.Q.
Download
Forum
Theme
Preprocessing
Standard Library
Hitchhiker's Guide
PDF Guide

Sequence Use Case Class Activity Component State Object Deployment Timing Rege

```
<style>
.red {
BackGroundColor Red
FontColor White
}
.blue {
BackGroundColor Blue
FontColor White
}
</style>

entity "Director" as DIRECTOR {
"No." as Number <<key>>
Name {
Fname
Lname
}
Born : DATE
Died<<red>>
Age<<blue>>
}

entity "Customer" as CUSTOMER {
Number <<key>>
Bonus <<derived>>
Name <<multi>>
}

entity "Movie" as MOVIE {
Code
}

relationship "was-rented-to" as RENTED_TO {
Date
}

RENTED_TO -1- CUSTOMER
RENTED_TO -N- MOVIE
RENTED_TO -(N,M)- DIRECTOR

entity "Parent" as PARENT {
}

entity "Member" as MEMBER {
}

CUSTOMER ->- PARENT
MEMBER -<- CUSTOMER

entity "Kid" as CHILD <<weak>> {
Name <<key>>
}

relationship "is-parent-of" as PARENT_OF <<identifying>> {
}

PARENT_OF -1- PARENT
PARENT_OF =N= CHILD

entity "Little Kid" as TODDLER {
FavoriteToy
}

entity "Primary-Aged Kid" as PRIMARY_AGE {
FavoriteColor
}

entity "Teenager" as TEEN {
Hobby
}

CHILD =>= d { TODDLER, PRIMARY_AGE, TEEN }

entity "Human" as PERSON {
}

PERSON ->- U { CUSTOMER, DIRECTOR }
@endchen
```

Home
What's New ?
Getting Started
Online Server
Running
F.A.Q.
Download
Forum
Theme
Preprocessing
Standard Library
Hitchhiker's Guide
PDF Guide

Sequence Use Case Class Activity Component State Object Deployment Timing Rege