Lecture 06: Normalization and data modelling

1. ***Normalization***
2. ***redundancy***
3. ***Normal forms***
4. ***UNF, 1NF, 2NF and 3NF***
5. ***Functional dependency***
6. ***Determinants***
7. ***Data modelling***
8. ***Notation***
9. ***Different models***
10. ***Concept definition***
11. ***Attribute definition***
12. ***Good and bad data models***

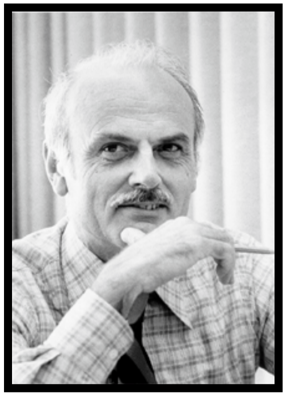
Pär Douhan, pdo@du.se

Normalization

We begin with normalization

Normalization

Normalization is a series of steps followed to obtain a database design that allows for efficient access and storage of data. These steps reduce data redundancy and the chances of data becoming inconsistent.



- Normalization is a process for deciding which attributes to group together in a table.
- Based on Ted Codd's normalization rules.
- It is a theory used to avoid poor design on your database.
- Where poor design causes problems in deleting and updating data, as well as a lot of physical redundancy.

Redundancy

There are different types of redundancy:

1. **Physical redundancy:** When we store the same data on multiple rows in a table. This is considered **something bad** in connection with data modeling. When we get this kind of redundancy, we have too low a degree of normalization.

2. **Infological redundancy:** Data that can be **derived** from already stored data. An example might be if we store data about the diameter property of different balls. If we were also to store data about the volume property, this would be an example of *infological redundancy*. This is because we can easily calculate the volume of a ball if we know its diameter, i.e. the volume can be derived from the diameter.

UNF

UNF, Unnormalized Form:

- *Table containing one or more rows of non-atomic values in cells.*

RENTED_MOVIES

cnr	fname	ename	mobile	movienr	title	genre	runtime	age_limit
1	Erik	Olsson	0730955565	1256, 856	Ouija, Unbroken	Horror, Drama	89, 137	18, 15
2	Sven	Eriksson	0730866635	857	Unbroken	Drama	137	15
3	Maria	Ek	0733359650	400	Duplex	Komedi	89	7

The table is in **UNF**. There are several cells that contain non-atomic values. If we look at customer with **cnr = 1**, we see examples of this in the cells title, genre, runtime and age limit. This will not work in a relational database.

1NF

1NF, First normal form:

- *Unique rows*
- *Atomic values in cells*

RENTED_MOVIES

cnr	fname	ename	mobile	movienr	title	genre	runtime	age_limit
1	Erik	Olsson	0730955565	1256	Ouija	Horror	89	18
1	Erik	Olsson	0730955565	856	Unbroken	Drama	137	15
2	Sven	Eriksson	0730866635	857	Unbroken	Drama	137	15
3	Maria	Ek	0733359650	400	Duplex	Komedi	89	7

The table now meets the requirements for INF. All cells contain atomic values and each row is unique. But there are still problems:

1. **Physical redundancy,** e.g. is Erik Olsson's mobile number is repeated in more the one cell.
2. **Hard to see what the table means.** Does it contain information about customers? Movies? Rented movies? It will be a problem to give the table a good name.

FD, Functional dependency

X -> Y

" Y is functionally dependent on X "

" X determines Y "

" X is a determinant, it determines Y "

" **On each row**, in the table Car, where regnr = 'FND770', it will always say 'dark green' in the color column".

Examples of some common determinants:

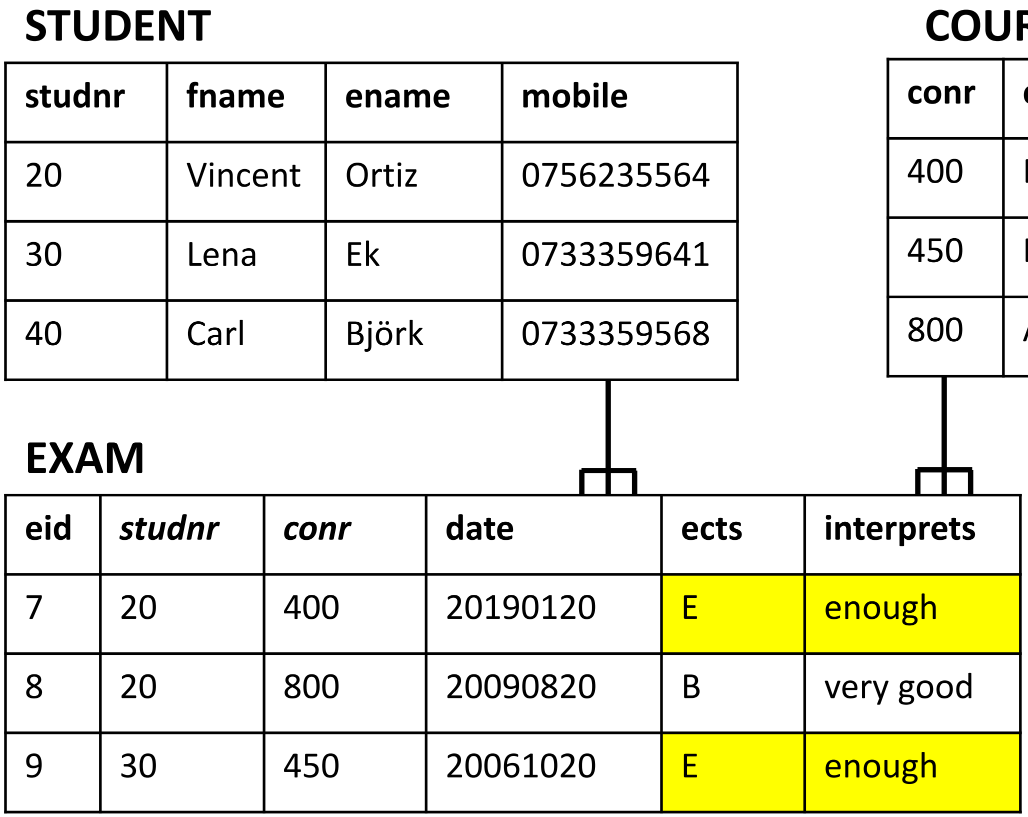
```
social security number -> first name
zip code -> city
```

```
order number -> order date
ISBN -> book title
```

2NF

2NF, Second normal form:

- *INF*
- *Each non-key attribute should be dependent on the entire PK*



STUDENT

studnr	fname	ename	mobile
20	Vincent	Ortiz	0756235564
30	Lena	Ek	0733359641
40	Carl	Björk	0733359568

COURSE

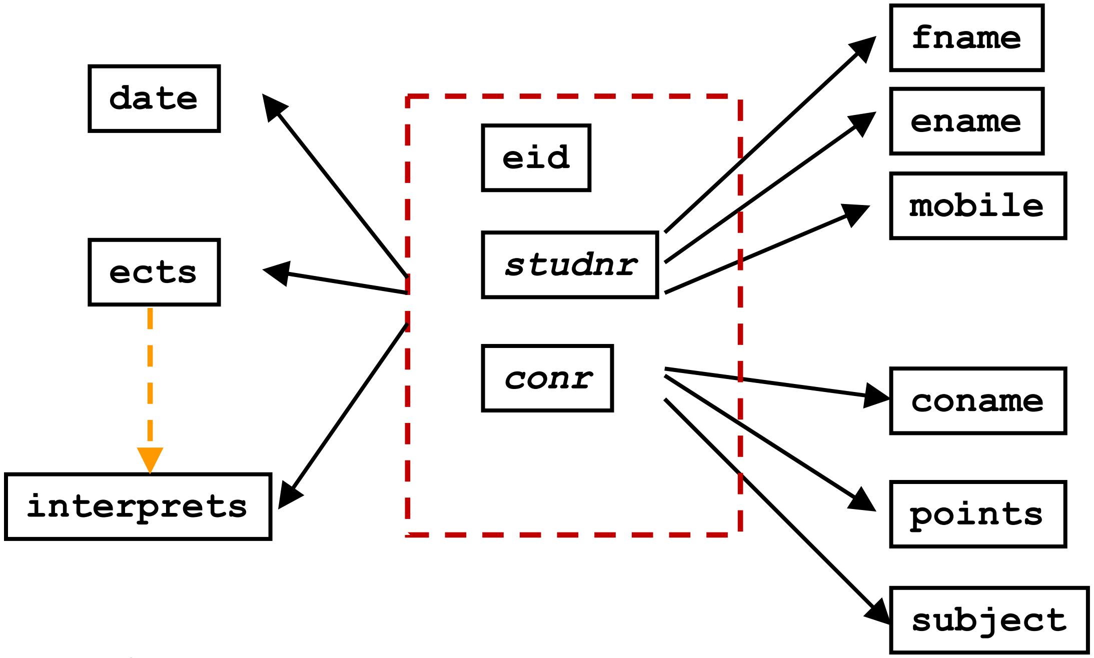
conr	coname	points	subject
400	Beginning C#	7.5	Informatics
450	Databases	7.5	Informatics
800	Accounting	15	Economy

EXAM

eid	*studnr*	*conr*	date	ects	interprets
7	20	400	20190120	E	enough
8	20	800	20090820	B	very good
9	30	450	20061020	E	enough

FD - 2NF

We draw the functional dependencies found in the previous example of 2NF:



Definition of 3NF:

- *2NF*
- *Must not contain any interdependencies between non-key attributes (transitive dependencies)*

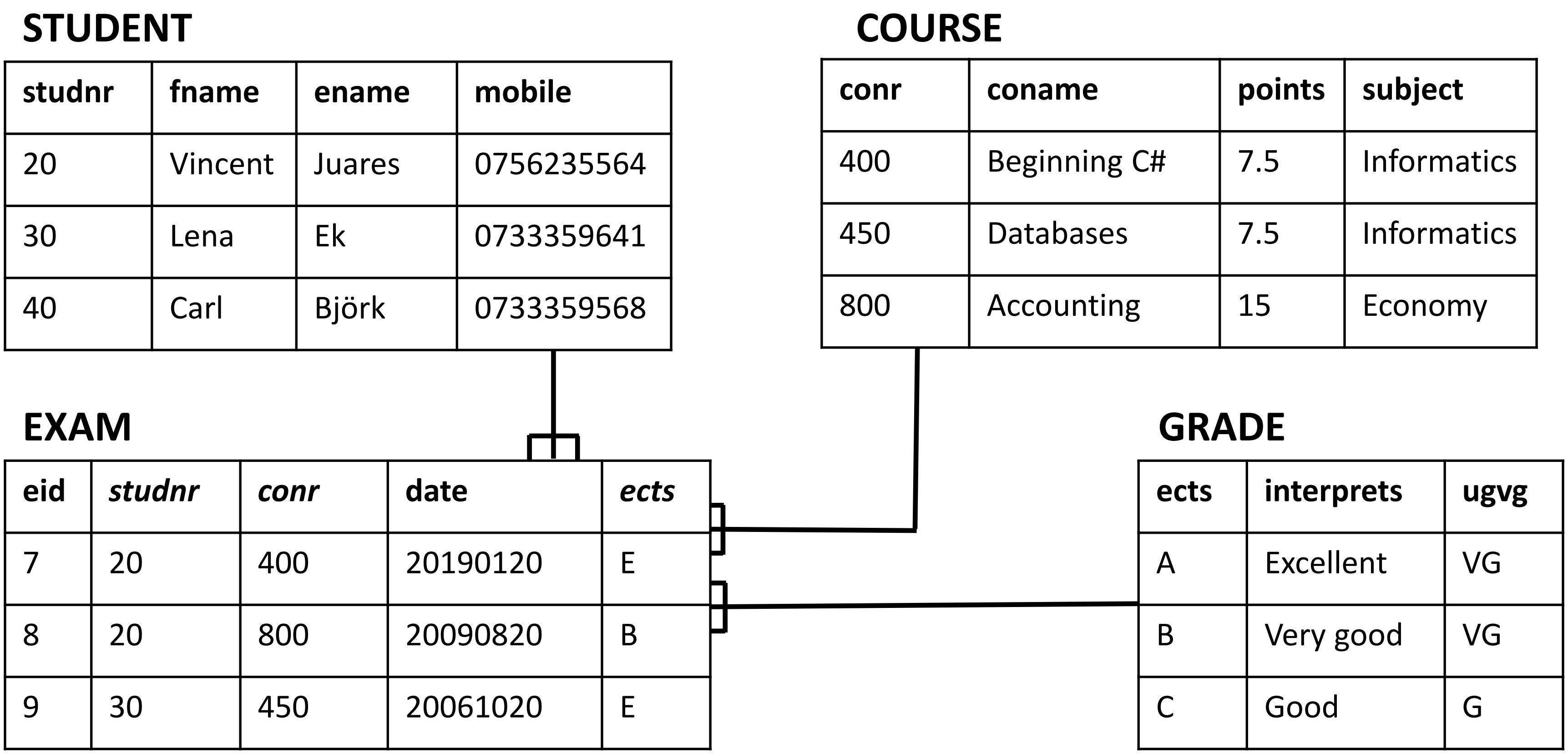
ects → interprets

In order to fulfill 3NF, the dependency must be removed!

3NF

3NF, Third normal form:

- *2NF*
- *Must not contain any interdependencies between non-key attributes (transitive dependencies)*



STUDENT

studnr	fname	ename	mobile
20	Vincent	Juares	0756235564
30	Lena	Ek	0733359641
40	Carl	Björk	0733359568

COURSE

conr	coname	points	subject
400	Beginning C#	7.5	Informatics
450	Databases	7.5	Informatics
800	Accounting	15	Economy

EXAM

eid	*studnr*	*conr*	date	*ects*
7	20	400	20190120	E
8	20	800	20090820	B
9	30	450	20061020	E

GRADE

ects	interprets	ugvg
A	Excellent	VG
B	Very good	VG
C	Good	G

Growing sideways

When creating a data model, we must ensure that tables do not need to grow sideways. That is we should not need to add columns during the operation of the system. Study the example below:

DISTRIBUTOR

distnr	name	contact1	contact2
20	ICA	Urban Karlsson	Roger Nyberg
30	COOP	Maria Ekholm	

If more contacts are added, then we need to add new columns to the table. This is a poor construction.

Growing sideways

We construct the data model so that we add new rows, instead of columns. We should be able to solve the problem with DML (insert) and not with DDL (alter).

DISTRIBUTOR

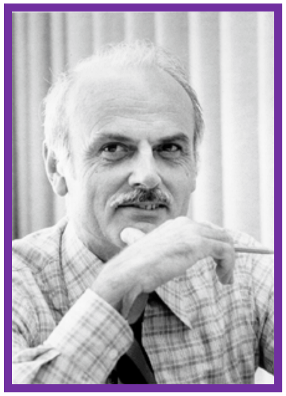
distnr	name
20	ICA
30	COOP

CONTACT

contactnr	*distnr*	contact
1	20	Urban Karlsson
2	20	Roger Nyberg
3	30	Maria Ekholm

If there is a need to add more contacts, we can solve this by adding new data in the Contact table.

Conclusions on normalization



- Make sure your data model always meets 3NF.
- This means that a table should store information about one "thing", e.g. *customers*, *vehicles*, *invoices*, *offers* etc.
- All columns that are not PK or FK should describe this "thing" and nothing else.
- This will make the data model flexible. That is it can easily be adapted to new situations.
- It will be easy to give the tables good names.
- The data model becomes clear and easy to understand.

Data models

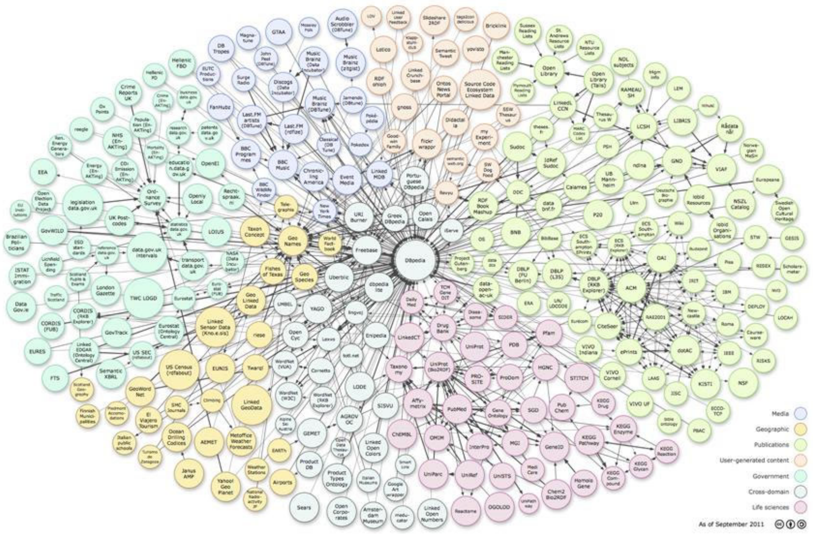
We continue with data models

Conceptual models (repetition)

One should start the *infological* part of the system development work by modeling the piece of reality that IS (the information system) will be about, and the business (activity) that IS will support.

This reality and business we also call the object system.

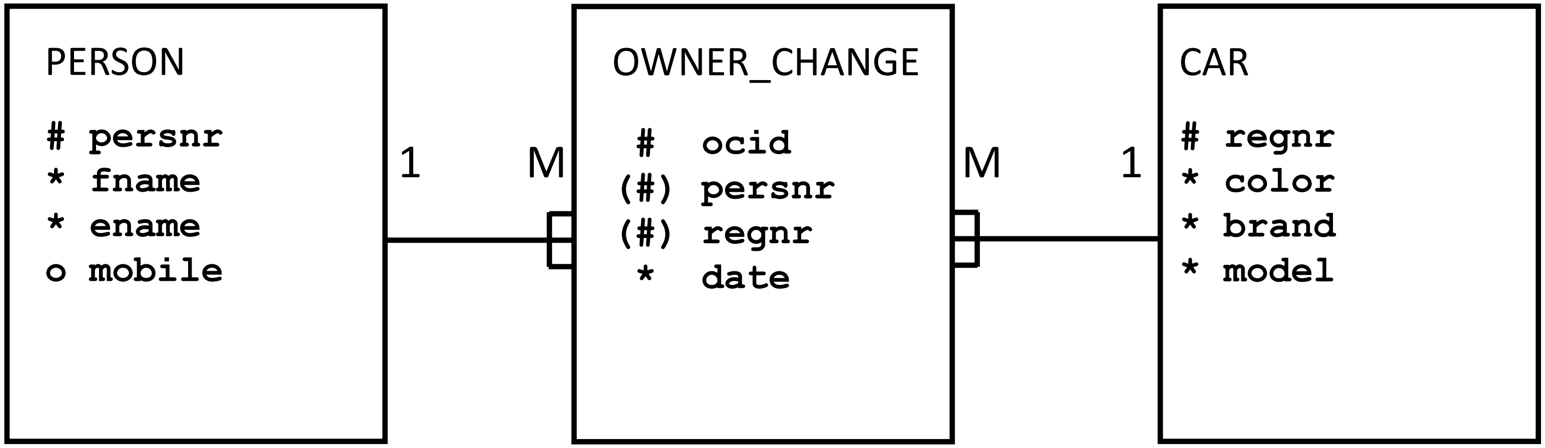
The Object system = Reality of interest = Universe of Discourse



The user's view of reality

The *object system* or *Universe of Discourse* is the **user's view** of a *"piece of"* reality.

Repetition



PERSON

# persnr
* fname
* ename
o mobile

1

M

OWNER_CHANGE

# ocid
(#) persnr
(#) regnr
* date

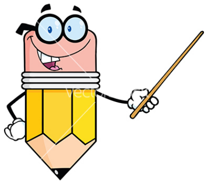
M

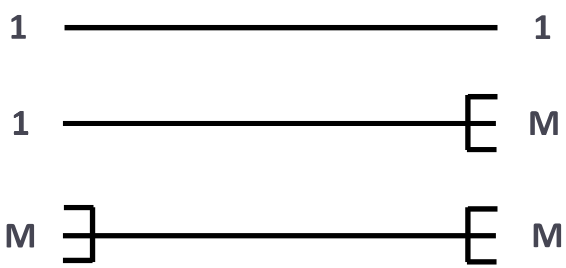
1

CAR

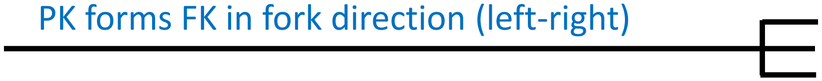
# regnr
* color
* brand
* model

# Primary key
(#) Foreign key
* Mandatory
o Optional

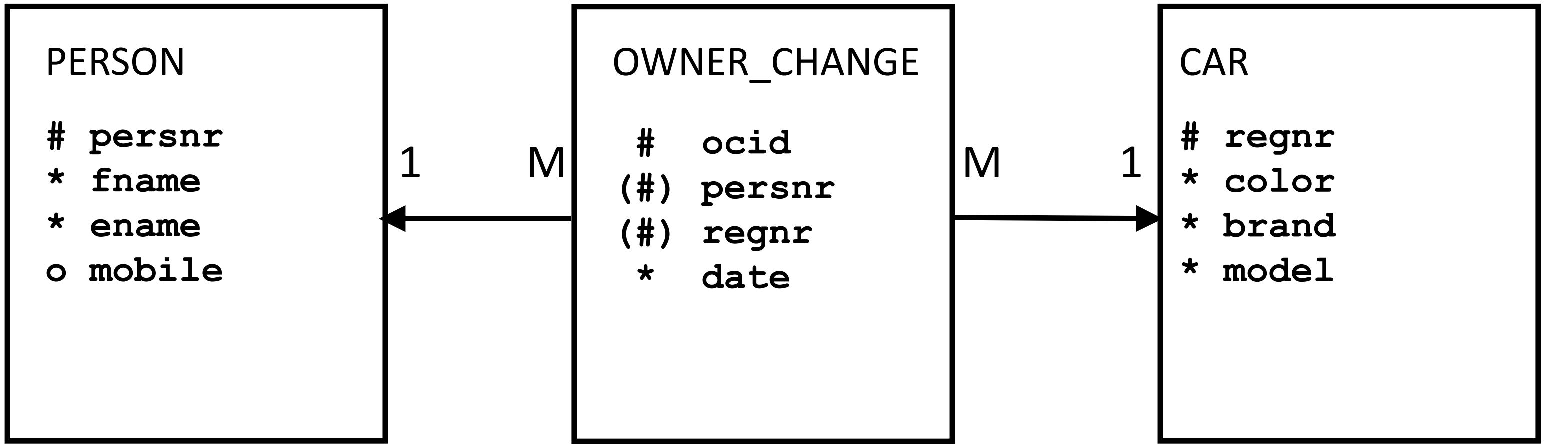




PK forms FK in fork direction (left-right)



Alternative notation



PERSON

```
# persnr
* fname
* ename
o mobile
```

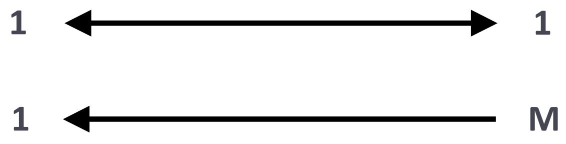
OWNER_CHANGE

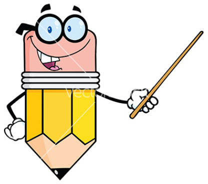
```
 #   ocid
(#)  persnr
(#)  regnr
 *   date
```

CAR

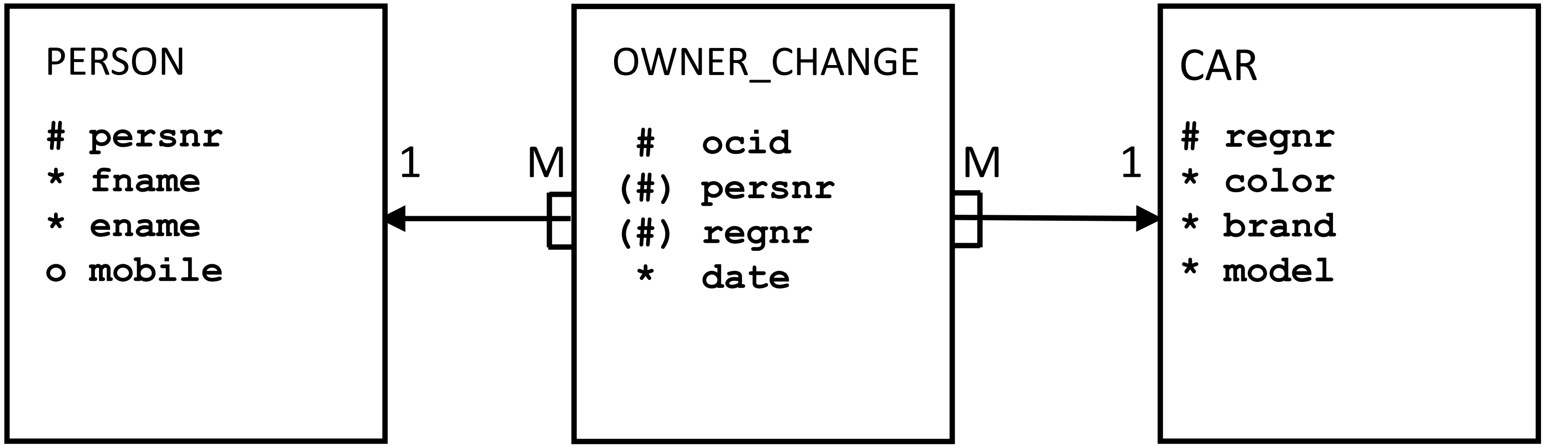
```
# regnr
* color
* brand
* model
```

Arrows and forks are used in the opposite way. An arrow is used to point out where an FK comes from, i.e. where it has its parent table (PK).





Alternative notation



```
PERSON

# persnr
* fname
* ename
o mobile
```

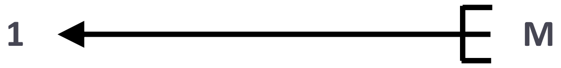
1 M

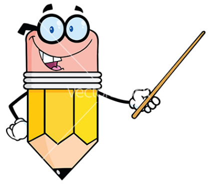
```
OWNER_CHANGE

 #   ocid
(#)  persnr
(#)  regnr
 *   date
```

M 1

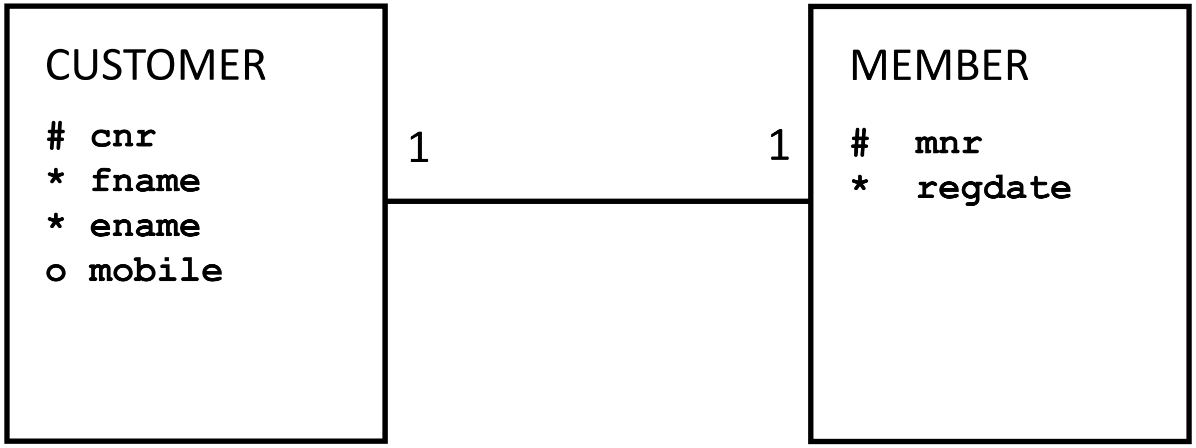
```
CAR

# regnr
* color
* brand
* model
```

Often, **both notation** methods are usually combined for the best clarity.





1:1



CUSTOMER

```
# cnr
* fname
* ename
o mobile
```

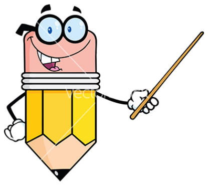
1

1

MEMBER

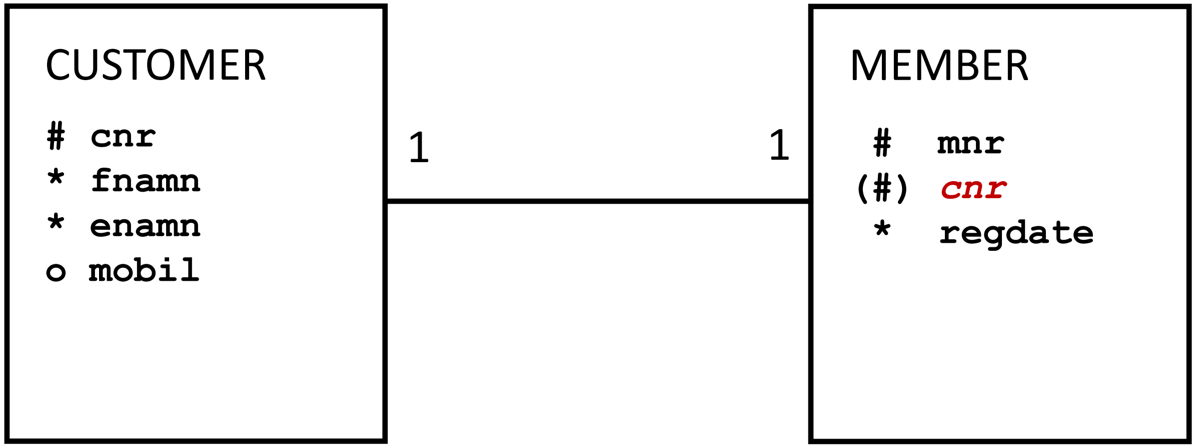
```
#   mnr
*   regdate
```

A customer can have a membership. A membership can only belong to one customer. This is an example of a 1:1 relation.



In which table should we put the FK? Should you place the FK in the Member table? Or should you place the FK in the Customer table?

Avoid creating NULL values



CUSTOMER

# cnr
* fnamn
* enamn
o mobil

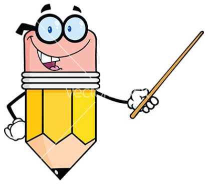
1

1

MEMBER

# mnr
(#) *cnr*
* regdate

We can choose whichever we want, but we should avoid creating unnecessary NULL values. If we enter mnr as FK in the Customer table, then every customer who does not want to become a member will receive a NULL value in the mnr column.



In this case, therefore, the best solution is to create the FK cnr in the Member table. In this way we do not get NULL values. This is because all members created must have a reference to a customer.

Different methods

Different methods for creating data models:

1 Top Down (OPR Framework)

- Identify important *entities*, *concepts* or *business objects* and their relationships. M:M relationships form *new entities*.

2 Action-oriented Conceptual modelling

- In this model the communicative objects are identified as own entities and not just a M:M relation. These objects are usually referred to as institutional objects. Examples of institutional objects can be ***order confirmation***, ***different offers***, ***a request to get paid*** (invoice).
- Sometimes you approach this way of thinking in the OPR model by saying that you *objectify a relationship*.

3 Bottom Up

- We can study functional dependencies between different attributes.

Common

Common to all models is:

- **Concept**- and **attribute catalog** are usually used to document the model.
- In the concept catalog, the concepts are documented. We give the concepts good names that everyone in the business knows. We describe the purpose of the concept in our system. *Why does this specific table exist? What is the purpose?*
- In the term catalog we document all the properties or attributes of the business objects. We decide which attributes should identify the objects.
- We decide which attributes are required to designate (describe the objects) in a good enough way for the business to function.
- Describe what function the attributes have. Why is this attribute included in our table?
- This documentation forms the basis for a *common business language*. This is very important!

Concept catalog

Examples of what it might look like:

Concept (Table)	Definition	Identifier	System
Customer	The table stores information about customers.	cust_id	eSales
Customer_order	A customer order means that a customer can order items at different times.	order_id	eSales
Invoice	A requirement to get paid by a customer for items that have been delivered to the customer	invoice_id	eSales

All IT systems must have a name!

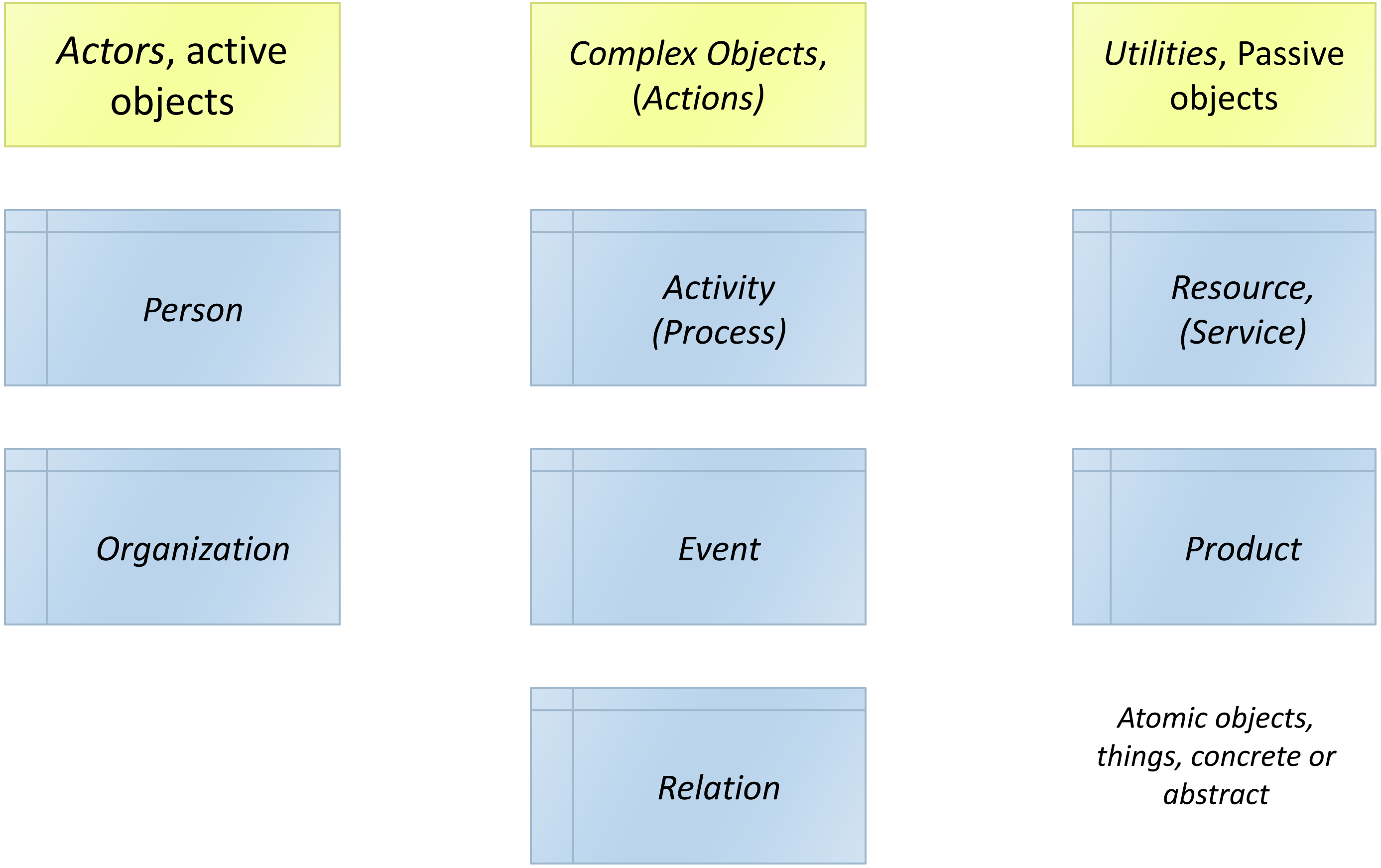
Attribute catalog

Examples of what it might look like:

Attribute (Column)	Definition	Identifier	Concept (Table)
custorder_id	Used to uniquely identify a customer order.	yes	Customer_order
cnr	Describes which customer owns the customer order.	no	Customer_order
orderdate	Describes when the customer order was created. Year, day and time.	no	Customer_order
cnr	Used to uniquely identify a customer.	yes	Customer

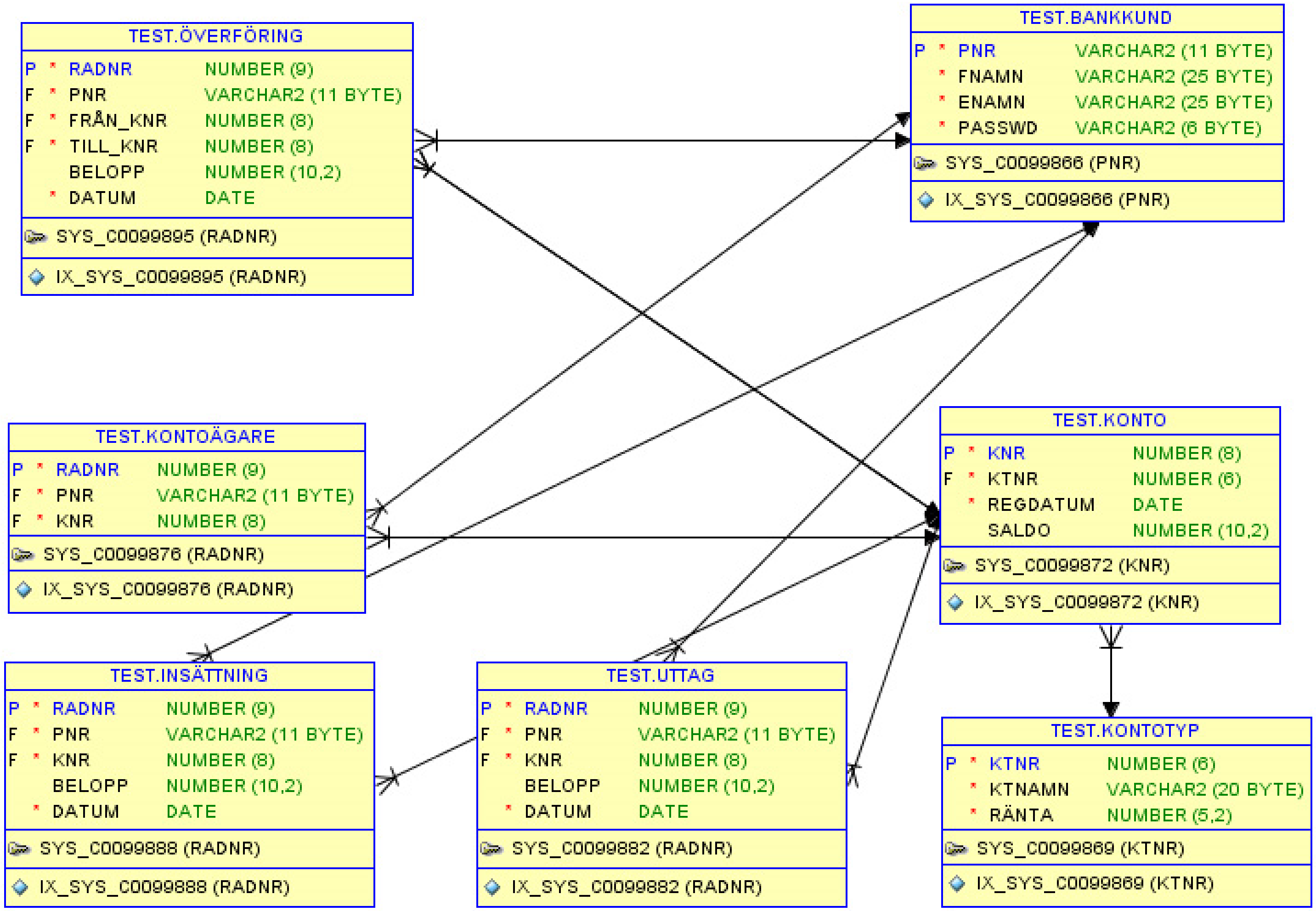
Draw clear models

Proposal from Bo Sundgren:



Actors, active objects	*Complex Objects, (Actions)*	*Utilities*, Passive objects
Person	*Activity (Process)*	*Resource, (Service)*
Organization	*Event*	*Product*
	Relation	*Atomic objects, things, concrete or abstract*

Example of a bad model



TEST.ÖVERFÖRING

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	FRÅN_KNR	NUMBER (8)
F	*	TILL_KNR	NUMBER (8)
		BELOPP	NUMBER (10,2)
	*	DATUM	DATE

SYS_C0099895 (RADNR)

IX_SYS_C0099895 (RADNR)

TEST.BANKKUND

P	*	PNR	VARCHAR2 (11 BYTE)
	*	FNAMN	VARCHAR2 (25 BYTE)
	*	ENAMN	VARCHAR2 (25 BYTE)
	*	PASSWD	VARCHAR2 (6 BYTE)

SYS_C0099866 (PNR)

IX_SYS_C0099866 (PNR)

TEST.KONTOÄGARE

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	KNR	NUMBER (8)

SYS_C0099876 (RADNR)

IX_SYS_C0099876 (RADNR)

TEST.KONTO

P	*	KNR	NUMBER (8)
F	*	KTNR	NUMBER (6)
	*	REGDATUM	DATE
		SALDO	NUMBER (10,2)

SYS_C0099872 (KNR)

IX_SYS_C0099872 (KNR)

TEST.INSÄTTNING

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	KNR	NUMBER (8)
		BELOPP	NUMBER (10,2)
	*	DATUM	DATE

SYS_C0099888 (RADNR)

IX_SYS_C0099888 (RADNR)

TEST.UTTAG

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	KNR	NUMBER (8)
		BELOPP	NUMBER (10,2)
	*	DATUM	DATE

SYS_C0099882 (RADNR)

IX_SYS_C0099882 (RADNR)

TEST.KONTOTYP

P	*	KTNR	NUMBER (6)
	*	KTNAMN	VARCHAR2 (20 BYTE)
	*	RÄNTA	NUMBER (5,2)

SYS_C0099869 (KTNR)

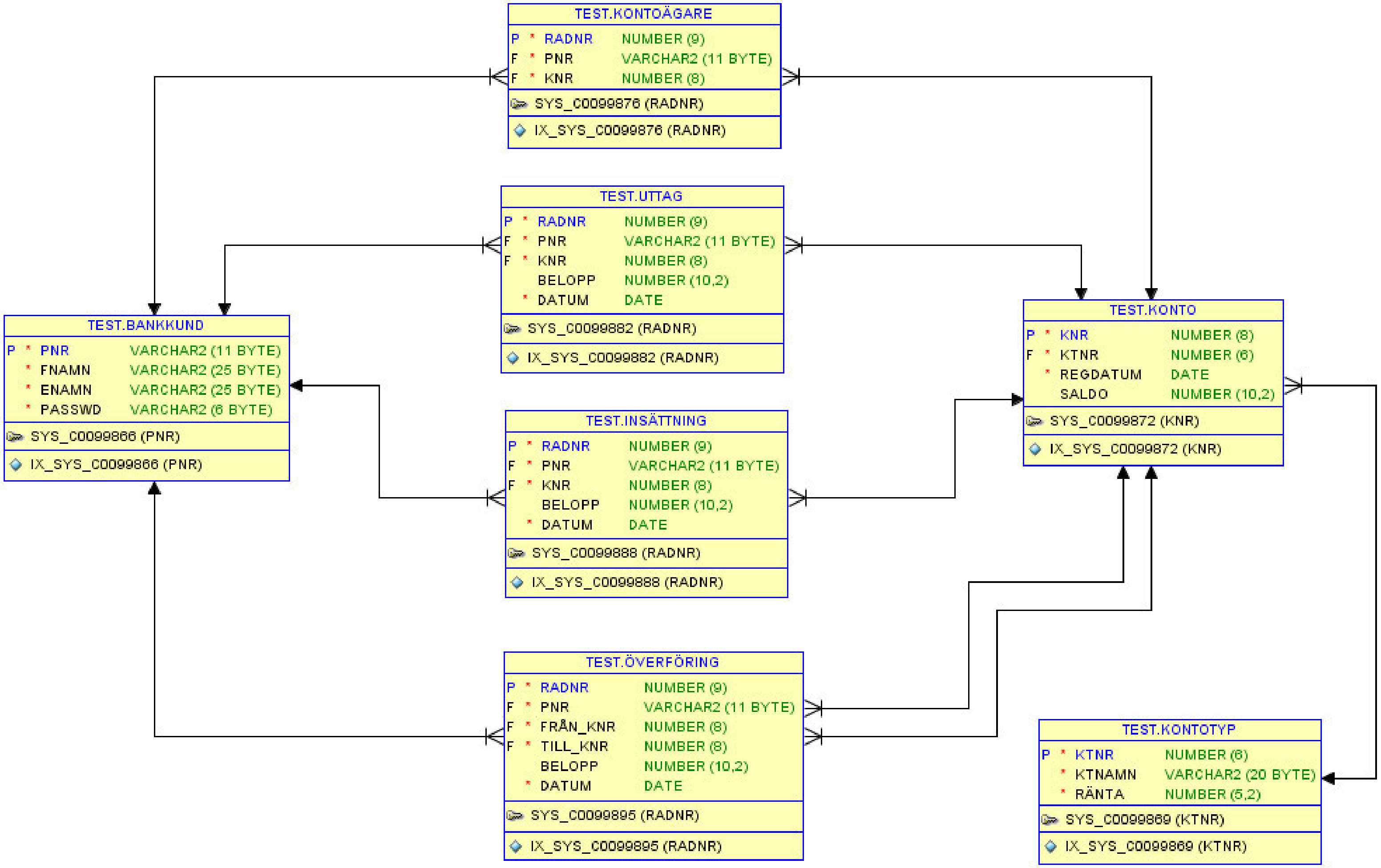
IX_SYS_C0099869 (KTNR)

An example of a good model

Active objects

Complex objects

Passive objects



TEST.KONTOÄGARE

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	KNR	NUMBER (8)

SYS_C0099876 (RADNR)

IX_SYS_C0099876 (RADNR)

TEST.UTTAG

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	KNR	NUMBER (8)
		BELOPP	NUMBER (10,2)
	*	DATUM	DATE

SYS_C0099882 (RADNR)

IX_SYS_C0099882 (RADNR)

TEST.BANKKUND

P	*	PNR	VARCHAR2 (11 BYTE)
	*	FNAMN	VARCHAR2 (25 BYTE)
	*	ENAMN	VARCHAR2 (25 BYTE)
	*	PASSWD	VARCHAR2 (6 BYTE)

SYS_C0099866 (PNR)

IX_SYS_C0099866 (PNR)

TEST.INSÄTTNING

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	KNR	NUMBER (8)
		BELOPP	NUMBER (10,2)
	*	DATUM	DATE

SYS_C0099888 (RADNR)

IX_SYS_C0099888 (RADNR)

TEST.KONTO

P	*	KNR	NUMBER (8)
F	*	KTNR	NUMBER (6)
	*	REGDATUM	DATE
		SALDO	NUMBER (10,2)

SYS_C0099872 (KNR)

IX_SYS_C0099872 (KNR)

TEST.ÖVERFÖRING

P	*	RADNR	NUMBER (9)
F	*	PNR	VARCHAR2 (11 BYTE)
F	*	FRÅN_KNR	NUMBER (8)
F	*	TILL_KNR	NUMBER (8)
		BELOPP	NUMBER (10,2)
	*	DATUM	DATE

SYS_C0099895 (RADNR)

IX_SYS_C0099895 (RADNR)

TEST.KONTOTYP

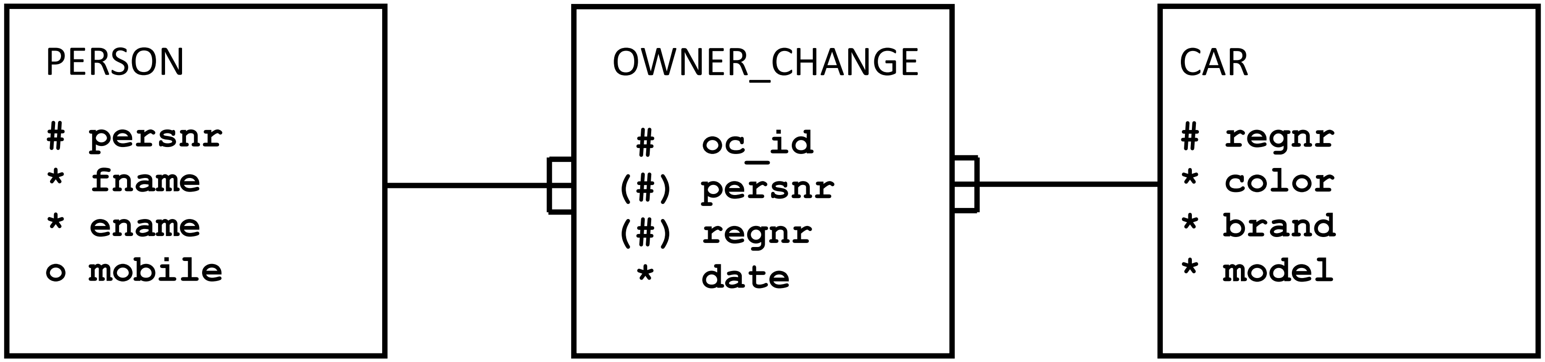
P	*	KTNR	NUMBER (6)
	*	KTNAMN	VARCHAR2 (20 BYTE)
	*	RÄNTA	NUMBER (5,2)

SYS_C0099869 (KTNR)

IX_SYS_C0099869 (KTNR)

Example

We will now study how to arrive at the following example model using the three methods.

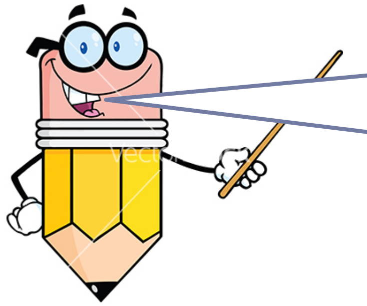


```
PERSON

# persnr
* fname
* ename
o mobile
```

```
OWNER_CHANGE

 #   oc_id
(#)  persnr
(#)  regnr
 *   date
```

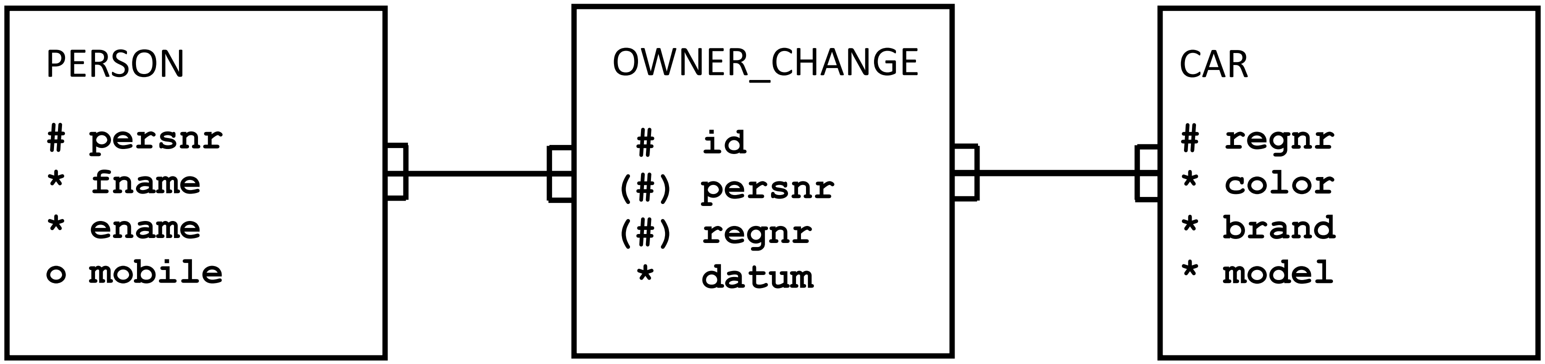
```
CAR

# regnr
* color
* brand
* model
```

Business description:



A person can be a registered owner of several vehicles. A vehicle can, if we look at time as a time interval, be owned by several people. When a change of ownership occurs, this is registered by an official at the Swedish Transport Administration.

Example of Top Down



PERSON

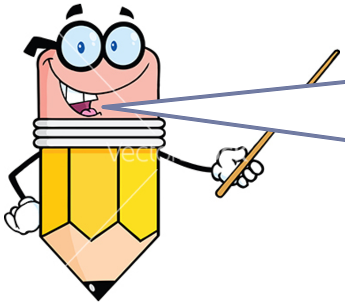
```
# persnr
* fname
* ename
o mobile
```

OWNER_CHANGE

```
 #   id
(#)  persnr
(#)  regnr
 *   datum
```

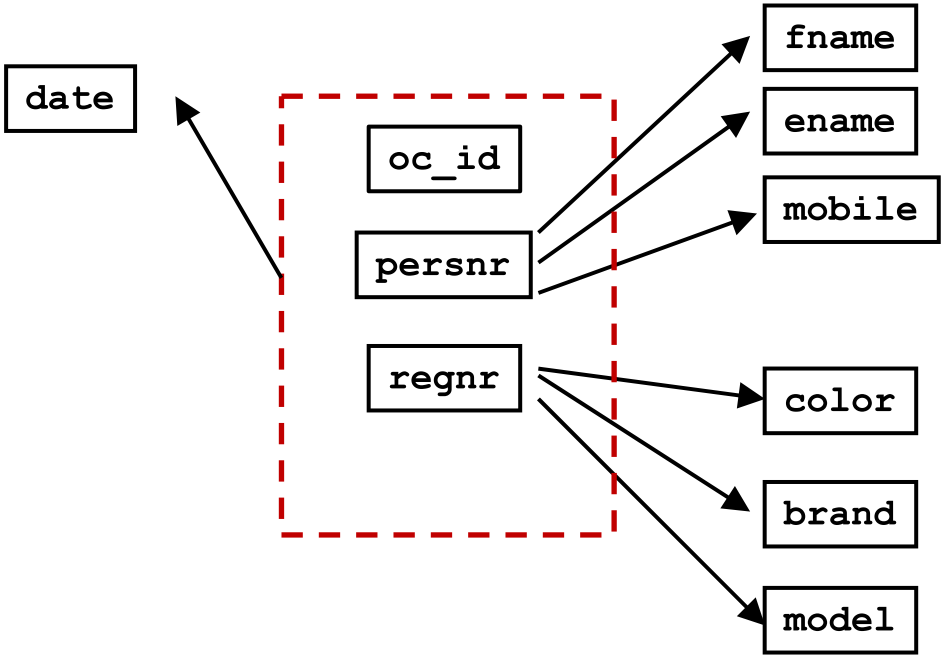
CAR

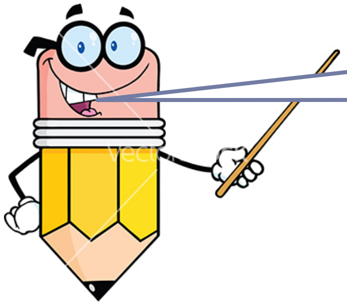
```
# regnr
* color
* brand
* model
```



We rapidly identify the objects Person and Vehicle. Furthermore, we find that the Person-Car relationship is of the type M:M. This means we must create a new table to remove the M:M relationship. PK forms FK in the direction of the fork: Person -> Owner change, Car -> Owner change. PK -> FK.

Example of functional dependencies

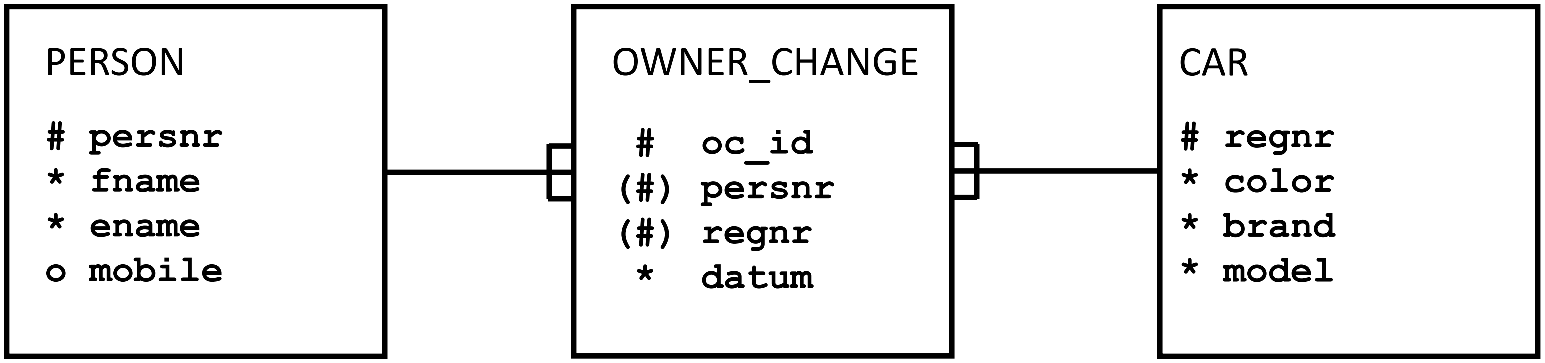




Three determinants result in three tables.

Action-oriented

By reading the business description, we see that the ***act of registering ownership change*** is performed by an official at the Swedish Transport Administration.



PERSON

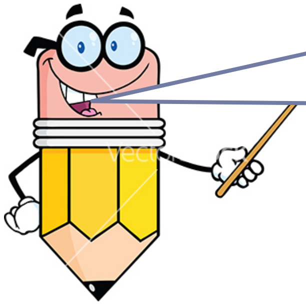
```
# persnr
* fname
* ename
o mobile
```

OWNER_CHANGE

```
 #   oc_id
(#)  persnr
(#)  regnr
 *   datum
```

CAR

```
# regnr
* color
* brand
* model
```



When we create a new instance of the Owner change class, i.e. when we make an insert in the table Owner change it means that an owner change is registered. This change of ownership means that a certain car gets a new owner. We need references (FK) to both a car and a person as well as a time when the exchange took place in order to create an owner_change object.

The End