Zawansowane Modelowanie i Analiza Systemów Informatycznych

(l-6)

Polsko-Japońska Wyższa Szkoła Technik Komputerowych
Katedra Systemów Informacyjnych

2013

Overview

Transformation procedure of ORM schema to the RDB schema

Relational schema design

An ORM conceptual schema can be mapped into a relational database schema by the mapping process (sometimes also called ORM to RDB transformation).

The mapping provides relational schema design in <u>Optimal Normal Form</u>: a set of normalised tables to store the information permitted by the conceptual schema.

Recommendation: review your knowledge about Normalization of RDB: 1, 2, 3, BCNF, 4 and 5 Normal forms.

Relational schema design

<u>Input:</u> ORM conceptual schema

<u>Output:</u> the list of

- Table names
- For each table the list of its column names
- The list of columns that form table key(s)
- Identification of foreign keys.

Alternative terminology

Table	Relation
Column	Attribute

Names construction

- **By an attribute in RDB terminology we understand an entity type (simple) combined with a role which it plays in a fact type,**

- **The names of attributes are (should be) typically derived from the semantics of the roles the entity types play in the fact types,**

- **The names of the relations are generally determined by the combinations of key attributes:**
  - Multiple roles UC – Name of the fact type
  - Single role UC – Name of the entity type ‘touching’ that role

Outline of the RDB construction procedure

Step1

Each flat (non-nested) fact type generates a relation. The uniqueness constraint (UC) of the fact type is the key of the relation,

Step 2

If a nested fact type plays a role in a non-nested fact type than it is represented in the relation schema by all attributes ‘contributing’ to this nested fact type (possibly recursively),

Step 3

Two relations with the same sets of keys should be combined into one relation,

Step 4

Consider special cases (1:1 realtionship and subtyping)

Illustration of the RDB design procedure

On the next slides, for the sake of simplicity, we assume that the names of entity types involved in fact types correspond to their roles in these fact types.

Notation

Keys are indicated by underlining involved attributes

In general

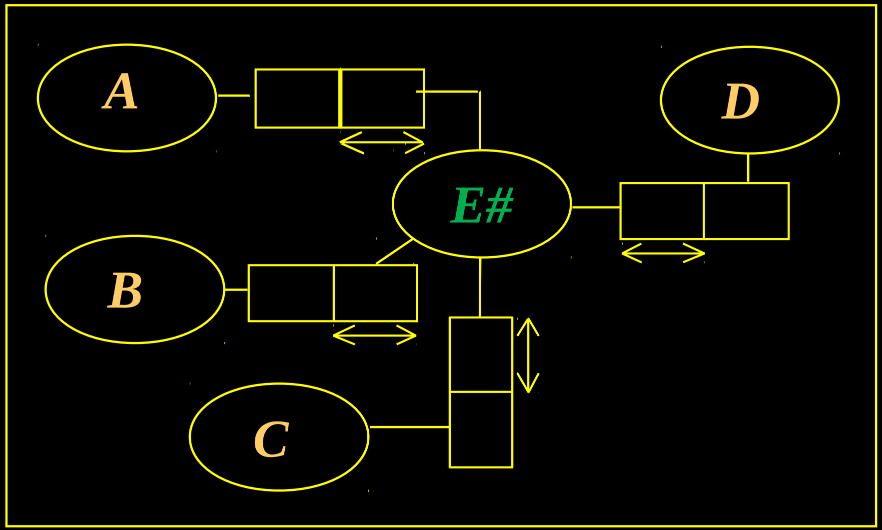
Single role uniqueness constraints

All binary fact types involving an entity type E, which have 'touching' roles with that entity type covered by single UC, contribute to the relation which has

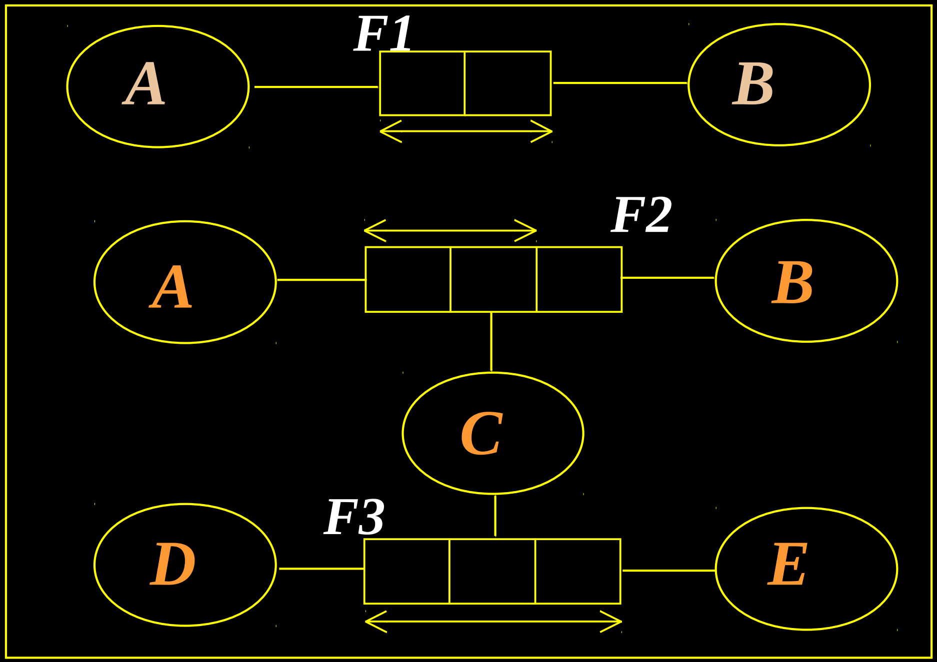
Attributes – entity types playing roles not covered by UC and identifier of E

Key - identifier of E

Name - the name of the entity type is to be considered as the name for the relation



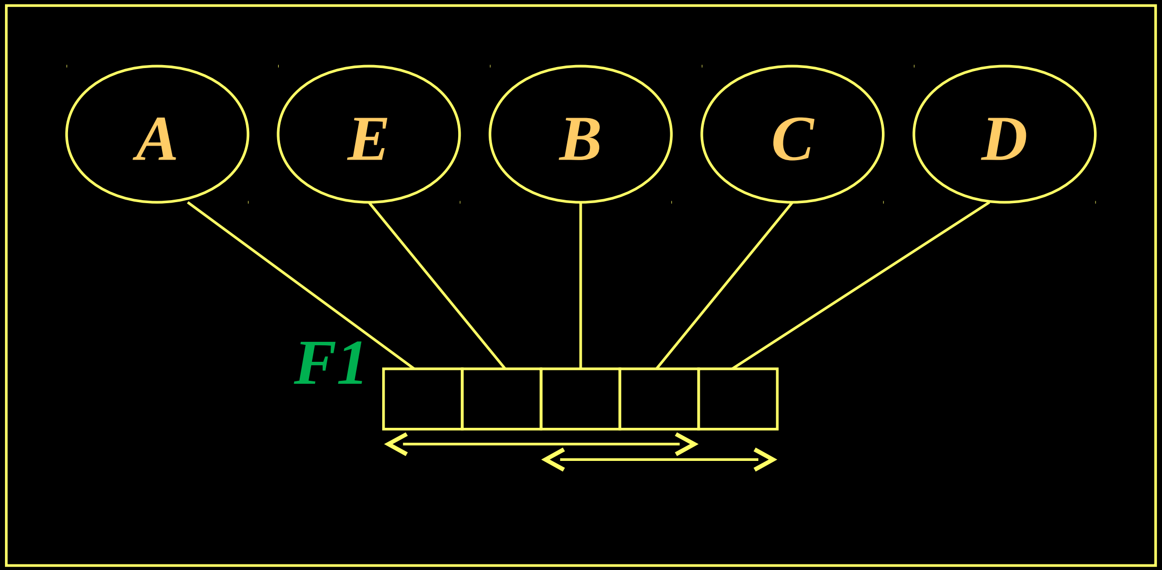
E (<u>E#</u> A B C D) Key E# (note; that we have combined 4 realtions having the same key E#, according to Step 3)



F1(AB) *Key AB*
F2(ACB) *Key AC*
F3(CDE) *Key CDE*

Multiple roles UC

If there are more uniqueness constraints, then each of them corresponds to a key of the relation.

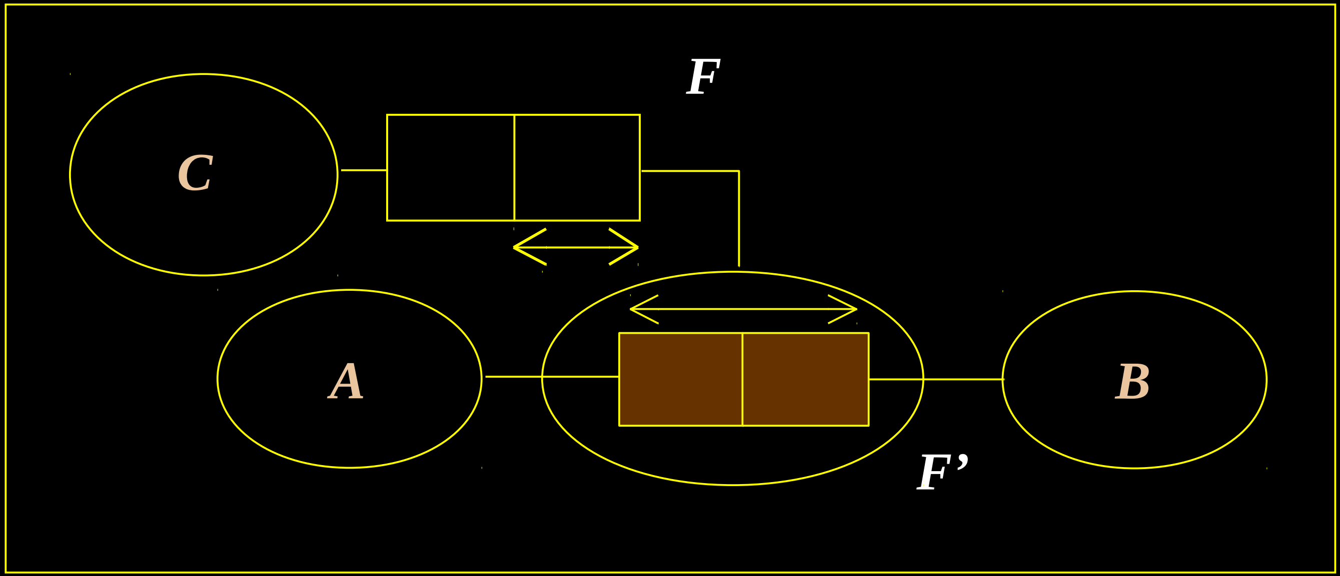


F1(ABCDE) ***Keys ABCE and BCD***

If a nested fact F' type plays a role in the flat fact type F then according to Step 2

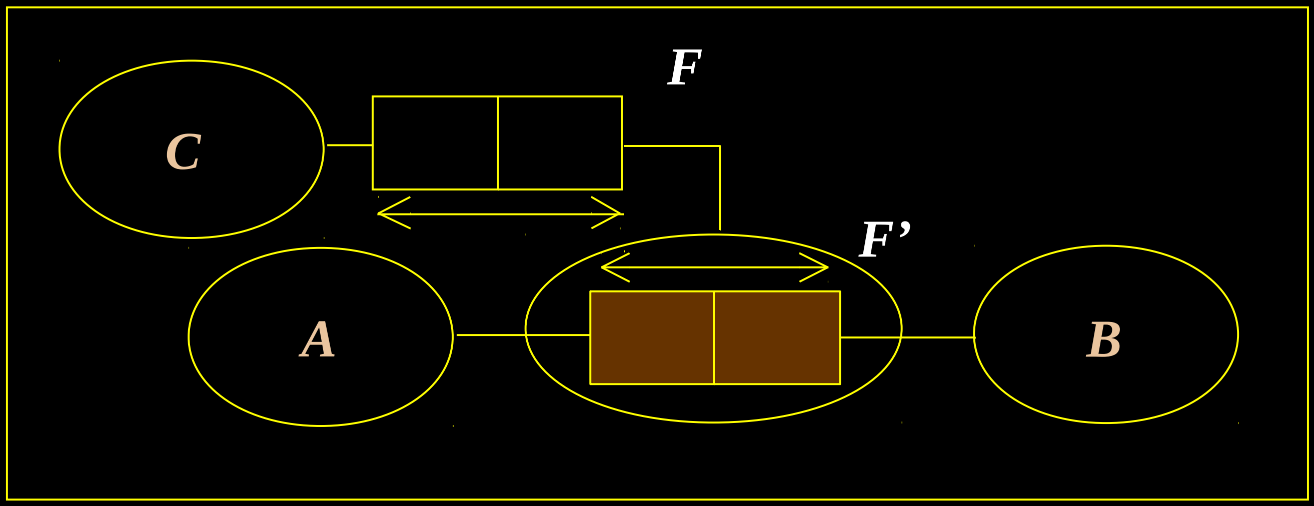
Create a relation for that flat fact type as that role would be 'played' by simple entity type. **Informally, we may use temporarily the name of that nested fact type F' (calling F'*) as one of the 'attributes' of that relation. Then by substituting F' *with the attributes generated by F' we finalise the design of the table.**

Since the nested fact type F' plays a role covered with single UC in F then the name of the relation is corresponding to F' and the key(s) of that relation are uniqueness constraints of F'



F'(F'*, C) = F'(<u>A B</u> C) ***Key AB***

If a role of nested fact type F' covered with a multiple role UC in F then the name of the generated relation corresponds to F, the key(s) of that relation are determined as concatenations of UCs of F' and uniqueness constraints of F



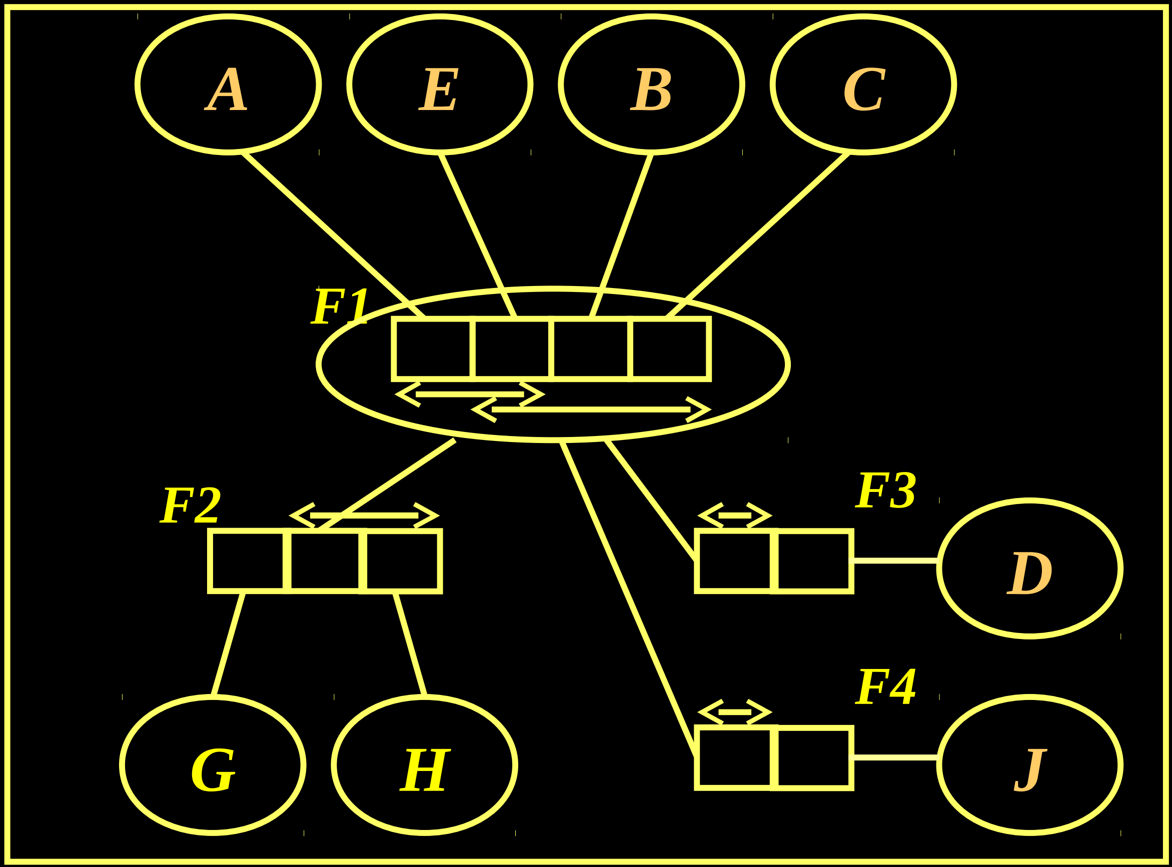
F(F'* C) = F(<u>A B C</u>)   Key ABC

One more example with useful informal step - see slide 11 yellow text (note the type problem!!!!)

F3 → F1 (F1 D) key F1*
F4 → F1 (F1 J) key F1*
Combine above into one table (the same key)
F1 (F1 D J) key F1**

F2 → F2 (F1 G H)*
with key F1 H*

After substitution of F1 with contributing attributes

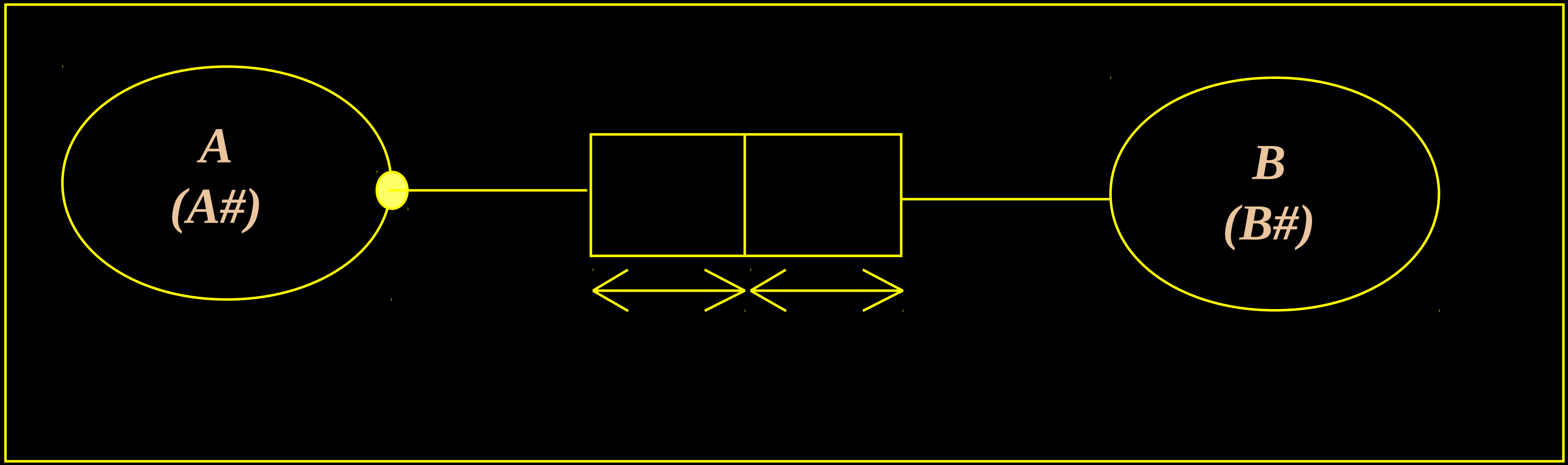


Resulting tables are

F1(A B C E D J)	*keys AE and BCE*
F2 (A B C E G H)	*keys A E H and B C E H*

Step 4 - Special cases of the transformation procedure:

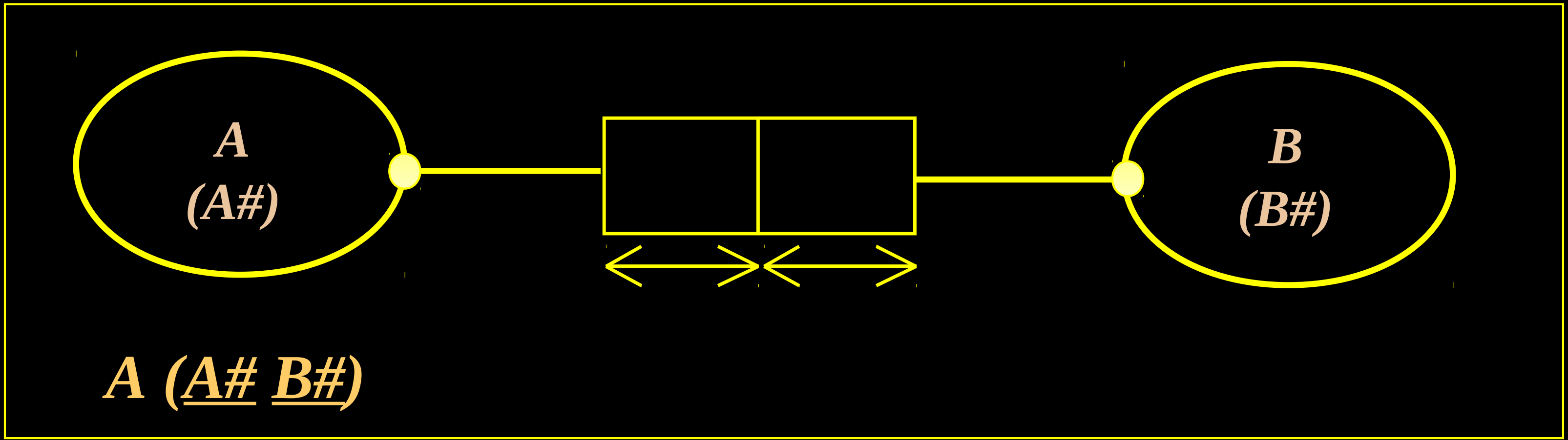
For each binary fact type with both roles marked with two UCs generate a relation with name associated with the entity type name involved in role marked with the mandatory role constratint (on figure below A).
Any other binary fact type involving such entity with UC on it contributes a column to that realtion as it would be in the case of step 3 for all binary fact types. For the illustarion see slide 19.

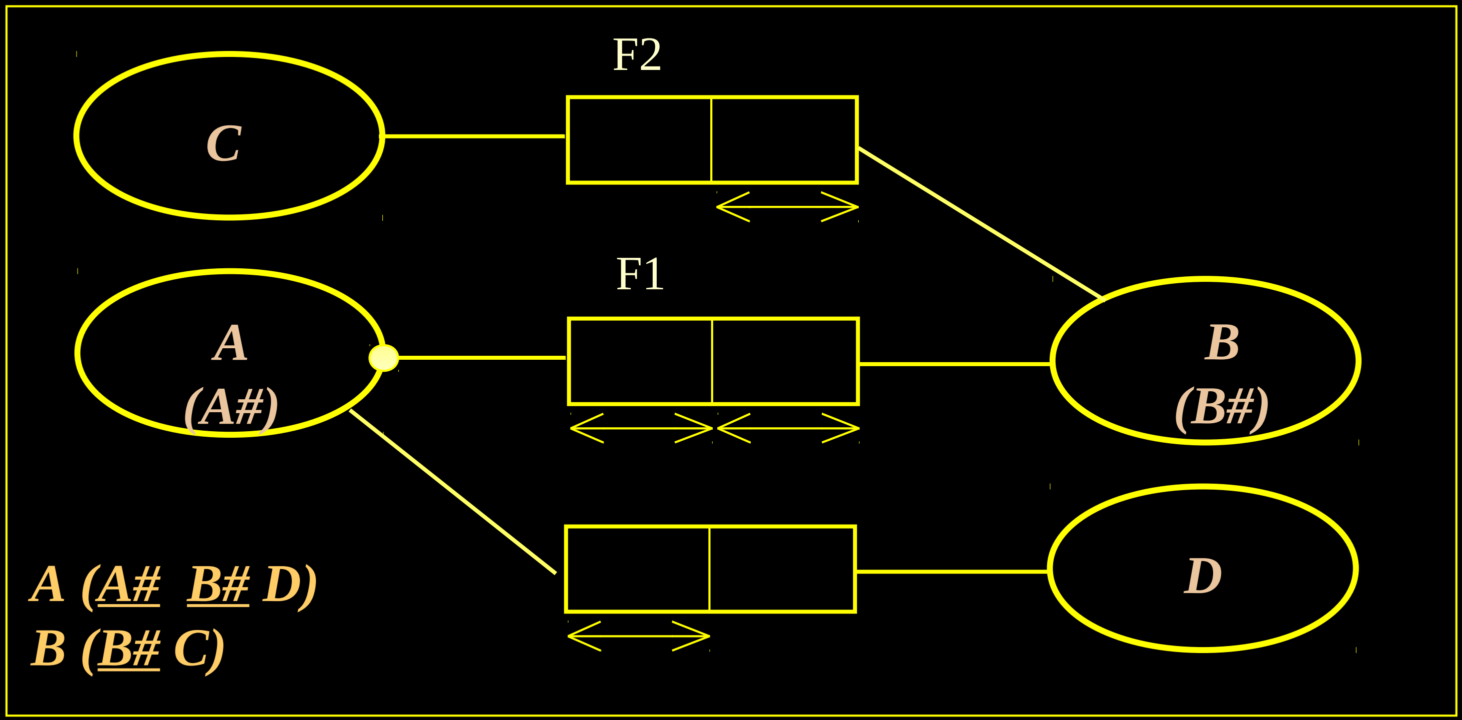


A (<u>A#</u> <u>B#</u>)

Special cases (cont):

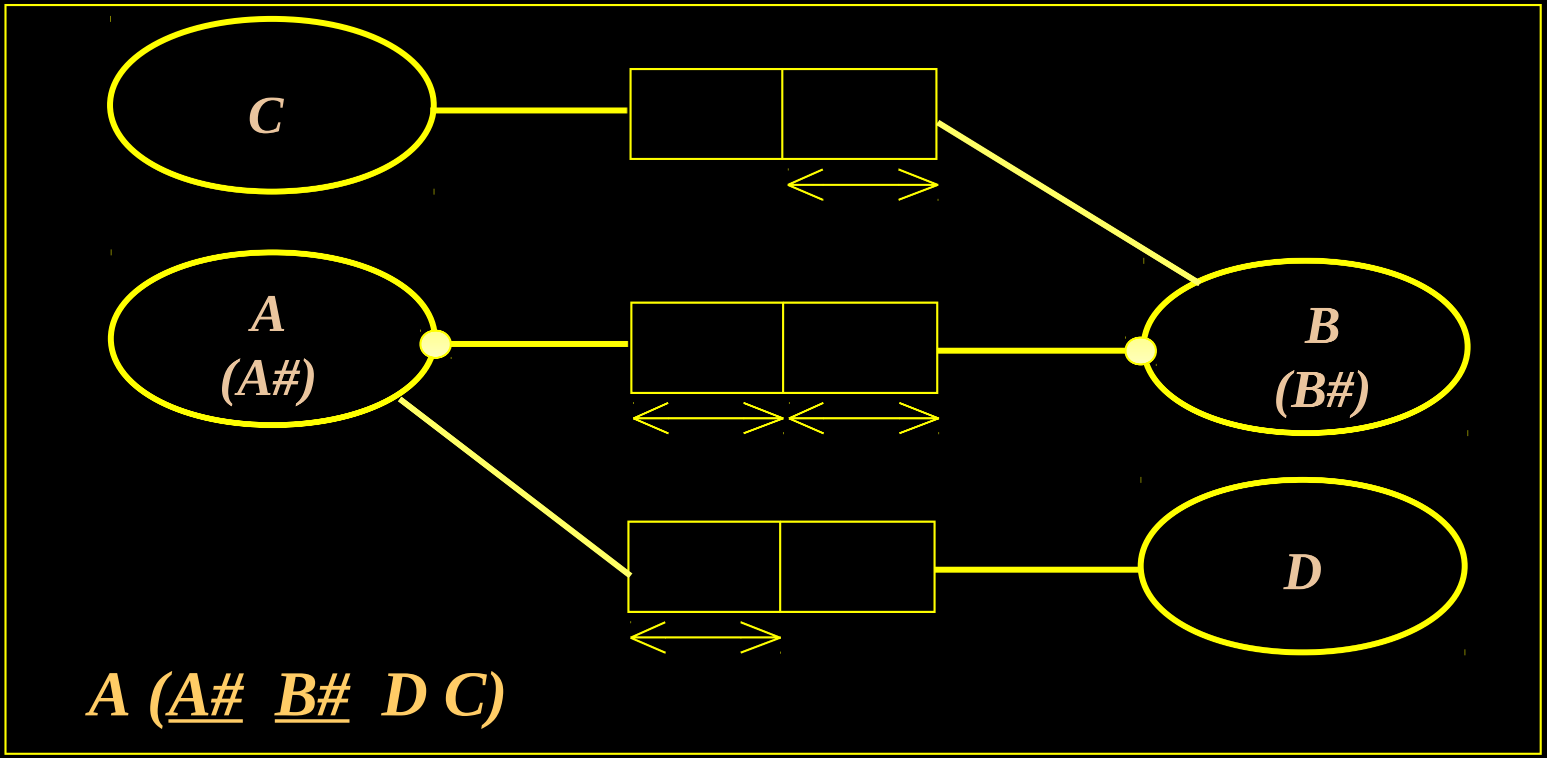
If both roles in a binary fact type are marked with a single UC, and both roles are total roles, then create a relation for this fact type. The identifiers of both entity types (A#, B#) below become the keys of the resulting relation. Any other binary fact types involving entity A or B with UC on it contributes a column to that relation. For the illustarion see slide 20.





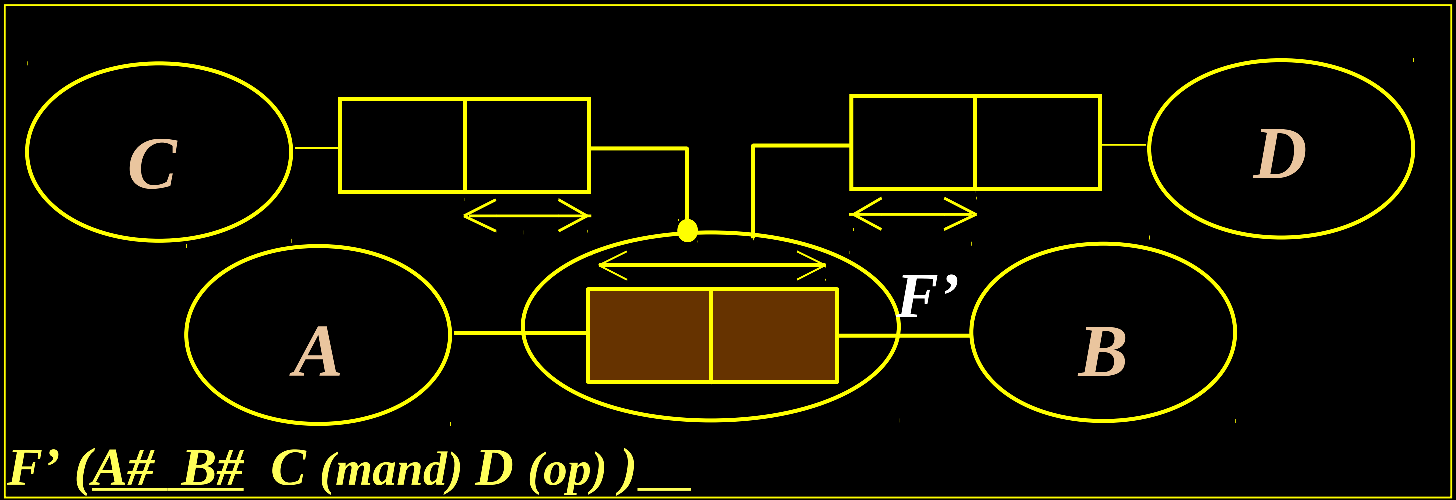
Note that the attribute B# in relation A may have different meaning than B# in relation B. Therefore, both relations are necessary.

Eg A- Department , B – Manager in (F1), B – Employee in F2. In such case B# in A is to be carry the semantics of the role of B in F1 – Manager in our example


C
A
(A#)
B
(B#)
D
A (A# B# D C)

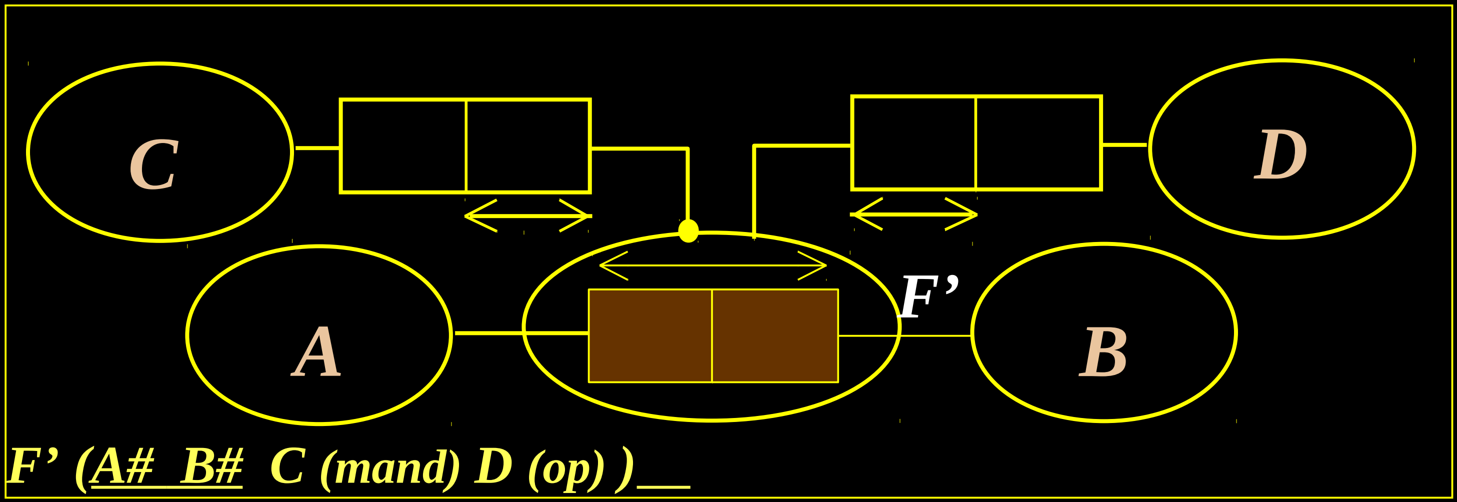
Mapping of Mandatory Roles

If an entity type (including nested fact type) is involved in another fact type through a mandatory role constraint, then (as the general principle) the resulting attribute should be declared mandatory and implemented as 'NOT NULL' . However, there could be some exceptions - in such case a satisfactory solutiom should be found.



Suppose that F' stands for Enrolment, A for Student, B – Subject, C – Final Result, D - …xxx

Each enrolment must end up in a final result and this will be known at the end of semester but the table is implemented in DB earlier. What would you suggest as a solution?



Solutions suggested:

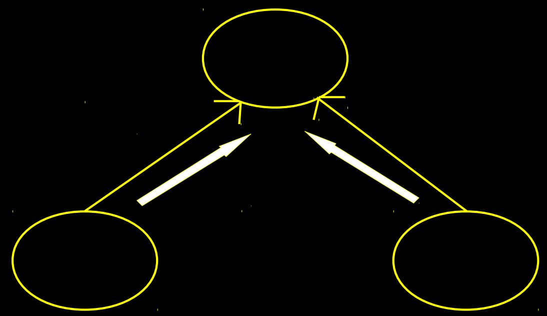
1 Do not declare column C 'not null', but remember that at the end of the semester all results are to be inserted (some additional db application program is needed)

2Declare column C 'not null', Insert some dummy value for each enrolment and at the end of the semester update it to the real final result.

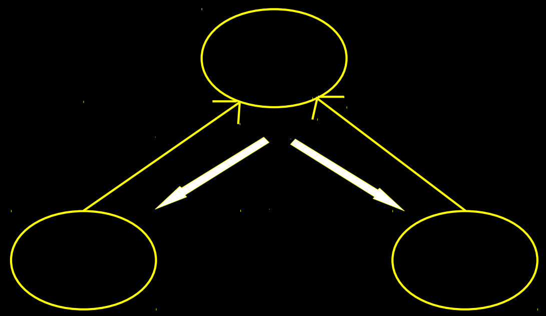
3CREATE the table without column C and ALTER the table when results are available (this is the worst solution as delays with some results can be expected and altered table does not accept 'not null' declaration for additional column anyway.

Step 4 - Special cases - subtyping

There are many ways to transform subtype structures. At one extreme, treat the subtype structure by ignoring the created subtype construction.

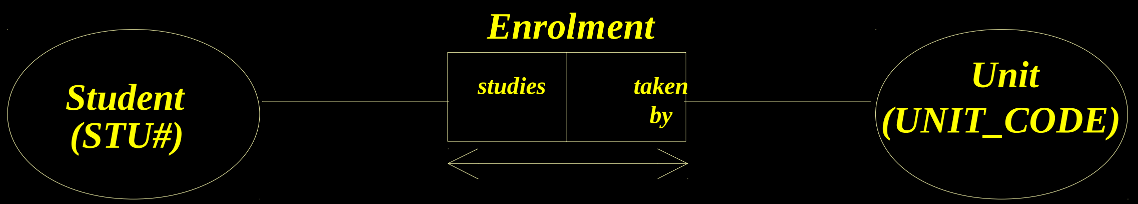


At another extreme, create a relation for each subtype i.e. 'pulling down' the supertype's fact types into each subtype.
Any compromise between the two extremes can be applicable.



SCHEMA TRANSFORMATION EXAMPLES

Example 1



The relation Enrolment is formed with two attributes: Student number and Unit code

Enrolment (Stu#, Unit_Code)

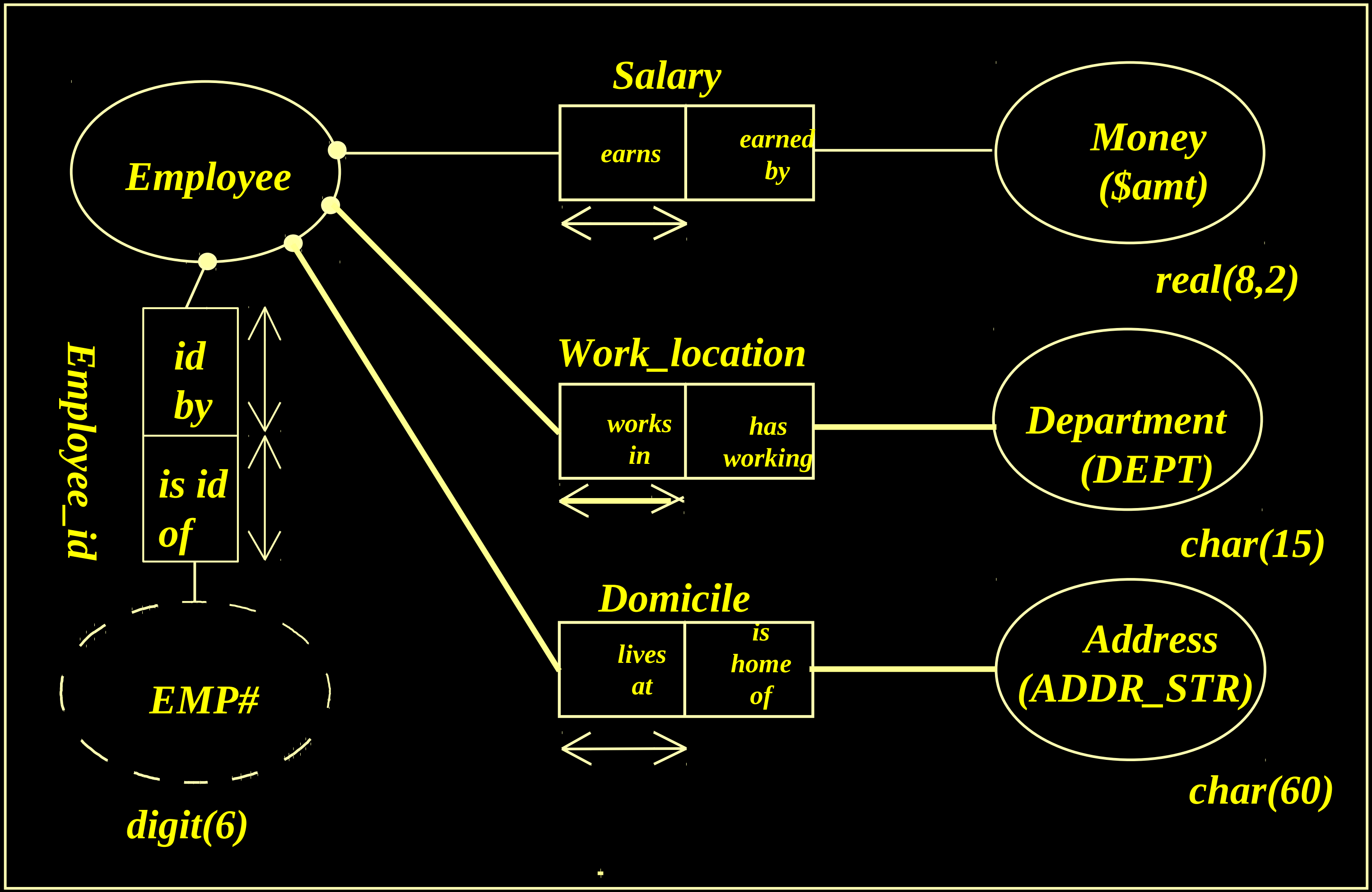
The key for the relation form two attributes

(Stu#, Unit_Code)

Enrolment

Stu #	Unit_Code

Example 2



Build a relation around Employee

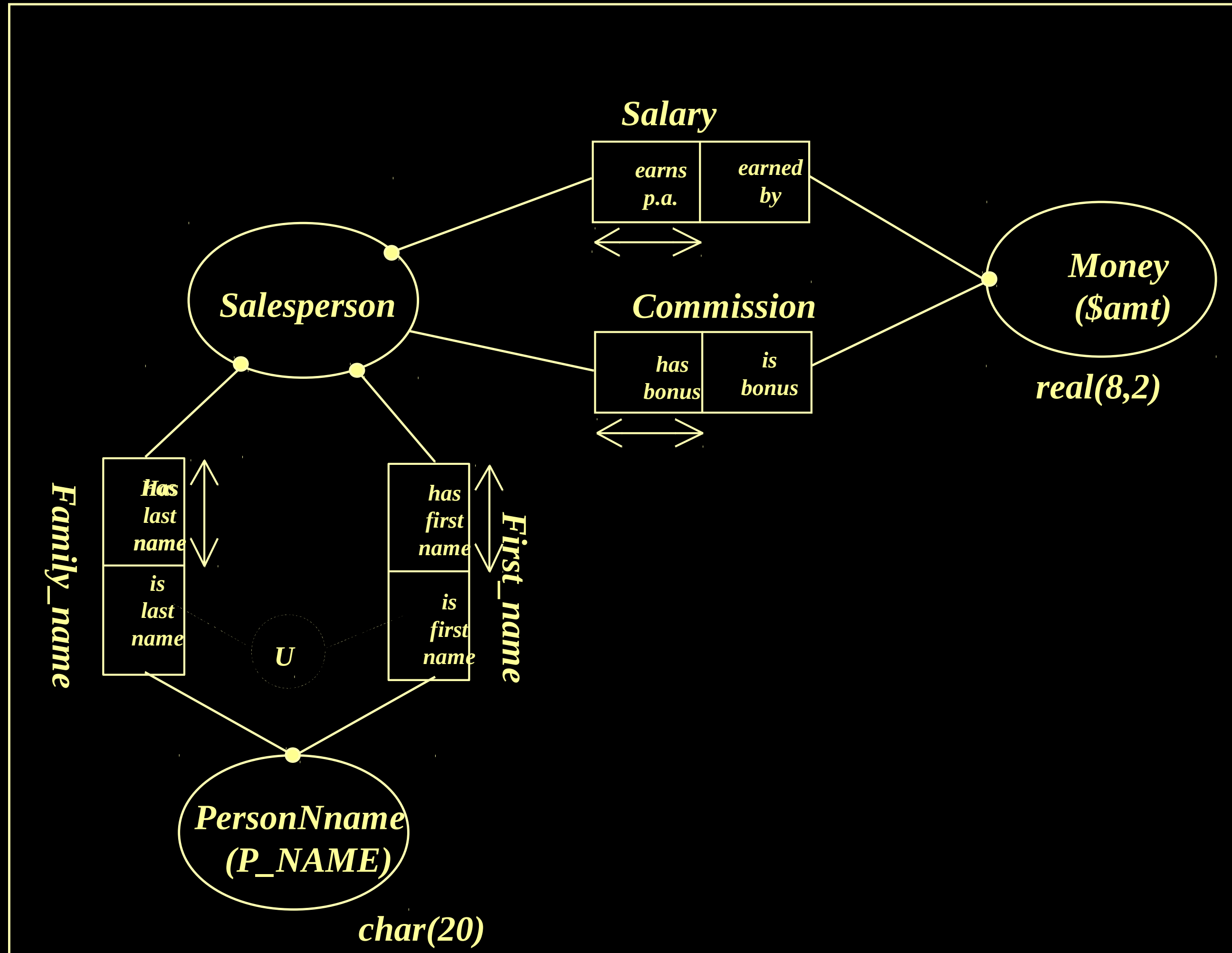
The Semantics of Relation:

All these binary FT describe certain properties of employees.
The natural name choice for the relation is Employee.
Other attributes are describing employee:

Empl Address,
Employing Department
Salary

The relation created is :
Employee (<u>Emp#</u> , Dept, Salary, H_Addr)   Key: Emp#

Example 3



Build a relation around Salesperson As in example 2

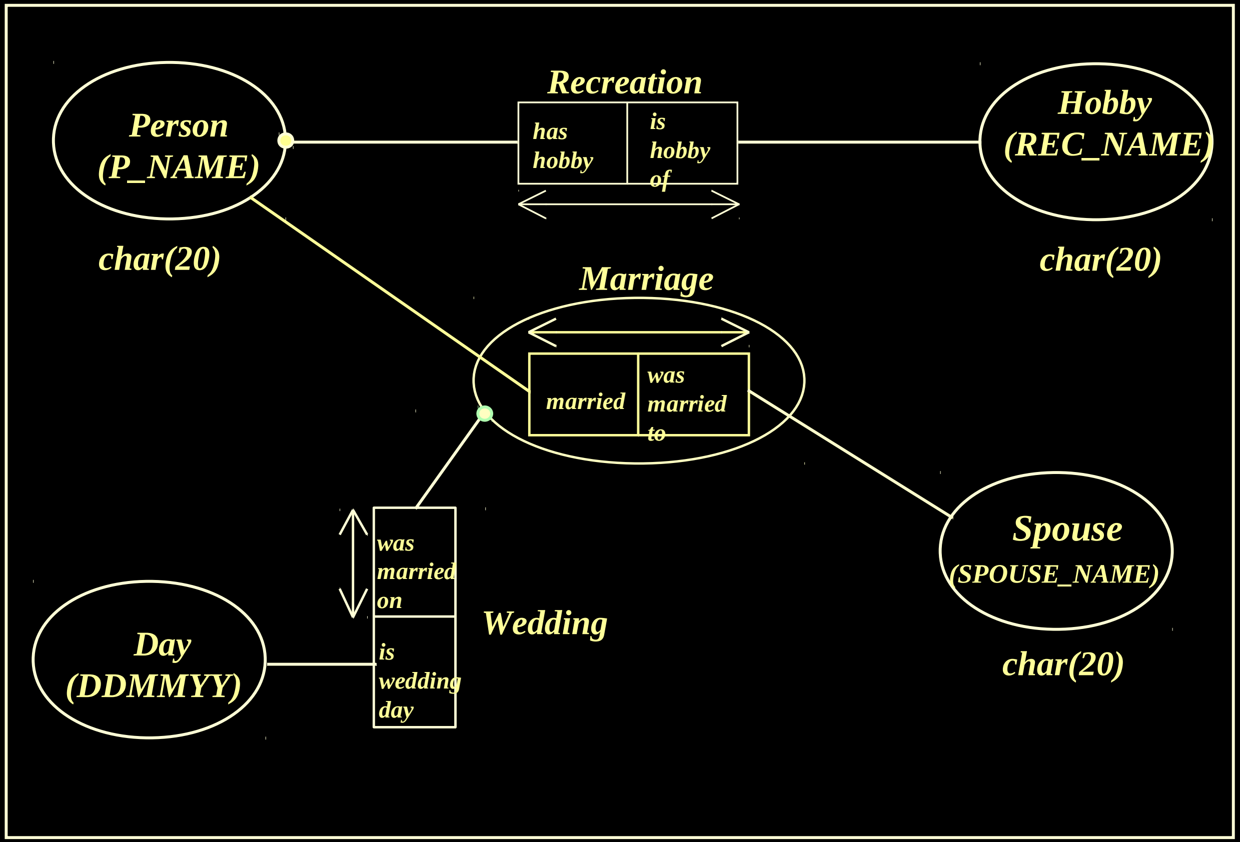
Salesperson	Salesperson identity	Earnings	Bonus (opt)

Salesperson (SalesPerson Identification, Salary, Bonus)
Key is SalesPerson Identification

Notice, that Salesperson entity type is not identified in 1-1 way by a single label type SalesPerson ID – no such label type)
However, the schema provides identification of Salesperson using a combination of first and last names.

The resulting relation is
Salesperson (SP_Fname SP_Lname, Salary, Bonus)
With a single key;   SP_Fname SP_Lname

Example 4



Recreation (P_Name Hobby) *Key: P_Name Hobby*

Marriage (P_Name SpouseName WeddingDate) *Key: P_Name SpouseName*

Example 5
Subtyping

R1 X B

R2 X K L

R3 X C

R4 V I

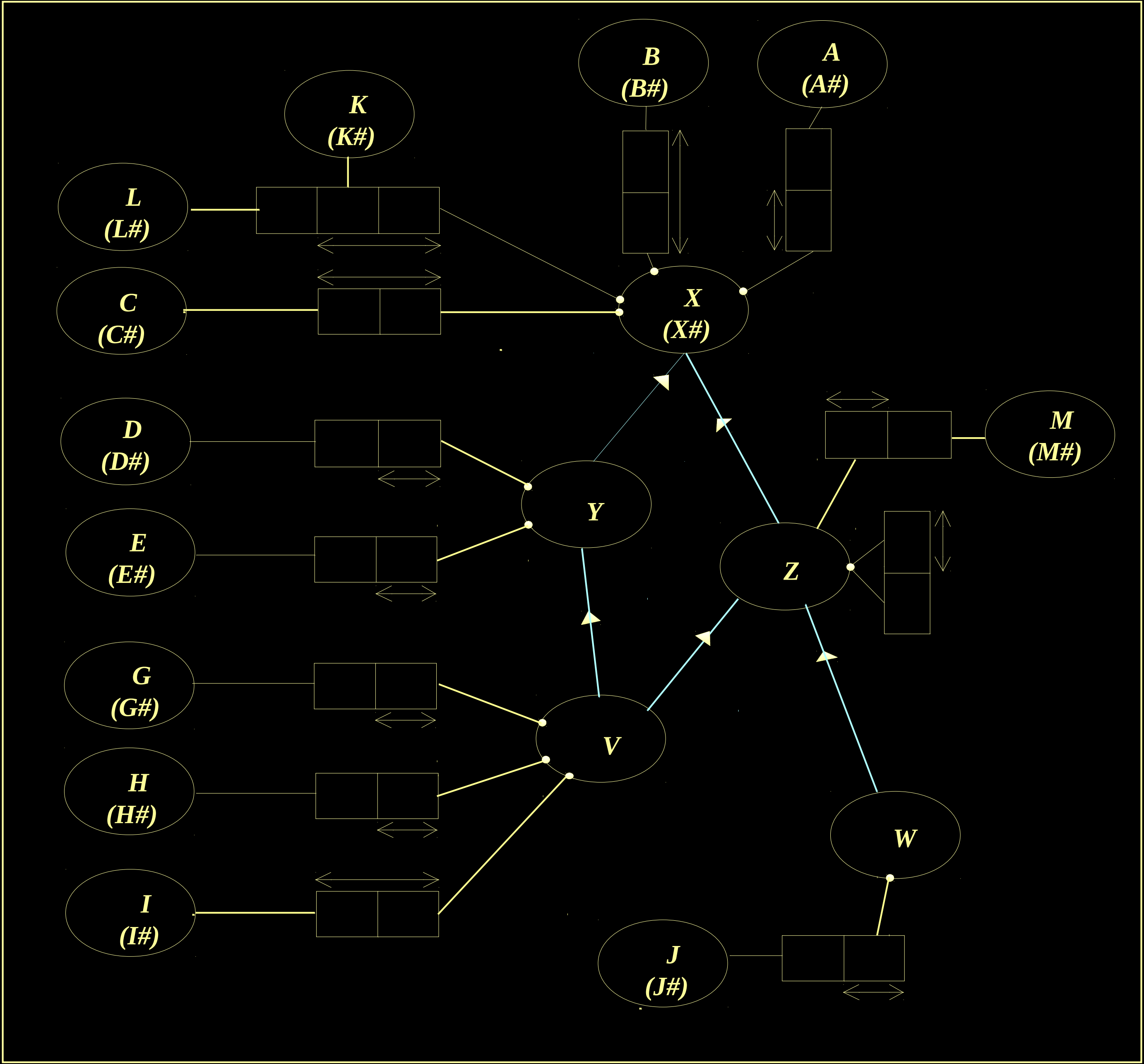
R5 X A

R6 Y D E

R7 Z M Z’

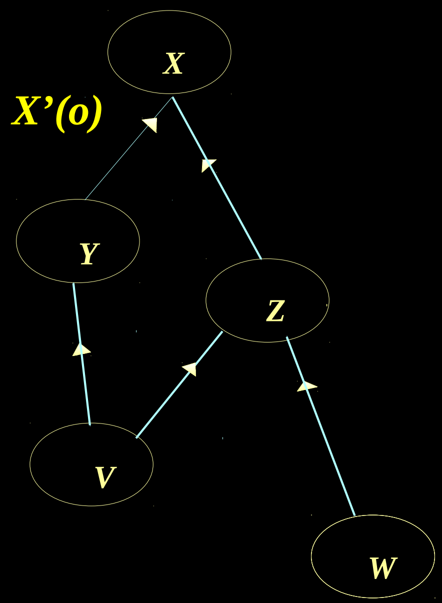
R8 V G H

R9 W J

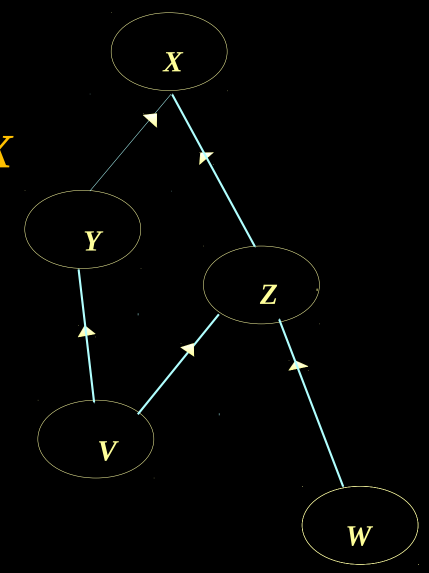


The design of relational database when subtyping is present and where certain subtypes are ABSORBED by their supertypes:

Absorbed	*Y by X*	*Z by X*	*Y, Z by X*
X B	X B	X B	X B
X K L	X K L	X K L	X K L
X C	X C	X C	X C
V I	V I	V I	V I
X A	X A D(o) E(o)	X A M(o) X'(o)	X A D E(o) M(o) X'(o)
Y D E		Y D E	
Z M Z'	Z M Z'		
V G H	V G H	V G H	V G H
W J	W J	W J	W J



(o) - optional



Absorbed:	W by Z	Z by X and V by Y	V, Y, Z by X
X B	X B	X B	X B
X K L	X K L	X K L	X K L
X C	X C	X C	X C
V I	V I	Y I	X I
X A	X A	X AM(o)X'(o)	X A D E M X' G H all (o) except A.
Y D E	Y D E	Y D EG(o) H(o)	
Z M Z'	Z M Z'J(o)		
V G H	V G H		
W J		W J	W J

Can V be absorbed by Y only? If V is absorbed by Y then what are other compulsory absorbptions?
How the design looks like if ALL subtypes are ABSORBED by supertype X?

In next ‘generic’ examples of relational design we adopt the following naming convention:

Capital letters A, B,… - the symbols for entity types (except F) ,

F1, F2, … - the denotations for fact types, F1*, F2* set of attributes defined by F1, F2, .. respectively

r1, r2, r3 … - denote uniquely the roles the entity types play in the fact types.

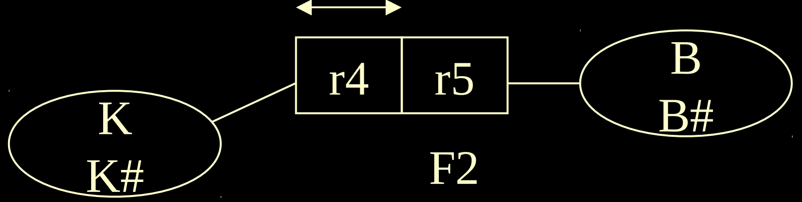
Combination (entity_type role_number) denotes the attribute generated by that entity type and that role as indicated on the Figure below: B5 is the attribute in the table generated by role r5.

A flat fact type may generate a relation that has its source name in another fact type. In this case, informally we use an arrow ‘ → ‘ between the fact id and relation as illustarted on the next slide.

In the case of an entity type being involved in a role with a single UC on the role ‘touching’ that entity type, the identity of that entity type – typically its label type – will serve as the attribute name.

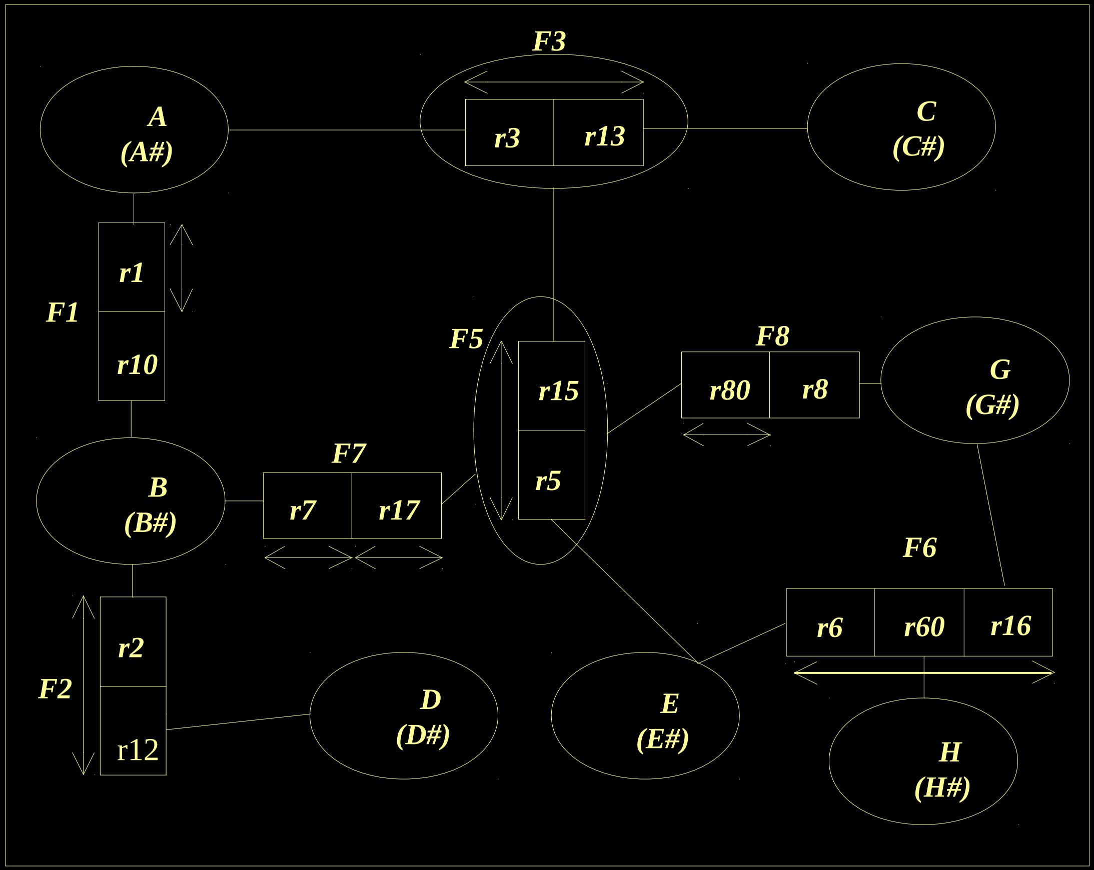
Example:

We will use K# in K(<u>K#</u> B5) <u>BUT NOT</u> K(<u>K4</u> B5)



Example 6

F1 → A (<u>A#</u> , B10)
F2 → F2 (<u>B2, D12</u>)
F6 → F6 (<u>E6,H60,G16</u>)
F8 → F5 (<u>F5*</u>, G8) =
F5 (<u>F3*, E5</u>, G8) =
F5 (<u>A3, C13, E5</u>, G8)
F7 → F5 (B#, F5*) =
F5 (B#, <u>A3, C13, E5</u>)
With B# as a possible alternative key

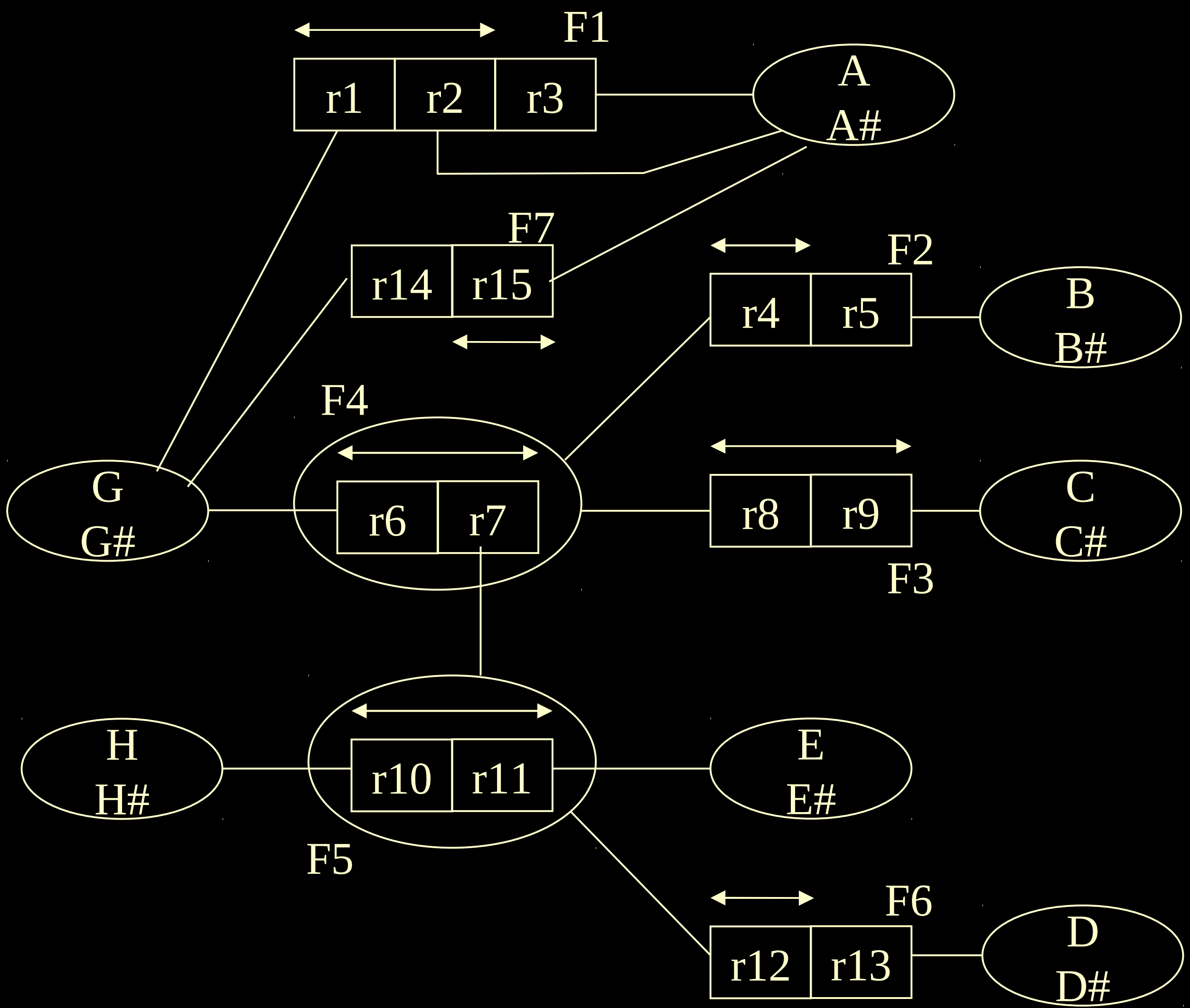


Both tables have the same key so they can be combined into one relation:

F5 (<u>A3, C13, E5</u>, G8, B#)

The semantics and other analysis is needed to decide if B# can be an alternative key for F5

Example 7

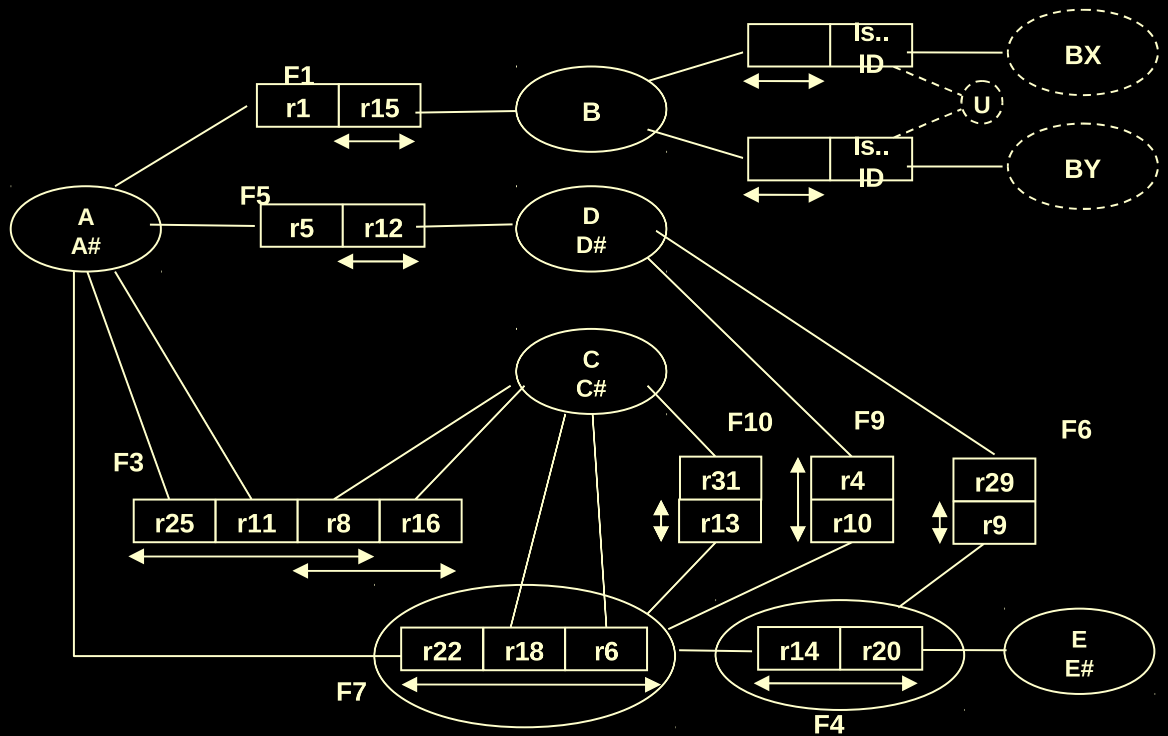


F1 → F1(G1 A2 A3)
F2 → F4 (F4* B5) =
F4 (G6 F5* B5) =
F4 (G6 H10 E11 B5)
F3 → F3(F4* C9) =
F3(G6 F5* C9) =
F3(G6 H10 E11 C9)
F6 → F5 (F5* D13) =
F5(H10 E11 D13)
F7 → A (A#, G14)

In case of transformation of F1, the resulting relation name is to be invented from the semantics of the key G1 A2.

In other cases they are typically inherited from the fact type name that provides the key.

Example 8



F1 → B(<u>B?</u> A1)
B is not identified here. So after its identification as label cobination BX BY we have B(<u>BX BY</u> A1)

F5 → D (<u>D#</u> A5)
F3 → F3 (<u>A25 A11 C8</u> C16)
other key is <u>C8 C16</u>

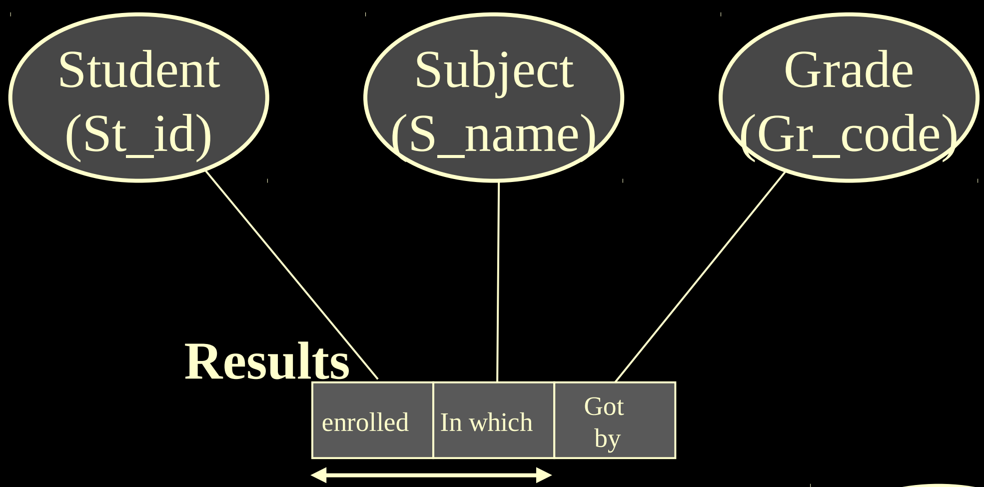
F10 → F7 (<u>F7*</u> C31) =
F7(<u>A22 C18 C6</u> C31)

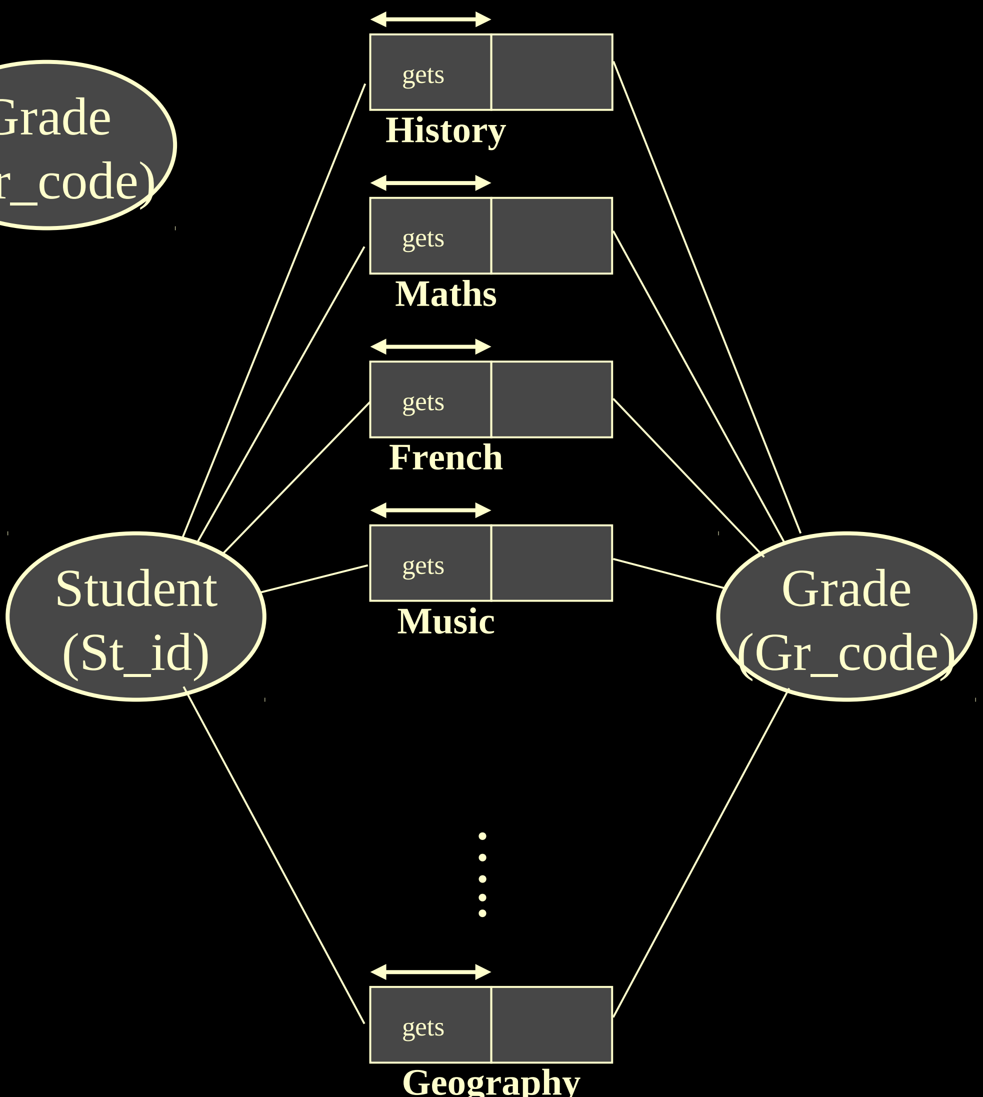
F9 → F9(<u>F7* D4</u>) =
F9(<u>A22 C18 C6 D4</u>)

F6 → F4 (<u>F4*</u> D29) =
F4 (<u>F7* E20</u> D29) =
F4 (<u>A22 C18 C6 E20</u> D29)

Schema equivalence and its impact on relational design

Example





Different schemas for the same UoD.

Resulting in different structure of data storage.

Storing the same info in different logical structures

Note that the second table contains the same information but occupies less space than the first one:

If there are 8 subjects then the first relation requires 8 records per student – 24 fields while the other one – 1 record with 9 fields.

StudentNo	Subject	Result
113456	History	A
113456	Maths	A
113456	French	B
…	…	
232425	History	C
…	…	…

Student	Math	History	Biolog	English	Sci	Music	PhEdu	Geogr
113456	A	A	C	B	C	B	C	A
232425	C	B	B	A	A	B	A	B

Foreign Keys

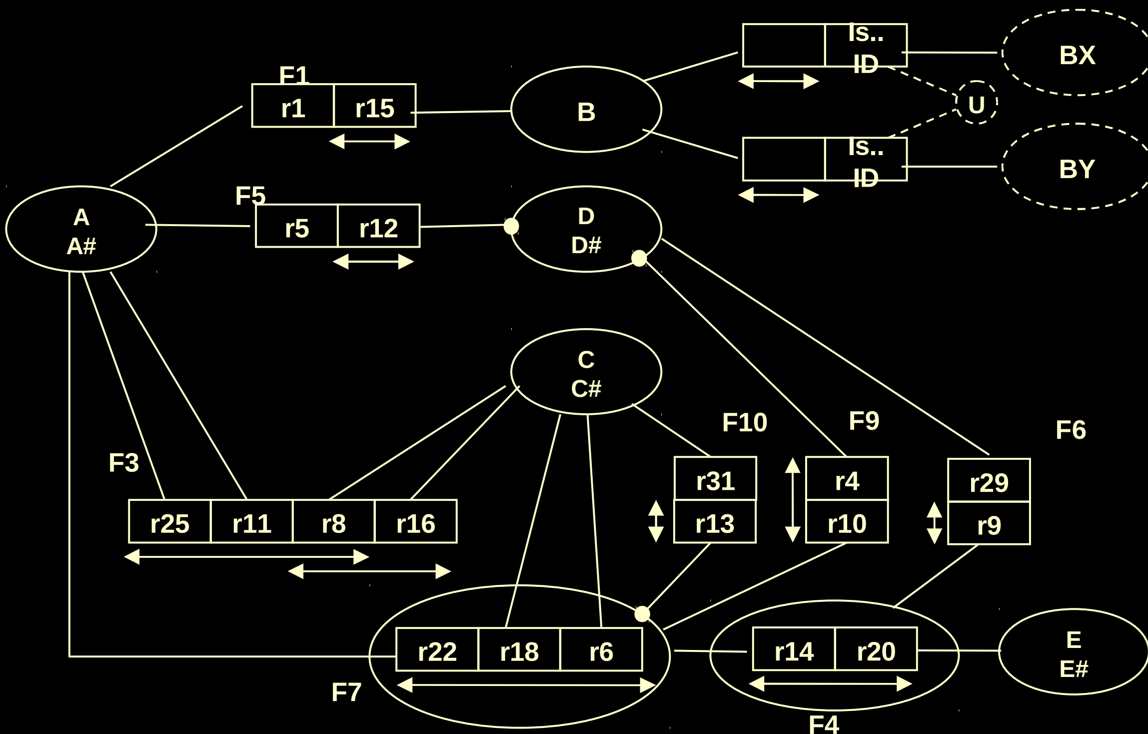
- **When construction of all relations is completed then one can identify foreign key connection between them. For this purpose for each key check if**
  - (not single attribute key) The combination of attributes forming that key is present in another relation,
  - Check if between the populations of the fact type generating the relations is a subset constraint or other constraint that secures the presence of values of each instance in foreign key in the referenced key.

  If both conditions are satisfied then there is a foreign key connection between these relations

  or

  - (single attribute key, and for simplicity use notation E#) check if the relevant to that key entity type E has played a role in another fact type. Then the attribute corresponding to that entity type and that role is a foreign key in that table and referencing the key E# in the table E

Example 9



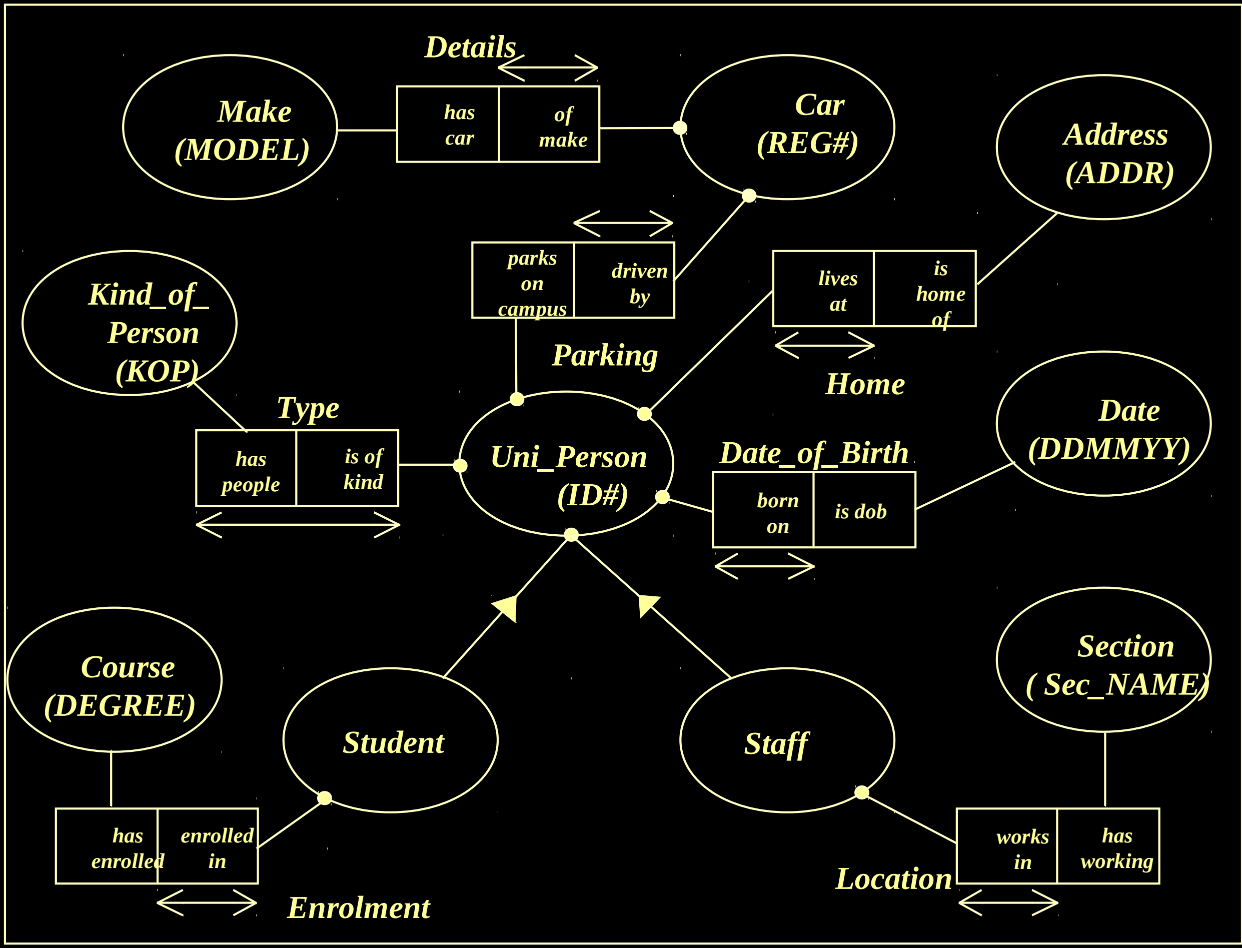
B(BX BY A1)
D (D# A5)
F3 (A25 A11 C8 C16)
other key is C8 C16
F7(A22 C18 C6 C31)
F9(A22 C18 C6 D4)
F4 (A22 C18 C6 E20 D29)

Forein keys :

D4 in F9 D29 in F4
both referencing D
A22C18C6 in F9 and
A22C18C6 in F4
both referencing F7

Note the mandatory role constraints on D (role r12) nd on F7 (role r13) secure the subset constraint between instances of foreign key and referenced key

Example 10



Type (<u>ID#, Kind_of_person</u>)

ID# references Uni_person

Car (<u>Reg#</u> , Model, ID#)

ID# references Uni_person

Uni_person (<u>ID#</u> , DoB, Home)

Student (<u>ID#</u> , Degree)

ID# references Uni_person

Staff (<u>ID#</u>, Sec_Name)

ID# references Uni_person

Summary

- **We introduced an outline of the transformation procedure from ORM schema to RDB schema.**

- **The porcedure has been illustrated by examples – more in additional studio's material.**

Finally, Object Role Modeling (ORM) is a powerful method for designing and querying database models at the conceptual level, where the application is described in terms easily understood by non-technical users. In practice, ORM data models often capture more business rules, and are easier to validate and evolve than data models in other approaches.