



POLITECNICO
MILANO 1863

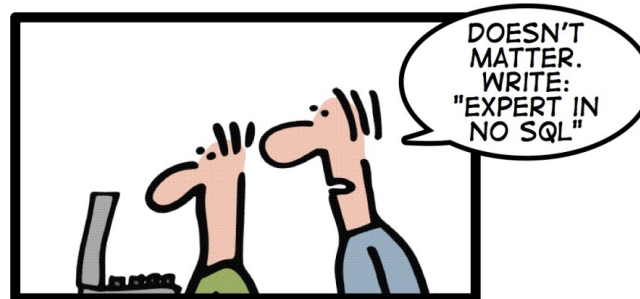
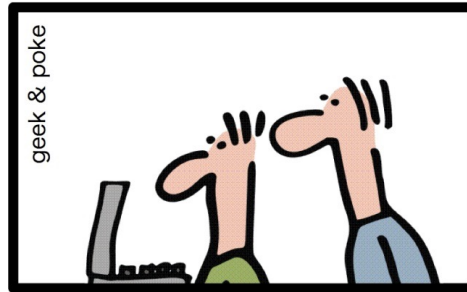
SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

E-R and Relational Data Models

Marco Brambilla

marco.brambilla@polimi.it

 @marcobrambi



Leverage the NoSQL boom

Agenda

Design Levels

ER Models

Relational Databases

Normalization

The levels

Conceptual database design

Constructing an information model, independent from all physical consideration for an enterprise

Logical database design

Building an organization database based on a specific data model

Physical database design

Implementing a database using specific data storage structure(s) and access methods

Equivalence to CIM, PIM, PSM in software modeling

Relational Database – Conceptual Design

Entities

Relationships

Attributes (simple, no composite, no derived)

Attribute domains

Key attributes

Relational Database – Logical Design

From ER model

Review

- Always binary with 1 to many relationship

- No complex relationship

- No redundant relationship

- No recursive relationship

- No relationship with attribute(s)

Relational Database – Logical Design

Normalize relations

- Primary key

- Foreign key

- Normal form, BCNF

Add constraints

Relational Database – Logical Design

Mapping logical database to DBMS

- Base relations

- Integrity Rules

- Referential integrity (delete & update)

 - No action

 - Cascade

 - Set null

 - Set default

 - No check

Relational Database – Physical Design

Implementing physical representation

- Analysis transactions

- File organization

- Indexes

- Disk space

Security

- Policy & procedure

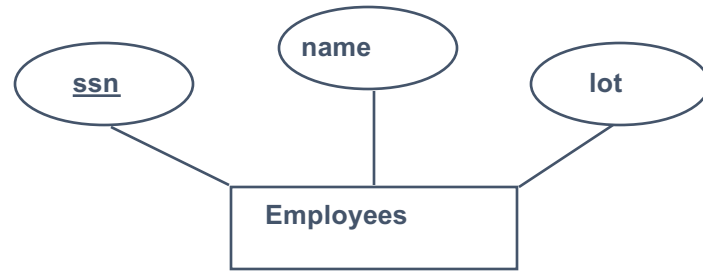
- User view

- Access rules

- Transmission

ER *Model*

ER Model Basics



Entity: Real-world object distinguishable from other objects. An entity is described using a set of *attributes*.

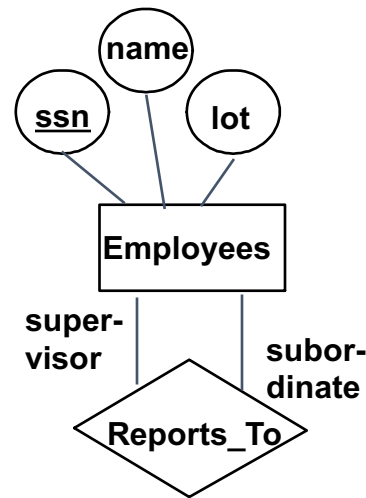
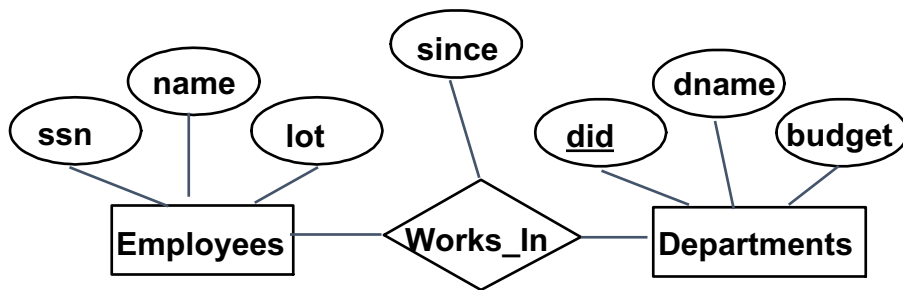
Entity Set: A collection of entities of the same kind.
E.g., all employees.

All entities in an entity set have the same set of attributes.

Each entity set has a *key* (a set of attributes uniquely identifying an entity).

Each attribute has a *domain*.

ER Model Basics



Relationship: Association among two or more entities.

Relationship Set: Collection of similar relationships.

An n -ary relationship set R relates n entity sets $E_1 \dots E_n$; each relationship in R involves entities $e_1 \in E_1, \dots, e_n \in E_n$

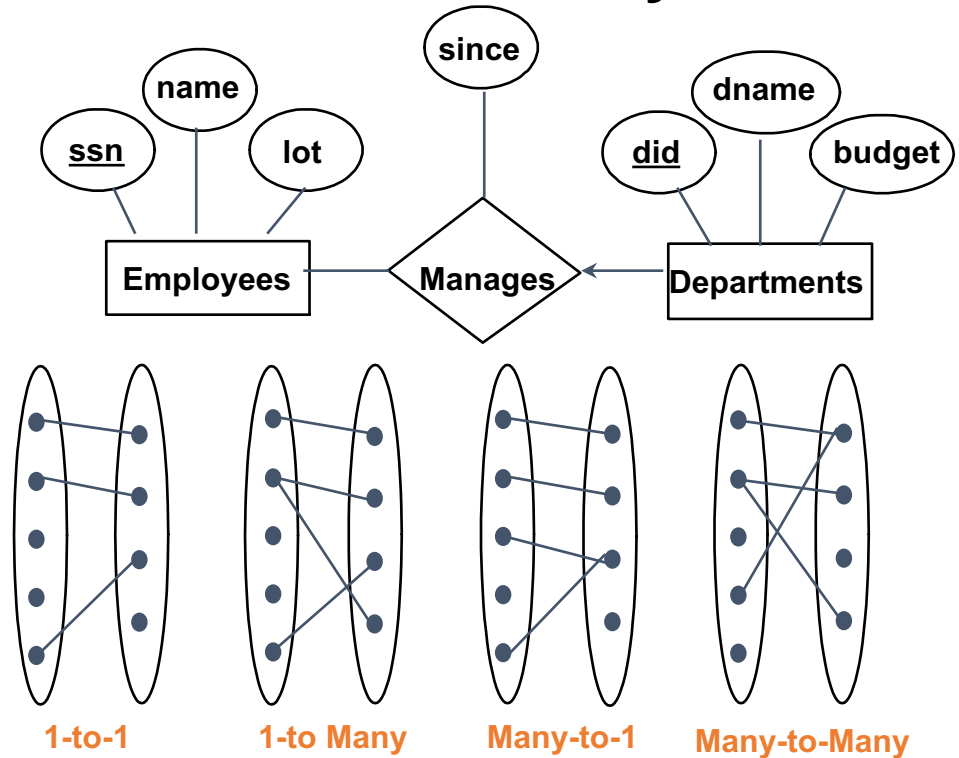
Same entity set could participate in different relationship sets, or in different “roles” in same set.

Relationship sets can also have *descriptive attributes* (e.g., the *since* attribute of **Works_In**). A relationship is uniquely identified by participating entities without reference to descriptive attributes.

Key Constraints (a.k.a. Cardinality)

Consider Works_In (in previous slide): An employee can work in many departments; a dept can have many employees.

In contrast, each dept has at most one manager, according to the key constraint on Manages.



Constraints are **IMPORTANT** because they must be **ENFORCED** when **IMPLEMENTING** the database

Participation Constraints

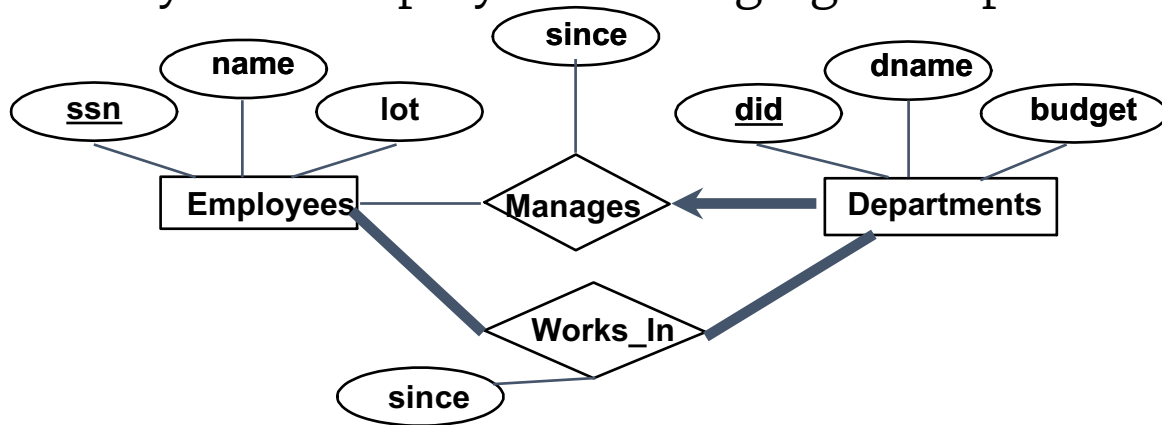
Does every department have a manager?

If so, this is a participation constraint: the participation of Departments in Manages is said to be *total (vs. partial)*.

Every Department MUST have at least an employee

Every employee MUST work at least in one department

There may exist employees managing no department

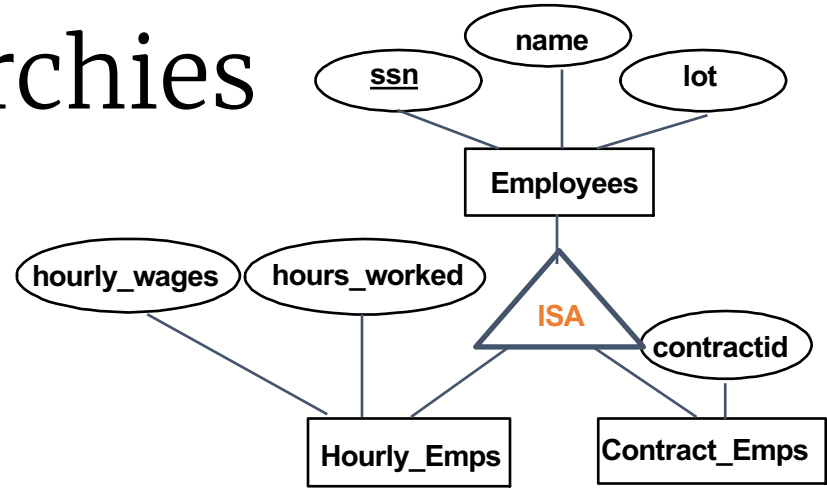


ISA ('is a') Hierarchies

Reasons for using ISA:

- To add descriptive attributes specific to a subclass.

- To identify entities that participate in a relationship.



Overlap constraints: Can Joe be an Hourly_Emps as well as a Contract_Emps entity? if so, specify => Hourly_Emps OVERLAPS Contract_Emps.

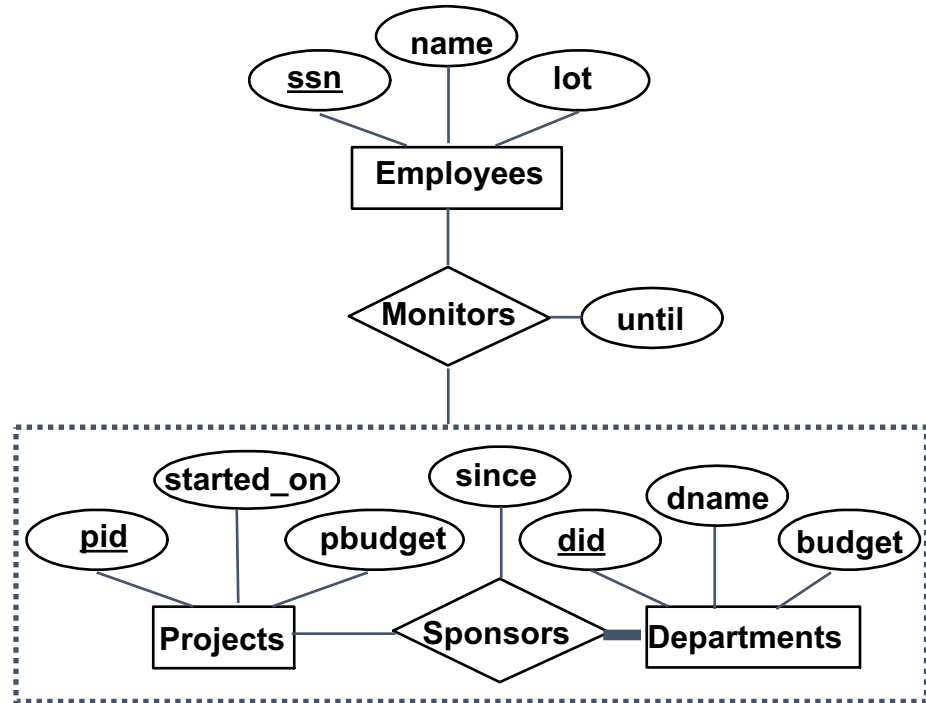
Covering constraints: Does every Employees' entity also have to be an Hourly_Emps or a Contract_Emps entity?. If so, write Hourly_Emps AND Contract_Emps COVER Employees.

Aggregation

Used when we have to model a relationship involving (entity sets and) a *relationship set*.

Aggregation allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.

Employees are assigned to monitor SPONSORSHIPS.



Conceptual Design Using the ER Model

Crucial design choices:

Should a concept be modeled as an entity or an attribute?

Should a concept be modeled as an entity or a relationship?

Identifying relationships: Binary or ternary? Aggregation?

Constraints in the ER Model:

A lot of data semantics can (and should) be captured.

But some constraints cannot be captured in ER diagrams.

Entity vs. Attribute

Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?

Depends upon the use we want to make of address information, and the semantics of the data:

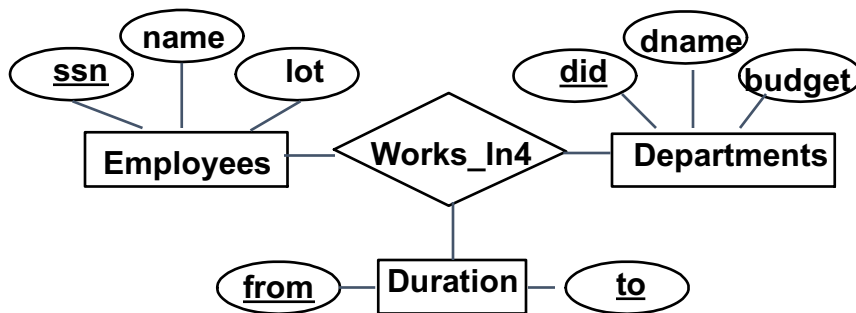
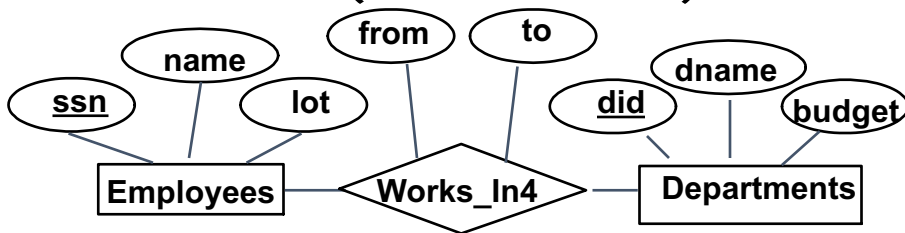
If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).

If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

Entity vs. Attribute (Contd.)

Works_In4 does not allow an employee to work in a department for two or more periods (a relationship is identified by participating entities).

Similar to the problem of wanting to record several addresses for an employee: We want to record *several values of the descriptive attributes for each instance of this relationship*. Accomplished by introducing new entity set, Duration.



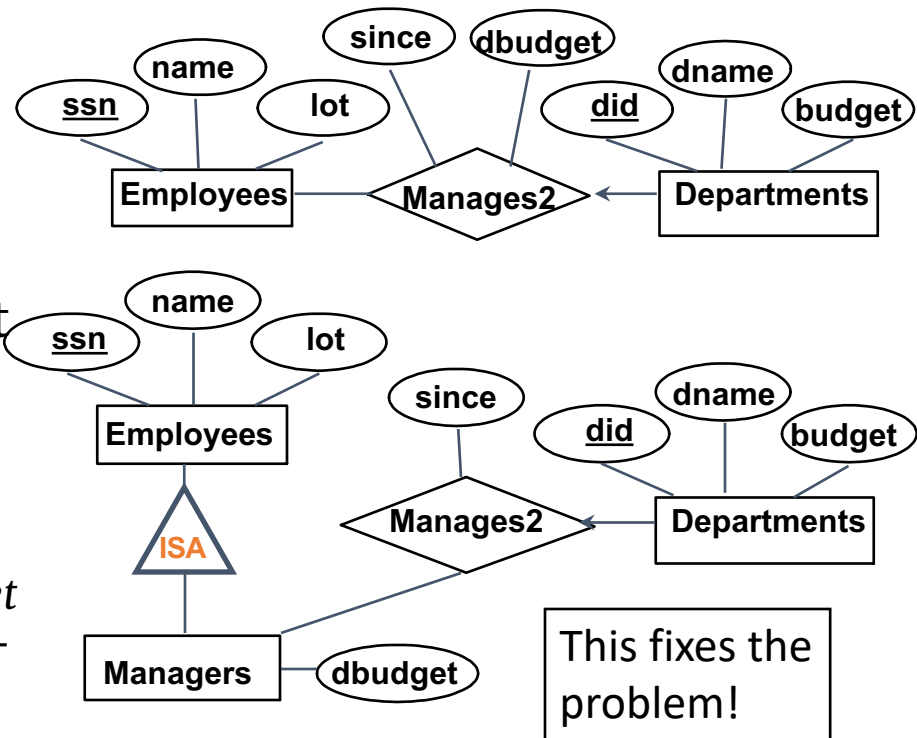
Entity vs. Relationship

First ER diagram OK if a manager gets a separate discretionary budget for each dept.

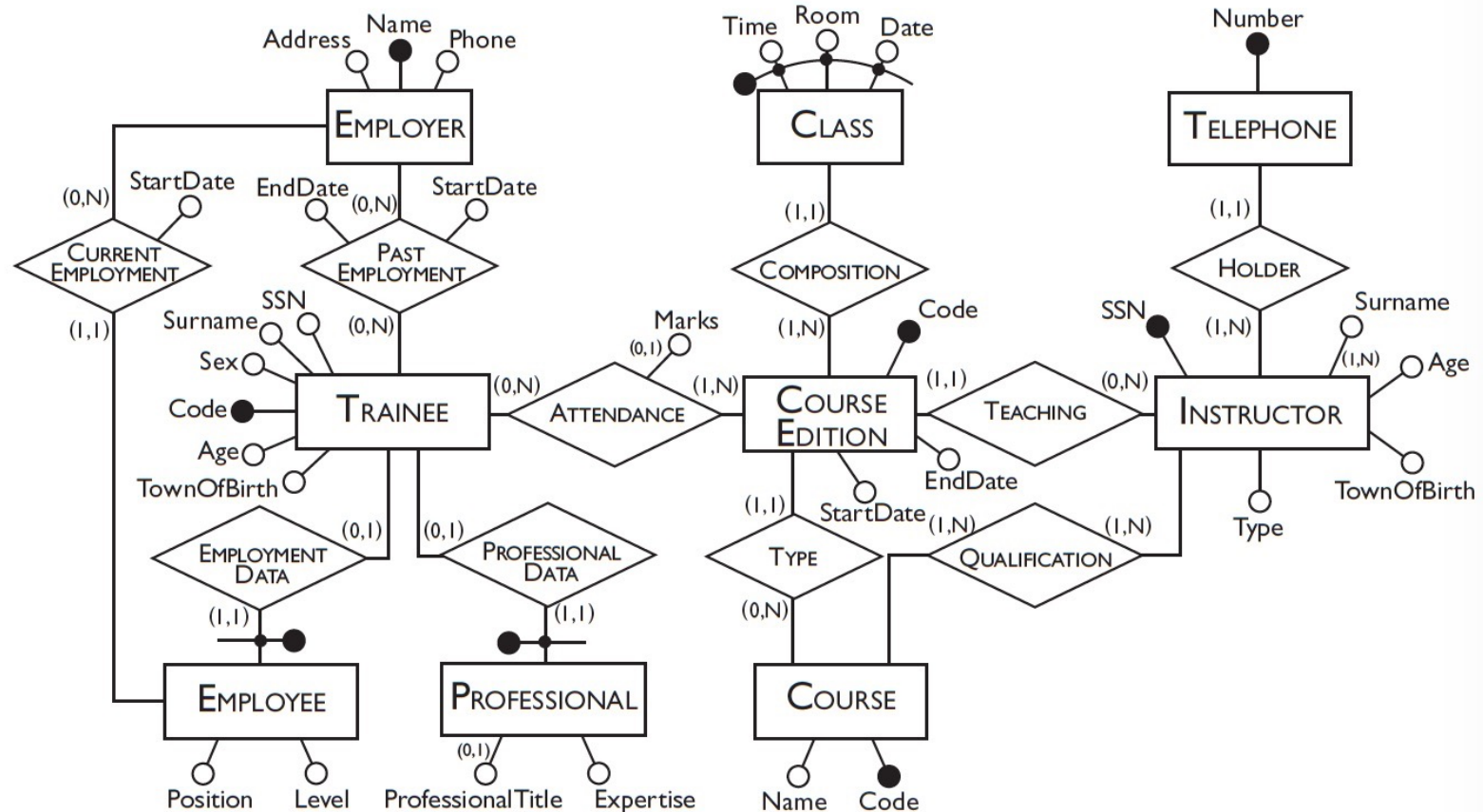
What if a manager gets a discretionary budget that covers *all* managed depts?

Redundancy: *dbudget* stored for each dept managed by manager.

Misleading: Suggests *dbudget* associated with department-mgr combination.

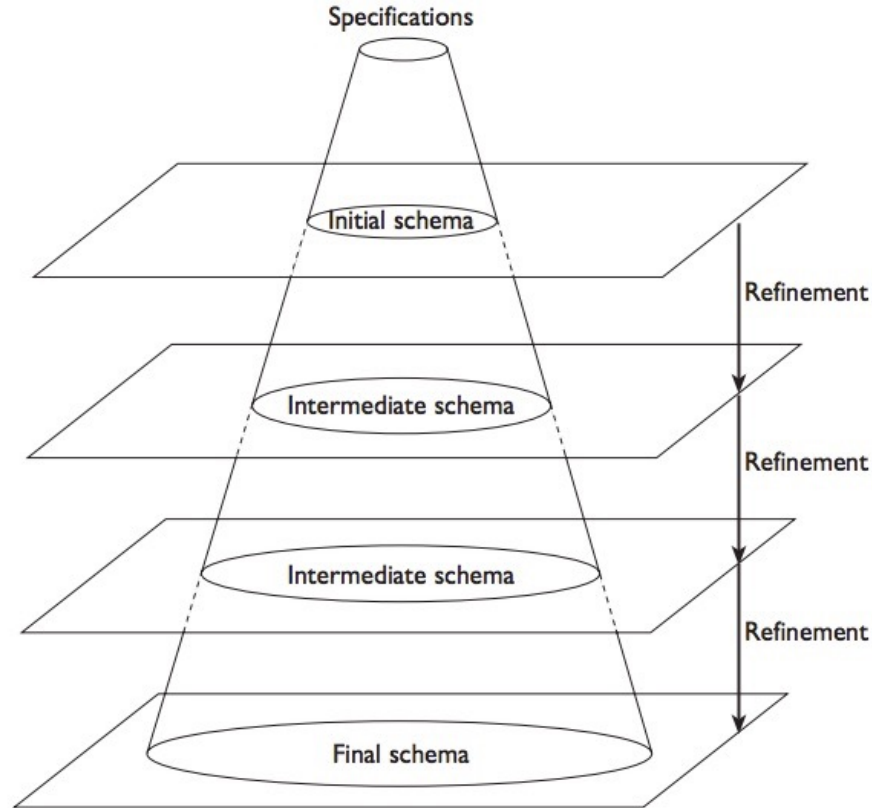


E-R Example


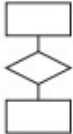



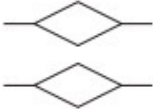

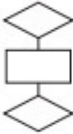






ER Design Methodology

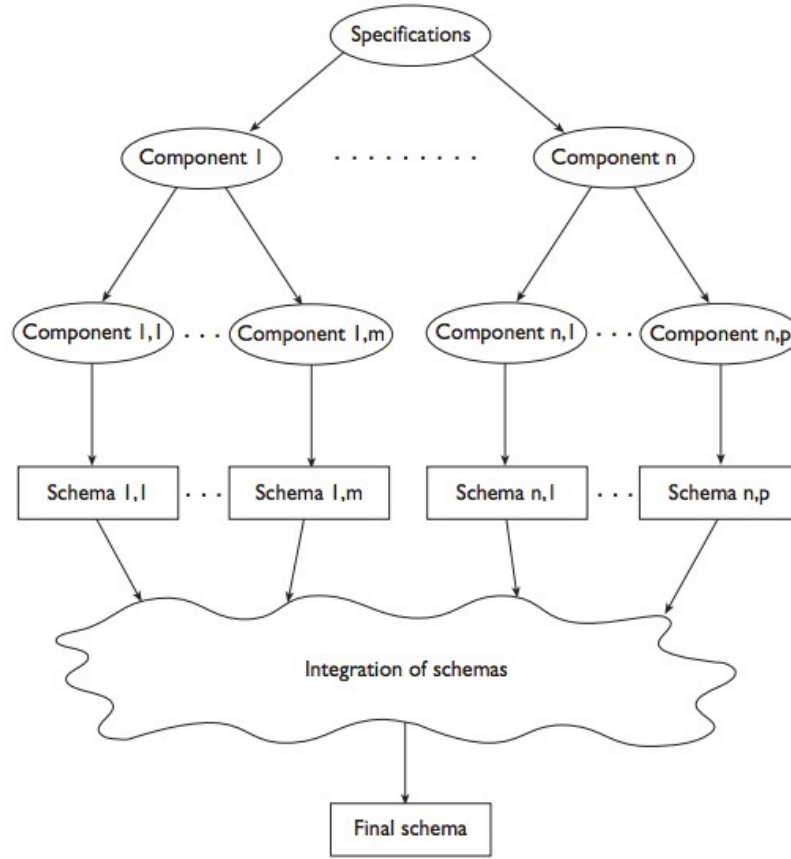
Top-down strategy





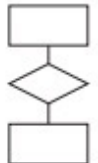

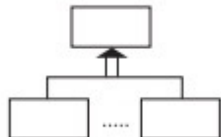




Top-down primitives

Transformation	Initial concept	Result
T_1 From one entity to two entities and a relationship between them		
T_2 From one entity to a generalization		
T_3 From one relationship to multiple relationships		
T_4 From one relationship to an entity with relationships		
T_5 Adding attributes to an entity		
T_6 Adding attributes to a relationship		

Bottom-up strategy

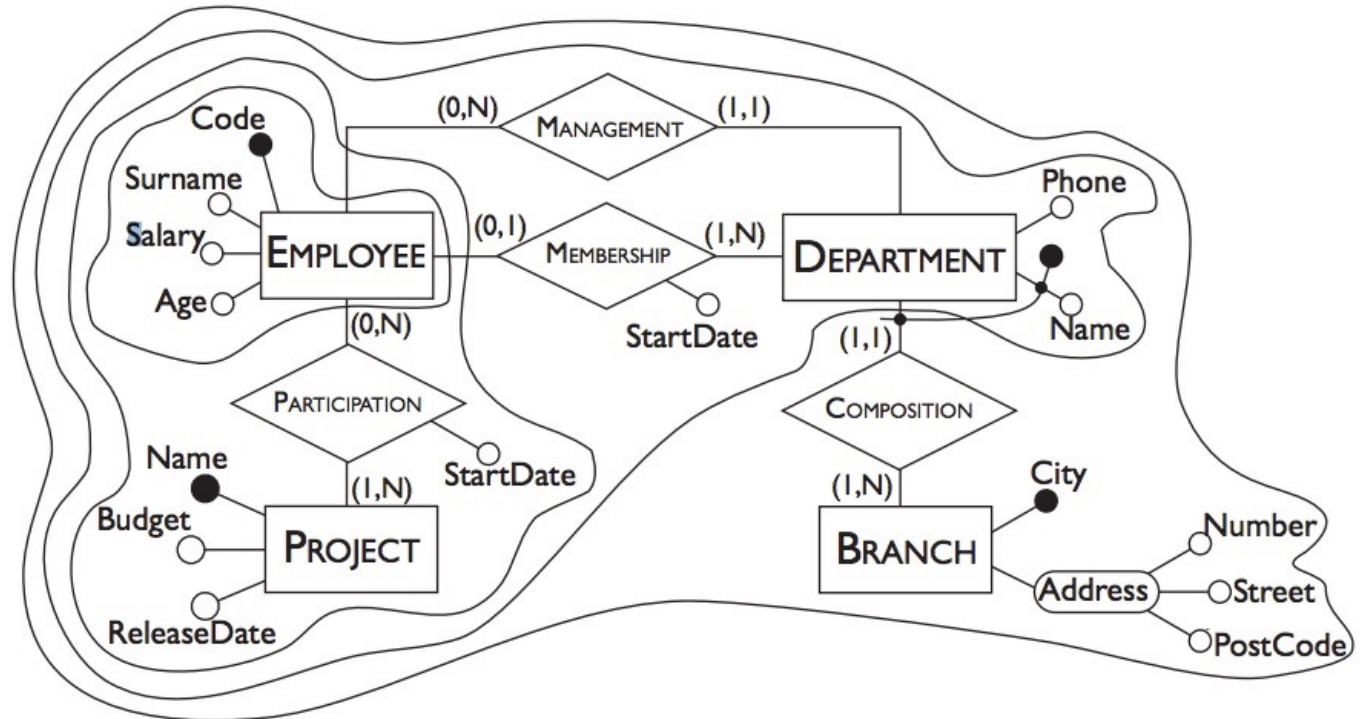


Top-down primitives

Transformation	Initial concept	Result
T_1 Generation of an entity		
T_2 Generation of a relationship		
T_3 Generation of a generalization		
T_4 Aggregation of attributes on an entity		
T_5 Aggregation of attributes on a relationship		

Inside-out strategy

Ex.



Relational Model

The relational model (aka. „SQL“)

Proposed by E. F. Codd in 1970

Made available in commercial DBMSs in 1981

It is based on (a variant of) the mathematical notion of **relation**

Relations are naturally represented by means of tables

Mathematical relations

D_1, D_2, \dots, D_n (n sets, not necessarily distinct)

cartesian product $D_1 \times D_2 \times \dots \times D_n$:

the set of all (ordered) n -tuples (d_1, d_2, \dots, d_n) such that $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n$

a **mathematical relation** on D_1, D_2, \dots, D_n :

a subset of the cartesian product $D_1 \times D_2 \times \dots \times D_n$.

D_1, D_2, \dots, D_n are the **domains of the relation**

n is the **degree** of the relation

the number of n -tuples is the **cardinality** of the relation; in practice, it is always finite

Mathematical relations, properties

A mathematical relation is a **set** of **ordered** n -tuples

(d_1, d_2, \dots, d_n) tali che $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n$

a set, so:

- there is no ordering between n -tuples

- the n -tuples are distinct from one another

the n -tuples are **ordered**: the i -th value come from the i -th domain

Mathematical relation, example

$games \subseteq string \times string \times integer \times integer$

Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	1	2
Roma	Milan	0	1

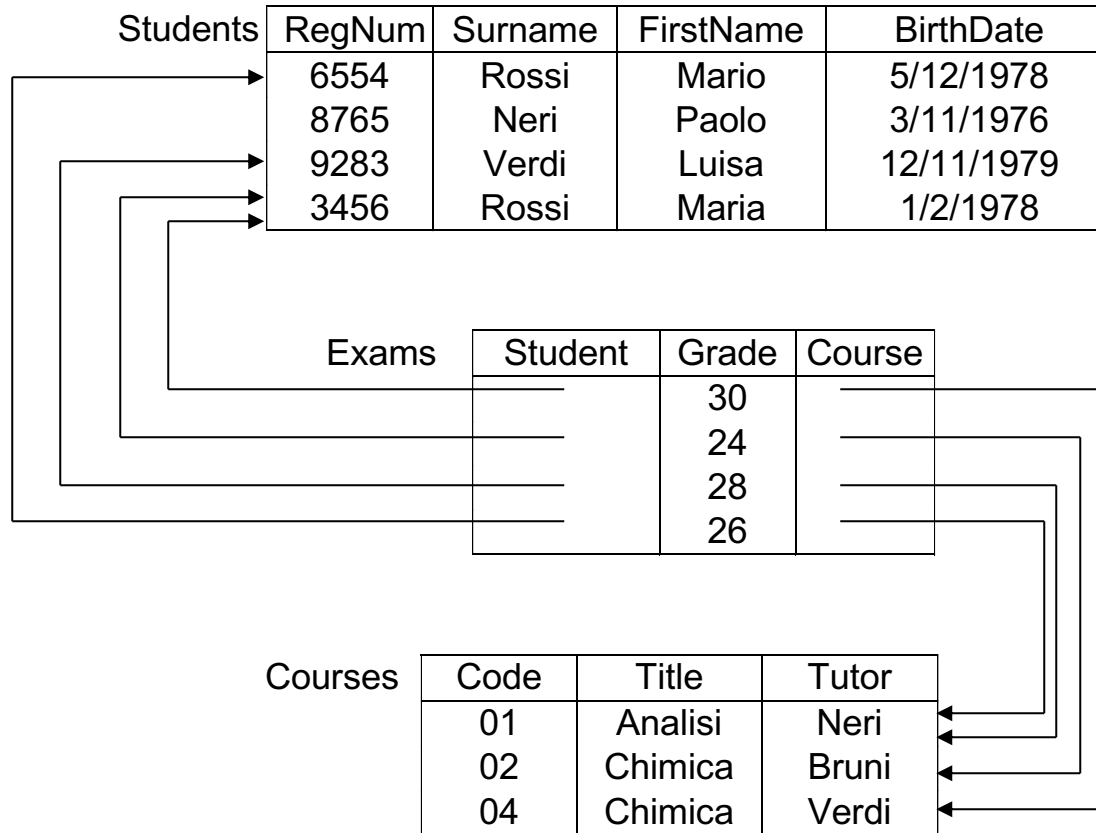
Each of the domains has two **roles**, distinguished by means of position

The structure is **positional**

Relations in the relational data model

We would like to have a **non-positional** structure
We associate a unique name (**attribute**) with
each domain; it “describes” the role of the
domain

HomeTeam	VisitingTeam	HomeGoals	VisitorGoals
Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	1	2
Roma	Milan	0	1



Definitions

Relation schema:

a name (of the relation) R with a set of attributes A_1, \dots, A_n

$$R(A_1, \dots, A_n)$$

Database schema:

a set of relation schemas with different names

$$\mathbf{R} = \{R_1(X_1), \dots, R_n(X_n)\}$$

Relation (instance) on a schema $R(X)$:

set r of tuples on X

Database (instance) on a schema $\mathbf{R} = \{R_1(X_1), \dots, R_n(X_n)\}$:

set of relations $\mathbf{r} = \{r_1, \dots, r_n\}$ (with r_i relation on R_i)

Incomplete information

The relational model imposes a rigid structure to data:

- information is represented by means of tuples
- tuples have to conform to relation schemas

Types of null value

(at least) 3 types of null values:

- **unknown value:** there is a domain value, but it is not known
- **non-existent value:** the attribute is not applicable for the tuple
- **no-information value:** we don't know whether a value exists or not; this is the disjunction (logical or) of the other two

DBMSs do not distinguish between the types: they implicitly adopt the no-information value

Integrity constraints

integrity constraint: a property that must be satisfied by all meaningful database instances;

it can be seen as a **predicate**: a database instance is **legal** if it satisfies all integrity constraints

types of constraints

- intrarelational constraints; special cases:

 - domain constraints

 - tuple constraints

- interrelational constraints

Keys

Key :

a set of attributes that uniquely identifies tuples in a relation

more precisely:

a set of attributes K is a **superkey** for a relation r if r does not contain two distinct tuples t_1 and t_2 with $t_1[K]=t_2[K]$; K is a **key** for r if K is a minimal superkey (that is, there exists no other superkey K' of r that is contained in K as proper subset)

RegNum	Surname	FirstName	BirthDate	DegreeProg
284328	Smith	Luigi	29/04/59	Computing
296328	Smith	John	29/04/59	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	01/05/61	Fine Art
965536	Black	Lucy	05/03/58	Fine Art

RegNum is a key:

- RegNum is a superkey

- it contains a sole attribute, so it is minimal

Surname, Firstname, BirthDate is another key:

- Surname, Firstname, BirthDate form a superkey

- no proper subset is also a superkey

Keys, schemas, and instances

Constraints correspond to properties in the real world to be modelled by our database therefore, they are relevant at the schema level (wrt the whole set of instances)

- we associate with a schema a set of constraints, and we consider as correct (legal, valid, ...) the instances that satisfy all the constraints

- individual instances could satisfy (“by chance”) other constraints

Primary keys

The presence of nulls in keys has to be limited

Practical solution: for each relation we select a **primary key** on which nulls are not allowed

notation: the attributes in the primary key are underlined

References between relations are realized through primary keys

Referential constraints (“foreign keys”)

Pieces of data in different relations are correlated by means of values of (primary) keys

Referential integrity constraints are imposed in order to guarantee that the values refer to actual values in the referenced relation

A database with referential constraints

Offences	<u>Code</u>	Date	Officer	Dept	Registartion
	143256	25/10/1992	567	75	5694 FR
	987554	26/10/1992	456	75	5694 FR
	987557	26/10/1992	456	75	6544 XY
	630876	15/10/1992	456	47	6544 XY
	539856	12/10/1992	567	47	6544 XY

Officers	<u>RegNum</u>	Surname	FirstName
	567	Brun	Jean
	456	Larue	Henri
	638	Larue	Jacques

Cars	<u>Registration</u>	<u>Dept</u>	Owner	...
	6544 XY	75	Cordon Edouard	...
	7122 HT	75	Cordon Edouard	...
	5694 FR	75	Latour Hortense	...
	6544 XY	47	Mimault Bernard	...

Referential constraints

A **referential constraint** imposes to the values on a set X of attributes of a relation R_1 to appear as values for the primary key of another relation R_2

In the example, we have referential constraints between
the attribute Officer of Offences and relation Officers
the attributes Registration and Department of Offences and
relation Cars

ER to Relational

Quick summary

Converting Conceptual to Relational

Step 1: Create a separate table to represent each entity in the conceptual model

1A: Each attribute of the entity becomes a column in the relational table

1B: Each instance of the entity set will become a row in the relational table

Steps 2-4 (see next) involve determining whether each relationship in the conceptual model should be represented as a separate table or as a posted foreign key

Redundancy and Load are important determinants

Redundancy = one fact in multiple places or multiple facts in one place

Load = the percentage of non-null values in a column

Participation Cardinalities communicate some of the information regarding redundancy and load

Relationship Conversion

Maximum Cardinalities

The general rule is to post into a “1” entity table

This avoids “repeating groups” redundancy

You can **NEVER** post into an “N” entity

This causes “repeating groups” redundancy

Minimum Cardinalities

The general rule is to post into a “1” (mandatory) entity table

This avoids null values in the foreign key column

This rule can be violated in some circumstances

Relationship Conversion

Step 2: Create a separate table to represent each many-to-many relationship in the conceptual model, I.e., for the following participation cardinality patterns

$(0,N)-(0,N)$ $(0,N)-(1,N)$ $(1,N)-(0,N)$ $(1,N)-(1,N)$

You must create a separate table to represent the relationship

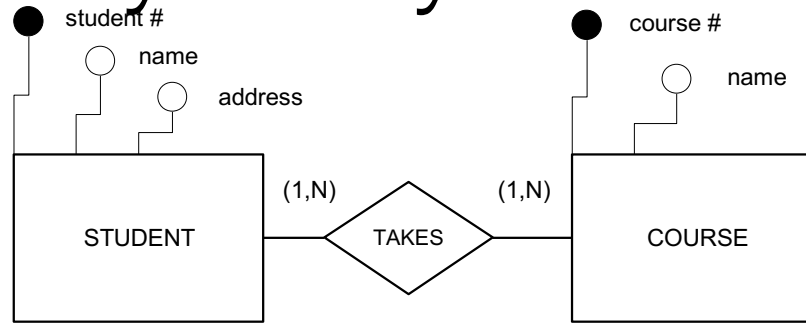
The primary keys of the related entity tables are posted into the relationship table to form its primary key. This kind of primary key is called a composite or concatenated primary key

This avoids redundancy

There are no exceptions to this rule!!!

If you post a foreign key in either direction, redundancy will be a problem for many-to-many relationships

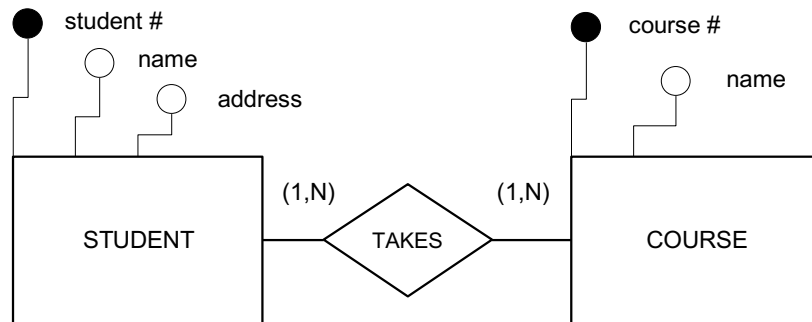
Example: Many-Many Relationships



<u>Student#</u>	Name	Address	Course#
1	Tony	Cleveland	Acg4401, Acg3101
2	Emily	New York	Acg4401, Acg3101
3	Leigh	Birmingham	Acg4401, Acg3101

<u>Course#</u>	Name	Student#
Acg4401	AIS	1, 2, 3
Acg3101	FAR 1	1, 2, 3

Example: Many-Many Relationship



<u>Student#</u>	Name	Address
1	Tony	Cleveland
2	Emily	New York
3	Leigh	Birmingham

<u>Course#</u>	Name
Acg4401	AI
Acg3101	FAR 1

<u>Student#</u>	<u>Course#</u>
1	Acg4401
1	Acg3101
2	Acg4401
2	Acg3101
3	Acg3101

Relationship Conversion

Step 3: For participation cardinality pattern (1,1)–(1,1), consider whether the two entities are conceptually separate or whether they should be combined

If they should remain separate, then

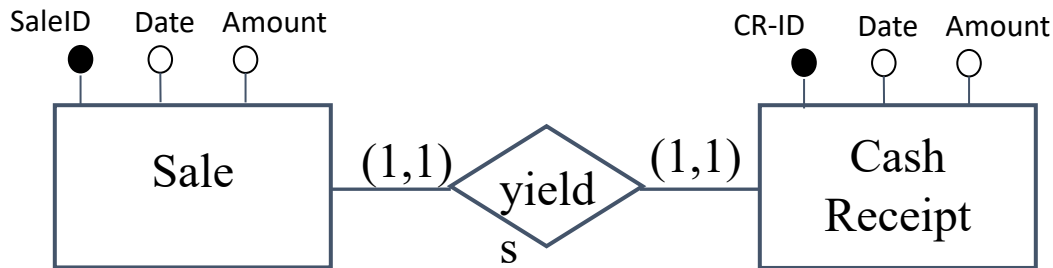
3A: Post the primary key from one entity's table into the other entity's table as a foreign key

3B: It doesn't matter which entity's primary key is posted into the other entity's table, but DO NOT post both

DO NOT make a separate table

Redundancy is automatically avoided and load is not an issue when you post a foreign key into either table in a (1,1)–(1,1) relationship

Example: (1,1)-(1,1)



<u>SaleID</u>	Date	Amount	CR-ID *
S1	6/12	\$10	CR1
S2	6/12	\$15	CR2
S3	6/13	\$12	CR3

<u>CR-ID</u>	Date	Amount	S-ID *
CR1	6/12	\$10	S1
CR2	6/12	\$15	S2
CR3	6/13	\$12	S3

Choose ONE of these; DO NOT do both!!!

Relationship Conversion

Step 4: For remaining relationships that have (1,1) participation by one entity set, post the related entity's primary key into the (1,1) entity's table as a foreign key

I.e., for the following participation cardinality patterns

(0,N)-(1,1) (1,N)-(1,1) (1,1)-(0,N) (1,1)-(1,N) (0,1)-(1,1) (1,1)-(0,1)

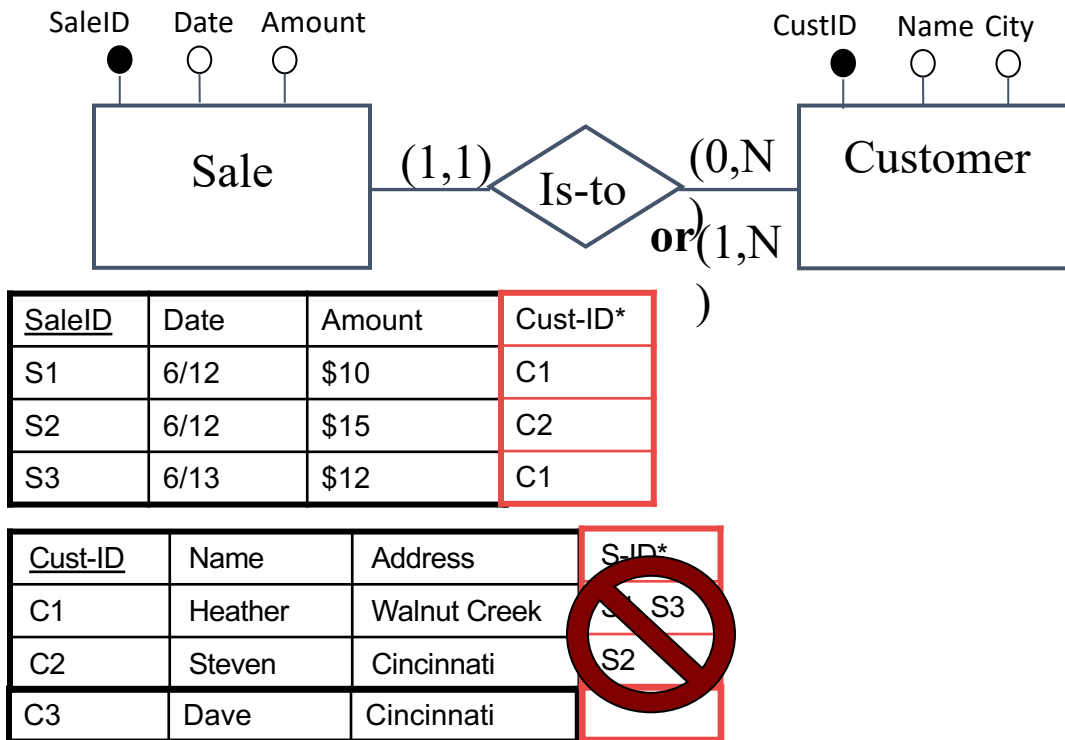
Do NOT make a separate table

Post a foreign key INTO the (1,1) entity's table from the other entity's table

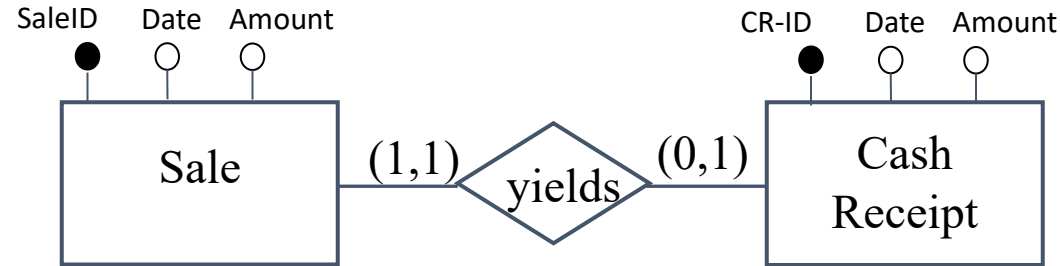
Redundancy is avoided and load is not an issue if you follow this instruction

If you post the opposite direction, either redundancy [for N maximums] OR load [for 0 minimums] will be a problem

Example 1: Posting into a (1,1)



Example 2: Posting into a (1,1)



<u>SaleID</u>	Date	Amount	CR-ID*
S1	6/12	\$10	CR1
S2	6/12	\$15	CR2
S3	6/13	\$12	CR3

<u>CR-ID</u>	Date	Amount	S-ID*
CR1	6/12	\$10	
CR2	6/12	\$15	S2
CR3	6/13	\$12	
CR4	6/13	\$1,000	



Relationship Conversion

Step 5: For remaining relationships that have (0,1) participation by one or both of the entities, consider load

I.e., for the following participation cardinality patterns

(0,N)-(0,1) (1,N)-(0,1) (0,1)-(0,N) (0,1)-(1,N) (0,1)-(0,1)

The rule for maximum cards requires posting into a (0,1) or making a separate table; you CANNOT post into the (0,N) or (1,N)

The rule for minimum cards says you really shouldn't post into the (0,1) because it will create null values that waste valuable space in the database

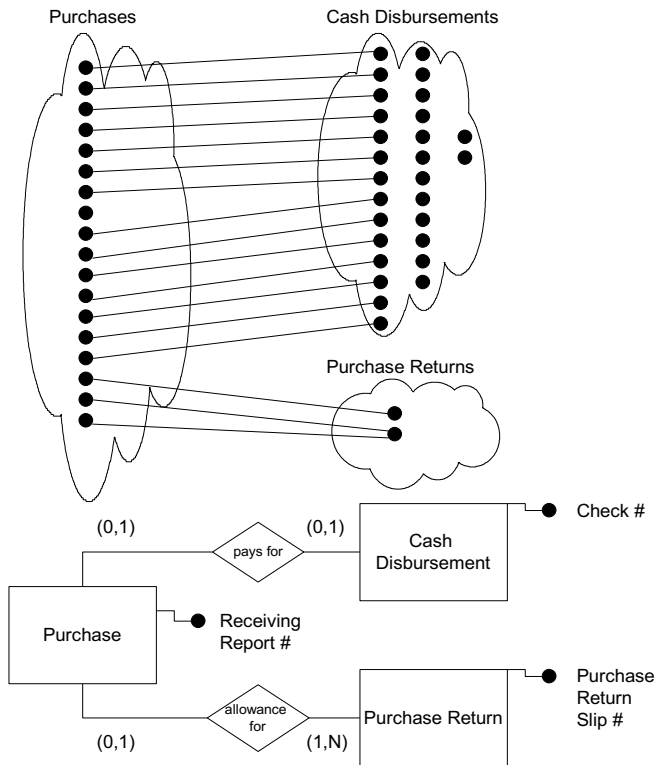
However, if a separate table would waste more space, then it is better to follow the maximum rule and break the minimum rule

5A: Post the related entity's primary key into the (0,1) entity's table as a foreign key for any relationships for which that results in a high load

5B: Create a separate table for any relationships for which posting a foreign key results in low load

Note: For (0,1)-(0,1), step 5A, post whichever direction results in highest load; if neither direction yields high load, then follow step 5B

Example: Load Considerations



Some cash disbursements (13/26) pay for purchases

If we post Receiving Report# into Cash Disbursement, 13 out of 26 will be non-null

This is a medium load

Might be worth breaking minimum rule

Consider other posting option

Most purchases (14/18) result in cash disbursements

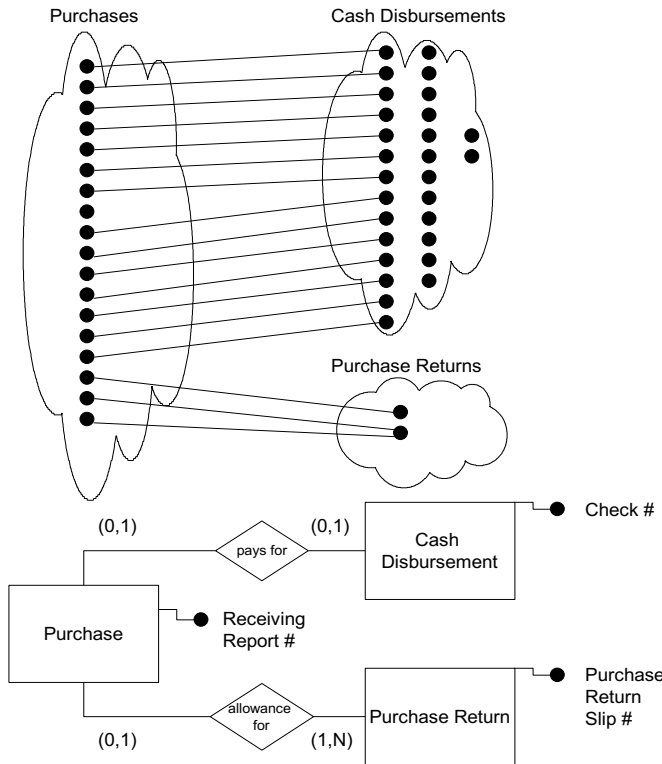
If we post Check# into Purchase, 14 out of 18 will be non-null

This is a high load

Worth breaking the minimum rule

Conclusion: post Check# into Purchase table to represent the "pays for" relationship

Example: Load considerations



Few purchases (3/18) result in purchase returns

If we post Purchase Return Slip# into Purchase, only 3 out of 18 will be non-null

This is low load

Must either make a separate table or consider posting the other direction

Can't post receiving report# into purchase return because one purchase return slip # can be associated with multiple purchases

Conclusion: Make a separate table to represent the "allowance for" relationship

Relationship Attribute Placement

If relationship becomes
a separate table, then
relationship attributes
are placed in that table

If relationship can be
represented by a posted
foreign key,
relationship attribute is
posted alongside the
foreign key

Fixing One Fact Multiple Places

Employee

<u>EmpID</u>	EmpName	Payrate	Hours Worked	Dept#	DeptName
8532	Andy	\$13	36	D423	Audit
7352	Jennifer	\$14	45	D423	Audit
215	Arlie	\$20	50	D777	ISAAS
4332	Craig	\$18	60	D821	Tax
74	Steven	\$22	64	D821	Tax

What facts are in multiple places in this table?

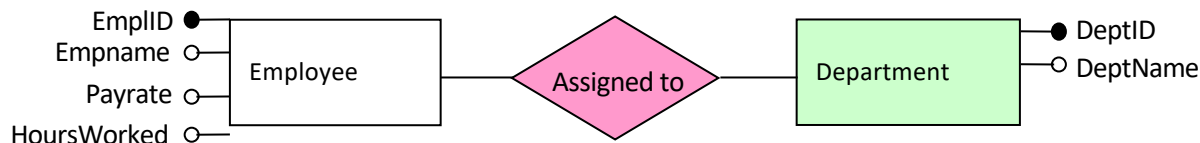
Reverse engineer to get the ER model that this table must represent

Is the ER model that results in this table correct?

What SHOULD the ER model have been instead?

What is the correct relational model?

Fixing One Fact Multiple Places



Employee

<u>EmplID</u>	EmpName	Payrate	Hours Worked	Dept#
8532	Andy	\$13	36	D423
7352	Jennifer	\$14	45	D423
215	Arlie	\$20	50	D777
4332	Craig	\$18	60	D821
74	Steven	\$22	64	D821

Department

<u>Dept#</u>	DeptName
D423	Audit
D777	ISAAS
D821	Tax

Normal Forms and Normalization

Normalization

A two step process that puts data into tabular form by removing repeating groups and then removes duplicated data from the relational tables.

Normalization theory is based on the concepts of **normal forms**.

There are currently 5 normal forms that have been defined.
(we cover 3.5)

Normalization

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified or updated.

This means that all tables in a relational database should be in the third normal form (3NF).

Normalization

A relational table is in 3NF if and only if all non-key columns are:

**Mutually independent and
Fully dependent upon the primary key**

Mutual independence means that no non-key column is dependent upon any combination of the other columns

The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF

In order to better understand the 2NF and higher forms, it is necessary to understand the concepts of functional dependencies

Functional Dependencies

the basis for the first three normal forms.

A column, Y, of the relational table R is said to be **functionally dependent** upon column X of R if and only if each value of X in R is associated with precisely one value of Y at any given time.

It is the same as saying the values of column X identify the values of column Y.

If column X is a primary key, then all columns in the relational table R must be functionally dependent upon X.

$$R.x \rightarrow R.y$$

column x functionally determines (identifies) column y

Example FD

Drinkers(name, addr, drinkLiked, manf, favdrink).

Reasonable FD's to assert:

1. name \rightarrow addr
2. name \rightarrow favdrink
3. drinkLiked \rightarrow manf

Functional Dependency: The value of one attribute (the *determinant*) determines the value of another attribute

Example DF

name	addr	drinkLiked	manf	favDrink
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

Because name \rightarrow addr

Because name \rightarrow favDrink

Because drinkLiked \rightarrow manf

Basic Idea

To know what FD's hold in a projection, we start with given FD's and find all FD's that follow from given ones. Then, restrict to those FD's that involve only attributes of the projected schema.

Normalization Goals

- Eliminate redundant data (for example, storing the same data in more than one table) and
- Ensure data dependencies make sense (only storing related data in a table).

they reduce the amount of space a database consumes and ensure that data is logically stored and easy to maintain

First Normalization Form 1FN

Eliminate duplicative columns from the same table.

No multivalued attributes

Every attribute value is atomic

(i.e. no sets of values within a column)

Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

All relations are in 1st Normal Form

Multivalued attributes

<u>Order_ID</u>	<u>Order_</u> Date	<u>Customer_</u> ID	<u>Customer_</u> Name	<u>Customer_</u> Address	<u>Product_ID</u>	<u>Product_</u> Description	<u>Product_</u> Finish	<u>Unit_</u> Price	<u>Ordered_</u> Quantity
1006	10/24/2004	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
					5	Writer's Desk	Cherry	325.00	2
					4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

Note: NOT 1FN, this is NOT a relation

1st NF example

Figure 5-26 INVOICE relation (1NF) (Pine Valley Furniture Company)

<u>Order_ID</u>	Order_ Date	Customer_ ID	Customer_ Name	Customer_ Address	<u>Product_ID</u>	Product_ Description	Product_ Finish	Unit_ Price	Ordered_ Quantity
1006	10/24/2004	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

Note: this is relation, but not a well-structured one

Second normal form 2NF

deals with relationships between composite key columns and non-key columns:

- Meet all the requirements of the first normal form.

- Any non-key columns must depend on the entire primary key. In the case of a composite primary key, this means that a non-key column cannot depend on only part of the composite key.

- Create relationships between these new tables and their predecessors through the use of foreign keys.

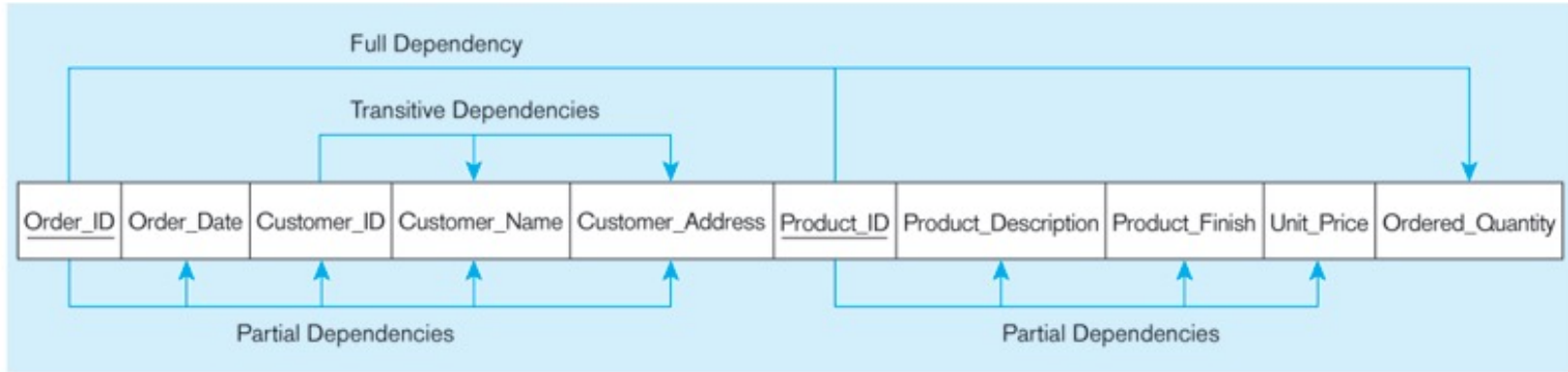
- A relation R is in 2nf if every non-primary attribute A in R is fully Functionally dependent on the primary key.

Second Normal Form

1NF PLUS *every non-key attribute is fully functionally dependent on the ENTIRE primary key*

Every non-key attribute must be defined by the entire key, not by only part of the key

No partial functional dependencies



Order_ID → Order_Date, Customer_ID, Customer_Name, Customer_Address

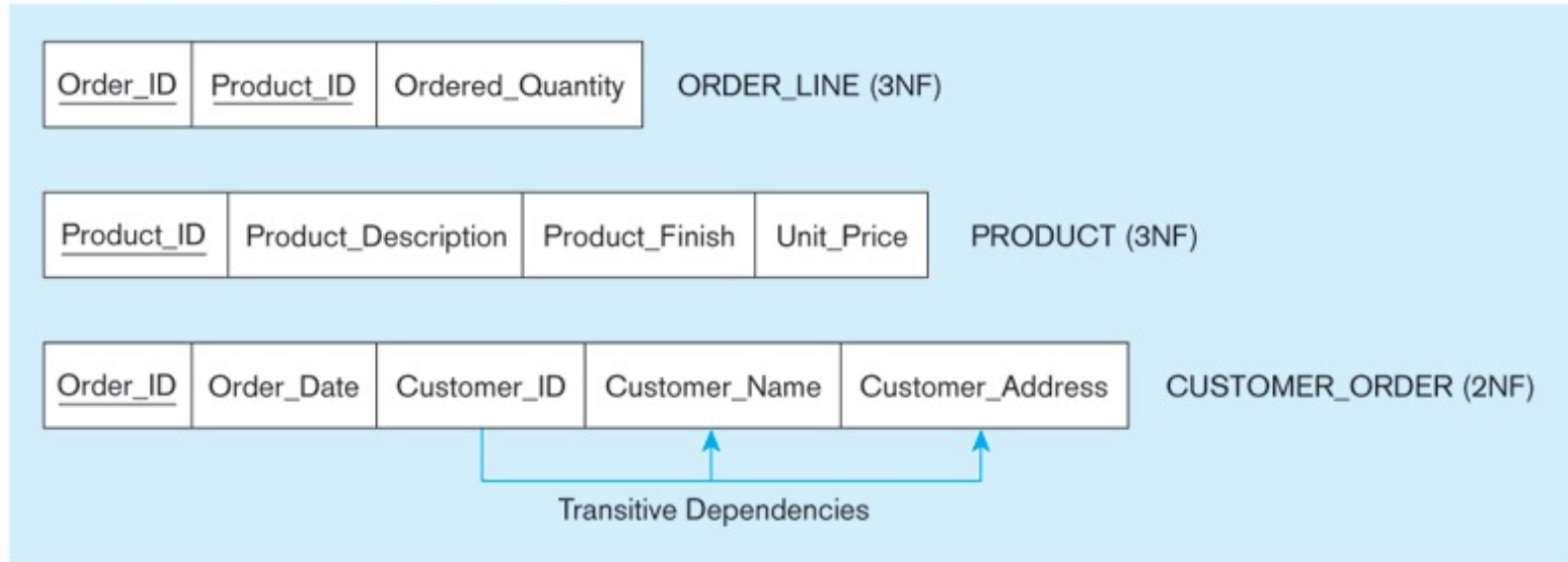
Customer_ID → Customer_Name, Customer_Address

Product_ID → Product_Description, Product_Finish, Unit_Price

Order_ID, Product_ID → Order_Quantity

Therefore, NOT in 2nd Normal Form

Getting it into Second Normal Form



Partial Dependencies are removed, but there are still transitive dependencies

Third normal form 3NF

Remove columns that are not dependent upon the primary key.

Third Normal Form (3NF) requires that all columns depend directly on the primary key.

Example:

Publisher (Publisher_ID, Name, Address, City, State, Zip)

Zip (Zip, City, State)

Third Normal Form

2NF plus *no transitive dependencies* (functional dependencies on non-primary-key attributes)

Note: this is called transitive, because the primary key is a determinant for another attribute, which in turn is a determinant for a third

Solution: non-key determinant with transitive dependencies go into a new table; non-key determinant becomes primary key in the new table and stays as foreign key in the old table

Getting it into Third Normal Form

<u>Order_ID</u>	Order_Date	<u>Customer_ID</u>
-----------------	------------	--------------------

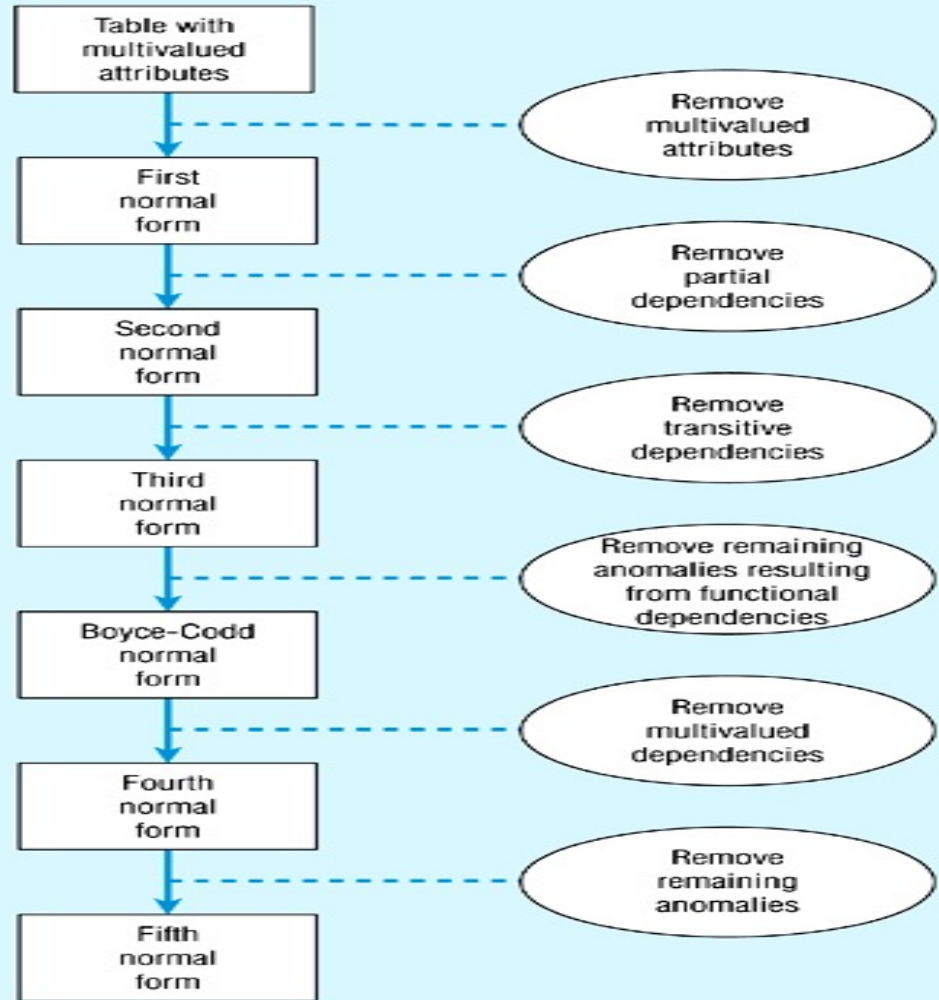
ORDER (3NF)

<u>Customer_ID</u>	Customer_Name	Customer_Address
--------------------	---------------	------------------

CUSTOMER (3NF)

Transitive dependencies are removed

Steps in Normalization



BCNF or 3.5NF

Boyce–Codd normal form (or BCNF or 3.5NF) is a slightly stronger version of the 3NF.

For every one of its dependencies $X \rightarrow Y$, at least one of the following conditions hold:

- $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$)
- X is a superkey for schema R

References

[Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone:](#)
“Database systems”, McGraw-Hill 1999

[R. Elmasri, S.B. Navathe:](#) Fundamentals of Database Systems,
Benjamin-Cummings, 2007

[Daniele Braga, Marco Brambilla, Alessandro Campi](#) “Eserciziario di
Basi di dati”, Esculapio, 2009



POLITECNICO
MILANO 1863

THANKS! QUESTIONS?

Marco Brambilla

@marcobrambi

marco.brambilla@polimi.it

<http://datascience.deib.polimi.it>

<http://home.deib.polimi.it/marcobrambi>