BASI DI DATI

Andrea De Lorenzo, University of Trieste

ORARIO LEZIONI

Giorno	Orario	Aula
Martedì	15:00 - 16:30	Aula E - C1
Mercoledì	11:15 - 12:45	Aula E - C1
Giovedì	10:00 - 11:30	Aula E - C1
Venerdì	14:00 - 15:30	Aula E - C1

http://delorenzo.inginf.units.it/

MODALITÀ LEZIONI

- Lezioni in **presenza**, trasmesse in streaming su MS Teams e registrate
- Registrazioni disponibili per circa 6 mesi nel Team del corso
- Accessi al Team del corso tramite codice

TZ3AP6H

MODALITÀ ESAME

TEST + PROGETTO + ORALE

Test a risposta multipla, basta passarlo una volta
Progetto va consegnato 3 giorni lavorativi prima dell’appello Se l’esame è mercoledì, venerdì è **troppo tardi** per inviare il progetto!

GESTIONE DELLE INFORMAZIONI

L’essere umano **genera** e **gestisce** tante informazioni:

- idee informali
- linguaggio naturale (scritto o parlato, in lingue diverse)
- disegni, grafici, schemi
- numeri
- codici

GESTIONE DELLE INFORMAZIONI

... salvate in tanti modi diversi

- memoria
- carta
- pietra
- scritta sul muro
- elettronica

GESTIONE DELLE INFORMAZIONI

Anche le organizzazioni generano informazioni:

- utenze telefoniche
- conti correnti
- studenti iscritti ad un corso di laurea
- quotazioni di azioni

CODIFICA DELLE INFORMAZIONI

ERA INFORMATICA

Le informazioni vanno codificate

- si aggiungono elementi artificiali
- primo esempio: anagrafe
- nome e cognome
- indirizzo
- codice fiscale

INFORMAZIONE VS DATO

Informazione: notizia, dato o elemento che consente di avere conoscenza più o meno esatta di fatti, situazioni, modi di essere

Dato: elemento di informazione costituito da simboli che debbono essere elaborati

CARTELLI STRADALI IN FINLANDIA

Lun - Ven

Sabato

Festivi

NUMERI

Come codifico i numeri?

- Numeri naturali: facile, in binario
- Numeri interi: devo decidere come rappresentare il segno
- Numeri razionali?

100100111100110011111

DATI E APPLICAZIONI

- I **dati** possono variare nel tempo (es: conto corrente)
- Le **modalità** con cui i dati sono rappresentati sono di solito stabili
- Le **operazioni** sui dati variano spesso (es: ricerche)

separare i dati dalle applicazioni che operano su essi

DATA BASE

GENERICAMENTE:

Collezione di dati, utilizzati per rappresentare le informazioni di interesse per una o più applicazioni di una organizzazione.

- schede perforate
- file CSV
- foglio di calcolo
- file XML
- Access

DATABASE MANAGEMENT SYSTEM

Software in grado di gestire collezioni di dati che siano:

- **grandi**: di dimensioni (molto) maggiori della memoria centrale
- **persistenti**: con un periodo di vita indipendente dalla singole esecuzioni dei programmi che le utilizzano
- **condivise**: utilizzate da applicazioni diverse

BASE DI DATI

GENERICAMENTE

Collezione di dati, utilizzati per rappresentare le informazioni di interesse per una o più applicazioni di una organizzazione

PER NOI

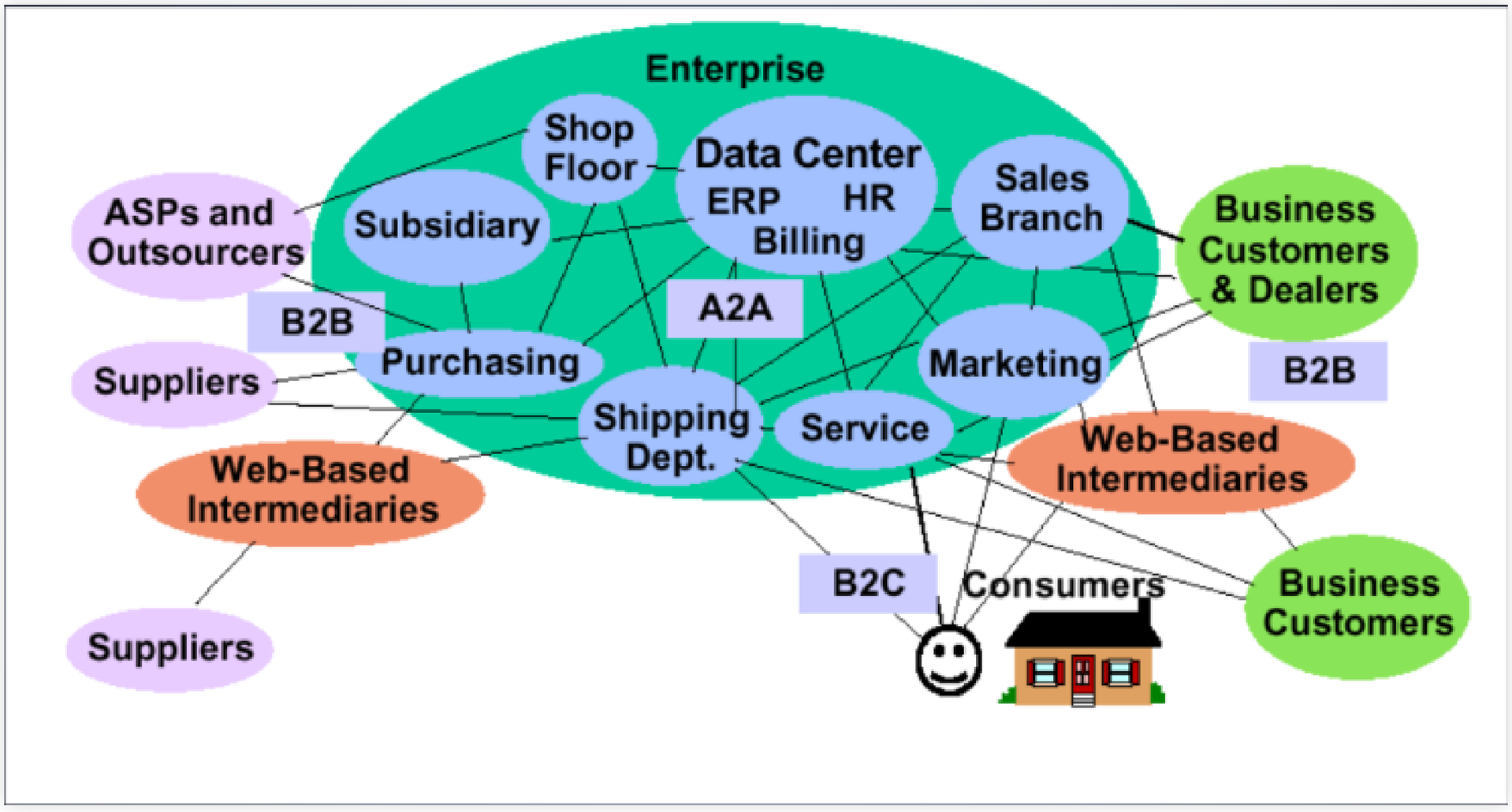
Collezione di dati gestita da un DBMS

DATABASE MANAGMENT SYSTEM

Un DBMS deve garantire:

- **affidabilità**: resistenza a malfunzionamenti hardware e software
- **privatezza**: con una disciplina e un controllo degli accessi
- **efficienza**: utilizzando al meglio le risorse di spazio e etempo del sistema
- **efficacia**: rendendo produttive le attività dei suoi utilizzatori

CONDIVISIONE



- più dipartimenti sono interessati agli stessi dati
- una base di dati è una risorsa integrata, condivisa

CONDIVISIONE

- L’integrazione e la condivisione permettono di
  - ridurre la *ridondanza* (evitando ripetizioni)
  - ridurre possibilità di incoerenza (o *inconsistenza*) fra i dati.
- Poiché la condivisione non è mai completa (o comunque non opportuna) i DBMS prevedono meccanismi per
  - *privatezza* dei dati
  - limitazione all’accesso (*autorizzazioni*).
- La condivisione richiede coordinamento degli accessi: *controllo della concorrenza*

EFFICIENZA

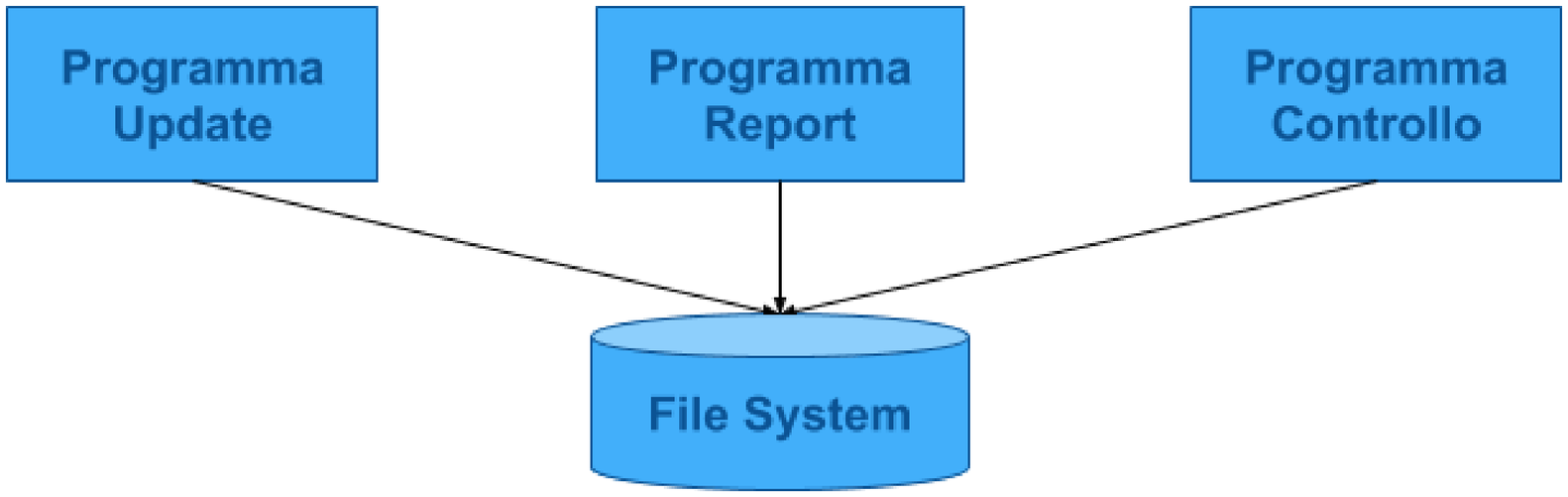
- Si misura in termini di tempo di esecuzione e spazio di memoria (principale e secondaria)
- I DBMS non sono necessariamente più efficienti dei file system
- L’efficienza è il risultato della qualità del DBMS e delle applicazioni che lo utilizzano

DBMS VS FS

	FS	DBMS
Grandi moli di dati	✓	✓
Persistenti	✓	✓
Condivisi	✓	✓
Affidabile	✓	✓
Privatezza	✓	✓
Efficienza	?	?
Efficacia	X	✓

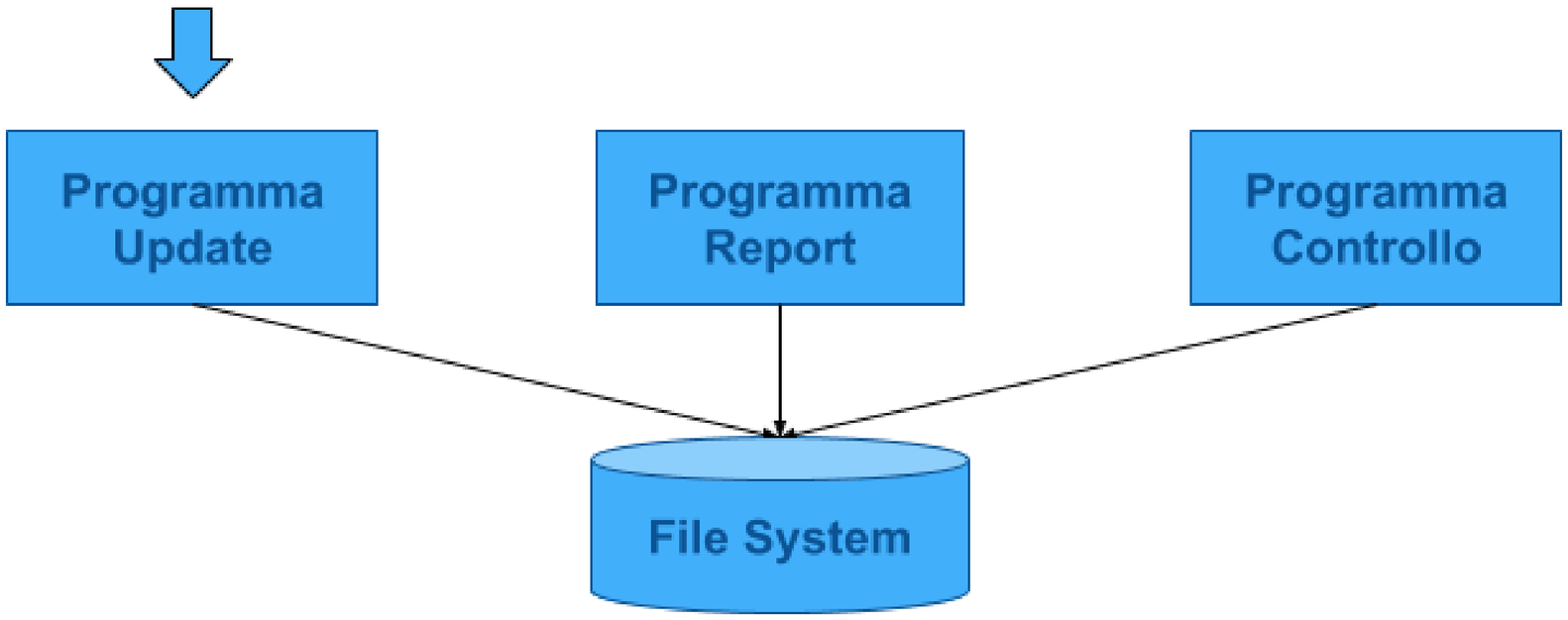
FILE SYSTEM

Descrizione dei dati contenuta nell'applicazione



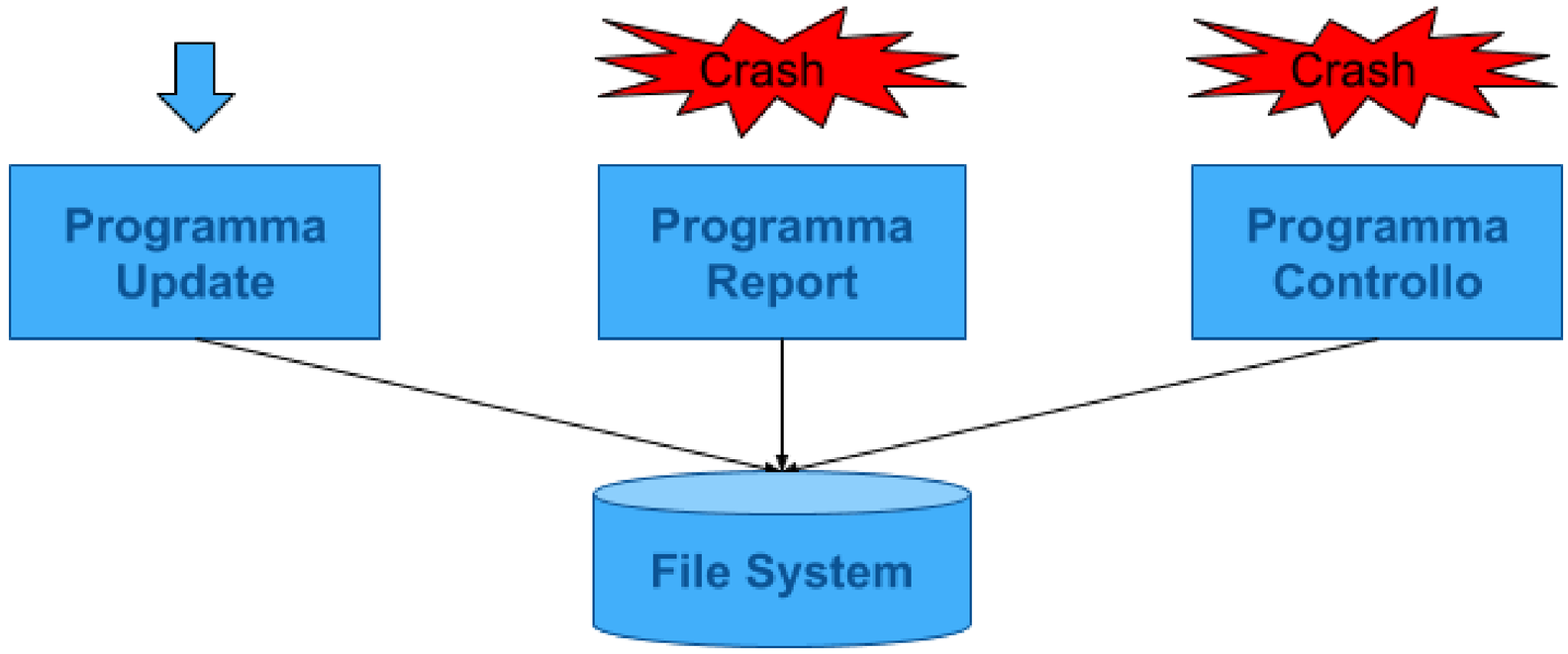
FILE SYSTEM

Descrizione dei dati contenuta nell'applicazione



FILE SYSTEM

Descrizione dei dati contenuta nell'applicazione



DBMS E DESCRIZIONE DEI DATI

- Il DBMS sa come persistere i dati, per l'applicazione è un atto di fede
- I dati sono *INDIPENDENTI* dalla forma fisica
- I programmi parlano con il DBMS per accedere ai dati

MODELLO CONCETTUALE

Analisi del problema

Modello astratto

Non dipende dallo strumento utilizzato

MODELLO LOGICO

Come rappresentare i dati individuati con il modello concettuale

- Livello intermedio tra utente e implementazione
- sottintende una specifica rappresentazione dei dati (tabelle, alberi, grafi, oggetti, …)

DATABASE SYSTEM

- Software
  - DBMS: interposto tra il DB e l'utente
  - Utility di supporto (sviluppo, backup)
- Utenti
  - Progettista
  - Sviluppatore
  - Amministratore
  - Utente finale

DATABASE SYSTEM

- Schemi (struttura dei dati)
- Dati
    - come vengono salvati
    - condivisione
    - concorrenza
    - ridondanza
- Hardware

BASE DI DATI

- **Genericamente**: collezione di dati, utilizzati per rappresentare le informazioni di interesse per una o più applicazioni di una organizzazione.
- **Per noi**: collezione di dati gestita da un DBMS
- **Per noi - 2**: collezione di dati *persistenti* usata dal sistema di una azienda e gestita da un DBMS

RIASSUNTO

- Data base
  - Dati, solo dati, niente altro che i dati
- Data Base Managmnet System
  - Software che gestisce i dati
    - Diversi vendor: IBM, Oracle, Microsoft
- Data Base System
  - DB + DBMS

VANTAGGI E SVANTAGGI

- Vantaggi
  - Dati sono risorsa in comune
  - DB fornisce un modello unificato del business
  - Controllo centralizzato dei dati, quindi standardizzazione ed economie di scala
  - Riduzione ridondanza ed inconsistenza
  - Indipendenza dei dati
- Svantaggi
  - Costo e complessità
  - Servizi ridondanti/non necessari

VERO BENEFICIO

INDIPENDENZA DEI DATI!

VERO BENEFICIO

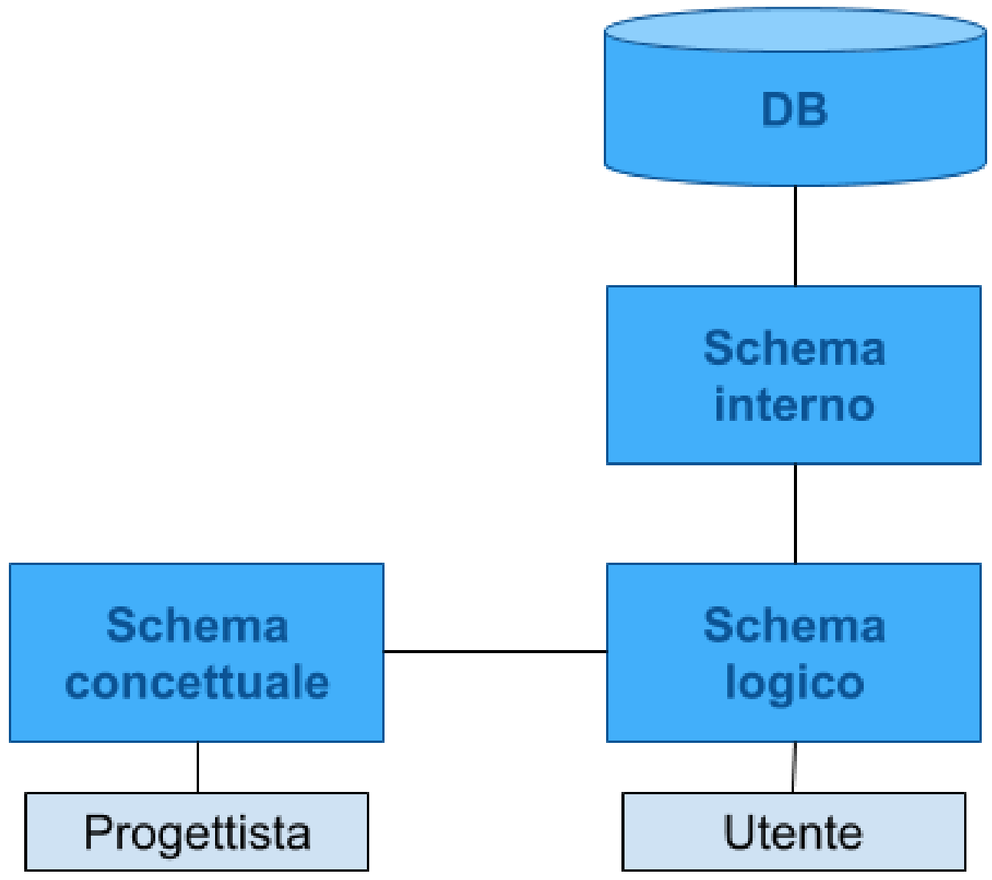
Nei vecchi sistemi il modo in cui venivano organizzati i dati e le tecniche per accedervi facevano parte della logica e del codice del programma.

SCHEMA ED ISTANZA

In ogni base di dati esistono:

- Schema:
    - invariante nel tempo
    - descrive la struttura
    - es: intestazione tabelle
- Istanza:
    - i valori attuali
    - possono cambiare
    - es: contenuto delle tabelle

SCHEMI


DB
Schema interno
Schema concettuale
Schema logico
Progettista
Utente

SCHEMI CONCETTUALI

Permettono di rappresentare i dati in modo indipendente da ogni sistema:

- cercando di descrivere i concetti del mondo reale
- sono utilizzati nelle fasi preliminare di progettazione

    Modello più diffuso: Entity-Relationship

SCHERMI INTERNI (O FISICI)

Rappresentazione dello schema logico per mezzo di strutture di memorizzazione

- file CSV
- file XML
- file binari

SCHEMI LOGICI

Come è organizzato il DB. Diverse soluzioni:

- gerarchico
- reticolare
- relazionale
- ad oggetti

INDIPENDENZA

Lo schema logico è **INDIPENDENTE** da quello fisico

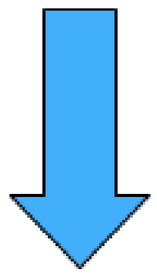
Es: una tabella è utilizzata sempre allo stesso modo qualunque sia la sua realizzazione fisica (che può variare nel tempo)

INDIPENDENZA

PROGETTISTA DB ≠ SVILUPPATORE SW

VISTA

L'amministratore del DB può modificare la struttura interna dei dati senza toccarne la visibilità esterna



IMMUNITÀ DELLE APPLICAZIONI A MODIFICHE DI STRUTTURA

VISTA

Corso	Docente	Aula
Reti	Bartoli	N3
Programmazione	Medvet	N3
ML	Medvet	G

Nome	Edificio	Piano
DS1	H3	3
N3	C2	2
G	Principale	PT

Corso	Docente	Aula	Edificio	Piano
Reti	Bartoli	N3	C2	2
Programmazione	Medvet	N3	C2	2
ML	Medvet	G	Principale	PT

VISTA

Nome	Cognome	Matricola	Media	ISEE
Tizio	Caio	IN0001	30	5000€
Bubba	Gump	IN0003	27	1000000e
Jean Luc	Picard	IN0004	19	40000€

Nome	Cognome	Matricola	ISEE
Tizio	Caio	IN0001	5000€
Bubba	Gump	IN0003	1000000e
Jean Luc	Picard	IN0004	40000€

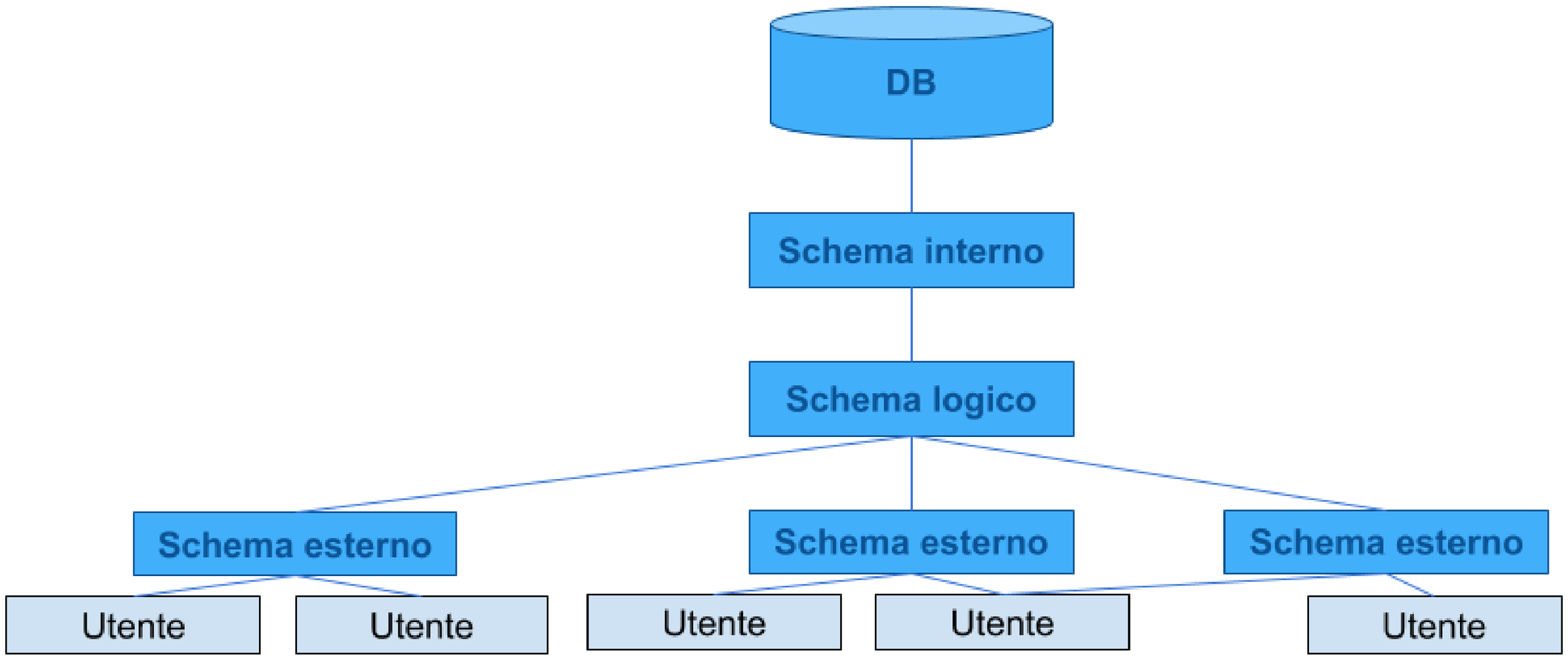
Nome	Cognome	Matricola	Media
Tizio	Caio	IN0001	30
Bubba	Gump	IN0003	27
Jean Luc	Picard	IN0004	19

SCHEMA ESTERNO

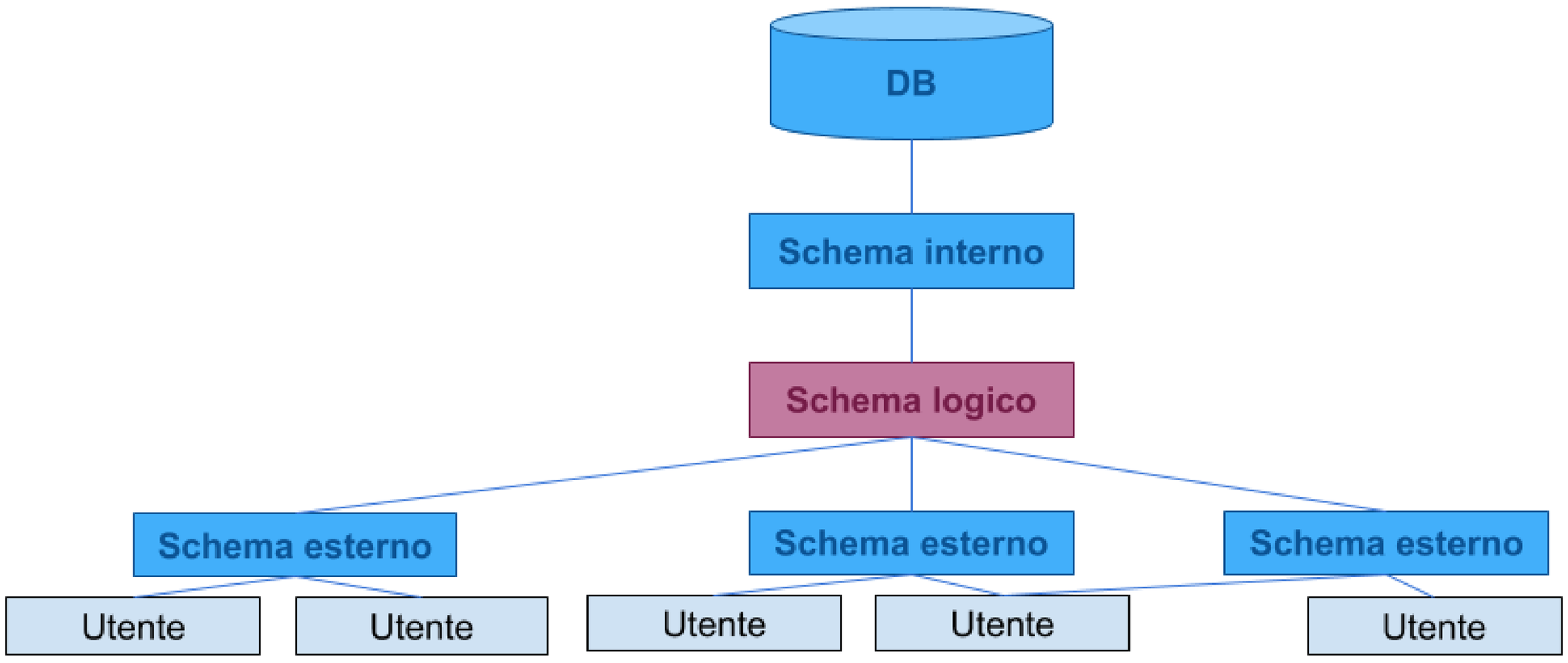
== VISTA

- Descrive parte della base di dati di un modello logico
- NON è una copia dei dati

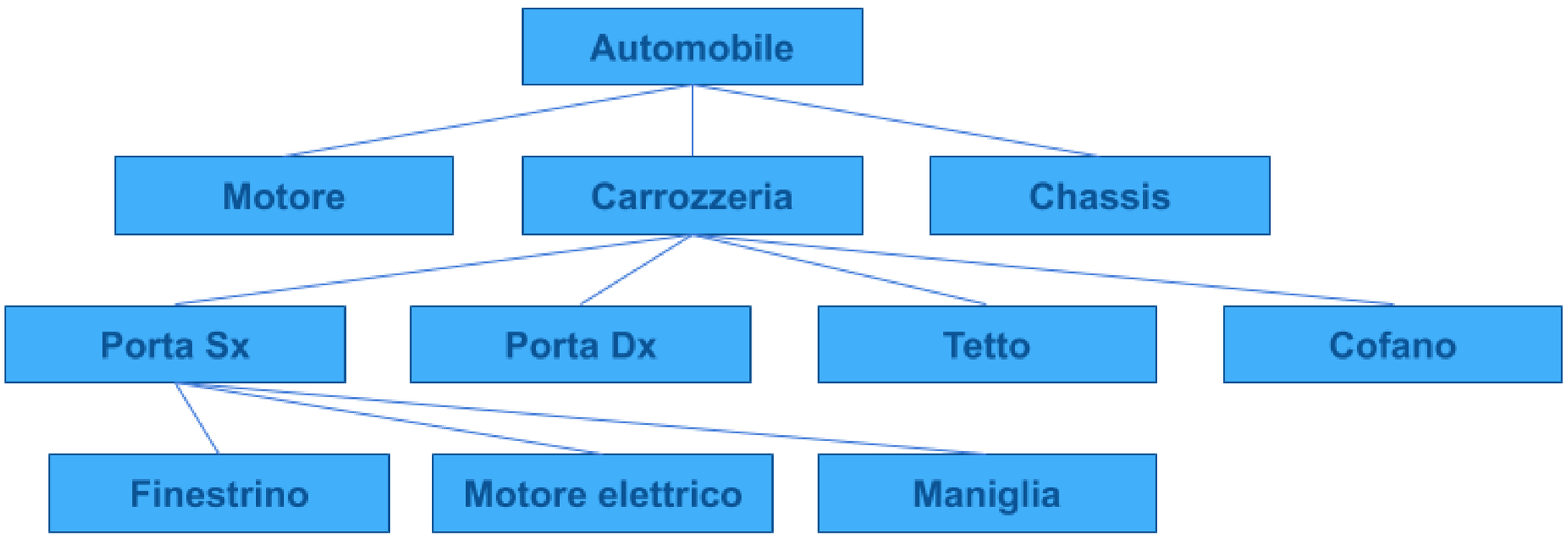
ARCHITETTURA ANSI/SPARC



SCHEMI LOGICI



SCHEMI LOGICI GERARCHICI

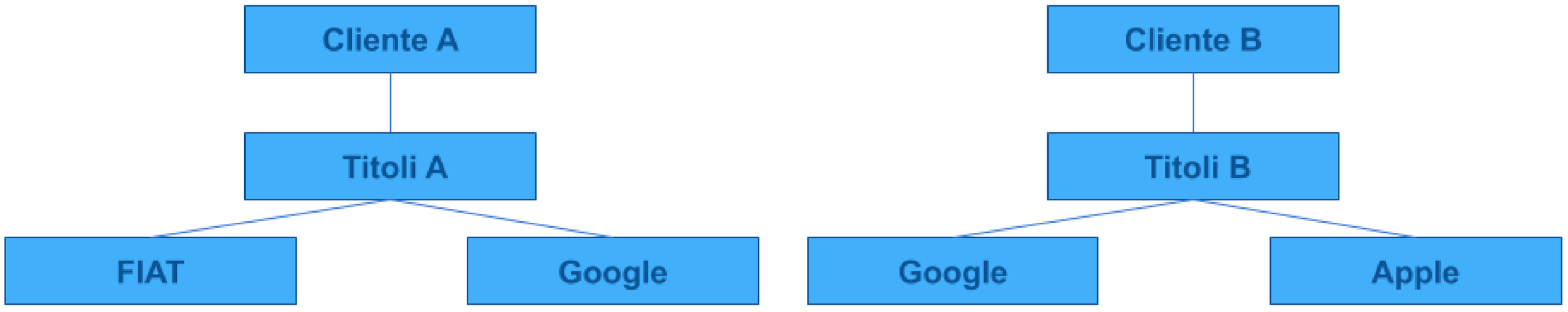


SCHEMI LOGICI GERARCHICI

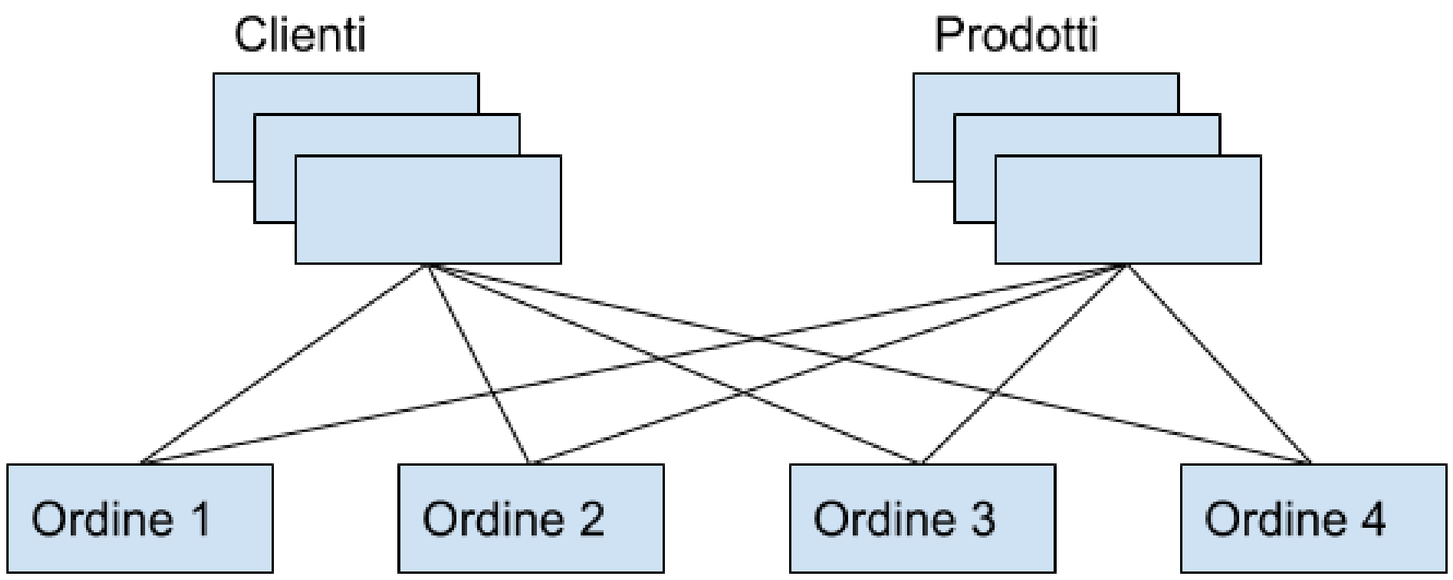
Problemi:

- accesso sequenziale: per arrivare al figlio devo attraversare tutti i nodi
- modifica parziale complicata
- cancellazione gerarchica
- stretto legame tra programma e struttura del database
- ridondanza

SCHEMI LOGICI GERARCHICI (RIDONDANZA)

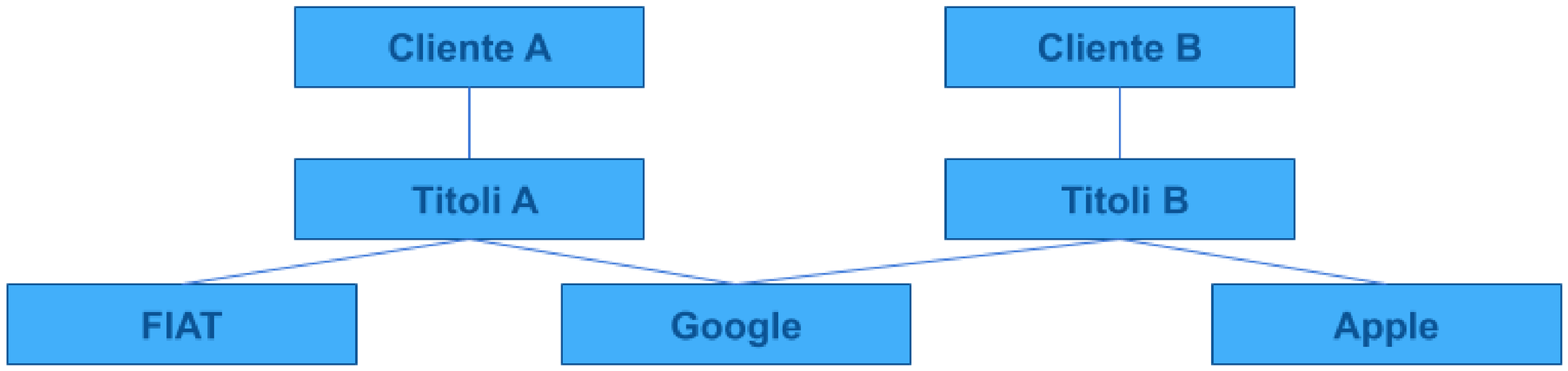


SCHEMI LOGICI RETICOLARI



- COBOL, 1970
- nodi collegati da PUNTATORI
- navigazione bi-direzionale

SCHEMI LOGICI RETICOLARI



SCHEMI LOGICI RELAZIONALI

CODD, 1980

Liberarsi dei puntatori fisici

- i dati sono organizzati in **tabelle** di valori
- le operazioni vengono eseguite sulle **tabelle**
- i risultati delle operazioni sono **tabelle**
- i riferimenti tra dati in strutture (tabelle) diverse sono rappresentati con **valori**

ELEMENTI DI UN DBR

Tabelle: organizzazione rettangolare di dati

- Record (righe) e campi (colonne) e domini dei dati
- I campi definiscono univocamente il tipo dei dati (dominio)
- I campi hanno un nome ed un ordine, le righe no
- Esistono tabelle vuote

ELEMENTI DI UN DBR

CHIAVI PRIMARIE

- Una (o più) colonne che identificano **UNIVOCAMENTE** il record
- Non possono essere duplicate
- Una tabella in cui ogni riga è diversa dalle altre è detta **RELAZIONE**

ELEMENTI DI UN DBR

RELAZIONI

- Non esistono relazioni padre-figlio
- Le relazioni sono rappresentate da DATI COMUNI manipolabili

CHIAVI ESTERNE (SECONDARIE, FOREIGN KEY)

- Una colonna in una tabella il cui valore corrisponde ad una chiave primaria
- Sono fondamentali nella creazione delle relazioni

DODICI REGOLE DI CODD

DODICI REGOLE DI CODD

1 - INFORMAZIONI

Tutte le informazioni in un DBR sono rappresentate esplicitamente da valori in tabelle (DEFINIZIONE)

DODICI REGOLE DI CODD

2 - ACCESSO GARANTITO

Ciascun valore deve essere raggiunto univocamente da un nome di tabella, chiave primaria e nome di colonna (CHIAVI PRIMARIE)

DODICI REGOLE DI CODD

3 - VALORI NULL

Sono supportati per rappresentare informazioni mancanti indipendentemente dal tipo di dato

DODICI REGOLE DI CODD

4 - SYSTEM TABLE

Un data base relazionale deve essere strutturato logicamente come i dati e gestibile con lo stesso linguaggio

DODICI REGOLE DI CODD

5 - LINGUAGGIO DI INTERROGAZIONE STANDARD

Un DBR può supportare diversi linguaggi, ma deve supportare un linguaggio “English like” dove sia possibile (DEFINIZIONE DI SQL):

- Definire dati
- Definire viste
- Manipolare dati
- Gestire l’integrità

DODICI REGOLE DI CODD

6 - VISTE MODIFICABILI

Le viste che sono modificabili teoricamente dall’utente lo devono essere anche dal sistema (cruciale per campi calcolati);

DODICI REGOLE DI CODD

6 - VISTE MODIFICABILI

Affinché una vista sia modificabile, il DBMS deve essere in grado di tracciare ciascuna colonna e ciascuna riga **UNIVOCAMENTE** fino alle tabelle origine

VISTE MODIFICABILI

Corso	Docente	Aula
Reti	Bartoli	N3
Programmazione	Medvet	N3
ML	Medvet	G

Nome	Edificio	Piano
DS1	H3	3
N3	C2	2
G	Principale	PT

Corso	Docente	Aula	Edificio	Piano
Reti	Bartoli	N3	C2	2
Programmazione	Medvet	N3	C2	2
ML	Medvet	G	Principale	PT

VISTE MODIFICABILI

Studente	Esame	Voto
Scaini	Reti	30
Scaini	ML	28
Bassi	ML	30
Bassi	Reti	30

Studente	Media
Scaini	29
Bassi	30

DODICI REGOLE DI CODD

7 - INSERIMENTO E UPDATE DA LINGUAGGIO

Inserire e aggiornare devono avere la stessa logica “a righe” dell’estrazione (SET ORIENTED)

DODICI REGOLE DI CODD

8 - INDIPENDENZA FISICA DEI DATI

I programmi applicativi non devono sentire alcuna modifica fatta sul metodo e la locazione fisica dei dati

DODICI REGOLE DI CODD

9 - INDIPENDENZA LOGICA DEI DATI

Le modifiche al livello logico non devono richiedere cambiamenti non giustificati alle applicazioni che utilizzano il database (VISTE)

DODICI REGOLE DI CODD

10 - INTEGRITÀ

Vincoli di integrità devono essere implementabili sul motore (cruciale)

DODICI REGOLE DI CODD

11 - INDIPENDENZA DI LOCALIZZAZIONE

La distribuzione di porzioni del database su una o più allocazione fisiche o geografiche deve essere invisibile agli utenti del sistema

DODICI REGOLE DI CODD

12 - DEVE PREVENIRE ACCESSI NON DESIDERATI:

Garantisce l’impossibilità di bypassare le regole di integrità

RIASSUNTO: DB RELAZIONALE

Data Base dove tutti i dati visibili all’utente sono organizzati strettamente in tabelle di valori, e dove tutte le operazioni vengono eseguite su tabelle e danno come risultato tabelle.

RELAZIONE

RELATION

Relazione matematica (teoria degli insiemi)

RELATIONSHIP

Associazione nel modello Entity-Relationship

RELAZIONE MATEMATICA

- $D_1, \ldots, D_n$ (n insiemi anche distinti) sono i **domini**
- prodotto cartesiano $D_1 \times \cdots \times D_n$
  - insieme di tutte le n-uple $(d_1, \ldots, d_n)$ tali che $d_1 \in D_1, \ldots, d_n \in D_n$
- relazione matematica su $D_1, \ldots, D_n$:
  - un sottoinsieme di $D_1 \times \cdots \times D_n$

RELAZIONE MATEMATICA (ESEMPIO)

$$D_1 = \{a, b\}$$

$$D_2 = \{x, y, z\}$$

$$D_1 \times D_2$$

a	x
a	y
a	z
b	x
b	y
b	z

RELAZIONE MATEMATICA (ESEMPIO)

$$D_1 = \{\mathrm{a}, \mathrm{b}\}$$

$$D_2 = \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$$

$$r \subseteq D_1 \times D_2$$

a	x
a	z
b	y

RELAZIONE MATEMATICA (PROPRIETÀ)

- una relazione matematica è un insieme di n-uple ordinate:
  - $(d_1, \ldots, d_n) \mid d_1 \in D_1, \ldots, d_n \in D_n$
- una relazione è un insieme:
  - non c'è ordinamento tra le n-uple
  - le n-uple sono distinte
  - ogni n-upla è ordinata: i-esimo valore proviene dall'i-esimo dominio

RELAZIONE MATEMATICA (ESEMPIO)

$$\text{Partite} \subseteq \text{string} \times \text{string} \times \text{int} \times \text{int}$$

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

- ciascuno dei domini ha due ruoli diversi, distinguibili attraverso la posizione
- la struttura è posizionale

STRUTTURA NON POSIZIONALE

A ciascun dominio si associa un nome (attributo), che ne descrive il "ruolo"

Casa	Fuori	RetiCasa	RetiFuori
Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	2	0
Roma	Milan	0	1

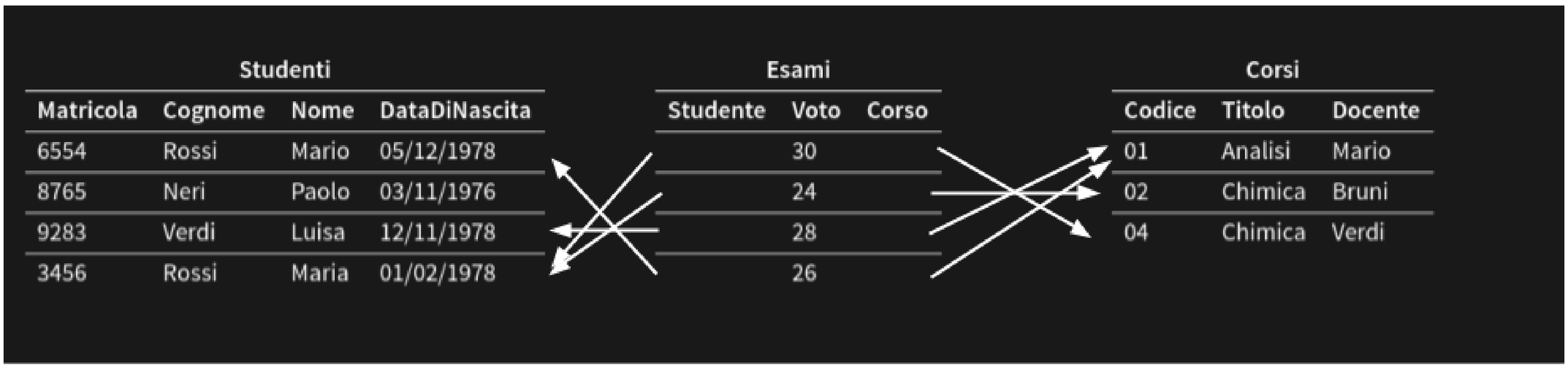
TABELLE E RELAZIONI

- Una tabella è una relazione se:
  - i valori di ogni colonna sono omogenei
  - le righe sono diverse fra di loro
  - le intestazioni delle colonne sono diverse tra di loro
- In una tabella che rappresenta una relazione:
  - l’ordinamento tra le righe è irrilevante
  - l’ordinamento tra le colonne è irrilevante

RELAZIONE

- Relation: relazione matematica (teoria degli insiemi)
- Relationship: rappresenta una associazione nel modello Entity-Relationship

PRIMA


Studenti
Matricola Cognome Nome DataDiNascita
6554 Rossi Mario 05/12/1978
8765 Neri Paolo 03/11/1976
9283 Verdi Luisa 12/11/1978
3456 Rossi Maria 01/02/1978
Esami
Studente Voto Corso
30
24
28
26
Corsi
Codice Titolo Docente
01 Analisi Mario
02 Chimica Bruni
04 Chimica Verdi

MODELLO BASATO SU VALORI

I riferimento fra dati in relazioni diverse sono rappresentati per mezzo di valori dei domini che compaiono nelle n-uple

DB RELAZIONALE

Studenti

Matricola	Cognome	Nome	DataDiNascita
6554	Rossi	Mario	05/12/1978
8765	Neri	Paolo	03/11/1976
9283	Verdi	Luisa	12/11/1978
3456	Rossi	Maria	01/02/1978

Esami

Studente	Voto	Corso
3456	30	04
3456	24	02
9283	28	01
6554	26	01

Corsi

Codice	Titolo	Docente
01	Analisi	Mario
02	Chimica	Bruni
04	Chimica	Verdi

STRUTTURA BASATA SU VALORI

VANTAGGI

- indipendenza dalla struttura fisiche (si potrebbe avere anche con puntatori HL)
- si rappresenta solo ciò che rilevante dal punto di vista dell’applicazione
- utente finale vede stessi dati del programmatore
- portabilità dei dati tra sistemi
- puntatori direzionali

RELAZIONE SU SINGOLI ATTRIBUTI

Studenti

Matricola	Cognome	Nome	DataDiNascita
6554	Rossi	Mario	05/12/1978
8765	Neri	Paolo	03/11/1976
9283	Verdi	Luisa	12/11/1978
3456	Rossi	Maria	01/02/1978

Studenti Lavoratori

Matricola
6554
3456

STRUTTURE NIDIFICATE

Da Filippo Via Roma 2, Roma		
Ricevuta Fiscale 1235 del 12/01/2020		
3	Coperti	3,00
2	Antipasti	6,20
3	Primi	12,00
2	Bistecche	18,00
-		
-		
	Totale	39,20

Da Filippo Via Roma 2, Roma		
Ricevuta Fiscale 1240 del 13/10/2020		
2	Coperti	2,00
2	Antipasti	7,00
2	Primi	8,00
2	Orate	20,00
2	Caffè	2,00
-		
	Totale	39,00

STRUTTURE NIDIFICATE

Ricevute

Numero	Data	Qtà	Descrizione	Importo	Totale
1235	12/10/2020	3	Coperti	3,00	39,20
		2	Antipasti	6,20	
		3	Primi	12,00	
		2	Bistecche	18,00	
1240	12/10/2020	2	Coperti	2,00	39,00
		...	...	...	

I valori devono essere semplici

RELAZIONI E STRUTTURE NIDIFICATE

Ricevute

Numero	Data	Totale
1235	12/10/2020	39,20
1240	12/10/2020	39,00

Dettaglio

Numero	Qtà	Descrizione	Importo
1235	3	Coperti	3,00
1235	2	Antipasti	6,20
1235	3	Primi	12,00
1235	2	Bistecche	18,00
1240	2	Coperti	2,00
...	...	...	...

SIAMO STAI BRAVI?

ABBIAMO RAPPRESENTATO TUTTI GLI ASPETTI DELLE RICEVUTE?

Dipende da cosa ci interessa:

- l’ordine delle righe è rilevante?
- possono esistere linee ripetute?
- Al bar, al servizio al tavolo, ad un gruppo:
  - Cliente: “Una birra”
  - Cameriere: “Se volete altre birre ditelo subito altrimenti non posso aggiungerle!”
- Sono possibili rappresentazioni diverse

RELAZIONI E STRUTTURE NIDIFICATE

Ricevute

Numero	Data	Totale
1235	12/10/2020	39,20
1240	12/10/2020	39,00

Dettaglio

Numero	Riga	Qtà	Descrizione	Importo
1235	1	3	Coperti	3,00
1235	2	2	Antipasti	6,20
1235	3	3	Primi	12,00
1235	4	2	Bistecche	18,00
1240	1	2	Coperti	2,00
...	...	...	...	...

INFORMAZIONI INCOMPLETE

OVVERO: GESTIRE I VALORI NULL

Ogni elemento in una tabella può essere o un valore del dominio oppure il valore nullo NULL

INFORMAZIONE INCOMPLETA

IL MODELLO RELAZIONALE IMPONE UNA STRUTTURA RIGIDA

- Le informazioni sono rappresentate per mezzo di n-uple
- Solo alcuni formati di n-upla sono ammessi: quelli che corrispondono agli schemi di relazione
- I dati disponibili possono non corrispondere al formato previsto

ESEMPIO

Capi di Stato

Nome	SecondoNome	Cognome
Franklin	Delano	Roosvelt
Winston		Churchill
Charles		De Gaulle
Josip		Stalin

NULL: COME FARE?

E SE USASSI IL NUMERO 0?

NON CONVIENE, ANCHE SE SPESSO SI FA, USARE VALORI DEL DOMINIO (0, STRINGA NULLA, 99, …)

- potrebbero non esistere valori “non utilizzati”
- valori “non utilizzati” potrebbero diventare significativi
- in fase di utilizzo sarebbe necessario tener conto del significato di questi valori

TIPI DI VALORE NULL

(ALMENO) 3 CASI DIFFERENTI

- valore sconosciuto (quanti anni ha?)
- valore inesistente (non ha il secondo nome)
- valore non applicabile (anagrafica unica studenti/professori, i professori hanno ufficio)

I DBMS NON DISTINGUONO I TIPI DI VALORE NULLO

TROPPI VALORI NULL

Studenti

Matricola	Cognome	Nome	DataDiNascita
6554	Rossi	Mario	05/12/1978
9283	Verdi	Luisa	12/11/1978
NULL	Rossi	Maria	01/02/1978

Esami

Studente	Voto	Corso
NULL	30	NULL
NULL	24	02
9283	28	01

Corsi

Codice	Titolo	Docente
01	Analisi	Mario
02	NULL	NULL
04	Chimica	Verdi

VINCOLI DI INTEGRITÀ

Esistono istanze di basi di dati che, pur sintatticamente corrette, non rappresentano informazioni possibili per l’applicazione di interesse.

DB ERRATO

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
7876463	Verdi	Luisa
7876463	Rossi	Maria

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
739430	24		04

VINCOLI DI INTEGRITÀ

PROPRIETÀ CHE DEVE ESSERE SODDISFATTA DALLE ISTANZA CHE RAPPRESENTANO INFORMAZIONI CORRETTE PER L'APPLICAZIONE

Un vincolo è una funzione booleana (un predicato): associa ad ogni istanza il valore vero o falso

VINCOLI DI INTEGRITÀ, PERCHÉ?

- Descrizione più accurata della realtà
- contributo alla “qualità dei dati”
- utili nella progettazione
- usati dai DBMS nelle interrogazioni

VINCOLI DI INTEGRITÀ: NOTA

Alcuni vincoli (ma non tutti) sono supportati dai DBMS

- Possiamo specificare tali vincoli e il DBMS ne impedisce violazione
- Se non supportati, la responsabilità della verifica è dell'utente/programmatore

TIPI DI VINCOLI

- vincoli intrarelazionali
  - vincoli su valori (o di dominio)
  - vincoli di n-upla
- vincoli interrelazionali

VINCOLI DI VALORE

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
7876463	Verdi	Luisa
7876463	Rossi	Maria

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
739430	24		04

$$Voto \geq 18 \;\&\&\; Voto \leq 30$$

VINCOLI DI n-UPLA

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
7876463	Verdi	Luisa
7876463	Rossi	Maria

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
739430	24		04

Lode solo se Voto == 30

VINCOLI DI n-UPLA

Stipendi

Impiegato	Lordo	Ritenute	Netto
Rossi	55.000	12.500	42.500
Verdi	45.000	10.000	35.000
Bruni	47.000	11.000	36.000

$$\text{Lordo} = (\text{Ritenute} + \text{Netto})$$

VINCOLI INTERRELAZIONALI

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
7876463	Verdi	Luisa
7876463	Rossi	Maria

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
739430	24		04

CHIAVI: IDENTIFICARE LE n-UPLE

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing inf	5/12/78
78763	Rossi	Mario	Ing inf	3/11/76
65432	Neri	Piero	Ing mecc	10/7/79
87654	Neri	Mario	Ing inf	3/11/76
67653	Rossi	Piero	Ing mecc	5/12/78

IDENTIFICARE LE n-UPLE

- non ci sono due ennuple con lo stesso valore sull'attributo Matricola
- non ci sono due ennuple uguali su tutti e tre gli attributi Cognome, Nome e Data di Nascita

CHIAVE

INSIEME DI ATTRIBUTI CHE IDENTIFICANO LE n-UPLE DI UNA RELAZIONE

CHIAVE

FORMALMENTE

- Un insieme di K attributi è **superchiave** per r se non contiene due n-uple distinte t_1 e t_2 con $t_1^K = t_2^K$
- K è **chiave** per r se è una **superchiave minimale** per r
- **superchiave minimale** = non contiene un’altra superchiave

UNA CHIAVE

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing inf	5/12/78
78763	Rossi	Mario	Ing inf	3/11/76
65432	Neri	Piero	Ing mecc	10/7/79
87654	Neri	Mario	Ing inf	3/11/76
67653	Rossi	Piero	Ing mecc	5/12/78

Matricola è una chiave:

- è superchiave
- contiene un solo attributo r quindi è minimale

UN'ALTRA CHIAVE

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing inf	5/12/78
78763	Rossi	Mario	Ing inf	3/11/76
65432	Neri	Piero	Ing mecc	10/7/79
87654	Neri	Mario	Ing inf	3/11/76
67653	Rossi	Piero	Ing mecc	5/12/78

Cognome, nome, nascita è un'altra chiave:

- è superchiave
- è minimale

UN'ALTRA CHIAVE

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing inf	5/12/78
78763	Rossi	Mario	Ing Civile	3/11/76
65432	Neri	Piero	Ing mecc	10/7/79
87654	Neri	Mario	Ing inf	3/11/76
67653	Rossi	Piero	Ing mecc	5/12/78

Non ci sono n-uple uguali su Cognome e Corso:

- Cognome e Corso formano una chiave
- È sempre vero o è un caso?

CHIAVI

ESISTENZA

- Una relazione non può contenere n-uple distinte ma uguali
- Ogni relazione ha come superchiave l'insieme degli attributi su cui è definita
- quindi ha (almeno) una chiave

CHIAVI

IMPORTANZA

- l’esistenza delle chiavi garantisce l’accessibilità a ciascun dato della base di dati
- le chiavi permettono di correlare i dati in relazioni diverse
    - il modello relazionale è basato su valori

CHIAVI E VALORI NULL

- In presenza di valori nulli, i valori della chiave non permetteranno
    - di identificare le n-uple
    - di realizzare facilmente i riferimenti da altre relazioni
- la presenza di valori nulli nelle chiavi deve essere limitata

CHIAVI E VALORI NULL

Matricola	Cognome	Nome	Corso	Nascita
NULL	NULL	Mario	Ing inf	5/12/78
78763	Rossi	Mario	Ing inf	3/11/76
65432	Neri	Piero	Ing mecc	10/7/79
87654	Neri	Mario	Ing inf	NULL
NULL	Rossi	Piero	NULL	5/12/78

CHIAVE PRIMARIA

- Chiave su cui non sono ammessi valori NULL
- Notazione: sottolineatura

Matricola	**Cognome**	**Nome**	**Corso**	**Nascita**
27655	NULL	Mario	Ing inf	5/12/78
78763	Rossi	Mario	Ing inf	3/11/76
65432	Neri	Piero	Ing mecc	10/7/79
87654	Neri	Mario	Ing inf	NULL
67653	Rossi	Piero	NULL	5/12/78

INTEGRITÀ REFERENZIALE

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
787643	27	e lode	03
787643	24		04

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
787643	Neri	Piero
787642	Bianchi	Luca

- informazioni in relazioni diverse sono correlate attraverso valori comuni
- in particolare, valori delle chiavi (primarie)
- le correlazioni debbono essere "coerenti"

INTEGRITÀ REFERENZIALE

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
787643	27	e lode	03
787647	24		04

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
787643	Neri	Piero
787642	Bianchi	Luca

INTEGRITÀ REFERENZIALE

Infrazioni

Codice	Data	Vigile	Prov	Targa
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

INTEGRITÀ REFERENZIALE

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Targa
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Auto

<u>Prov</u>	<u>Targa</u>	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

VINCOLO DI INTEGRITÀ REFERENZIALE

Un vincolo di integrità referenziale (“foreing key”) fra attributi X di una relazione r_1 e un’altra relazione r_2 impone ai valori su X in r_1 di comparire come valori della chiave primaria di r_2

VINCOLO DI INTEGRITÀ REFERENZIALE

VINCOLI DI INTEGRITÀ REFERENZIALE FRA

- attributo `Vigile` della relazione `Infrazioni` e la relazione `Vigili`
- attributi `Prov` e Numero di `Infrazioni` e la relazione `Auto`

VINCOLI SU PIÙ ATTRIBUTI

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Targa
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Auto

<u>Prov</u>	<u>Targa</u>	Cognome	Nome
MI	E39548	Rossi	Mario
TO	F34268	Rossi	Mario
PR	839548	Neri	Luca

INTEGRITÀ REFERENZIALE E VALORI NULL

Impiegati

Matricola	Cognome	Progetto
34321	Rossi	IDEA
53524	Neri	XYZ
64521	Verdi	NULL
73321	Bianchi	IDEA

Progetti

Codice	Inizio	Durata	Costo
IDEA	01/2000	36	200
XYZ	07/2001	24	120
BOH	09/2001	24	150

AZIONI COMPENSATIVE

VIENE ELIMINATA UNA n-UPLA CAUSANDO UNA VIOLAZIONE

- Comportamento “standard”:
  - Rifiuto dell’operazione
- Azioni compensative
  - Eliminazione in cascata
  - Introduzione di valori nulli

ELIMINAZIONE IN CASCATA

Impiegati

<u>Matricola</u>	Cognome	Progetto
34321	Rossi	IDEA
~~53524~~	~~Neri~~	~~XYZ~~
64521	Verdi	NULL
73321	Bianchi	IDEA

Progetti

<u>Codice</u>	Inizio	Durata	Costo
IDEA	01/2000	36	200
~~XYZ~~	~~07/2001~~	~~24~~	~~120~~
BOH	09/2001	24	150

INTRODUZIONE DI VALORI NULL

Impiegati

Matricola	Cognome	Progetto
34321	Rossi	IDEA
53524	Neri	NULL
64521	Verdi	NULL
73321	Bianchi	IDEA

Progetti

Codice	Inizio	Durata	Costo
IDEA	01/2000	36	200
~~XYZ~~	~~07/2001~~	~~24~~	~~120~~
BOH	09/2001	24	150

VINCOLI MULTIPLI SU PIÙ ATTRIBUTI

Auto

<u>Prov</u>	<u>Targa</u>	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

Incidenti

<u>Codice</u>	Data	ProvA	TargaA	ProvB
34321	1/2/95	TO	E39548	MI
64521	5/4/96	PR	839548	TO

SQL

BENEFICI SQL

- Indipendenza dai venditori di HW e SW
- Portabilità attraverso varia piattaforme HW
- Coperto da standard internazionali SQL1, SQL2 e SQL3
- Strategico per IBM, Oracle, Microsoft, …
- Linguaggio per data base relazionali (unico)
- Strutturato ad alto livello (English-like)

BENEFICI SQL

- Linguaggio programmazione (Statico/Dinamico/API)
- In grado di fornire viste diverse del data base
- Linguaggio completo (IF, triggers, …) con T-SQL e PL-SQL
- Definizione dinamica dei dati
- Client/Server

SQL STANDARD?

IN REALTÀ OGNI MOTORE FA UN PO’ COME VUOLE

http://troels.arvin.dk/db/rdbms/

PORTABILITÀ: DAVVERO?

NON SI PUÒ FARE TUTTO

- codici di errore non standard
- tipi di dati non sempre supportati
- tabelle di sistema non sono uguali
- definisce solo linguaggio statico, non dinamico
- sorting

SQL BASICS

DATA DEFINITION LANGUAGE (DDL)

CREATE/DROP/ALTER TABLE/VIEW/INDEX

DATA MANIPULATION LANGUAGE (DML)

SELECT - INSERT - DELETE - UPDATE

SQL BASICS

DATA CONTROL LANGUAGE (DCL)

GRANT - REVOKE

TRANSACTION CONTROL LANGUAGE (TCL O T-SQL)

COMMIT - ROLLBACK

PROGRAMMING LANGUAGE (PL)

DECLARE - OPEN - FETCH - CLOSE

ELENCARE I DATABASE

```
SHOW DATABASES;
```

Ritorna l’elenco dei Database presenti nel DBMS

I comandi possono occupare anche più righe e terminano con il `;`

CREARE UN DATABASE

```
CREATE DATABASE nomeDataBase;
```

Crea un nuovo DataBase con il nome specificato e lo rende accessibile all'utente ROOT.

```
CREATE DATABASE IF NOT EXISTS nomeDataBase;
```

Crea il DB solo se non esiste già

ELIMINARE UN DATABASE

```
DROP DATABASE [IF EXISTS] nomeDataBase;
```

usiamo `[...]` per indicare le parti opzionali dei comandi.

SELEZIONARE UN DATABASE

```
USE nomeDataBase;
```

Tutti i comandi ora saranno riferiti a questo DB.

DEFINIZIONE DI DATI

Istruzione `CREATE TABLE`;

- definisce uno schema di relazione e ne crea un’istanza vuota
- specifica attributi, domini e vincoli

```
CREATE TABLE [IF NOT EXISTS] nomeTabella(
        nomeAttributo1 tipo,
        attributo2 tipo,
        ...
        attributoN tipo
    )
```

DOMINI - NUMERI INTERI

Tipo	Byte	Minimo	Massimo
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-2^{63}	$2^{63} - 1$

`INT(N)` : suggeriamo al motore di usare `N` caratteri per mostrare il dato; es: `INT(11)`

DOMINI - NUMERI RAZIONALI

VIRGOLA MOBILE

- float - 4 bytes
- double - 8 bytes

VIRGOLA FISSA

- `numeric(i,n)` salva esattamente `n` cifre decimali
- `decimal(i,n)` salva almeno `n` cifre decimali

DOMINI - TESTO

Tipo	Descrizione
CHAR	Stringa di lunghezza fissa non binaria
VARCHAR	Stringa di lunghezza variabile non binaria
BINARY	Sequenza binaria a lunghezza fissa
VARBINARY	Sequenza binaria a lunghezza variabile

SALVATI IN TABELLA

DOMINI - GENERICO

Tipo	Descrizione
TINYBLOB	Binary Large OBject piccolo
BLOB	Binary Large OBject
MEDIUMBLOB	Binary Large OBject medio
LONGBLOB	Binary Large OBject grande

SALVATAGGIO DEDICATO

DOMINI - TESTO

Tipo	Descrizione
TINYTEXT	Stringa non binaria piccola
TEXT	Stringa non binaria
MEDIUMTEXT	Stringa non binaria medio
LONGTEXT	Stringa non binaria grande

SALVATAGGIO DEDICATO

DOMINI - TEMPO

- `YEAR` - anno nel formato `YYYY`
- `DATE` - data nel formato `YYYY-MM-DD`
- `TIME` - tempo nel formato `hh:mm:ss`
- `DATETIME` - tempo nel formato `YYYY-MM-DD hh:mm:ss`
- `TIMESTAMP` - come `DATETIME`, ma si aggiorna da solo

DOMINI - SPAZIO

Tipo	Descrizione
GEOMETRY	Valore spaziale di qualsiasi tipo
POINT	Coordinate X, Y
LINESTRING	Curva (uno o più POINT)
POLYGON	Un poligono
	… e molti altri

DOMINI - STRINGHE

STO SALVANDO TESTO O SEQUENZE DI BYTE?

- testo: devo convertire la stringa in sequenza di byte (charset)
- sequenza e basta: posso salvarla così com’è

DOMINI - STRINGHE

DIMENSIONE FISSA O VARIABILE?

- fissa: devo indicare una dimensione (max 255)
- variabile: occupa `lunghezza + 1`; posso indicare una lunghezza massima

DOMINI - STRINGHE

valore	CHAR(4)	spazio	VARCHAR(4)	spazi
`''`	`'____'`	4 byte	`''`	1 byte
`'ab'`	`'ab  '`	4 byte	`'ab'`	3 byte
`'abcd'`	`'abcd'`	4 byte	`'abcd'`	5 byte
`'abcdef'`	`'abcd'`	4 byte	`'abcd'`	5 byte

DOMINI - STRINGHE

DOVE SALVO IL DATO?

- nella tabella: più rapido accedere al dato per interrogazioni
- storage dedicato: anche se ho molti dati la tabella resta piccola

STUDENTI ED ESAMI

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
7876463	Neri	Piero
7876462	Bianchi	Luca

Corsi

Codice	Titolo	Docente
01	Analisi	Mario
02	Chimica	Bruni
04	Chimica	Verdi

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
7876463	24		04

CREARE UNA TABELLA

Studenti

<u>Matricola</u>	Cognome	Nome
276545	Rossi	Mario
7876463	Neri	Piero
7876462	Bianchi	Luca

```
CREATE TABLE Studenti(
    matricola int(11),
    cognome varchar(45),
    nome varchar(45)
);
```

CANCELLARE UNA TABELLA

```
DROP TABLE [IF EXISTS] nomeTabella;
```

… intuitivo

```
DROP TABLE [IF EXISTS] nomeTabella1,
           nomeTabella2,
           nomeTabella3,
           ...;
```

VINCOLI

Posso definire dei vincoli:

- `PRIMARY KEY` - chiave primaria (una sola, implica `NOT NULL`)
- `NOT NULL`
- `UNIQUE` - definisce chiavi
- `CHECK` - vedremo più avanti

VINCOLI - CHIAVE PRIMARIA

Studenti

<u>Matricola</u>	Cognome	Nome
276545	Rossi	Mario
7876463	Neri	Piero
7876462	Bianchi	Luca

```sql
CREATE TABLE Studenti(
    matricola int(11) PRIMARY KEY,
    cognome varchar(45),
    nome varchar(45)
);
```

VINCOLI - CHIAVE PRIMARIA

Studenti

<u>Matricola</u>	Cognome	Nome
276545	Rossi	Mario
7876463	Neri	Piero
7876462	Bianchi	Luca

```
CREATE TABLE Studenti(
    matricola int(11),
    cognome varchar(45),
    nome varchar(45),
    PRIMARY KEY (matricola)
);
```

VINCOLI - PROIBIRE I NULL

Studenti

<u>Matricola</u>	Cognome	Nome
276545	Rossi	Mario
7876463	Neri	Piero
7876462	Bianchi	Luca

```
CREATE TABLE Studenti(
    matricola int(11) PRIMARY KEY,
    cognome varchar(45) NOT NULL,
    nome varchar(45) NOT NULL
);
```

VINCOLI - PROIBIRE I NULL

Corsi

<u>Codice</u>	Titolo	Docente
01	Analisi	Mario
02	Chimica	Bruni
04	Chimica	Verdi

```
CREATE TABLE Corsi(
    codice int(11) PRIMARY KEY,
    titolo varchar(45) NOT NULL,
    docente varchar(45)
);
```

VINCOLI - CHIAVI COMPOSTE

Esami

<u>Studente</u>	Voto	Lode	<u>Corso</u>
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
7876463	24		04

```sql
CREATE TABLE Esami(
    studente int(11) PRIMARY KEY,
    voto smallint NOT NULL,
    lode bool,
    corso int(11) PRIMARY KEY
);
```

VINCOLI - CHIAVI COMPOSTE

Esami

<u>Studente</u>	Voto	Lode	<u>Corso</u>
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
7876463	24		04

```
CREATE TABLE Esami(
    studente int(11),
    voto smallint NOT NULL,
    lode bool,
    corso int(11),
    PRIMARY KEY (studente, corso)
);
```

NOT NULL + UNIQUE = PRIMARY KEY

```
CREATE TABLE Corsi(
        codice int(11) NOT NULL UNIQUE,
        titolo varchar(45) NOT NULL,
        docente varchar(45)

    );

    CREATE TABLE Esami(
        studente int(11) NOT NULL UNIQUE,
        voto smallint NOT NULL,
        lode bool,
        corso int(11) NOT NULL UNIQUE
    );
```

UNIQUE E NULL MULTIPLI

In general, a unique constraint is violated when there is more than one row in the table where the values of all of the columns included in the constraint are equal. However, two null values are not considered equal in this comparison. That means ***even in the presence of a unique constraint it is possible to store duplicate rows that contain a null value*** *in at least one of the constrained columns. This behavior conforms to the SQL standard, but we have heard that other SQL databases might not follow this rule. So be careful when developing applications that are intended to be portable.*

PostgreSQL Manual

UNIQUE SU PIÙ COLONNE

```
CREATE TABLE nomeTabella (
        id int(11) PRIMARY KEY,
        campo1 int(19),
        campo2 int(12),
        CONSTRAINT [nome] UNIQUE(campo1, campo2)
    );
```

AUTO INCREMENT

- il motore si occupa di incrementare il contatore numerico
- identifico in modo chiaro una n-upla
- ottimo come chiave primaria!

```
CREATE TABLE Studenti(
        matricola int(11) PRIMARY KEY AUTO_INCREMENT,
        cognome varchar(45),
        nome varchar(45)
    );
```

CHECK

SERVE PER SPECIFICARE VINCOLI COMPLESSI

NETTO = LORDO - TRATTENUTE

- non supportato in MySql
- MySql IGNORA il comando dato alla creazione della tabella

AUTO_INCREMENT: DETTAGLI

COSA SUCCEDE SE CANCELLO UNA RIGA?

- non riciclo!
- successivo = inserito automaticamente +1

AUTO_INCREMENT: DETTAGLI

COSA SUCCEDE SE IMPOSTO UN VALORE?

- se non è duplicato viene accettato
- successivo = inserito manualmente +1

AUTO_INCREMENT: DETTAGLI

COSA SUCCEDE SE MODIFICO UN VALORE?

- se non è duplicato viene accettato
- successivo = inserito automaticamente +1

VALORI PREDEFINITI

```
CREATE TABLE nome(
        nomeAttributo tipo DEFAULT valore
    );
```

VALORI PREDEFINITI

```
CREATE TABLE Corsi(
        codice int(11) PRIMARY KEY,
        titolo varchar(45) NOT NULL DEFAULT "nuovo",
        docente varchar(45)
    );
```

COMMENTI

```
CREATE TABLE nome(
        nomeAttributo tipo COMMENT "commento"
    );
```

COMMENTI

```sql
CREATE TABLE Corsi(
        codice int(11) PRIMARY KEY,
        titolo varchar(45) NOT NULL
        COMMENT "Titolo del corso",
        docente varchar(45)
    );
```

CHIAVI PRIMARIE E NULL

Software

Modulo	**Versione**	**Tipo**	**Data**
Esse3	1.00	alfa	10/10/2014
Esse3	1.00	beta	10/10/2014
Esse3	1.00	NULL	16/11/2014
Esse3	1.02	alfa	18/12/2014
Esse4	1.00	alfa	12/01/2015

Modulo, Versione e Tipo sono una PK?

- identificano n-upla
- non contengono altre superchiavi

CHIAVI PRIMARIE E NULL

Software

<u>Modulo</u>	<u>Versione</u>	<u>Tipo</u>	Data
Esse3	1.00	alfa	10/10/2014
Esse3	1.00	beta	14/11/2014
Esse3	1.00	NULL	16/11/2014
Esse3	1.02	alfa	18/12/2014
Esse4	1.00	alfa	12/01/2015

- NULL = assenza di informazioni
- '' = informazione nota, pari a VUOTO
- NULL è uguale a NULL?

CHIAVI PRIMARIE E NULL

Software

Modulo	Versione	Tipo	Data
Esse3	1.00	alfa	10/10/2014
Esse3	1.00	beta	14/11/2014
Esse3	1.00	''	16/11/2014
Esse3	1.02	alfa	18/12/2014
Esse4	1.00	alfa	12/01/2015

VINCOLI INTERRELAZIONALI

- `FOREIGN KEY` e `REFERENCES` e permettono di definire vincoli di integrità referenziale
- di nuovo due sintassi
  - per singoli attributi (non in MySql)
  - su più attributi
- è possibile definire azioni compensative

FOREIGN KEY

colonne che sono FK

REFERENCES

colonne nella relazione (tabella) esterna

STUDENTI ED ESAMI

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
7876463	Neri	Piero
7876462	Bianchi	Luca

Corsi

Codice	Titolo	Docente
01	Analisi	Mario
02	Chimica	Bruni
04	Chimica	Verdi

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
7876463	27	e lode	03
7876463	24		04

VINCOLI INTERRELAZIONALI

```
CREATE TABLE Esami(
        studente int(11),
        voto smallint NOT NULL,
        lode bool,
        corso int(11),
        PRIMARY KEY (studente, corso),
        FOREIGN KEY (studente) REFERENCES Studenti(matricola)
    );
```

DEFINIZIONE COMPATTA

Non funziona ovunque (MySql)

```
CREATE TABLE Esami(
        studente int(11) REFERENCES Studenti(matricola),
        voto smallint NOT NULL,
        lode bool,
        corso int(11),
        PRIMARY KEY (studente, corso)
    );
```

VINCOLI INTERRELAZIONALI

```
CREATE TABLE Esami(
        studente int(11),
        voto smallint NOT NULL,
        lode bool,
        corso int(11),
        PRIMARY KEY (studente, corso),
        FOREIGN KEY (studente) REFERENCES Studenti(matricola)
        FOREIGN KEY (corso) REFERENCES Corsi(codice)
    );
```

VINCOLI INTERRELAZIONALI CON NOME

```
CREATE TABLE Esami(
        studente int(11),
        voto smallint NOT NULL,
        lode bool,
        corso int(11),
        PRIMARY KEY (studente, corso),
        CONSTRAINT FK_Studente FOREIGN KEY (studente) REFERENC
        CONSTRAINT FK_Corso FOREIGN KEY (corso) REFERENCES Cor
    );
```

DISATTIVARE I VINCOLI INTERRELAZIONALI

- sto caricando i dati: ordine importante, altrimenti errore
- voglio disattivare temporaneamente i vincoli
- disattivazione
  - `SET foreign_key_checks = 0`
- riattivazione
  - `SET foreign_key_checks = 1`

INTEGRITÀ REFERENZIALE

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Targa
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Automobile

<u>Prov</u>	<u>Targa</u>	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

Vigili

<u>Matricola</u>	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

INTEGRITÀ REFERENZIALE

Vigili

<u>Matricola</u>	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

```
CREATE TABLE Vigili(
    matricola int(11) PRIMARY KEY AUTO_INCREMENT,
    cognome varchar(45) NOT NULL,
    nome varchar(45) NOT NULL
);
```

INTEGRITÀ REFERENZIALE

Automobile

<u>Prov</u>	<u>Targa</u>	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

```
CREATE TABLE Automobile(
    prov char(2),
    targa char(6),
    cognone varchar(45) NOT NULL,
    nome varchar(45) NOT NULL,
    PRIMARY KEY (prov, targa)
);
```

INTEGRITÀ REFERENZIALE

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Targa
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

```
CREATE TABLE Infrazioni(
    codice int(11) PRIMARY KEY AUTO_INCREMENT,
    data datetime,
    vigile int(11),
    prov char(2),
    targa char(6),
    FOREIGN KEY (vigile) REFERENCES Vigili(matricola),
    FOREIGN KEY (prov, targa)
    REFERENCES Automobile(prov, targa)
)
```

CANCELLAZIONE

```
CREATE TABLE Infrazioni(
        codice int(11) PRIMARY KEY AUTO_INCREMENT,
        data datetime NOT NULL,
        vigile int(11),
        prov char(2) NOT NULL,
        targa char(6) NOT NULL,
        FOREIGN KEY (vigile) REFERENCES Vigili(matricola)
        ON DELETE SET NULL ON UPDATE CASCADE,
        FOREIGN KEY (prov, targa) REFERENCES
        Automobile(prov, targa)
    );
```

CAMBIARE LO SCHEMA

Infrazioni

Codice	Data	dataModifica	Vigile	Prov	Targa
34321	1/2/95	1/2/95	3987	MI	39548K
53524	4/3/95	16/4/91	3295	TO	E39548
64521	5/4/96	11/2/84	3295	PR	839548
73321	5/2/98	31/12/98	9345	PR	839548

CAMBIARE LO SCHEMA

- `DROP` tabella
- ricreare tabella
- … perdo tutti i dati

MODIFICARE TABELLE

```
ALTER TABLE nomeTabella
    azione1
    [, azione2, ...]
```

- aggiungere/togliere colonne
- cambiare il tipo di dato
- rinominare la tabella
- definire chiavi primarie, esterne, ecc..

AGGIUNGERE COLONNE

```
ALTER TABLE nomeTabella
 ADD COLUMN definizioneColonna
 [ FIRST | AFTER nomeColonna ]
```

AGGIUNGERE COLONNE

```
ALTER TABLE Infrazioni
    ADD COLUMN dataModifica TIMESTAMP
    AFTER data
```

RIMUOVERE COLONNE

```
ALTER TABLE nomeTabella
    DROP COLUMN nomeColonna
```

RIMUOVERE COLONNE

```
ALTER TABLE infrazioni
    DROP COLUMN dataModifica
```

MODIFICARE COLONNE

```
ALTER TABLE nomeTabella
    CHANGE COLUMN nomeOriginale
    nomeNuovo tipo
```

MODIFICARE COLONNE

```
ALTER TABLE infrazioni
     CHANGE COLUMN dataModifica
     dataUltimaModifica TIMESTAMP
```

MODIFICARE COLONNE

```
ALTER TABLE automobili
    CHANGE COLUMN cognome
    VARCHAR(100)
```

RINOMINARE TABELLE

```
ALTER TABLE nomeTabella
     RENAME TO nuovoNome
```

RINOMINARE TABELLE

```
ALTER TABLE infrazioni
    RENAME TO odiateInfrazioni
```

AGGIUNGERE/RIMUOVERE FOREIGN KEY

```
ALTER TABLE nomeTabella
     ADD CONSTRAINT nome
     FOREIGN KEY (...)
     REFERENCES tabella(...)
```

```
ALTER TABLE nomeTabella
     DROP FOREIGN KEY nome
```

DATABASE DI PROVA

CLASSICMODELS

VENDITA MODELLINI

https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database-aspx/

DATABASE DI PROVA

CLASSICMODELS

- clienti
- dipendenti
- uffici di vendita
- prodotti
- ordini
- pagamenti

DATABASE DI PROVA

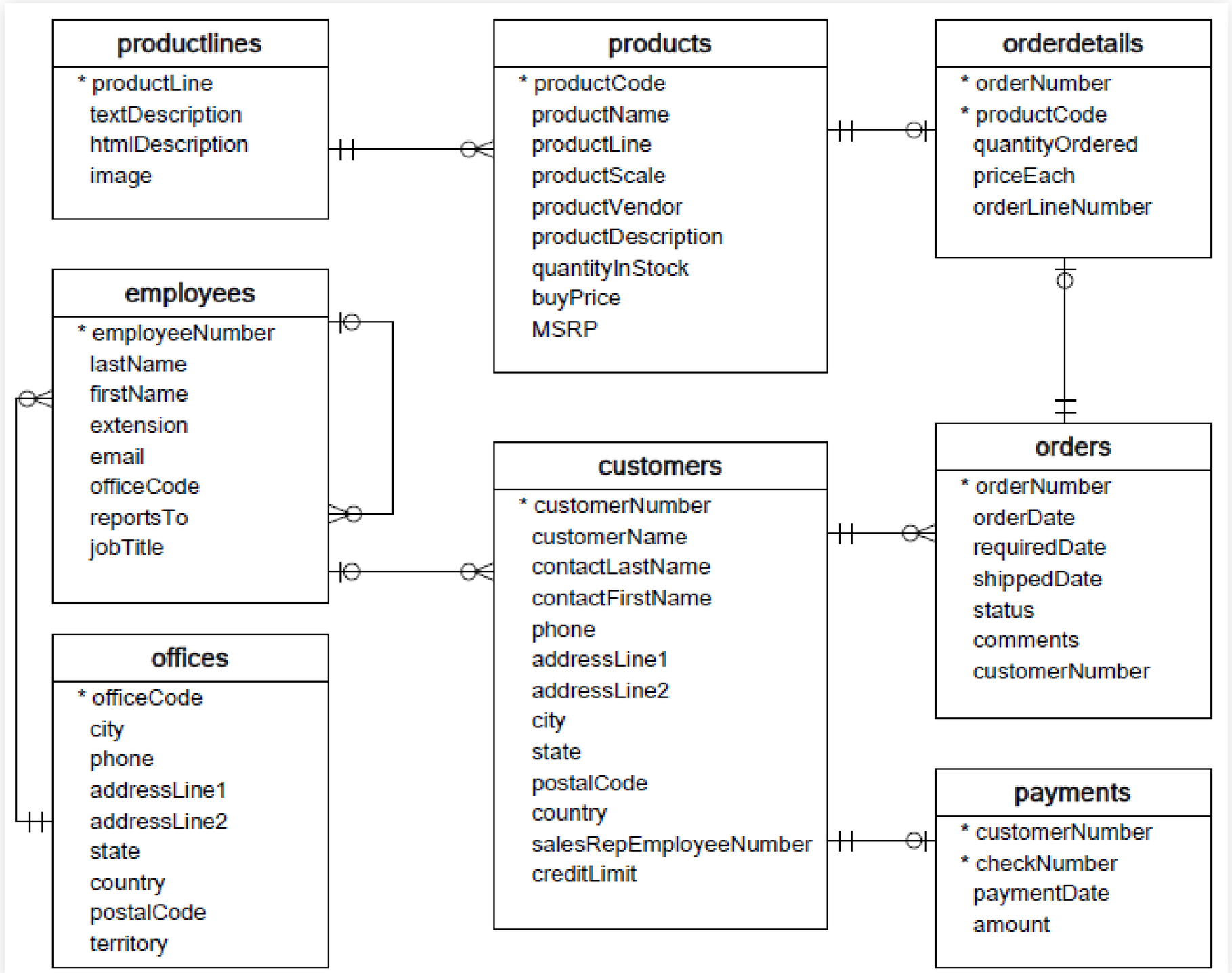
CLASSICMODELS

- **customers**: dati dei clienti
- **products**: modellini disponibili
- **productlines**: linee di prodotti
    - auto classiche, moto, navi, treni
- **orders**: ordini fatti dai clienti

DATABASE DI PROVA

CLASSICMODELS

- **orderdetails**: dettagli di ogni ordine dei clienti
    - linee dell’ordine
- **payments**: pagamenti fatti dai clienti
- **employees**: informazioni sui dipendenti
    - chi è il capo di chi, telefoni, ecc
- **offices**: uffici di vendita


productlines
* productLine
textDescription
htmlDescription
image
products
* productCode
productName
productLine
productScale
productVendor
productDescription
quantityInStock
buyPrice
MSRP
orderdetails
* orderNumber
* productCode
quantityOrdered
priceEach
orderLineNumber
employees
* employeeNumber
lastName
firstName
extension
email
officeCode
reportsTo
jobTitle
customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit
orders
* orderNumber
orderDate
requiredDate
shippedDate
status
comments
customerNumber
offices
* officeCode
city
phone
addressLine1
addressLine2
state
country
postalCode
territory
payments
* customerNumber
* checkNumber
paymentDate
amount

ELENCARE ELEMENTI DELLA TABELLA

ELENCARE TUTTI I MODELLINI IN VENDITA

```
SELECT * FROM products;
```

ISTRUZIONE SELECT (BASE)

```
SELECT attributo1 [, attributo2, ...]
 FROM tabella1 [, tabella2, ...]
 [WHERE condizione]
```

- `*` = tutti gli attributi
- `FROM` = da dove (per ora)
- `WHERE` = quali ennuple

TUTTO È UNA TABELLA

- selezione: `WHERE`
  - selezione una “sottotabella”
- proiezione: elenco attributi
  - mostro una tabella scegliendo quali colonne mostrare

SELEZIONE

MOSTRA I MODELLINI CHE COSTANO MENO DI 75$

```
SELECT * FROM  products WHERE MSRP < 75;
```

SELEZIONE E PROIEZIONE

MOSTRA NOME, PREZZO DI ACQUISTO E DI VENDITA DEI MODELLINI CHE COSTANO MENO DI 75$

```
SELECT productName, buyPrice, MSRP
 FROM products
 WHERE MSRP < 75;
```

RINOMINARE ATTRIBUTI

ISTRUZIONE AS

```
SELECT productName AS nomeProdotto,
 productVendor AS nomeVenditore
 FROM products;
```

SELECT, ABBREVIAZIONI

```
SELECT productName, buyPrice, MSRP
 FROM products
 WHERE MSRP < 75;
```

in realtà stiamo scrivendo

```
SELECT p.productName, p.buyPrice, p.MSRP
 FROM products p
 WHERE p.MSRP < 75;
```

SELECT, ABBREVIAZIONI

```sql
SELECT * FROM products;
```

in realtà stiamo scrivendo

```sql
SELECT productCode, productName, productLine,
 productScale, productVendor,
 productDescription, quantityInStock,
 buyPrice, MSRP
 FROM products;
```

SELECT, ABBREVIAZIONI

```sql
SELECT * FROM products;
```

in realtà stiamo scrivendo

```sql
SELECT productCode, productName, productLine,
 productScale, productVendor,
 productDescription, quantityInStock,
 buyPrice, MSRP
 FROM products
 WHERE true;
```

CONDIZIONI: TESTO ESATTO

MOSTRA TUTTI I DIPENDENTI DI NOME LESLIE

```
SELECT * FROM employees
 WHERE firstName = 'Leslie';
```

VIRGOLETTE SINGOLE O DOPPIE?

- `"Leslie"` o `'Leslie'`?
  - indifferente!
- standard ANSI: `'Leslie'`

VIRGOLETTE SINGOLE O DOPPIE?

PER INSERIRE ‘ IN UNA STRINGA? ES: `Ci'ao`

- delimitata da `" "`: `"Ci'ao"`
- delimitata da `' '`: raddoppio → `'Ci''ao'`

PER INSERIRE ‘ IN UNA STRINGA? ES: `Ci"'"ao`

- delimitata da `" "`: raddoppio → `"Ci""ao"`
- delimitata da `' '`: `'Ci"ao'`

CONDIZIONI: TESTO INCOMPLETO

MOSTRA TUTTI I DIPENDENTI IL CUI COGNOME FINISCE PER “SON”.

```
SELECT * FROM employees
 WHERE lastName LIKE '%son';
```

CONDIZIONI: TESTO INCOMPLETO

MOSTRA TUTTI I DIPENDENTI IL CUI COGNOME *NON* FINISCE PER “SON”.

```
SELECT * FROM employees
 WHERE lastName NOT LIKE '%son';
```

CONDIZIONI: TESTO INCOMPLETO

- % = zero o più caratteri
- _ = esattamente un carattere
- per cercare il carattere % uso \%
- per cercare il carattere _ uso _

CONDIZIONI: TESTO INCOMPLETO

MOSTRA TUTTI I DIPENDENTI IL CUI NOME FINISCE PER "ARRY" E DAVANTI HA UNA SOLA LETTERA

```
SELECT * FROM employees
 WHERE firstName LIKE '_arry';
```

CONDIZIONI: TESTO INCOMPLETO

MOSTRA TUTTI I PRODOTTI CHE HANNO UNA SCALA DIVISIBILE PER 10 E MINORE DI 100 (ES: 1:10, 1:20, 1:30, ...)

```sql
SELECT * FROM products
 WHERE productScale LIKE '1:_0';
```

CONDIZIONI: TESTO INCOMPLETO

MOSTRA TUTTI I DIPENDENTI IL CUI NOME INIZIA CON M E LA CUI TERZA LETTERA È UNA R

```
SELECT * FROM employees
 WHERE firstName LIKE 'M_r%';
```

PIÙ CONDIZIONI

MOSTRA I MODELLINI CHE COSTANO MENO DI 75 E CHE ABBIAMO COMPRATO A PIÙ DI 30

```
SELECT productName, MSRP, buyPrice
 FROM products
 WHERE MSRP < 75 AND buyPrice > 30;
```

PIÙ CONDIZIONI

MOSTRA I MODELLINI CHE COSTANO MENO DI 75 O PIÙ DI 150

```
SELECT productName, MSRP
 FROM products
 WHERE MSRP < 75 OR MSRP > 150;
```

PIÙ CONDIZIONI

MOSTRA I MODELLINI CHE COSTANO MENO DI 75 O PIÙ DI 150 E CHE COMUNQUE ABBIAMO COMPRATO A PIÙ DI 30

```
SELECT productName, MSRP, buyPrice
 FROM products
 WHERE (MSRP<75 OR MSRP>150) AND buyPrice>30;
```

INTERVALLI

SELEZIONA I VALORI COMPRESI TRA X E Y (INCLUSI)

```
SELECT ... FROM ...
 WHERE colonna BETWEEN x AND y;
```

INTERVALLI

MOSTRA I PAGAMENTI CON IMPORTI COMPRESI TRA 5.000 ED 8.000

```sql
SELECT * FROM payments
 WHERE amount BETWEEN 5000 AND 8000;
```

INTERVALLI

MOSTRA I PAGAMENTI CON IMPORTI COMPRESI TRA 5.000 ED 8.000

```sql
SELECT * FROM payments
 WHERE amount BETWEEN 5000 AND 8000;
```

forma equivalente

```sql
SELECT * FROM payments
 WHERE amount >= 5000 AND
 amount <= 8000;
```

INTERVALLI

PRENDI TUTTI I DIPENDENTI IL CUI NOME INIZIA CON UNA LETTERA TRA B ED F

```sql
SELECT * FROM employees
 WHERE firstName BETWEEN 'B' AND 'F';
```

LISTE

CONTROLLA SE IL VALORE È PRESENTE IN UNA LISTA DI VALORI

```
SELECT ... FROM ...
 WHERE colonna IN (val1, val2, ...)
```

LISTE

MOSTRA CODICE UFFICIO, CITTÀ E NUMERO DI TELEFONO DEGLI UFFICI IN FRANCIA O AMERICA

```
SELECT officeCode, city, phone
 FROM offices
 WHERE country IN ('USA', 'FRANCE');
```

LISTE

```sql
SELECT officeCode, city, phone
 FROM offices
 WHERE country IN ('USA', 'FRANCE');
```

forma equivalente

```sql
SELECT officeCode, city, phone
 FROM offices
 WHERE country = 'USA' OR
 country = 'FRANCE';
```

LISTE

MOSTRARE I MODELLINI DEL TIPO "PLANES", "SHIPS" O "CLASSIC CARS"

```
SELECT * FROM products
 WHERE productLine
 IN ('Planes','Ships','Classic Cars');
```

GESTIRE I NULL

```
SELECT ... FROM ...
 WHERE colonna = "";
```

NO

GESTIRE I NULL

```
SELECT ... FROM ...
 WHERE colonna IS NULL;
```

GESTIRE I NULL

MOSTRA GLI ORDINI NON SPEDITI

```
SELECT * FROM orders
 WHERE shippedDate IS NULL;
```

GESTIRE I NULL

MOSTRA GLI ORDINI CREATI DOPO IL 30/04/2005 E NON SPEDITI

```sql
SELECT * FROM orders WHERE
 orderDate > '2005-04-30' AND
 shippedDate IS NULL;
```

ESPRESSIONI

MOSTRA I PREZZI DI VENDITA SENZA L'IVA (PREZZO / 1.22)

```
SELECT productName, MSRP/1.22 AS noIVA
 FROM products;
```

ESPRESSIONI

MOSTRA I PRODOTTI CON UN MARGINE (PREZZO - PREZZO ACQUISTO) SUPERIORE A 50

```
SELECT productName, MSRP, buyPrice
 FROM products
 WHERE MSRP-buyPrice > 50;
```

FUNZIONI

STRINGHE

- `length()`
- `reverse()`
- `right()`
- `trim()`
- `...`

FUNZIONI

MOSTRA I PRODOTTI CON NOMI DI ALMENO 15 CARATTERI.

```
SELECT productName, length(productName)
 FROM products
 WHERE length(productName) >= 15;
```

FUNZIONI

DATA E ORA

- `day()`
- `year()`
- `now()`
- `month()`
- `monthname()`
- `...`

FUNZIONI

MOSTRA I PRODOTTI ORINATI NEL MESE DI GENNAIO

```
SELECT * from orders
 WHERE month(orderDate) = 1;
```

ORDINAMENTO

STABILIRE L'ORDINE DI PRESENTAZIONE DEI RISULTATI

```
SELECT ... FROM ... WHERE ...
 ORDER BY col1 [ASC|DESC], col2 [ASC|DESC], ...
```

- `ASC`: crescente → DEFAULT!
- `DESC`: decrescente

ORDINAMENTO

MOSTRA I MODELLINI ORDINANDOLI PER PREZZO DI VENDITA CRESCENTE

```
SELECT productName, MSRP
 FROM products
 ORDER BY MSRP;
```

ORDINAMENTO

MOSTRA I CLIENTI ORDINANDOLI PER PAESE CRESCENTE E CREDITO MASSIMO DECRESCENTE

```
SELECT customerName, country, creditLimit
 FROM customers
 ORDER BY country, creditLimit DESC;
```

ORDINAMENTO

MOSTRA GLI ORDINI ORDINANDOLI IN BASE ALLO STATUS IN CUI SI TROVANO (IN CORSO, IN ATTESA, CANCELLATI, ECC).

```
SELECT * FROM orders
 order by status;
```

NO

ORDINAMENTO

MOSTRA GLI ORDINI ORDINANDOLI IN BASE ALLO STATUS IN CUI SI TROVANO (IN CORSO, IN ATTESA, CANCELLATI, ECC)

```
FIELD(text, str1, str2, str3, ...)
```

Ritorna la posizione della stringa `text` nella lista `str1, str2, str3, ...`

ORDINAMENTO

MOSTRA GLI ORDINI ORDINANDOLI IN BASE ALLO STATUS IN CUI SI TROVANO (IN CORSO, IN ATTESA, CANCELLATI, ECC).

```
SELECT * FROM orders
 ORDER BY FIELD(status, 'In Process',
 'On Hold', 'Cancelled', 'Resolved',
 'Disputed','Shipped');
```

UNIAMO IL TUTTO...

MOSTRA I PRODOTTI VENDUTI A MENO DI 100€, METTENDO IN CIMA QUELLI CON IL MARGINE PIÙ ALTO

```
SELECT productName,
 MSRP-buyPrice as margine
 FROM products
 WHERE MSRP < 100
 ORDER BY msrp-buyPrice DESC
```

RIGHE DUPLICATE

OGNI TANTO LE NOSTRE QUERY RITORNANO RIGHE DUPLICATE: COME FACCIAMO AD ELIMINARE I DOPPIONI?

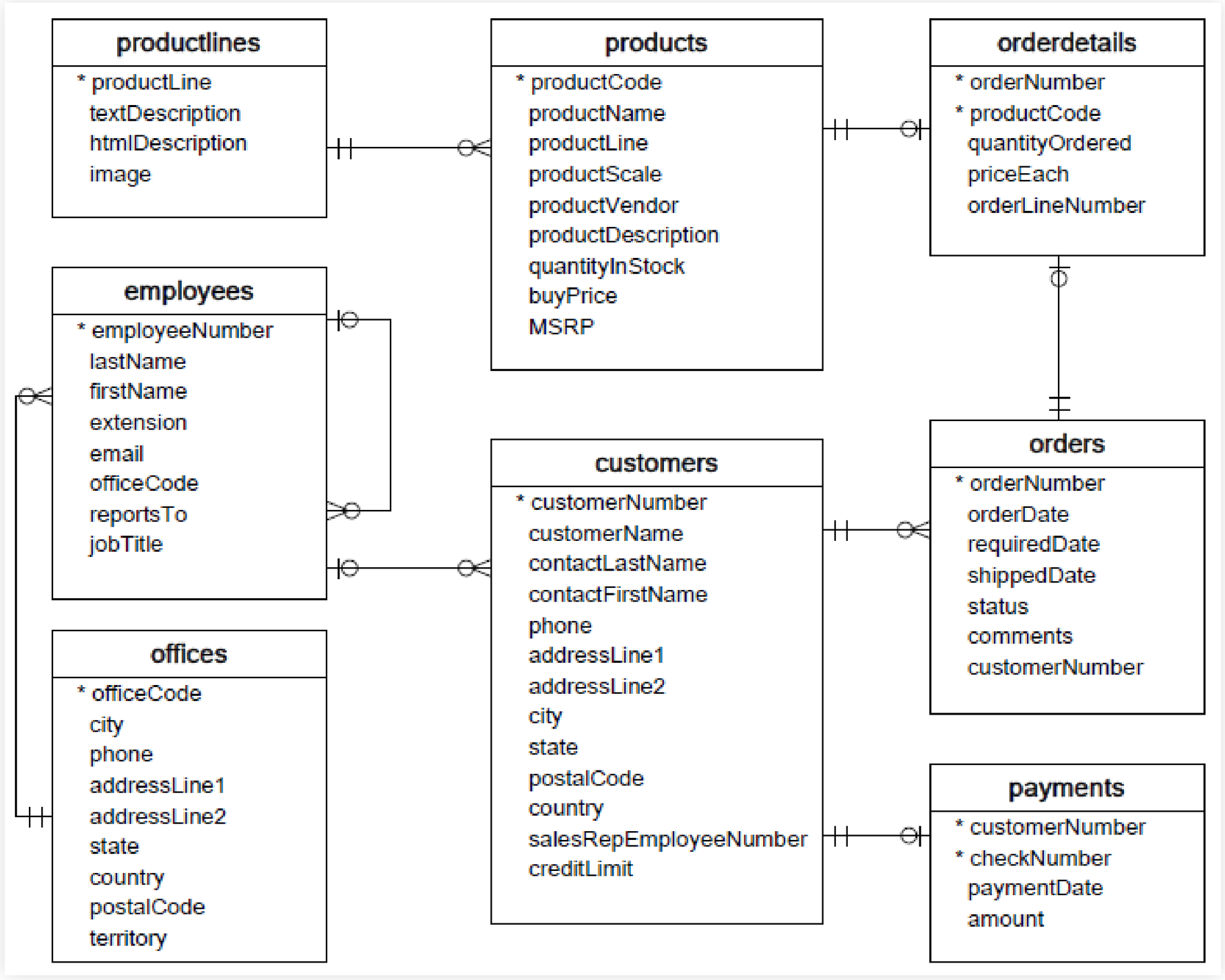
```
SELECT DISTINCT ... FROM ...
```

RIGHE DUPLICATE

MOSTRA TUTTE LE CITTÀ IN CUI SI TROVANO I MIEI CLIENTI, ORDINANDOLE ALFABETICAMENTE

```
SELECT DISTINCT city FROM customers
 order by city;
```

DECODIFICARE LE RELAZIONI



PRODOTTO CARTESIANO

CREO IL PRODOTTO CARTESIANO DI PIÙ TABELLE

```
SELECT ... FROM tabella1, tabella2, ...
```

Risultato: una riga per ogni combinazione di valori tra le righe di tabella1 e di tabella2

PRODOTTO CARTESIANO

CREA IL PRODOTTO CARTESIANO TRA LA TABELLA DEGLI

impiegati e quella dei clienti

```
SELECT * FROM customers, employees;
```

CROSS JOIN

CREA IL PRODOTTO CARTESIANO TRA LA TABELLA DEGLI

impiegati e quella dei clienti

```
SELECT * FROM customers, employees;
```

Forma esplicita:

```
SELECT * FROM customers CROSS JOIN employees;
```

PRODOTTO CARTESIANO FILTRATO

NON VOGLIO VEDERE TUTTE LE COMBINAZIONI, SOLO FILTRARE TABELLA!

```
SELECT ... FROM tabella1, tabella2, ...
 WHERE condizione sui valori comuni (PK e FK)
```

Creo una riga per ogni combinazione di valori tra le righe della tabella1 e della tabella2, ma poi salvo solo quelle sensate

PRODOTTO CARTESIANO FILTRATO

MOSTRA PER OGNI CLIENTE IL NOME DEL VENDITORE ASSOCIATO

```
SELECT customerName, salesRepEmployeeNumber,
 lastName, employeeNumber
 FROM customers, employees
 WHERE salesRepEmployeeNumber = employeeNumber
```

INNER JOIN

MODO MIGLIORE PER SCRIVERE IL TUTTO

```
SELECT ... FROM tabella1
 INNER JOIN tabella2
 ON PK = FK
```

INNER JOIN

MOSTRA PER OGNI CLIENTE IL NOME DEL VENDITORE ASSOCIATO

```
SELECT customerName, salesRepEmployeeNumber,
 lastName, employeeNumber
FROM customers
INNER JOIN employees
ON salesRepEmployeeNumber = employeeNumber;
```

INNER JOIN: AMBIGUITÀ

OGNI TANTO PK E FK HANNO STESSO NOME

```
SELECT ... FROM tabella1
 INNER JOIN tabella2
 ON tabella2.PK = tabella1.FK
```

INNER JOIN: AMBIGUITÀ

MOSTRA PER OGNI PRODOTTO LA DESCRIZIONE DELLA LINEA DI PRODOTTI CUI APPARTIENE

```
SELECT productCode, productName, textDescription
 FROM products INNER JOIN productlines
 ON products.productline =
 productlines.productline;
```

INNER JOIN: AMBIGUITÀ

```
SELECT productCode, productName, textDescription
 FROM products INNER JOIN productlines
 ON products.productline = productlines.productline;
```

INNER JOIN: AMBIGUITÀ

FORMA EQUIVALENTE

```
SELECT productCode, productName, textDescription
 FROM products p1
INNER JOIN productlines p2
ON p1.productline = p2.productline;
```

INNER JOIN

MOSTRA PER OGNI PRODOTTO LA DESCRIZIONE DELLA LINEA DI PRODOTTI CUI APPARTIENE

```
SELECT productCode, productName, textDescription
 FROM products INNER JOIN productlines
 ON products.productline =
 productlines.productline;
```

CRITERI DI JOIN

SE GLI ATTRIBUTI HANNO LO STESSO NOME TRA LE RELAZIONI, SI PUÒ USARE UNA FORMA ABBREVIATA

```
SELECT productCode, productName,
 textDescription
 FROM products INNER JOIN productlines
 USING(productline);
```

CRITERI DI JOIN

MOSTRA TUTTI GLI IMPIEGATI E LA CITTÀ IN CUI SI TROVA L'UFFICIO CUI AFFERISCONO

```
SELECT firstName, lastName, city
 FROM employees INNER JOIN offices
 USING (officeCode);
```

CRITERI DI JOIN

Se gli unici nomi di attributi in comune tra due tabelle sono quelli di FK/PK, si può usare una forma ANCORA più abbreviata

```
SELECT ... FROM tabella1
 NATURAL JOIN tabella2
```

PERICOLOSE!

Cosa succede se aggiungo/cambio colonne?

CRITERI DI JOIN

MOSTRA PER OGNI PRODOTTO LA DESCRIZIONE DELLA LINEA DI PRODOTTI CUI APPARTIENE

```
SELECT productCode, productName,
 textDescription
 FROM products NATURAL JOIN productlines;
```

PROBLEMA

- Voglio vedere i clienti ✓
- Voglio vedere il nome dei venditori assegnati ✓
- Voglio vedere anche i clienti senza venditore assegnato (?)

LEFT OUTER JOIN

MOSTRA TUTTI I DATI DELLA PRIMA TABELLA, E SE POSSIBILE ASSOCIA LE INFORMAZIONI DELLA SECONDA

```
SELECT ... FROM tabella1
 LEFT OUTER JOIN tabella2
 ON PK = FK
```

LEFT OUTER JOIN

MOSTRA TUTTI I CLIENTI; SE IL CLIENTE HA UN VENDITORE ASSOCIATO, MOSTRANE I DATI

```
SELECT customerName, concat(firstName,' ',lastName)
 FROM customers LEFT OUTER JOIN employees
 ON salesRepEmployeeNumber = employeeNumber
```

LEFT OUTER JOIN

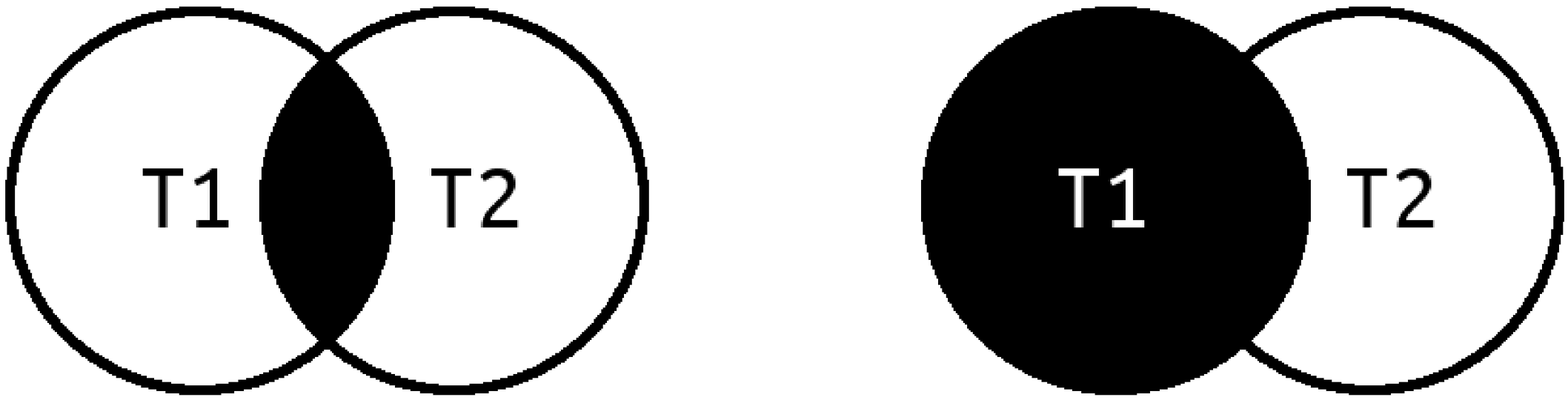
MOSTRA TUTTI I CLIENTI; SE IL CLIENTE HA UN VENDITORE ASSOCIATO, MOSTRANE I DATI

```
SELECT customerName, concat(firstName,' ',lastName)
 FROM customers LEFT OUTER JOIN employees
 ON salesRepEmployeeNumber = employeeNumber
```

forma equivalente

```
SELECT customerName, concat(firstName,' ',lastName)
 FROM customers LEFT JOIN employees
 ON salesRepEmployeeNumber = employeeNumber
```

INNER VS LEFT OUTER JOIN



Inner

Left Outer

LEFT OUTER JOIN

MOSTRA TUTTI I CLIENTI ED I RELATIVI ORDINI, INCLUSI I CLIENTI CHE NON HANNO FATTO ORDINI

```
SELECT c.customerNumber, c.customerName,
 o.orderNumber, o.status
 FROM customers c LEFT JOIN orders o
 ON c.customerNumber = o.customerNumber;
```

LEFT OUTER JOIN

MOSTRA TUTTI I CLIENTI CHE NON HANNO ORDINI

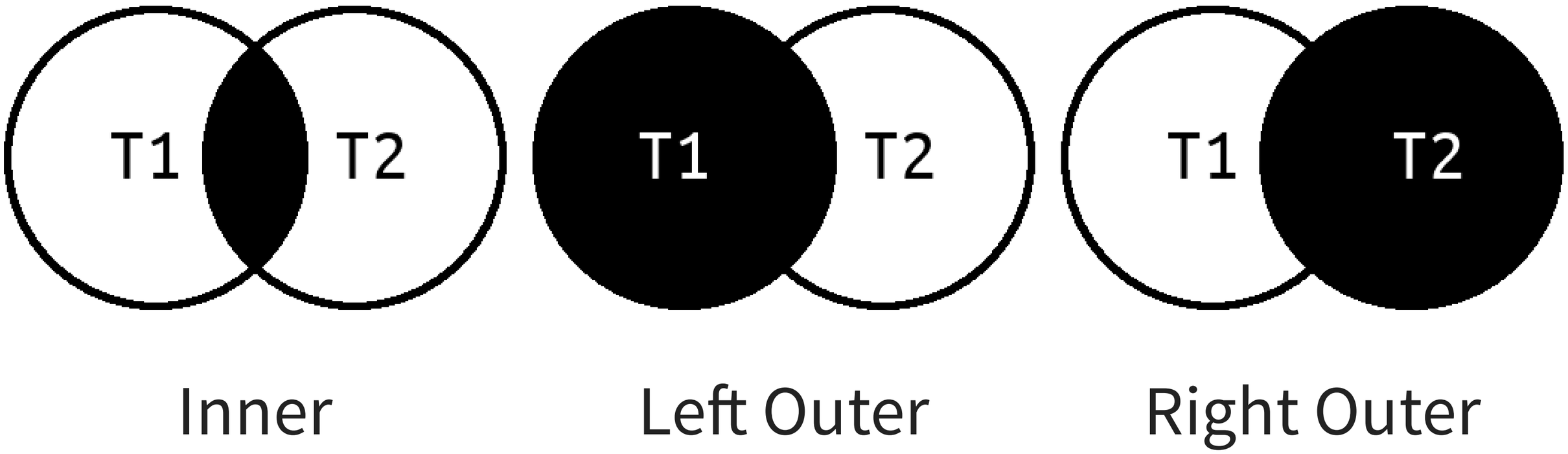
```
SELECT c.customerNumber, c.customerName,
 orderNumber, o.status
 FROM customers c LEFT JOIN orders o
ON c.customerNumber = o.customerNumber
WHERE orderNumber is NULL
```

RIGHT OUTER JOIN

MOSTRA TUTTI I DATI DELLA *SECONDA* TABELLA, E SE POSSIBILE ASSOCIA LE INFORMAZIONI DELLA PRIMA

```
SELECT ... FROM tabella1
 RIGHT OUTER JOIN tabella2
 ON PK = FK
```

INNER VS LEFT VS RIGHT OUTER JOIN



Inner Left Outer Right Outer

RIGHT OUTER JOIN

MOSTRA TUTTI I CLIENTI ED I RELATIVI ORDINI, INCLUSI I CLIENTI CHE NON HANNO FATTO ORDINI

```
SELECT c.customerNumber, c.customerName,
 orderNumber, o.status
 FROM orders o RIGHT JOIN customers c
 ON c.customerNumber = o.customerNumber
```

JOIN MULTIPLE

DECODIFICARE IL CONTENUTO DI PIÙ TABELLE IN UNA SOLA QUERY

```
SELECT ... FROM tabella1
      [INNER|LEFT|RIGHT]JOIN tabella2
      ON PK = FK
      [INNER|LEFT|RIGHT]JOIN tabella3
      ON PK = FK
```

JOIN MULTIPLE

MOSTRA TUTTI I CLIENTI, IL NOME DELL'IMPIEGATO ASSOCIATO ED IL NUMERO DI TELEFONO DELL'UFFICIO

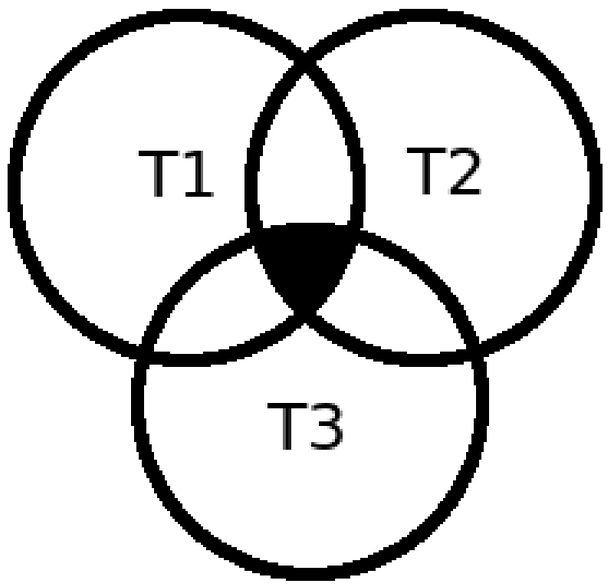
```
SELECT c.customerName, e.firstName, o.phone
     FROM customers c
     LEFT JOIN employees e
     ON c.salesRepEmployeeNumber = e.employeeNumber
     LEFT JOIN offices o
     USING (officeCode)
```

JOIN MULTIPLE

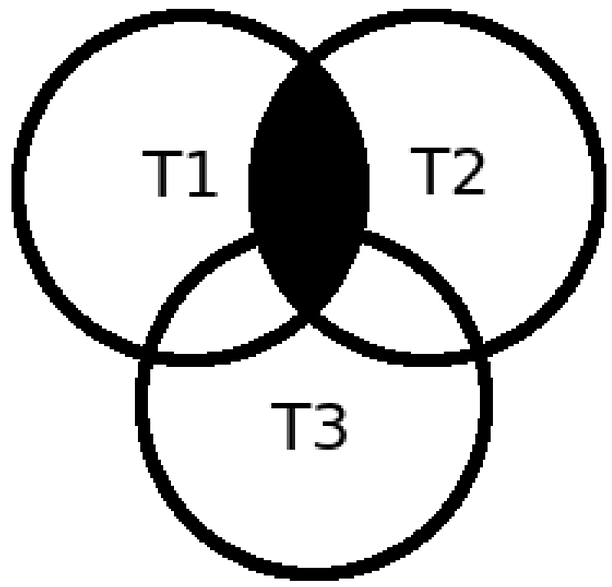
STAMPARE OGNI RIGA DELL'ORDINE, INDICANDO IL NOME DEL CLIENTE, NUMERO D'ORDINE ED IL NOME DEL PRODOTTO ORDINATO

```
SELECT c.customerName, o.orderNumber, p.productName
     FROM orderdetails d
     INNER JOIN orders o
     USING (orderNumber)
     INNER JOIN customers c
     USING (customerNumber)
     INNER JOIN products p
     USING (productCode)
     ORDER BY o.orderNumber, d.orderLineNumber;
```

JOIN MULTIPLE



Inner + Inner



Inner + Left Outer

CROSS JOIN

ID	Nome	Prezzo	Linea
1	Audi A5	50,00	A
2	Mercedes C	45,00	A
3	Smart	25,00	B

ID	Nome	Link
A	Auto Sportive	...
B	Micro Auto	...

ID	Nome	Prezzo	Linea	ID	Nome	Link
1	Audi A5	50,00	A	A	Auto Sportive	...
1	Audi A5	50,00	A	B	Micro Auto	...
2	Mercedes C	45,00	A	A	Auto Sportive	...
2	Mercedes C	45,00	A	B	Micro Auto	...
3	Smart	25,00	B	A	Auto Sportive	...
3	Smart	25,00	B	B	Micro Auto	...

INNER JOIN

ID	Nome	Prezzo	Linea
1	Audi A5	50,00	A
2	Mercedes C	45,00	A
3	Smart	25,00	B

ID	Nome	Link
A	Auto Sportive	...
B	Micro Auto	...

ID	Nome	Prezzo	Linea	ID	Nome	Link
1	Audi A5	50,00	A	A	Auto Sportive	...
2	Mercedes C	45,00	A	A	Auto Sportive	...
3	Smart	25,00	B	B	Micro Auto	...

INNER JOIN E NULL

ID	Nome	Prezzo	Linea
1	Audi A5	50,00	A
2	Mercedes C	45,00	NULL
3	Smart	25,00	B

ID	Nome	Link
A	Auto Sportive	...
B	Micro Auto	...

ID	Nome	Prezzo	Linea	ID	Nome	Link
1	Audi A5	50,00	A	A	Auto Sportive	...
3	Smart	25,00	B	B	Micro Auto	...

LEFT OUTER JOIN

ID	Nome	Prezzo	Linea
1	Audi A5	50,00	A
2	Mercedes C	45,00	NULL
3	Smart	25,00	B

ID	Nome	Link
A	Auto Sportive	...
B	Micro Auto	...

ID	Nome	Prezzo	Linea	ID	Nome	Link
1	Audi A5	50,00	A	A	Auto Sportive	...
2	Mercedes C	45,00	NULL	NULL	NULL	NULL
3	Smart	25,00	B	B	Micro Auto	...

JOIN MULTIPLE

ID	Nome	Colore	Linea
1	Audi A5	1	A
2	Mercedes C	2	A
3	Smart	2	B

ID	Nome	Link
A	Auto Sportive	...
B	Micro Auto	...

ID	Colore
1	Rosso
2	Blu

ID	Nome	Prezzo	Linea	ID	Nome	Link		
1	Audi A5	1	A	A	Auto Sportive	...	1	Rosso
2	Mercedes C	2	A	A	Auto Sportive	...	2	Blu
3	Smart	2	B	B	Micro Auto	...	2	Blu

FULL OUTER JOIN

MOSTRA TUTTI I DATI DELLA PRIMA TABELLA, E SE POSSIBILE ASSOCIA LE INFORMAZIONI DELLA SECONDA.

MOSTRA COMUNQUE TUTTI I DATI DELLA SECONDA TABELLA.

```
SELECT ... FROM tabella1
     FULL OUTER JOIN tabella2
     ON PK = FK
```

JOIN MULTIPLE

ID	Veicolo	Colore	Persona
1	Automobile	1	4
2	Bici	2	NULL
3	Moto	NULL	1
4	Scooter	3	3

ID	Cognome
1	Rossi
2	Bianchi
3	Bassi
4	Scaini

ID	Colore
1	Verde
2	Giallo
3	Blu

Persone che possiedono veicoli colorati

```
SELECT v.veicolo, c.colore,  p.cognome
     FROM veicolo v
     INNER JOIN persona p ON  v.id = p.id
     INNER JOIN colore c ON v.colore = c.id ;
```

Veicolo	Colore	Cognome
Automobile	Verde	Scaini
Scooter	Blu	Bassi

JOIN MULTIPLE

ID	Veicolo	Colore	Persona
1	Automobile	1	4
2	Bici	2	NULL
3	Moto	NULL	1
4	Scooter	3	3

ID	Cognome
1	Rossi
2	Bianchi
3	Bassi
4	Scaini

ID	Colore
1	Verde
2	Giallo
3	Blu

Persone che possiedono veicoli colorati o nessun veicolo

```
SELECT v.veicolo, c.colore,  p.cognome
     FROM persona p
     LEFT JOIN veicolo ON v.id = p.id
     INNER JOIN colore c ON v.colore = c.id ;
```

Veicolo	Colore	Cognome
Automobile	Verde	Scaini
Scooter	Blu	Bassi

JOIN MULTIPLE

ID	Veicolo	Colore	Persona
1	Automobile	1	4
2	Bici	2	NULL
3	Moto	NULL	1
4	Scooter	3	3

ID	Cognome
1	Rossi
2	Bianchi
3	Bassi
4	Scaini

ID	Colore
1	Verde
2	Giallo
3	Blu

Persone che possiedono veicoli colorati o nessun veicolo

```
SELECT v.veicolo, c.colore,  p.cognome
     FROM veicolo v
     INNER JOIN colore c ON v.colore = c.id
     RIGHT JOIN persona p ON v.id = p.id;
```

Veicolo	Colore	Cognome
Automobile	Verde	Scaini
Scooter	Blu	Bassi
Rossi	NULL	NULL
Bianchi	NULL	NULL

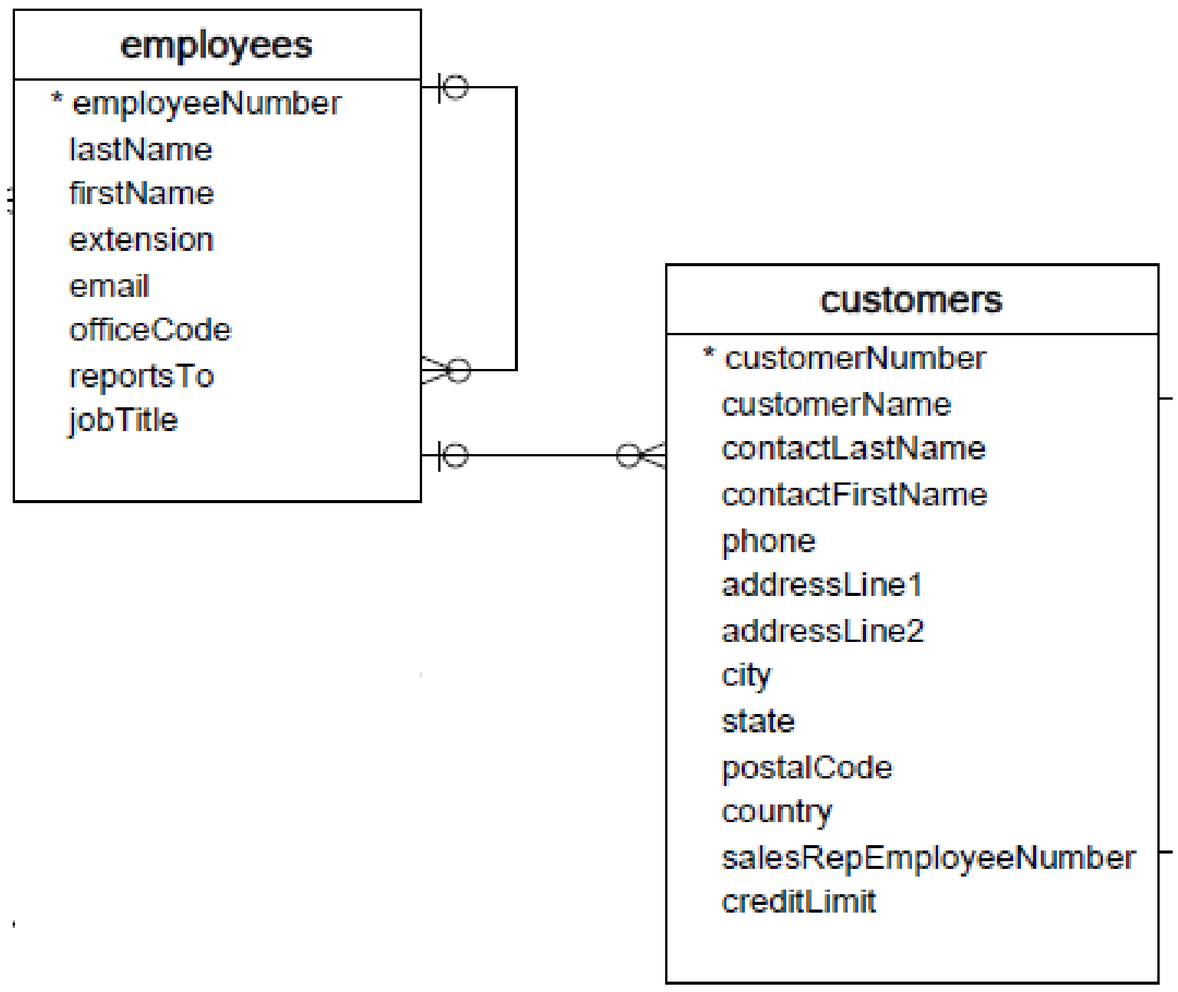
SELF JOIN

JOIN DI UNA TABELLA CON SE STESSA

```
SELECT ... FROM tabella1
      [LEFT|RIGHT|INNER] JOIN tabella1
      ON PK = FK
```

- avrò sicuramente nomi di attributi duplicati
- dovrò introdurre degli alias

SELF JOIN



employeeNumber	firstName	lastName	reportsTo
1002	Diane	Murphy	NULL
1056	Mary	Patterson	1002
1076	Jeff	Firrelli	1056
1088	William	Patterson	1056

SELF JOIN

MOSTRA TUTTI I DIPENDENTI ED IL NOME DEL LORO CAPO

```
SELECT m.employeeNumber, m.firstName,
     m.lastName, m.reportsTo, c.firstName,
     c.lastName FROM employees m
LEFT JOIN employees c
ON m.reportsTo = c.employeeNumber;
```

SELF JOIN

MOSTRA TUTTE LE COPPIE DI CLIENTI CHE ABITANO NELLA STESSA CITTÀ

```sql
SELECT c1.city, c1.customerName,
    c2.customerName
    FROM customers c1 INNER JOIN customers c2
    ON c1.city = c2.city;
```

NO

SELF JOIN

MOSTRA TUTTE LE COPPIE DI CLIENTI CHE ABITANO NELLA STESSA CITTÀ

```
SELECT c1.city, c1.customerName,c2.customerName
     FROM customers c1 INNER JOIN customers c2
     ON c1.city = c2.city AND
     c1.customername <> c2.customerName
```

SELF JOIN

MOSTRA TUTTE LE COPPIE DI CLIENTI CHE ABITANO NELLA STESSA CITTÀ

```
SELECT c1.city, c1.customerName,c2.customerName
     FROM customers c1 INNER JOIN customers c2
         ON c1.city = c2.city AND
     c1.customername <> c2.customerName
```

forma equivalente

```
SELECT c1.city, c1.customerName,c2.customerName
     FROM customers c1 INNER JOIN customers c2
     ON c1.city = c2.city
     WHERE c1.customername <> c2.customerName
```

UNION JOIN

UNISCE I RISULTATI DI PIÙ QUERY

```
SELECT ... FROM ...
 UNION [DISTINCT | ALL]
 SELECT ... FROM ...
 [UNION [DISTINCT | ALL]
 SELECT ... FROM ...]
```

ATTENZIONE!

- stesso numero di attributi
- attributi omogenei

UNION

```
SELECT ... FROM ...
 UNION [DISTINCT | ALL]
 SELECT ... FROM ...
 [UNION [DISTINCT | ALL]
 SELECT ... FROM ...]
```

- `DISTINCT`: default, elimina duplicati
- `ALL`: se specificato, NON elimina duplicati

UNION

MOSTRA L'IDENTIFICATIVO ED IL NOME DI TUTTI GLI IMPIEGATI E DI TUTTI I CLIENTI

```
SELECT customerNumber AS id,contactLastname AS name
 FROM customers
UNION
SELECT employeeNumber AS id, firstname AS name
FROM employees;
```

UNION

MOSTRA L'IDENTIFICATIVO ED IL NOME DI TUTTI GLI IMPIEGATI E DI TUTTI I CLIENTI

```
SELECT customerNumber,contactLastname
 FROM customers
 UNION
 SELECT employeeNumber, firstname
FROM employees;
```

- Se non specifico l'alias prende i nomi della prima query

UNION

MOSTRA L'ID ED IL NOME DI TUTTI GLI IMPIEGATI E DI TUTTI I CLIENTI, SCRIVENDO PER OGNUNO COSA SIA

```
SELECT customerNumber AS id,
 contactLastname AS name, "Cliente" AS tipo
 FROM customers
 UNION
SELECT employeeNumber AS id,
concat(firstname," ",lastname) AS name,
"Impiegato" AS tipo
FROM employees;
```

UNION E ORDINAMENTO

E SE VOLESSI ORDINARE I RISULTATI?

```
SELECT ... FROM ...
 UNION [DISTINCT | ALL]
 SELECT ... FROM ...
 ORDER BY criteri
```

ATTENZIONE!

- l’ultima riga è riferita al RISULTATO della union, non all’ultima query
- se si inserisce la clausola all’interno della singola query verrà ignorata

UNION E ORDINAMENTO

POSSO USARE LE PARENTESI PER FARE ORDINE

```
(SELECT ... FROM ...)
 UNION [DISTINCT | ALL]
 (SELECT ... FROM ...)
ORDER BY criteri
```

UNION ED ORDINAMENTO

MOSTRARE ID E NOME DI CLIENTI ED IMPIEGATI ORDINANDOLI PER NOME

```
(SELECT customerNumber AS id,
 contactLastname AS name
 FROM customers)
UNION
(SELECT employeeNumber AS id, firstname AS name
FROM employees)
ORDER BY name;
```

UNION ED ORDINAMENTO

MOSTRARE I PAESI IN CUI C'È UN UFFICIO O UN CLIENTE, ORDINATI PER NOME

```
SELECT country FROM offices
 UNION
 SELECT country FROM customers
 ORDER BY country;
```

INTERSECT

RESTITUISCE L'INTERSEZIONE DI PIÙ QUERY

```
SELECT ... FROM ...
 INTERSECT
 SELECT ... FROM ...
 INTERSECT
SELECT ... FROM ...
```

Non c'è in MySql, servono query nidificate

RAGGRUPPARE I DATI

VOGLIO RAGGRUPPARE LE ENNUPLE IN SOTTOGRUPPI IN BASE AD UNO O PIÙ VALORI

```
SELECT a1 , a2 ,...,an
FROM tabella1 WHERE condizioni
GROUP BY a1, a2,...,an
```

RAGGRUPPARE I DATI

MOSTRA TUTTI GLI STATI DEGLI ORDINI ESISTENTI

```
SELECT status
FROM orders
GROUP BY status
```

RAGGRUPPARE I DATI

MOSTRA TUTTI GLI STATI DEGLI ORDINI FATTI PRIMA DEL 31/12/2003

```sql
SELECT status
FROM orders
WHERE orderDate < "2003-12-31"
GROUP BY status;
```

RAGGRUPPARE I DATI

MOSTRA TUTTI GLI STATI DEGLI ORDINI FATTI PRIMA DEL 31/12/2003

```sql
SELECT status
FROM orders
WHERE orderDate < "2003-12-31"
GROUP BY status;
```

forma equivalente

```sql
SELECT DISTINCT status
FROM orders
WHERE orderDate < "2003-12-31"
```

FUNZIONI DI AGGREGAZIONE

PERMETTONO DI EFFETTUARE CALCOLI SU TUTTI I VALORI CHE L'ATTRIBUTO ASSUME NELLA QUERY ESEGUITA

```
SELECT a1, a2, ..., an, aggregatore(ax)
FROM tabella1 WHERE condizioni
```

Esempio:

- quanti ordini sono stati fatti?
- quanto costa in media un prodotto?

FUNZIONI DI AGGREGAZIONE

```sql
SELECT a1, a2, ..., an, aggregatore(ax)
FROM tabella1 WHERE condizioni
```

Aggregatori:

- `COUNT`: conta il numero di valori presenti
- `SUM`: somma dei valori
- `AVG`: media dei valori
- `MAX`/`MIN`: massimo e minimo

FUNZIONI DI AGGREGAZIONE

QUANTI DIPENDENTI CI SONO IN AZIENDA?

```sql
SELECT count(*)
FROM employees;
```

FUNZIONI DI AGGREGAZIONE - NULL

Differenza fra

```
SELECT count(*)
FROM employees
```

e

```
SELECT count(reportsTo)
FROM employees
```

FUNZIONI DI AGGREGAZIONE - DISTINCT

QUANTI CAPI CI SONO IN AZIENDA?

```
SELECT count( distinct reportsTo)
FROM employees;
```

FUNZIONI DI AGGREGAZIONE

QUANTI PAGAMENTI HO RICEVUTO?

```sql
SELECT count(*)
FROM payments;
```

FUNZIONI DI AGGREGAZIONE

QUANTI SOLDI HO RICEVUTO CON I PAGAMENTI?

```sql
SELECT SUM(amount)
FROM payments;
```

FUNZIONI DI AGGREGAZIONE

QUAL È IL PREZZO MEDIO DI VENDITA DI UN PRODOTTO?

```
SELECT avg(MSRP)
FROM products;
```

FUNZIONI DI AGGREGAZIONE

QUAL È IL PREZZO MEDIO DI VENDITA DI UN PRODOTTO? QUALE IL MASSIMO? QUALE IL MINIMO?

```
SELECT avg(MSRP), max(MSRP), min(MSRP)
FROM products;
```

FUNZIONI DI AGGREGAZIONE ERRATE

ATTENZIONE A NON CHIEDERE COSE ASSURDE

```
SELECT avg(MSRP), productName
FROM products
```

- quale productName devo mostrare?
- standard ANSI: errore
- MySql risponde… con il primo valore!

FUNZIONI DI AGGREGAZIONE E RAGGRUPPAMENTI

VOGLIO RAGGRUPPARE LE ENNUPLE IN SOTTOGRUPPI IN BASE AD UNO O PIÙ VALORI, MOSTRANDO ANCHE VALORI CALCOLATI SU OGNI GRUPPO

```
SELECT a1, a2 , ... , an, aggregatore(ax)
FROM tabella1 WHERE condizioni
GROUP BY a1, a2, ... ,an
```

FUNZIONI DI AGGREGAZIONE E RAGGRUPPAMENTI

MOSTRA GLI STATI DEGLI ORDINI E QUANTI ORDINI SI TROVANO IN CIASCUNO STATO

```sql
SELECT status, count(*)
FROM orders
GROUP BY status;
```

FUNZIONI DI AGGREGAZIONE E RAGGRUPPAMENTI

MOSTRA QUANTI PRODOTTI HO PER OGNI CATEGORIA ED IL PREZZO MEDIO DI VENDITA

```sql
SELECT productLine, count(*), avg(MSRP)
FROM products
GROUP BY productLine;
```

AGGREGHIAMO...

MOSTRARE QUANTI ORDINI HO SPEDITO OGNI GIORNO

```
SELECT count(*), shippedDate
FROM orders
GROUP BY shippedDate
```

AGGREGHIAMO...

MOSTRARE QUANTI ORDINI HO SPEDITO NEI VARI MESI (UNA RIGA PER MESE)

- PostgreSQL: usate `date_part('month', attributo)`

```
SELECT count(*), month(shippedDate)
FROM orders
WHERE shippedDate IS NOT NULL
GROUP BY month(shippedDate);
```

AGGREGHIAMO...

MOSTRARE QUANTI ORDINI HO SPEDITO NEI VARI MESI (UNA RIGA PER MESE ED ANNO)

- PostgreSQL (mese): usate `date_part('month', attributo)`
- PostgreSQL (anno): usate `date_part('year', attributo)`

```
SELECT count(*), month(shippedDate),
year(shippedDate)
FROM orders
WHERE shippedDate IS NOT NULL
GROUP BY year(shippedDate), month(shippedDate);
```

AGGREGHIAMO…

MOSTRARE PER OGNI ORDINE: IL NOME DEL CLIENTE, LA DATA DELL’ORDINE ED IL TOTALE DELL’ORDINE

Un passo alla volta:

- calcoliamo il totale di ogni ordine
- estraiamo i dati dalle altre tabelle

AGGREGHIAMO...

MOSTRARE PER OGNI ORDINE: IL TOTALE DELL'ORDINE

```
SELECT orderNumber,
sum(quantityOrdered*priceEach)
FROM orderdetails
GROUP BY orderNumber;
```

AGGREGHIAMO...

MOSTRARE PER OGNI ORDINE: IL NOME DEL CLIENTE, LA DATA DELL'ORDINE ED IL TOTALE DELL'ORDINE

```
SELECT customerName, orderDate,
sum(quantityOrdered*priceEach)
FROM orderdetails
INNER JOIN orders USING (orderNumber)
INNER JOIN customers USING (customerNumber)
GROUP BY orderNumber;
```

AGGREGHIAMO...

MOSTRARE QUANTI ORDINI HA FATTO OGNI CLIENTE, METTENDO IN CIMA QUELLI PIÙ ASSIDUI

```sql
SELECT count(*) nOrdini, customerNumber
FROM orders
GROUP BY customerNumber
ORDER BY nOrdini DESC;
```

AGGREGHIAMO…

MOSTRARE L’ESTRATTO CONTO DEL CLIENTE 124

- pagamenti cliente: importo negativo
- debiti cliente: importo positivo

Un passo alla volta:

1. estrarre i pagamenti fatti con la relativa data
2. prendere i totali degli ordini del cliente
3. unire i due risultati, ordinandoli per data

AGGREGHIAMO...

MOSTRARE L'ESTRATTO CONTO DEL CLIENTE 124

ESTRARRE I PAGAMENTI FATTI CON LA RELATIVA DATA

```
SELECT amount*-1, paymentDate FROM payments
WHERE customerNumber = 124
```

AGGREGHIAMO...

MOSTRARE L'ESTRATTO CONTO DEL CLIENTE 124

PRENDERE I TOTALI DEGLI ORDINI DEL CLIENTE

```
SELECT sum(quantityOrdered*priceEach),orderDate
FROM orderdetails INNER JOIN orders
USING (orderNumber)
WHERE customerNumber = 124
GROUP BY orderNumber;
```

AGGREGHIAMO...

MOSTRARE L'ESTRATTO CONTO DEL CLIENTE 124

UNIRE I DUE RISULTATI, ORDINANDOLI PER DATA

```
SELECT amount*-1, paymentDate FROM payments
WHERE customerNumber = 124
UNION
SELECT sum(quantityOrdered*priceEach),orderDate
FROM orderdetails INNER JOIN orders
USING (orderNumber)
WHERE customerNumber = 124
GROUP BY orderNumber
ORDER BY paymentDate;
```

FILTRARE DATI AGGREGATI

COME APPLICARE UN FILTRO AL RISULTATO DI UNA FUNZIONE DI AGGREGAZIONE?

```
SELECT a1, a2, ... ,an, aggregatore(ax)
    FROM tabella1 WHERE condizioni
    GROUP BY a1, a2, ... ,an
    HAVING condizioniAggregate
```

FILTRARE DATI AGGREGATI

MOSTRARE TUTTI GLI ORDINI IL CUI TOTALE È < 10.000

```
SELECT orderNumber,
    sum(quantityOrdered*priceEach) as tot
FROM orderdetails
GROUP BY orderNumber
HAVING tot < 10000;
```

FILTRARE DATI AGGREGATI

MOSTRARE TUTTI GLI ORDINI IL CUI TOTALE È < 10.000 E PER I QUALI VERRANNO SPEDITI PIÙ DI 100 PEZZI

```
SELECT orderNumber,
    sum(quantityOrdered) as q,
    sum(quantityOrdered*priceEach) as tot
    FROM orderdetails
    GROUP BY orderNumber
    HAVING tot < 10000 AND q > 100;
```

FILTRARE DATI AGGREGATI

MOSTRARE TUTTI GLI ORDINI IL CUI TOTALE È < 10.000 E CHE NON SONO STATI SPEDITI

Hint: ordini spediti hanno status "Shipped"

```
SELECT ordernumber, status,
    SUM(quantityOrdered*priceeach) total
    FROM orderdetails
    INNER JOIN orders USING(ordernumber)
    GROUP BY ordernumber
    HAVING status <> 'Shipped' AND total < 10000;
```

FILTRARE DATI AGGREGATI

```
SELECT ordernumber, status,
    SUM(quantityOrdered*priceeach) total
    FROM orderdetails
    INNER JOIN orders USING(ordernumber)
    GROUP BY ordernumber
    HAVING status <> 'Shipped' AND total < 10000;
```

forma equivalente

```
SELECT ordernumber, status,
    SUM(quantityOrdered*priceeach) total
    FROM orderdetails
    INNER JOIN orders USING(ordernumber)
    WHERE status <> 'Shipped'
    GROUP BY ordernumber
    HAVING total < 10000;
```

SUBQUERY

POSSO ANNIDIARE LE QUERY UNA DENTRO L'ALTRA

```
SELECT a1,a2,...,an,(QUERY singolo val.)
    FROM (QUERY)
    WHERE a1 > (QUERY singolo val.)
    AND a2 IN (QUERY singolo attrib.)
```

Per adesso:

- le subquery vivono di vita propria
- possiamo scriverle separatamente, poi incorporarle

SUBQUERY - SINGOLO VALORE

MOSTRARE PER OGNI ARTICOLO IL PREZZO DI VENDITA ED IL PREZZO DEL PRODOTTO PIÙ CARO

Hint: subquery nella clausola `SELECT`

```
SELECT productName, MSRP,
    (SELECT max(MSRP)
    FROM products) as massimo
    FROM products;
```

SUBQUERY - SINGOLO VALORE

MOSTRARE I DATI DEL PAGAMENTO PIÙ ALTO RICEVUTO

Hint: subquery nella clausola `WHERE`

```sql
SELECT customerNumber, checkNumber, amount
    FROM payments
    WHERE amount =
    (SELECT MAX(amount)
    FROM payments);
```

SUBQUERY - SINGOLO VALORE

MOSTRA I PAGAMENTI SUPERIORI ALLA MEDIA

```
SELECT customerNumber, checkNumber, amount
    FROM payments
    WHERE amount >
    (SELECT AVG(amount)
    FROM payments);
```

SUBQUERY - SINGOLO VALORE

MOSTRARE I CLIENTI CHE NON HANNO FATTO ORDINI

Hint: subquery nella clausola `WHERE ... NOT IN`

```
SELECT customername
    FROM customers
    WHERE customerNumber NOT IN
    (SELECT DISTINCT customernumber
    FROM orders);
```

SUBQUERY - SINGOLO ATTRIBUTO

MOSTRARE I CLIENTI CHE NON HANNO FATTO ORDINI

```
SELECT customername
    FROM customers
    WHERE customerNumber NOT IN
    (SELECT DISTINCT customernumber
    FROM orders);
```

forma equivalente

```
SELECT customername
    FROM customers
    LEFT JOIN orders USING (customerNumber)
```

SUBQUERY - FROM

MOSTRARE IL NUMERO MASSIMO, MINIMO E MEDIO DI PEZZI INSERITI NEGLI ORDINI

Hint: creare prima la query che somma le righe dell'ordine

```sql
SELECT max(items), min(items),
    floor(avg(items)) as media
    FROM (SELECT orderNumber,
    SUM(quantityOrdered) AS items
    FROM orderdetails
    GROUP BY orderNumber) AS lineitems;
```

SUBQUERY CORRELATE

Finora:

- posso eseguire la subquery da sola
- il motore la esegue una volta
- … non è sempre così

SUBQUERY CORRELATE

```
SELECT a1,a2,...,an
    FROM tab1 WHERE
    a1 > (SELECT c1
    FROM tab2
    WHERE tab2.c2 > tab1.a1)
```

- non eseguibile “da sola”
- eseguita per ogni riga della query principale
- visibilità variabili: solo da query a subquery

SUBQUERY CORRELATE

MOSTRARE I PRODOTTI IL CUI PREZZO DI ACQUISTO È SUPERIORE ALLA MEDIA DELLA LINEA CUI AFFERISCONO

```
SELECT productname, buyprice
    FROM products AS p
    WHERE buyprice > (
    SELECT AVG(buyprice)
    FROM products
    WHERE productline = p.productline);
```

EXISTS

OPERATORE BOOLEANO (PER WHERE): RITORNA VERO SE UNA SOTTOQUERY HA VALORI

```
SELECT a1,a2,...,an
    FROM tab
    WHERE EXISTS (QUERY singolo val.)
```

LIMITI

NON VOGLIO TUTTE LE RIGHE CHE SODDISFANO IL FILTRO, SOLO LE TOP N

```
SELECT a1,a2,...,an
    FROM tab
    WHERE ...
    LIMIT numero
```

SUBQUERY CORRELATE

MOSTRARE I PRIMI 5 CLIENTI (CODICE CLIENTE, NOME E LIMITE DI CREDITO)

```
SELECT customernumber,
    customername,
    creditlimit
FROM customers
LIMIT 5;
```

SUBQUERY CORRELATE

MOSTRARE I 5 CLIENTI CON IL CREDITO PIÙ ELEVATO

```
SELECT customernumber, customername,
    creditlimit
    FROM customers
ORDER BY creditlimit DESC
LIMIT 5;
```

LIMITI

NON VOGLIO TUTTE LE RIGHE CHE SODDISFANO IL FILTRO: SOLO LE PRIME N A PARTIRE DALLA RIGA X.

```
SELECT a1,a2,...,an
    FROM tab
    WHERE ...
    LIMIT X, N
```

- X → dalla riga Xesima dei risultati (si parte da 0)
- N → quante righe prendere

LIMITI

MOSTRARE IL SECONDO PRODOTTO PIÙ COSTOSO (BUYPRICE) IN LISTINO

```
SELECT productName, buyprice
    FROM products
    ORDER BY buyprice DESC
    LIMIT 1, 1;
```

AGGIUNGERE DATI

COMO POSSONO INSERIRE UNA NUOVA n-UPLA?

```
INSERT INTO tabella(col1, col2, ...)
    VALUES (valore1, valore2, ...)
    [, (valore1, valore2, ...), ...]
```

- se imposto TUTTI gli attributi della tabella, posso omettere i nomi delle colonne (col1, col2,…)
- attributi `AUTO_INCREMENT: NULL`

AGGIUNGERE DATI

INSERIRE UN NUOVO UFFICIO A TRIESTE

Hint: `INSERT INTO tabella (col1,col2,...) VALUES (valore1,valore2,...)`

```
INSERT INTO offices
    VALUES (8, 'Trieste', '+30 040558555',
    'Via Valerio 10', null, null,
    'Italy', '34100', 'EMEA')
```

AGGIUNGERE DATI E SUBQUERY

VORREI USARE UNA SUBQUERY PER ALIMENTARE L'INSERIMENTO DEI DATI

```
INSERT INTO tabella(col1, col2, ...)
    SELECT ...
```

Il risultato della `SELECT` deve fornire:

- lo stesso numero di attributi della tabella di destinazione
- attributi dello stesso dominio di destinazione

AGGIUNGERE DATI

DUPLICARE L'ORDINE 10425

1. creare l'ordine 10426 (a mano, tabella orders)
2. prendere tutte le righe di 10425 (orderdetails)
3. inserire tutte queste n-uple nella tabella (orderdetails)

AGGIUNGERE DATI

DUPLICARE L'ORDINE 10425

creare l’ordine 10426 (a mano, tabella orders)

Hint: `INSERT INTO tabella (col1,col2,...) VALUES (valore1,valore2,...)`

```
INSERT INTO orders
    VALUES (10426, '2014-11-11',
    '2014-11-30', null,
    'In Process',
    'Duplica 10425', 119);
```

AGGIUNGERE DATI

DUPLICARE L'ORDINE 10425

prendere tutte le righe di 10425 (orderdetails)

Hint: conta l'ordine delle colonne, vi servirà FK 10426

```
SELECT 10426 AS numero, productCode,
    quantityordered, priceEach,
    orderLineNumber
    FROM orderdetails
    WHERE orderNumber = 10425;
```

AGGIUNGERE DATI

DUPLICARE L'ORDINE 10425

inserire tutte queste n-uple nella tabella (orderdetails)

Hint: `INSERT INTO tabella (col1,col2,...) SELECT ...`

```mysql
INSERT INTO orderdetails
    SELECT 10426 AS numero, productCode,
    quantityordered, priceEach,
    orderLineNumber
    FROM orderdetails
    WHERE orderNumber=10425;
```

MODIFICARE DATI

COME POSSO MODIFICARE n-UPLE ESISTENTI?

```
UPDATE tabella
SET col1 = valore1
[, col2 = val2...]
[WHERE condizione]
```

- ogni campo può assumere un valore esplicito, o il risultato di una funzione, sottoquery, ecc…
- se non uso la clausola `WHERE`, aggiorno tutte le n-uple della tabella

MODIFICARE DATI

CAMBIARE INDIRIZZO EMAIL A MARY PATTERSON

Impiegato 1056, mettete una email a scelta

Hint: `UPDATE tabella SET col1 = valore1 WHERE condizione`

```sql
UPDATE employees
    SET email = 'mary.patterson@classicmodelcars.com'
    WHERE employeeNumber = 1056;
```

MODIFICARE DATI

CAMBIARE PREZZO DI ACQUISTO E VENDITA DELLA '2001 FERRARI ENZO'

Hint: `UPDATE tabella SET col1 = valore1 WHERE condizione`

```sql
UPDATE products
    SET msrp = 500, buyprice = 200
    WHERE productName = '2001 Ferrari Enzo';
```

MODIFICARE DATI

AUMENTARE DEL 5% TUTTI I PREZZI DI VENDITA

Hint: `UPDATE tabella SET col1 = valore1 WHERE condizione`

```
UPDATE products
    SET msrp = msrp*1.05;
```

MODIFICARE DATI

ASSOCIARE AI CLIENTI SENZA VENDITORE L’AGENTE CON MATRICOLA PIÙ ALTA (APPENA ARRIVATO)

1. individuare i clienti senza venditore
2. trovare l’agente con matricola più alta
3. aggiornare i dati

MODIFICARE DATI

ASSOCIARE AI CLIENTI SENZA VENDITORE L'AGENTE CON MATRICOLA PIÙ ALTA

Individuare i clienti senza venditore

```
SELECT customerNumber
    FROM customers
    WHERE salesRepEmployeeNumber IS NULL;
```

MODIFICARE DATI

ASSOCIARE AI CLIENTI SENZA VENDITORE L’AGENTE CON MATRICOLA PIÙ ALTA

Trovare l'agente con matricola più alta

Hint: venditore ha `jobTitle = 'Sales Rep'`

```
SELECT max(employeeNumber)
    FROM employees
    WHERE jobTitle = 'Sales Rep';
```

MODIFICARE DATI

ASSOCIARE AI CLIENTI SENZA VENDITORE L'AGENTE CON MATRICOLA PIÙ ALTA

Aggiornare i dati

```
UPDATE customers
    SET salesRepEmployeeNumber =
    (SELECT max(employeeNumber)
    FROM employees
    WHERE jobTitle = 'Sales Rep')
    WHERE salesRepEmployeeNumber IS NULL;
```

ATTENZIONE

```
UPDATE tabella
    SET col1 = col1 +1, col2 = col1
```

- MySql: la seconda operazione è fatta con il valore aggiornato di `col1`
- SQL standard: la seconda operazione è fatta con il valore originale di `col1`

ELIMINARE DATI

COME POSSO ELIMINARE DATI DALLA TABELLA?

```
DELETE FROM tabella
[WHERE condizioni]
```

oppure

```
DELETE FROM tabella
[WHERE condizioni]
```

ONCE AND FOR ALL:

- non si torna indietro
- prima di eseguire la query, provare a metterci una `SELECT * ` per vedere che succede

ELIMINARE DATI

ELIMINARE TUTTI I CLIENTI ITALIANI

Hint: `DELETE FROM tab WHERE ...`

```sql
DELETE FROM customers
    WHERE country = "Italy";
```

ELIMINARE DATI

PERCHÈ NON FUNZIONA?

Opzioni di MySql WorkBench:

```
Error Code: 1175. You are using safe update
    mode and you tried to update a table without
    a WHERE that uses a KEY column To disable
    safe mode, toggle the option in Preferences ->
    SQL Queries and reconnect.
```

ELIMINARE DATI

PERCHÈ NON FUNZIONA?

Vincoli interrelazionali:

```
Error executing SQL statement.
    ERROR: update or delete on table "customers" violates fore
    Dettaglio: Key (customernumber)=(249) is still referenced
```

Devo prima modificare le altre tabelle!

CHECK

PERMETTE DI INTRODURRE VINCOLI DI INTEGRITÀ GENERICI

```
CREATE TABLE nomeTab(
    attr1 tipo1 CHECK (condizione),
    attr2 tipo2,...,
    CHECK (condizione))
```

- le condizioni sono espressioni booleane
- possono essere complesse (es: subquery)
- non funzionano in MySql

CHECK

- il genere può essere solo M o F
- stipendio inferiore a quello del capo

```
CREATE TABLE Impiegato(
matricola integer,
cognome character(20),
sesso character NOT NULL
    CHECK (sesso in ('M','F')),
stipendio integer, superiore integer,
    CHECK (stipendio <=
        (SELECT stipendio
        FROM Impiegato J
        WHERE superiore = J.matricola))
```

ASSERTION

PERMETTONO DI INTRODURRE VINCOLI DI INTEGRITÀ A LIVELLO DI SCHEMA

```
CREATE ASSERTION nome CHECK (condizione)
```

Stesse note del check

ASSERTION

LA TABELLA IMPIEGATO DEVE AVERE ALMENO UN NOMINATIVO

```
CREATE ASSERTION AlmenoUnImpiegato
    CHECK ((SELECT count(*) FROM Impiegato) >= 1)
```

VARIABILI

POSSO DEFINIRE VARIABILI DA USARE NELLE QUERY

```
SET @variabile = valore;
SELECT @variabile;
```

- vivono e muoiono nella sessione
- il valore può essere anche una query che ritorna un solo dato
- MySQL: case insensitive, solo tipi semplici (integer, decimal, string, …)

VARIABILI

CREARE UNA VARIABILE DI NOME “PIPPO”, ASSEGNARCI IL VALORE “HELLO WORD!” E MOSTRARNE IN CONTENUTO

```
SET @pippo = 'Hello, World!';
SELECT @pippo;
```

VARIABILI

SALVARE NELLA VARIABILE “PREZZO” IL PREZZO PIÙ ALTO (MSRP) PRESENTE A LISTINO E MOSTRARNE IL VALORE

```
SET @prezzo = (SELECT max(msrp)
FROM products);
SELECT @prezzo;
```

VARIABILI

MOSTRARE I PRODOTTI IN CUI IL VALORE MSRP È PARI ALLA VARIABILE APPENA IMPOSTATA

```
SELET * FROM products
WHERE MSRP = @prezzo;
```

ESECUZIONE QUERY

Cosa accade quando invio una query al server?

1. **Parser**: trasforma il testo in un albero di comandi
2. **PreProcessor**: la sintassi è corretta?
3. **Security**: l'utente può fare questo?
4. **Optimizer**: posso riscrivere la query in modo più intelligente?
5. **Execution Engine**: effettua l'operazione
6. **Trasmissione Dati**

E se devo eseguire spesso la stessa query??

PROFILING (MYSQL)

COSA FA IL MOTORE?

```
SET profiling = 1;
esecuzione comandi
SHOW PROFILES;
SHOW PROFILE [FOR QUERY n];
SET profiling = 0;
```

- `SHOW PROFILES`: storico dei tempi di esecuzione
- `SHOW PROFILE`: come ho impiegato il tempo nell'ultima query/query specificata?

PROFILING

COSA POSSO VEDERE?

Tipo

- `ALL`: tutte le informazioni
- `CPU`: tempo CPU per user/system
- `SWAPS`: utilizzo della memoria su disco
- `SOURCE`: nome della funzione/libreria usati

PRIVILEGI

MOSTRA TUTTI I DATI DELL'ULTIMA QUERY

```
SHOW PROFILE ALL FOR QUERY 4;
```

PREPARED STATEMENT

POSSO PRECOMPILARE LE QUERY CHE USO PIÙ SPESSO

```
PREPARE nomeStatement FROM 'query';
EXECUTE nomeStatement USING p1, p2,...;
DEALLOCATE PREPARE nomeStatement;
```

- `PREPARE`: crea una query riutilizzabile, che può ricevere parametri
- `EXECUTE`: esegue il comando salvato
- `DEALLOCATE PREPARE`: elimina il comando
- vivono e muoiono nella sessione

PREPARED STATEMENT: PREPARE

CREARE LO STATEMENT (MYSQL)

```
PREPARE nomeStatement FROM
    'SELECT a1,a2,...
    FROM tabella
    WHERE a1 = ? AND a2 = ?';
```

- la query da eseguire è passata come stringa
- ogni “?”” corrisponde ad un parametro che verrà comunicato in sede di esecuzione

PREPARED STATEMENT: PREPARE

CREARE LO STATEMENT (POSTGRESQL)

```
PREPARE nomeStatement(type1, type2, ...) AS
        SELECT a1,a2,...
        FROM tabella
        WHERE a1 = $1 AND a2 = $2;
```

- specifico i tipi tra parentesi
- si fa riferimento ai paramentri tramite `$`

PREPARED STATEMENT: PREPARE

CREARE LO STATEMENT “STMT1” IL QUALE SELEZIONA PRODUCTCODE E PRODUCTNAME DAL LISTINO MOSTRANDO SOLO LE ENNUPLE CON MSRP MAGGIORE DEL PARAMETRO CHE VERRÀ FORNITO.

Hint: `PREPARE nomeStatement FROM 'SELECT a1,a2,... FROM tabella WHERE a1 = ? AND a2 = ?';`

```
PREPARE stmt1 FROM
    'SELECT productCode, productName
    FROM products WHERE MSRP > ?';
```

PREPARED STATEMENT: EXECUTE

ESEGUIRE LO STATEMENT (MYSQL)

```
EXECUTE nomeStatement
    [USING @var1, @var2,...];
```

- i parametri devono essere variabili
- in teoria sono opzionali (posso creare statement senza parametri, ma è meglio evitarlo)

PREPARED STATEMENT: EXECUTE

ESEGUIRE LO STATEMENT (POSTGRESQL)

```
EXECUTE nomeStatement(arg1, arg2, ...);
```

PREPARED STATEMENT: PREPARE

USANDO STMT1 MOSTRARE I PRODOTTI CON PREZZO SUPERIORE AI 100$

Hint: `EXECUTE nomeStatement [USING @var1, @var2,...];`

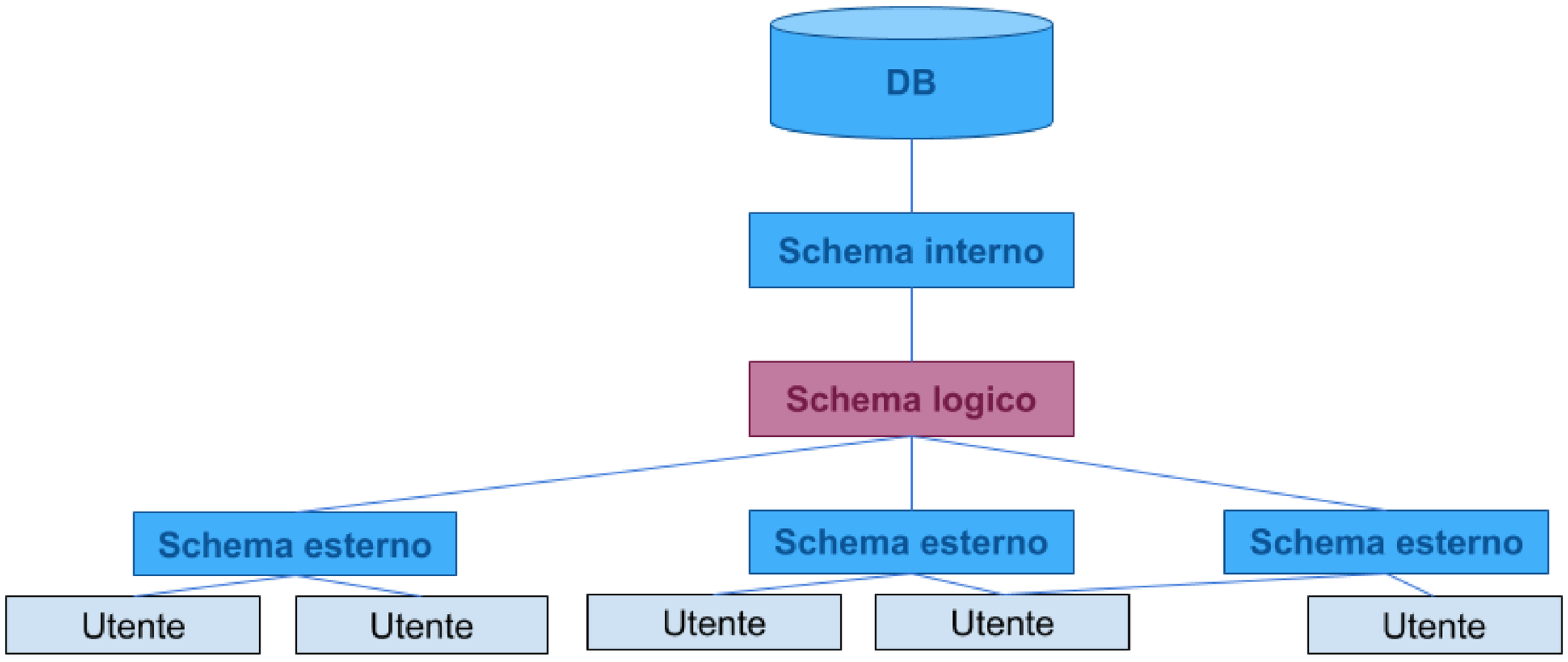
```
SET @MSRP = 100;
EXECUTE stmt1 USING @MSRP;
```

PREPARED STATEMENT: DEALLOCATE

ELIMINARE LO STATEMENT

```
DEALLOCATE PREPARE nomeStatement;
DROP PREPARE nomeStatement;
```

SCHEMA ESTERNO



SCHEMA ESTERNO

CREARE UNO SCHEMA ESTERNO CHE MOSTRI IL LISTINO AI CLIENTI (CODICE E NOME PRODOTTO, MSRP).

```
PREPARE stmtListinoClienti FROM
    'SELECT productCode, productName, MSRP FROM products';
```

Problemi:

- codice inserito nell’applicazione
- sparisce quando chiudo la connessione
- va creato per ogni connessione
- accesso diverso da una normale tabella

VISTE

RAPPRESENTAZIONE ALTERNATIVA DEI DATI

- SQL SELECT salvata nel motore
- persistente: non muore alla chiusura della connessione
- posso usare query per interrogarla
- posso aggiornare i dati (con dei limiti)

VISTE

VANTAGGI

- semplificare query complesse
- nascondere dati sensibili
- gestire gli accessi
- campi calcolati appaiono come colonne normali
- compatibilità

VISTE

SVANTAGGI

- se modifico le tabelle, devo aggiornare le viste
- non accetta parametri
- non è compilata
- performance leggermente peggiori (in particolare se uso viste che contengono altre viste)

VISTE

COME FUNZIONANO? [MYSQL]

MERGE (predefinito)

- nella query inserisco il codice contenuto nella vista
- eseguo il codice ottenuto

VISTE

COME FUNZIONANO? [MYSQL]

TEMPTABLE (materialized)

- creo una tabella temporanea in cui salvo il risultato
- obbligatoria se la vista non ha una relazione biunivoca con la tabella sottostante:
  - funzioni di aggregazione, group by, having
  - distinct, limit
  - union
  - subquery nella lista degli attributi

VISTE

DEFINIZIONE DI UNA VISTA

```
CREATE VIEW nomeVista AS
    SELECT ...
```

Dettagli:

- la SELECT può essere eseguita da sola
- posso usare subquery nella clausola WHERE
- non posso usarle nella clausola FROM

VISTE

CREARE UNO SCHEMA ESTERNO “VIEWLISTINOCLIENTI” CHE MOSTRI IL LISTINO PER I CLIENTI (CODICE E NOME PRODOTTO, MSRP), QUINDI MOSTRARNE I DATI

```
CREATE VIEW viewListinoClienti AS
    SELECT productCode, productName, MSRP
    FROM products;
    SELECT * from viewListinoClienti;
```

VISTE

CREARE UNO SCHEMA ESTERNO “VIEWTOTALEORDINI” CHE MOSTRI IL NUMERO DELL’ORDINE E L’IMPORTO TOTALE

Hint: `CREATE VIEW nomeVista AS SELECT ...`

```sql
CREATE VIEW viewTotaleOrdini AS
    SELECT orderNumber,
    SUM(quantityOrdered * priceEach) total
    FROM orderdetails
    GROUP by orderNumber
```

VISTE

USANDO “VIEWTOTALEORDINI” MOSTRARE IL TOTALE DELL’ORDINE 10102

```
SELECT total
    FROM viewTotaleOrdini
    WHERE orderNumber = 10102;
```

VISTE

COSA FA UNA VISTA?

```
SHOW CREATE VIEW nomeVista;
```

ELIMINARE UNA VISTA

```
DROP VIEW nomeVista;
```

VISTE

MODIFICARE UNA VISTA

```
ALTER VIEW nomeVista AS nuovaSELECT;
```

VISTE MODIFICABILI

Posso modificare i dati di una vista se la SELECT:

- è riferita ad una sola tabella
- non contiene GROUP BY o HAVING
- non contiene DISTINCT
- non fa riferimento a viste non modificabili
- la selezione non contiene espressioni

VISTE MODIFICABILI

CREARE LA VISTA “OFFICEINFO” MOSTRANDO CODICE UFFICIO, TELEFONO E CITTÀ DEGLI UFFICI. PROVARE A MODIFICARE QUALCHE DATO (ES: N. TEL.)

```
CREATE VIEW officeInfo
    AS SELECT officeCode, phone, city
    FROM offices;
```

```
UPDATE officeInfo
    SET phone = '+39 040 55558555'
    WHERE officeCode = 4;
```

CONTROLLO ACCESSI

CONNESSIONE

- utente e password valide?
- [connessione da client autorizzato]?

CONTROLLO ACCESSI

RICHIESTA

- l’utente connesso può fare questa operazione?
- può accedere a questo DB?
- può accedete a questa tabella?
- può accedete a questo attributo?
- può eseguire questa procedura?

AGGIUNGERE UTENTI

DIPENDE DAL MOTORE

MySQL:

```
CREATE USER nome@host
IDENTIFIED BY 'password'
```

Oracle:

```
CREATE USER nome
IDENTIFIED BY password
```

SQL Server

```
CREATE USER nome
WITH PASSWORD = 'password'
```

AGGIUNGERE UTENTI

DETTAGLI PER MYSQL:

- posso usare wildards nell'host se le racchiudo tra apici - `'%'` per ogni host
- `'nome@host'` crea un utente con username `nome@host` legato all'host `%`
- `FLUSH PRIVILEGES` forza il reload dei dati

AGGIUNGERE UTENTI

CREARE L'UTENTE "PIPPO" CON PASSWORD "PLUTO" CHE POSSA CONNETTERSI SOLO DAL VOSTRO COMPUTER

```
CREATE USER pippo@localhost
IDENTIFIED BY 'pluto';
```

CONTROLLO ACCESSI

12 REGOLE DI CODD

1. INFORMAZIONI: tutte le informazioni in un DBR sono rappresentate esplicitamente da valori in tabelle (DEFINIZIONE)

CONTROLLO ACCESSI - MYSQL

CONNESSIONE

- Utente e password valide?
- Connessione da un client autorizzato?

DATABASE `mysql`

- Tabella `user`

```
INSERT INTO user(host,user,passwo
VALUES('localhost','pippo',
PASSWORD('pluto'));
FLUSH PRIVILEGES;
```

MODIFICARE GLI UTENTI

CAMBIARE LA PASSWORD

```
SET PASSWORD FOR user@host =
PASSWORD('Secret1970');
```

ELIMINARE UN UTENTE

```
DROP USER user@host;
```

ASSEGNARE I PERMESSI

UN UTENTE APPENA CREATO NON PUÒ FARE NULLA

```
GRANT privilegio (colonne)
ON risorsa
TO account
[WITH GRANT OPTION]
```

- `privilegio`: tipo di operazione permessa
- `colonne`: se si applica solo ad alcune colonne
- `risorsa`: database.tabella — wildcard: *
- `account`: utente@host
- `WITH GRANT OPTION`: l'utente può propagare i permessi ad altri

PRIVILEGI

- `ALL`: tutti
- `ALTER`: modificare tabella
- `CREATE`: creare oggetti
- `DELETE`: eliminare ennuple
- `SELECT`: leggere i dati
- `UPDATE`: modificare i dati
- … e tanti altri

PRIVILEGI

PERMETTERE ALL'UTENTE PIPPO DI LEGGERE, MODIFICARE E CANCELLARE DATI AL DB DEI MODELLINI.

Hint: `GRANT privilegio (colonne) ON risorsa TO account`

```
GRANT SELECT, UPDATE, DELETE ON
classicmodels.* TO 'pippo'@'%';
```

PRIVILEGI

CREARE UN ALTRO AMMINISTRATORE

```
GRANT ALL ON *.* TO 'super'@'localhost'
WITH GRANT OPTION;
```

PERMETTERE AD UN UTENTE DI LEGGERE E MODIFICARE I NUMERI DI TELEFONO DEI CLIENTI E DI VEDERNE I NOMI

```
GRANT SELECT (phone, customerName),
UPDATE (phone)
ON classicmodels.customers
TO 'someuser'@'somehost';
```

VISUALIZZARE I PERMESSI

POSSO VEDERE I PRIVILEGI DI OGNI UTENTE

```
SHOW GRANTS FOR utente;
```

REVOCARE I PERMESSI

Sintassi molto simile a GRANT

```
REVOKE privilege_type [(column_list)]
[, priv_type [(column_list)]]...
ON [object_type] privilege_level
FROM user [, user]...
```

REVOCARE I PERMESSI

```
REVOKE UPDATE, DELETE ON
classicmodels.* FROM
'rfc'@'localhost';
```

TRANSAZIONI

INSIEME DI OPERAZIONI DA CONSIDERARE INDIVISIBILE (“ATOMICO”), CORRETTO ANCHE IN PRESENZA DI CONCORRENZA E CON EFFETTI DEFINITIVI

ACID

PROPRIETÀ TRANSAZIONI:

- Atomicità
- Consistenza
- Isolamento
- Durabilità (persistenza)

ATOMICITÀ

LA SEQUENZA DI OPERAZIONI SULLA BASE DI DATI VIENE ESEGUITA PER INTERO O PER NIENTE.

Esempio: trasferimento di fondi da un conto A ad un conto B: o si fanno il prelevamento da A e il versamento su B o nessuno dei due

CONSISTENZA

AL TERMINE DELL'ESECUZIONE DI UNA TRANSAZIONE, I VINCOLI DI INTEGRITÀ DEBBONO ESSERE SODDISFATTI

DURANTE L'ESECUZIONE CI POSSONO ESSERE VIOLAZIONI, MA SE RESTANO ALLA FINE ALLORA LA TRANSAZIONE DEVE ESSERE ANNULLATA PER INTERO ("ABORTITA")

ISOLAMENTO

L’EFFETTO DI TRANSAZIONI CONCORRENTI DEVE ESSERE COERENTE (AD ESEMPIO EQUIVALENTE ALL’ESECUZIONE SEPARATA)

Esempio: se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente si deve evitare di trascurarne uno

DURABILITÀ

LA CONCLUSIONE POSITIVA DI UNA TRANSAZIONE CORRISPONDE AD UN IMPEGNO (“COMMIT”) A MANTENERE TRACCIA DEL RISULTATO IN MODO DEFINITIVO, ANCHE IN PRESENZA DI GUASTI E DI ESECUZIONE CONCORRENTE

TRANSAZIONI

LA SINTASSI DIPENDE DAL MOTORE

In MySQL:

- `START TRANSACTION`: specifica l'inizio della transazione (le operazioni non vengono eseguite sulla base di dati)
- `COMMIT`: le operazioni specificate a partire dal begin transaction vengono eseguite
- `ROLLBACK`: si rinuncia all'esecuzione delle operazioni specificate dopo l'ultimo begin transaction

TRANSAZIONI

SQL Server

- `BEGIN TRANSACTION`
- `COMMIT WORK`
- `ROLLBACK WORK`

ESEMPIO TRANSAZIONE

```
start transaction;

select @orderNumber := max(orderNUmber) from orders;
set @orderNumber = @orderNumber + 1;

insert into orders(orderNumber, orderDate, requiredDate, shipp
values(@orderNumber, now(), date_add(now(), INTERVAL 5 DAY),
date_add(now(), INTERVAL 2 DAY), 'In Process', 145);

insert into orderdetails(orderNumber, productCode, quantityOrd
values(@orderNumber,'S18_1749', 30, '136', 1),
(@orderNumber,'S18_2248', 50, '55.09', 2);

commit;
```

TRANSAZIONI IN UN DBMS

DUE MODULI FONDAMENTALI:

- gestore della concorrenza
    - garantisce isolamento e consistenza
    - scheduler delle operazioni
- gestore dell’affidabilità
    - garantisce atomicità e durevolezza
    - consentire il recupero in caso di guasti

GESTORE DELL'AFFIDABILITÀ

IDEE DI BASE:

- registrare tutte le azioni eseguite in un file di registro ("log")
- se si rompe qualcosa durante la transazione, so come tornare indietro

ATTENZIONE:

Qualcosa è sempre in memoria e può sempre essere perso

LOG DELLE TRANSAZIONI

COME SALVO LE TRANSAZIONI?

- Write Ahead Logging:
  - il log contiene i blocchi modificati
  - commit = copiare i dati dal log al file del DB
    - scelto da quasi tutti i motori
- Command Logging:
  - il log contiene lo storico delle istruzioni
  - commit = eseguire realmente le operazioni

REDO LOGGING

SOLUZIONE DI MYSQL (WRITE AHEAD)

- salvo i dati in un log che risiede in memoria
- sposto piccole porzioni in un log su disco
- ogni tanto unisco il log su disco ai dati reali

SQL

PROGRAMMING LANGUAGE (PL)

OGGETTI PROGRAMMABILI IN SQL

È possibile racchiudere comandi SQL in oggetti programmabili la cui definizione rimane nelle tabelle di sistema:

- facilitano il riutilizzo del piano di esecuzione (precompilate)
- incapsulano la logica applicativa di accesso ai dati
- nascondono al client la complessità della base dati

OGGETTI PROGRAMMABILI IN SQL

VISTE

STORED PROCEDURE

insieme di comandi SQL con parametri di input e output che possono restituire recordset

OGGETTI PROGRAMMABILI IN SQL

TRIGGER

particolari stored procedure che vengono associate ad una operazione su un oggetto e invocate automaticamente

USER DEFINED FUNCTION

consentono di raggruppare e riutilizzare codice SQL solitamente ripetuto all'interno di SP e trigger

ASSEMBLIES SQLCLR

semplificando: stored procedure scritte in linguaggi

STORED PROCEDURE

SUBROUTINE CHE CONTENGONO TUTTO IL CODICE NECESSARIO PER EFFETTUARE UNA OPERAZIONE

- incapsulano task ripetitivi
- codice ammesso: tutto (DDL, DML, TSQL,…)
- accettano parametri di input
- producono zero, uno o più output
    - parametri di output
    - resultset

STORED PROCEDURE

FONDAMENTALI PER INCAPSULARE LA LOGICA DI

- accesso alle tabelle
- manipolazione dei dati

STORED PROCEDURE

PERMETTONO LA CREAZIONE DI UN LIVELLO DI ASTRAZIONE DEL MODELLO FISICO DEL DATABASE

- aiutano a mantenere un alto disaccoppiamento
- garantiscono la possibilità di intervenire sul database senza necessariamente modificare le applicazioni che lo usano
  - introduzione di nuove funzionalità necessarie ad altre applicazioni (un db non è privato)
  - miglioramento performance

STORED PROCEDURE: VANTAGGI

- mascherano lo schema logico del DB
- riutilizzo del codice
- permettono l'implementazione di arbitrari meccanismi di sicurezza
- migliorano le performance (cached execution plans)
    - MySql: ricompilata per ogni connessione
- riducono il traffico di rete

STORED PROCEDURE: SVANTAGGI

- aumentano il carico (CPU, memoria) del DBMS
- difficile farne il debug
  - MySql: non possibile
- sintassi particolare
- non sono transazionali di per se
  - …ma possono diventarlo usanto T-SQL

CREARE STORED PROCEDURE

Anche qui dipende dal motore…

```
CREATE PROCEDURE nome()
BEGIN
... codice
END
```

MS SQL Server:

```
CREATE PROCEDURE nome
AS [BEGIN]
... codice
[END]
```

CREARE STORED PROCEDURE

CREARE UNA SP CHE PRENDA TUTTI I DATI DEI PRODOTTI

```
CREATE PROCEDURE sp_getAllProducts()
BEGIN
    SELECT * FROM products;
END
```

Problema: potrei avere più istruzioni, e non voglio che il DBMS le interpreti una alla volta (ogni volta che trova un “;”)

CREARE STORED PROCEDURE

MySql: cambio il delimitatore

```
DELIMITER $$
CREATE PROCEDURE nome()
BEGIN
... codice1;
... codice2;
END $$
DELIMITER ;
```

CREARE STORED PROCEDURE

SQL Server: GO!

```
CREATE PROCEDURE nome
AS [BEGIN]
... codice1;
... codice2;
[END]
GO
```

ESEGUIRE STORED PROCEDURE

DIPENDE DAL MOTORE

MySql

```
CALL nomeStoredProcedure()
```

SQL Server

```
EXEC nomeStoredProcedure
```

PostgreSQL

```
SELECT nomeStoredProcedure()
```

Oracle

```
EXECUTE nomeStoredProcedure()
```

STORED PROCEDURE

CREARE UNA SP CHE MOSTRI I DATI DI TUTTI I DIPENDENTI CHE SIANO "SALESREP"

Hint: `CREATE PROCEDURE nome() BEGIN ... codice END`

```
DELIMITER $$
CREATE PROCEDURE sp_getSalesRep()
BEGIN
SELECT * FROM employees
WHERE jobTitle = 'Sales Rep';
END
```

VISUALIZZARE LE STORED PROCEDURE

PER VEDERE TUTTE LE SP NEL MOTORE:

```
SHOW PROCEDURE STATUS
[WHERE condizioni]
```

Condizioni:

- `db`: nome del DB
- `name`: nome della SP
- posso usare uguaglianza, `LIKE`, `OR`, `AND`, ...

VISUALIZZARE ED ELIMINARE SP

PER VEDERE IL CODICE DI UNA SP:

```
SHOW CREATE PROCEDURE spNome
```

PER ELIMINARE UNA SP:

```
DROP PROCEDURE spNome
```

MODIFICARE STORED PROCEDURE

DIPENDE DAL MOTORE.

MySql

```
DROP + CREATE
```

SQL Server

```
ALTER nomeStoredProcedure
AS
...codice
GO
```

PARAMETRI

UNA STORED PROCEDURE PUÒ RICEVERE DEI PARAMETRI:

```
CREATE PROCEDURE nomeSP(
    nomePar1 tipoPar1,
    nomePar2 tipoPar2, ...
)
BEGIN
... codice
END
CALL nomeSP(par1, par2,...)
```

PARAMETRI

CREARE UNA SP CHE MOSTRI I DATI DI TUTTI I DIPENDENTI CHE SIANO DELLA TIPOLOGIA PASSATA COME PARAMETRO (PRINCIPAL, SALES REP, ECC.)

```
CREATE PROCEDURE sp_getEmployeeByType(
tipoImp varchar(50))
BEGIN
    SELECT * FROM employees WHERE jobTitle = tipoImp;
END
```

PARAMETRI

I PARAMETRI SONO PASSATI:

- in sola lettura (solo input) → `IN`
    - è l'opzione di default
- in sola scrittura (solo output) → `OUT`
    - durante l'esecuzione uso variabili
- leggibili e scrivibili (bidirezionali) → `INOUT`

```
CREATE PROCEDURE nomeSP(
    direzione nomePar1 tipoPar1,
    direzione nomePar2 tipoPar2,...)
BEGIN ... codice END
```

PARAMETRI

CREARE UNA SP CHE FUNGA DA CONTATORE

```
CREATE PROCEDURE sp_conta(
    INOUT count INT(4),
    IN inc INT(4))
BEGIN
    SET count = count + inc;
END
```

VERIFICA

```
SET @counter = 1;
CALL sp_conta(@counter,1); -- 2
CALL sp_conta(@counter,1); -- 3
CALL sp_conta(@counter,5); -- 8
SELECT @counter; -- 8
```

PARAMETRI

CREARE UNA SP CHE RADDOPPI IL VALORE PASSATO

```
CREATE PROCEDURE sp_raddoppia(
    INOUT valore int(11))
BEGIN
    SET valore = valore * 2;
END
```

VERIFICA

```
set @val = 10;
select @val;
CALL sp_raddoppia(@val);
select @val;
```

PARAMETRI

CREARE UNA SP CHE PRENDA IN INPUT IL NUMERO D'ORDINE E RITORNI IL NUMERO DI OGGETTI COMPRATI

```
CREATE PROCEDURE sp_contaOggettiInOrdine(
    IN oNumber INT,
    OUT numberObjects INT)
BEGIN
    SET numberObjects = (
        SELECT sum(quantityOrdered) FROM
        orderdetails WHERE orderNumber = oNumber
    );
END
```

VERIFICA

```
CALL sp_contaOggettiInOrdine(10100, @numObj);
select @numObj; --- 151
```

SELECT INTO

SE DEVO SCRIVERE DIRETTAMENTE IL RISULTATO DI UNA QUERY IN UNA VARIABILE

```
CREATE PROCEDURE sp_contaOggettiInOrdine(
    IN oNumber INT,
    OUT numberObjects INT)
BEGIN
    SELECT sum(quantityOrdered)
    INTO numberObjects
    FROM orderdetails
    WHERE orderNumber = oNumber);
END
```

SELECT INTO

CREARE UNA SP CHE RICEVA IN INPUT LO STATUS DELL'ORDINE E DICA QUANTI ORDINI VI SONO

```
CREATE PROCEDURE sp_contaOrdini(
    IN orderStatus VARCHAR(25),
    OUT total INT)
BEGIN
    SELECT count(orderNumber)
    INTO total FROM orders
    WHERE status = orderStatus;
END
```

VERIFICA

```
CALL sp_contaOrdini('Shipped',@total);
SELECT @total; --- 303
```

PIÙ ISTRUZIONI

CREARE UNA SP CHE RICEVA IN INPUT IL CODICE CLIENTE E RESTITUISCA TANTI VALORI:

- numero ordini spediti (status = Shipped)
- numero ordini cancellati (Canceled)
- numero ordini risolti (Resolved)
- numero ordini confutati (Disputed)

Provate con il cliente 141:

```
CALL sp_getOrderByCust(141,@shipped,@canceled,
@resolved,@disputed);
SELECT @shipped,@canceled,@resolved,@disputed;
```

PIÙ ISTRUZIONI

```
DELIMITER $$
CREATE PROCEDURE get_order_by_cust(IN cust_no INT,
OUT shipped INT, OUT canceled INT,
OUT resolved INT, OUT disputed INT)
BEGIN
-- shipped
SELECT count(*) INTO shipped FROM orders
WHERE customerNumber = cust_no AND status = 'Shipped';
-- canceled
SELECT count(*) INTO canceled FROM orders
WHERE customerNumber = cust_no AND status = 'Canceled';
-- resolved
SELECT count(*) INTO resolved FROM orders
WHERE customerNumber = cust_no AND status = 'Resolved';
-- Disputed
```

VARIABILI

MYSQL

LE VARIABILI HANNO TRE LIVELLI DI VISIBILITÀ:

- globali (@@): tutti le vedono
- connessione (@): connessione vede le proprie
  - `SET @variabile = 10`
- locali (senza @): nascono e muoiono nella SP
  - `DECLARE nomeVariabile tipoVariabile`
  - `SET nomeVariabile = valore`

CONDIZIONI

NELLE STORED PROCEDURE POSSO INSERIRE IF:

```
IF espressione THEN
    comandi
ELSEIF espressione THEN
    comandi
ELSE
    comandi
END IF;
```

CONDIZIONI

CREARE UNA SP CHE PRENDA IN INPUT IL CODICE CLIENTE E RESTITUISCA UNA STRINGA CHE VALE:

- PLATINUM se il credito è > 50.000
- GOLD se il credito è > 10.000 e <= 50.000
- SILVER altrimenti

Provate con il cliente 103:

```
CALL sp_getCustomerLevel(103, @livello);
SELECT @livello;
```

CONDIZIONI

```
DELIMITER $$
CREATE PROCEDURE sp_getCustomerLevel(
    IN custNo int(11),
    OUT customerLevel varchar(10))
BEGIN
    DECLARE creditlim double;
    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = custNo;
    IF creditlim > 50000 THEN
        SET customerLevel = 'PLATINUM';
    ELSEIF creditlim >= 10000 THEN
        SET customerLevel = 'GOLD';
    ELSE
        SET customerLevel = 'SILVER';
```

CICLI

POSSO RIPETERE PIÙ VOLTE LA STESSA OPERAZIONE:

```
WHILE espressione DO
comandi
END WHILE;
```

```
REPEAT
comandi
UNTIL espressione
END REPEAT;
```

Dettagli:

- `LEAVE`: esce dal ciclo
- `ITERATE`: procede con l'iterazione successiva

CICLI

SP CHE CALCOLI LA SERIE DI FIBONACCI

```
DELIMITER $$
CREATE PROCEDURE sp_fibonacci(IN n int, OUT out_fib int)
BEGIN
    DECLARE m INT default 0;
    DECLARE k INT default 1;
    DECLARE i INT default 1;
    DECLARE tmp INT;
    WHILE (i<=n) DO
        set tmp = m+k;
        set m = k;
        set k = tmp;
        set i = i+1;
    END WHILE;
    SET out_fib = m;
END;
```

GESTIRE GLI ERRORI

VOGLIO GESTIRE GLI ERRORI NELLA SP

```
DECLARE azione HANDLER FOR
    condizione [BEGIN] codice [END]
```

- `condizione`: cosa vogliamo intercettare
- `codice`: cosa fare
- `azione`: come comportarsi dopo aver eseguito il codice
    - `CONTINUE` → continua con il resto
    - `EXIT` → termina l'esecuzione

GESTIRE GLI ERRORI

```
DECLARE azione HANDLER FOR
    condizione [BEGIN] codice [END]
```

Condizioni:

- codice errore MySql
    - Es: 1062, chiave duplicata
- SQLSTATE ‘codiceNumerico’
    - Es: 22012, divisione per zero

GESTIRE GLI ERRORI

SQLSTATE

- `SQLWARNING`: scorciatoria per `SQLSTATE` che iniziano con `01`
- `NOT FOUND`: scorciatoria per `SQLSTATE` che iniziano con `02`
- `SQLEXCEPTION`: scorciatoria per `SQLSTATE` che non iniziano con `00`, `01` o `02`

GESTIRE GLI ERRORI

IN CASO DI ERRORE, ANNULLARE LA TRANSAZIONE E DARE UN MESSAGGIO

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Errore: ho annullato tutto!';
    END;
```

SEGNALARE ERRORI

POSSO LANCIARE MESSAGGI DI ERRORE

```
SIGNAL SQLSTATE 'codice'
SET MESSAGE_TEXT = 'testo'
```

Codice definito dall’utente: 45000

SEGNALARE ERRORI

SCRIVERE UNA SP CHE RITORNI IL NUMERO DI ORDINI DI UN CLIENTE. SE NON ESISTE, DARE UN ERRORE

Hint: `SIGNAL SQLSTATE 'codice' SET MESSAGE_TEXT = 'testo'`

```
CREATE PROCEDURE sp_countOrders(customerNo int)
BEGIN
    DECLARE conteggio INT;
    SELECT count(*) INTO conteggio FROM customers
    WHERE customerNumber = customerNo;
    IF conteggio = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Errore gen
    END IF;
    SELECT count(*) FROM orders WHERE customerNumber = custome
END
```

CURSORI

PERMETTONO DI PROCESSARE SINGOLE RIGHE DI UN RESULTSET

- **Read-only**: non posso aggiornare i dati usando il cursore
- **Non-scrollable**: posso scorrere il dataset senza cambiarne l’ordinamento
- **Asensitive**: puntano ai dati reali, non ad una copia → rapidi a crearsi, ma modifiche fatte ai dati da altre connessioni si ripercuotono sul cursore

CURSORI

DEFINISCO IL NOME DEL CURSORE E LA QUERY CHE USERÀ

```
DECLARE nomeCursore CURSOR FOR
SELECT ...
```

CURSORI

APRO IL CURSORE, ESEGUENDO LA QUERY

```
OPEN nomeCursore
```

CURSORI

…USO IL CURSORE IN UN CICLO

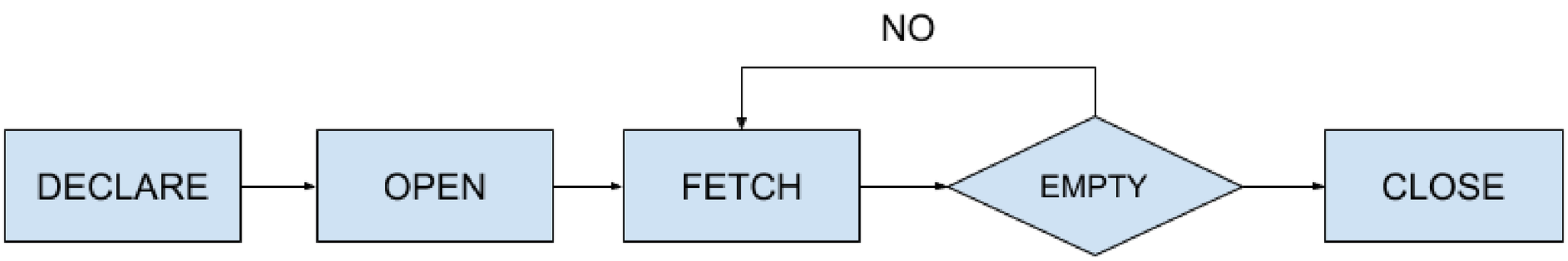
```
FETCH nomeCursore INTO var1, var2, ...
```

CURSORI

CHIUDO IL CURSORE (LIBERO MEMORIA):

```
CLOSE nomeCursore
```

CURSORI



CURSORI

QUANDO SMETTO DI USARE IL CURSORE?

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
```

Uso un handler: quando non ho altri dati, si verifica `NOT FOUND`

Terminerò il ciclo quando `finished == 1`

CURSORI

CREARE UNA SP CHE RITORNI IN UN SINGOLO VALORE TUTTI GLI INDIRIZZI EMAIL DEI DIPENDENTI

Hints:

- `DECLARE nomeCursore CURSOR FOR SELECT ...`
- `OPEN/CLOSE nomeCursore`
- `FETCH nomeCursore INTO var1,var2,...`
- `DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;`

CURSORI

CREARE UNA SP CHE RITORNI IN UN SINGOLO VALORE TUTTI GLI INDIRIZZI EMAIL DEI DIPENDENTI

```
CREATE PROCEDURE sp_buildEmailList (INOUT email_list varchar(4
BEGIN
    DECLARE finished INTEGER DEFAULT 0;
    DECLARE v_email varchar(100) DEFAULT "";

    DECLARE email_cursor CURSOR FOR SELECT email FROM employee

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
    OPEN email_cursor;
    WHILE (finished = 0) DO
        FETCH email_cursor INTO v_email;
        IF finished = 0 THEN
            SET email_list = CONCAT(v_email,";",email_list);
        END IF;
    END WHILE;
```

USER DEFINED FUNCTION

- Scalar Functions
  - Simile ad una built-in function
  - Ritorna un singolo valore costruito con una serie di statements
- Multi-Statement Table-valued Functions
  - Contenuto simile ad una stored procedure
  - Referenziata come una Vista
- In-line Table-valued Functions
  - Simile ad una Vista con parametri
  - Ritorna una tabella come risultato di uno statement SELECT singolo

UDF VS SP

- Risultato
    - SP: restituisce 0 o N valori
    - UDF: restituisce sempre 1 valore
- Parametri
    - SP: input/output
    - UDF: solo input
- Modifiche
    - SP: non può modificare il DB
    - UDF: solo SELECT (Myql, Oracle: tutto, ma meglio evitare)

UDF VS SP

- Chi richiama chi
  - SP: può richiamare UDF
  - UDF: non può richiamare SP
- SELECT
  - SP: non può essere usata in una SELECT
  - UDF: può essere usata in una SELECT
- RecordSet
  - SP: se ritorna una tabella non posso riusarla (no select)
  - UDF: posso usarla come una normale tabella

USER DEFINED FUNCTION

COME DEFINIRE UNA UDF

```
CREATE FUNCTION function_name
(param1 tipo1,param2 tipo2,...)
RETURNS tipo
[NOT] DETERMINISTIC
BEGIN
    statements
END
```

Si usa come una funzione normale (select, ecc)

DETERMINISTIC O NO?

AIUTO IL MOTORE A CAPIRE COME OTTIMIZZARE

- Deterministic: se l’input e lo stato del DB non variano, l’output non varia
- NON Deterministic: l’output può variare anche se l’input non varia
  - now()
  - rand()

DETERMINISTIC: E SE SBAGLIO?

IL MOTORE SI FIDA

- dico `Deterministic` ma non lo è: risultati non corretti (l'execution planner può decidere che non occorre ricalcolare)
- dico `NON Deterministic` ma lo è: prestazioni peggiori (ricalcolo anche se non serve)

USER DEFINED FUNCTION

CREARE UNA UDF CHE RICEVE IN INPUT UN NUMERO E NE RESTITUISCA IL DOPPIO, ED USARLA IN UNA QUERY

Hint: `CREATE FUNCTION function_name (param1 tipo1,param2 tipo2,...) RETURNS tipo [NOT] DETERMINISTIC BEGIN statements END`

```
CREATE FUNCTION udf_raddoppia (numero int)
RETURNS INT DETERMINISTIC
BEGIN
    RETURN numero * 2;
END
```

USER DEFINED FUNCTION

CREARE UNA UDF CHE RICEVA IL CREDITO DEL CLIENTE E RESTITUISCA IL LIVELLO (PLATINUM, ...)

```
CREATE FUNCTION udf_customerLevel
(p_creditLimit double)
RETURNS VARCHAR(10) DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);
    IF p_creditLimit > 50000 THEN
        SET lvl = 'PLATINUM';
    ELSEIF p_creditLimit >= 10000 THEN
        SET lvl = 'GOLD';
    ELSE
        SET lvl = 'SILVER';
    END IF;
    RETURN (lvl);
END
```

USER DEFINED FUNCTION

CREARE UNA UDF CHE RICEVA IL CODICE CLIENTE E RESTITUISCA IL NUMERO DI ORDINI CHE HA FATTO.

```
CREATE FUNCTION udf_contaOrdini(cliente int)
RETURNS INT(11)
BEGIN
    DECLARE conteggio INT;
    SELECT count(*) INTO conteggio FROM orders
    WHERE customerNumber = cliente;
    RETURN conteggio;
END
```

TRIGGER

OPERAZIONI DA ESEGUIRE QUANDO SI VERIFICA UN CERTO EVENTO

- non viene richiamata direttamente
- parte automaticamente quando si effettua una operazione su una certa tabella (insert, delete, update)
- possono essere legati ad eventi temporali
- limiti MySQL: non possono usare UDF, SP, prepared statements

TRIGGER

VANTAGGI

- ulteriore controllo dell'integrità dei dati
  - controlli non possibili per limiti del motore (es: CHECK in MySql)
  - controlli aggiuntivi sulla logica del programma
- molto comodi per l'audit (registrazione) delle modifiche

TRIGGER

CREAZIONE DI UN TRIGGER

```
CREATE TRIGGER nome quando
ON nomeTabella
FOR EACH ROW
BEGIN
    codice
END
```

- ogni trigger ha un nome
- ogni trigger è riferito ad una tabella

TRIGGER: QUANDO

TRALASCIAMO I TRIGGER TEMPORALI

Che operazione controlliamo?

- `INSERT`, `UPDATE` o `DELETE`

Quando devo eseguire il trigger?

- `BEFORE`
  - es: i dati sono coretti?
- `AFTER`
  - registro chi ha modificato i dati, ricalcolo valori

TRIGGER: GRANULARITÀ

STATEMENT LEVEL (DEFAULT SQL SERVER)

- il trigger viene eseguito una volta sola per ogni comando che lo ha attivato, indipendentemente dal numero di tuple modificate
- è il modo più vicino all'approccio tradizionale dei comandi SQL, che sono di norma set-oriented

TRIGGER: GRANULARITÀ

ROW LEVEL

- FOR EACH ROW, unico per MySQL
- il trigger viene eseguito una volta per ciascuna tupla che è stata modificata dal comando
- consente di scrivere i trigger in modo più semplice
- può essere meno efficiente

TRIGGER: OLD & NEW

PERMETTONO DI DISTINGUERE IL RECORD PRIMA E DOPO LA MODIFICA

- OLD: valore precedente
  - usabile nel DELETE
  - usabile nel BEFORE UPDATE
- NEW: valore dopo le modifiche
  - usabile nell'INSERT
  - usabile nel BEFORE UPDATE

Es: `OLD.contactLastName`

CREIAMO UNA TABELLA DI AUDIT

```
CREATE TABLE employees_audit (
    id int(11) NOT NULL AUTO_INCREMENT,
    employeeNumber int(11) NOT NULL,
    lastname varchar(50) NOT NULL,
    changedon datetime DEFAULT NULL,
    changedBy varchar(50) DEFAULT NULL,
    action varchar(50) DEFAULT NULL,
    PRIMARY KEY (id)
)
```

TRIGGER CHE REGISTRA LE MODIFICHE

```
DELIMITER $$
CREATE TRIGGER trg_beforeUpdateEmployees
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employees_audit
    SET action = 'update',
    employeeNumber = OLD.employeeNumber,
    lastname = OLD.lastname,
    changedon = NOW(),
    changedby = user();
END$$
DELIMITER ;
```

TRIGGER CHE REGISTRA LE MODIFICHE

```sql
UPDATE employees
SET lastName = 'Phan'
WHERE employeeNumber = 1056
```

TRIGGER PER IL CONTROLLO DEI DATI

SE QUALCOSA NON VA SI SEGNALA UN ERRORE

La sintassi è la stessa vista nelle SP

```
SIGNAL sqlstate '45001' SET message_text = "No way !";
```

TRIGGER PER IL CONTROLLO DEI DATI

CREARE UN TRIGGER CHE VERIFICHI NON VENGA INCREMENTATO IL LIMITE DI CREDITO

Hint: `IF NEW.creditLimit > OLD.creditLimit`

```
CREATE TRIGGER trg_beforeUpdateCustomer
BEFORE UPDATE ON customers
FOR EACH ROW BEGIN
    IF NEW.creditLimit > OLD.creditLimit THEN
        SIGNAL sqlstate '45001' SET message_text = "Basta cred
    END IF;
END
```

CONFLITTI TRA TRIGGER

SE VI SONO PIÙ TRIGGER ASSOCIATI ALLO STESSO EVENTO, SQL:1999 PRESCRIVE

- vengono eseguiti i trigger BEFORE statement-level
- vengono eseguiti i trigger BEFORE row-level
- si applica la modifica e si verificano i vincoli di integrità definiti sulla base di dati
- vengono eseguiti i trigger AFTER row-level
- vengono eseguiti i trigger AFTER statement-level

CONFLITTI TRA TRIGGER

SE VI SONO PIÙ TRIGGER DELLA STESSA CATEGORIA, L'ORDINE DI ESECUZIONE VIENE SCELTO DAL SISTEMA IN UN MODO CHE DIPENDE DALL'IMPLEMENTAZIONE

MODELLO DI ESECUZIONE

- SQL:1999 prevede che i trigger vengano gestiti in un Trigger Execution Context (TEC)
- l'esecuzione dell'azione di un trigger può produrre eventi che fanno scattare altri trigger, che dovranno essere valutati in un nuovo TEC interno
- in ogni istante possono esserci più TEC per una transazione, uno dentro l'altro, ma uno solo può essere attivo

MODELLO DI ESECUZIONE

- per i trigger row-level il TEC tiene conto di quali tuple sono già state considerate e quali sono da considerare
- si ha quindi una struttura a stack
  - TEC0 -> TEC1 -> … -> TECn
- quando un trigger ha considerato tutti gli eventi, il TEC si chiude e si passa al trigger successivo

INTERAZIONE TRA TRIGGER

Dipartimento

NroDip	MatricolaMGR
1	50

Progetto

NroProg	Obiettivo
10	NO
20	NO

Impiegato

Matricola	Nome	Salario	NDip	NProg
50	Rossi	59.000	1	20
51	Verdi	56.000	1	10
52	Bianchi	50.000	1	20

TRIGGER T1: BONUS

Evento: update di Obiettivo in Progetto

Condizione: Obiettivo = ‘SI’

Azione: incrementa del 10% il salario degli impiegati coinvolti

TRIGGER T1: BONUS

```
CREATE TRIGGER Bonus
AFTER UPDATE OF Obiettivo ON Progetto
FOR EACH ROW
WHEN NEW.Obiettivo = 'SI'
BEGIN
    update Impiegato
    set Salario = Salario*1.10
    where NProg = NEW.NroProg;
END;
```

TRIGGER T2: CONTROLLAINCREMENTO

Evento: update di Salario in Impiegato

Condizione: nuovo salario maggiore di quello del manager

Azione: decrementa il salario rendendolo uguale a quello del manager

TRIGGER T2: CONTROLLAINCREMENTO

```
CREATE TRIGGER ControllaIncremento
AFTER UPDATE OF Salario ON Impiegato
FOR EACH ROW
DECLARE X number;
BEGIN
    SELECT Salario into X FROM Impiegato JOIN Dipartimento
    ON Impiegato.Matricola = Dipartimento.MatricolaMGR
    WHERE Dipartimento.NroDip= NEW.NDip;
    IF NEW.Salario > X
        update Impiegato set Salario = X
        where Matricola = NEW.Matricola;
    ENDIF;
END;
```

TRIGGER T3: CONTROLLADECREMENTO

Evento: update di Salario in Impiegato

Condizione: decremento maggiore del 3%

Azione: decrementa il salario del solo 3%

TRIGGER T3: CONTROLLADECREMENTO

```
CREATE TRIGGER ControllaDecremento
AFTER UPDATE OF Salario ON Impiegato
FOR EACH ROW
WHEN (NEW.Salario < OLD.Salario * 0.97)
BEGIN
    update Impiegato
    set Salario=OLD.Salario*0.97
    where Matricola = NEW.Matricola;
END;
```

ATTIVAZIONE DI T1

```
UPDATE Progetto SET Obiettivo = 'SI' WHERE NroProg = 10
```

Evento: update dell'attributo Obiettivo in Progresso

Condizione: vera

Azione: si incrementa del 10% il salario di Verdi

Progetto

NroProg	Obiettivo
10	SI
20	NO

Impiegato

Matricola	Nome	Salario	NDip	NProg
50	Rossi	59.000	1	20
51	Verdi	61.600	1	10
52	Bianchi	50.000	1	20

ATTIVAZIONE DI T2

Evento: update di Salario in Impiegato

Condizione: vera (il salario dell'impiegato Verdi supera quello del manager Rossi)

Azione: si modifica il salario di Verdi rendendolo uguale a quello del manager Rossi

Impiegato

Matricola	Nome	Salario	NDip	NProg
50	Rossi	59.000	1	20
51	Verdi	59.000	1	10
52	Bianchi	50.000	1	20

ATTIVAZIONE DI T3

Evento: update dell'attributo salario in Impiegato

Condizione: vera (il salario di Verdi è stato decrementato per più del 3%)

Azione: si decrementa il salario di Verdi del solo 3%

Impiegato

Matricola	Nome	Salario	NDip	NProg
50	Rossi	59.000	1	20
51	Verdi	59.752	1	10
52	Bianchi	50.000	1	20

- Si attiva nuovamente T3 - condizione è falsa
- Si attiva T2 - condizione vera

ATTIVAZIONE DI T2

Impiegato

Matricola	Nome	Salario	NDip	NProg
50	Rossi	59.000	1	20
51	Verdi	59.000	1	10
52	Bianchi	50.000	1	20

- Si attiva nuovamente T3 - condizione è falsa
- L'attivazione dei trigger ha raggiunto lo stato di terminazione

TECNICHE DI ACCESSO AI DATI

ACCEDERE AI DATI

DATA CONSUMER

Tool e linguaggi che lavorano con i dati

DATA PROVIDERS

Sorgente dei dati

BREVE STORIA DELL'ACCESSO AI DATI

- API proprietarie (es: VB Objects)
- Data Access Objects (DAO/Jet)
- Open Database Connectivity (ODBC)
- OLE for Databases (OLE/DB)
- ActiveX Data Objects (ADO)
- .NET→ ADO.NET
- Object-relational Mapping (ORM)

SISTEMI PROPRIETARI

- dipendono da Data Consumer
  - linguaggio e piattaforma
- dipendono da Data Provider
- …a volte ancora necessari
  - es: accedere ad un MDB in linux

ODBC

- sviluppato da Microsoft e ceduto al W3C
- pensato per motori relazionali
- indipendente da DBMS e sistema operativo

ODBC

COMPONENTI PRINCIPALI

- ODBC Driver: layer tra l'applicazione ed il DBMS
- l'applicazione usa un Driver Manager per accedere ai vari driver
- Data Source Names (DNS): informazioni per la connessione; gestiti dal Driver Manager

ODBC

PRO:

- non intrusivo sul server; sono i Driver ad avere un’interfaccia verso i Data Provider

CONTRO:

- non sempre esiste il driver (gratis)…
- API difficili da usare
- richiedono molto codice da implementare nell’applicazione

ACTIVEX DATA OBJECTS (ADO)

- interfaccia più user-friendly per il programmatore
- parte di Microsoft Data Access Components (MDAC)
- qualsiasi fonte ODBC o OLE/DB
- accessibile da diversi linguaggi (C++, Java, .NET, …)

ADO CONNECTION STRING

CI SONO TANTI POSSIBILI PROVIDER

- serve la password?
- devo dare un nome di file?
- l’indirizzo di un server?

ADO CONNECTION STRING

STRINGA DI CONNESSIONE:

- serie di coppie chiave-valore
- separatore: “;”
- chiave=valore

ADO CONNECTION STRING

ODBC

DSN=PropDB;Uid=admin;Pwd=;

ACCESS

Provider='Microsoft.JET.OLEDB.4.0';Data Source='C:\test.mdb'

ADO CONNECTION STRING

SQL Server

Server=myServerAddress;Database=myDB;UserId=myUsername;Password=myPassw

MYSQL

Server=myServerAddress;Database=myDB;Uid=myUsername;Pwd=myPassword;

http://www.connectionstrings.com/

ADO .NET

EVOLUZIONE DI ADO

ADO .NET è una collezione di classi, interfacce, strutture e tipi che gestiscono l’accesso ai dati da fonti relazionali all’ interno del .NET Framework

ADO VS ADO .NET

ADO

- progettato per accessi connessi
- totalmente legato al modello fisico dei dati
- `RecordSet` è il contenitore dei dati
- `RecordSet` è una tabella che contiene tutti i dati
- se si vuole estrarre dati da più di una tabella: JOIN
- i dati sono “flattened”: si perdono le relazioni, la navigazione è sequenziale

ADO VS ADO .NET

ADO .NET

- progettato per accessi disconnessi
- può modellare i dati a livello logico
- il DataSet rimpiazza il RecordSet
- DataSet contiene tabelle multiple
- estrarre dati da più di una tabella non richiede una JOIN
- Relationships conservate e la navigazione è relazionale

CLASSI PRINCIPALI DI ADO .NET

Connection

- usato per parlare al DB
- le proprietà includono dataSource, username e password

Command

- Uno statement SQL o Stored Procedure

CLASSI PRINCIPALI DI ADO .NET

DataReader

- sola lettura monodirezionale, ma molto veloce
- connesso
- una vista unidirezionale connessa dei dati (simile ad ADO Recordset)

DataAdapter

- lettura e scrittura
- disconnesso
- gestisce DataSet

JDBC

EQUIVALENTE JAVA DI ADO .NET

Esistono 4 tipologie diverse di JDBC

TIPO 1: JDBC-ODBC BRIDGE

- il driver dipende dal SO (chiamate ad ODBC)
- PRO: se ho il driver ODBC posso accedere al DB
- CONTRO: overhead, ed il driver ODBC deve essere installato sul client
- fornito in Java: sun.jdbc.odbc.JdbcOdbcDriver

TIPO 2: NATIVE-API DRIVER

- comunica con le API del DB (dipende dal SO)
- PRO: più veloce del Tipo 1
- CONTRO: le librerie client del DB vanno installate sulle macchine

TIPO 3: NETWORK-PROTOCOL DRIVER

- MiddleWare Driver
- comunica con un application server (es: J2EE) che converte le chiamate nel linguaggio del DB

TIPO 4: DATABASE-PROTOCOL DRIVER

- Pure Java Driver
- PRO: Platform-independent
- CONTRO: ogni DBMS deve scrivere il suo

ADO .NET -> JDBC

SqlConnection -> Connection

SqlCommand -> Statement

SqlDataReader -> ResultSet

CARICARE IL DRIVER

```
// The newInstance() call is a work around for some
// broken Java implementations
Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
```

CONNETTERSI AL DB

```
con = DriverManager.getConnection("jdbc:mysql://localhost/" +
                                  "DBNAME?" +
                                  "user=XXX&" +
                                  "password=YYY");
```

ESEGUIRE UNA QUERY

```
Statement stmt = null;
ResultSet rs = null;

try {
    stmt = con.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");
}
catch (SQLException ex){
    // handle any errors
}
finally {
    // some controls...
    rs.close();
    stmt.close();
}
```

LEGGERE I RISULTATI

```
ResultSet rs = stm.executeQuery("select * from persone");
while (rs.next()) {
    String col1 = rs.getString("colonna1");
}
```

MODIFICARE I DATI

```
stm.executeUpdate("UPDATE tabella SET colonna = val ...");
stm.executeUpdate("INSERT ...");
stm.executeUpdate("DELETE ...");
```

PREPARED STATEMENT

```
String sql = "SELECT nome FROM persone WHERE cognome = ?";
PreparedStatement prepared = connection.prepareStatement(sql);
prepared.setString(1, "Rossi");
ResultSet rs = stm.executeQuery();
```

PREPARED STATEMENT

```
String sql = "insert into persone (cognome, nome, eta) values
PreparedStatement prepared = connection.prepareStatement(sql);
prepared.setString(1, "Marroni");
prepared.setString(2, "Enrico");
prepared.setInt(3, 55);
prepared.executeUpdate();
```

CHIAMARE UNA SP

```
CallableStatement cStm = con.prepareCall("{call sp_name(?, ?)}
cStm.setString(1, "abcdefg");
boolean hadResults = cStm.execute();
```

PARAMETRI SP

```
CallableStatement cStm = con.prepareCall("{call sp_name(?, ?)}
cStm.registerOutParameter(2, Types.INTEGER);
// or cStmt.registerOutParameter("inOutParam", Types.INTEGER);
cStm.setString(1, "abcdefg");
cStm.setString(2, 1);
// or cStmt.setString("inputParam", 1);
boolean hadResults = cStm.execute();
```

RISULTATO SP

```
boolean hadResults = cStm.execute();
while (hadResults) {
    ResultSet rs = cStm.getResultSet();
    hadResults = cStm.getMoreResults();
}
int outputValue = cStm.getInt(2); // index-based
```

... COSE DA NON FARE

```
Scanner in = new Scanner(System.in);
String s = in.nextLine();
String sql = "SELECT * FROM Users WHERE UserId = "+s+";"
ResultSet rs = stm.executeQuery(sql);
while (rs.next()) {
    String col1 = rs.getString("...");
}
```

SQL INJECTIONS

Input: `100 OR 1=1`

```
String sql = "SELECT * FROM Users WHERE UserId = "+s+";"
```

una volta inseriti i valori diventa:

```
SELECT * FROM Users WHERE UserId = 100 OR 1=1;
```

SQL INJECTIONS

Input: `100 OR 1=1`

```
String sql = "SELECT UserId, Name, Password " +
             "FROM Users WHERE UserId = " + s + ";"
```

una volta inseriti i valori diventa:

```
SELECT UserId, Name, Password
FROM Users WHERE UserId = 100 OR 1=1;
```

SQL INJECTIONS

Input: `100; DROP TABLE customers`

```
String sql = "SELECT * FROM Users WHERE UserId = "+s+";"
```

una volta inseriti i valori diventa:

```
SELECT * FROM Users WHERE UserId = 100; DROP TABLE customers;
```

SQL INJECTIONS

```
Scanner in = new Scanner(System.in);
String user = in.nextLine();
String password = in.nextLine();
String sql = "SELECT * FROM Users WHERE " +
    "Name ='" + user + "' AND Pass ='" + password + "';"
ResultSet rs = stm.executeQuery(sql);
if (rs.next()) {
    // autenticato
}
```

SQL INJECTIONS

Input utente: `' or ''='`

Input password: `' or ''='`

```
String sql = "SELECT * FROM Users WHERE " +
    "Name ='" + user + "' AND Pass ='" + password + "'"
```

una volta inseriti i valori diventa:

```
SELECT * FROM Users WHERE
Name ='' or ''='' AND Pass ='' or ''='';
```

SQL INJECTIONS

```
Scanner in = new Scanner(System.in);
String categoria = in.nextLine();
String sql = "SELECT name, description " +
    "FROM products "+
    "WHERE category ='" + categoria + "' and release = 1;"
ResultSet rs = stm.executeQuery(sql);
while (rs.next()) {
    // autenticato
}
```

SQL INJECTIONS

Input categoria: `' UNION SELECT user, password FROM users--`

```
String sql = "SELECT name, description " +
    "FROM products "+
    "WHERE category ='" + categoria + "' and release = 1;"
```

una volta inseriti i valori diventa:

```
SELECT name, description FROM products
WHERE category =''
UNION SELECT user, password FROM users--' and release = 1;
```

SQL INJECTIONS

Cisco Data Center Network Manager 11.2.1 - 'getVmHostData' SQL Injection

EDB-ID:	CVE:
48019	2019-15984 2019-15976

Stringa inserita: `";UPDATE ... --`

PYTHON

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost", user="userdb",
  password="***", database="classicmodels"
)

mycursor = mydb.cursor()
sql = "SELECT * FROM employees WHERE lastName = %s"
adr = ("Smith", )
mycursor.execute(sql, adr)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

OBJECT-RELATIONAL MAPPING

INTEGRAZIONE TRA LINGUAGGI DI PROGRAMMAZIONE AD OGGETTI E DBMS RELAZIONALI

```
String sql = "SELECT ... FROM persons WHERE id = 10";
Statement = con.createStatement();
ResultSet res = cmd.executeQuery(sql);
String name = res[0]["FIRST_NAME"];
```

diventa

```
Person p = repository.GetPerson(10);
String name = p.FirstName;
```

SQL? INUTILE?

- impossibile usare ORM se non si sa progettare DB
- molti sistemi non li usano
- a volte meno efficiente che non scrivere query a mano

OBJECT-RELATIONAL MAPPING

COSA SI FA?

- conversione di oggetti in valori scalari
- mantenimento delle relazioni tra gli oggetti

TANTI FRAMEWORK DIVERSI RISOLVONO IL PROBLEMA

OBJECT-RELATIONAL MAPPING

JAVA PERSISTANCE API (JPA)

- Hibernate
- EclipseLink
- TopLink

OBJECT-RELATIONAL MAPPING

FRAMEWORK .NET

- LINQ to SQL (solo SQL server)
- ADO Entity Framework
- NHibernate

HIBERNATE

- utilizzabile in JAVA (estende JPA)
- utilizzabile in C# (NHibernate)
- supporta moltissimi DBMS diversi

CONFIGURARE [N]HIBERNATE

OCCORRE SPECIFICARE

- lato Java: quale implementazione JPA usare
- Connection Proveider
- dettagli della connessione
- strategia generazione delle tabelle
- lista delle entità

CONFIGURAZIONE A RUN-TIME

FILE DI CONFIGURAZIONE

MAPPING

```
public class Studente{
    private long idStud;
}
```

… come gli dico che `id` è la chiave primaria?

MAPPING

IMPOSTAZIONE CHIAVE PRIMARIA

- file XML che definisce le impostazioni
- annotazioni (attributi in C#)

```
@Id
@GeneratedValue(strategy = GenerationType.A
private long idStud;
```

MAPPING

COME FACCIO MAPPING?

```java
@Entity
@Table(name = "Studente")
public class Studente{
    @Id
    @Column(name = "idStud")
    private long idStud;
    @Column(name = "nome")
    String nome;
}
```

ENTITYMANAGER, FACTORY & CO

ENTITYMANAGERFACTORY

- apre il database
- aggiorna la struttura del database
- operazione pesante e complicata
- ne basta una per applicazione (static)

ENTITYMANAGER, FACTORY & CO

ENTITYMANAGER

- connessione di breve durata al DB
- permette di eseguire operazioni sul DB

ENTITYMANAGER, FACTORY & CO

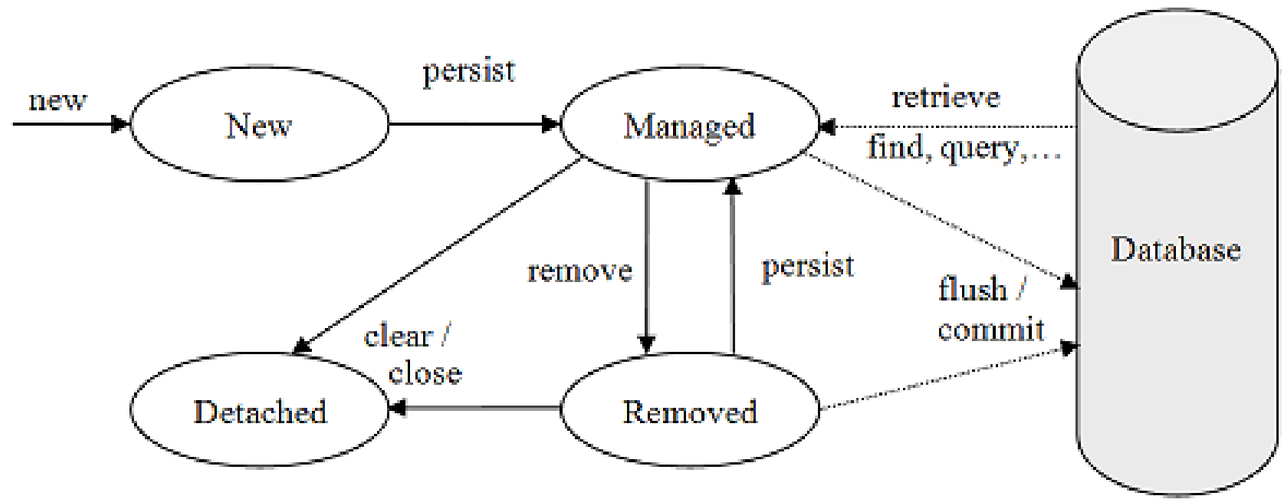
ENTITYTRANSACTION

- racchiude operazioni che modificano il contenuto del DB

QUERY

- utilizzano una sintassi particolare (JPQL)

ENTITY OBJECT LIFE CYCLE



PERSIST O MERGE?

`persist()`

- si applica su nuove entità
- genera una INSERT in SQL
- non restituisce nulla

PERSIST O MERGE?

`merge()`

- entità nuova o detached
- genera una INSERT o una UPDATE
- restituisce la versione managed dell'oggetto

TUTTO VA SEMPRE RACCHIUSO IN UNA TRANSAZIONE

OTTENERE OGGETTI

RICERCA PER CHIAVE PRIMARIA

```
Employee employee = em.find(Employee.class, 1);
```

JPA QUERY LANGUAGE

```
Query q1 = em.createQuery
```

OTTENERE OGGETTI

JPA CRITERIA QUERY

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Country> q = cb.createQuery(Country.
class);
Root<Country> c = q.from(Country.class);
q.select(c);
```

NAMED QUERY

```
@NamedQuery(name="Country.findAll", query="SELECT c FROM
Country c")
```

ACCESSO LAZY

Un oggetto è *lazy* se non contiene tutti i dati ma sa come ottenerli

Effettua la query solo quando servono i dati:

- ottengo studente XYZ, non vado a leggere subito i dati di tutti gli esami e di tutti i corsi collegati
- opzione dell'annotazione della relationship

```
@OneToMany(fetch=javax.persistence.FetchType.LAZY)
```

TRIGGERS

EQUIVALENTE DEI TRIGGERS NEL MONDO JPA

creo un metodo privato ma senza parametri

- `@PreUpdate`
- `@PrePersist`
- `...`

STORED PROCEDURE

NON POSSO CREARLE CON JPA

POSSO RICHIAMARLE (JPA >= 2.1)

```
@NamedStoredProcedureQuery(
    name = "ReadAddressById",
    resultClasses = Address.class,
    procedureName = "READ_ADDRESS",
    parameters = {
        @StoredProcedureParameter(mode=IN,
        name="P_ADDRESS_ID", type=Long.class)
    }
)
@Entity
public class Address {
...
}
```

VERSIONI

HIBERNATE PUÒ TENERE TRACCIA DELLE MODIFICHE OPERATE SUGLI OGGETTI

```
@org.hibernate.envers.Audited
```

VERSIONI

POSSO RICERCARE VERSIONI PRECEDENTI DELLO STESSO OGGETTO:

```
AuditReader reader = AuditReaderFactory.get(em)
Event firstRevision = reader.find(Event.class, 2L, 1);
Event secondRevision = reader.find(Event.class, 2L, 2);
```

PROGETTAZIONE BASE DI DATI

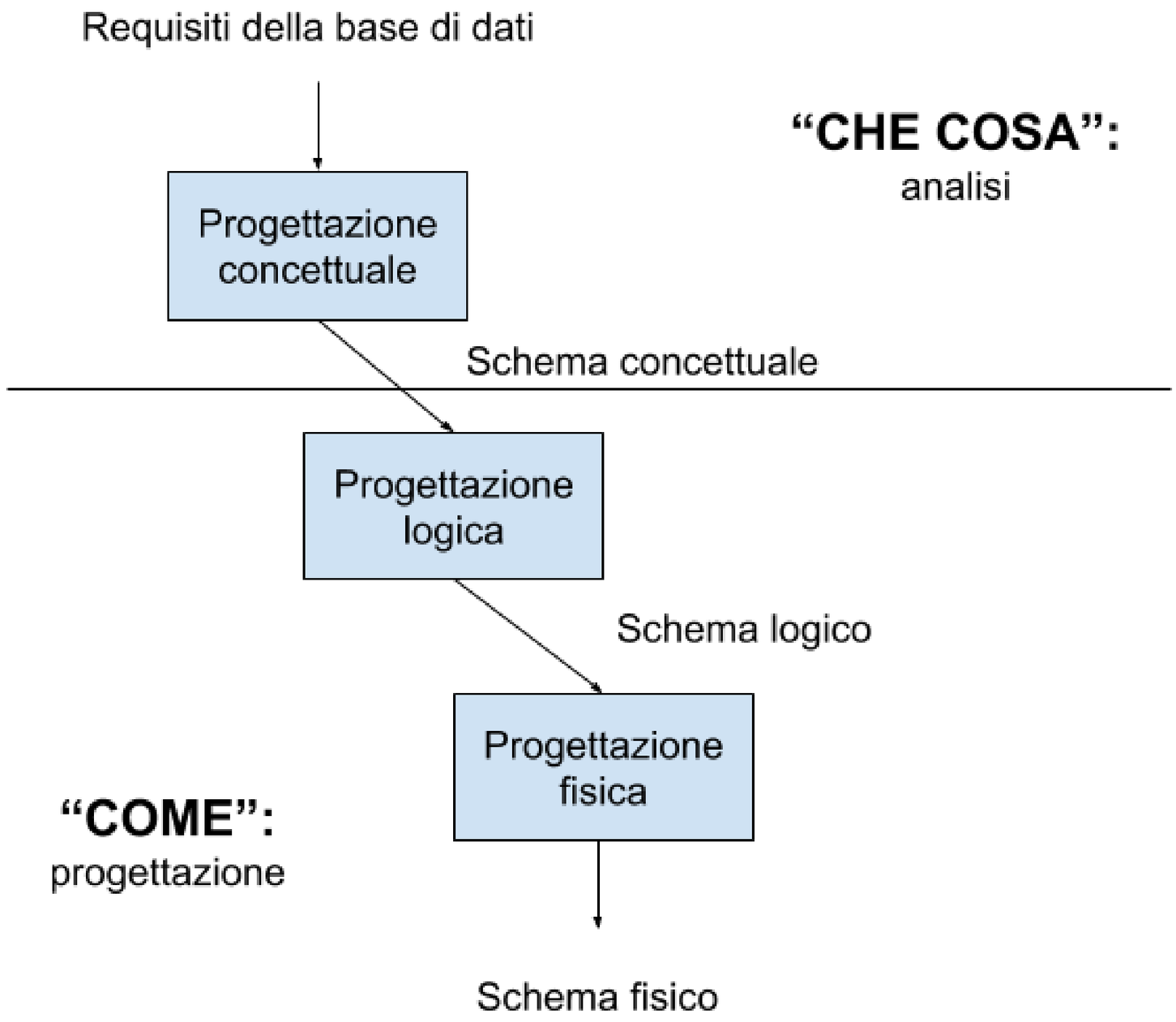
RICORDATE?

IN OGNI BASE DI DATI ESISTONO:

- lo **schema**: descrive la struttura della base di dati e non varia nel tempo
    - nel modello relazionale: intestazioni delle tabelle
- l'**istanza**: i valori attuali, che possono variare anche rapidamente
    - nel modello relazionale: le righe delle tabelle

CIRCLE OF LIFE

- Studio di fattibilità: definizione costi e priorità
- **Raccolta e analisi dei requisiti: studio delle proprietà del sistema**
- **Progettazione: di dati e funzioni**
- Realizzazione (prototipo?)
- Validazione e collaudo: sperimentazione
- Funzionamento: il sistema diventa operativo


Requisiti della base di dati
“CHE COSA”:
analisi
Progettazione concettuale
Schema concettuale
Progettazione logica
Schema logico
Progettazione fisica
“COME”:
progettazione
Schema fisico

MODELLI PRINCIPALI

- modelli logici
- modelli concettuali

MODELLI LOGICI

UTILIZZATI NEI DBMS ESISTENTI PER L’ORGANIZZAZIONE DEI DATI

- utilizzati dai programmi
- indipendenti dalle strutture fisiche
- esempi: relazionale, reticolare, gerarchico…

MODELLI CONCETTUALI

RAPPRESENTANO I DATI IN MODO INDIPENDENTE DA OGNI SISTEMA

- cercano di descrivere i concetti del mondo reale
- sono utilizzati nelle fasi preliminari di progettazione
- il più noto è il modello Entity-Relationship

MODELLI CONCETTUALI, PERCHÉ?

Proviamo a modellare una applicazione definendo direttamente lo schema logico della base di dati

DA DOVE COMINCIAMO?

dobbiamo pensare subito a come correlare le varie tabelle (chiavi etc.)

MODELLI CONCETTUALI, PERCHÉ?

MODELLI CONCETTUALI

- permettono di rappresentare le classi di dati di interesse e le loro correlazioni
- prevedono efficaci rappresentazioni grafiche (utili anche per documentazione)
- indipendenti dal DBMS scelto

MODELLO ENTITY-RELATIONSIP

ENTITÀ - RELAZIONE

Il più diffuso modello concettuale, ne esistono molte versioni (più o meno) diverse

ASPETI CHIAVI MODELLO E-R

- entità
- relationship
- attributo
- identificatore
- generalizzazione

ENTITÀ

CLASSE DI OGGETTI (FATTI, PERSONE, COSE) DELLA APPLICAZIONE DI INTERESSE CON PROPRIETÀ COMUNI E CON ESISTENZA AUTONOMA

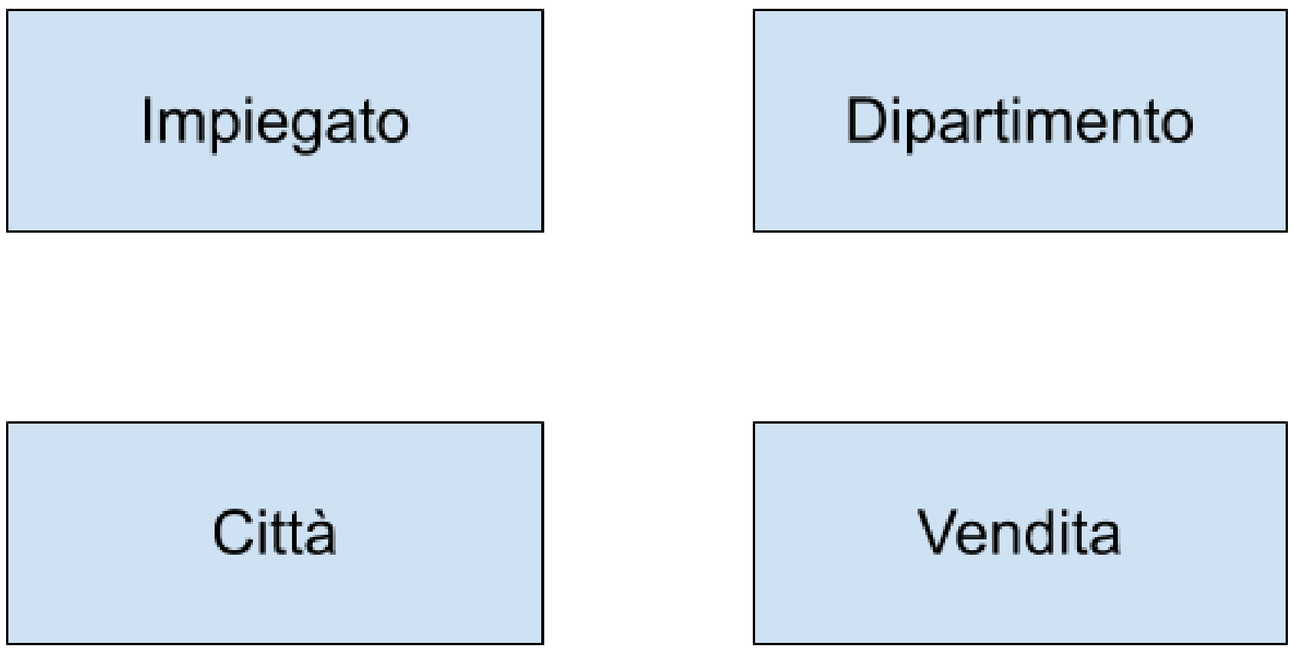
ESEMPI:

impiegato, città, conto corrente, ordine, fattura

ISTANZA: ELEMENTO DELLA CLASSE

nello schema concettuale rappresentiamo le entità, non le singole istanze

RAPPRESENTAZIONE DI ENTITÀ



- nomi espressivi
- opportune convenzioni: singolare

RELATIONSHIP

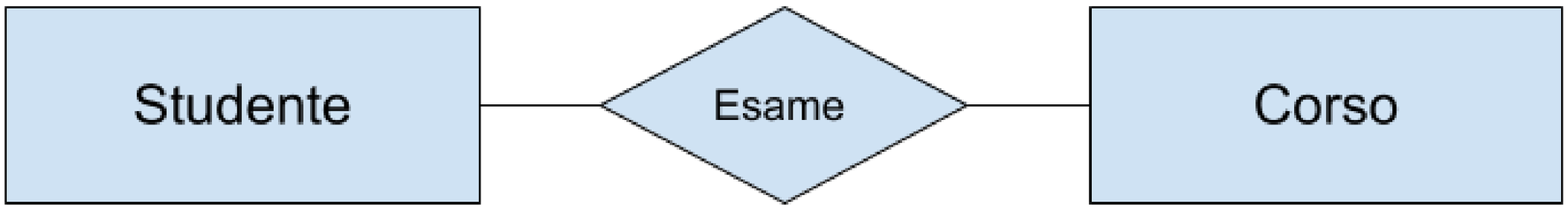
LEGAME LOGICO FRA DUE O PIÙ ENTITÀ, RILEVANTE NELL'APPLICAZIONE DI INTERESSE

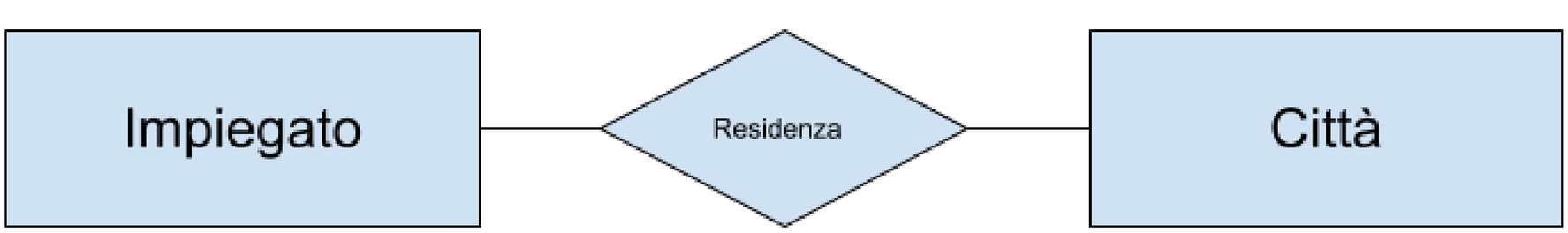
Esempi:

- Residenza (fra persona e città)
- Esame (fra studente e corso)

Chiamata anche: relazione, correlazione, associazione

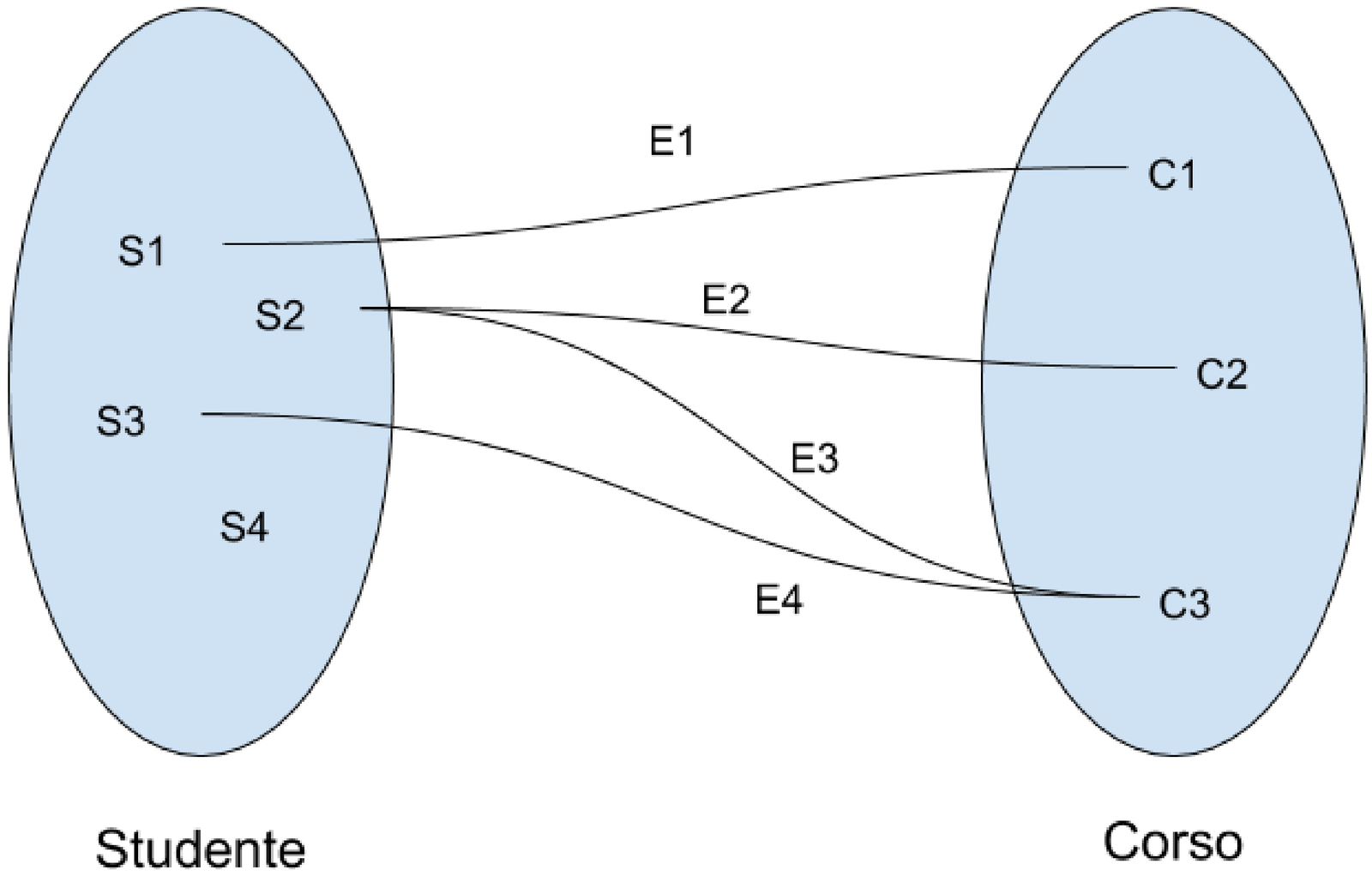
RAPPRESENTAZIONE DI RELAZIONE



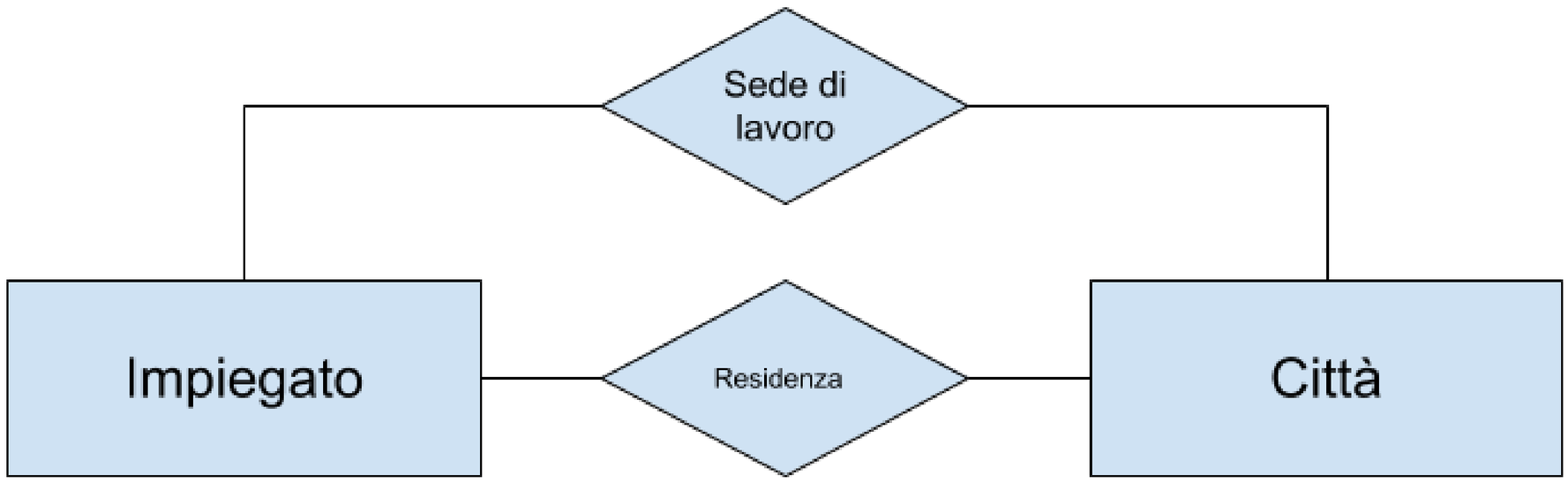


- nomi espressivi
- opportune convenzioni: singolare e sostantivi invece che verbi (se possibile)

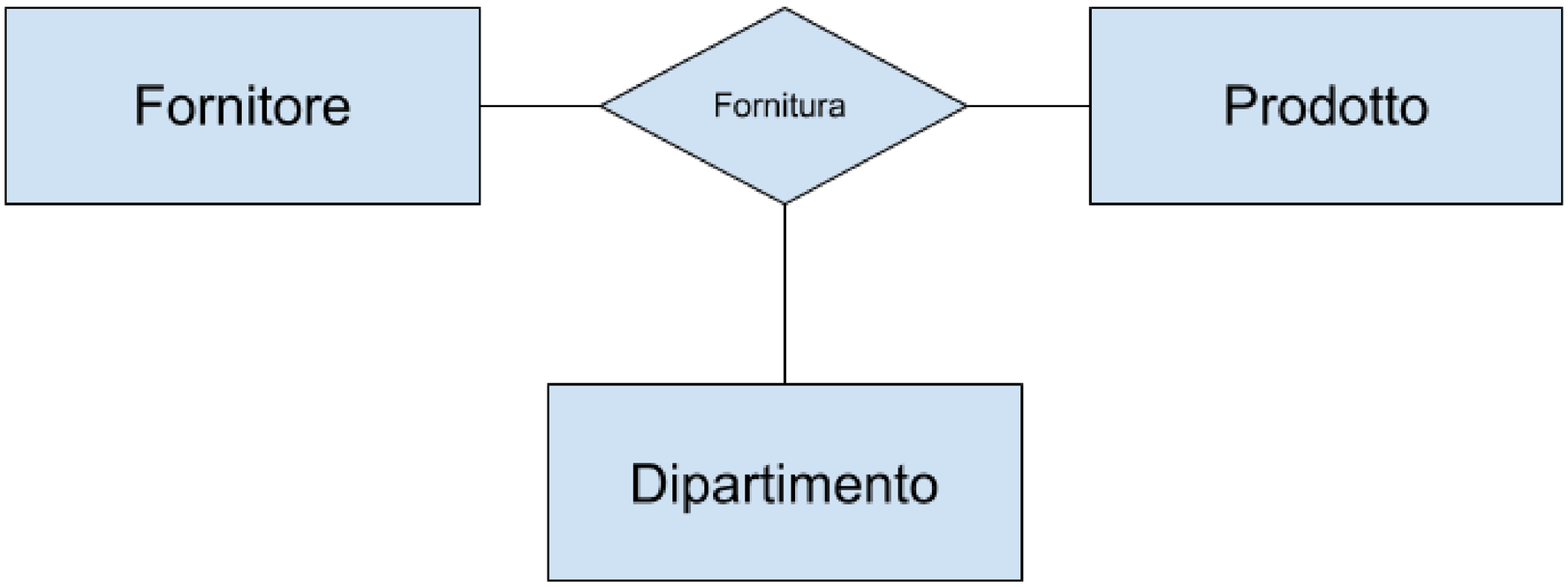
ESEMPI DI ISTANZE



DUE RELAZIONI SULLA STESSA ENTITÀ

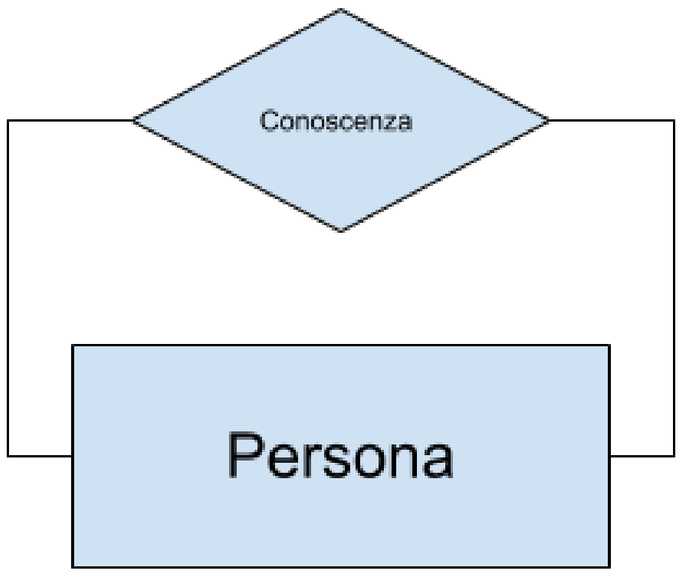


RELATIONSHIP N-ARIA

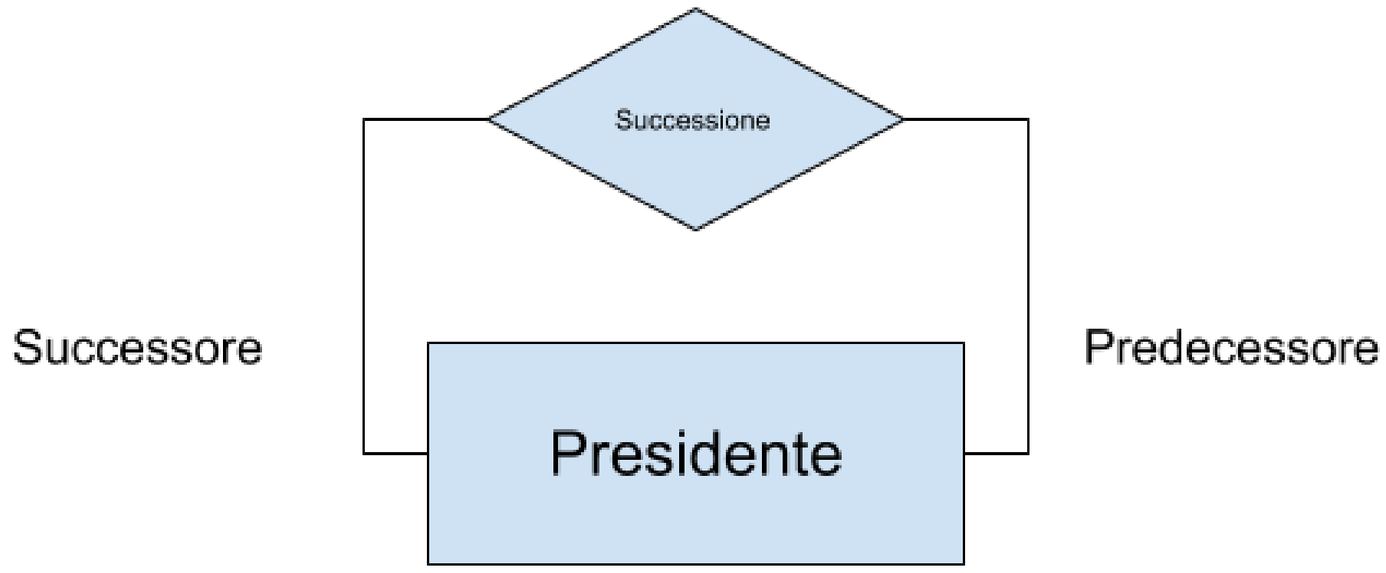


RELATIONSHIP RICORSIVA

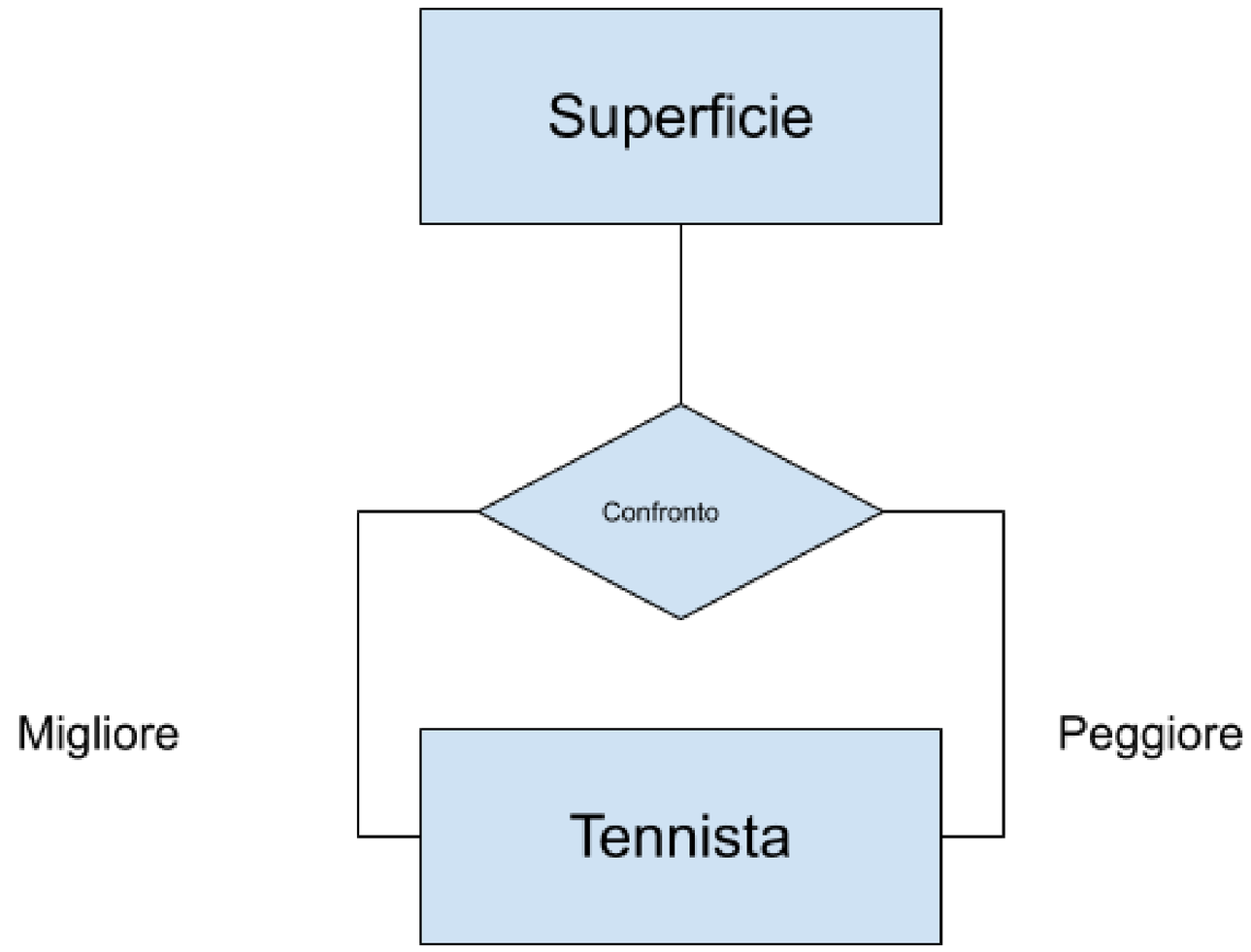
COINVOLGE DUE VOLTE LA STESSA ENTITÀ



RELATIONSHIP RICORSIVA CON RUOLI



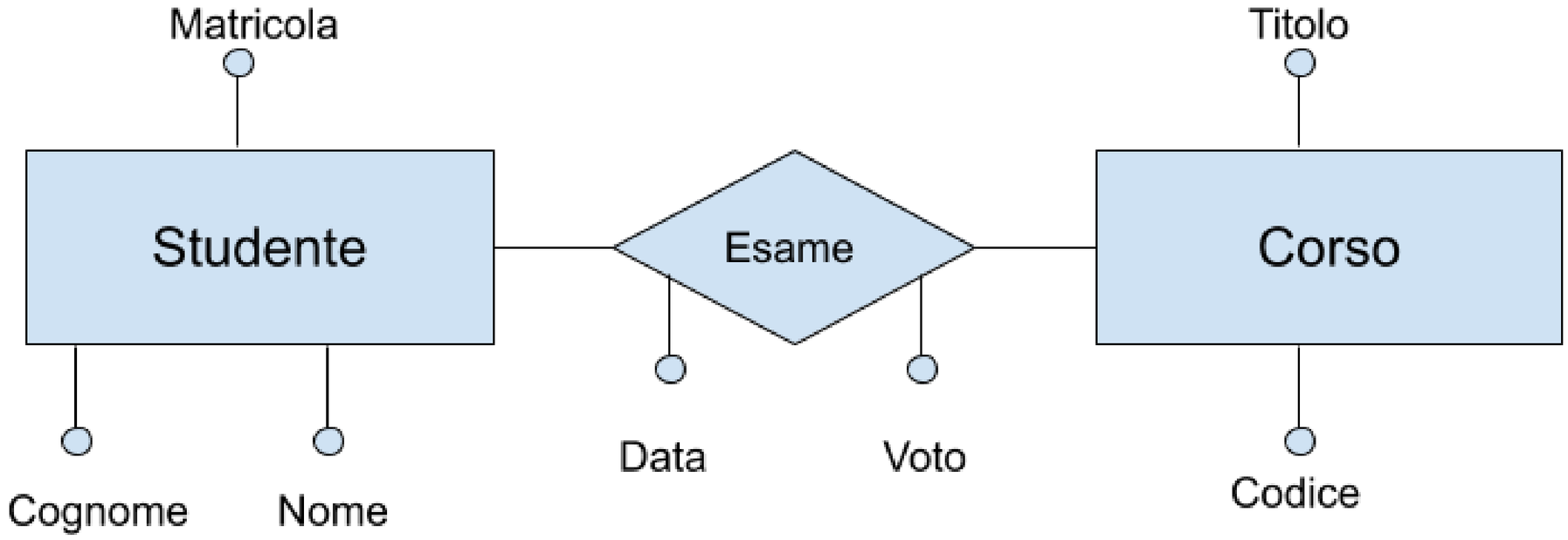
RELATIONSHIP TERNARIA RICORSIVA



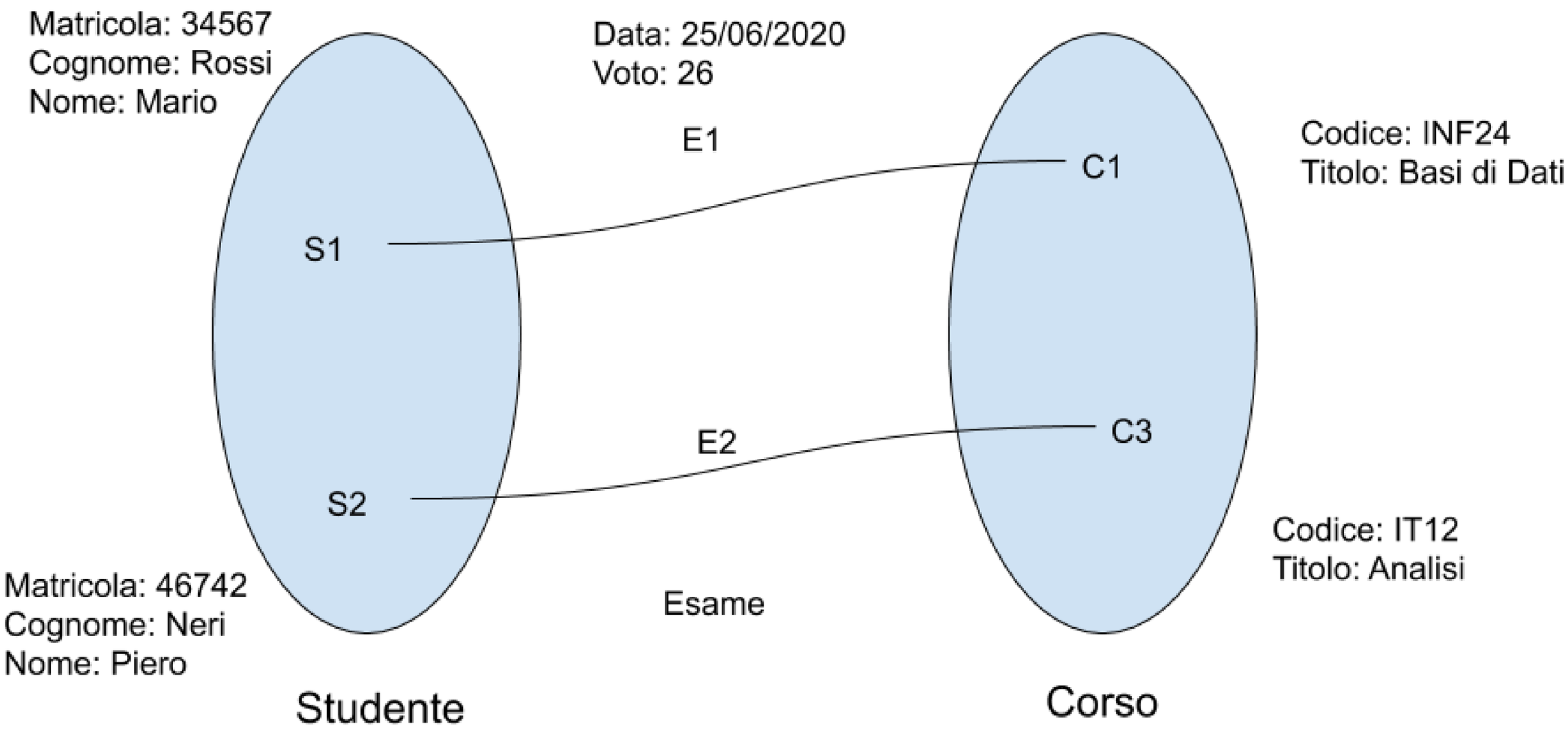
ATTRIBUTO

- proprietà elementare di un’entità o di una relationship, di interesse per l’applicazione
- associa ad ogni occorrenza di entità o relationshio un valore del dominio dell’attributo

RAPPRESENTAZIONE DI ATTRIBUTI



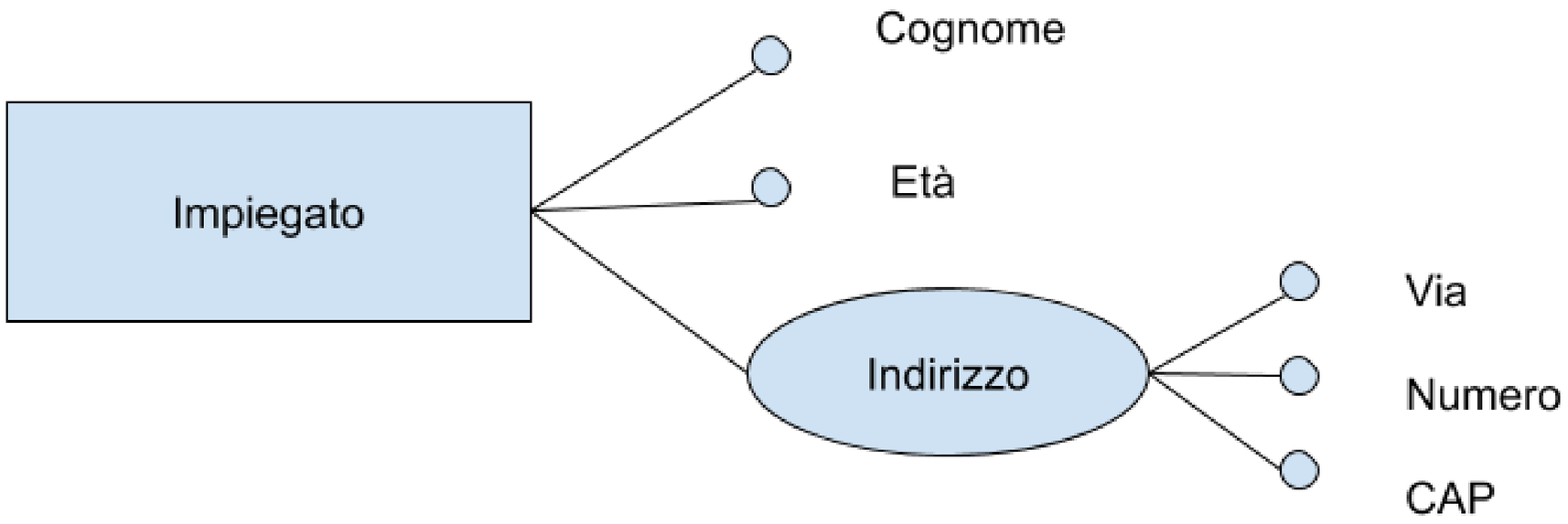
ESEMPI DI ISTANZE



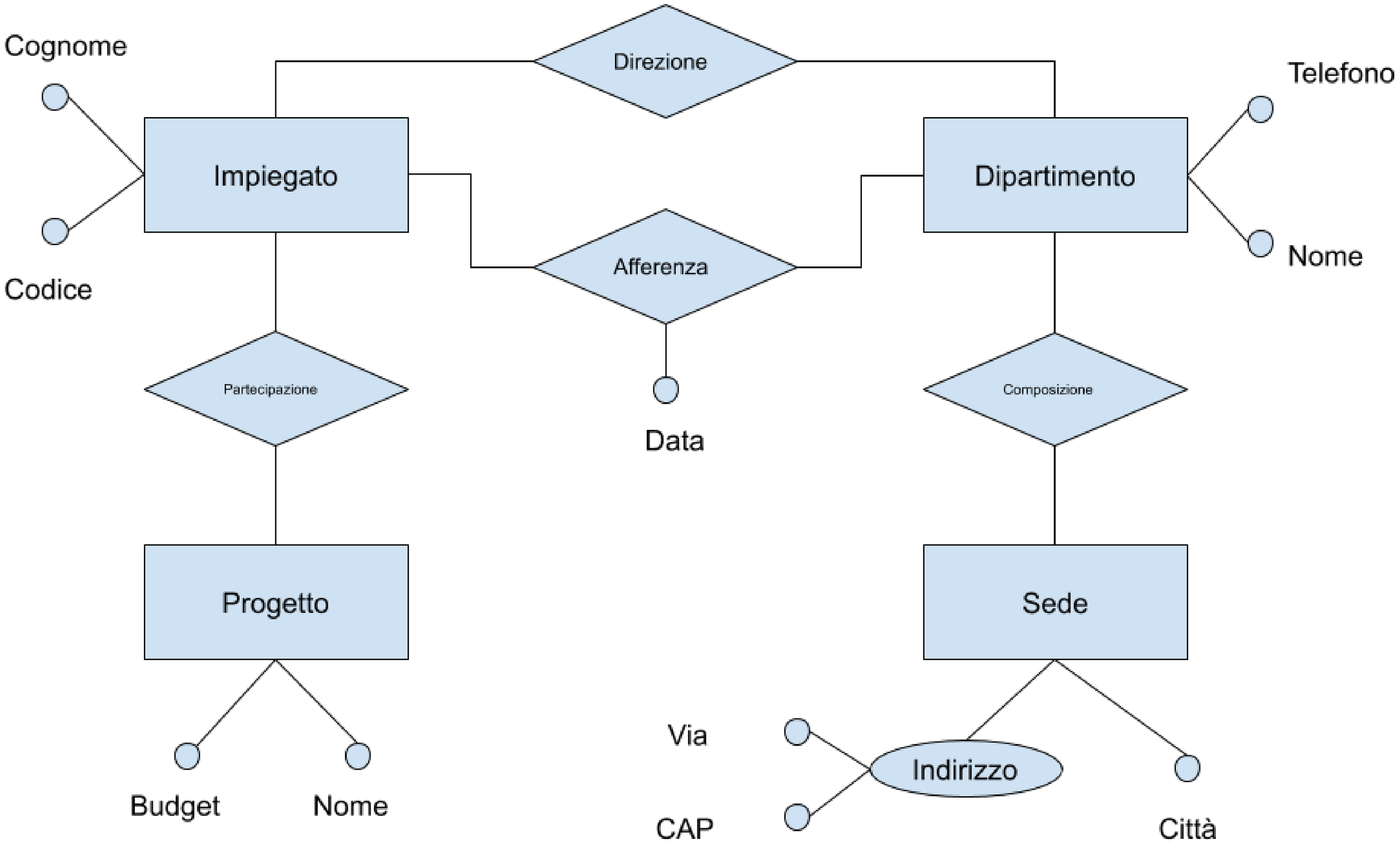
ATTRIBUTI COMPOSTI

- raggruppano attributi di una medesima entità o relationship che presentano affinità nel loro significato o uso
- es: Via, Numero civico e CAP formano “Indirizzo”

RAPPRESENTAZIONE GRAFICA



RAPPRESENTAZIONE GRAFICA



ALTRI COSTRUTTI DEL MODELLO E-R

- cardinalità
- identificatore
- generalizzazione

ALTRI COSTRUTTI DEL MODELLO E-R

CARDINALITÀ

- di relationship
- di attributo

IDENTIFICATORE

- interno
- esterno

CARDINALITÀ

Coppia di valori associati ad ogni entità che partecipa ad una relationship

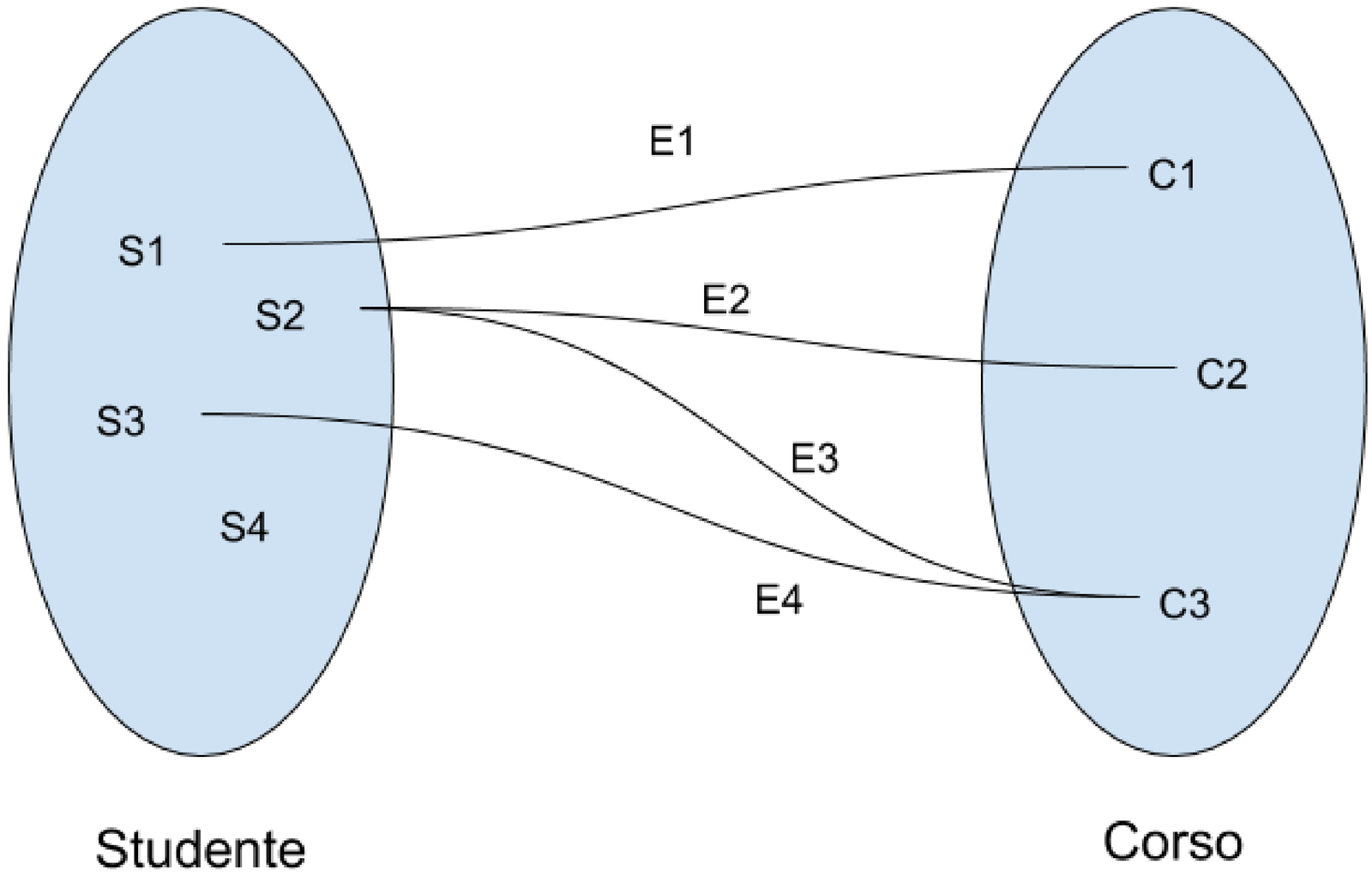
Specificano il numero min e max di volte che un'occorrenze di una rentità può essere associata ad un'occorrenza di un'altra entità coinvolta nella relationship

ESEMPIO DI CARDINALITÀ

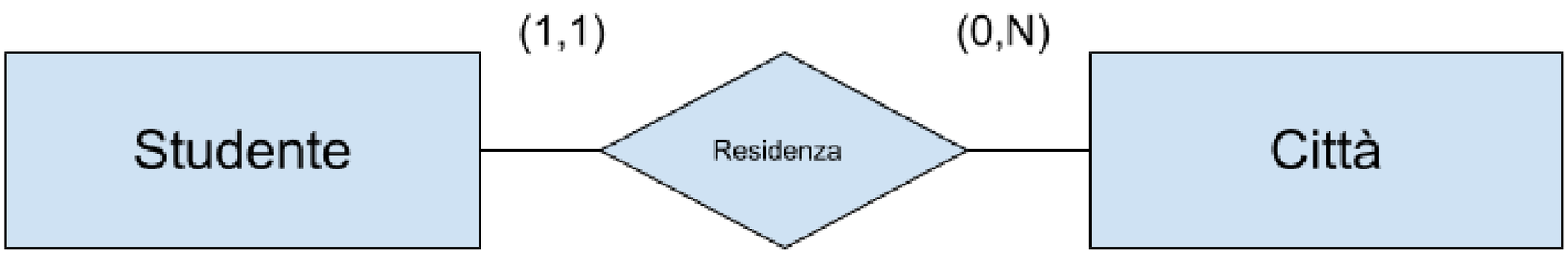
Per semplicità useremo tre simboli:

- 0 e 1 per la cardinalità minima
  - 0 = "partecipazione opzionale"
  - 1 = "partecipazione obbligatoria"
- 1 e N per la cardinalità massima
  - N = nessun limite

ISTANZE DI ESAME



CARDINALITÀ DI RESIDENZA

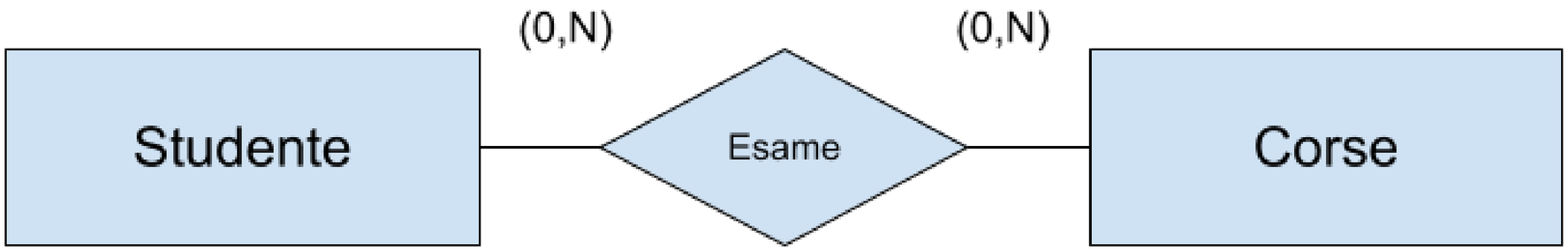


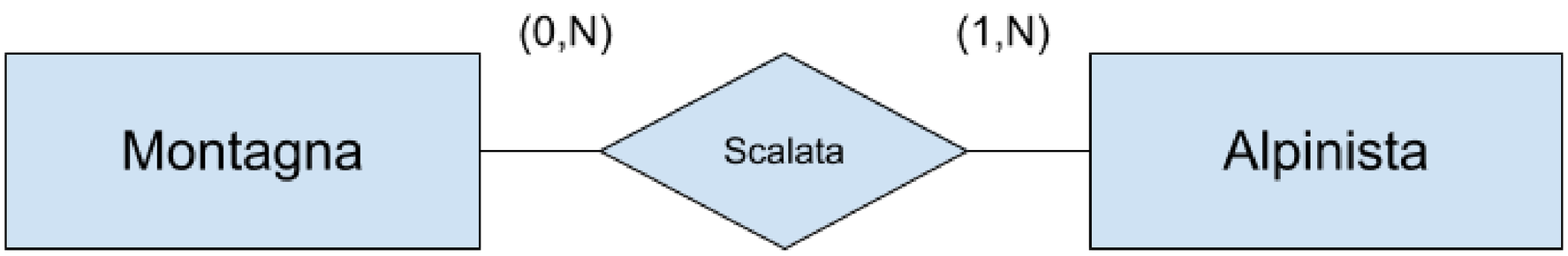
TRE TIPI DI RELATIONSHIPS

CON RIFERIMENTO ALLE CARDINALITÀ MASSIME, ABBIAMO REALTIONSHIP:

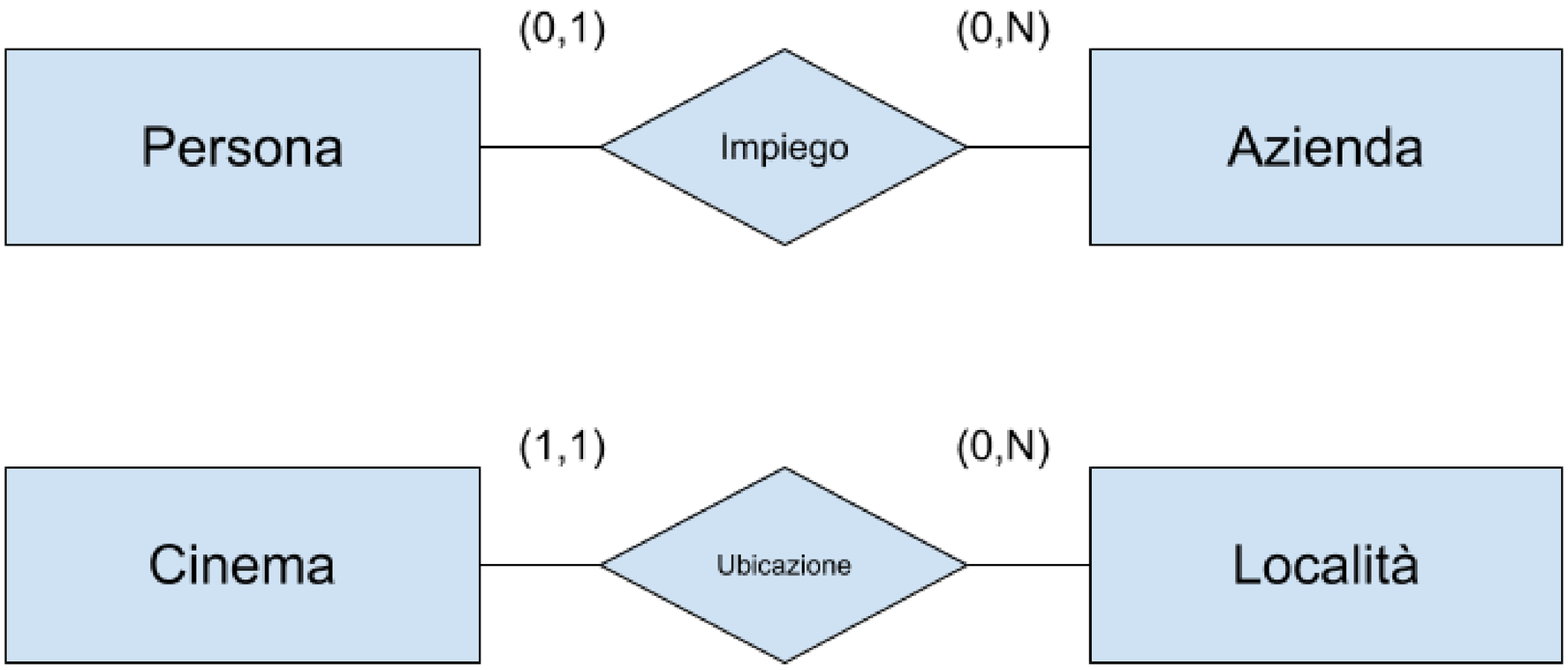
- uno a uno
- uno a molti
- molti a molti

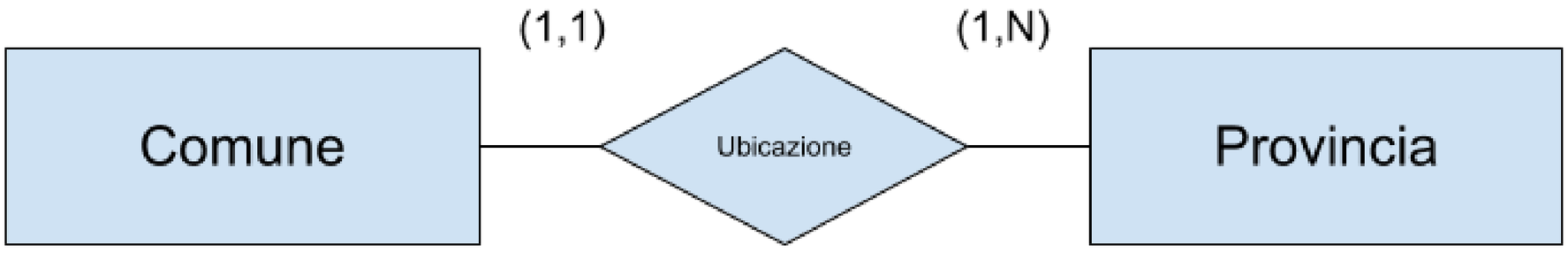
RELATIONSHIP MOLTI A MOLTI





RELATIONSHIP UNO A MOLTI

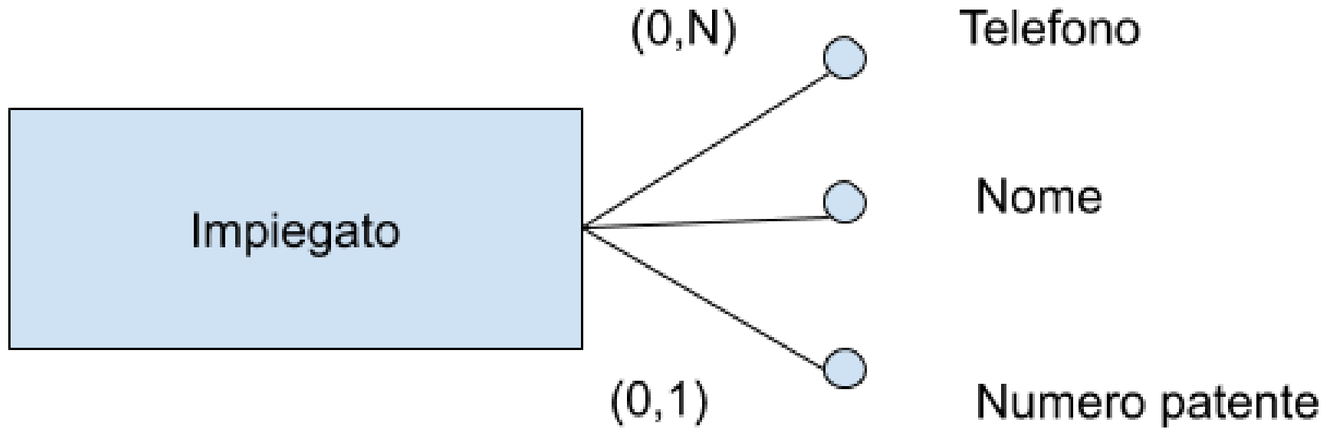




CARDINALITÀ DI ATTRIBUTI

È possibile associare delle cardinalità anche agli attributi, con due scopi:

- indicare opzionalità ("informazione incompleta")
- indicare attributi multivalore



IDENTIFICATORE DI UNA ENTITÀ

STRUMENTO PER L'IDENTIFICAZIONE UNIVOCA DELLE OCCORRENZE DI UN'ENTITÀ COSTITUITO DA:

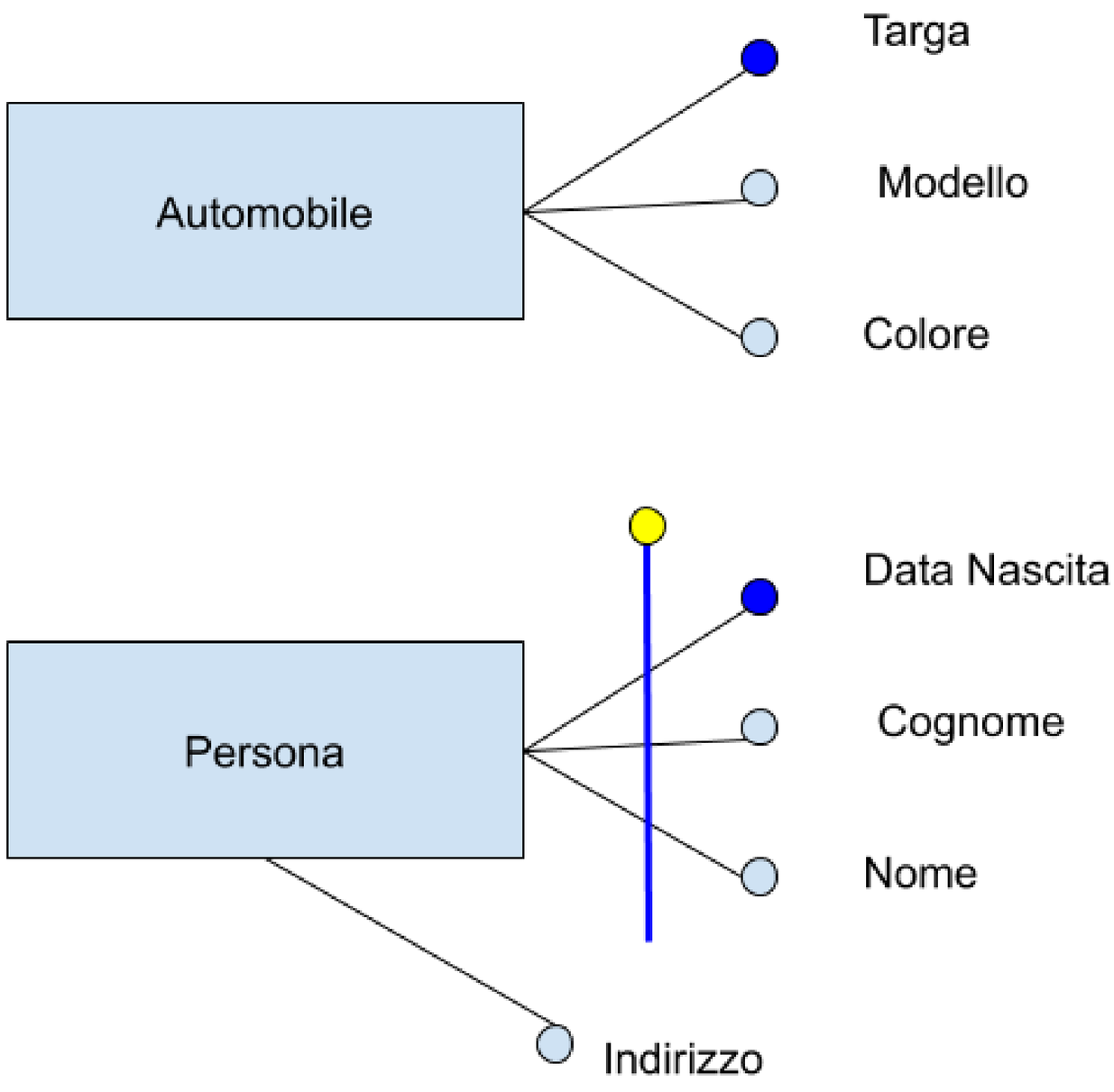
- attributi dell'entità → identificatore interno
- (attributi +) entità esterne attraverso relationship → identificatore esterno

IDENTIFICATORE DI UNA ENTITÀ

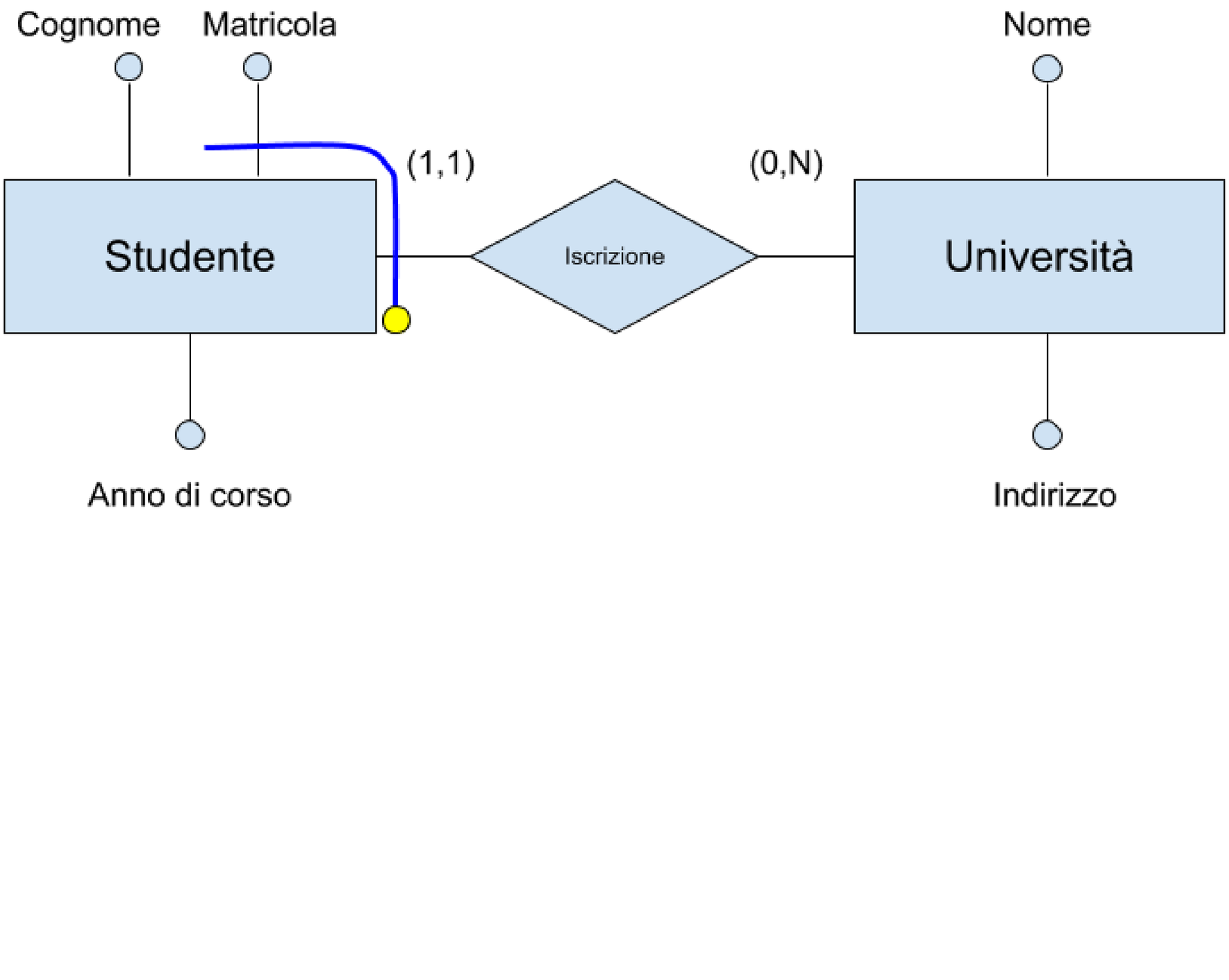
ATTENZIONE:

- ogni entità deve possedere almeno un identificatore, ma può averne in generale più di uno
- una identificazione esterna è possibile solo attraverso una relationship a cui l’entità da identificare partecipa con cardinalità (1,1)

IDENTIFICATORI INTERNI



IDENTIFICATORI ESTERNI



GENERALIZZAZIONE

METTE IN RELAZIONE UNA O PIÙ ENTITÀ $E_1, E_2, \ldots, E_n$ CON UNA ENTITÀ E

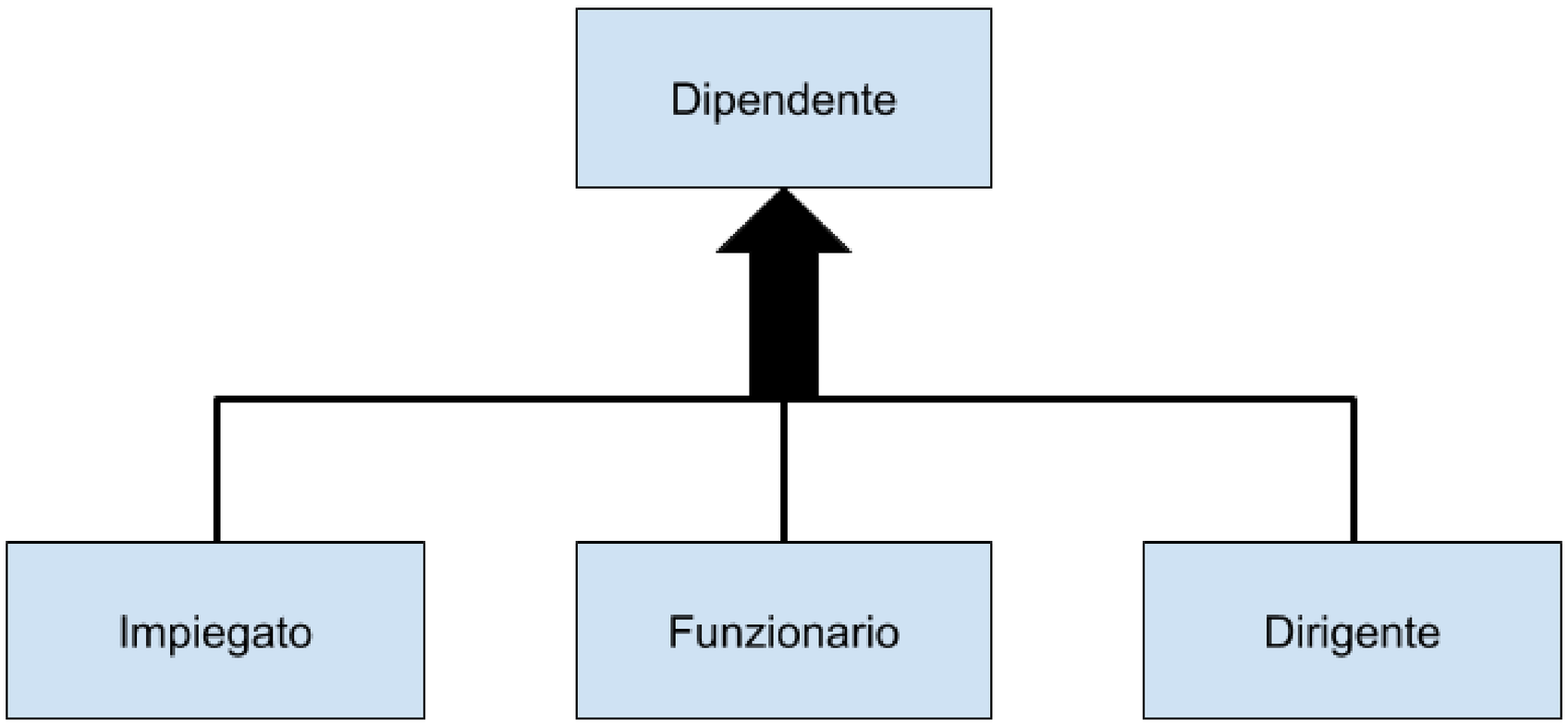
- E è **generalizzazione** di $E_1, E_2, \ldots, E_n$
- $E_1, E_2, \ldots, E_n$ sono **specializzazioni** (o sottotipi) di E

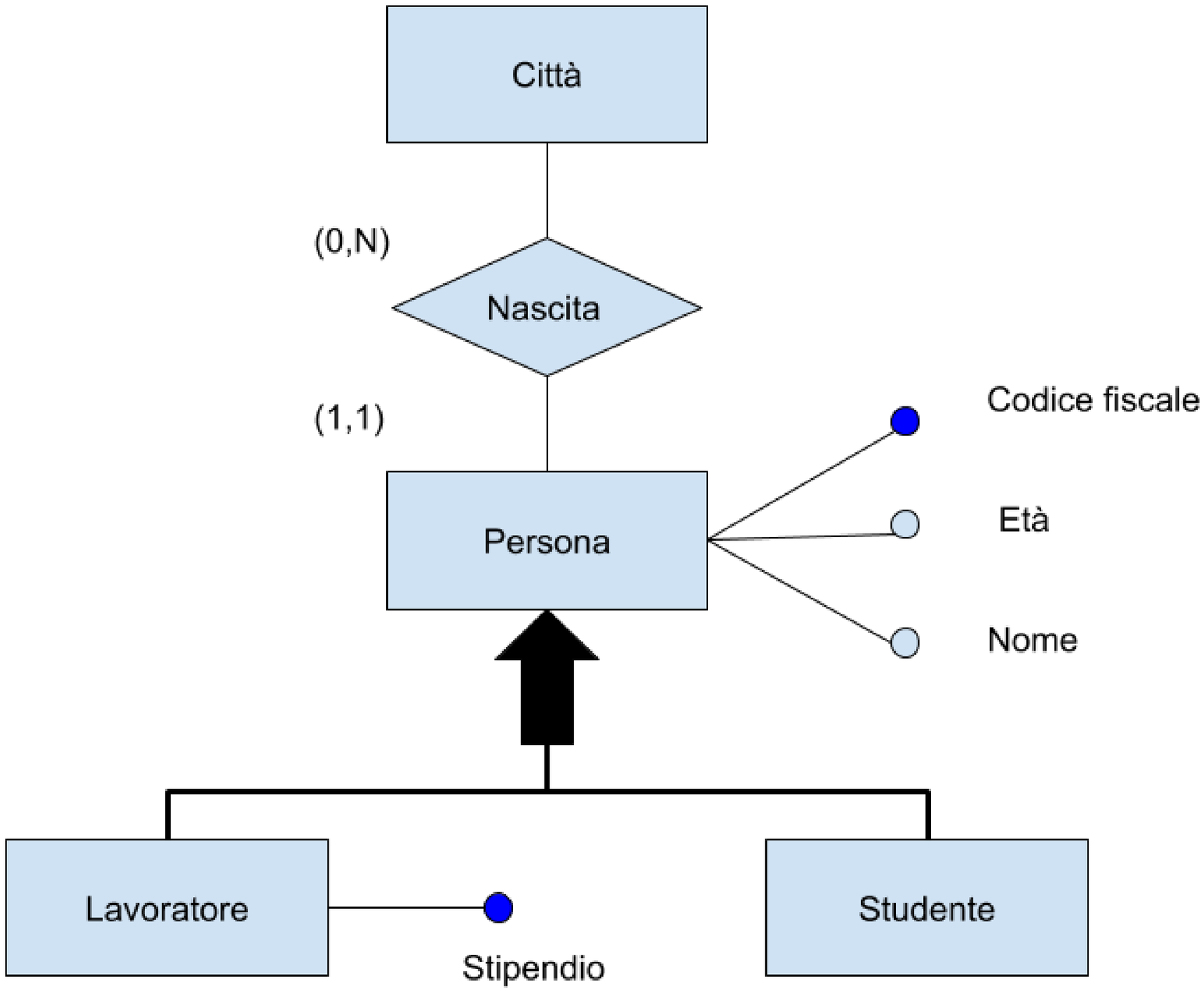
GENERALIZZAZIONE

EREDITARIETÀ

Tutte le proprietà (attributi, relationship, altre generalizzazioni) dell’entità genitore vengono ereditate dalle entità figlie e non rappresentate esplicitamente

RAPPRESENTAZIONE GRAFICA




Città
(0,N)
Nascita
(1,1)
Persona
Codice fiscale
Età
Nome
Lavoratore
Stipendio
Studente

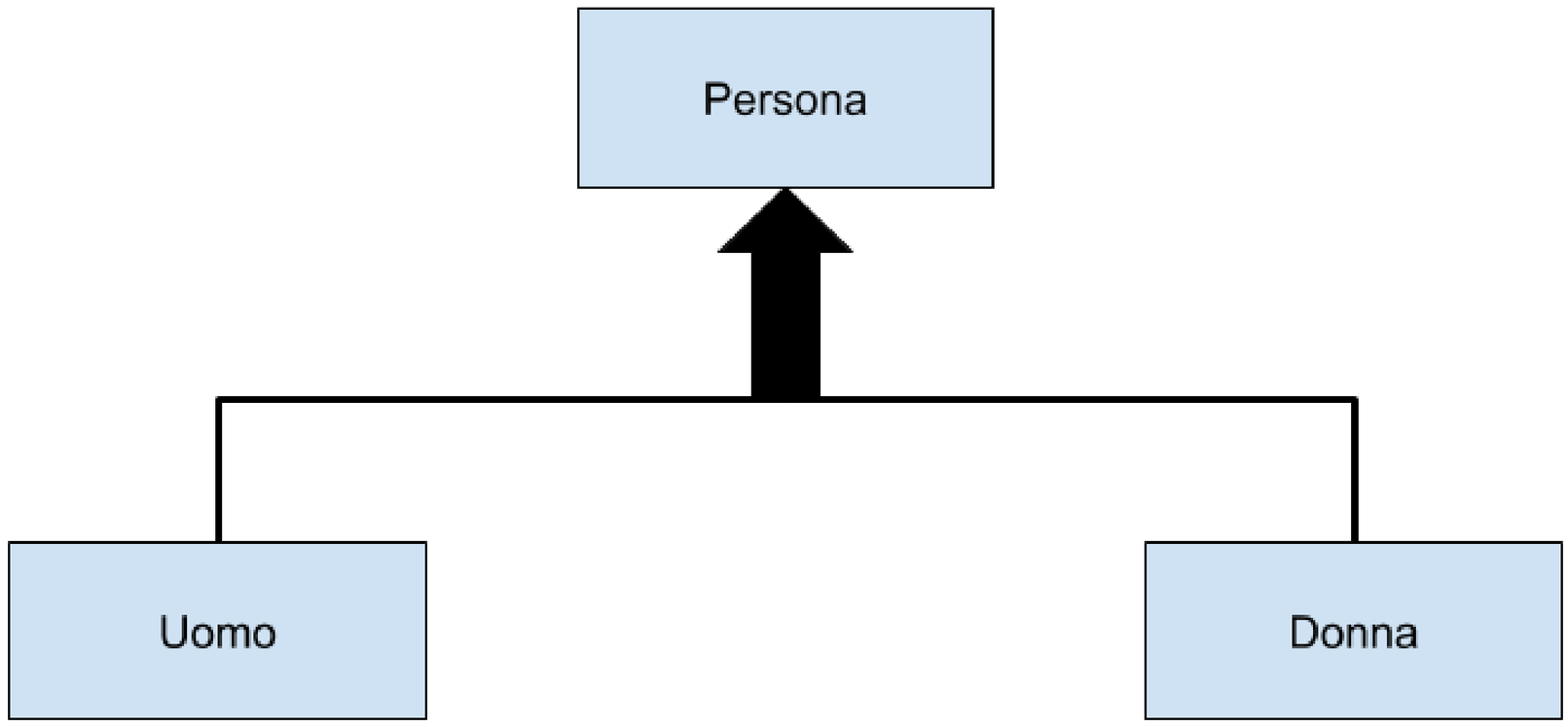
TIPI DI GENERALIZZAZIONI

- **totale** se ogni occorrenza dell’entità genitore è occorrenza di almeno una delle entità figlie, altrimenti è **parziale**
- **esclusiva** se ogni occorrenza dell’entità genitore è occorrenza di al più una delle entità figlie, altrimenti è **sovrapposta**

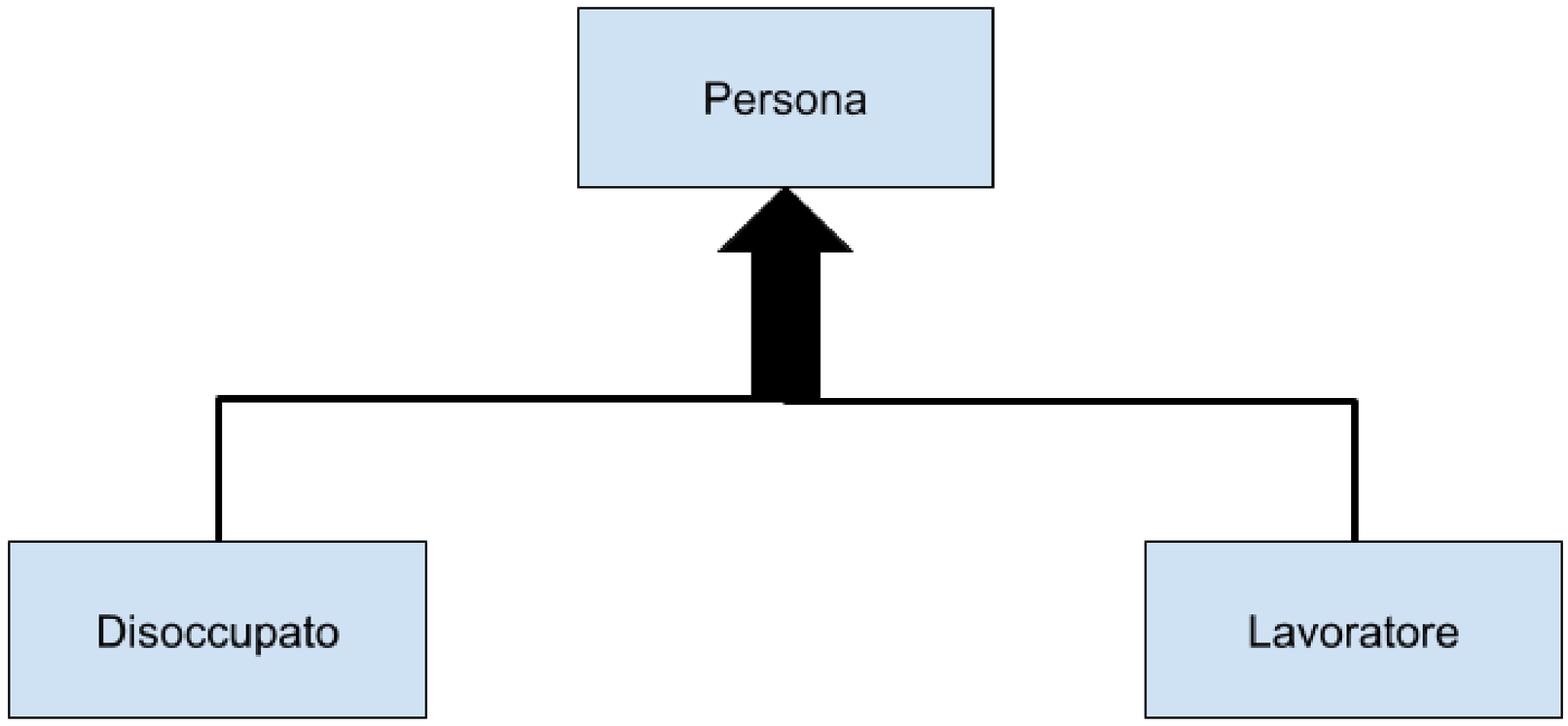
TIPI DI GENERALIZZAZIONI

Consideriamo (senza perdita di generalità) solo generalizzazioni esclusive e distinguiamo fra totali e parziali

GENERALIZZAZIONE TOTALE



GENERALIZZAZIONE PARZIALE

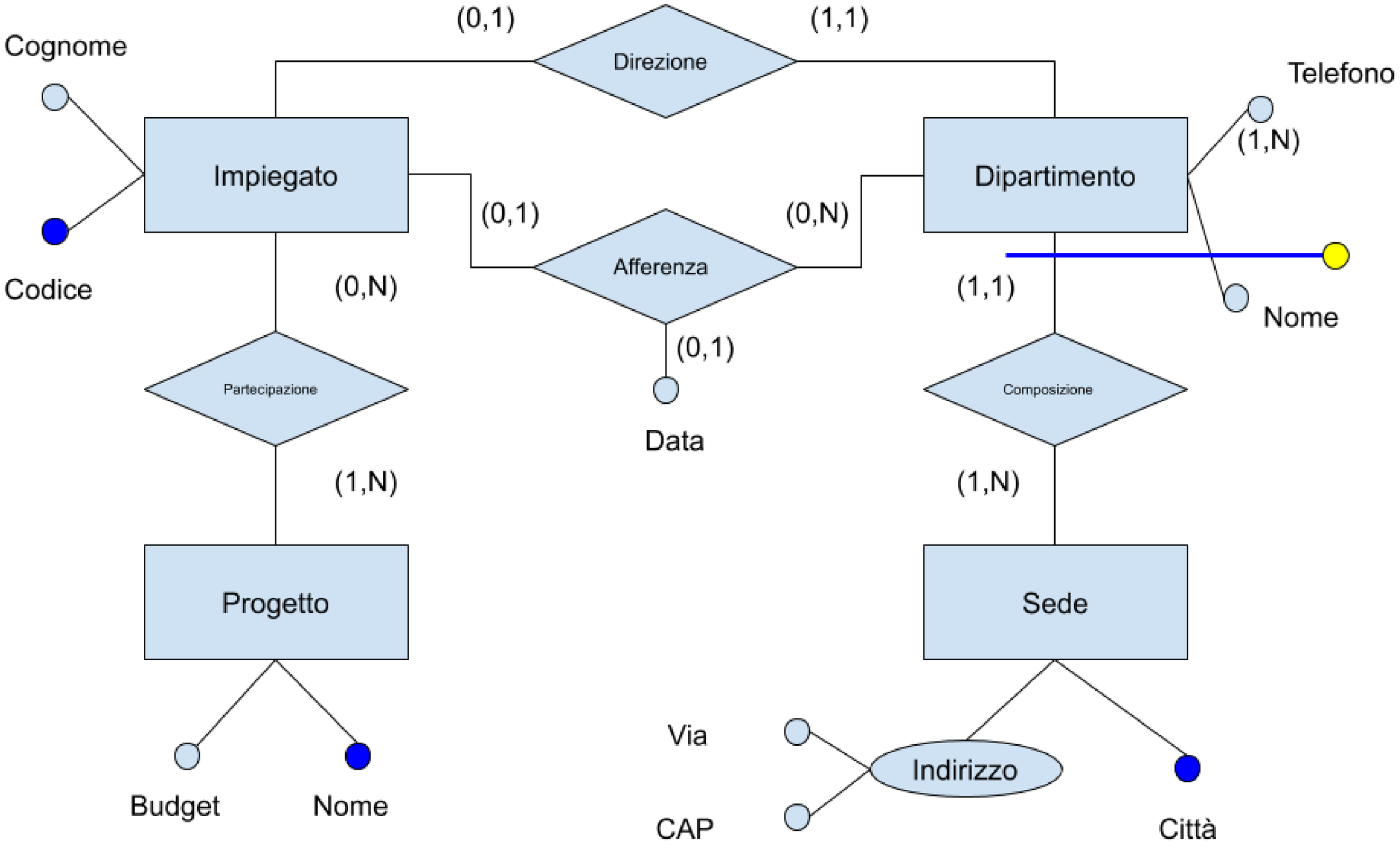


DOCUMENTI AGGIUNTIVI

DIZIONARIO DEI DATI

- entità
- relationship

VINCOLI NON ESPRIMIBILI


Cognome
(0,1)
Direzione
(1,1)
Telefono
Impiegato
Dipartimento
(1,N)
Codice
(0,1)
(0,N)
Afferenza
(0,N)
(1,1)
Nome
Partecipazione
Composizione
(0,1)
Data
(1,N)
(1,N)
Progetto
Sede
Via
Indirizzo
Budget
Nome
CAP
Città

DIZIONARIO DEI DATI (ENTITÀ)

Entità	Descrizione	Attributi	Identificato
Impiegato	Dipendente dell’azienda	Codice, Cognome, Stipendio	Codice
Progetto	Progetti aziendali	Nome, Budget	Nome
Dipartimento	Struttura aziendale	Nome, Telefono	Città
Sede	Sede	Città,	Città

DIZIONARIO DEI DATI (RELATIONSHIP)

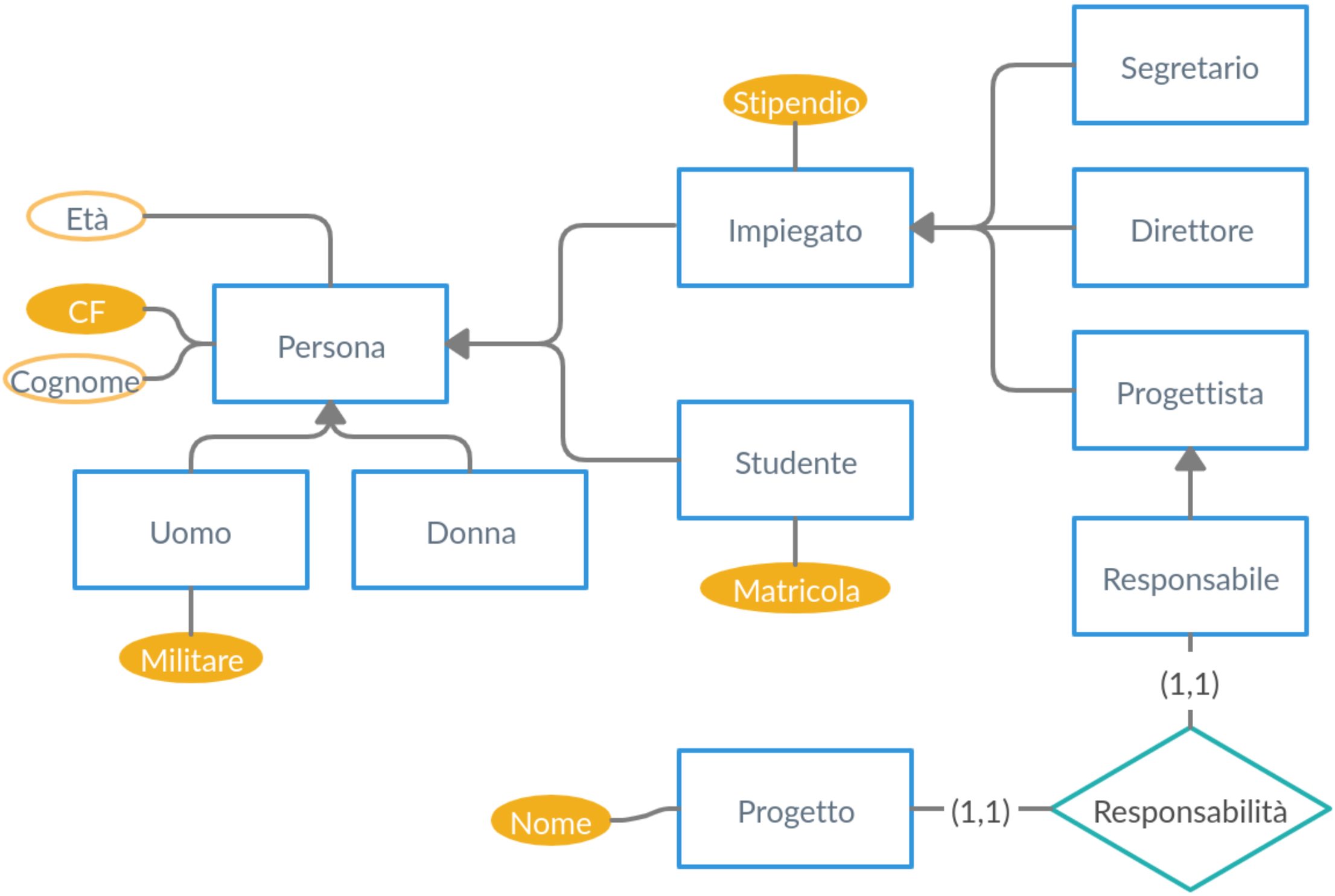
Relazioni	Descrizione	Componenti	Attr
Direzione	Direzione di dipartimento	Impiegato, Dipartimento, Stipendio	
Afferenza	Afferenza a dipartimento	Impiegato, Dipartimento	Dat
Partecipazione	Partecipazione a un progetto	Impiegato, Progetto	
Composizione	Composizione	Dipartimento,	

VINCOLI NON ESPRIMIBILI

Vincoli di integrità sui dati
(1) Il direttore di un dipartimento deve afferire a tale dipartimento
(2) Un impiegato non deve avere uno stipendio maggiore del direttore del suo dipartimento
(3) Un dipartimento con sede a Roma deve essere diretto da un impiegato con anzianità > 10
(4) Un impiegato che non afferisce a nessun dipartimento non deve partecipare a nessun

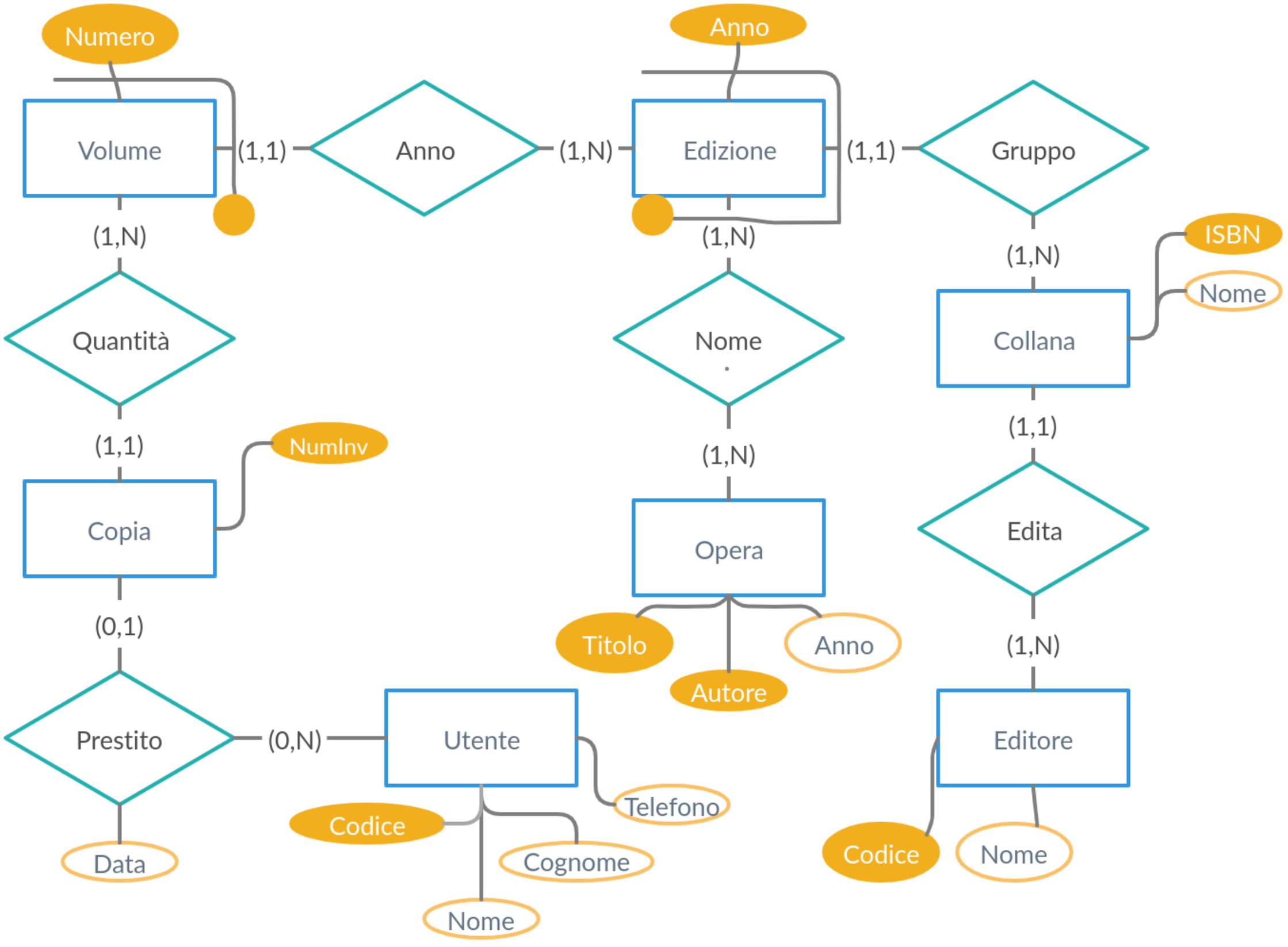
ESERCIZIO: ANAGRAFICA

Le persone hanno CF, cognome ed età; gli uomini anche la posizione militare; gli impiegati hanno lo stipendio e possono essere segretari, direttori o progettisti (un progettista può essere anche responsabile di progetto); gli studenti (che non possono essere impiegati) un numero di matricola; esistono persone che non sono né impiegati né studenti (ma i dettagli non ci interessano)


Segretario
Stipendio
Età
Impiegato
Direttore
CF
Persona
Progettista
Cognome
Studente
Uomo
Donna
Matricola
Responsabile
(1,1)
Militare
Nome
Progetto
(1,1)
Responsabilità

ESERCIZIO: BIBLIOTECA

- oggetto dei prestiti sono esemplari (detti anche copie) di singoli volumi, identificati attraverso un numero di inventario;
- ogni volume è relativo ad una specifica edizione (che può essere articolata in più volumi, anche in modo diverso dalle altre edizioni) di un'opera
- un volume può essere presente in più copie
- una edizione è caratterizzata dall'opera, dalla collana e dall'anno
    - riassumendo ed esemplificando, è possibile prendere in prestito la seconda copia del terzo volume de "I Miserabili", edizione Mondadori, collana Oscar, del 1975
- ogni collana ha un nome e un codice e un editore
- ogni editore ha un nome e un codice
- ogni opera ha un titolo, un autore e un anno di prima pubblicazione
- per ogni prestito in corso (quelli conclusi non interessano), sono rilevanti la data prevista di restituzione e l'utente (che può avere più volumi in prestito contemporaneamente), con codice identificativo, nome, cognome e recapito telefonico


Numero
Volume
(1,1)
Anno
(1,N)
Anno
Edizione
(1,1)
Gruppo
(1,N)
(1,N)
ISBN
Nome
Collana
Quantità
(1,1)
NumInv
Copia
(1,N)
Nome
(1,N)
Opera
(1,1)
Edita
(0,1)
Titolo
Anno
Autore
(1,N)
Prestito
(0,N)
Utente
Editore
Telefono
Codice
Cognome
Codice
Nome
Data
Nome

STRATEGIE DI PROGETTO

TOP-DOWN

BOTTOM-UP

TOP-DOWN

- definisco il problema in linea generale
- rifinisco le varie parti sempre di più
- contro: devo finire tutta la progettazione prima di poter iniziare
- pro: completa comprensione del sistema

BOTTOM-UP

- parti individuali sono specificate in dettaglio
- connessione tra le varie parti
- contro: posso fare degli errori nella progettazione dei singoli moduli
- pro: posso iniziare subito a sviluppare (e testare)

ANALISI DEI REQUISITI

DIVERSE ATTIVITÀ (INTERCONNESSE)

- acquisizione dei requisiti
- analisi dei requisiti
- costruzione dello schema concettuale
- costruzione del glossario

ANALISI DEI REQUISITI

POSSIBILI FONTI

- utenti, attraverso interviste
- documentazione esistente: normative, regolamenti interni, procedure aziendali e realizzazioni preesistenti
- modulistica

ACQUISIZIONE PER INTERVISTE

Il reperimento dei requisiti è un’attività difficile e non standardizzabile

- utenti diversi possono fornire informazioni diverse
- verificare la coerenza, chiedere esempi
- utenti a livello più alto hanno spesso una visione più ampia ma meno dettagliata
- le interviste portano spesso ad un’acquisizione dei requisiti per raffinamenti successivi

SCRIVERE I REQUISITI

REGOLE GENERALI

- standardizzare la struttura delle frasi
- suddividere le frasi articolate
- separare le frasi sui dati da quelle sulle funzioni
- costruire un glossario dei termini
- individuare omonimi e sinonimi e unificare i termini

ESEMPIO: SOCIETÀ DI FORMAZIONE

Società di formazione (1)

Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti. Per gli studenti (circa 5000), identificati da un codice, si vuole memorizzare il codice fiscale, il cognome, l'età, il sesso, il luogo di nascita, il nome dei loro attuali datori di lavoro, i posti dove hanno lavorato in precedenza insieme al periodo, l'indirizzo e il numero di telefono, i corsi che hanno frequentato (i corsi sono in tutto circa 200) e il giudizio finale.

ESEMPIO: SOCIETÀ DI FORMAZIONE

Società di formazione (2)

Rappresentiamo anche i seminari che stanno attualmente frequentando e, per ogni giorno, i luoghi e le ore dove sono tenute le lezioni. I corsi hanno un codice, un titolo e possono avere varie edizioni con date di inizio e fine e numero di partecipanti. Se gli studenti sono liberi professionisti, vogliamo conoscere l'area di interesse e, se lo possiedono, il titolo. Per quelli che lavorano alle dipendenze di altri, vogliamo conoscere invece il loro livello e la posizione ricoperta.

ESEMPIO: SOCIETÀ DI FORMAZIONE

Società di formazione (3)

Per gli insegnanti (circa 300), rappresentiamo il cognome, l'età, il posto dove sono nati, il nome del corso che insegnano, quelli che hanno insegnato nel passato e quelli che possono insegnare. Rappresentiamo anche tutti i loro recapiti telefonici. I docenti possono essere dipendenti interni della società o collaboratori esterni.

GLOSSARIO DEI TERMINI

Termine	Descrizione	Sinonimi	Collegamenti
Partecipante	Persona che partecipa ai corsi	Studente	Corso, Società
Docente	Docente dei corsi. Può essere esterno	Insegnante	Corso
Corso	Corso organizzato dalla società. Può avere più edizioni	Seminario	Docente
Società	Ente presso cui i partecipanti lavorano o hanno lavorato	Posti	Partecipante

GRUPPI DI FRASI OMOGENEE

Frasi di carattere generale

Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti.

GRUPPI DI FRASI OMOGENEE

Frasi relative ai partecipanti

Per i partecipanti (circa 5000), identificati da un codice, rappresentiamo il codice fiscale, il cognome, l'età, il sesso, la città di nascita, i nomi dei loro attuali datori di lavoro e di quelli precedenti (insieme alle date di inizio e fine rapporto), le edizioni dei corsi che stanno attualmente frequentando e quelli che hanno frequentato nel passato, con la relativa votazione finale in decimi.

GRUPPI DI FRASI OMOGENEE

Frasi relative ai datori di lavoro

Relativamente ai datori di lavoro presenti e passati dei partecipanti, rappresentiamo il nome, l'indirizzo e il numero di telefono.

GRUPPI DI FRASI OMOGENEE

Frasi relative ai corsi

Per i corsi (circa 200), rappresentiamo il titolo e il codice, le varie edizioni con date di inizio e fine e, per ogni edizione, rappresentiamo il numero di partecipanti e il giorno della settimana, le aule e le ore dove sono tenute le lezioni.

GRUPPI DI FRASI OMOGENEE

Frasi relative a tipi specifici di partecipanti

Per i partecipanti che sono liberi professionisti, rappresentiamo l'area di interesse e, se lo possiedono, il titolo professionale. Per i partecipanti che sono dipendenti, rappresentiamo invece il loro livello e la posizione ricoperta.

GRUPPI DI FRASI OMOGENEE

Frasi relative ai docenti

Per i docenti (circa 300), rappresentiamo il cognome, l'età, la città di nascita, tutti i numeri di telefono, il titolo del corso che insegnano, di quelli che hanno insegnato nel passato e di quelli che possono insegnare. I docenti possono essere dipendenti interni della società di formazione o collaboratori esterni.

DAI REQUISITI A E-R

- **se ha proprietà significative e descrive oggetti con esistenza autonoma**
  - entità
- **se è semplice e non ha proprietà**
  - attributo
- **se correla due o più concetti**
  - relazione
- **se è caso particolare di un altro**
  - generalizzazione

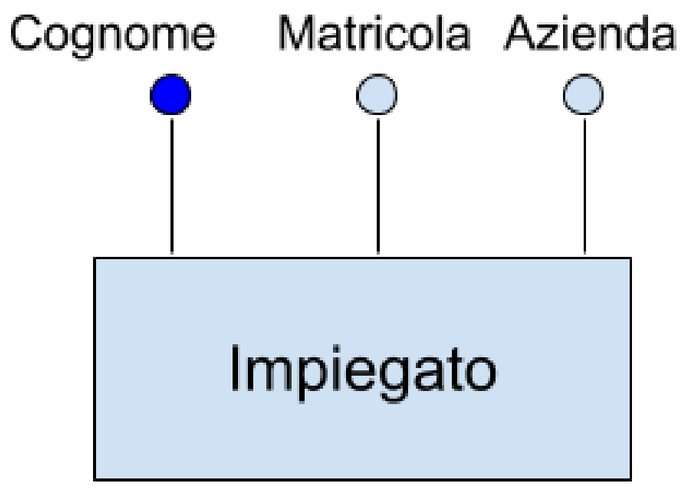
DESIGN PATTERNS

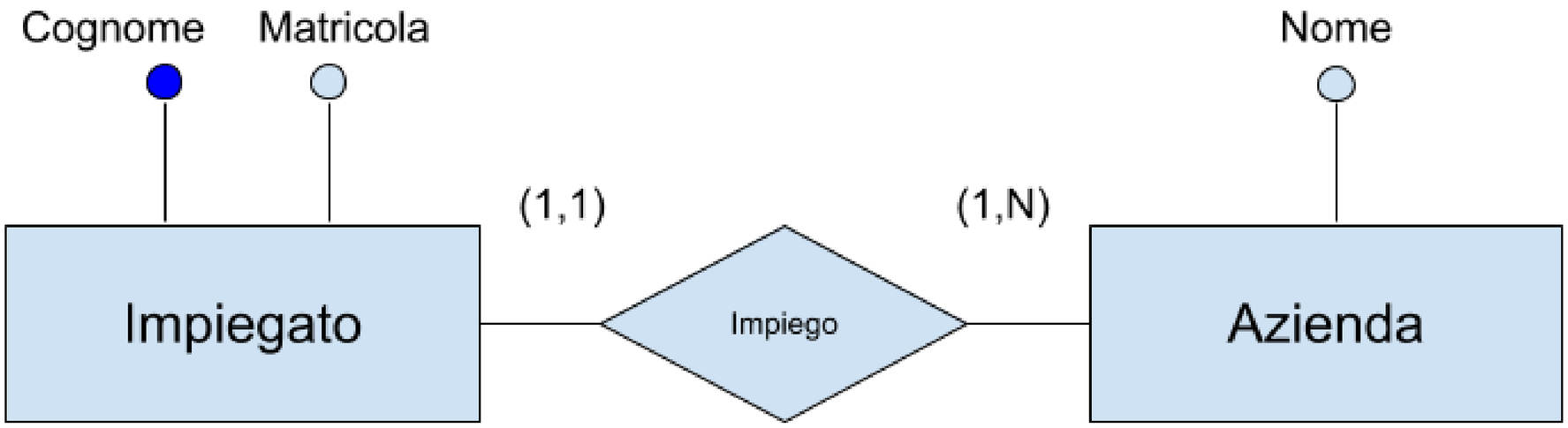
SOLUZIONI PROGETTUALI A PROBLEMI COMUNI

LARGAMENTE USATI NELL'INGEGNERIA DEL SOFTWARE

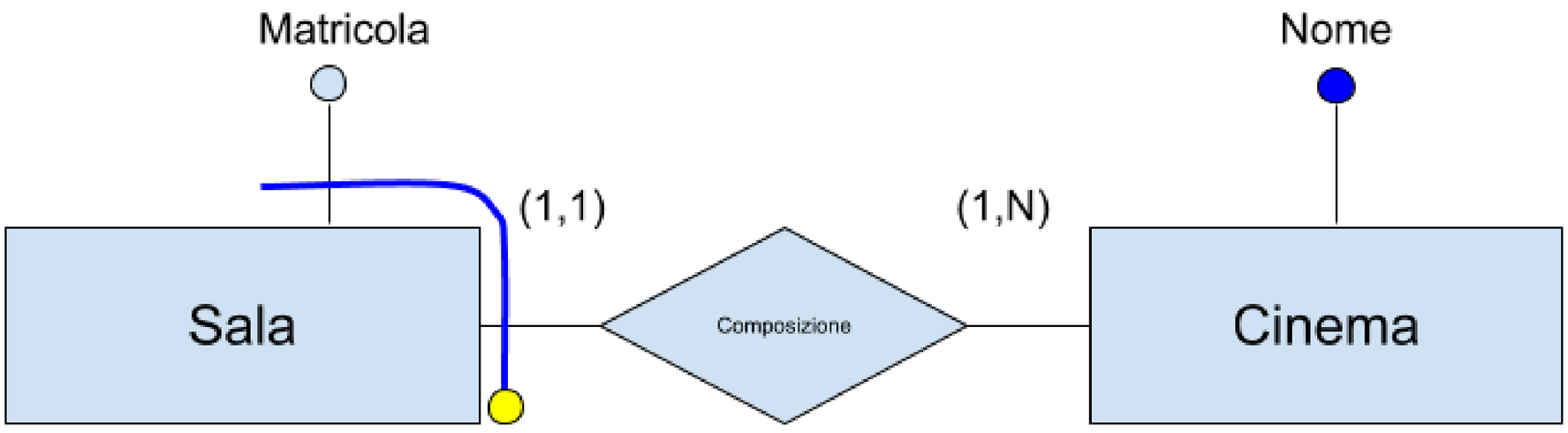
Design Patterns: Elements of Reusable Object-Oriented Software (1997)

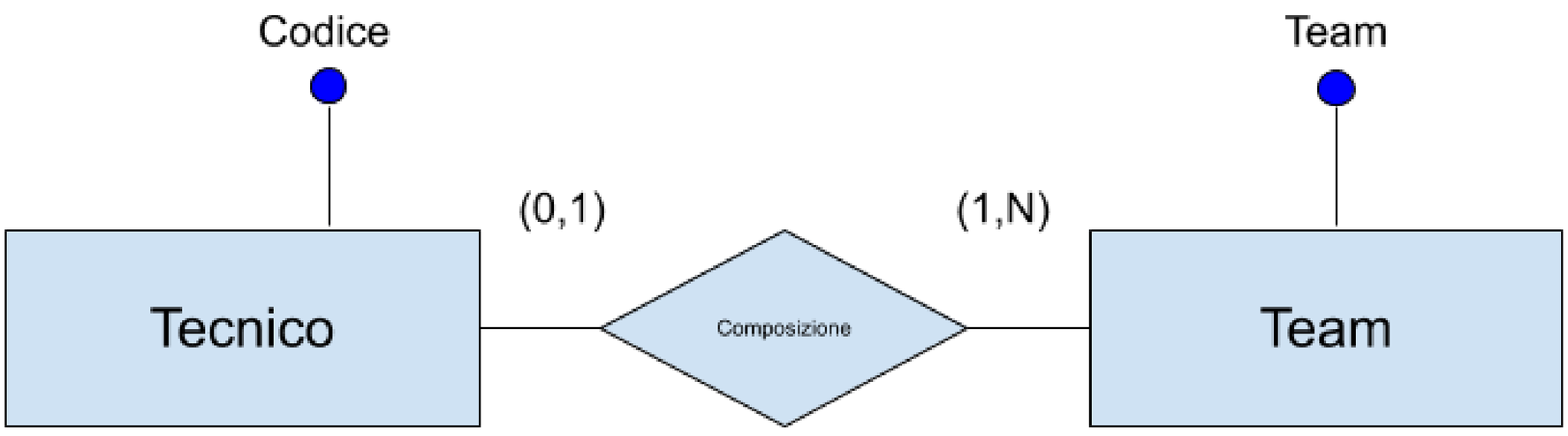
REIFICAZIONE DI ATTRIBUTO DI ENTITÀ



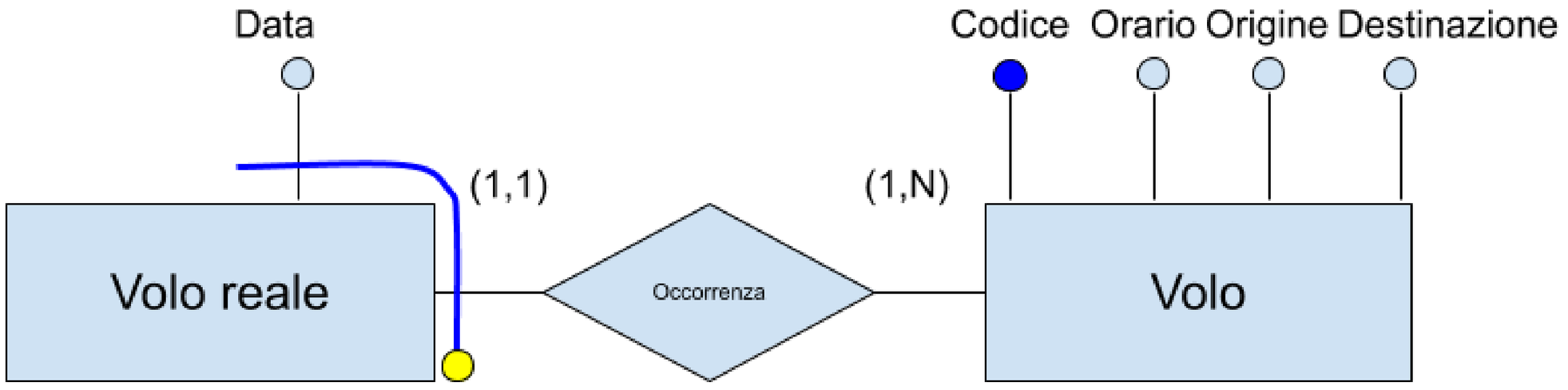


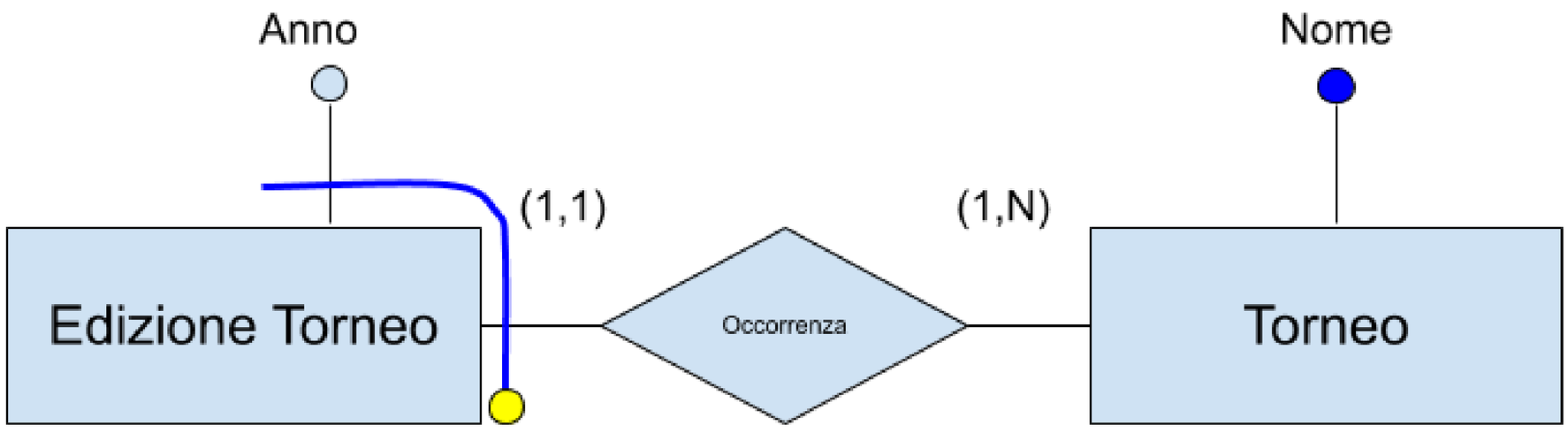
PART-OF


Matricola
Nome
(1,1)
(1,N)
Sala
Composizione
Cinema


Codice
Team
(0,1)
(1,N)
Tecnico
Composizione
Team

INSTANCE-OF


Data
Codice
Orario
Origine
Destinazione
(1,1)
(1,N)
Volo reale
Occorrenza
Volo


Anno
Nome
(1,1)
(1,N)
Edizione Torneo
Occorrenza
Torneo