IL MODELLO RELAZIONALE (CAPITOLO 2)

Codd 1970

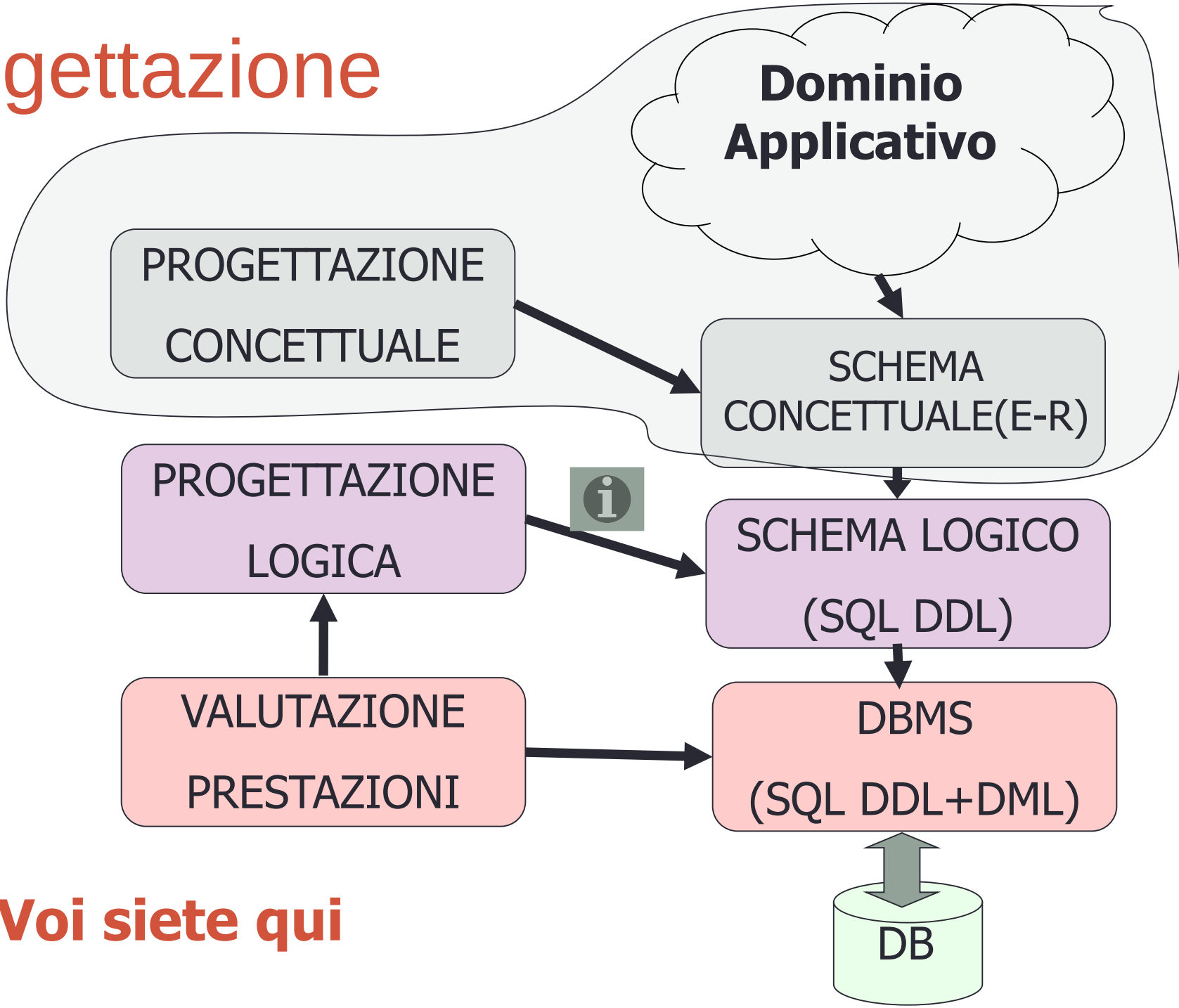
Indipendenza dei dati

Distinzione nella descrizione dei dati tra livello fisico e livello logico

Vendors

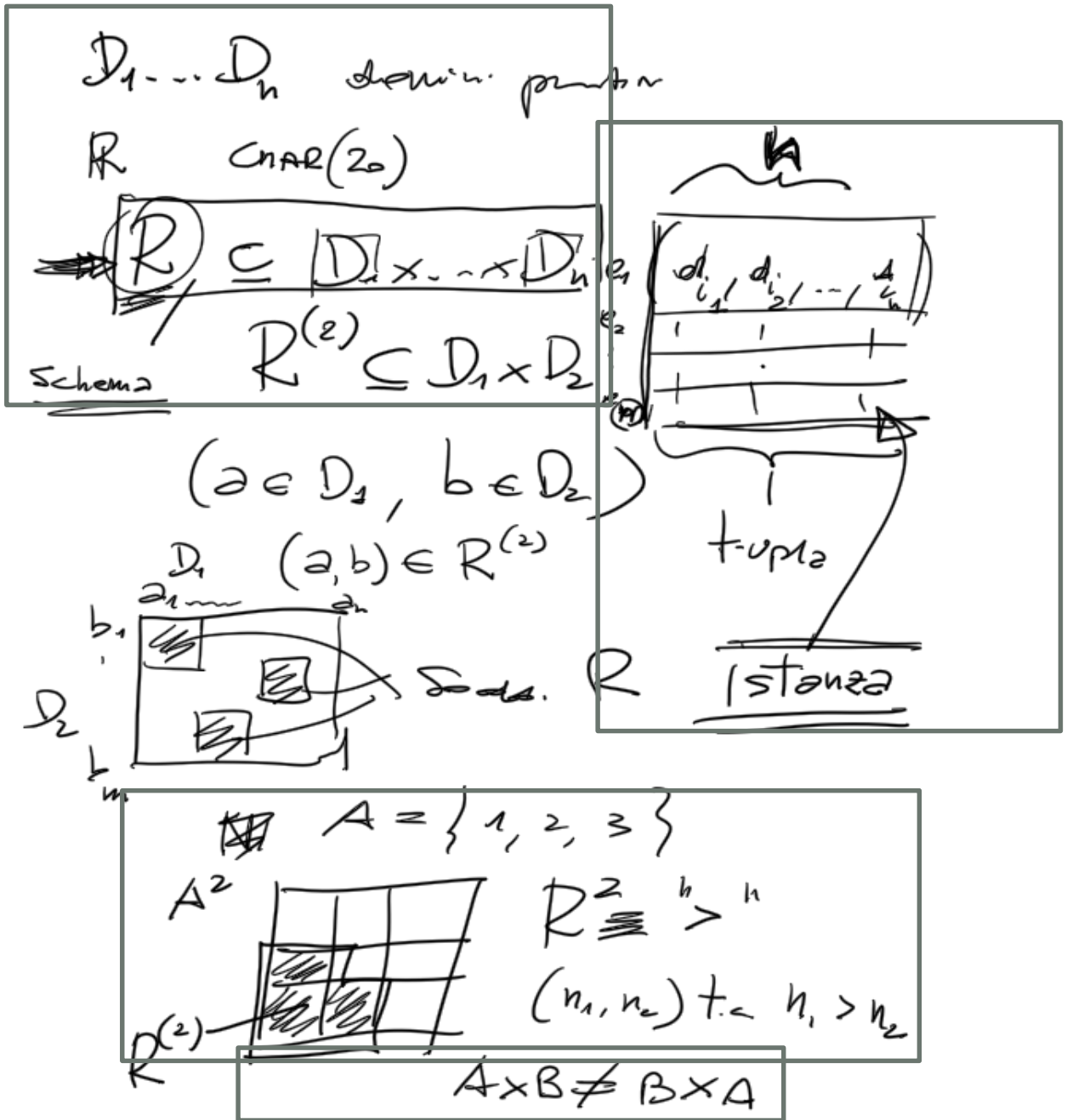
IBM,Informix,Microsoft,Oracle,Sybase

Progettazione



Voi siete qui

Relazioni:
una prospettiva … algebrica



Database Relazionale : Definizioni

- **Database Relazionale**: Un insieme di relazioni

- **Relazione**: insieme di righe o tuple distinte

  - **Istanza**: una TABELLA con righe e colonne
  - **Cardinalità** = numero di righe
  - **Grado** = numero di campi (colonne)

  - **Schema**: specifica il nome della relazione, il nome ed il tipo di ogni colonna

Esempio: Relazione Studenti

- Istanza

sid	nome	login	età	gpa
0012	Rossi	ro@ec	18	3.4
0072	Bianchi	bi@ec	19	3.2
0033	Bianchi	bi@tt	18	3.8

  - cardinalita’ 3
  - grado 5

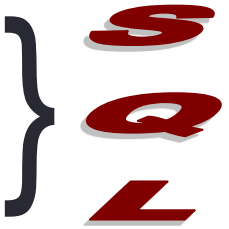
Schema

```
Studenti( sid:string,
          nome:string,
          login:string,
          età:integer,
          gpa:real)
```

- Non e’ definito alcun ordinamento tra le *n*-uple/righe
- Le *n*-uple/righe di una relazione sono distinte
  - Due *n*-ple uguali (per tutti i valori) sono LA STESSA *n*-pla
- Ciascuna *n*-upla e’ ordinata al suo interno; all’i-esimo valore corrisponde l’i-esimo dominio definito nello schema (Constraint/Vincolo di Dominio)

Linguaggi di interrogazione

- Uno dei vantaggi principali del modello relazionale è che esso supporta un intuitivo **metodo di interrogazione**.
- Le interrogazioni possono essere scritte intuitivamente e successivamente analizzate dal DBMS
- Due tipologie di linguaggi di interrogazione:
  - DML Data Manipulation Language
  - DDL Data Definition Language



Il linguaggio di interrogazione SQL

S

sid	nome	login	eta'	gpa
0012	Rossi	ro@ec	18	3.4
0072	Bianchi	bi@ec	19	3.2
0033	Bianchi	bi@tt	18	3.8

```
SELECT *
FROM Studenti S
WHERE S.eta'=18
```

sid	nome	login	eta'	gpa
0012	Rossi	ro@ec	18	3.4
0033	Bianchi	bi@tt	18	3.8

Estrae dalla tabella S tutti gli studenti diciottenni!

Il linguaggio di interrogazione SQL

S

sid	nome	login	eta'	gpa
0012	Rossi	ro@ec	18	3.4
0072	Bianchi	bi@ec	19	3.2
0033	Bianchi	bi@tt	18	3.8

P

sid	cid	grade	gpa
0012	Mate1	C	3.4
0012	Bd1	A	3.2
0033	Bd1	A	3.8

```
SELECT S.nome,P.cid
FROM Studenti S,
          pianoDS P
WHERE S.sid=P.sid
AND P.grade='A'
```

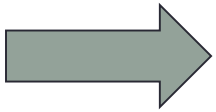
S.nome	P.cid
Rossi	Bd1
Bianchi	Bd1

Vincolo di integrità referenziale

Creazione di Tabelle in SQL

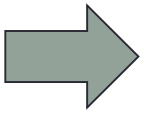
- Creare la relazione

  Studenti



```
CREATE TABLE Studenti
      (sid:CHAR(20),
       nome: CHAR(10),
       login: CHAR(10),
       eta': INTEGER,
       gpa:REAL)
```

- Creare la tabella **PianoDS** contenente informazioni riguardanti i corsi che ogni studente segue



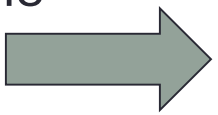
```
CREATE TABLE PianoDS
      (sid: CHAR(20),
       cid:CHAR(20),
       grade: CHAR(2))
```

Cancellazione e Modifica di Relazioni

- Cancellare la relazione

  `Studenti`.

- L'informazione relativa allo schema ed alle n-uple viene cancellato



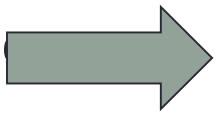
```
DELETE TABLE Studenti
```

- Modificare lo schema della relazione **`Studenti`**

  aggiungendo il campo

  `primoAnno`.

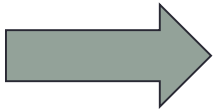
- (Ogni n-upla dell'istanza viene estesa con un valore **`null`** in corrispondenza del nuovo



```
ALTER TABLE Studenti
ADD COLUMN primoAnno
```

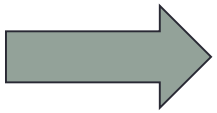
Inserimento e Cancellazione di tuple

- Inserire una tupla nella tabella Studenti.
- (ATTENZIONE:Vincolo di Dominio)



```
INSERT INTO  TABLE Studenti
(sid,nome,login,eta',gpa)
 VALUES ('566','Verdi',
                've@ee' ,18,3.2)
```

- Cancellare tutte le tuple che soddisfano una certa condizione



```
DELETE
FROM Studenti S
WHERE S.nome='Verdi'
```

Vincoli di Integrità (ICs)

- **Vincolo di Integrità**
  - Condizione che deve essere soddisfatta da ogni istanza del database
- Il Vincolo di Integrità viene:
  - **Definito in fase di progettazione**
  - **Verificato ogni volta che la relazione viene modificata**
- Una istanza di una relazione è legale se soddisfa tutti i vincoli di integrità
- Il DBMS scarta le operazioni che non rispettano i vincoli di integrità

Vincolo di Chiave

- Un insieme ***K*** di campi di una relazione ***r*** e':
- **Superchiave** di ***r*** se **K** identifica in maniera univoca una t-upla
  - Esempio: K=**`(sid, name)`** in **`Studenti`**
- **Chiave** per una relazione ***r*** se ***K*** e' superchiave minimale, cioè non esiste alcun altra superchiave ***K'*** di ***r*** che sia contenuta in ***K*** come sottoinsieme proprio
  - Esempio: K=**`(sid)`** in **`Studenti`**
- **Chiave Primaria** la chiave scelta tra più chiavi candidate

SQL: Vincolo di chiave

```
CREATE TABLE Studenti
      (sid: CHAR(20),
       nome: CHAR(10),
       login: CHAR(10),
       età : INTEGER,
       gpa: REAL
      UNIQUE (nome,età),
      CONSTRAINT Matricola PRIMARY KEY (sid))
```

VINCOLO DI CHIAVE

VINCOLO di CHIAVE PRIMARIA

Vincolo di Integrità Referenziale (*Foreign Key Constraints*)

- **Foreign Key**: insieme di campi in una relazione che fanno riferimento a campi di un'altra relazione.
  - **Vincolo di integrità referenziale** fra un insieme di attributi X di una relazione R1 ed un'altra relazione R2
  - è **soddisfatto se i valori su X di ciascuna tupla dell'istanza di R1 compaiono come valori della chiave (primaria) dell'istanza di R2**.
-

SQL: Foreign Key

```
CREATE TABLE PianoDS
   (sid: CHAR(20),
    cid: CHAR(20),
    grade CHAR(2),
    PRIMARY KEY (sid,cid),
    FOREIGN KEY(sid) REFERENCES Studenti)
```

P

S

sid	cid	grade
0012	Mate1	C
0012	Bd1	A
0033	Bd1	B

sid	nome	login	eta'	gpa
0012	Rossi	ro@ec	18	3.4
0072	Bianchi	bi@ec	19	3.2
0033	Bianchi	bi@tt	18	3.8

Cosa accade se si tenta di inserire in **PianoDS** una tupla con sid non presente nella tabella **Studenti**?

E se viene cancellata una tupla di **S** utilizzata da **PianoDS**?

SQL: Integrity Constraint

Su DELETE ed UPDATE 4 operazioni:

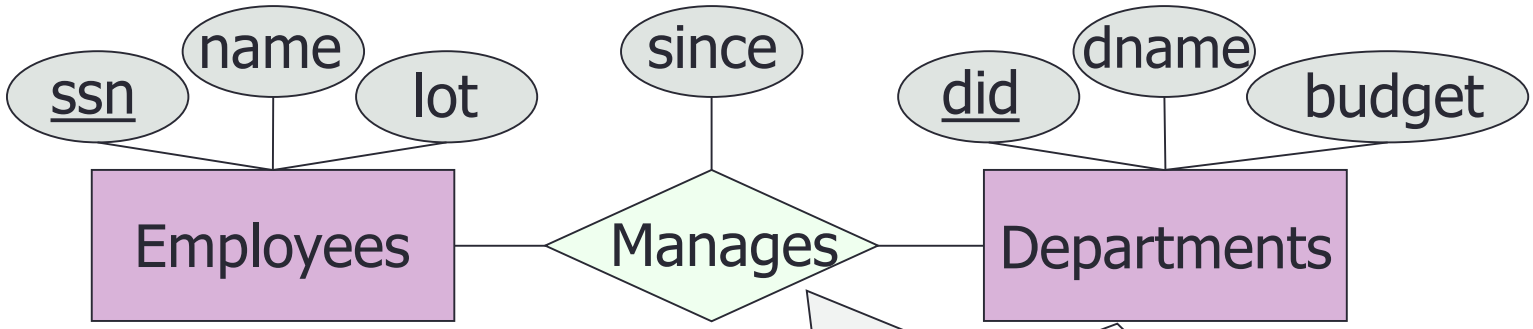
- CASCADE (delete tuple e tutte quelle che fanno riferimento ad essa)
- NO ACTION (delete/update rifiutati)
- SET NULL/DEFAULT

```
CREATE TABLE Iscrizioni
(studid: CHAR(20),
 cid:CHAR(20),
 grade: CHAR(2),
 PRIMARY KEY(studid,cid),
 FOREIGNB KEY (studid)
 REFERENCES Studenti,
  ON DELETE CASCADE,
  ON UPDATE NO ACTION)
```

Vincoli di Integrità

- I Vincoli di integrità (ICs) aiutano ad imporre una interpretazione ai dati che rispetta la semantica (il significato) delle informazione nel dominio della applicazione
- E’ possibile dire se una certa istanza verifica (viola) tutti gli ICs ma la verita’ di un ICs non dipende mai da UNA SOLA istanza:
  - Un IC deve valere per tutte le istanze possibili (la presente e le future!)

Disegno dello schema Logico-
Da schema E-R a Relazionale

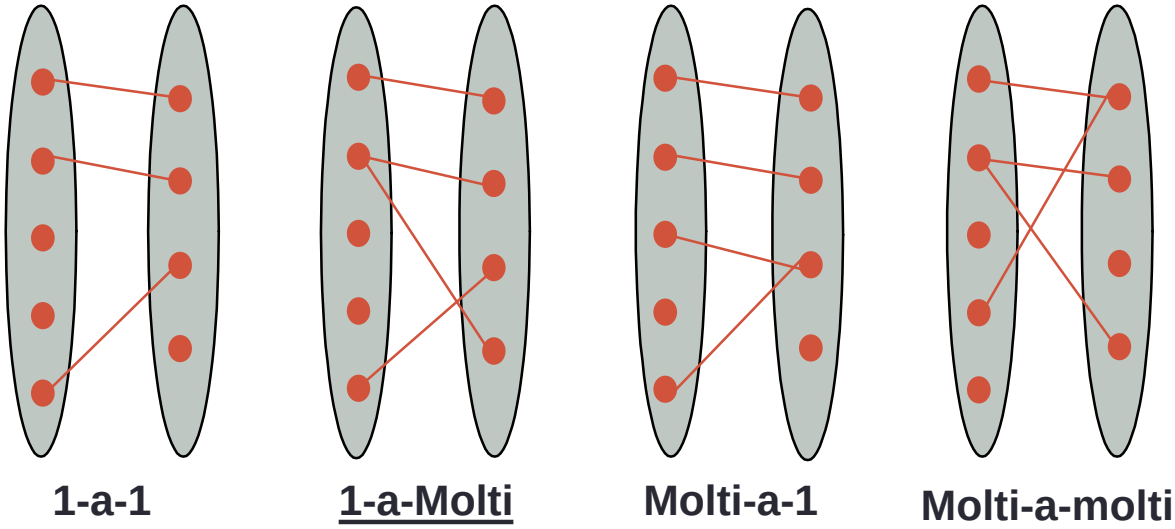


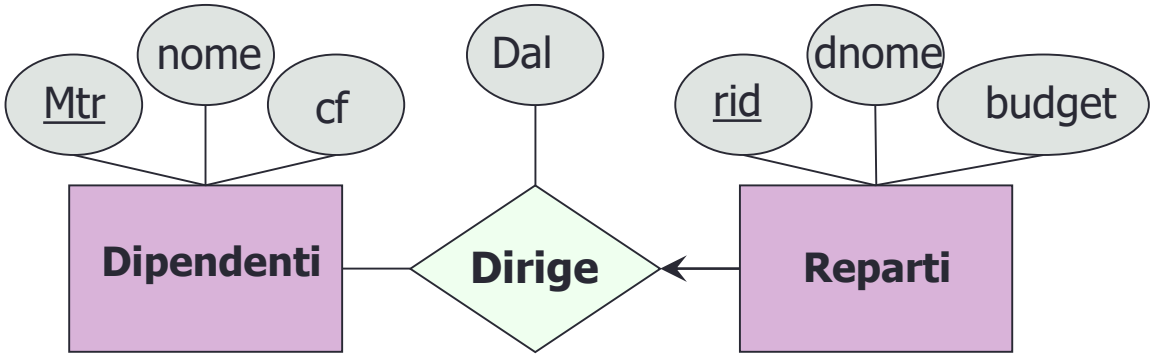

```
CREATE TABLE Employees
        (ssn CHAR(11),
         name CHAR(20),
         lot INTEGER,
        PRIMARY KEY (ssn)
        )
```

```
CREA
                                    REFERENCES Employees,
        FOREIGN KEY (did)
          REFERENCES Departments)
```

```
CREATE TABLE Departments
        (did INTEGER,
         dname CHAR(20),
         budget REAL,
        PRIMARY KEY (did)
        )
```

Vincolo di CHIAVE (remind)



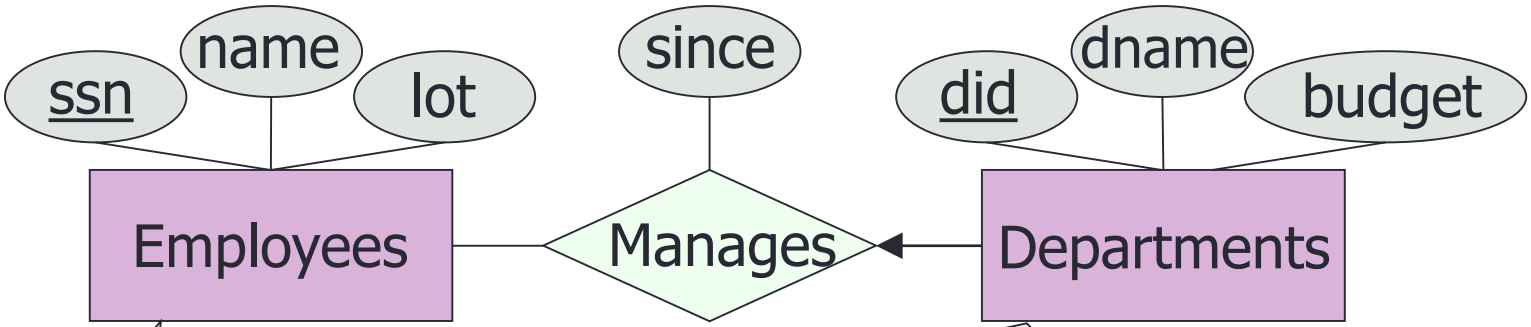


VINCOLO DI CHIAVE:
Ogni Reparto può essere gestito al più da un dipendente.

Esempi di Vincoli di Chiave

- Molti a uno:
  - GiocatoreSerie_A – gioca_per - Squadra
  - Album – contiene – TitleTrack
- Uno-a-Molti
  - Scrittore – scrive – libro
- Molti-a-Molti
  - Studente – segue – Corso
  - Compilation – contiene - Canzone

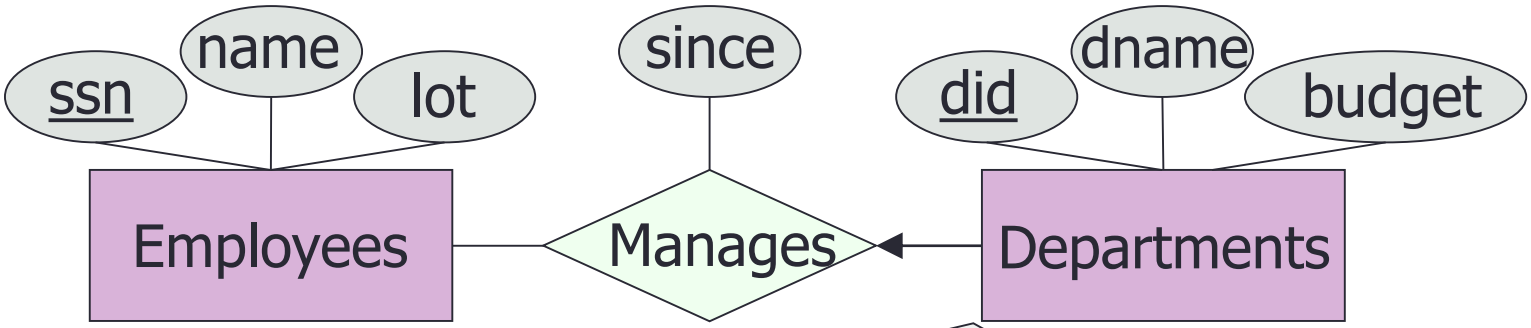
Da schema E-R con Vincolo di Chiave ⇒ schema Relazionale




```
CREATE TABLE Employees
        (ssn CHAR(11),
         name CHAR(20),
         lot INTEGER,
         PRIMARY KEY (ssn)
        )
```

```
CREATE TABLE Dep_Mgr
        ( did INTEGER,
          dname CHAR(20),
          budget REAL,
          ssn    CHAR(11),
          since   DATE,
          PRIMARY KEY (did),
          FOREIGN KEY (snn)
             REFERENCES Employees)
```

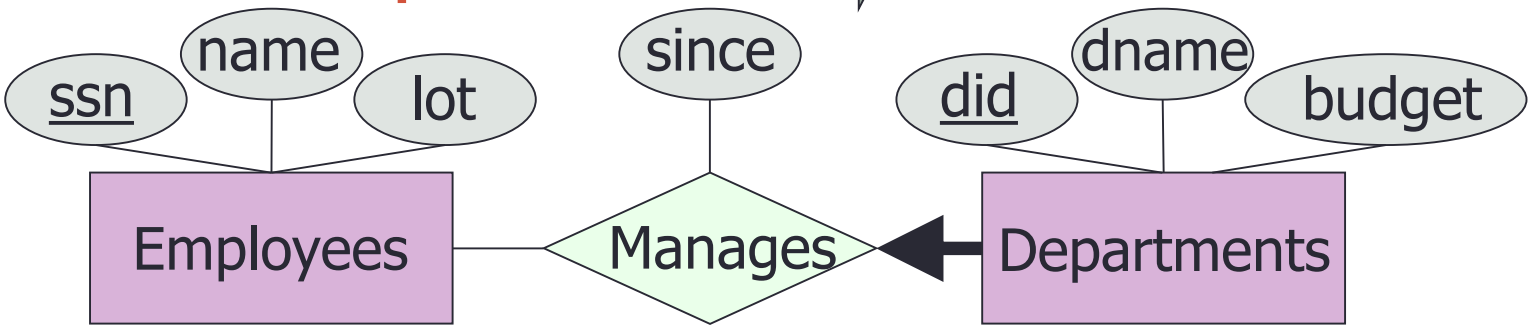
Da schema E-R con Vincolo di Chiave schema Relazionale



La tabella Dep_Mgr rappresenta sia l'entita' Deps che la relazione Manages con (l'unico) manager rappresentato tramite l'attributo **ssn**

```
CREATE TABLE Dep_Mgr
      ( did INTEGER,
        dname CHAR(20),
        budget REAL,
        ssn    CHAR(11),
        since   DATE,
        PRIMARY KEY (did),
        FOREIGN KEY (snn)
           REFERENCES Employees)
```

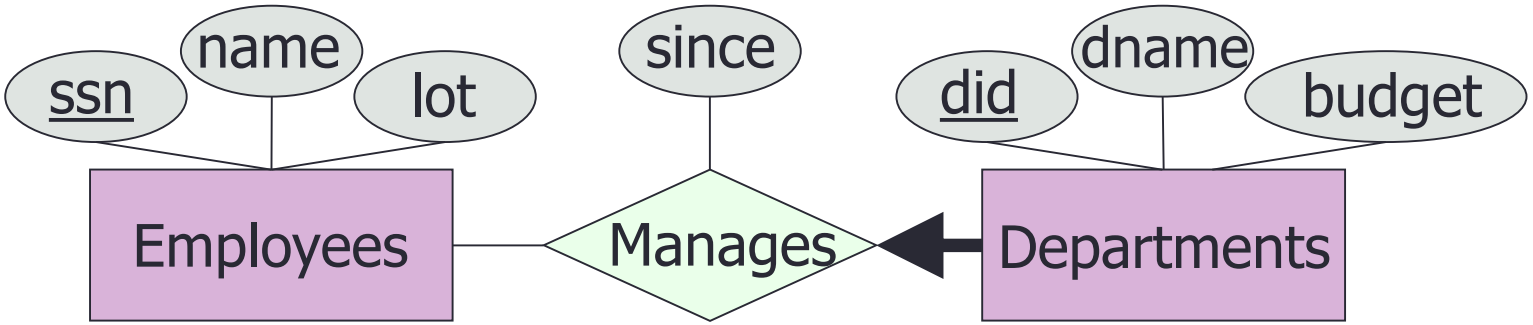
Da schema E-R con Vincolo di Partecipazione ⇒ schema Relazionale




```
CREATE TABLE Employees
        (ssn CHAR(11),
         name CHAR(20),
         lot INTEGER,
         PRIMARY KEY (ssn)
        )
```

```
CREATE TABLE Dep_Mgr
        (did INTEGER,
         dname CHAR(20),
         budget REAL,
         ssn CHAR(11) NOT NULL,
          since   DATE,
          PRIMARY KEY (did),
          FOREIGN KEY (snn)
            REFERENCES Employees
             ON DELETE NO ACTION)
```

Da schema E-R con Vincolo di Partecipazione ⇨ schema Relazionale

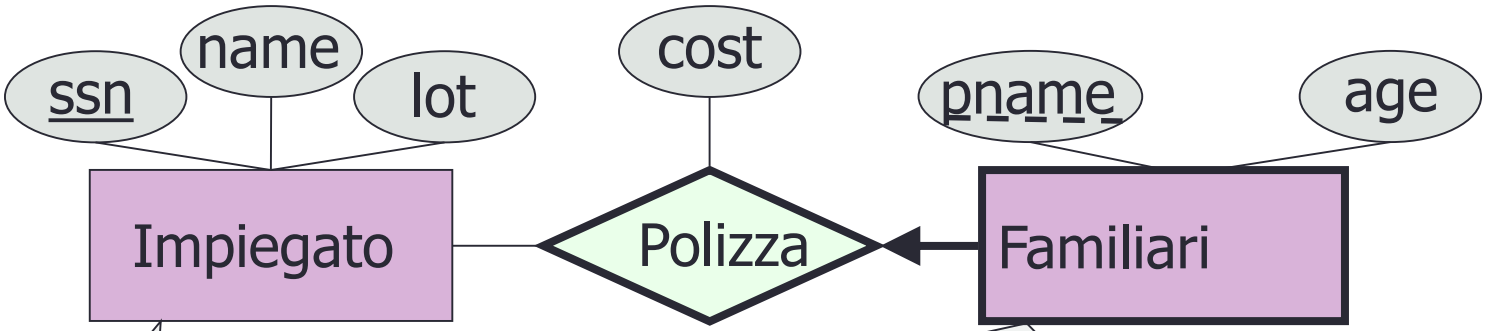


Il vincolo NOT NULL impone che nessuna tupla di Dep_Mgr possa escludere il manager sottotinteso. Questo (insieme al vincolo di foreign key) realizza la semantica di partecipazione totale.

```
CREATE TABLE Dep_Mgr
    did INTEGER,
    name CHAR(20),
    budget REAL,
    snn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (snn)
      REFERENCES Employees
      ON DELETE NO ACTION)
```

Da schema E-R con Weak Entity set

schema Relazionale




```
CREATE TABLE Impiegato
        (ssn CHAR(11),
         name CHAR(20),
         lot INTEGER,
         PRIMARY KEY (ssn)
        )
```

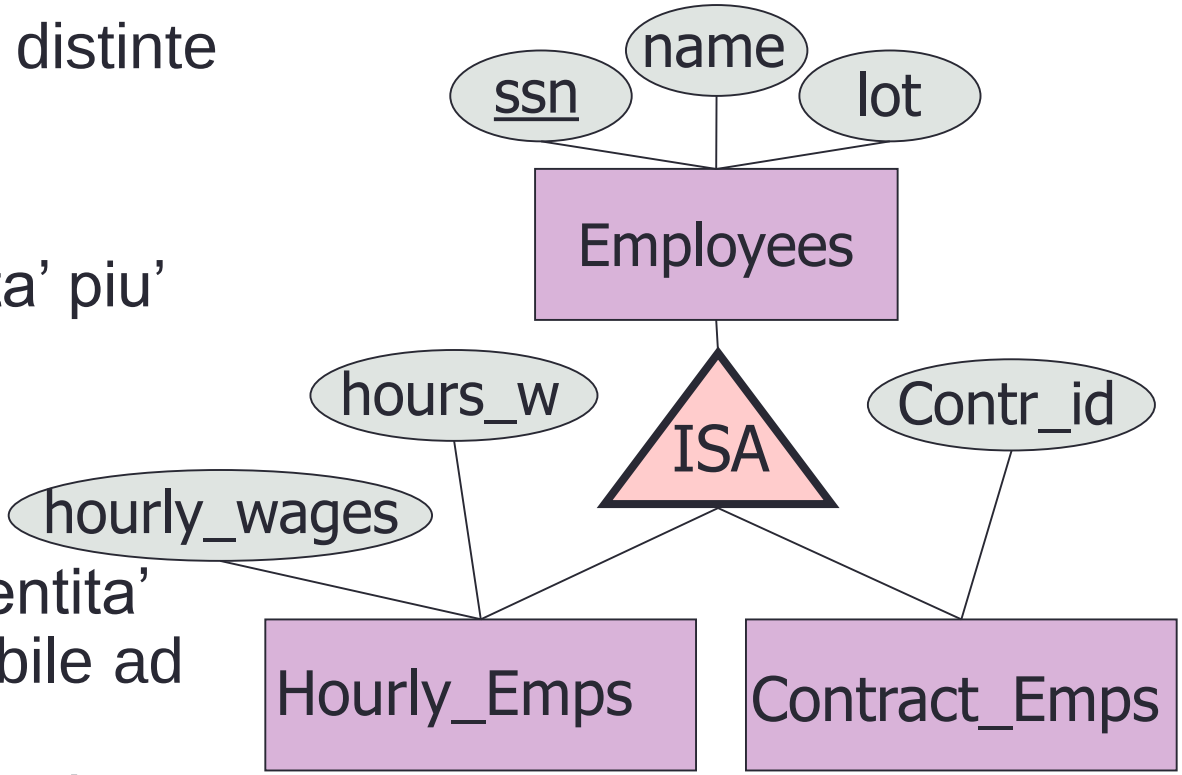
```
CREATE TABLE Polizza_Familiare
        (pname CHAR(20),
         age INTEGER,
         cost      REAL,
         ssn CHAR(11) NOT NULL,
         PRIMARY KEY (pname,ssn),
         FOREIGN KEY (snn)
              REFERENCES Employees
                 ON DELETE CASCADE)
```

Da schema E-R con gerarchia ISA ⇒ schema Relazionale

- Tre approcci:
- Mantengo le tre entità distinte

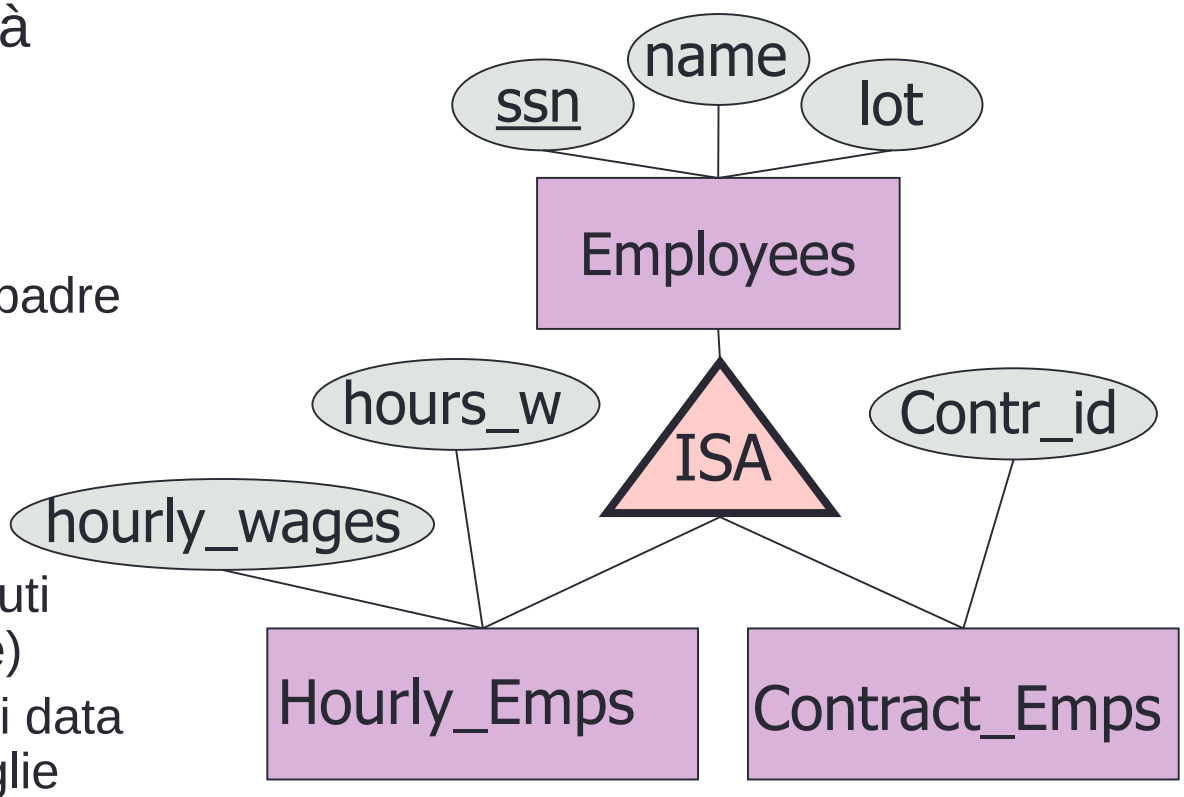
- Collasso verso l'alto mantenendo solo l'entita' piu' generale

- Collasso verso il basso mantenendo SOLO le entita' specifiche (non applicabile ad ISA che ricoprono solo parzialmente la entita' piu' generale)



Da schema E-R con gerarchia ISA ⇒ schema Relazionale

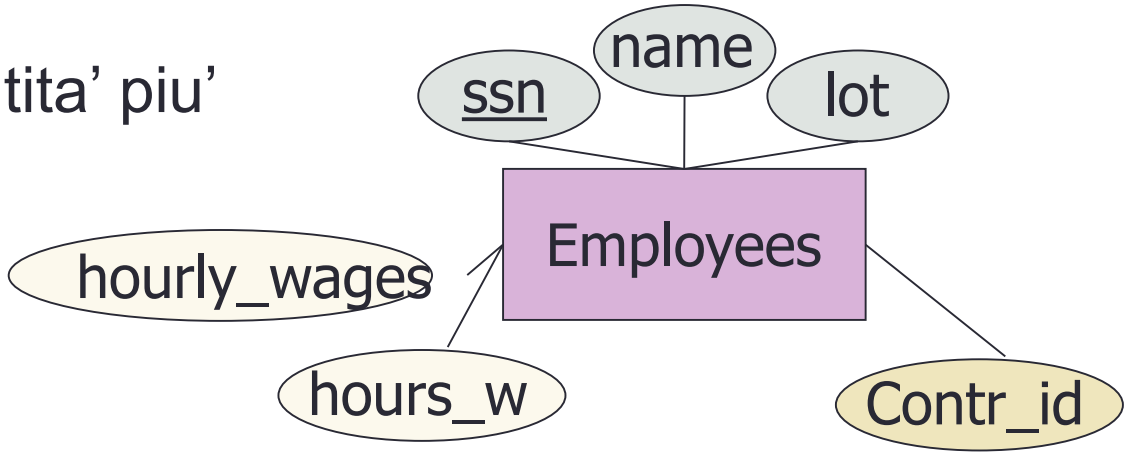
- Tre approcci (1):
- Mantengo le tre entità distinte
  - Tre tabelle
  - Stessa chiave primaria
  - Foreign key dai figli al padre



- Utile quando:
  - Molti accessi agli attributi condivisi (tabella padre)
  - Molti attributi specifici (i data records delle tabelle figlie hanno una taglia significativa)

Da schema E-R con gerarchia ISA schema Relazionale

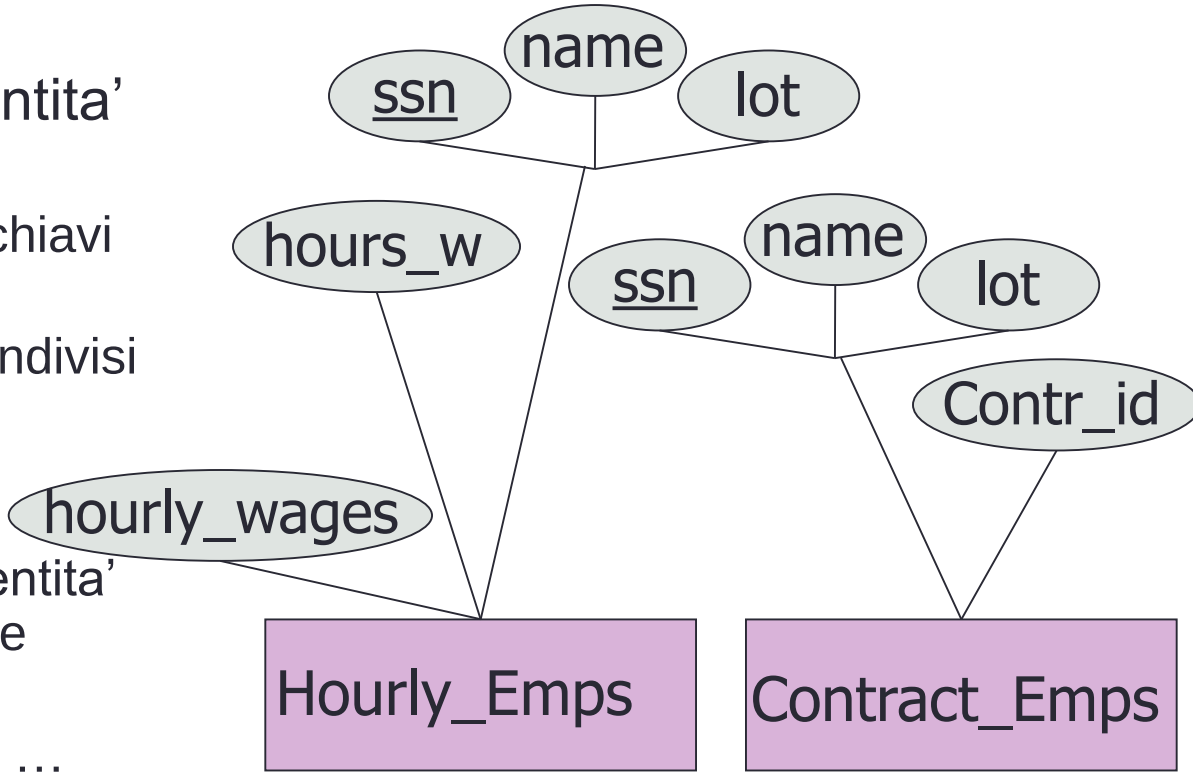
- Tre approcci (2):
- Collasso verso l'alto mantenendo solo l'entita' piu' generale
  - Una tabella
  - Tutti gli attributi



- Utile
  - Nessuna preferenza nell'accesso all'una o all'altra entita' specifica
  - Pochi attributi distinguono le due entità figlie => basta un solo data record poiché non c'e' molta ridondanza

Da schema E-R con gerarchia ISA ⇒ schema Relazionale

- Tre approcci (3):
- Collasso verso il basso mantenendo SOLO le entita' specifiche
  - Due tabelle distinte con chiavi primarie indipendenti
  - Si ripetono gli attributi condivisi
- Utile se:
  - Tipi di accesso alle due entita' specifiche completamente differenti e …
  - Pochi attributi condivisi e …
  - La ISA non ricopre parzialmente la entita' piu' generale



Le Viste (1)

Una **Vista** è una relazione definita non in termini di tuple, quindi non esplicitamente registrata in un database, ma calcolata da una definizione (di DDL).

```
CREATE VIEW StudentiInCorso (nome, corso)
     AS SELECT S.nome, P.corso
FROM Studenti S, PianoDS P
WHERE S.sid=P.sid AND S.age<21
```

Le viste (2)

- Le viste rappresentano dei **dizionari alternativi** per le applicazioni
- Alle relazioni dello schema logico (materializzate nelle tabelle di un DB) vengono sostituite **nuove relazioni** che "**virtualizzano**" le informazioni sottostanti:
  - Ne consentono un uso selettivo, in modo **da separare informazioni distinte delle stesse entità per diverse applicazioni**
  - Ne nascondono **dettagli irrilevanti**
  - Ne nascondono **dettagli critici** per aumentare la sicurezza
  - Ne **migliorano l'astrazione**, poiche' possono esprimere relazioni piu' ricche (nell'esempio "**`StudentiInCorso`**" e' una informazione piu' specifica di **`Studenti`** e di **`Piano di Studio`**)

Le viste (3)

- Alle viste possono essere assegnati dei diritti d’accesso (GRANTS) selettivi per le diverse applicazioni

- Per rimuovere una vista:

```
DROP VIEW <nome_view>
```

Le Viste: Vantaggi

- **Indipendenza dei dati**
- Una vista presenta le informazioni di interesse per l'applicazione, viene interrogata come le tabelle base. Alle variazioni irrilevanti delle tabelle debbono essere solo ridefinite le viste ma non le applicazioni che ne fanno uso

- **Sicurezza**
- Nasconde i dati delle altre relazioni dello schema
- Posso sfruttare le "grant" di una vista