

BASI DI DATI (ESAME)

1) DBMS

Un DATABASE è una collezione di dati gestita da un **DBMS** (Database Management System). In un database troviamo:

- **SCHEMA** = nome tabella + **ATTRIBUTI** (colonne);
- **ISTANZE** = **N-UPLE** (righe), ovvero i dati.

In un'architettura standard troviamo 3 livelli di astrazione per il DBMS:

- **SCHEMA ESTERNO**;
- **SCHEMA LOGICO** = descrizione del database mediante il modello logico;
- **SCHEMA INTERNO** = rappresentazione dello schema logico con memorie fisiche.

Ci deve essere INDIPENDENZA tra i dati sia fisica (DBMS indipendente dalla struttura fisica dei dati) sia logica (struttura esterna indipendente dal livello logico).

Ci sono 2 tipi di linguaggi di accesso ai dati:

- **DDL** (Data Definition Language) = creazione del database, delle tabelle e dei dati;
- **DML** (Data Manipulation Language) = interrogazioni e aggiornamenti sui dati (già nel database).

Usare un DBMS ha pro e contro:

- Vantaggi: modello unificato, indipendente e affidabile con controllo centralizzato sui dati;
- Svantaggi: costoso e integrato (non scorporabile).

Ci sono diverse tipologie di UTENTI di un database:

- **AMMINISTRATORE** = ha accesso a tutto il database (e lo gestisce) e gestisce i PRIVILEGI;
- **PROGETTISTI** e **PROGRAMMATORI**;
- **UTENTI**:
 - Finali = usano le **TRANSAZIONI**;
 - Casuali = formulano **INTERROGAZIONI**.

⚠ Il Creatore di una risorsa del database e l'Amministratore possono concedere i PRIVILEGI DI ACCESSO a determinati utenti (di tipo: INSERT, UPDATE, DELETE, SELECT, REFERENCES, USAGE):

● ● ●

```
1 GRANT ALL PRIVILEGES
2 ON PRODOTTI TO Rossi, Bianchi;
```

● ● ●

```
1 REVOKE SELECT
2 ON PRODOTTI FROM Rossi;
```

● ● ●

```
1 GRANT SELECT
2 ON FORNITORI TO Rossi
3 WITH GRANT OPTION;
```

● ● ●

```
1 REVOKE UPDATE
2 ON FORNITORI FROM ROSSI
3 CASCADE;
```

Il CASCADE nell'ultimo esempio mi evidenzia che, rimuovendo il privilegio all'utente, lo rimuovo anche a tutti gli utenti a cui lui lo ha dato (nel caso in cui avesse il WITH GRANT OPTION); di default troviamo il RESTRICT, ovvero se l'utente a cui cerco di togliere il privilegio lo ha passato ad altri, non posso toglierglielo.

⚠ Le TRANSAZIONI sono sequenze di istruzioni SQL viste come un unico blocco logico; possono finire con:

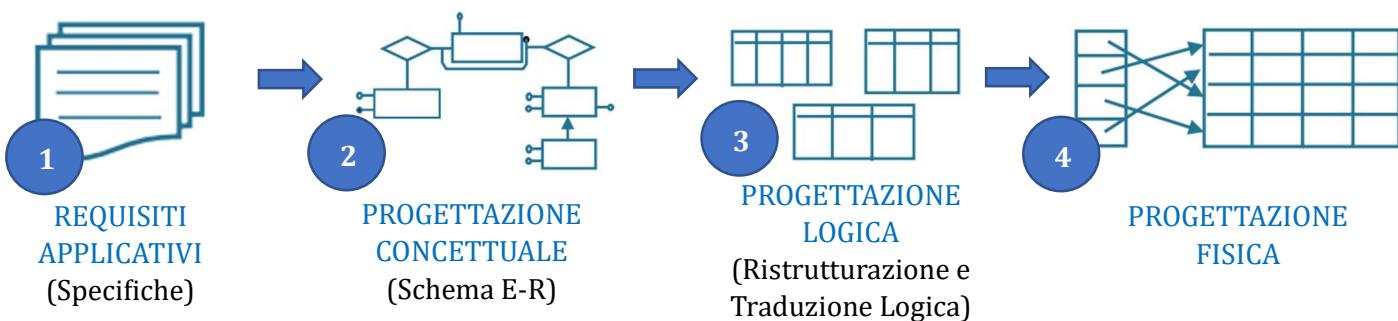
- **COMMIT** = terminata con successo → modifiche effettuate;
- **ROLLBACK** = terminata con errore → modifiche annullate.

Le transazioni godono delle proprietà **ACID**, ovvero Atomicity (indivisibile) – Consistency (da stato consistente a nuovo stato consistente) – Isolation (indipendenza dalle altre transazioni) – Durability (se commit, modifiche persistenti).

2) PROGETTAZIONE



FASI DI PROGETTAZIONE:



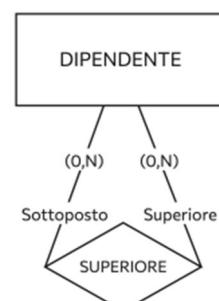
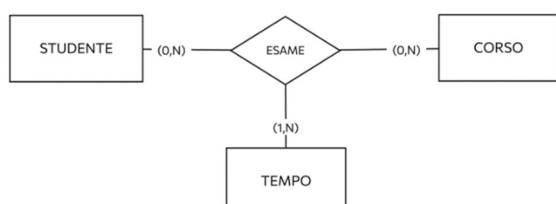
2 SCHEMA ENTITÁ-RELAZIONE (E-R):

- E-R**: è il modello CONCETTUALE più diffuso; basato su:
- **ENTITÁ** = CLASSI DI OGGETTI del mondo reale con proprietà comuni ed esistenza autonoma (1 occorrenza dell'entità = 1 oggetto di quella classe);
 - **RELAZIONE** = LEGAME LOGICO tra entità (1 occorrenza = 1 n-upla) [le n-uple sono tutte diverse].

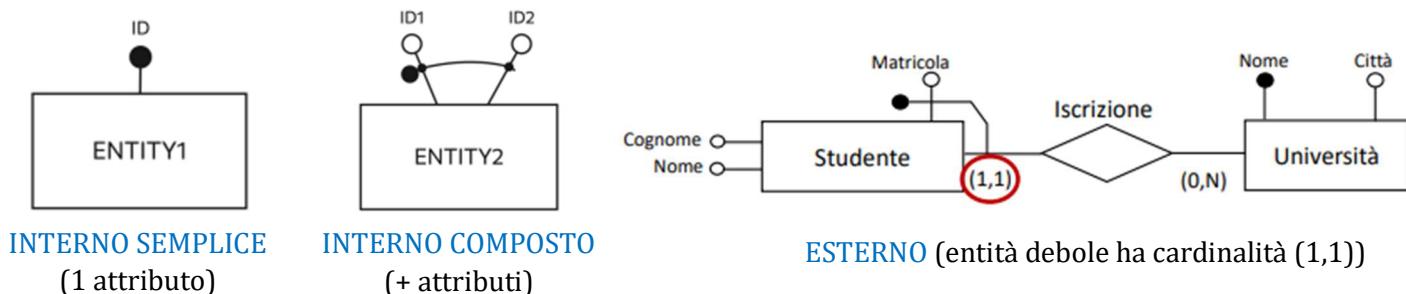


⚠ Le tuple (n,n) indicano la **CARDINALITÀ**, ovvero il n° MIN e MAX di occorrenze di una relazione a cui può partecipare l'occorrenza dell'entità [nell'esempio, PERSONA può avere RESIDENZA in minimo 1 e in massimo 1 CITTÁ, mentre CITTÁ può avere da 1 a N PERSONE RESIDENTI].

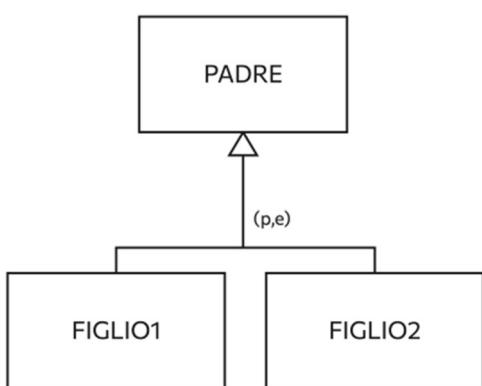
Esistono relazioni **BINARIE**, **TERNARIE**, n-arie e anche **RICORSIVE**:



Ogni entità/relazione può avere degli **ATTRIBUTI** (semplici o composti), ovvero una proprietà; per le entità un attributo deve essere **IDENTIFICATORE** (ovvero **CHIAVE UNIVOCA**) che può essere:



Oltre agli attributi, altri elementi sono le **GENERALIZZAZIONI-SPECIALIZZAZIONI**:



- **PADRE** = GENERALIZZAZIONE dei figli;
- **FIGLIO** = SPECIALIZZAZIONI del padre.

Ogni occorrenza del figlio è anche occorrenza del padre, mentre ogni proprietà del padre è anche una proprietà del figlio (ereditarietà).

La tupla (p,e) indica il tipo della generalizzazione:

- o **TOTALE** (t) se ogni occorrenza del padre è almeno 1 dei due figli [altrimenti **PARZIALE** (p)];
- o **ESCLUSIVA** (e) se ogni occorrenza del padre è al massimo 1 dei 2 figli [altrimenti **SOVRAPPOSTA** (s)].

⚠ Il **SOTTOINSIEME** è una generalizzazione con 1 solo figlio (dunque per forza (p,e))!

3

PROGETTAZIONE LOGICA:

- si compone di 2 fasi:
- A) **RISTRUTTURAZIONE** dello schema E-R;
 - B) **TRADUZIONE** dello schema concettuale ristrutturato in quello logico.

A) RISTRUTTURAZIONE = eliminazione di tutti i costrutti senza rappresentazione nel modello relazionale:

- scomposizione o accorpamento degli **ATTRIBUTI COMPOSTI**;
- sostituzione degli **ATTRIBUTI MULTIVALORE** (1,N) con una nuova relazione;
- eliminazione delle **GENERALIZZAZIONI** con:
 - o **ACCORPAMENTO NEL PADRE** = mettere gli attributi dei figli nel padre come opzionali e aggiungere l'attributo "tipo" per distinguere tra i due figli;
 - o **ACCORPAMENTO NEI FIGLI** = mettere gli attributi del padre in entrambi i figli e, nelle relazioni in cui era coinvolto il padre, sostituire ad esso i figli con cardinalità opzionale;
 - o **SOSTITUZIONE DELLA GENERALIZZAZIONE CON RELAZIONI IS_A**.
- analisi delle **RIDONDANZE**;
- **PARTIZIONAMENTO** dei concetti.

B) TRADUZIONE = tradurre lo schema E-R nel modello relazionale (TABELLE), facendo attenzione ai VINCOLI DI INTEGRITÀ REFERENZIALE presenti tra le tabelle.

⚠ Bisogna prestare particolare attenzione alla **GESTIONE DEL TEMPO** nel modello E-R:

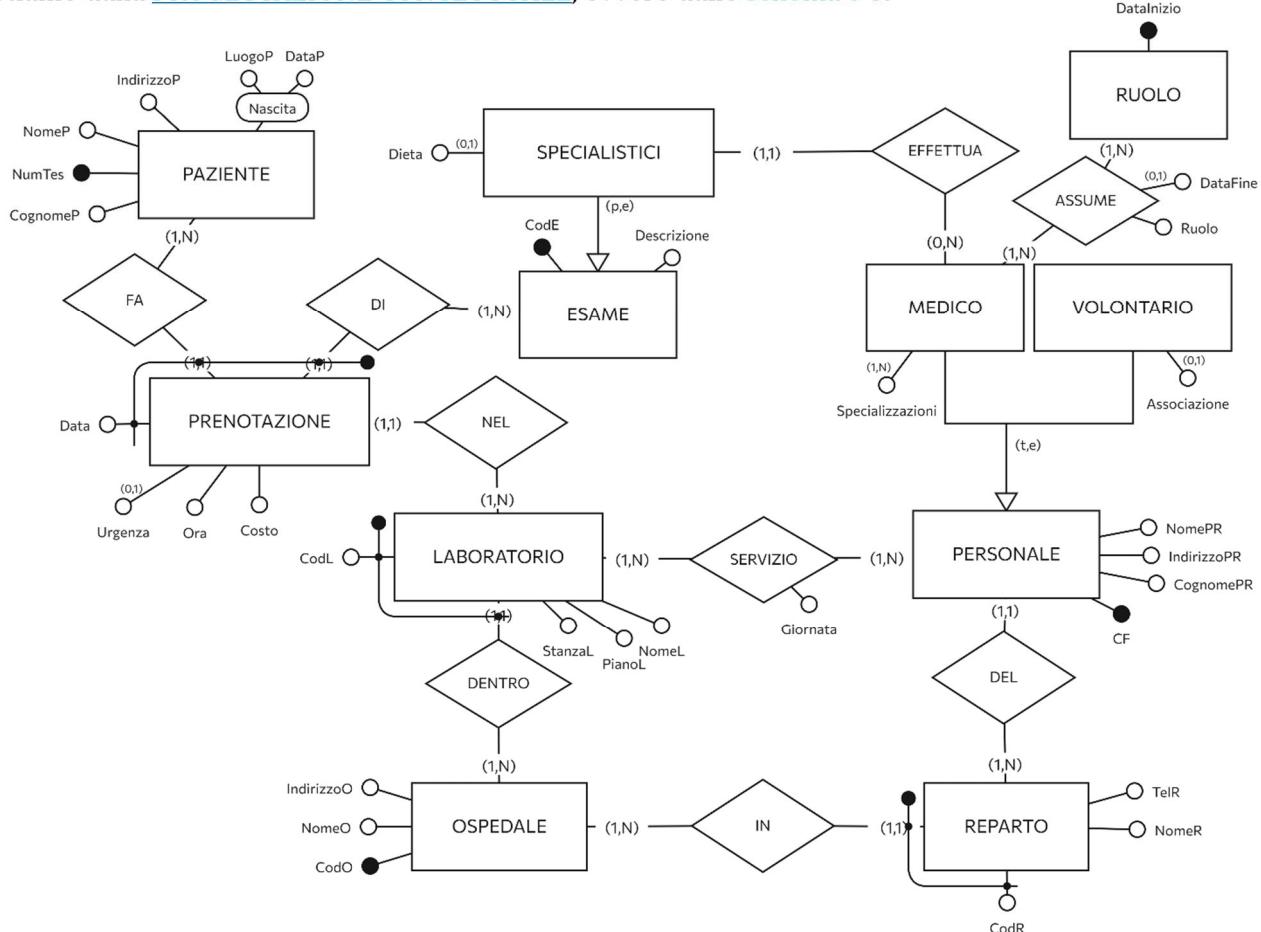
- **ATTRIBUTI TEMPORALI** = rappresentano eventi singoli (es. data di nascita);
- **ENTITÀ TEMPORALE** con attributo "tempo" in relazione con l'entità di interesse (eventualmente con id esterno per creare doppia chiave id-data) [+ **ENTITÀ STORICIZZATA**].

💡 Per fissare meglio i concetti, vediamo un **ESERCIZIO COMPLETO DI PROGETTAZIONE** (logica + concettuale):

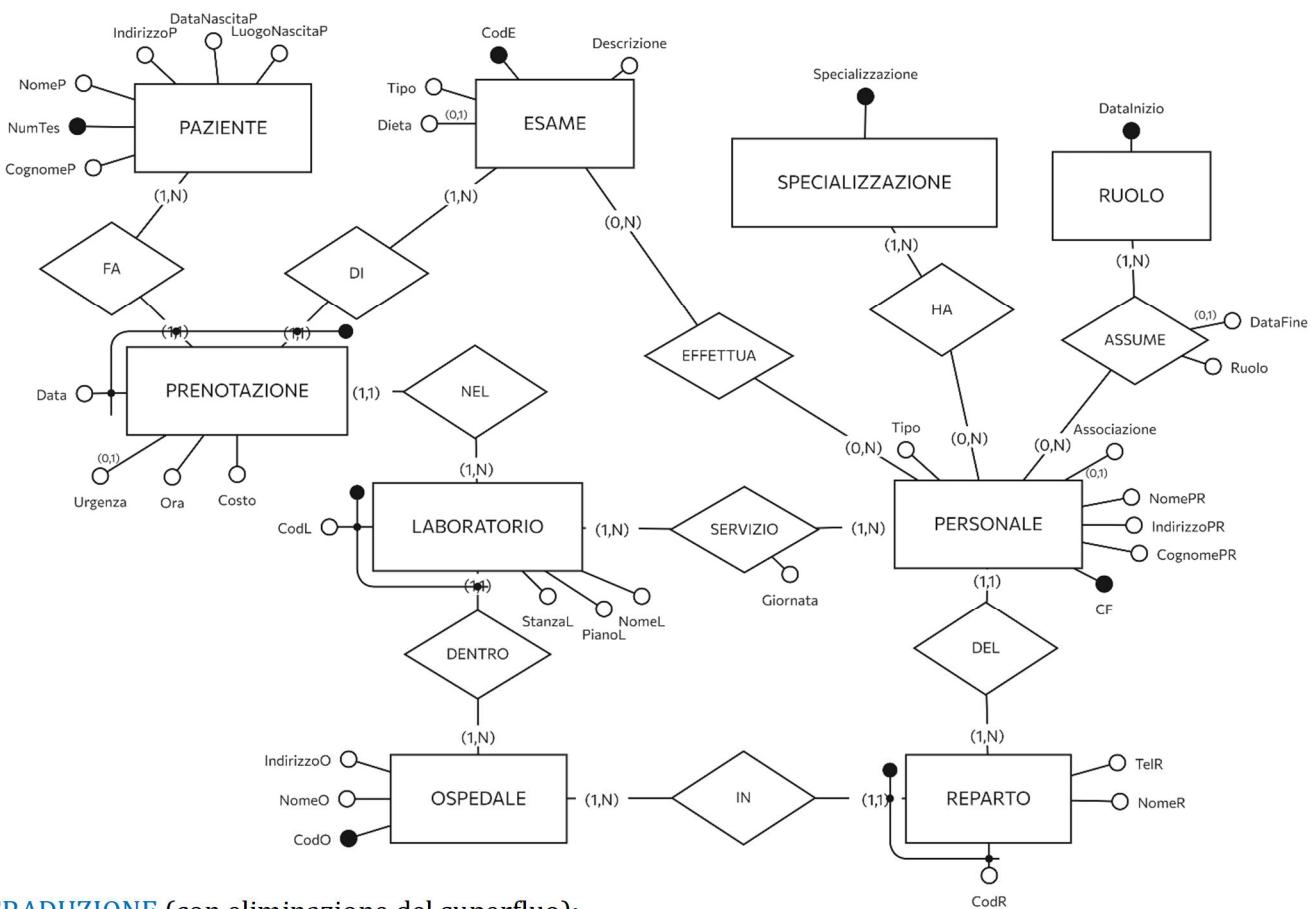
Si vuole rappresentare una base dati per la gestione di un sistema di prenotazioni di esami medici all'interno di una Azienda Sanitaria Locale (ASL), tenendo conto delle informazioni seguenti:

- Ciascun **paziente** è caratterizzato da numero della tessera sanitaria, nome, cognome, indirizzo, data di nascita, luogo di nascita e età. Gli **ospedali** della ASL sono caratterizzati da un codice numerico, da un nome e un indirizzo;
- Ogni ospedale è suddiviso in **reparti** identificati da un codice numerico univoco all'interno dell'ospedale di appartenenza e caratterizzati dal nome del reparto e numero di telefono. Il **personale** del reparto è identificato attraverso il codice fiscale. Sono noti inoltre il nome, il cognome e l'indirizzo di domicilio. Tra il personale, nel caso dei **medici** del reparto è noto l'elenco delle specializzazioni conseguite, mentre per il personale **volontario** è noto il nome dell'associazione di appartenenza, se disponibile;
- Gli **esami** medici che possono essere eseguiti sono caratterizzati da un codice numerico e da una descrizione testuale (ad esempio radiografia). Nel caso di esami **specialistici** si memorizzano inoltre il medico che effettua la visita e la descrizione della dieta da seguire (se necessaria). I **laboratori** che eseguono gli esami sono identificati da un codice univoco all'interno di un ospedale della ASL e sono caratterizzati dal nome del laboratorio, dal piano di ubicazione e dal numero di stanza;
- Per ogni componente del personale di laboratorio si memorizzano le **giornate** e i laboratori in cui presta servizio. Si tenga presente che nel corso della stessa giornata ogni componente del personale può prestare servizio presso più laboratori;
- Per effettuare un esame è necessario eseguire una **prenotazione**. Per ogni prenotazione di un esame da parte di un paziente si vuole memorizzare la data e l'ora dell'esame, il laboratorio presso cui è eseguito, il costo del ticket e se tale esame è prescritto con urgenza. Si tenga presente che ogni paziente può effettuare più prenotazioni dello stesso esame in date diverse. Si noti inoltre che lo stesso esame non può essere ripetuto nello stesso giorno dallo stesso paziente, neppure in laboratori diversi;
- Ogni medico può assumere ruoli diversi nel corso della sua carriera (ad esempio assistente, primario, ecc.). Si vuole tenere traccia dei **ruoli** assunti da ogni medico nel corso di tutta la sua carriera e dei periodi di tempo in cui ha assunto tali ruoli (data di inizio, data di fine). Si tenga presente che ogni medico non può assumere contemporaneamente più ruoli, mentre può assumere lo stesso ruolo in periodi di tempo diversi.

→ Partiamo dalla **PROGETTAZIONE CONCETTUALE**, ovvero dallo **schema e-r**:



→ Arriviamo alla PROGETTAZIONE LOGICA, ovvero RISTRUTTURAZIONE:



e TRADUZIONE (con eliminazione del superfluo):

- ENTITÀ SENZA ID ESTERNO:

PAZIENTE (NumTes, NomeP, CognomeP, IndirizzoP, DataNascitaP, LuogoNascitaP)

ESAME (CodE, Descrizione, Tipo, Dieta*)

PERSONALE (CF NomePR, CognomePR, IndirizzoPR, Tipo, Associazione*)

OSPEDALE (CodO, NomeO, IndirizzoO)

RUOLO (DataInizio)

SPECIALIZZAZIONE (Specializzazione)

- ENTITÀ CON ID ESTERNO:

LABORATORIO (CodL, CodO, NomeL, StanzaL, PianoL)

REPARTO (CodR, CodO, NomeR, TelR)

PRENOTAZIONE (NumTes, CodE, Data, Ora, Costo, Urgenza*)

- RELAZIONI:

FA (NumTes, CodE, Data)

DI (CodE, NumTes, Data)

NEL → PRENOTAZIONE (NumTes, CodE, Data, Ora, Costo, Urgenza*, CodL, CodO)

DENTRO (CodL, CodO)

IN (CodR, CodO)

DEL → PERSONALE (CF NomePR, CognomePR, IndirizzoPR, Tipo, Associazione*, CodR, CodO)

SERVIZIO (CodL, CodO, Giornata)

EFFETTUА → ESAME (CodE, Descrizione, Tipo, Dieta*, CF*)

HA (Specializzazione, CF)

ASSUME (CF DataInizio, Ruolo, DataFine*)

→ Dunque lo **SCHEMA LOGICO FINALE** è:

PAZIENTE (NumTes, NomeP, CognomeP, IndirizzoP, DataNascitaP, LuogoNascitaP)

ESAME (CodE, Descrizione, Tipo, Dieta*, CF*)

PERSONALE (CF, NomePR, CognomePR, IndirizzoPR, Tipo, Associazione*, CodR, CodO)

OSPEDALE (CodO, NomeO, IndirizzoO)

LABORATORIO (CodL, CodO, NomeL, StanzaL, PianoL)

REPARTO (CodR, CodO, NomeR, TelR)

PRENOTAZIONE (NumTes, CodE, Data, Ora, Costo, Urgenza*, CodL, CodO)

SERVIZIO (CodL, CodO, Giornata)

HA (Specializzazione, CF)

ASSUME (CF, DataInizio, Ruolo, DataFine*)

→ I **VINCOLI DI INTEGRITÀ REFERENZIALE** sono:

ESAME (CF) REFERENCES PERSONALE (CF)

PERSONALE (CodR, CodO) REFERENCES REPARTO (CodR, CodO)

REPARTO (CodO) REFERENCES OSPEDALE (CodO)

LABORATORIO (CodO) REFERENCES OSPEDALE (CodO)

PRENOTAZIONE (NumTes) REFERENCES PAZIENTE (NumTes)

PRENOTAZIONE (CodE) REFERENCES ESAME (CodE)

PRENOTAZIONE (CodL, CodO) REFERENCES LABORATORIO (CodL, CodO)

ASSUME (CF) REFERENCES PERSONALE (CF)

HA (CF) REFERENCES PERSONALE (CF)

SERVIZIO (CodL, CodO) REFERENCES LABORATORIO (CodL, CodO)

3) MODELLO RELAZIONALE

Il **MODELLO RELAZIONALE** è basato sul concetto di RELAZIONE (rappresentata come **TABELLA**); come abbiamo già visto, di questa tabella distinguiamo:

- **SCHEMA** = nome tabella + **ATTRIBUTI** (ovvero le colonne), con un **GRADO** (n° attributi);
- **ISTANZE** = **N-UPLE** (righe con i dati), con una **CARDINALITÀ** (n° n-uple).

⚠ Le n-uple non sono ordinate, ma sono **DISTINTE** (non duplicati); gli attributi non sono ordinati!

In una tabella vanno rispettati dei **VINCOLI DI INTEGRITÀ**:

- **INTRA-RELAZIONALI**:
 - **UNICITÀ** DEI DATI (CHIAVE);
 - **DOMINIO** (condizioni sul singolo attributo [es. $0 < \text{voto} \leq 30$]);
 - **N-UPLA** (condizioni su più attributi);
- **INTER-RELAZIONALI** (o **REFERENZIALE**) = informazioni in relazioni diverse devono avere uguali valori per uguali attributi.

⚠ **CHIAVE** = insieme **minimo** di attributi che identifica in modo **univoco** le n-uple di una relazione (se è univoco, ma **non minimo** parliamo di **SUPERCHIAVE** \neq chiave)!

ALGEBRA RELAZIONALE = estensione dell'algebra insiemistica che gode della proprietà di **chiusura** (risultato è ancora una relazione). Gli **OPERATORI** si distinguono in:

N° OPERANDI	FUNZIONE SVOLTA
<ul style="list-style-type: none">- UNARI:<ul style="list-style-type: none">○ SELEZIONE (σ_p)○ PROIEZIONE (π_p)	<ul style="list-style-type: none">- INSIEMISTICI:<ul style="list-style-type: none">○ PROD. CARTESIANO (\times)○ UNIONE (\cup)○ INTERSEZIONE (\cap)○ DIFFERENZA ($-$)
<ul style="list-style-type: none">- BINARI:<ul style="list-style-type: none">○ PROD. CARTESIANO (\times)○ JOIN (\bowtie)○ UNIONE (\cup)○ INTERSEZIONE (\cap)○ DIFFERENZA ($-$)○ DIVISIONE (\setminus)	<ul style="list-style-type: none">- RELAZIONALI:<ul style="list-style-type: none">○ SELEZIONE (σ_p)○ PROIEZIONE (π_p)○ JOIN (\bowtie)○ DIVISIONE (\setminus)

💡 Per capire meglio, vediamo degli **ESERCIZI**:

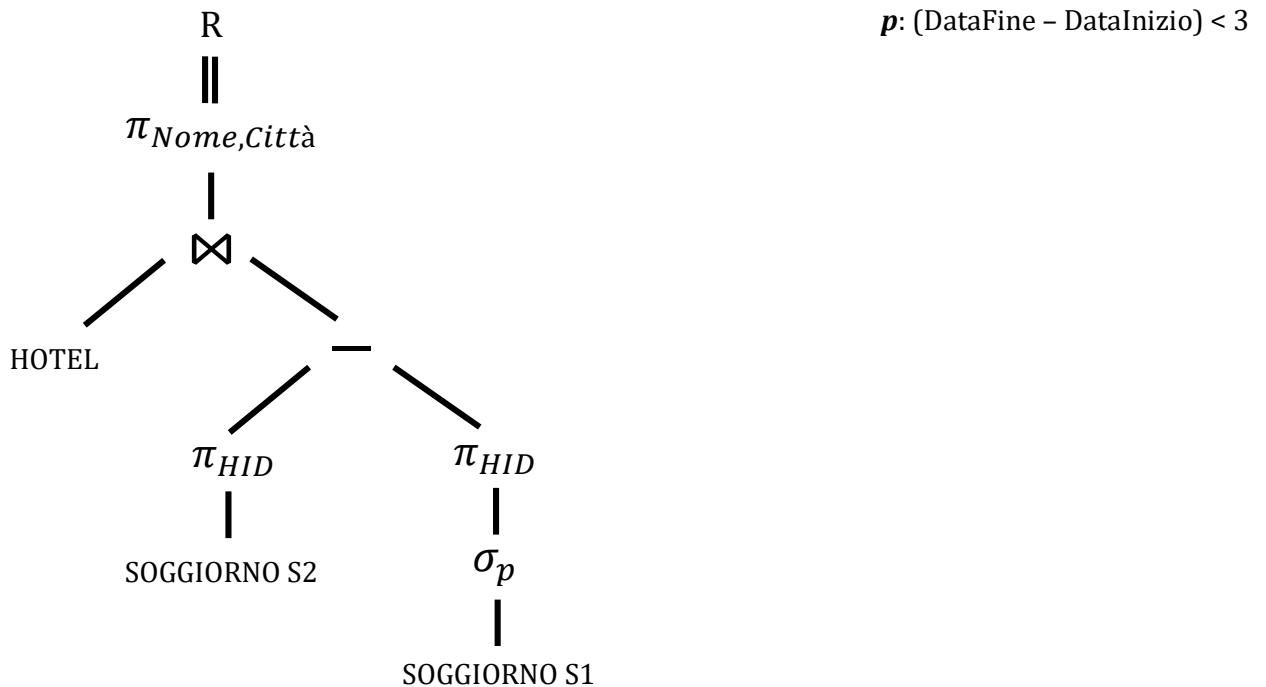
1) Dato il Database

CLIENTE (CID, Nome, Cognome, DataNascita)

HOTEL (HID, Nome, Città, Regione, NumStelle)

SOGGIORNO (CID, HID, DataInizio, DataFine)

→ Visualizzare il nome e la città degli hotel che non hanno mai ospitato clienti per soggiorni di durata inferiore ai 3 giorni (durata espressa come differenza tra DataFine e DataInizio).



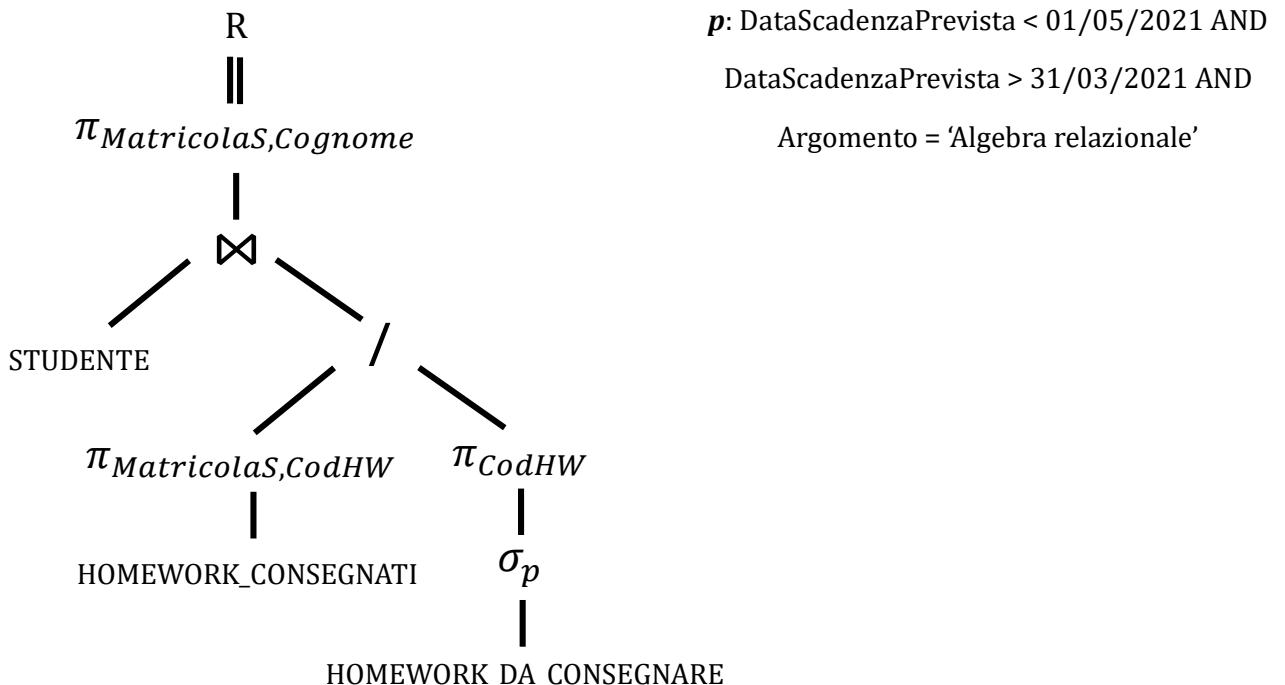
2) Dato il Database

STUDENTE(MatricolaS, Nome, Cognome, CorsoDiLaurea)

HOMEWORK_DA_CONSEGNARE(CodHW, Titolo, Argomento, DataScadenzaPrevista)

HOMEWORK_CONSEGNATI(MatricolaS, CodHW, DataConsegna)

→ Visualizzare la matricola e il cognome degli studenti che hanno consegnato tutti gli homework di argomento "Algebra relazionale" con data di consegna prevista nel mese di Aprile 2021.



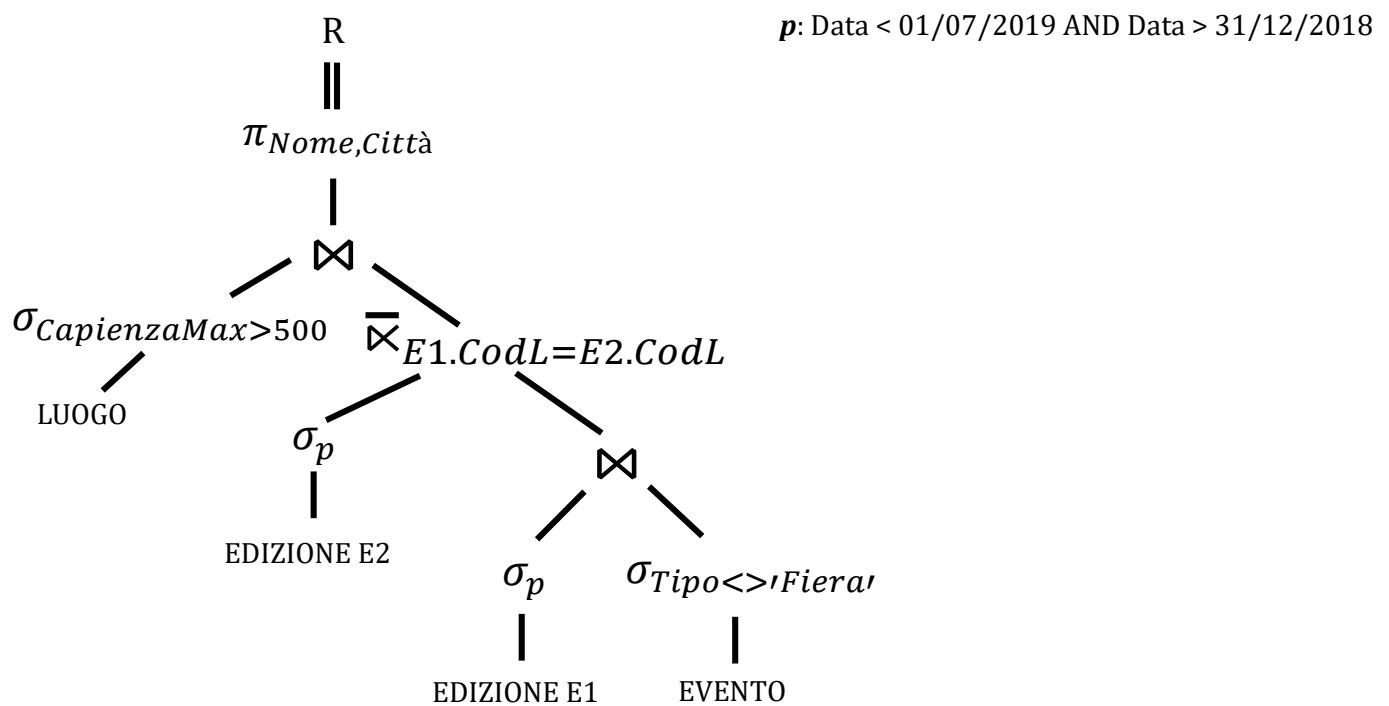
3) Dato il Database

LUOGO (CodL, Nome, Città, CapienzaMax)

EVENTO (CodE, Titolo, Tipo)

EDIZIONE (CodE, Data, CodL)

→ Visualizzare nome e città dei luoghi con capienza massima superiore a 500 che hanno ospitato solo eventi di tipo “fiera” nel primo semestre del 2019.



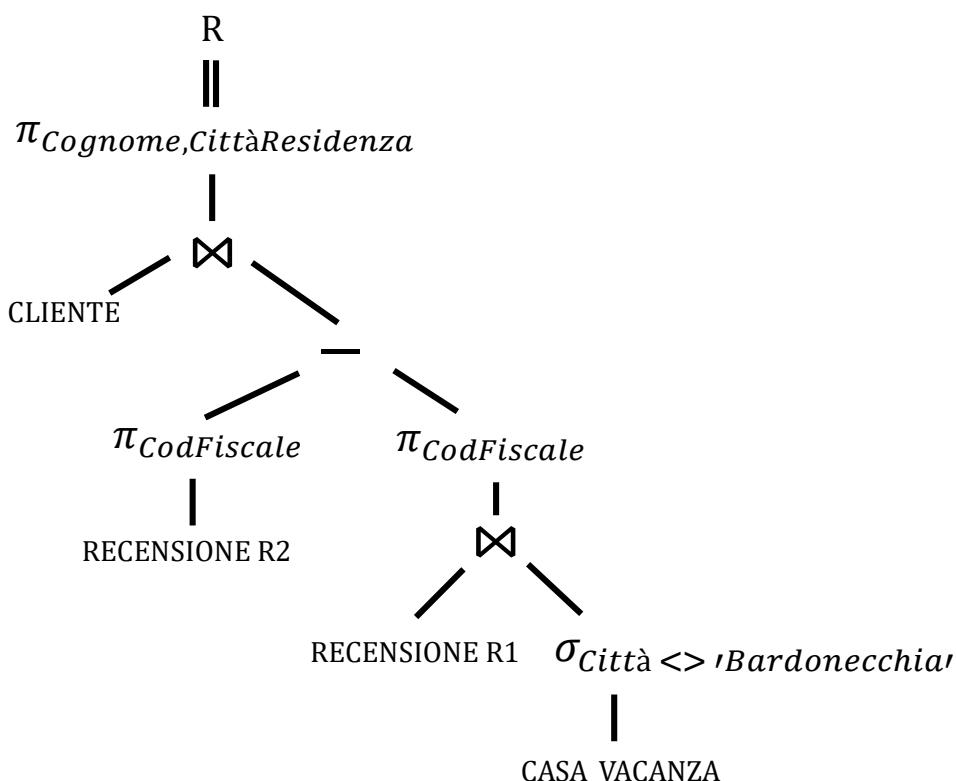
4) Dato il Database

CLIENTE (CodFiscale, Nome, Cognome, DataNascita, CittàResidenza)

CASA_VACANZA (CID, Nome, Tipologia, Indirizzo, Città, PrezzoSettimana)

RECENSIONE (CodFiscale, CID, Data, Testo, Punteggio)

→ Visualizzare cognome e città di residenza dei clienti che hanno recensito solo case vacanze situate presso la città di Bardonecchia.



⚠ Facciamo solo un focus sui possibili OPERATORI DI JOIN che non abbiamo visto negli esempi:

- **THETA-JOIN** (\bowtie_p) = come il NATURAL JOIN (\bowtie) ma la condizione di join tra le due tabelle è espressa dal predicato (ovvero non viene preso il campo in comune di default, ma specifichiamo il join). A differenza del NATURAL JOIN contiene anche duplicati;
- **SEMI-JOIN** (\bowtie_p) = come il THETA-JOIN ma viene preso solo schema del 1° operando;
- **OUTER-JOIN** = come i join già visti ma per le tuple che non rispettano il predicato mettiamo NULL:
 - o **LEFT** (\bowtie_p) = THETA-JOIN + le n-uple del 1° operando (con quelle del 2° = NULL);
 - o **RIGHT** (\bowtie_p) = THETA-JOIN + le n-uple del 2° operando (con quelle del 1° = NULL);
 - o **FULL** (\bowtie_p) = unione di LEFT e RIGHT.

⚠ Negli esercizi alla fine non si usa nemmeno il **PRODOTTO CARTESIANO** (\times) che genera una tabella con tutte le n-uple del 1°operando unite a tutte le n-uple del 2°!

3) SQL

È un linguaggio per gestire i **DATABASE RELAZIONALI** (Structured Query Language); è interattivo, compilato, dichiarativo e a livello di set (risultato è sempre una relazione). È sia **DML** (Data Manipulation Language) sia **DDL** (Data Definition Language).

Dal punto di vista DML → vediamo i costrutti principali con degli esempi con il database

PRODOTTI (CodP, NomeP, Colore, Taglia*, Magazzino)

FORNITORI (CodF, NomeF, NSoci, Sede)

FORNITURE (CodF, CodP, Qta)

A) Seleziona il codice prodotto e la taglia americana (ovvero taglia - 14) dei prodotti con nome che inizia per C.

```
1 SELECT CodP, Taglia - 14 AS TagliaUSA  
2 FROM PRODOTTI  
3 WHERE NomeP LIKE 'C%';
```

⚠ Con gli operatori **LIKE** e **NOT LIKE** c'è:

- '%' per N caratteri qualsiasi;
- '_' per 1 carattere qualsiasi;

Nome **NOT LIKE** '_e%' → la 2^a lettera ≠ 'e'

B) Seleziona i prodotti con taglia maggiore di 44 oppure che potrebbero avere taglia maggiore di 44 (non specificata la taglia), ordinando il risultato in ordine crescente di nome e decrescente di taglia.

```
1 SELECT *  
2 FROM PRODOTTI  
3 WHERE Taglia > 44 OR Taglia IS NULL  
4 ORDER BY NomeP, Taglia DESC;
```

⚠ Con '*' seleziono tutti i campi di PRODOTTO; con **IS NULL** seleziono anche i prodotti con taglia non specificata. Infine con **ORDER BY**:

- ordino in maniera crescente i nomi dei prodotti (**ASC** è di default);
- ordino in maniera decrescente le taglie dei prodotti (specifico **DESC**).

C) Seleziona i nomi dei fornitori che hanno venduto più di 100 unità del prodotto 'P2'.

```
1 SELECT F.NomeF  
2 FROM FORNITORI F, FORNITURE FP  
3 WHERE F.CodF = FP.CodF AND FP.CodP = 'P2'  
4 AND FP.Qta > 100;
```

⚠ Dato che NomeF è presente solo in FORNITORI, dove però non sono presenti informazioni sui prodotti, dobbiamo fare il **JOIN** su FORNITORI e FORNITURE, in modo da ottenere il NomeF, passando però dalle informazioni richieste sui prodotti.

⚠ Una scrittura alternativa del **JOIN** (al posto di **FROM + WHERE + '='**) è:

```
1 SELECT F.NomeF  
2 FROM FORNITORI F INNER JOIN FORNITURE FP ON F.CodF = FP.CodF  
3 WHERE FP.CodP = 'P2' AND FP.Qta > 100;
```

⚠ Al posto di **INNER**, potevo anche avere **OUTER** (FULL, LEFT o RIGHT).

D) Seleziona per ogni prodotto la quantità totale di pezzi forniti e selezionare la quantità massima.

```
1 SELECT MAX(Tot)  
2 FROM (  
3     SELECT CodP, SUM(Qta) AS Tot  
4     FROM FORNITURE  
5     GROUP BY CodP  
6 );
```

⚠ Creo la tabella derivata interna dove ho per ogni CodP, CodP e la **SUM** delle sue Qta (che rinomino 'Tot'). Da qui nel livello più esterno, seleziono il **MAX** delle somme (se ci avesse chiesto il minimo usavo **MIN**).

E) Seleziona la media del numero di fornitori per prodotto.

```
1 SELECT AVG(NumFperP)
2 FROM (
3     SELECT COUNT(DISTINCT CodF) AS NumFperP
4     FROM FORNITURE
5     GROUP BY CodP
6 );
```

⚠ Nella tabella derivata interna uso **COUNT** per contare per ogni CodP il suo numero di CodF (senza ripetizioni tramite **DISTINCT**), che rinomino **NumFperP**. Da qui, ne faccio la media tramite **AVG**.

F) Trovare la quantità totale di pezzi forniti per i prodotti per cui sono forniti in totale almeno 100 pezzi.

```
1 SELECT CodP, SUM(Qta) AS Tot
2 FROM FP
3 GROUP BY CodP
4 HAVING SUM(Qta) >= 100;
```

⚠ La clausola **HAVING** si per esprimere condizioni sulle funzioni aggregate (come un WHERE ma per le condizioni su SUM, MAX, MIN, AVG, COUNT [dove appunto non posso usare il WHERE]).

⚠ Ricorda che non si possono mettere nella **SELECT** attributi diversi da quelli contenuti nella **GROUP BY**!

G) Trovare il nome dei fornitori che non forniscono il prodotto 'P2'.

```
1 SELECT NomeF
2 FROM FORNITORI
3 WHERE CodF NOT IN (
4     SELECT CodF
5     FROM FORNITURE
6     WHERE CodP = 'P2'
7 );
```

⚠ La clausola **NOT IN** si usa come l'ANTI-JOIN; viene seguita dalla tabella derivata dove mettiamo l'insieme che vogliamo escludere. Analogamente è opposto è l'uso della clausola **IN**.

```
1 SELECT NomeF
2 FROM FORNITORI F
3 WHERE NOT EXISTS (
4     SELECT *
5     FROM FORNITURE FP
6     WHERE CodP = 'P2' AND F.CodF = FP.CodF
7 );
```

⚠ Lo stesso esercizio lo si può risolvere con l'operatore **NOT EXISTS** (che ha come opposto **EXISTS**):

H) Trovare il codice dei fornitori che forniscono tutti i prodotti (come l'operatore DIVISIONE in algebra).

```
1 SELECT CodF
2 FROM FORNITURE
3 GROUP BY CodF
4 HAVING COUNT(*) = (
5     SELECT COUNT(*)
6     FROM PRODOTTI
7 );
```

⚠ Nella tabella derivata interna conto il n° di prodotti; a questo punto mi basta trovare i fornitori che hanno un n° di prodotti = a quel numero.

→ Vediamo ora una panoramica di altri costrutti che non ho mai usato negli esercizi; trova le città con:

Fornitori **o** Magazzini

```
1 SELECT Sede
2 FROM FORNITORI
3 UNION
4 SELECT Magazzino
5 FROM PRODOTTI;
```

Fornitori **e** Magazzini

```
1 SELECT Sede
2 FROM FORNITORI
3 INTERSECT
4 SELECT Magazzino
5 FROM PRODOTTI;
```

Fornitori **ma non** Magazzini

```
1 SELECT Sede
2 FROM FORNITORI
3 EXCEPT
4 SELECT Magazzino
5 FROM PRODOTTI;
```

⚠ Ricorda che UNION elimina i duplicati (se si vogliono tenere i duplicati, usare UNION ALL)!

⚠ Negli esercizi ho sempre usato le TABELLE DERIVATE, ma esistono anche le CTE (Common Table Expression), che sono tabelle temporanee scritte attraverso il WITH prima dell'interrogazione (e non dentro l'interrogazione stessa); in questo modo sono riutilizzabili senza doverle ridefinire più volte.

Dal punto di vista DDL

A) GESTIONE DELLE N-UPLE DI UNA TABELLA:

- Inserire (INSERT) nella tabella FORNITURE la riga F2, P2, 200.

```
1 INSERT INTO FORNITURE (CodF, CodP, Qta)
2   VALUES ('F2', 'P2', 200);
```

⚠ Il CodF 'F2' e il CodP 'P2' devono essere già esistenti nelle tabelle FORNITORI e PRODOTTI, altrimenti non rispetta il vincolo di integrità referenziale.

- Eliminare (DELETE) dalla tabella FORNITURE i fornitori F1.

```
1 DELETE FROM FORNITURE
2 WHERE CodF = 'F1';
```

⚠ Sempre per rispettare il vincolo di integrità referenziale bisognerebbe eliminare anche dalla tabella FORNITORI i fornitori con CodF = 'F1'.

- Aggiornare (UPDATE) la tabella PRODOTTI dando ai prodotti con codice P1 il colore 'Giallo', aumentando la taglia di 2 e mettendo a NULL il magazzino.

```
1 UPDATE PRODOTTO
2 SET Colore = 'Giallo', Taglia = Taglia + 2, Magazzino = NULL
3 WHERE CodP = 'P1';
```

B) GESTIONE DELLA TABELLA:

- Creare (CREATE) la tabella FORNITORI.

```
1 CREATE TABLE FORNITORI(
2   CodF CHAR(5),
3   NomeF CHAR(20),
4   NSoci SMALLINT,
5   Sede CHAR(15)
6 );
```

⚠ La tabella è ancora incompleta, dopo la completiamo nella parte dei vincoli [✳]. I termini **viola** (CHAR, ...) sono i **DOMINI** (ovvero i tipi di dato).

- Modificare (ALTER) una tabella:

- Aggiungere la colonna NDipendenti alla tabella FORNITORI.
- Mettere a zero il valore di default della colonna Qta della tabella FORNITURE.
- Eliminare la colonna NSoci dalla tabella FORNITORI.

```
1 ALTER TABLE FORNITORI
2 ADD COLUMN NDipendenti SMALLINT;
```

```
1 ALTER TABLE FORNITURE
2 ALTER COLUMN Qta SET DEFAULT 0;
```

```
1 ALTER TABLE FORNITORI
2 DROP COLUMN NSoci RESTRICT;
```

- Eliminare (**DROP**) la tabella PRODOTTI e le “viste” in cui essa compare.

```
1 DROP TABLE PRODOTTI CASCADE;
```

⚠ I due termini evidenziati si usano per:

- **RESTRICT** = se la tabella da rimuovere è presente in qualche **vincolo**, non la si elimina;
- **CASCADE** = se la tabella da rimuovere compare in delle “**viste**”, eliminiamo anche queste;

→ Per verificare l'**integrità dei dati**, si possono usare:

1. **PROCEDURE APPLICATIVE** = verificano tutto ciò che va verificato;
2. **VINCOLI DI INTEGRITÀ** = il controllo è affidato automaticamente al DBMS; si dividono in:
 - o **VINCOLI DI TABELLA** (ovvero sulle colonne di una tabella) [⊗]:

```
1 CREATE TABLE FORNITORI(
2   CodF CHAR(5) PRIMARY KEY,
3   NomeF CHAR(20) NOT NULL,
4   NSoci SMALLINT CHECK(NSoci >= 0),
5   Sede CHAR(15),
6   UNIQUE(CodF, NomeF)
7 );
```

⚠ Vediamo:

- **PRIMARY KEY** = chiave primaria;
- **NOT NULL** = non nullabile;
- **CHECK (Predicato)** = vincolo logico di tupla;
- **UNIQUE** = attributo univoco;

Da cui vediamo che **NOT NULL UNIQUE** ci segnala una chiave candidata. Tutti questi vincoli possono essere **SINGOLI** (dopo l'attributo singolo) o **MULTIPLI** (al fondo con l'elenco degli attributi).

- o **VINCOLI DI INTEGRITÀ REFERENZIALE** (ovvero tra tabelle):

```
1 CREATE TABLE FORNITURE (
2   CodF CHAR(5),
3   CodP CHAR(6),
4   Qta INTEGER,
5   PRIMARY KEY (CodF, CodP),
6   FOREIGN KEY (CodF) REFERENCES FORNITORI(CodF)
7   ON DELETE NO ACTION
8   ON UPDATE CASCADE,
9   FOREIGN KEY (CodP) REFERENCES PRODOTTI (CodP)
10 );
```

⚠ Il termine **FOREIGN KEY** ci segnala che l'attributo in questione è chiave esterna rispetto all'attributo corrispondente della tabella scritta dopo **REFERENCES**. Nella prima chiave troviamo anche scritto che:

- se viene cancellato CodF di FORNITORI, non viene fatto nulla sul CodF di FORNITURE;
- se viene aggiornato CodF di FORNITORI, aggiornarlo in cascata anche in FORNITURE;

⚠ Quando si tenta di eseguire un'operazione che viola un vincolo, il sistema può **impedirla** (crash) o eseguire un'azione **compensativa** (raggiicare il problema)!

3. **TRIGGER** = implementano delle **REGOLE ATTIVE (AZIONI)** **ESEGUITE AUTOMATICAMENTE** in risposta a degli **eventi (ATTIVAZIONE)** sotto determinate **condizioni (VALUTAZIONE)** [secondo il modello “**ECA**”, ovvero **Evento-Condizione-Azione**].

⚠ Dato che un trigger può attivare altri trigger (**TRIGGER IN CASCATA**) viene imposto un **LIMITE** per assicurare la **TERMINAZIONE** (ed eventualmente la **CONFLUENZA**, ovvero arrivare in un unico stato finale) [tramite il GRAFO di attivazione dei trigger].

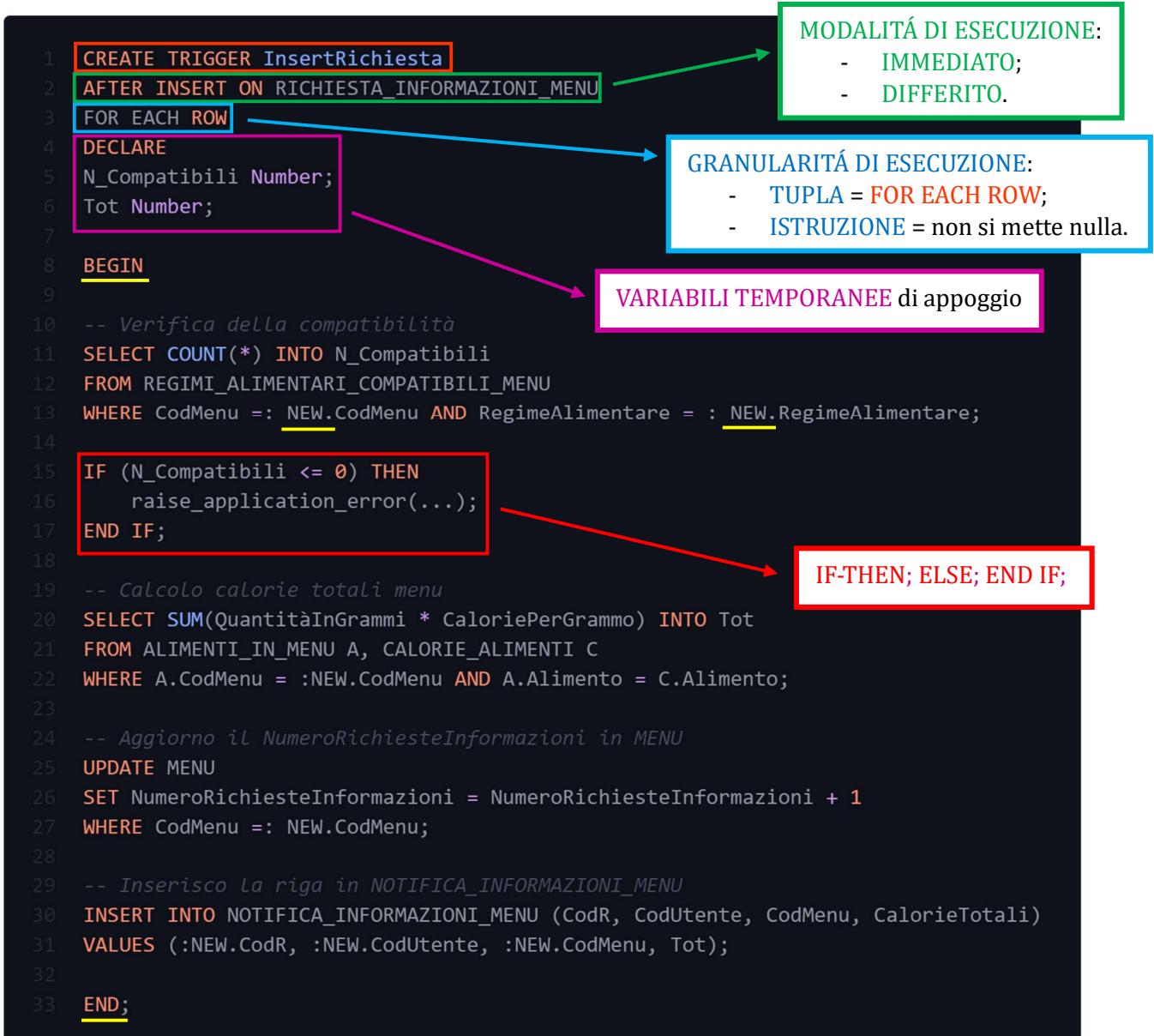
⚠ La tabella principale modificata dal trigger (ovvero quella dopo INSERT, UPDATE, DELETE...) si chiama **TABELLA MUTANTE**!

Dato il Database

MENU (CodMenu, Nome, Descrizione, NumeroRichiesteInformazioni)
REGIMI_ALIMENTARI_COMPATIBILI_MENU (CodMenu, RegimeAlimentare)
ALIMENTI_IN_MENU (CodMenu, Alimento, QuantitàInGrammi)
CALORIE_ALIMENTI (Alimento, CaloriePerGrammo)
RICHIESTA_INFORMAZIONI_MENU (CodR, CodUtente, CodMenu, RegimeAlimentare)
NOTIFICA_INFORMAZIONI_MENU (CodR, CodUtente, CodMenu, CalorieTotali)

Viene aggiunta una richiesta di informazioni per un menù (INSERT ON RICHIESTA_INFORMAZIONI_MENU) con specificato CodUtente, CodMenu e RegimeAlimentare. Bisogna:

- Verificare la compatibilità tra menu e regime alimentare;
- Se compatibile, calcolare le calorie totali per il menu;
- Inserire le informazioni in NOTIFICA_INFORMAZIONI_MENU;
- Incrementare in MENU il NumeroRichiesteInformazioni.



⚠ Se la richiesta del Trigger è su tupla e si può racchiudere in unico predicato (es. Qtà > 100), allora si può interporre tra FOR EACH ROW e BEGIN la clausola **WHEN (Predicato)**!

⚠ Con “`raise_application_error(...)`” stiamo usando un **ROLLBACK** (errore, dunque non finiamo la procedura)!

⚠️ Ora vediamo altre 2 cose mai usate nel corso:

- **DIZIONARIO DEI DATI** = contiene i **METADATI** di un database (ovvero le informazioni sui dati e sulle strutture + le statistiche). Viene gestito dal DBMS, ma può essere interrogato con SQL. In Oracle abbiamo 3 tipi di informazioni nel dizionario:
 - **USER_** = metadati sui dati dell'utente corrente (USER_TABLES, USER_TAB_STATISTICS, USER_TAB_COL_STATISTICS);
 - **ALL_** = metadati sui dati di tutti gli utenti;
 - **DBA_** = metadati delle tabelle di sistema.
- **VISTA** = tabella virtuale il cui contenuto è definito mediante una query SQL (non memorizzato nel database, ma usabile come una tabella); vengono usate al posto delle TABELLE DERIVATE se vogliamo riusarle. Possiamo:
 - **Creare** una vista:

```
1 CREATE VIEW PICCOLI_FORNITORI (CodF, NomeF, Sede) AS
2 SELECT CodF, NomeF, Sede
3 FROM F
4 WHERE NSoci < 3;
```
 - **Eliminare** una vista:

```
1 DROP VIEW PICCOLI_FORNITORI;
```
 - **Aggiornare** una vista:
 - Viste **AGGIORNABILI** = ogni riga della vista corrisponde ad 1 riga della tabella di base;
 - Viste **NON AGGIORNABILI** = nel blocco più esterno della query SQL che la definisce:
 - non c'è la chiave primaria della tabella di base;
 - contiene join 1 a N oppure N a N;
 - contiene funzioni aggregate;
 - contiene DISTINCT.