Appunti Basi Di Dati

Francesco Emanuele Corrado 29407A

a.a. 2024/2025

Contents

Figure 1: Logo Unimi

Basi di Dati

Appunti del corso di Basi Di Dati del prof. Montanelli Stefano di Corrado Francesco, anno 2024/2025

01 - Dati e informazioni

Un *dato* è la registrazione simbolica di un elemento della realtà. In una *base di dati* i dati sono correlati con lo scopo di fornire un'interpretazione e trasformarli in informazione

- **Dati non strutturati** (es. Ricetta): Disponibili tipicamente in formato testuale. Sono memorizzati su file o sistemi di gestione documentale e possono presentare informazioni strutturate, ma la cui struttura non è esplicita e richiede di essere estratta dal file di origine
- **Dati semi strutturati** (es. XML, HTML, JSON): Disponibili in diversi formati memorizzati in DBMS specifici o su file. Presentano informazioni strutturate, ma con pochi vincoli di coerenza dei dati e frequente rindondanza (possono essere inseriti dati errati o ripetuti)
- **Dati (apparentemente) strutturati** (es. Excel): Disponibili prevalentemente in formato tabellare, tipicamente memorizzati in DBMS relazionali o fogli di calcolo. La struttura è rigida, definita da un insieme di campi che si applicano a tutti gli oggetti descritti, ma non è detto che siano rispettati vincoli di formato e coerenza dei dati e che non vi sia rindondanza

Schema e istanza

Il livello di strutturazione di una base di dati dipende dalla presenza o mancanza di uno **schema** predefinito dei dati e dal livello di normatività di tale schema in merito alle regole (**vincoli**) che definiscono una descrizione degli oggetti rappresentati (**istanza**) come valida rispetto allo schema

Gestione dei dati

Per gestire i dati efficacemente ed efficientemente è necessario superare i seguenti problemi:

Caratteristiche delle basi di dati

- **Autodescrizione** -> DDL, DML
- **Indipendenza (fisica o logica)** -> Astrazione
- **Viste multiple** -> multi-utenza, DQL
- **Controllo della concorrenza** -> condivisione
- **Controllo dell'accesso** - DCL

In generale ogni collezione di dati può considerarsi una base di dati, tuttavia si assume che una base di dati abbia alcune caratteristiche distintive:

- Rappresenta un aspetto **limitato della realtà**, riferito a un sottoinsieme dei dati reali disponibili denominato universo del discorso

- Costituisce un'insieme di dati **logicamente coerenti e dotati di un significato intrinseco**, una collezione casuale di dati non è una base di dati
- Una base di dati è **progettata, costruita e popolata per uno scopo specifico**, rivolto a un gruppo di utenti determinato e ad alcune specifiche applicazioni a cui gli utenti sono interessati (da cui dicendono vincoli e regole)

02 - Introduzione ai DBMS

Definizione:

Un sistema di gestione di basi di dati (**DataBase Managment System - DBMS**) è un sistema software che permette agli utenti di creare, mantenere, manipolare e condividere collezioni di dati che siano *grandi*, *condivise* e *persistenti*, assicurando la loro *affidabilità* e *privatezza*. Un DBMS deve essere efficiente ed efficace. Una base di dati è una collezione di dati gestita da un DBMS

Architettura a tre livelli

- Livello esterno
- Livello logico
- Livello interno (fisico)

Schema funzionale di un DBMS

Utenze di un DBMS

- **Amministratori**: Sono utenti del DBMS con *privilegi amministrativi* rispetto a una o più basi di dati ospitete gli utenti con accesso a
  - Funzionalità DDL, DML, DQL, DCL
  - Privilegi e sicurezza
  - Ottimizzazione
  - Memorizzazione fisica
- **Superuser**: Il superuser di un DBMS è un amministratore con privilegi massimi, uno per DBMS
- **Utenti**: Gli utenti hanno privilegi specifici e limitati rispetto a uno o più basi di dati ospitate

Accesso da un DBMS

I DBMS adottano una comunicazione client/server. L'accesso può avvenire da postazioni locali (pipe o OS) o remote (socket TCP/IP)

Gli utenti del DBMS (DBA, DB User) utilizzano un software client per comunicare con il server (con connessioni locali o remote): - Riga di comando - Interfaccia grafica (GUI) - Inerfaccia Web

03 - Introduzione al Modello Relazionale

Storia

È stato proposto da E.F.Codd nel 1970 per favorire l'indipendenza dei dati dallo strumento usato per la memorizzazione. Il modello è stato reso disponibie come modello logico dei DBMS reali nel 1981

Definizioni fondamentali

Il modello relazionale è basato sulla nozione di *relazione matematica* intesa come sottoinsieme del prodotto cartesiano fra due o più insiemi di dati detti domini. Esempi:

D1 = {cane, gatto} D2 = {bianco, nero, marrone}
D1 = {the godfather, the batman} D2 = {crime, comedy, action}
D1 = {x, y, z, k} D2 = {a, b} D3 = {alfa, beta}

Il modello relazionale definisce un insieme di vincoli sui dati ed è associato ad un linguaggio per l'interrogazione delle basi di dati relazionali denominato **algebra relazionale**

La rappresentazione più intuitiva è una **tabella**. Un bd relazionale è quindi rappresentato come una collezione di tabelle

Ogni tabella ha un nume unico nella bd e:
- una riga di tabella detta **ennupla, record o istanza**: rappresenta una corrispondenza fra valori - ogni colonna ha associato un nome distinto di attributo Ak, ad ogni attributo Ak corrisponde un insieme Dk di possibili valori detto dominio

Per dominio si intende una collezione di valori atomici. In pratica i domini sono definiti a partire dai tipi di dati, come stringhe interi o dati. Se non si definisse un nome univoco per ogno dominio della relazione occorrerebbe fare riferimento all'ordine dei domini nella relazione per interpretare i dati

Le righe di una tabella sono diverse fra loro: una relazione non contiene mai ennuple identiche (orientamento ai valori). L'ordinamento delle righe e delle colonne **è irrilevante** perchè sono identificate rispettivamente per conteuto e per nome, e non per posizione

Esempi

Si considerino i seguenti domini (cioè insiemi di possibili valori sui quali sono definiti due attributi)

D1 = { cane, gatto }
D2 = { bianco, nero, marrone }

Una relazione R (tabella) è definita come sottoinsieme del prodotto cartesiano dei domini degli attributi sui quali è definita la relazione R R $A \subseteq B$ D1 x D2

animale(tipo, colore)

prodotto cartesiano della relazione animale

tipo	colore
cane	bianco
cane	marrone
cane	nero
gatto	bianco
gatto	marrone
gatto	nero

Esempio 2

D1 = { the godfather, the batman }
D2 = { crime, comedy, action }

movie(title, genre)
the godfather, crime
the godfather, comedy
the godfather, action
the batman, crime
the batman, comedy
the batman, action

Ipotizziamo una relazione movie con relativi domini (tipi di dato)

```
movie(title, year, length)
movie(varchar(100), char(4), integer)
movie(D1, D2, D3)
```

R(D1, D2, D3) -> definizione della relazione in base ai domini degli attributi
R(A1, A2, A3) -> definizione della relazione in base al nome degli attributi (schema)

Una tupla in una relazione è un record i cui valori sono posizionalmente coerenti con i domini della relazione t(d1, d2, d3) -> d1 $x \in A$ D1, d2 $x \in A$ D2, d3 $x \in A$ D3

t[Ak] -> questo denota il valore dell'attributo Ak nella tupla/record t
t[A2] -> d2

Schema di una base di dati BD = { R1(X1), R2(X2), ..., Rn(Xn) }

X1 = A11, ..., A1k
...
Xn = An1, ..., Anh

04 - Vincoli del Modello Relazionale

Cardinalità e grado

Grado -> numerosità dei suoi attributi
Cardinalità -> numerosità delle tuple presenti in un certo istante (numero di record)

Non ci possono essere due tabelle con lo stresso nome, gli attributi possono essere uguali

Esempio

```
movie(id, title, year, length) -> grado 4
person (id, firstname, lastname, givename, birthdate) -> grado 5
genre(name) -> grado 1
country(name, abbreviation) -> grado 2
```

Il nome dell'attributo (nome) è uguale ma indica due cose diverse, quindi si può fare

Dot Notation -> es. `genre.nome`, `country.nome`, serve a identificare univocamente attributi con lo stesso nome. Non sono obbligato a specificare il nome della tabella a meno che non ci sia ambiguità (es devo usare genere e country nello stesso comando)

Vincoli

Vincoli: caratteristiche che i dati devono avere per essere ammessi in un record. Non può stare in un DB se non soddisfa i vincoli della relazione

`NULL` -> Nel modello relazionale ha valore binario C'è/non c'è ovvero il valore non è presente. Non si può stabilire se manca perchè non c'è o manca perchè non dovrebbe esserci (es. data di morte: potrebbe non essere stata inserita, oppure potrebbe non essere ancora morto)

```
person (id, firstname, *lastname, *givename, birthdate)
```

Asterisco davanti al nome (*lastname) indica un valore che può essere `NULL` per convenzione

`NOT NULL` -> Indica che un valore non c'è ma dovrebbe esserci (es. data di nascita: tutti la hanno, se non è presente non è stata inserita)

```
person (id, firstname, lastname, givename, birthdate)
```

Nel modello relazionale di default i valori deve essere `NOT NULL`
In SQL un valore è potenzialmente `NULL` a meno che non sia specificato

Esempio

givename	deathdate	alive
John	2025-03-05	N
Alice	2025-03-04	Y
Marta	[NULL]	N
Bob	[NULL]	Y

Alice viola un vincolo di integrità dei dati perchè ha una data di morte ma è viva

Bob e Marta invece sono due diversi valori di `NULL`: Marta usa NULL per indicare che la data manca (perchè lei è morta ma non è stata inserita) Bob invece ha `NULL` perchè è ancora vivo ed è giusto che manchi

Vincoli di integrità

Quando `deathdate` non è nullo, il valore di `alive` deve essere `N`

```
check ((deathdate IS NOT NULL AND alive = TRUE) OR (deathdate IS
   NULL))
```

Consideriamo la relazione rating e i vincoli di integrità che potrebbero essere definiti sullo schema:

```
rating(check_date, source, movie, scale, votes, score)
```

Esempio

```
check(score>=0 AND score <= scale)
```

Vuol dire: è vero che score è >= 0 e che score è <= scale? Serve come controllo, se ritorna `false` smette l'esecuzione

```
check(scale in {5, 10, 100})
```

La scala di valutazione ammessa è solo scala 5 10 o 100

Vincoli di chiave

I vincoli di chiave:

- **Superchiave (SK)**
  - Una qualsiasi combinazione di attributi che garantisce l'univocità dei valore nella enuple della relazione
  - Gli attributi K della relazione R sono superchiave se non esistono due tuple t1, t2 in R per le quali t1[K] = t2[K]
- **Chiave (K)**
  - Una chiave è una superchiave minimale
  - K è un insieme di attributi di superchiave di R. K è anche chiave di R se per un qualunque insieme $S \subseteq K$, possono esistere due tuple R tali che t1[K-S] = t2[K-S]. Fondamentalmente K è chiave quando nessun attributo di K può essere eliminato senza perdere la proprietà di superchiave
  - In SQL le chiavi sono definite con il vincolo UNIQUE
- **Chiave Primaria (PK)**

- Vincolo di entity integrity
- Una chiave primaria è una chiave sulla quale non sono possibili valori `NULL`
- Ogni relazione ha una e una sola chiave primaria (c'è sempre non meno e non più di una PK)
- In SQL le chiavi primarie sono definite con il vincolo `PRIMARY KEY`

Esempi

```
movie(id, title, year, length, budget, plot)
id, title, year, length, budget, plot -> SK
id -> SK, K
id, title -> SK
id, year -> SK
id, title, year -> SK
title, year, length, budget, plot -> SK
title, year, length -> SK, K
```

Tutte queste sono superchiavi perchè sono identificative, non esistono delle tuple dove a sinistra delle freccie gli elementi non sono identici. Le chiavi sono attrivuti identificativi alle quali non si può togliere nulla per perdere le proprietà

Queste tuple di movie violano la chiave `title, year, length`?

```
movie(id, title, year, length)
('The␣batman', 2023, 145)
('The␣batman', [NULL], 145)
('The␣batman', 2023, [NULL])
```

No perchè il valore `NULL` non è un valore

Questi vincoli denotano che la terna `title, year, length` è una chiave primaria candidata

```
UNIQUE(title, year, length)
NOT NULL(title)
NOT NULL(year)
NOT NULL(length)
```

Questi vincoli denotano che l'attributo id è una chiave primaria candidata

```
NOT NULL(id)
UNIQUE(id)
PRIMARY KEY(id)
```

Vincoli su country

```
country(iso3, name)
iso3 -> SK, K, PK
iso3, name -> SK
name -> SK, K

NOT NULL(iso3)
NOT NULL(name)
PRIMARY KEY(iso3)
```

Vincoli su cinema

```
cinema(name, city, address, phone)
name, city, address, phone -> SK
```

```
name, city, address -> SK
name, city -> SK, K
name, address -> SK
address -> SK, K

NOT NULL(name)
NOT NULL(city)
NOT NULL(address)
PRIMARY KEY(address)
```

Integrità referenziale

```
movie(id, title, year, length, budget, plot)
rating(check_date, source, movie, scale, votes, score)
```

Vincolo di integrità referenziale interessa due relazioni R1, R2

R1 -> relazione che referenzia (nell'esempio è rating)
R2 -> relazione referenziata (nell'esempio è movie)

R1 contiene un attributo (o insieme di attributi) X detto chiave esterna (o foreign key - FK) (nell'esempio X=rating.movie)
R2 contiene un attributo (o insieme di attributi) Y detto attributo referenziato che è chiave per R2 (nell'esempio Y= movie.id)

05 - Algebra Relazionale

Algebra Lineare: Linguaggio di interrogazione procedurale che definisce le operazioni necessarie per estrarre dati da una o più relazioni di un database

```
movie(id, title, year, length, plot)
rating(check_date, source, movie, score, scale, votes)
```

Esempio di interrogazione: trovare il titolo delle pellicole prodotte nel 2010

L'algebra prevede: - Operatori unari: proiezione (π), selezione (σ), ridenominazione (ρ) - Operatori binari insiemistici: unione, intersezione, sottrazione - Operatori binari join: theta join, equijoin, naturaljoin

Operatori Binari

Unione

Date due relazioni R1 e R2, l'operatore di unione restituisce gli elementi appartenenti a R1 o R2:

$R1 \cup R2 = \{ r1 \in R1 \vee r2 \in R2\}$

Intersezione

Date due relazioni R1 e R2, l'operatore di intersezione restituisce gli elementi appartenenti a R1 e R2:

$R1 \cap R2 = \{ r1 \in R1 \wedge r2 \in R2\}$

Differenza

Date due relazioni R1 e R2, l'operatore di differenza restituisce gli elementi appartenenti a R1 e non appartenenti a R2:

$R1 - R2 = \{ r1 \in R1 \wedge r2 \notin R2\}$

Come si applicano

Condizioni per applicare unione/intersezione/sottrazione a R1(X) e R2 (Y):

- Il grado di grado(R1) = grado(R2) (Devono avere lo stesso grado)
- dom(x_i) = dom(y_i) per ogni x_i ∈ R1, y_i ∈ R2 (Sono definite sugli stessi attributi)

Esempio di sottrazione: Trovare le pellicole del 2010 che non sono thriller

```
movie(id, title, year)
genre(movie, genre)
```

Relazione R1: una relazione con grado 1 che contiene l'attributo `movie.id` e i record dei movie prodotti nell'anno 2010
Relazione R2: una relazione con grado 1 che contiene l'attributo `genre.movie` delle pellicole thriller

R1 - R2 fornisce il risultato

Esempio di unione: Trovare le pellicole che sono di genere thriller o crime

Soluzione Personale:

Relazione R1: una relazione con grado 1 che contiene l'attributo `genre.movie` delle pellicole thriller
Relazione R2: una relazione con grado 1 che contiene l'attributo `genre.movie` delle pellicole crime

R1 ∪ R2 fornisce il risultato

Esempio di intersezione: Trovare le pellicole che sono di genre comedy e romance

Soluzione Personale:

Relazione R1: una relazione con grado 1 che contiene l'attributo `genre.movie` delle pellicole comedy
Relazione R2: una relazione con grado 1 che contiene l'attributo `genre.movie` delle pellicole romance

R1 ∩ R2 fornisce il risultato

Operatori Unari

Selezione σ

σ(p) R esegue una selezione delle ennuple della relazione R restituendo una relazione risultato che contiene le ennuple di R che soddisfano il predicato p. Può utilizzare operatori booleani: AND, OR, NOT

Esempio

Trovare le pellicole del 2010:

```
σ(year = '2010') movie
```

Trovare le pellicole prodotte fra il 2010 e il 2020:

```
σ(year >= '2010' AND year <= '2020') movie
```

Proiezione π

π(x) R esegue una selezione degli attributi di R restituendo una relazione risultato che contiene solo gli attributi x di R

Esempio

Trovare le pellicole del 2010 che non sono thriller:

Soluzione precedente usando proiezione:

```
R1 = π(id) [σ(year= '2010') movie]
R2 = π(movie) [σ(genre='thriller') genre]
risultato = R1 - R2
[π(id) [σ(year= '2010')  movie]] - [π(movie) [σ(genre='thriller')
   genre]]
```

Trovare le pellicole che sono di genre comedy e romance

```
mv_comedy = π(movie) [σ(genre='comedy') genre]
mv_romance = π(movie) [σ(genre='romance') genre]
risultato = mv_comedy ∩ mv_romance
```

Alcune implementazioni scorrette:

```
S1 = π(movie) [σ(genre='comedy' AND genre='romance') genre]
S2 = π(movie) [σ(genre='comedy' OR genre='romance') genre]
```

Trovare le persone che hanno interpretato come attore il personaggio 'dexter':

```
person(id, given_name, birth_date, death_date)
crew(movie, person, role, character)
crew.person -> FK -> person.id

π(person) [σ(character='dexter' ∧ role='actor') crew]
```

Soluzione ottimizzata:

```
π(person) [σ(character='dexter') [σ(role='actor') crew] ]
```

Operatori di join

Trovare il nome delle persone che hanno interpretato come attore il personaggio 'dexter':

Operazione preliminare al join: prodotto cartesiano

Date le relazioni R1(X) e R2(Y) il prodotto cartesiano R1 x R2 restituisce una relazione risultato definita su XY popolata con ogni possibile combinazione delle ennuple di R1 e R2

- R1(A, B)
- R2(C, D)

R1:

A	B
a1	b1
a2	b2

R2:

C	D
c1	d1
c2	d2
c3	d3

R1 x R2 = (A, B, C, D)

A	B	C	D
a1	b1	c1	d1
a1	b1	c2	d2
a1	b1	c3	d3
a2	b2	c1	d1
a2	b2	c2	d2
a2	b2	c3	d3

```
movie(id, title)
genre(movie, genre)

movie
m1    batman
m2    spiderman

genre
m1    action
m1    comics
m2    dark

genre
movie x genre = (id, title, movie, genre)
```

id	title	movie	genre
m1	batman	m1	action
m1	batman	m1	comics
m1	batman	m2	dark
m2	spiderman	m1	action
m2	spiderman	m1	comics
m2	spiderman	m2	dark

Le righe 1 2 e 6 sono incluse nel join perchè il genere del film corrisponde all'ID corrispondente corretto secondo le condizioni del join

Operazione di theta join $\bowtie$

L'operazione di theta join $\bowtie$ è una selezione sul risultato del prodotto cartesiano in base a una condizione theta Θ

Risultato join = R1 $\bowtie_{\Theta}$ R2

R1(A, B)
R2(C, D)

join = R1 $\bowtie_{(A \geq C)}$ R2

A	B	C	D
1	b1	1	d1
1	b1	2	d2
1	b1	3	d3

A	B	C	D
2	b2	1	d1
2	b2	2	d2
2	b2	3	d3

Le righe 1 4 e 5 sono incluse nel theta join pechè $\bowtie_{(A \geq C)}$

Operazione di theta equijoin

Quando la condizione di theta join è un'uguaglianza si parla di equijoin

es. Trovare il titolo delle pellicole con i rispettivi generi

risultato = π(id, title, genre) [movie $\bowtie_{(movie.id=genre.movie)}$ genre]

Restituire il nome delle persone nate dopo il 2000 che recitano in film thriller

```
movie(id, title)
genre(movie, genre)
crew(person, movie)
person(id, given_name, birth_date)

person_2000 = π(id, given_name) [σ(birth_date >= '2001-01-01')
   person]
--schema: person_2000(id, given_name)

movie_thriller = π(movie) [σ(genre='thriller') genre]
--schema: movie_thriller(movie)

person_2000_crew = π(id, given_name, movie) person_2000 *
[ρ(id<-person) crew ]
--schema: person_2000_crew(id, given_name, movie)

risultato = π(id, given_name) [person_2000_crew * movie_thriller]
--schema: risultato(id, given_name, movie)
```

Operazione di join naturale

Un equijoin che considera l'uguaglianza di tutti gli attributi con il medesimo nome nelle due relazioni è setto join naturale

```
R1(A, B)
R2(B, C)

R1 * R2(A, B, C)
```

Operazione di ridenominazione

(ρ): ρ(x->y) R -> La ridenominazione cambia il nome di un attributo x della relazione R. denominandolo y

```
R1(A, B, C)
ρ(C->D) R1 --  R1(A, B, D)
```

ESERCIZI

Sapendo che:

```
movie(id, title, budget, year, length, plot)
rating(movie, source, check_date, scale, score, votes)
genre(movie, genre)
crew(person, movie, p_role, character)
person(id, given_name, birth_date)
released(movie, country, released, title)
produced(movie, country)
```

Trovare il titolo delle pellicole con valutazione (rating) maggiore di 8

Soluzione personale:

```
ratings_8 = π(movie) [σ(score >= '8') rating]
--schema ratings_8(movie)

result = π(title) [movie ⋈ ratings_8]
--schema result(title)
```

Trovare le pellicole thriller con valutazione sopra 8 su scala 10 (verificare entrambe le condizioni)

Soluzione personale

```
ratings_8 = π(movie) [σ(score >= '8' ∧ scale = 10) rating]
--schema ratings_8(movie)

thrillers = π(movie) [σ(genre ='thriller') genre]
--schema: thrillers(movie)

result = π(title) [movie ⋈ (ratings_8 ⋈ thrillers)]
--schema result(title)
```

Trovare il nome dei registi di film thriller

Soluzione personale

```
thrillers = π(movie) [σ(genre ='thriller') genre]
--schema: thrillers(movie)

directors = π(person, movie) [σ(p_role = 'director') crew]
-- Schema: directors(person, movie)

result = π(id, given_name) [person ⋈ (directors ⋈ thrillers)]
-- Schema: result(id, given_name)
```

Trovare i film le cui recensioni sono SEMPRE/TUTTE superiori a 8

Soluzione prof

```
σ(movie) [σ(score>8) rating]
```

Soluzione sbagliata, perchè

rating

movie	score
m1	7.5
m2	8.1
m3	9.0

movie	score
m1	8.5

In questo caso m1 viene giustamente escluso avendo una valutazione di 7.5, ma viene successivamente incluso perchè ha una valutazione di 8.5 –> Errore!

Soluzione corretta:

```
--Seleziono tutte le ennuple che hann valutazione <= 8
A = π(movie) [σ(score<=8) rating]

--Restituisco le pellicole che non rientrano in A
result = [π(movie) rating] - A
```

Trovare le pellicole distribuite (released) sia in USA sia in FRA (intersezione)

Soluzione personale

```
usa = π(movie) [σ(country='USA') released]
--schema: usa(movie)

fra = π(movie) [σ(country='FRA') released]
--schema: fra(movie)

result = π(movie) [usa ∩ fra]
--schema: result(movie)
```

Trovare le pellicole che NON sono prodotte in GBR (sottrazione)

Soluzione personale

```
gb = π(movie) [σ(country='GBR') produced]
--schema gb(movie)

result = [π(movie) produced] - gb
--schema result(movie)
```

Soluzione professore

```
A: π(movie) [σ(country='GBR') produced]
risultato: [π(id) movie] - A
```

Operazione di divisione

Date due relazioni R(XY) e S(Y) la relazione risultato di R÷S(X) è definita come segue:
R÷S(X) = { t1 su X | (∀) t2 (∈) S,(∃) t (∈) R con t[X]=t1 (∧) t[Y]=t2 }

Esempio di divisione: Si considerino le seguenti relazioni

```
movie_product(name, country)
country(name)
```

La divisione restituisce le pellicole di movie_production che sono state prodotte in tutti i paesi di country

Date due relazioni

R(XY) S(Y)

R/S(X) = { t1 su X tc}

R:

a	b	c
a1	b1	c1
a2	b1	c2
a1	b2	c3
a1	b2	c1

S:

b
b1
b2

R÷S

a	c
a1	c1

La divisione è utile quando sto cercando una relazione con tutti i record di un'altra relazione

Esercizio

Trovare titolo e anno dei film che sono thriller, drama e action

```
movie(id, title, year)
genre(movie, genre)

G = σ(genre= 'thriller' or genre='action' or genre = 'drama') genre
-- sintassi alternativa
G = σ(genre in ('thriller', 'action', 'drama')) genre

-- rinomino movie in G
H = ρ(movie->id) G

-- join naturale fra movie e H
J = movie * H

risultato(id, title, year) = J ÷ [ π(genre) G ]
```

In questo esempio il risultato includerà i film di J che contengono tutti i generi inclusi in G, SOLO se ci sono tutti e tre. Un film che ad esempio è un thriller e un action non sarà incluso in quanto non è anche drama

movie

id	title	year
1	m1	2010
2	m2	2011

G

id	genre
1	thriller
1	drama
2	thriller
2	drama
2	action

J

id	title	year	genre
1	m1	2010	thriller
1	m1	2010	drama
2	m2	2011	thriller
2	m2	2011	drama
2	m2	2011	action

π(genre) G

genre
thriller
drama
action

risultato

id	title	year
2	m2	2011

Soluzione alternativa:

```
A: π(movie) [ σ(genre= 'thriller') genre ]
B: π(movie) [ σ(genre= 'drama') genre ]
C: π(movie) [ σ(genre= 'action') genre ]
risultato = A ∩ B ∩ C
```

Questa soluzione alternativa funziona perchè i generi di interesse sono noti a priori. La divisione è utile (e non sostituibile) quando il contenuto della relazione S al denominatore non è noto a priori

Esercizi Slide

Sapendo che:

```
continent(id, name)
country(id, name, continent (continent), currency)
city(id, name, country (country), population)
politician(id, name, birth_place (country), gender, birth_date,
   party)
govern(city (city), head (politician), election_year)
country_borders(country_a (country), country_b (country))
```

Trovare il nome delle città in cui non è utilizzato il dollaro come moneta (currency)

Soluzione Personale

```
country(id, name, continent (continent), currency)
city(id, name, country (country), population)

--A: Trovo tutti i paesi dove è usato il dollaro
A = π (id) [σ(currency='dollar') country ]
--B: Trovo tutte le città che non sono in quei paesi
B = [π (country) city] - A
--result: join con city per trovare la città
result = π(id) [ city * B ]
```

Soluzione Prof

```
-- alternativa (più efficiente)
-- A: paesi che non usano il dollaro
A: π(id) [σ(currency<>'dollaro') country]
-- join con city per trovare le città
B: π(id) [ city ⋈(city.country=country.id) A ]
```

Trovare il nome dei politici che non hanno governato città con più di 500K abitanti

Soluzione Personale

```
city(id, name, country (country), population)
politician(id, name, birth_place (country), gender, birth_date,
   party)
govern(city (city), head (politician), election_year)

-- A: Città con più di 500k abitanti
A = π(id)[σ(population > 500k) city]
--B: Politici che hanno governato in A
B = π(head) [ govern ⋈(govern.city=A.id) A ]
or
B = π(head) [govern ⋈(id=city) A]
--result: Politici totali - B
result = [π (id) politician] - B
```

Trovare i paesi europei che non confinano con l'Italia

Soluzione personale

```
country_borders(country_a, country_b)
country(id, name, continent, currency)

-- A, B, C: trovare i paesi che confinano con l'Italia
-- Il paese confinante potrebbe trovarsi in country_a o country_b,
   ecco perchè unione
A = π(country_b) [σ(country_a = 'ITA') country_borders]
B = π(country_a) [σ(country_b = 'ITA') country_borders]
C = A ∪ B
-- D: sottrarre C dall'elenco di tutti i country
D = [π(id) country] - C
-- o se voglio una ricerca più specifica escludendo a priori gli
   altri continenti
```

```
D = [π(id) [σ(continent = 'EU') country]] - C
```

Dopo una unione, come si chiama la colonna nella unione? A ∪ B, in C la colonna si chiamerà come la colonna del primo dei due -> in questo caso quindi `country_b`

c_a	c_b
ITA	FRA
AUS	ITA
FRA	ITA

Trovare le città che sono state governate da politici sia di destra sia di sinistra

Soluzione personale

```
politician(id, name, birth_place (country), gender, birth_date,
   party)
govern(city (city), head (politician), election_year)

--A, B: Politici di destra e di sinistra
A = π(id) [σ(party='right') politician]
B = π(id) [σ(party='left') politician]
--C, D: Città governate da politici di destra e città governate da
   politici di sinistra
C = π(city) [govern ⋈(head=id) A]
D = π(city) [govern ⋈(head=id) B]
--result: Intersezione tra le due
result = π(city) [C ∩ D]
```

Soluzione Prof

```
politician(id, name, party)
govern(city, head)
-- A: join fra govern e politician
A = [ govern ⋈(head=id) politician ]
-- B: città governate da politici di destra
B = π(city) [ σ(party = 'repubblicano') A ]
-- C: città governate da politici di sinistra
C = π(city) [ σ(party = 'democratico') A ]
-- D: città governate da politici sia di destra sia di sinistra
D = B ∩ C
-- soluzione alternativa con join
A(id, name, party, city, head)
F = σ(party = 'democratico' ∨ party = 'repubblicano') A
E = ρ(party->party')␣[ρ(city->city') F]
soluzione =  A  ⋈(city=city'␣∧ party<>party')  E
```

A =

id	name	party	city	head
p1	X	rep.	c2	p1
p2	Y	rep.	c1	p2
p3	Z	dem.	c1	p3
p1	X	rep.	c2	p1

Trovare il nome dei politici con meno di 40 anni eletti dopo il 2020

```
politician(id, name, birth_place (country), gender, birth_date,
   party)
govern(city (city), head (politician), election_year)

--A: Politici con meno di 40 anni
A = π(id) [σ(birth_date >= 01.01.1985) politician]
--B : Politici eletti dopo il 2020
B = π(head) [σ(election_year > 2020) govern]
--C: Theta join tra i due
C = π(id) [A ⋈(id=head) B]
--result: Estraggo da politician i nomi dagli ID
result = π(name) [C ⋈(id=politician.id) politician]
```

Trovare il nome dei politici che sono stati a capo di città appartenenti a paesi in cui non sono nati

Soluzione personale

```
city(id, name, country (country), population)
country(id, name, continent (continent), currency)
politician(id, name, birth_place (country), gender, birth_date,
   party)
govern(city (city), head (politician), election_year)

--A: Estraggo id e birth_place (country) di ogni poltico
A = π(id, birth_place) [politician]
--B: Estraggo le città governate da ciascun politico
B = π(city, head) [govern]
--C: Theta join tra i due
C = π(head, country) [A ⋈(id=city) B]
--D: Sottraggo da tutte le città quelle di C
D = π(head, country) [(A ⋈(id=city) B) - C]
--result: Estraggo per ogni head il nome
result = π(name) [D ⋈(head=politician.id) politician]
```

Trovare i continenti in cui non ci sono città governate da donne

Soluzione personale

```
continent(id, name)
country(id, name, continent (continent), currency)
city(id, name, country (country), population)
politician(id, name, birth_place (country), gender, birth_date,
   party)
govern(city (city), head (politician), election_year)

--A: estraggo le politiche donne
A = π(id) [σ(gender = 'woman') politician]
--B: estraggo tutte le città governate da politiche donne
B = π(city) [A ⋈(id=head) govern]
--C: estraggo tutti i paesi con città governate da donne
C = π(country) [B ⋈(city=id) city]
--D: estraggo tutti i continenti con città governate da donne
D = π(continent) [C ⋈(country=id) country]
```

```
--result: sottraggo da tutti i continenti D
result = [π(id) [continent]] - D
```

Trovare i politici che hanno governano più di una città

Soluzione personale

```
politician(id, name, birth_place (country), gender, birth_date,
   party)
govern(city (city), head (politician), election_year)

-- A: Trovo tutti i politici del governo
A = π(head, city) [govern]
-- B: metto in self-join A per trovare i record con lo stesso
   valore di head e diverso valore di city
B = ρ(head->head')␣[ρ(city->city') A]
-- result: metto in self-join A per trovare i record con lo stesso
   valore di head e diverso valore di city
result = π(head) [A ⋈(city<>city'∧ head=head') B]
```

Trovare i politici che hanno governato tutte le città di San Marino

Soluzione personale

```
country(id, name, continent (continent), currency)
city(id, name, country (country), population)
govern(city (city), head (politician), election_year)

-- A: Trovare le città di San Marino
A =  π(city.id) [city ⋈(city.country=country.id) [σ(name = 'San␣
   Marino') country]]
-- Trovo tutti i paesi record di San Marino
-- Unisco con city con la condizione country = country id per
   trovare tutte le città di San Marino, restituisco solo gli id
-- result: Divisione (trova tutti i politici che hanno governato
   tutte le città di San Marino)
result = [π(head, city) govern] ÷ [ρ(id->city) A]
```

Trovare le città governate da più di un politico dopo il 2020

Soluzione personale

```
govern(city (city), head (politician), election_year)

-- A: Trovo i record con election_year > 2020
A = π(head, city) [σ(election_year > 2020) govern]
-- B: Rinomino
B = ρ(head->head')␣[ρ(city->city') A]
-- result: metto in self-join A per trovare i record con lo stesso
   valore di city e diverso valore di head
result = π(city) [A ⋈(city=city'∧ head<>head') B]
```

A

city	head
c1	p1

city	head
c2	p2
c1	p3

06 - SQL

SQL (structured quety language)

- DDL (create table, create view, create trigger…)
- DML (insert, delete, update)
- DQL (data query language - select)
- DCL

Query semplici

Selezionare il titolo delle pellicole del 2010

```
movie(id, official_title, year, budget, length, plot)
```

```
select id, official_title, year
from imdb.movie
where year = '2010';
```

Selezionare i record delle pellicole con titolo INTERSTELLAR

```
select *
from imdb.movie
where official_title = 'Interstellar';
```

Superare il problema del “case”

Soluzione A: funzioni lower/upper

```
select *
from imdb.movie
where lower(official_title) = 'interstellar';
```

Soluzione B: usare ilike

```
select *
from imdb.movie
where official_title ilike 'interstellar';
```

Superare il problema degli spazi bianchi all’inizio e alla fine di una stringa:
Utilizzo trim - ltrim - rtrim

```
select *
from imdb.movie
where trim(official_title) ilike 'interstellar';
```

Selezionare pellicole che hanno murder nel titolo

Operatore like
% : segnaposto che indica una stringa di qualsiasi lunghezza
_ : segnaposto che indica una stringa di esattamente 1 carattere

```
select *
from imdb.movie
where (official_title ilike '%murder%') and not (official_title
   ilike '%murder') and not (official_title ilike 'murder%');
```

Soluzione alternativa

```
select *
from imdb.movie
where (official_title ilike '% murder %')
```

Funzionamento

- murder% = tutti i titoli con murder all'inizio
- %murder = tutti i titoli con murder alla fine
- %murder% = ovunque nella stringa

Selezionare le pellicole nel cui titolo compaiano happy e family in qualsiasi posizione

```
select *
from imdb.movie
where (official_title ilike '%happy%family%') or (official_title
   ilike '%family%happy%')
```

Soluzione alternativa:

```
select *
from imdb.movie
where (official_title ilike '%happy%') and (official_title ilike
   '%family%')
```

Selezionare le persone che hanno 'Mark' nel nominativo

```
select *
from imdb.person
where given_name ilike '%mark%';
```

Selezionare le persone che hanno 'Mark' nel nome

```
select *
from imdb.person
where first_name ilike '%mark%';
```

Selezionare le persone che si chiamano 'Mark' di cognome

```
select *
from imdb.person
where last_name ilike 'Mark';
```

Operatori SQL per la selezione: - booleani: AND, OR, NOT
- uguaglianza: =
- disuguaglianza: <>
- confronto: > >= < <=
- BETWEEN
- LIKE
- IN
- IS (NOT) NULL

Selezionare il titolo delle pellicole del 2010

```
select id, official_title
from imdb.movie
where year = '2010';
```

Selezionare tutti gli attributi delle pellicole del 2010 di durata superiore all'ora

```
select *
from imdb.movie
where year = '2010' and length > 60;
```

Selezionare tutti gli attributi delle pellicole del 2010 di durata compresa fra una e due ore

```
select *
from imdb.movie
where year = '2010' and length >= 60 and length <= 120;
```

Nota: i valori vengono valutati in ordine. Se year='2010 è falso, non verifica neanche le condizioni successive e così via

Selezionare tutti gli attributi delle pellicole di durata compresa fra una e due ore (estremi inclusi) realizzate in anni diversi dal 2010

```
select *
from imdb.movie
where year <> '2010' and length >= 60 and length <= 120
```

Nota: <> è il diverso
Sintassi alternativa:

```
select *
from imdb.movie
where not(year='2010') and length >= 60 and length <= 120;
```

Versione con between:

```
select *
from imdb.movie
where year <> '2010' and length between 10 and 120;
```

Posso usare le parentesi per forzare la precedenza tra gli operatori, senza parentesi la precedenza è da sinistra a destra, ad esempio:

```
select *
from imdb.movie
where year = '2010' or (length >= 60 and length <= 120);
```

Così facendo considero prima `and` tra le length e poi `or` con year, senza parentesi il contrario

Selezionare le pellicole di genere Drama, Thriller o Crime

```
select *
from imdb.genre
where lower(genre) = 'drama' or genre ilike 'thriller' or
   lower(genre) = 'crime' ;
```

Nota: lower serve a prendere il valore selezionato con la lettera minuscola, per evitare problemi confrontando stringhe

Versione con operatore IN

```
select *
from imdb.genre
where lower(genre) in ('drama', 'thriller', 'crime');
```

IN è più compatto e leggibile, però bisogna tenere conto delle stringhe maiuscole e non si può usare `ilike`

Differenza con algebra relazionale: le tuple di una tabella sono oggetti, quindi sono indipendenti, es una pellicola thriller e crime avrà due righe. Posso usare `distinct` per eliminare i duplicati

```
select distinct movie
from imdb.genre
where lower(genre) in ('drama', 'thriller', 'crime');
```

Questa è identica all'algebra relazionale. Se io eseguo

```
select distinct movie, genre
from imdb.genre
where lower(genre) in ('drama', 'thriller', 'crime')
```

Posso avere duplicati, in quanto la `distinct` lavora su tutta la clausola select, quindi se un film ha lo stesso nome ma genere diverso esso sarà considerato come due record

`order by` ordina il risultato in base all'attributo specificato:

```
select distinct movie, genre
from imdb.genre
where lower(genre) in ('drama', 'thriller', 'crime')
order by movie;
```

Se applico l'attributo `DESC` ne inverto l'ordine:

```
select distinct movie, genre
from imdb.genre
where lower(genre) in ('drama', 'thriller', 'crime')
order by movie DESC;
```

Restituire le persone nate dopo il 1970 ordinando il risultato in base al nome e all'anno di nascita

```
select *
from imdb.person
where birth_date > '1970-12-31'
order by birth_date, given_name;
--oppure birth_date >= '1970-01-01'
```

A parità di giorno di nascita sono ordinati per nome. Se volessi farlo uguale ma con i nomi decrescenti:

```
select *
from imdb.person
where birth_date > '1970-12-31'
order by birth_date, given_name DESC;
```

Trovare le persone nate nel 1971

```
select *
from imdb.person
where extract(year from birth_date) = '1971';
```

Con `extract` posso ad esempio estrarre dalle date attributi come year month o day

Trovare le persone nate a partire dal 1970 ordinando il risultato in base all'anno e al nome

```
select extract(year from birth_date)::char(4) as "anno di
   nascita", given_name as "nome della persona"
from imdb.person
where extract(year from birth_date) >= '1970'
order by extract(year from birth_date), given_name;
```

Con as birth_year sto assegnando un nome nella select con la rinominazione
:: è l'operatore di cast in postgres: posso forzare il tipo di dato della prima colonna a char(4), ovvero una stringa di lunghezza 4 caratteri

Trovare le persone delle quali si conosce la data di decesso

```
select *
from imdb.person
where death_date is not null
```

Non posso usare invece `where death_date <> NULL`, con `NULL` bisogna usare `is not`, di conseguenza:

Trovare le persone che sono ancora in vita

```
select *
from imdb.person
where death_date is null;
```

Trovare le persone di cui conosciamo la data di nascita ma non conosciamo la data di decesso

```
select *
from imdb.person
where birth_date is not null and death_date is null;
```

Aggiornare il valore di una tabella:
update imdb.person set bio= '' where is = '0080580';

Trovare le persone che non hanno bio (o perchè null o perchè stringa vuota o stringa di blank)

```
select *
from imdb.person
where bio is null or trim(bio) = '';
```

trim riduce gli spazi alla stringa vuota: se una stringa è composta da tanti spazi vuoti con trim diventano una stringa vuota

Interrogazione di più tabelle

Selezionare il titolo delle pellicole prodotte negli Stati Uniti

Prodotto Cartesiano

cardinalità:
card(movie) = 1033
card(produced) = 1332
card(produced x movie) = 1033 x 1332

```
select m.id, m.official_title
from imdb.movie m, imdb.produced p
where m.id = p.movie and country = 'USA';
```

Sintassi alternativa usando inner join

```
select m.id, m.official_title
from imdb.movie m inner join  imdb.produced p on  m.id = p.movie
where country = 'USA';
```

Trovare le pellicole prodotte in due paesi diversi

```
select p1.movie, p1.country as country1, p2.country as country2
from imdb.produced p1, imdb.produced p2
where p1.movie = p2.movie and p1.country < p2.country
```

In algebra si risolve con un self-join dove cerco due record aventi il medesimo valore di produced.movie e diversi valori di produced.country

Esercizi

Selezionare i paesi nei quali sono state distribuite le pellicole del 2010 (si restituisca anche il titolo della pellicola, sia quello ufficiale, sia quello usato nella distribuzione dove presente)

Soluzione personale

```
select m.id, m.official_title, r.country, r.title
from imdb.movie m, imdb.released r
where m.year = '2010' and m.id = r.movie;
```

Selezionare le pellicole per le quali non è noto il titolo di distribuzione in Italia

Soluzione personale

```
select m.id, m.official_title, r.title
from imdb.movie m, imdb.released r
where m.id = r.movie and r.country = 'ITA' and r.title is null;
```

Fine esercizi

Selezionare il nome degli attori che hanno recitato nel film Inception

```
select person.*
from imdb.movie inner join imdb.crew on movie.id = crew.movie
   inner join imdb.person on person.id = crew.person
where official_title ilike 'inception' and p_role = 'actor';
```

Nota: person.* è un modo per indicare di prendere tutti i valori della tabella person Un'altra soluzione all'es utilizzando il concetto di vista:

```
create view imdb.movie_person as (
select
    movie.id as movie_id,
    movie.official_title,
    crew.p_role,
    person.id as person_id,
    person.given_name,
    person.birth_date
from imdb.movie
inner join imdb.crew on movie.id = crew.movie
inner join imdb.person on person.id = crew.person
);
```

Una vista è una tabella virtuale basata sul risultato di una query. In questo caso per visualizare imbd.movie_person dovrei fare:

```
select * from imdb.movie_person
```

Trovare le persone di inception (dalla vista appena creata)

```
select *
from imdb.movie_person
where official_title ilike 'inception' and p_role = 'actor';
```

Esercizio

Selezionare gli attori che hanno recitato in pellicole del 2010

Soluzione personale

```
select m.id, p.given_name, m.year, c.p_role
from imdb.movie m inner join imdb.crew c on m.id = c.movie inner
   join imdb.person p on p.id = c.person
where year = '2010' and c.p_role = 'actor'
```

Fine esercizio

Selezionare le persone che sono decedute in un paese diverso da quello di nascita

```
select l_birth.person, p.given_name, l_birth.country as
   birth_country, l_death.country as death_country
from imdb.location l_birth, imdb.location l_death, imdb.person p
where l_birth.person = l_death.person and l_birth.d_role = 'B' and
   l_death.d_role = 'D' and l_birth.country <> l_death.country and
   l_birth.person = p.id;
```

Sselezionare i film che non hanno materiali associati
A: movie(id, official_title,...)
B: material(id, description, language, movie)
Usiamo la clausola **except** per fare la set difference (except) A-B

```
select movie.id
from imdb.movie
except
select distinct material.movie
from imdb.material;
```

Explain Analyze: per avere informazioni sul tempo di esecuzione o il costo di una query possiamo anteporre alla query il comando `explain analyze` per ottenere queste informazioni, ad esempio riferendoci alla query precedente:

```
explain analyze
select movie.id
from imdb.movie
except
select distinct material.movie
from imdb.material;
```

Esercizio

Selezionare i paesi nei quali non sono prodotti film

Soluzione personale

```
select country.name
from imdb.country
except
select distinct produced.country
from imdb.produced;
```

Fine esercizio

Se invece devo trovare l'intersezione tra due query posso usare l'operatore `intersect` per avere l'intersezione dei risultati di ciascuna query ad esempio:

Trovare le pellicole che sono prodotte in ITA e USA

```
select movie
from imdb.produced
where country = 'ITA'
intersect
select movie
from imdb.produced
where country = 'USA';
```

Vengono trovate prima le pellicole prodotte in italia, poi in USA e viene fatta un'intersezione tra le due. Se volessi anche restituire i titoli:

```
select id, official_title
from imdb.produced inner join imdb.movie on movie = id
where country = 'ITA'
intersect
select id, official_title
from imdb.produced inner join imdb.movie on movie = id
where country = 'USA';
```

Potrei anche semplificare il processo utilizzando una vista come abbiamo trattato in precedenza

```
create view movie_production as (
select *
from imdb.produced inner join imdb.movie on movie = id);

select id, official_title
from movie_production
where country = 'ITA'
intersect
select id, official_title
from movie_production
where country = 'USA';
```

Un'altro modo di effettuare questa operazione è tramite le CTE (common table expressions) che servono a semplificare operazioni complesse in parti più semplici, esempio in questo caso:

```
with mp as (
select *
from imdb.produced inner join imdb.movie on movie = id)
select id, official_title
from mp
where country = 'ITA'
intersect
```

```
select id, official_title
from mp
where country = 'USA';
```

Sto assegnando al termine `mp` una query tramite `with`. Successivamente potrò utilizzare `mp` in altre query
Un'altra soluzione ancora potrebbe essere quella realizzata con le subquery

```
select id, official_title
from imdb.produced inner join imdb.movie on movie = id
where country = 'ITA' and id in (
select movie
from produced
where country = 'USA');
```

Oppure con self-join

```
select usa.movie, official_title
from produced usa inner join produced ita on usa.movie = ita.movie
   inner join imdb.movie on usa.movie = id
where usa.country = 'USA' and ita.country = 'ITA';

select usa.movie, official_title
from produced usa, produced ita, imdb.movie
where usa.country = 'USA' and ita.country = 'ITA' and usa.movie =
   ita.movie and usa.movie = id ;
```

Esercizi

Selezionare i film per i quali esistono materiali multimediali di tipo immagine o materiali testuali di qualche genere

Soluzione personale

```
SELECT DISTINCT m.id, m.official_title
FROM movie m
LEFT JOIN material mat ON m.id = mat.movie -- un film può avere 0
   o più materiali. Non vanno esclusi perchè magari ha 0 materiali
   ma 1 immagine
LEFT JOIN multimedia mm ON mat.id = mm.material -- un film può
   avere 0 o 1 contenuto multimedial
WHERE mm.material IS NOT NULL  -- immagini esistenti -- film che
   hanno almeno un'immagine
   OR (mat.description IS NOT NULL AND mat.description <>
      'cover'); --o almeno un materiale testuale
```

Selezionare i film per i quali esistono materiali multimediali di tipo immagine e materiali testuali di qualche genere

Soluzione personale

```
-- Seleziono le due condizioni separatamente e le interseco
SELECT m.id, m.official_title
FROM movie m
JOIN material mat ON m.id = mat.movie
JOIN multimedia mm ON mat.id = mm.material
```

```
INTERSECT

SELECT m.id, m.official_title
FROM movie m
JOIN material mat ON m.id = mat.movie
WHERE mat.description IS NOT NULL AND mat.description <> 'cover';
```

Fine esercizi

Restituire il titolo dei film con durata superiore alla durata di Inception. Versione con subquery

```
SELECT id, official_title, length
FROM imdb.movie
WHERE length > ANY (
SELECT length
FROM imdb.movie
WHERE official_title = 'Inception');
```

In questo caso ANY confronta length con tutti i valori della sottoquery, trattandosi solo di un film di durata 148 la riga diventa `WHERE length > 148`. Soluzione alternativa:

```
SELECT distinct m1.id, m1.official_title, m1.length
FROM imdb.movie m1 INNER JOIN imdb.movie m2
    ON m1.length > m2.length
WHERE  m2.official_title = 'Inception' order by m1.official\_title;
```

Esercizi

È possibile ottenere il risultato della query con ALL anche con self-join?

Soluzione personale

```
SELECT m1.id, m1.official_title, m1.length
FROM imdb.movie m1
JOIN imdb.movie m2 ON m2.official_title = 'Inception'
GROUP BY m1.id, m1.official_title, m1.length
HAVING COUNT) = COUNT(CASE WHEN m1.length > m2.length THEN 1 END)ORDER BY
   m1.official_title;
```

Restituire le coppie di attori che hanno recitato insieme in almeno due film diversi

Soluzione personale

```
select c1.person as actor1, c2.person as actor2, count(distinct
   c1.movie)
from imdb.crew c1 join imdb.crew c2 on c1.movie = c2.movie
where c1.p_role = 'actor' AND c2.p_role = 'actor' AND c1.person <
   c2.person
group by actor1, actor2
having count(distinct c1.movie) >= 2
```

Selezionare le persone che hanno recitato in film nei quali erano registi

Soluzione personale

```
select c1.person, c1.movie, c1.p_role, c2.p_role
from imdb.crew c1 join imdb.crew c2 on c1.movie = c2.movie and
   c1.person = c2.person
where c1.p_role = 'actor' and c2.p_role = 'director'
```

Selezionare le pellicole prodotte in Italia e Stati Uniti

Soluzione personale

```
select p1.movie, p1.country, p2.country
from imdb.produced p1 join imdb.produced p2 on p1.movie = p2.movie
where p1.country = 'ITA' and p2.country = 'USA'
```

Fine esercizi

Selezionare le pellicole prodotte solo in Italia

```
select p.movie
from imdb.produced p
where p.country = 'ITA' and movie not in (
select movie
from imdb.produced
where country <> 'ITA');

-- Altra soluzione
select p.movie
from imdb.produced p
where p.country = 'ITA'
except
select movie
from imdb.produced
where country <> 'ITA';

-- Soluzione con join
WITH itamovies AS (
select p.movie
from imdb.produced p
where p.country = 'ITA' ),
nonitamovies AS (
select movie
from imdb.produced
where country <> 'ITA')
SELECT DISTINCT itamovies.*
FROM itamovies LEFT JOIN nonitamovies ON itamovies.movie =
   nonitamovies.movie
WHERE nonitamovies.movie IS NULL;

-- Soluzione senza with
SELECT DISTINCT itamovies.*
FROM imdb.produced itamovies LEFT JOIN imdb.produced nonitamovies
   ON itamovies.movie = nonitamovies.movie and
   nonitamovies.country <> 'ITA'
WHERE nonitamovies.movie IS null and itamovies.country = 'ITA';
```

Soluzione con join esterno. Considerare esempio dei seguenti record: - 001 - ITA - 002 - ITA - 002 - USA

Join esterno: restituire il titolo di tutti i film con i relativi generi

```
select id, official_title, genre
from imdb.movie inner join imdb.genre on movie.id = genre.movie
```

A join B on c1: l join interno (inner join) restituisce i record di A e B nel prodotto cartesiano che soddisfano c1. Come possiamo includere nel risultato anche i record di A che non soddisfano mai la condizione di join c1?
Uso left join

```
select id, official_title, genre
from imdb.movie left join imdb.genre on movie.id = genre.movie

-- Equivalente con right join
select id, official_title, genre
from imdb.genre right join imdb.movie on movie.id = genre.movie
```

Esiste un terzo tipo di join esterno: full join

```
select id, official_title, genre
from imdb.genre full join imdb.movie on movie.id = genre.movie
```

Movie

id	title
1	m1
2	m2
3	m3

Genre

movie	genre
1	thriller
1	drama
2	comics

movie join genre on movie.id = genre.movie

id	title	movie	genre
1	m1	1	thriller
1	m1	1	drama
2	m2	2	comics

movie left genre on movie.id = genre.movie

id	title	movie	genre
1	m1	1	thriller
1	m1	1	drama
2	m2	2	comics
3	m3	null	null

Questa versione cosa restituisce? È equivalente a inner join perché non esistono tuple spurie di genre.

```
select id, official_title, genre
from imdb.movie right join imdb.genre on movie.id = genre.movie
```

Per ogni persona mostrare il nome e il country dove è deceduto incluse le persone per le quali non abbiamo un country di decesso. Questa soluzione non è corretta: i record spuri di person non soddisfano mai la condizione di selezione d_role = 'D' e quindi vengono esclusi dal risultato

```
select id, given_name, country, d_role
from imdb.person left join imdb.location on person.id =
   location.person
where d_role = 'D'
order by country
```

Come evitare il problema di tuple spurie di person eliminate per una condizione di selezione sulla tabella location?

```
with deaths as (
select *
from imdb.location
where d_role = 'D')
select id, given_name, country, d_role
from imdb.person left join deaths on person.id = deaths.person
order by country ;

-- Soluzione alternativa
select id, given_name, country, d_role
from imdb.person left join imdb.location on person.id =
   location.person and d_role = 'D'
order by country
```

Trovare le coppie di pellicole che non hanno generi in comune

```
select *
from imdb.genre g1 join imdb.genre g2 on g1.movie <> g2.movie
```

Soluzione con set difference: A. trovo il totale delle coppie, B: trovo le coppie di pellicole con generi in comune e le escludo

```
select distinct g1.movie, g2.movie
from imdb.genre g1 inner join imdb.genre g2 on g1.movie > g2.movie
except
select distinct g1.movie, g2.movie
from imdb.genre g1 inner join imdb.genre g2 on g1.movie > g2.movie
where g1.genre = g2.genre
```

Soluzione alternativa con operatore exists: una coppia m1, m2 è nel risultato se non esiste una coppia di record con m1, m2 dove i generi coincidono

```
with couples as (
select distinct g1.movie as movie1, g2.movie as movie2
from imdb.genre g1 inner join imdb.genre g2 on g1.movie > g2.movie
)
select *
from couples c
where not exists (
select *
from imdb.genre g1, imdb.genre g2
where g1.movie = c.movie1 and g2.movie = c.movie2 and g1.genre =
   g2.genre);
```

Selezionare i film che non sono stati distribuiti nei paesi nei quali sono stati prodotti. Un movie m viene inserito nel risultato se non esiste un record della tabella produced p per il quale esiste un record di release relativo allo stesso movie e paese di produzione

```
select id, official_title
from imdb.movie m
where not exists (
select *
from produced p
where p.movie = m.id and exists (
select *
from imdb.released r
where r.movie = m.id and p.country = r.country));

-- Soluzione alternativa
select id, official_title
from imdb.movie m
where not exists (
select *
from produced p inner join imdb.released r on p.movie = r.movie
   and  p.country = r.country
where p.movie = m.id );
```

Operatori Aggregati

In SQL gli operatori aggregati sono i seguenti:
- min/max
- avg
- sum
- count

E possono essere utilizzati per operazioni come questa: selezionare il film di durata maggiore/minore

```
select max(length) as durata_massima, min(length) as durata_minima
from imdb.movie
```

In questo caso vengono resituiti il massimo e il minimo valore di length presenti in imdb.movie.
Oppure: selezionare il film di durata maggiore o minore e restituire il titolo

```
select id, official_title
from imdb.movie
where length =
(select max(length) as durata_massima
from imdb.movie) or length =
(select min(length) as durata_minima
from imdb.movie)

-- soluzione alternativa
with mmax as (
select max(length) as durata_massima
from imdb.movie),
mmin as (select min(length) as durata_minima
from imdb.movie)
select m.id, m.official_title
```

```
from imdb.movie m, mmax, mmin
where m.length = mmax.durata_massima or m.length =
   mmin.durata_minima;
```

Stampa tutti i film che hanno durata uguale alla durata max o minima in imdb.movie
Trovare il film di durata maggiore fra quelle prodotte nel 2010

```
select max(length) as durata_massima_2010, min(length) as
   durata_minima_2010
from imdb.movie
where year = '2010';
```

Esercizio

Soluzione senza utilizzare l'operatore max. Ai movie con durata non nulla tolgo i movie che hanno durata inferiore ad almeno un altro movie (si ricordi la soluzione in algebra a questo tipo di esercizi)

Soluzione personale

```
select m1.length as durata_massima_2010
from imdb.movie m1
where m1.year = '2010' and m1.length is not null
and not exists(
    select 1
    from imdb.movie m2
    where m2.year = '2010' and m2.length > m1.length
);
```

Fine esercizio

Con `avg` posso restituire ad esempio la durata media delle pellicole. `round` arrotonda il risultato al numero di decimali specificato

```
select round(avg(length), 2)
from imdb.movie m
```

Con `sum` posso restituire ad esempio la durata complessiva delle pellicole del 2010

```
select sum(length)
from imdb.movie m
where year = '2010'
```

Con `count` posso restituire ad esempio il numero di pellicole memorizzate

```
select count( * )
from imdb.movie
```

Restituire il numero di pellicole per le quali è noto l'anno di produzione

```
select count(year)
from imdb.movie

select count)from imdb.moviewhere year is not null;
```

Restituire il numero di pellicole per le quali è noto il titolo

```
select count(official_title)
from imdb.movie
```

Restituire il numero di titoli diversi delle pellicole

```
select count(distinct official_title)
from imdb.movie
```

Trovare la durata media dei film del 2010

```
select avg(length), sum(length), count(length),
   sum(length)::numeric/count(length) as media_calcolata
from imdb.movie
where year = '2010'
```

In questo caso utilizzando `sum` e `count` possiamo dimostrare il calcolo della media dividendoli (che può semplicemente essere fatto con `avg`). Viene inoltre utilizzato `::numeric` per converire il risultato di `sum` a un numerico decimale, così ottieniamo la media calcolata correttamente e senza arrotondamenti (come avverrebbe invece senza esprimere `sum` in numeric).

Nelle operazioni come `sum count` o `avg` che abbiamo appena visto, i valori `NULL` vengono automaticamente esclusi dai calcoli

```
select * from
imdb.movie m
where length is null and year = '2010'
```

Restituire il numero di pellicole per ogni anno disponibile (con ordinamento)

```
select year, count( * )
from imdb.movie
group by year
order by 2 desc
```

In questo caso la query funziona in questo modo: `select year, count(*)` seleziona le colonne degli anni e conta tutte le righe per ogni gruppo, in questo caso per ogni anno come specificato in `group by orphans`. Infinire `order by 2 desc` indica di ordinare in maniera decrescente i valori della seconda colonna, ovvero count(*)

Restituire per ciascun film il numero di persone coinvolte per ciascun ruolo

```
select movie, p_role, count( * )
from imdb.crew
group by movie, p_role
order by movie
```

Restituire anche il titolo della pellicola

```
select movie, official_title, p_role, count( * )
from imdb.crew inner join imdb.movie on movie.id = crew.movie
group by movie, p_role, official_title
order by movie
```

Questa soluzione è equivalente alla precedente?

```
select official_title, p_role, count( * )
from imdb.crew inner join imdb.movie on movie.id = crew.movie
group by p_role, official_title
order by movie
```

No, è cruciale includere la chiave degli oggetti nella clausola di raggruppamento, altrimenti si rischia di raggruppare pellicole diverse con titolo coincidente. Manca `group by movie`

Per ogni film trovare il numero di attori

```
select movie, count(person)
from imdb.crew
where p_role = 'actor'
group by movie
```

Esercizio

Restituire la durata media delle pellicole per ogni anno (con ordinamento)

Soluzione personale

```
select year, avg(length)
from imdb.movie
where length is not null and year is not null
group by year
order by 2 desc
```

Fine esercizio

Restituire il numero di valutazioni per ogni film

```
select movie, count( * ) as numero_recensioni
from imdb.rating
group by movie
union
(select id, 0
from imdb.movie
except
select movie, 0
from imdb.rating)

-- Soluzione ottimizzata
select id, count(rating.movie), count( * ) as numero_recensioni
from imdb.movie left join imdb.rating on movie.id = rating.movie
group by id;
```

Restituire le pellicole che hanno più di 10 attori

```
select movie, count(person)
from imdb.crew
where p_role = 'actor'
group by movie
having count) > 10
```

Esercizio

Restituire le persone che hanno svolto più di un ruolo

Soluzione personale

```
select given_name, count(distinct p_role) as number_of_roles
from imdb.crew c join imdb.person p on c.person = p.id
group by given_name
having count(distinct p_role) > 1
```

Fine esercizio

Restituire il miglior rating di ciascun film

```sql
select id, max(score/scale)
from imdb.movie left join imdb.rating on movie.id = rating.movie
group by id
order by 2 desc;
```

Esercizio

Restituire gli anni nei quali ci sono più di 10 film a partire dal 2010

Soluzione personale

```sql
select year, count(id) as total_movies
from imdb.movie
where year >= '2010'
group by year
having count(id) > 10
order by year
```

Fine esercizio

Selezionare l'attore che ha recitato nel maggior numero di film

```sql
-- questa soluzione è scorretta: non considera eventuali attori
   con il medesimo numero di partecipazioni al valore massimo
select person, count(distinct movie)
from imdb.crew
where p_role = 'actor'
group by person
order by 2 desc

-- soluzione corretta
with recitazioni as (
select person, count(distinct movie) as n_partecipazioni
from imdb.crew
where p_role = 'actor'
group by person),
max_recitazioni as (
select max(n_partecipazioni) as max_partecipazioni
from recitazioni)
select id, given_name, n_partecipazioni
from imdb.person inner join recitazioni on person.id =
   recitazioni.person, max_recitazioni
where n_partecipazioni = max_partecipazioni;

-- soluzione alternativa
select person, count(distinct movie) as n_partecipazioni
from imdb.crew
where p_role = 'actor'
group by person
having count(distinct movie) >= all (
select count(distinct movie)
from imdb.crew
where p_role = 'actor'
group by person
);
```

Esercizi

Selezionare i film con cast più numeroso della media

Soluzione personale

```
WITH movie_cast AS (
    SELECT movie, count(p_role) AS n_cast
    FROM imdb.crew
    WHERE p_role = 'actor'
    GROUP BY movie
),
avg_cast AS (
    SELECT avg(n_cast) AS avg_cast FROM movie_cast
)
SELECT official_title, n_cast
FROM imdb.movie INNER JOIN movie_cast ON movie.id =
   movie_cast.movie, avg_cast
WHERE n_cast > avg_cast
ORDER BY n_cast DESC
```

Selezionare i film nel cui cast non figurano attori nati in paesi dove il film è stato prodotto

Soluzione personale

```
SELECT DISTINCT p.movie, m.official_title
FROM imdb.produced p INNER JOIN imdb.movie m ON p.movie = m.id
WHERE NOT EXISTS (
    SELECT 1
    FROM imdb.crew c
    JOIN imdb.person pers ON c.person = pers.id
    JOIN imdb.location l ON l.person = pers.id
    WHERE c.movie = p.movie AND c.p_role = 'actor' AND l.d_role =
       'B' AND l.country = p.country
);
```

Selezionare il titolo dei film che hanno valutazioni superiori alla media delle valutazioni dei film prodotti nel medesimo anno

Soluzione personale

```
WITH year_reviews AS (
    SELECT year, avg(score) AS average_year_score
    FROM imdb.movie m INNER JOIN imdb.rating r ON m.id = r.movie
    GROUP BY year
)
SELECT official_title, score, y.average_year_score
FROM imdb.movie m
INNER JOIN imdb.rating r ON m.id = r.movie
INNER JOIN year_reviews y ON m.year = y.year
WHERE score > average_year_score
```

Fine esercizi

Selezionare i film con cast più numeroso della media dei film del medesimo genere: consideriamo come cast i ruoli di actor e director e consideriamo una sola volta le partecipazioni ai film con ruoli diversi

```
with movie_cast as (
select movie, count(distinct person) as n_person
from imdb.crew
where p_role in ('actor', 'director')
group by movie),
avg_genre as (
select genre, avg(n_person) as avg_cast
from imdb.movie left join imdb.genre on movie.id = genre.movie
   left join movie_cast on movie.id = movie_cast.movie
group by genre
)
select m.id, m.official_title, g.genre, n_person
from imdb.movie m left join imdb.genre g on m.id = g.movie left
   join movie_cast on m.id = movie_cast.movie
where movie_cast.n_person >
(select avg_cast
from avg_genre
where g.genre = avg_genre.genre);

-- controprova su Crime
-- trovo la media dei cast per le pellicole Crime
with movie_cast(movie, n_person) as (
select movie, count(distinct person)
from imdb.crew
where p_role in ('actor', 'director')
group by movie)
select genre, avg(n_person) as avg_cast
from imdb.movie left join imdb.genre on movie.id = genre.movie
   left join movie_cast on movie.id = movie_cast.movie
where genre = 'Crime'
group by genre
```

Esercizio

Selezionare i film che sono stati distribuiti in tutti i paesi (un film m è nel risultato se non esiste un paese p per il quale non esiste la distribuzione di m in p)

Soluzione personale

```
SELECT m.id, m.official_title
FROM imdb.movie m
WHERE NOT EXISTS (
    SELECT *
    FROM imdb.country c
    WHERE NOT EXISTS (
        SELECT *
        FROM imdb.released r
        WHERE r.movie = m.id AND r.country = c.name
    )
)
```

Fine esercizio

Selezionare le persone che hanno recitato in tutti i film di genere Crime. Data la persona A, non esiste un film di genere Crime in cui A non abbia recitato: se A ha recitato in tutti i movie

Crime, non deve esistere un movie Crime per il quale non esiste la partecipazione di A

```
select id, given_name
from imdb.person p
where not exists (
select *
from imdb.genre g
where g.genre = 'Crime' and not exists (
select *
from imdb.crew
where p_role = 'actor' and p.id = crew.person and g.movie =
   crew.movie));
```

Query Ricorsive

Data una pellicola specifica, suggerire le pellicole simili: sono interessato alla pellicola `0013444`, e i film simili ad essa possono essere trovati con

```
select movie2
from imdb.sim
where movie1 = '0013444';
```

Tuttavia sono simili a 0013444 anche le pellicole simili a quelle restituite dalla query precedente: se `0018756` è una pellicola restituita dalla query sopra, anche le pellicole simili a `0018756` sono indirettamente simili a `0013444`. Possiamo risolvere l'esercizio con una query ricorsiva che esplora il contenuto della tabella sim partendo dai record di `0013444` e visitando in ampiezza la tabella sim come fosse la tabella di adiacenze di un grafo

Un esempio di query ricorsiva per ottenere i numeri da 1 a 10 è la seguente:

```
with recursive t(n) as (
select 1
union
select n+1 from t
where n < 10)
select n from t;
```

Query per restituire tutti i sopra-generi di Poliziesco

```
with recursive search_parent(the_genre, parent_genre) as (
select genre_name, genre_parent
from imdb.genre_taxonomy
WHERE genre_name = 'Poliziesco'
union
select sp.the_genre, gt.genre_parent
from search_parent sp inner join imdb.genre_taxonomy gt on
   sp.parent_genre = gt.genre_name
)
select parent_genre
from search_parent
where parent_genre is not null;
```

Come funziona la query? La query contiene una parte base

```
select genre_name, genre_parent
from imdb.genre_taxonomy
WHERE genre_name = 'Poliziesco'
```

Che seleziona le righe della tabella `genre_taxonomy` dove il genere è Polizesco. Questa query va unita alla parta ricorsiva:

```
select sp.the_genre, gt.genre_parent
from search_parent sp
inner join imdb.genre_taxonomy gt on sp.parent_genre =
   gt.genre_name
```

Che per ogni riga trovata da se stessa `search_parent` cerca nella tabella `genre_taxonomy` il record in cui `genre_name` corrisponde al `parent_genre` trovato alla riga precedente. In pratica risale di un livello della gerarchia trovando il genitore del genitore e così via finchè ci sono sopra-generi (ovvero quando `genre_parent` è `NULL`). Infine con la query finale stampa tutti i risultati `not null`

Un altro esempio: restituire i primi due sopra-generi di Cronaca nera

```
with recursive search_parent(the_genre, parent_genre, distance) as
   (
select genre_name, genre_parent, 1
from imdb.genre_taxonomy
where genre_name = 'Cronaca␣nera'
union
select sp.the_genre, gt.genre_parent, sp.distance+1
from search_parent sp inner join imdb.genre_taxonomy gt on
   sp.parent_genre = gt.genre_name
where distance < 2
)
select parent_genre, distance
from search_parent
where parent_genre is not null;
```

Ora applichiamo una query ricorsiva al database imdb per trovare le pellicole simili a 0013444 fino a una distanza 3 nella tabella sim

```
with recursive search_sim(movie, s_movie, distance) as (
select movie1, movie2, 1
from imdb.sim
where movie1 = '0013444' and movie2 <> '0013444'
union
select ss.movie, si.movie2, distance+1
from search_sim ss inner join imdb.sim si on ss.s_movie = si.movie1
where distance < 3
)
select s.*, m.official_title
from search_sim s inner join imdb.movie m on s.s_movie = m.id;
```

Dove nella funzione ricorsiva la prima parte

```
select movie1, movie2, 1
from imdb.sim
where movie1 = '0013444' and movie2 <> '0013444'
```

Seleziona tutti i film direttamente simili al film dato, e la distanza è 1. Successivamente nella parte ricorsiva

```
select ss.movie, si.movie2, distance+1
```

```
from search_sim ss
inner join imdb.sim si on ss.s_movie = si.movie1
where distance < 3
```

Per ogni film trovato al passo precedente trova film simili, aumentando ogni volta la distanza di 1. La query resituisce solo i film con distanza < 3

Trigger

Una base di dati contiene diverse tipologie di oggetti tra le quali:
- Tabelle
- Viste
- Stored Procedure (function)
- Trigger
- Asserzioni

I trigger (innesco) permettono di definire un comportamento automatico (attivo) della base di dati a fronte di un evento sui dati (ad esempio aggiornare una tabella se eseguo un `UPDATE` o un `INSERT` di un nuovo valore) Ad esempio potrei voler materializzare nel db i conteggi delle persone di ciascun film in ciascun ruolo. Per farlo potrei creare una tabella con i conteggi memorizzati in modo da evitare il calcolo della query ad ogni richiesta, ad esempio:

```
create table movie_counts (
movie varchar references movie(id) on update cascade on delete
   cascade,
p_role varchar,
m_count integer,
primary key(movie, p_role)
);
```

Posso inizializzare la tabella con i valori di base inizializzati a zero con una query come questa

```
insert into movie_counts (movie, p_role, m_count)
with roles as (
  select distinct p_role
  from imdb.crew
),
movie_roles as (
  select id, p_role
  from imdb.movie, roles
)
select movie_roles.id, movie_roles.p_role, count(person)
from movie_roles
left join imdb.crew on movie_roles.id = crew.movie and
   movie_roles.p_role = crew.p_role
group by movie_roles.id, movie_roles.p_role
order by 3;
```

Per aggiornare la tabella devo creare una funzione che sarà utilizzata dal trigger ad ogni aggiornamento. Una funzione potrebbe essere come questa

```
create function do_count_increment() returns trigger as $$
begin

    update movie_counts set m_count = m_count + 1 where movie =
       new.movie and p_role = new.p_role;
```

```
return new;
end;
$$ language 'plpgsql';
$$
```

La funzione `do_count_increment()` aumenta il valore di `m_count` di 1 in `movie_counts` dove il film e il ruolo corrispondono a quelli che verranno recepiti dal trigger. Questa funzione può ora essere chiamata da un trigger apposito che la attiverà al momento giusto

```
create trigger update_counts after insert on imdb.crew for each
   row execute procedure do_count_increment();
```

Quando dichiaro un trigger ho diverse opzioni tra cui scegliere, come scegliere se eseguirlo for each statment/for each row (ovvero se eseguirlo una volta sola per tutte le righe oppure su ogni riga) oppure before/after (per decidere se le operazioni del trigger vadano effettuate prima o dopo l'evento che lo attiva). Ora che la funzione e il trigger sono attivi, effettuando operazioni di inserimento su imdb.crew come ad esempio

```
insert into imdb.crew (movie, person, p_role) values ('1670998',
   '000044', 'director');
```

Il trigger `update_counts` verrà attivato chiamando la funzione `do_count_increment()` che aggiornerà la tabella `movie_counts`

Viste

In SQL una vista è una tabella virtuale che non memorizza dati propri ma mostra il risultato di una query predefinita che ogni volta viene interrogata. Ad esempio una vista per ottengere tutti i film thriller è la seguente

```
create view imdb.thriller_movies as
select * from imdb.genre
where genre = 'Thriller'
with check option;
```

Se a questa vista provo ad eseguire

```
insert into imdb.thriller_movies values ('0044000', 'Thriller');
```

L'operazione andrà a nuon fine in quanto si tratta di un'operazione valida. Se invece tentassi di fare

```
insert into imdb.thriller_movies values ('0044000', 'Super␣
   Horror');
insert into imdb.thriller_movies values ('0044000', 'Super␣
   Comedy');
```

Otterrei degli errori, in quanto il film in questione `0044000` non è ne di genere Super Horror ne Super Comedy, e violerebbero le condizioni della vista

07 - Progettazione

Progettare una base di dati utilizzando lo schema E-R (entity relationship) si divide in diverse fasi:
- **Progettazione Concettuale**: rappresenta le specifiche informali della realtà di interesse in termini di una descrizione formale e completa, indipendente dai criteri di rappresentazione. Il

prodotto di questa fase è detto schema concettuale, e rappresenta i dati ad un'alto livello di astrazione
- **Progettazione Logica**: traduzione dello schema concettuale definito nella fase precedente in termini del modello di rappresentazione dei dati adottato dal sistema di gestione di base di dati a disposizione
- **Progettazione Fisica**: lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione dei dati (organizzazione dei file e degli indici). Il prodotto di questa fase viene denominato schema fisico e fa riferimento a un modello fisico dei dati e dipende dal modello specifico di gestione di basi di dati scelto e si basa sui criteri di organizzazione fisica dei dati in quel sistema

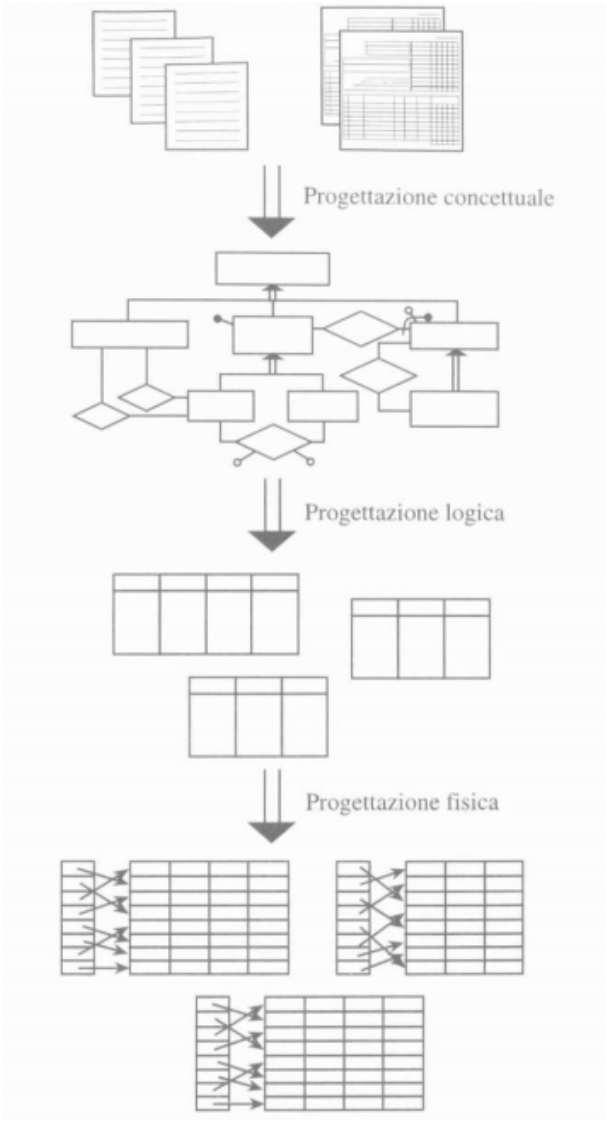

Figure 2: Progettazione

Il modello E-R

Il modello E-R è un modello concettuale di dati che fornisce una serie di costrutti atti a descrivere la realtà di interesse in una maniera facile da comprendere e che prescinde dai criteri di organizzazione dei dati nei calcolatori. Questi costrutti vengono utilizzati per definire schemi che descrivono l'organizzazione e la struttura delle occorrenze dai dati.

I costrutti del modello E-R

- **Entità**: reppresentano classi di oggetti che hanno proprietà in comune ai fini

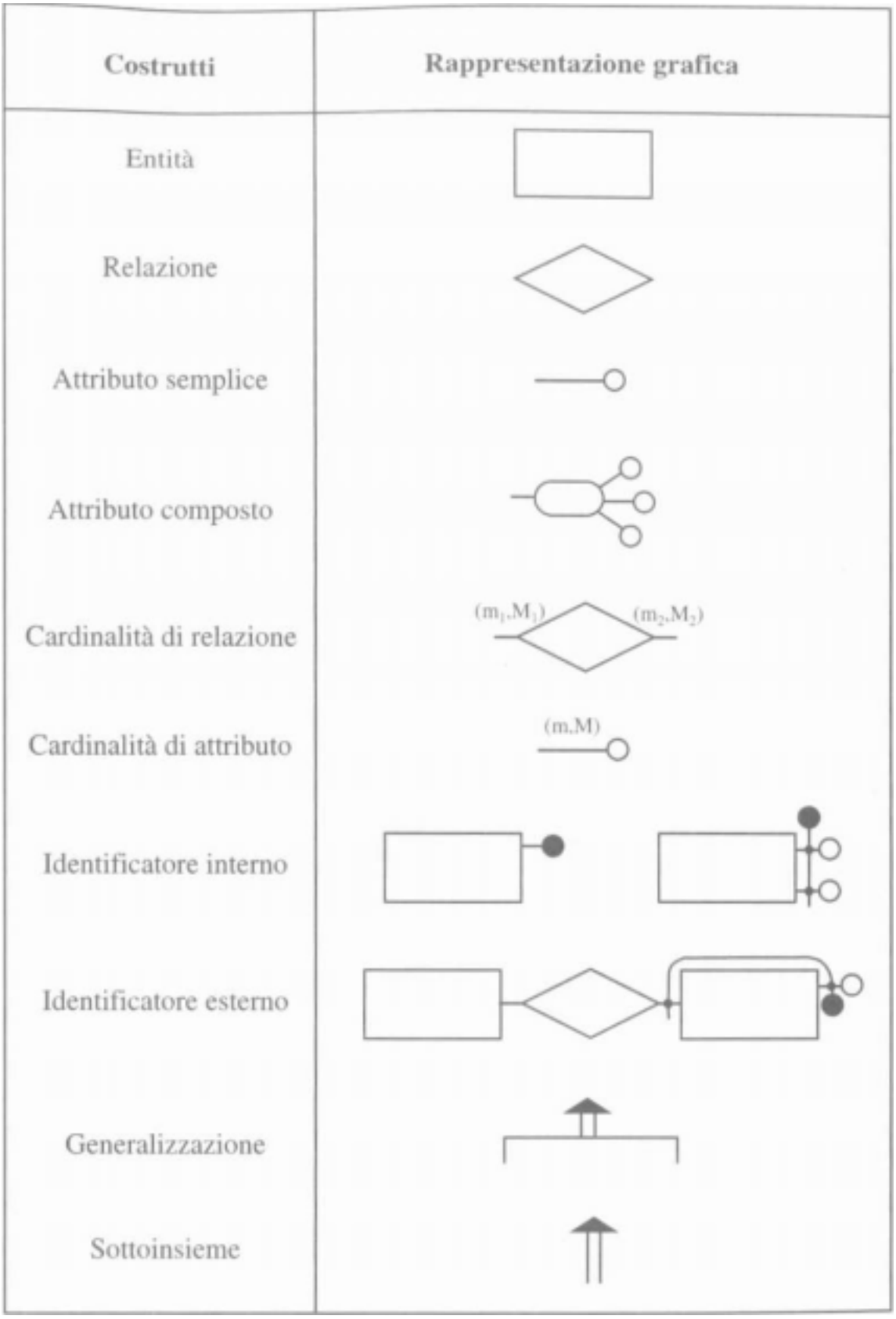

Figure 3: Costrutti modello E-R

dell'applicazione di interesse (es. Città, Dipartimento, Impiegato, Acquisto, Vendita). Un occorrenza di un'entità è un oggetto della classe che l'entità rappresenta: Roma Milano e Palermo sono occorrenze dell'entità Città. Un occorrenza di entità non è un valore che identifica un'oggetto ma l'oggetto stesso, per cui ad esempio un impiegato esiste indipendentemente dal fatto di avere nome cognome o età ecc. Questa è una differenza con il modello relazionale dove per rappresentare un oggetto dobbiamo conoscere alcune sue proprietà. Un entità viene rappresentata tramite un rettangolo con il nome dell'entità all'interno

- **Relazioni**: rappresentano legami logici significativi per l'applicazione di interesse tra due o più entità (ad esempio Residenza può essere una relazione tra Città e Impiegato oppure Esame può essere una relazione tra Studente e Corso). Una relazione viene rappresentata tramire un rombo con il nome della relazione all'interno
- **Attributi**: rappresentano le proprietà elementari di entità o relazioni che sono di interesse ai fini dell'applicazione (es. Cognome, Stipendio ed età sono possibili attributi di Impiegato). Un attributo associa a ciascuna occorrenza di entità un valore appartenente al suo dominio: ad esempio Cognome può avere come dominio le stringhe di 20 caratteri mentrè Età gli interi tra 18 a 65. In certe situazioni può essere comodo raggruppare più attributi insieme: sono gli **attributi composti**, ad esempio l'attributo Indirizzo può essere formato a sua volta da Via Numero Civico e CAP

- **Cardinalità delle relazioni**: vengono specificate per ciascuna partecipazione di entità ad una relazione e descrivono il numero minimo e massimo di occorrenze di relazione a cui l'entità può partecipare. Ad esempio consideranzo una relazione Assegnamento tra le entià Impiegato e Incarico e specifichiamo per impiegato cardininalità (1,5) minimo 1 massimo 5, questo vuol dire che un'Impiegato deve essere assegnato ad almeno un'incarico ma non a più di cinque. Se per Incarico specifico una cardinalità (0,50) questo significa che un'incarico può non essere assegnato a nessuno ma può essere assegnato a massimo 50 persone. La cardinalità viene rappresentata tra parentesi. In teoria si può assegnare un qualsiasi valore alla cardinalità (sempre premettendo che il minimo deve essere minore del massimo) ma nella maggior parte dei casi si usano solo tre valori: zero, uno ed N che indica un numero intero maggiore di uno. In particolare: (0,N) indica partecipazione opzionale, (1,N) indica partecipazione obbligatoria. Le relazioni vengono classificate in base al valore massimo della relazione
  - Uno a Uno: per tutte le entità coinvolte la cardinalità max è 1
  - Uno a Molti: una entità ha max 1 e l'altra ha max M
  - Molti a Molti: tutte le entità hanno N come cardinalità max

- **Cardinalità degli attributi**: sono come le cardinalità delle relazioni ma relative agli attributi, e si rappresentano anche allo stesso modo ovvero con le parentesi. Nella maggior parte dei casi la cardinalità di un attributo è (1,1) e si può omettere di scriverla. Come per la relazioni se la cardinalità mimima è 0 un attributo è detto opzionale mentre se è 1 è detto obbligatorio. Viene chiamato invece multivalore se la cardinalità massima è pari a N.
- **Identificatori delle entità**: vengono specificati per ciascuna entità di uno schema e descrivono i concetti dello schema che permettono di identificare in maniera univoca le occorrenze delle entità. Ad esempio Automobile con attributi Targa Modello e Colore sarà identificata dalla targa in quanto non esistono due auto con la stessa targa, oppure Persona può essere data dal mix di Nome Cognome e Data di Nascita se abbiamo specificato che non possono esserci due persone con lo stesso nome nate la stessa data. Gli identificatori composti vengono indicati con un segno nero.
- **Identificatori Esterni**: gli identificatori esterni sono un particolare tipo di identificatore. Considerando ad esempio uno schema che rappresenta uno studente che si iscrive

ad un'università, potrebbe sembrare che la matricola sia sufficiente a rappresentare univocamente lo studente. Tuttavia ciò non accade perchè dato che il sistema rappresenta più università, potrebbero esserci studenti di diverse università con la stessa matricola. Per questo motivo per identificare uno studente oltre al numero di matricola serve anche la relativa universià. Questo identificatore è reso possibile dalla relazione uno-a-molti tra lo studente e l'università, che permette ad ogni studente di essere associato ad una sola università: se questa relazione non esistesse non sarebbe possibile utilizzre università come identificatore. Un entità E può essere identificata da altre entità solo se tali entità sono coinvolte in una relazione a cui E partecipa con cardinalità (1,1). Università in questo caso è un **identificatore esterno** . Si può quindi affermare che
  - Un identificatore può coinvolgere uno o più attributi, ognuno dei quali deve avere cardinalità (1,1)

  - Un'identificazione esterna può coinvolgere una o più entità ognuna delle quali deve essere membro di una relazione alla quale l'entità da identificare partecipa con cardinalità (1,1)

  - Un'identificazione esterna può coinvolgere un'entità che è a sua volta identificata esternamenta, purchè non vengano generati cicli di identificazioni esterne

  - Ogni entità deve avere almeno un identificatore (interno o esterno) ma ne può avere in generale più di uno. Nel caso di più identificatori gli attributi e le entità coinvolte in alcune identificazioni (tranne una) possono essere opzionali
- **Generalizzazione e Specializzazione**: esse rappresentano legami logici tra più entità dove una è detta genitore e le altre figlie. Quella genitore è detta generalizzazione delle figlie, le figlie sono chiamate specializzazione del genitore. Ad esempio Persona è una generalizzazione di Uomo e Donna (e viceversa Uomo e Donna sono specializzazioni di Persona) oppure Professionista è una generalizzazione di Ingegnere Medico e Avvocato (mentre essi sono specializzazioni di Professionista). Valgono le seguenti proprietà:
  - Ogni occorrenza di un'entità figlia è anche occorrenza dell'entità genitore (es. un'occorrenza di Avvocato è occorrenza di Professionista)
  - Ogni proprietà dell'entità genitore (attributi identificatori ecc) è anche una proprietà delle entità figlie. Ad esempio se Persona ha gli attributi Cognome ed Età, anche Uomo e Donna le avranno. Inoltre l'identificatore Persona è un'identificatore valido anche per le entità Uomo e Donna: questa proprietà delle generalizzazioni è nota con il nome di ereditarietà

Le generalizzazioni vengono rappresentate tramite delle frecce che congiungono le entità figlie con l'entità genitore. Le generalizzazioni possono essere classificate sulla base di due proprietà:
- Una generalizzazione è *totale* se ogni occorrenza dell'entità genitore è una occorrenza di almeno una delle entità figlie, altrimenti è parziale
- Una generalizzazione è esclusiva se ogni occorrenza dell'entità genitore è al più un'occorrenza di una delle entità figlie, altrimenti è sovrpposta

Ad esempio: la generalizzazione Persona Uomo e Donna è totale, perchè gli uomini e le donne rappresentano la totalità delle persone. La generalizzazione Professionista Avvocato Ingenere Dottore è parziale ed esclusiva, perchè ciascun professionista ha una sola professione e oltre a queste tre ce ne possono essere altre. Infine una generalizzazione tra Persona Studente e Lavoratore è parziale e sovrapposta perchè esistono studenti che sono anche lavoratori

08 - Normalizzazione

Dipendenza Funzionale: Una dipendenza funzionale X -> Y tra due sottoinsiemi di attributi X e Y di una relazione R stabilisce un vincolo sulle ennuple che possono formare uno stato di relazione r di R Il vincolo stabilisce che, per ogni coppia di ennuple t1 e t2 in r per cui t1[X] = t2[X], si ha t1[Y] = t2[Y], ovvero t1[X] = t2[X] -> t1[Y] = t2[Y] Se X è una chiave di R, allora X -> Y vale per ogni sottoinsieme Y di attributi di R. X -> Y NON implica Y -> X

Si consideri R1:
R1(person, first_name, last_name, given_name, movie, title, year, character)

X -> Y cioè Y dipende funzionalmente da X
movie -> title cioè title dipende funz. da movie

$\forall\, t_1, t_2 \in R_1$:
t1[X] = t2[X] -> t1[Y] = t2[Y]
t1[movie] = t2[movie] -> t1[title] = t2[title]

first_name, last_name -> given_name – dipendenza non corretta
title -> movie – questa dipendenza è valida solo se title è univoco

Regole di Inferenza

1. **Regola riflessiva:** $X \supseteq Y \models X \to Y$
2. **Regola di arricchimento:** $\{X \to Y\} \models XZ \to YZ$
3. **Regola transitiva:** $\{X \to Y,\, Y \to Z\} \models X \to Z$
4. **Regola di decomposizione:** $\{X \to YZ\} \models X \to Y,\, X \to Z$
5. **Regola di unione:** $\{X \to Y,\, X \to Z\} \models X \to YZ\$$
6. **Regola pseudo-transitiva:** $\{X \to Y,\, WY \to Z\} \models WX \to Z$

Alcuni esempi delle regole applicate:

Regola riflessiva:
X = movie, title
Y = title
movie, title -> title

Regola di arricchimento:
data (movie -> year) allora deduciamo:
movie, title -> year, title

Regola di decomposizione:
data (person -> first_name, last_name) deduciamo:
person -> first_name
person -> last_name

Regola di unione:
date (movie -> year) e (movie -> title) deduciamo:
movie -> title, year

In genere, è il progettista della base di dati a individuare le dipendenze funzionali in base alla propria conoscenza del dominio Dato un insieme iniziale di dipendenze funzionali F, è possibile dedurne altre in base a speci che regole di inferenza L'insieme F+ delle dipendenze funzionali individuate dal progettista unito all'insieme delle dipendenze inferite prende il nome di *chiusura di F*

Normalizzazione di relazioni

Le forme normali sono proprietà delle relazioni definite con riferimento alle dipendenze funzionali che sono soddisfatte quando non sussistono anomalie. Se una relazione non è compatibile con una forma normale, la si decompone in relazioni più piccole che rispettino la forma normale desiderata. L'obiettivo è ottenere uno schema che soddisfare le seguenti proprietà:
- Garantire join senza perdita: se ricostruiamo una relazione dalle sue parti decomposte non dobbiamo generare ennuple non inizialmente presenti - Garantire la conservazione delle dipendenze: ogni dipendenza funzionale deve essere rispettata nello schema normalizzato

Le forme normali sono le seguenti:
- BCNF (Boyce-Codd)
- 3NF
- 2NF
- 1NF

Decomporre uno schema di relazione R significa creare due o più relazioni che sostituiscono R

Le quattro forme normali

Forma Normale di Boyce-Codd

Uno schema di relazione R è in forma normale di Boyce-Codd (BCNF) se, ogni volta che sussiste in R una dipendenza funzionale non banale X -> A, X è una superchiave di R

Normalizzazione: decomporre R in modo che esista una diversa relazione per ogni dipendenza. La chiave di ciascuna relazione sarà il componente di sinistra della dipendenza

Attenzione! **La BCNF non è sempre raggiungibile**

Consideriamo il caso di R1
R1(person, first_name, last_name, given_name, movie, title, year, character)
chiave di R1: (person, movie)

Dipendenze funzionali su R1:
movie -> title
movie -> year
movie -> title, year (per regola di unione)
person -> first_name, last_name
person -> given_name
person, movie -> character

Decomposizione per raggiungere BCNF:
RA(movie, title, year)(PK:movie)
RB(person, first_name, last_name, given_name)(PK:person)
RC(person, movie, character)(PK:person,movie)

Consideriamo un caso alternativo con le seguenti dipendenze:
movie -> title
title -> year
person -> first_name, last_name
person -> given_name
person, movie -> character

La dipendenza (movie -> year) diventa banale perchè derivabile per transitività a partire dalle prime due dipendenze

RA(movie, title)(PK:movie)
RB(title, year)(PK:title)

RC(person, first_name, last_name, given_name)(PK:person)
RD(person, movie, character)(PK:person,movie)

Consideriamo R2
R2(movie, person, country)
La relazione descrive una persona e i film ai quali ha partecipato. Country rappresenta la sede della produzione del film e coincide con il paese di residenza dell'attore durante le riprese

dipendenze:
movie -> country
person -> country

chiave di R2: person, movie

Normalizzazione per BCNF:
RA(movie, country)(PK:movie)
RB(person, country)(PK:person)

RA

movie	country
The Machinist	GBR
Inception	USA
The DK rises	USA
The GG	USA

RB

person	country
Christian Bale	GBR
Marion Cotillar	USA
Leo DiCaprio	USA

Join fra RA e RB:

```
select *
from RA join RB on RA.country = RB.country
```

RA-RB

movie	person	country
The Machinist	Christian Bale	GBR
Inception	Marion Cotillar	USA
Inception	Leo DiCaprio	USA
The DK rises	Marion Cotillar	USA
The DK rises	Leo DiCaprio	USA
The GG	Marion Cotillar	USA
The GG	Leo DiCaprio	USA

Siamo in presenza di un *loss join* (join con perdita). Non è una perdita di record, ma una perdita di consistenza dei dati (ci sono record che non compaiono nella relazione R2). Per verificare se una decomposizione porta a un loss join:

Data R2 e date RA e RB come decomposizione, la decomposizione è lossless se valgono le seguenti regole
- $RA \cup RB = R2$
- $RA \cap RB = R_0$ dove R_0 è chiave in RA o RB

Decomposizione alternativa corretta:
RA1(movie, country)(PK:movie)
RB1(person, country)(PK:person)
RC1(person, movie)(PK:person, movie)

RA1 join RC1 ON RA1.movie = RC1.movie JOIN RB1 ON RC1.person = RB1.person

Decomposizione errata:
RA2(person, country)(PK:person) RB2(person, movie)(PK:person, movie)

insert into RB2 values ('Christian Bale', 'The DK rises');

La dipendenza di (movie -> country) è persa. Il join di RA2 e RB2 produce una relazione diversa da quella di partenza. Di conseguenza possiamo definire il

loseless-join: se divido una relazione in relazioni più piccole, facendo il join tra esse devo riottenere la relazione originale

Terza forma normale (3NF)

Uno schema di relazione R è in terza forma normale (3NF) se, per ogni dipendenza funzionale non banale X -> A di R, è soddisfatta almeno una delle seguenti condizioni:
- X contiene una chiave di R (X è superchiave)
- A appartiene ad almeno una chiave di R

Normalizzazione: decomporre R in modo che esista una diversa relazione per ogni dipendenza. Mantenere una relazione che contenga la chiave della relazione di partenza

E' dimostrato che la 3NF è sempre raggiungibile senza perdita e conservando le dipendenze

Seconda forma normale (2NF)

La seconda forma normale interessa le relazioni che hanno chiave composta e si basa sul concetto di dipendenza funzionale completa: Una dipendenza funzionale X -> Y è completa se la rimozione di qualsiasi attributo A da X comporta che la dipendenza non sia più valida
Una dipendenza X -> Y è parziale se $\exists\, A \in X : (X - A) \to Y$
Uno schema di relazione R è in seconda forma normale (2NF) se ogni attributo non primo A di R dipende funzionalmente in modo completo dalla chiave primaria di R (anche transitivamente)
Un attributo A dello schema R è primo se e solo se fa parte di almeno una chiave di R. In caso contrario A è detto non primo

Normalizzazione: data una chiave primaria composta X, decomporre R realizzando una relazione che conservi X e, per ogni dipendenza parziale (X − A) -> Y , una distinta relazione con schema (X − A) ∪ Y e chiave primaria X − A. >Le relazioni con chiave composta da un solo attributo sono sempre in 2FN

Prima forma normale (1NF)

Uno schema di relazione R(X) è detto in prima forma normale (1NF o at) se ogni attributo appartenente a X è un attributo semplice, cioè atomico. La prima forma normale esclude attributi multivalore e attributi strutturati

Esempi ed esercizi

Consideriamo R3:
R3(movie, country, agency)

La relazione descrive un film con i relativi paesi in cui è stato distribuito e per ciascuno di essi l'agenzia di distribuzione. Si sappia che ogni agenzia è attiva in un solo paese:
R3(movie, country, agency)

Chiave di R3: movie, country

Dipendenze di R3:
movie, country -> agency
agency -> country

Normalizzazione: Non è possibile raggiungere BCNF perchè non posso decomporre senza perdere (movie, country -> agency) e (agency -> country). In questo caso raggiungiamo 3NF poichè R3 è già in 3NF, perchè: - (movie, country -> agency) rispetta BCNF - (agency -> country) rispetta 3NF essendo country (attributo A) parte della chiave di R3

Consideriamo R4
La relazione descrive l'associazione fra attori e film riportando la data di nascita dell'attore
R4(person, movie, birthdate)

chiave di R4? (person, movie)

quali sono le dipendenze funzionali su R4?
person -> birthdate

In quale forma normale siamo? non siamo in BCNF e neppure in 3NF

Decomponiamo R4 in (questa decomposizione soddisfa BCNF):
RA(person, birthdate)(PK:person)
RB(person, movie)(PK:movie, person)

Consideriamo R5
La relazione descrive il luogo di nascita di una persona in termini di città e paese dove le città hanno nome univoco
R5(person, city, country)

chiave di R5: person

dipendenze su R5:
person -> city
city -> country

forma normale? siamo in 2NF (la relazione non ha chiave composta e non soddisfa 3NF a causa di city -> country)

decomposizione:
RA(person, city)(PK:person)
RB(city, country)(PK:city)

Considerazioni applico le regole di inferenza e deduco le seguenti dipendenze:
person -> country (regola transitiva)
person -> city, country (regola di unione)

Se queste ultime dipendenze sono prioritarie, la relazione R5 non può essere decomposta senza perdere la dipendenza (person -> city, country)

Riprendiamo il caso di R1:
R1(person, first_name, last_name, given_name, movie, title, year, character)

chiave di R1: person, movie

movie -> title
movie -> year

movie -> title, year (per regola di unione)
person -> first_name, last_name
person -> given_name
person, movie -> character

La relazione è in 1NF.

decomposizione in BCNF:
RA(movie, title, year)(PK:movie)
RB(person, first_name, last_name, given_name)(PK:person)
RC(person, movie, character)(PK:person,movie)

Consideriamo la relazione 9.11
Un giocatore può giocare in una sola squadra (o nessuna).
Un allenatore può allenare una sola squadra (o nessuna).
Una squadra ha un solo allenatore.
Una squadra ha diversi giocatori.
Una squadra appartiene a una sola città.
R(squadra, allenatore, città, giocatore)

chiave di R: giocatore

dipendenze di R:
squadra -> allenatore
allenatore -> squadra
giocatore -> squadra
squadra -> città

Siamo in 2NF.

Decompongo e ottengo uno schema in 3NF (non posso arrivare a BCNF a causa di allenatore -> squadra)
RA(squadra, allenatore)(PK:squadra)
RB(giocatore, squadra)(PK:giocatore)
RC(squadra, città)(PK:squadra)

Altra decomposizione ancora in 3NF:
RA(squadra, allenatore, città)(PK:squadra)
RB(giocatore, squadra)(PK:giocatore)

09 - Progettazione Fisica

DBMS e sistema operativo

I dati sono memorizzati nella memoria secondaria per questioni di persistenza. Qui sono organizzati in blocchi di dimensione fissa. Anche se sono salvati nella memoria secondaria, la loro elaborazione avviene nella memoria principale

Il **Buffer** permette la gestione dei dati in memoria principale e applica strategie per minimizzare il trasferimento dei dati da elaborare da/verso la memoria secondaria. È organizzato in **pagine** che hanno dimensioni pari a X blocchi

Fattore di blocco (Blocking factor)

Il ***blocking factor*** o fattore di blocco (bfr) rappresenta il numero massimo di record che possono essere memorizzati in un blocco. Si calcola come:

$\text{bfr} = \lfloor \frac{B}{R} \rfloor$

Dove B è la dimensione del blocco, R è la dimensione media del record. Se i record hanno tutti la stessa dimensione in byte allora parliamo di ***record a lunghezza fissa***, altrimenti ***record a grandezza variabile***

Strutture primarie dei file

La struttura primaria stabilisce il criterio che determina la disposizione delle tuple/record nei file. Le strutture primarie possono essere suddivise in tre tipologie in base al metodo d'accesso ai dati:
- Sequenziale
- Calcolato (hash)
- Albero

Strutture ad accesso sequenziale

Nelle strutture sequenziali i file sono costituiti da blocchi logicamente consecutivi e le tuple sono inserite rispettando un criterio sequenziale:

- A: ***Struttura non ordinate (file* heap*)***: È una sequenza indotta dall'ordine di inserimento. L'inserimento è efficiente (prestando attenzione ai vincoli di chiave), la ricerca lineare e la cancellazione logica con periodiche ristrutturazioni
- B: ***Struttura sequenziale ad array***: È possibile solo con record a lunghezza fissa. Il file occupa n blocchi e ciascun blocco ospita m posizioni dell'array. Ogni tupla ha un indice i che determina la posizione della tupla nel file (condizione raramente soddisfatta dai dati). Gli inserimenti e le ricerche sono efficienti
- C: ***Struttura sequenziale ordinata***: L'ordinamento fisico dei dati nel file è coerente con un campo detto pseudochiave o chiave. Le operazioni sono efficienti sul campo chiave (sia per ricerche puntuali sia per selezioni su intervallo). Richiede l'uso di indici per ricerche efficienti (es. dicotomiche). Inserimenti e cancellazioni possono essere costosi, avvenire su un file di overflow e richiedere periodiche ristrutturazioni. Il campo di ordinamento di una struttura ordinata può essere costituito da uno o più attributi della relazione. L'ordinamento avviene sul primo attributo, quindi sui successivi a parità di valore sul primo attributo (e successivamente sui precedenti). Il campo di ordinamento della struttura non è necessariamente la chiave primaria della relazione

Strutture ad accesso calcolato (hash)

Nelle strutture ad accesso calcolato, la posizione di una tupla nel file dipende dal valore assunto da un campo chiave. Si utilizza una ***funzione hash*** h per trasformare il valore del campo chiave k in un indice di posizione nel file (ex. h(k)= k mod N dove N è la numerosità delle posizioni a disposizione).

È applicabile solo con record a lunghezza fissa. La ricerca puntuale è efficiente, la ricerca per intervallo no. Richiede strategie di gestione delle collisioni (catene di overflow)

Esempio di query con criterio di ricerca puntuale:

```
select *
from imdb.country
where iso3 = 'ABW'
```

Esempio di query con criterio di ricerca su intervallo:

```
select *
from imdb.country
where iso3 between 'ABW' and 'ALB'
```

Alberi (radici)

L'organizzazione ad albero può essere impiegata sia per realizzare strutture primarie (ex. strutture contenenti i dati) sia strutture secondarie (ex. strutture ausiliare per favorire l'accesso ai dati memorizzati in altre strutture). Distinguiamo
- Indici primari
- Indici secondari

Indici primari

Un indice primario contiene i dati della relazione (i record) al suo interno, e garantisce l'accesso ai dati in base alla pseudochiave usata come campo di ordinamento dei record determinandone la posizione.

Ciascuna voce dell'indice ha la forma <k, p> dove k è il valore della pseudochiave e p è un puntatore a un'area di memoria. Il puntatore p può fare riferimento all'inizio del blocco dove il record con chiave k è memorizzato, oppure può tenere conto dell'offset all'interno del blocco

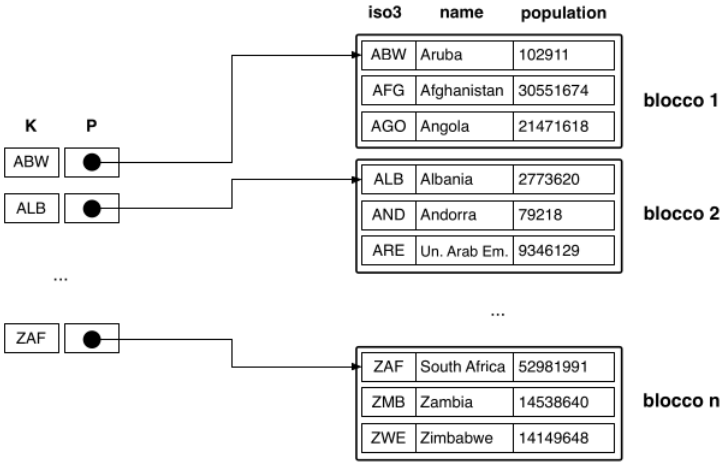

Figure 4: Indice Primario

Il numero delle voci dell'indice primario è uguale al numero di blocchi che costituiscono il file. In genere gli indici possono essere ***densi*** ovvero contenere una voce per ogni valore della pseudochiave, oppure ***sparsi***, ovvero contenere voci solo per alcuni dei valori della pseudochiave

Un'indice primario è sempre un indice sparso, perchè un blocco contiene più record

Esempi utilizzando gli indici primari:

Esempio 1
Record di dimensione fissa R = 100 byte.
File con r = 30.000 record e blocchi B = 1024 byte.
Quante voci conterrà l'indice?

Calcoliamo il blocking factor (cioè quanti record sono memorizzati in ogni blocco):

bfr = $\lfloor \frac{B}{R} \rfloor = \lfloor \frac{1024}{100} \rfloor = 10$ record per blocco

Quanti blocchi sono occupati dai record della tabella:
blocchi = 30000 / 10 = 3000

Quante voci di indice primario mi servono per gestire 3000 blocchi?
In un indice primario, ho bisogno di una chiave per ogni blocco (che punta al primo record del blocco), quindi posso gestire 3000 blocchi con altrettante voci di indice: 3000

Esempio 2
I valori della pseudochiave dell'indice occupano 3 byte.
Il puntatore al blocco è un indirizzo di 6 byte.
Quanti blocchi occupa l'indice?

Ogni record dell'indice occupa 3+6 = 9 byte (dimensione del record dell'indice):

bfr = $\lfloor \frac{B}{R} \rfloor = \lfloor \frac{1024}{9} \rfloor = 113$ record

Quanti blocchi mi servono? considero che l'indice contiene 3000 voci:
blocchi = 3000 / 113 = 27 blocchi

Indici secondari

Un indice secondario fornisce un'ulteriore struttura di accesso a un file per il quale ci sia già un indice primario. Il file contenente i record può essere non ordinato, ad accesso calcolato (hash) o ordinato, ma non rispetto al campo di indicizzazione secondaria

Il campo di ordinamento secondario può essere una chiave (ovvero avere valori univoci) o un qualsiasi attributo della relazione (anche non univoco)

Un'indice secondario è sempre un indice denso, perchè un indice secondario deve contenere tutti i valori del campo di ordinamento

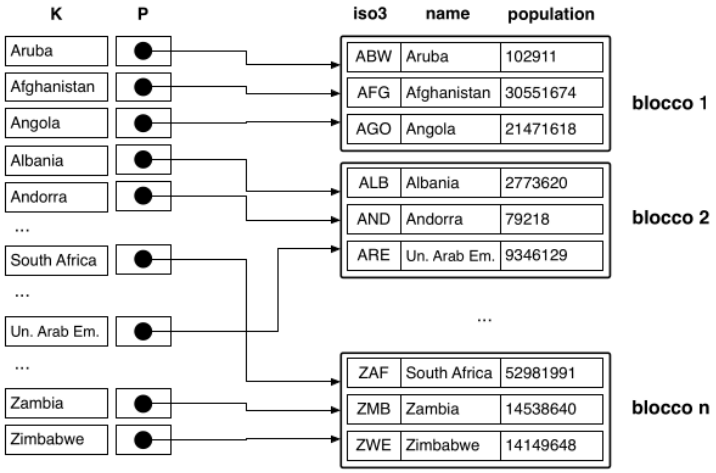

Figure 5: Indice Secondario

Esempio utilizzando gli indici secondari

Consideriamo un file che abbia r = 30.000 record di dimensione R = 100 byte e B = 1024 byte per blocco

Quanti accessi sarebbero mediamente necessari con una ricerca lineare su un file non ordinato? Quanti accessi sarebbero necessari con una ricerca binaria assumendo una dimensione di R = 15 byte per ciascuna voce dell'indice?

In media ho bisogno di 1500 accessi assumendo che la chiave di ricerca sia equiprobabile fra tutti i valori.

Per sapere il numero di accessi, abbiamo bisogno di sapere quanto è grande l'indice, cioè quanti blocchi occupa:

bfr = $\lfloor \frac{B}{R} \rfloor = \lfloor \frac{1024}{15} \rfloor = 68$ record per blocco, cioè voci di indice per blocco

Quanti blocchi per memorizzare l'indice? blocchi = ceil(30000/68) = 442 blocchi

accessi = ceil(log2 442) = 9

In verità gli accessi sono 10: 9 per interrogare l'indice + 1 per caricare il record cercato

Indici secondari su campi non chiave

Se il campo di indicizzazione secondario può avere valori duplicati, vi sono tre opzioni di memorizzazione dell'indice

- Inserire più voci dell'indice con il medesimo valore di K
- Usare un record a lunghezza variabile per l'indice in modo da inserire più puntatori per ogni voce dell'indice
- Mantenere voci a lunghezza fissa, ma inserendo un ulteriore livello per i puntatori

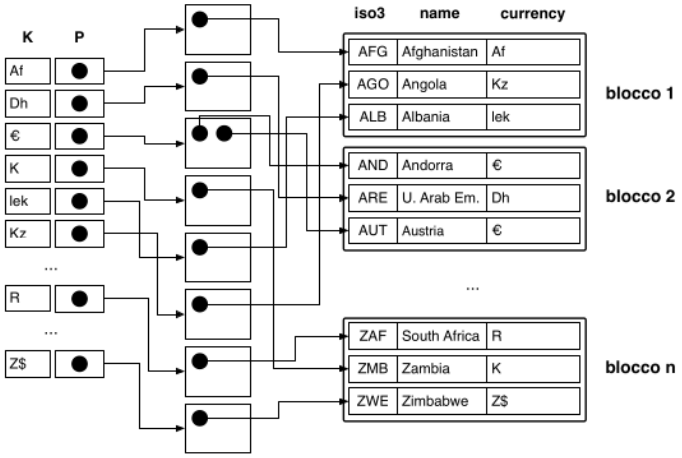

Figure 6: Indice Secondario Non Chiave

Considerazioni

- Gli indici sono di piccole dimensioni
- Le ricerche sui file di indice sono efficienti (occupano poche pagine e possono essere interamente carucati nel buffer)
- Essendo ordinati, gli indici rendono efficienti sia le ricerche puntiali sie le ricerche per intervallo
- Gli indici hanno tempi di accesso logaritmico in funzione del numero di blocchi occupati

Alberi di ricerca

Un albero di ricerca di ordine p è un alberto tale per cui ogni nodo contiene al massimo *p-1* valori di ricerca e i p puntatori sono definiti come:

$\langle P_1, K_1, P_2, K_2, ..., P_{q-1}, K_{q-1}, P_q \rangle$

Vincoli: - in ciascun nodo: $K_1 < K_2 < ... < K_{q-1}$
- per tutti i valori X del sottoalbero a cui si riferisce P_i si ha $K_{i-1} < X < K_i$

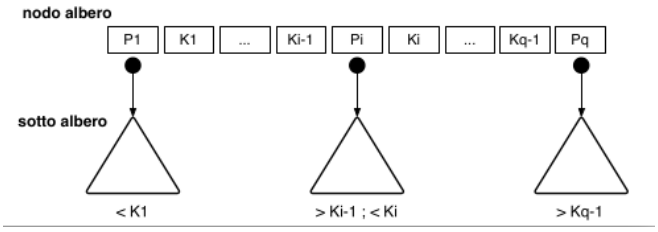

Figure 7: Albero

La caratteristica più importante nella gestione di un alberto di ricerca è mantenerne il bilanciamento in modo che: - I nodi siano distribuiti uniformemememente e la profondità dell'alvero sia minimizzata - Rendere uniforme la velocità in modo che il tempo medio per trovare una qualsiasi chiave sia lo stesso

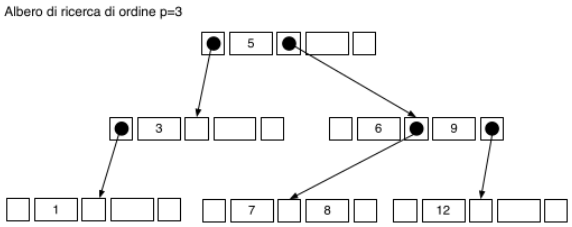

Figure 8: Albero di ricerca di ordine p=3

10 - Sicurezza delle basi di dati

Le basi di dati devono essere dotate di sistemi di sicurezza, principalmente con l'obbiettivo di:

- **Segretezza**: proteggere le informazioni da letture non autorizzate
- **Integrità**: protezione dei dati da modifiche o cancellazioni non autorizzate
- **Disponibilità**: garanzia che l'accesso ai dati sia sempre fornito a chi è un utente legittimo

Questi obbiettivi vengono realizzati mediante varie tecniche:

- **Autenticazione**: meccanismi per verificare l'identità dell'utente che si connette al sistema
- **Controllo dell'accesso**: meccanismi che per ogni richiesta di accesso ai dati verificano che l'utente sia autorizzato a compiere l'accesso
- **Crittografia**: meccanismi che consentono di cifrare i dati in modo che possano essere decifrati solo da utenti autorizzati

Autenticazione e crittografia sono solitamente gestite in maniera esterna, mentre il controllo dell'accesso è gestito a livello di DBMS

Controllo dell'accesso

Il Controllo dell'accesso regola le operazioni che si possono compiere sulle informazioni e le risorse in una base di dati. Lo scopo è limitare e controllare le operazioni che gli utenti effettuano, prevenendo azioni accidentali o deliberate che potrebbero compromettere l'integrità e la segretezza dei dati. Le risorse sono costituite dai dati, memorizzati in oggetti a cui si vuole garantire protezione. I soggetti sono agenti (utenti o programmi in esecuzione) che richiedono di poter esercitare privilegi (come lettura, scrittura o esecuzione) sui dati

Politiche di sicurezza: norme e principi che esprimono le scelte di fondo dell'organizzazione relativamente alla sicurezza dei propri dati

Sono implementate mediante traduzione in un insieme di regole di autorizzazione che stabiliscono le operazioni ed i diritti che gli utenti possono esercitare sui vari oggetti del sistema. Il *Reference Monitor* è un meccanismo di controllo che ha il compito di stabilire se l'utente è autorizzato (totalmente o parzialmente) a compiere l'accesso. La politica di sicurezza adottata dipende principalmente da fattori organizzativi, quali l'ambiente di installazione, le esigenze degli utenti, i regolamenti dell'organizzazione, o i vincoli di natura legale. Principalmente sono di due tipi:

- *Politiche per l'amministrazione della sicurezza*
- *Politiche per il controllo dell'accesso ai dati*

Politiche per l'amministrazione della sicurezza

Stabiliscono chi concede e revoca i diritti di accesso

- *Centralizzata*: un unico soggetto, detto DBA, controlla l'intera base di dati
- *Decentralizzata*: più soggetti sono responsabili del controllo di porzioni diverse dalla base di dati
- *Ownership*: l'utente che crea un oggetto (il proprietario) gestisce le autorizzazzioni sull'oggetto

Politiche per il controllo dell'accesso

Le politiche per il controllo dell'accesso stabiliscono se e come i soggetti possono accedere a quali dati contenuti nel sistema, e se come possono venire trasmessi i diritti di accesso

- *Need-to-know*: privilegio minimo, molto restrittivo, permette ad ogni utente l'accesso solo ai dati strettamente necessari per le proprie attività
- *Maximized Sharing*: massima condivisione, consente agli utenti il massimo accesso alle informazioni nella base di dati, mantenendo comunque informazioni riservate

Need-to-know offre ottime garanzie di sicurezza ed è adatta alla basi di dati con molte esigenze di protezione. Puà portare ad un sistema eccessivamente protetto, negando accessi che non comprometerebbero il sistema. Maximized sharing invece soddisfa il maggior numero di richieste di accesso e viene utilizzata in ambienti dove non è necessaria una forza esigenza di protezione

Tipologie di Sistema

- Sistema aperto: L'accesso è consentito a meno che non sia esplicitamente negato. Le regole di autorizzazione indicano per ogni soggetto i diritti che egli non può esercitare sugli oggetti del sistema: questi diritti sono i soli che gli saranno negati
- Sistema chiuso. L'acceso è permesso solo se esplicitamente autorizzato. Le regole di autorizzazione indicano per ogni soggetto i diritti che egli può esercitare sugli oggetti del sistema questi diritti sono i soli che verranno accordati dal meccanismo di controllo

Un sistema chiuso implementa la politica del minimo privilegio (need to know), un sistema aperto implementa la politica della massima condivisione (maximized sharing). Un sistema chiuso offre maggiori garanzie di sicurezza: una regola inavvertitamente cancellata o non inserita restringe ulteriormente l'accesso, mentre un sistema aperto permette accessi non autorizzati. La maggior parte delle basi di dati oggi esistenti sono sistemi chiusi

La *granularità* dei permessi definisce la tipologia di oggetti a cui il controllo dell'accesso deve essere effettuato. Il requisito minimo è la possibilità di specificare regole di autorizzazione sugli oggetti a cui l'utente può accedere: nelle basi di dati relazionali la granularità è rappresentata dall relazione o dagli attributi di una relazione

Tipologie di controllo

- Controllo dipendente dal nome: l'accesso è basato sul nome dell'oggetto
- Controllo dipendente dal contenuto: l'accesso è subordinato al valore di uno o più attributi dell'oggetto (es., l'utente X può accedere ai dati degli impiegati il cui stipendio non supera una certa soglia)
- Controllo dipendente dal contesto: l'accesso è subordinato al valore di variabili di sistema (es., data, tempo); es., i dati sugli impiegati possono essere acceduti solo in orario di lavoro
- Controllo dipendente dalla storia degli accessi: l'accesso è subordinato alla storia degli accessi eseguiti precedentemente (es., un utente può accedere ad un determinato dato solo se il numero di accessi da lui compiuti su quel dato in un determinato intervallo di tempo non supera una certa soglia)

Politiche discrezionali e mandatorie

Politiche discrezionali: richiedono che vengano specificati i diritti che ogni soggetto possiede sugli oggetti del sistema sottoforma di regole di autorizzazione. Il meccanismo di controllo esamina le richieste di accesso accordando solo quelle che sono autorizzate da una regola. Gli utenti possono a loro discrezione concedere o revocare i diritti di accesso sugli oggetti

Il vantaggio delle politiche discrezionali è il fatto che esse sono estremamente flessibili e si adattano a moltissimi contesti applicativi. Di contro, non impongono restrizioni sull'uso che viene fatto del dato una volta acceduto, non fornendo quindi alcun controllo sul flusso di informazioni del sistema

Si ha un usso tra un oggetto X e un oggetto Y quando si effettua una lettura del valore di X e una scrittura del valore in Y

Politiche mandatorie: per le basi di dati che richiedono elevati livelli di sicurezza (e.g., basi di dati governative) le politiche discrezionali non sono sufficienti. Contenendo diverse informazioni vitali, a diversi livelli di sensitività, i controlli sul usso di dati sono essenziali. Regolano l'accesso ai dati mediante la definizione di classi di sicurezza per i soggetti e gli oggetti del sistema. Le classi di sicurezza sono ordinate parzialmente da una relazione d'ordine. Ogni classe di sicurezza assegnata ad un oggetto rappresenta il livello di sensitività dell'oggetto: maggiore è la classe assegnata ad un oggetto, più ingente sarà il danno derivante dal rilascio delle informazioni in esso contenute a soggetti non autorizzati. La classe di sicurezza assegnata ad un soggetto è una misura del grado di fiducia che si ha nel fatto che tale soggetto non commetta violazioni

Servono a difendere da attacchi sofisticati da parte di utenti (e.g., Cavallo di Troia o Trojan):

Attacco Trojan

User X possiede il file A User Y possiede il file B User Y possiede il file P

P è una stored procedure creata da Y con codice malevolo nascosto

User Y concede il privilegio di esecuzione su P a user X User Y concede il privilegio di scrittura su B a user X Il codice malevolo copia il file A nel file B (lo può fare perchè B gli ha concesso la scrittura)

User Y legge il contenuto di A copiato in B anche se non ne ha il permesso, perchè è stato X ad attivare la procedura P con i suoi privilegi, permettendo a P di copiare A in B

Riassunto Controllo dell'accesso e Politiche mandatorie

Controllo dell'accesso: Il controllo dell'accesso è regolato da una serie di assiomi di sicurezza che stabiliscono le relazioni (in base al modo di accesso considerato) che devono essere verificate fra la classe di un soggetto e quella di un oggetto affinché al primo sia concesso di esercitare un modo di accesso sul secondo. Queste politiche sono applicate in ambienti, ad esempio quello militare, dove la quantità di informazioni da proteggere è elevata, ci sono forti esigenze di protezione ed è possibile classificare rigidamente gli elementi del sistema. I sistemi che adottano una politica mandatoria sono spesso indicati come sistemi multilivello

Politiche mandatorie: Possono essere classificate anche come politiche per il controllo del flusso, poiché evitano che le informazioni una volta accedute vengano trasferite verso oggetti con classificazione inferiore e quindi più accessibili (vedere esempio Cavallo di Troia). C'è un controllo completo sul sistema di autorizzazione. La essibilità è però ridotta e la circolazione di informazioni tra gli utenti è più difficile. Le politiche mandatorie e discrezionali non sono mutuamente esclusive, possono cioè essere applicate insieme. La politica mandatoria non controlla più le richieste di accesso ma le autorizzazioni che vengono assegnate ad un soggetto. Alla politica discrezionale è affidato il compito di controllare le richieste di accesso

Il System R

Il modello implementa una politica di tipo discrezionale e supporta il controllo dell'accesso in base sia al nome che al contenuto. Si tratta di un sistema chiuso: un accesso è concesso solo se esiste una esplicita regola che lo autorizza. L'amministrazione dei privilegi è decentralizzata mediante ownership: quando un utente crea una relazione, riceve automaticamente tutti i diritti di accesso su di essa ed anche la possibilità di delegare ad altri tali privilegi

GRANT

Tramite il comando SQl `GRANT` è possibile concedere privilegi ad altri utenti su una struttura della base di dati, ad esempio tabelle e attributi

```
GRANT Lista Privilegi | ALL [PRIVILEGES] ON Lista
Relazioni | Lista Viste TO Lista Utenti | PUBLIC [WITH
GRANT OPTION]
```

REWOKE

Tramite il comando SQl `REWOKE` è possibile revocare permessi precedentemente concessi

```
REVOKE Lista Privilegi | ALL [PRIVILEGES] ON Lista
Relazioni | Lista Viste FROM Lista Utenti | PUBLIC
```

Un utente può revocare solo privilegi che lui stesso ha concesso. Quando si esegue una operazione di revoca, l'utente a cui i privilegi vengono revocati perde tali privilegi, a meno che essi non gli provengano anche da altre sorgenti indipendenti da quella che effettua la revoca

Delega dei privilegi

La delega dei privilegi avviene mediante la grant option: se un privilegio è concesso con grant option l'utente che lo riceve può non solo esercitare il privilegio, ma anche concederlo ad altri. Un utente può concedere un privilegio su una determinata relazione solo se è il proprietario della relazione, o se ha ricevuto tale privilegio da altri con grant option. Se la clausola WITH GRANT OPTION non è specificata l'utente che riceve i privilegi non può concederli ad altri utenti. Se ne deduce che i privilegi posseduti da un utente sono divisi in:

- delegabili: privilegi concessi con grant option
- non delegabili: concessi senza grant option

SYSAUTH e SYSCOLAUTH

Le regole di autorizzazione specificate dagli utenti sono memorizzate in due cataloghi di sistema di nome `sysauth` e `syscolauth`, implementati come relazioni

id_utente	nome	creator	T	D	I	S	U	GO
bianchi	impiegato	bianchi	R	Y	Y	Y	all	Y
verdi	impiegato	bianchi	R	N	Y	Y	N	Y
rossi	impiegato	bianchi	R	N	N	Y	N	Y
rossi	impiegato	bianchi	R	N	Y	Y	N	N

Figure 9: Relazione Sysauth

Dove:

- T: type (Relation o View)
- D: privilegio `DELETE`
- I: privilegio `INSERT`
- S: privilegio `SELECT`
- U: privilegio di update sulle colonne

- GO: l'utente possiede la Grant Option?

Syscolauth possiede solo la colonna `GO`

Uso del catalogo relazionale

- Quando un utente u esegue un comando di GRANT, il meccanismo di controllo accede ai cataloghi SYSAUTH e SYSCOLAUTH per determinare se u ha il diritto di delegare i privilegi specificati nel comando
- L'insieme dei privilegi delegabili che l'utente u possiede è intersecato con l'insieme dei privilegi specificati nel comando di GRANT
- Se l'intersezione è vuota, il comando non viene eseguito
- Se l'intersezione coincide con i privilegi specificati nel comando, vengono concessi tutti i privilegi specificati
- Altrimenti il comando viene eseguito parzialmente, cioè solo i privilegi contenuti dell'intersezione vengono accordati

Revoca Ricorsiva

L'operazione di revoca di un privilegio è ricorsiva: è revocato il privilegio oggetto del comando di revoca e tutti i privilegi che non avrebbero potuto essere concessi se l'utente specificato nel comando di revoca non avesse ricevuto il privilegio revocato. Un'operazione di revoca del privilegio m sulla relazione R all'utente *u1* da parte dell'utente *u2* ha l'effetto di far perdere a *u1* il privilegio m sulla relazione R (se *u1* non ha ottenuto tale privilegio da altri utenti). Ha inoltre l'effetto di modficare il sistema portandolo in uno stato equivalente a quello in cui si sarebbe trovato se *u2* non avesse mai concesso a *u1* il privilegio di accesso m sulla relazione R

Per attuare la revoca ricorsiva è necessario determinare se un privilegio proviene da fonti indipendenti rispetto a quella specificata nel comando di revoke. Sysauth e Syscolauth sono modificati per mantenere, per ogni privilegio, anche l'utente che ha concesso il privilegio, denominato grantor e il timestamp che denota il tempo in cui il privilegio è stato concesso:

- Il valore 0 indica che l'utente non ha quel privilegio
- Un valore t <> 0 indica che privilegio è stato garantito all'utente al tempo t
- Privilegi garantiti con lo stesso comando di GRANT hanno lo stesso timestamp

11 - Le Transazioni

L'idea di fondo di una transazione è di collocare in un'unica operazione atomica un insieme di operazioni sui dati in modo che si giunga a uno dei due seguenti esiti:

- Tutte le operazioni vanno a buon fine e la transazione è approvata
- Un'operazione va in errore e la transazione è annullata: nessuna delle operazioni nella transazione ha effetto sullo stato della base di dati

Le operazioni interne alla transazione (e il loro esito) non sono visibili ad altre transazioni. Esse servono per:

- Mantenere il corretto comportamento e l'integrità della base di dati anche in presenza di singole operazioni che falliscono
- Mantenere il corretto il comportamento della base di dati anche in presenza di operazioni concorrenti, ovvero eseguite in tempi diversi sugli stessi dati
- Una transazione trasforma lo stato corretto di un database in un altro stato corretto del database. Durante l'esecuzione della transazione, lo stato può essere temporaneamente non corretto
- Al termine della transazione tuttavia gli esiti possibili sono solo due: raggiungimento di un nuovo stato corretto (**COMMIT**) o ritorno a uno stato corretto precedente (**ROLLBACK**)

Concorrenza nell'accesso ai dati

Le operazioni di un DBMS sono eseguite tramite le primitive di read dei dati da memoria secondaria a memoria principale e write dei dati da memoria principale e memoria secondaria. Quando più utenti o applicazioni lavorano in modo concorrente sui dati i tempi di lettura e scrittura possono provocare anomalie:

- **Lost update**: due utenti U1, U2 leggono il medesimo dato X ed eseguono un'operazione di aggiornamento. Poiché U2 legge il dato X prima che U1 scriva, l'aggiornamento effettuato da U1 viene perso
- **Dirty read**: l'utente U1 aggiorna un dato X e successivamente fallisce. Il dato X (temporaneamente) aggiornato da U1 viene letto dall'utente U2 prima che X venga riportato al suo valore originale precedente la modifica effettuata da U1
- **Incorrect summary**: se un utente calcola una funzione aggregata su un insieme di record mentre un altro utente effettua un aggiornamento sui record, il primo client può calcolare la funzione aggregata considerando alcuni valori prima dell'aggiornamento e alcuni valori dopo l'aggiornamento

Proprietà delle transazioni

Il DBMS ha come comportamento predefinito:

- Modalità AUTOCOMMIT: ogni operazione SQL viene considerata una transazione (non servono né BEGIN né COMMIT)
- Quando viene lanciato il comando COMMIT/ROLLBACK la transazione corrente termina e ne può iniziare una nuova
- Il ROLLBACK può essere eseguito automaticamente in caso di errore o anche invocato esplicitamente dall'utente nella de nizione della transazione per catturare eventuali condizioni logiche non soddisfatte

Il DBMS esegue transazioni concorrenti in modo tale da garantire le seguenti proprietà:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Atomicity

- L'esecuzione di una transazione deve essere per definizione totale o nulla e non sono ammesse esecuzioni parziali
- Se si verifica un errore durante l'esecuzione di una transazione prima del COMMIT, gli effetti parziali della transazione non sono memorizzati nel database
- La cancellazione delle modi che parziali di una transazione si definisce ROLLBACK

Consistency

- Ogni transazione può assumere di lavorare su un DB dove tutti i vincoli di integrità sono verificati
- Ogni transazione deve lasciare un DB che soddisfa tutti i vincoli di integrità
- Ai fini delle transazioni, i vincoli possono essere speci cati come `IMMEDIATE` o `DEFERRED`
  - `IMMEDIATE`: verifica di integrità eseguita durante l'esecuzione della transazione. L'istruzione che causa violazione viene cancellata (UNDO) senza imporre un abort della transazione
  - `DEFERRED`: verifica di integrità eseguita al termine della transazione, dopo il COMMIT. Se qualche vincolo viene violato, viene eseguito il ROLLBACK della transazione disfacendo (UNDO) l'intera sequenza di comandi che costituiscono la transazione

Isolation

- Ogni transazione deve essere eseguita in modo isolato e indipendente dalle altre transazioni
- L'eventuale fallimento di una transazione non deve interferire con le altre transazioni in esecuzione
- Le operazioni sono alternate in modo che l'esecuzione sia equivalente a qualche ordine sequenziale (seriale) delle transazioni (nozione di serializzabilità)

Durability

- Anche detta persistency, stabilisce gli effetti delle operazioni di una transazione giunta al COMMIT non debbano essere persi
- Il DBMS mantiene registri di log dove sono annotate tutte le operazioni su un DB
- Questo permette di evitare eventuali perdite di dati dovute a malfunzionamenti nell'intervallo di tempo fra l'impegno a memorizzare le modifiche e l'effettiva scrittura sui dischi da parte del DBMS

Esercizi svolti SQL

I seguenti eserczi sono tratti da *Eserciziario di Basi di Dati - Sara Foresti, Eros Pedrini, Sabrina De Capitani di Vimercati*

Esercizio 1

```
PITTORE(Nome, DataNascita, DataMorte, LuogoNascita)
QUADRO(Titolo, NomePittore, NomeMuseo, Data)
MUSEO(Nome, Indirizzo, Città)
```

Nota: Se un pittore risulta ancora in vita allora la sua data di morte sarà impostata a NULL.

Interrogazione 1
Determinare il nome dei pittori nati a Parigi dopo il 1968 che hanno dipinto almeno tre quadri esposti al 'MOMA' di New York.

```
SELECT p.Nome
FROM Pittore p
JOIN Quadro q ON p.Nome = q.NomePittore
JOIN Museo m ON q.NomeMuseo = m.Nome
WHERE p.LuogoNascita = 'Parigi'
  AND p.DataNascita > 1968
  AND m.Nome = 'MOMA'
  AND m.Città = 'New␣York'
GROUP BY p.Nome
HAVING COUNT) >= 3;
```

Interrogazione 2
Determinare i nomi dei musei che contengono almeno un quadro dipinto fra il 1300 ed il 1600 ma non contengono quadri dipinti da Leonardo da Vinci.

SELECT m.Nome FROM Museo m JOIN Quadro q ON m.Nome = q.NomeMuseo WHERE Data >= 1-1-1300 AND Data <= 31-12-1600 EXCEPT(SELECT *FROM Quadro q WHERE q.NomePittore = 'Leonardo Da Vinci') GROUP BY m.Nome HAVING COUNT()* >= 1

Esercizio 2

```
FESTA(Codice, Costo, NomeRistorante)
REGALO(NomeInvitato, CodiceFesta, Regalo)
INVITATO(Nome, Indirizzo, Telefono)
```

Interrogazione 1
Determinare, per ogni festa, il nome dell'invitato più generoso, ovvero dell'invitato che ha portato il maggior numero di regali.

```
SELECT r.CodiceFesta, r.NomeInvitato
FROM Regalo AS r
GROUP BY r.CodiceFesta, r.NomeInvitato
HAVING COUNT) >= ALL (SELECT COUNT(
            FROM Regalo as r0
            WHERE r0.CodiceFesta = r.codiceFesta
            GROUP BY r0.NomeInvitato)
```

Interrogazione 2
Determinare il nome degli invitati che hanno partecipato alla festa più costosa.

```
SELECT DISTINCT NomeInvitato
FROM Regalo
WHERE CodiceFesta IN (
    SELECT Codice
    FROM Festa
    WHERE Costo >= ALL (
                SELECT Costo
                FROM Festa))
```

Interrogazione 3
Determinare il codice delle feste dove almeno un invitato ha portato tre regali.

```
SELECT DISTINCT r.CodiceFesta
FROM Regalo r
GROUP BY r.NomeInvitato, r.CodiceFesta
HAVING COUNT) >= 3
```

Esercizio 2

```
MEDICO(Codice, Nome, Cognome, Specializzazione, Città, Telefono)
PAZIENTE(CodiceSSN, Nome, Cognome, DataNascita, Città, Telefono)
VISITA(CodiceMedico, CodicePaziente, Data, Diagnosi,
   CodiceMedicinale)
MEDICINALE(Codice, Nome, PrincipioAttivo, Prezzo)
```

Interrogazione 1
Determinare nome e cognome dei pazienti, nati dopo il 1980, che assumono il medicinale 'Aulin' per curare l'emicrania.

```
SELECT DISTINCT P.Nome, P.Cognome
FROM Paziente P
JOIN Visita V ON P.CodiceSSN = V.CodicePaziente
JOIN Medicinale M ON CodiceMedicinale = M.Codice
WHERE DataNascita > "31-12-1979" AND Diagnosi = 'Emicrania' AND
   M.Name = 'Aulin'
```

Interrogazione 2
Determinare il codice sanitario dei pazienti che nel 2005 hanno speso più di 150 euro per curare l'anemia (si consideri la visita contestuale all'acquisto dei medicinali prescritti).

```
SELECT P.CodiceSSN
FROM Paziente P
```

```
JOIN Visita V ON CodiceSSN = CodicePaziente
JOIN Medicinale M ON M.Codice = CodiceMedicinale
WHERE V.DATA >= '1-1-2005' AND V.Data <= '31-12-2005' AND Diagnosi
   = 'Anemia'
GROUP BY CodiceSSN
HAVING SUM(Prezzo) > 150
```

Interrogazione 3
Determinare nome e cognome dei medici cardiologi che non hanno mai visitato pazienti della loro città per problemi di ipertensione.

```
SELECT M.Nome, M.Cognome
FROM Medico M
WHERE Specializzazione = 'Cardiologo'
EXCEPT(
    SELECT M.Nome, M.Cognome
    FROM Medico M
    JOIN Visita V ON CodiceMedico = M.Codice
    JOIN Paziente P ON CodicePaziente = CodiceSSN
    WHERE Specializzazione = 'Cardiologo' AND Diagnosi =
       'Ipertensione' AND M.Città = P.Città
)
```

Esame 10/06/2025 Prova C

Date le seguenti tabelle dello schema relazionale "eCommerce"

```
CREATE TABLE client (
  cid integer PRIMARY KEY,
  name varchar NOT NULL,
  email varchar UNIQUE NOT NULL,
  r_date date
);

CREATE TABLE order (
  oid integer PRIMARY LEY
  cid integer NOT NULL REFERENCES client(cid) ON DELETE CASCADE,
  o_date date NOT NULL,
  status varchar CHECK (status IN ('delivered', 'travelling',
     'canceled'))
);

CREATE TABLE product(
  pid integer PRIMARY KEY,
  name varchar NOT NULL,
  price numeric(6,2) NOT NULL,
  category varchar
);

CREATE TABLE oder_detail(
  oid integer,
  pid integer,
  quantity integer,
```

```
  PRIMARY KEY(oid, pid),
  FOREIGN KEY (oid)
    REFERENCES order(id)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY (pid)
    REFERENCES product(id)
    ON UPDATE CASCADE
);
```

order:

oid	cid	o_date	status
101	?	4-05-20	travelling
102	?	4-06-01	delivered
103	?	4-05-22	canceled
104	?	4-06-04	travelling

product:

pid	name	price	category
1	laptop	?	informatics
2	?	?	equipment
3	?	?	devices

order_detail(S):

oid	pid	qty
101	1	4
101	2	3
101	3	6
102	3	4
102	1	1
103	2	9
103	3	1
104	1	2
104	2	6
104	3	2

Esercizio 1 (8 punti)
A. Quale proprietà differenzia le nozioni di superchiave e chiave
B. Quale comando SQL permette di terminare con successo la transazione corrente?
C. Quelle basate su hashing sono strutture primarie o secondarie?
D. Quale politica di controllo dell'accesso permette ad ogni utente l'accesso solo ai dati strettamente necessari per eseguire la propria attività?
E. Con riferimento alla istanze della pagina precedente, mostrare la relazione risultato della seguente query SQL: `SELECT r.oid FROM order_detail s RIGHT JOIN order R ON r.oid=s.oid AND quantity > 5 WHERE s.pid IS NULL`
F. Scrivere l'espressione di algebra relazionale che restituisce il codice degli ordini che contengono laptop

Esercizio 2 (fino a 15 punti)
Si svolgano le seguenti interrogazioni considerando lo schema relazionale eCommerce:
I. [SQL] Restituire i clienti che non hanno ordini in spedizione (status = “travelling”).
II. [SQL] Restitire la spesa complessiva di ogni cliente. Restituire anche i clienti con spesa a zero (clienti senza ordini). Restituire il nome del cliente nel risultato e ordinare per spesa decrescente.
III. [SQL] Restituire la categoria con il maggior numero di prodotti.
IV. [SQL] Restituire i clienti che hanno comprato la stessa quantità di uno specifico prodotto in due ordini diversi.
V. [ALG] Restituire il nome del cliente con l’ordine in consegna (status = “travelling) più recente (order.o_date più alto).

Esercizio 3 (fino a 5 punti)
Si definisca lo schema relazionale per il seguente schema concettuale. Si minimizzino gli schemi di relazione dove possibile. Si usi la sottolineatura continua per le chiavi primarie, la sottolineatura tratteggiata per le chiavi esterne, l’asterisco per gli attributi potenzialmente nulli.

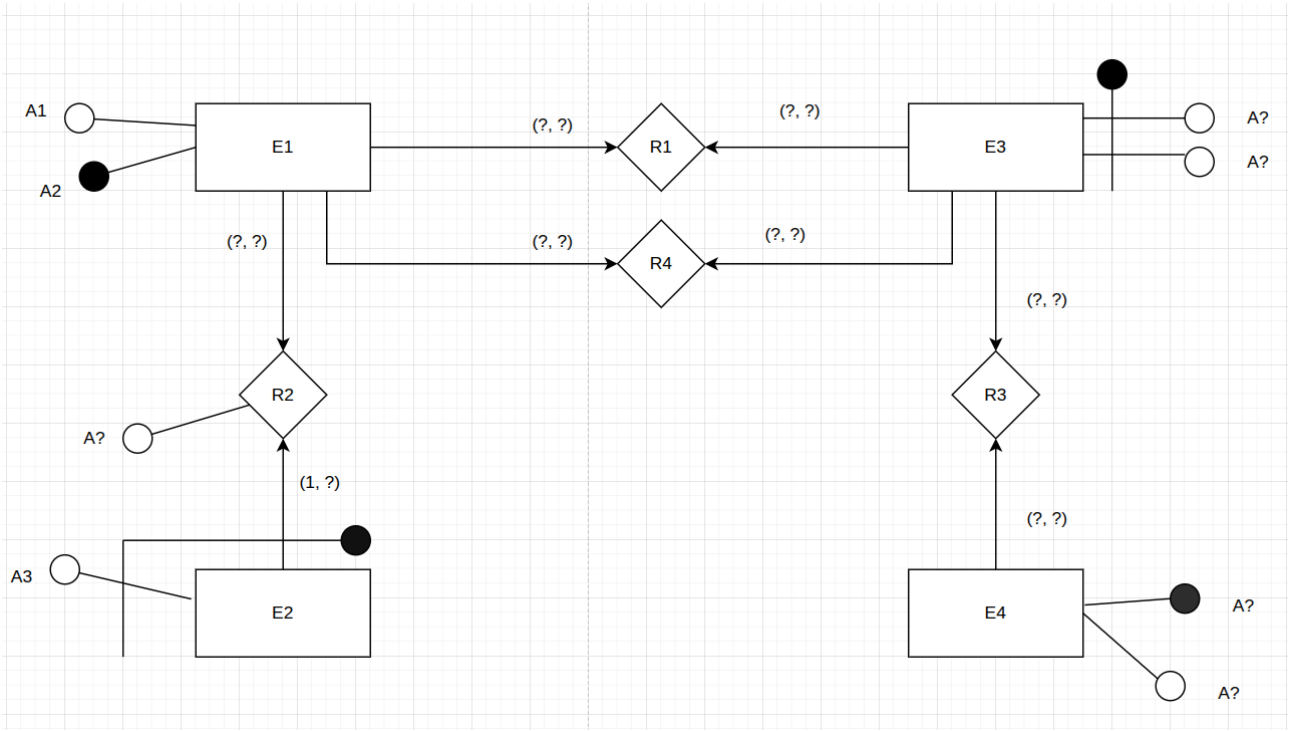

Figure 10: Semplifica

Esercizio 4 (fino a 3 punti)
Si considerti la relazione: R(A, B, C, D, E, F, G, H) dove valgono le seguenti dipendenze funzionali:
d1: A –> B, C, D
d2: C –> D
d3: F –> G, H
d4: A, F –> E
Si fornisca una decomposizione sulla base delle dipendenze date, specificando le chiavi di ciascuna relazione e la forma normale raggiunta con la decomposizione.

SOLUZIONI CHATGPT

Esercizio 1

A. Una superchiave è qualunque insieme di attributi che identifica univocamente le tuple; una chiave (o chiave candidata) è una superchiave minima, ovvero che non contiene nessun

sottoinsieme che a sua volta identifichi univocamente le tuple.
B. Il comando SQL per terminare con successo la transazione corrente è `COMMIT`
C. Le strutture basate su hashing vengono tipicamente usate per indici secondati, perchè l'hashing non perserva l'ordinamento degli attributi
D. La politica è il principio del "need to know" (o least privilege): a ciascun utente vengono concessi solo i permessi strettamente necessari.
E. Un RIGHT JOIN con condizione quantity > 5 e poi WHERE S.pid IS NULL restituisce quegli oid in cui non esiste alcuna riga di order_detail con quantity > 5. Con i dati forniti, l'unico ordine che non ha neanche un dettaglio con quantità > 5 è l'ordine 102, perché: 101 ha 6 unità di pid=3 > 5, 103 ha 9 unità di pid=2 > 5, 104 ha 6 unità di pid=2 > 5. Quindi il risultato è: 102
F.

RESULT = π(pid) [σ(pid = 1) order_detail]

Esercizio 2:

I.

```
SELECT DISTINCT O.cid
FROM order O
WHERE O.cid NOT IN(
    SELECT cid
    FROM order
    WHERE status = 'travelling'
)
```

II.

```
SELECT C.name, COALESCE(SUM(P.price * OD.quantity), 0) AS
   total_spent
FROM client C
LEFT JOIN "order" O ON C.cid = O.cid
LEFT JOIN order_detail OD ON O.oid = OD.oid
LEFT JOIN product P ON OD.pid = P.pid
GROUP BY C.name
ORDER BY total_spent DESC;
```

III.

```
SELECT category
FROM product
GROUP BY category
ORDER BY COUNT) DESCLIMIT 1;
```

IV.

```
SELECT DISTINCT C.cid, C.name
FROM client C
JOIN "order" O1 ON C.cid = O1.cid
JOIN order_detail OD1 ON O1.oid = OD1.oid
JOIN "order" O2 ON C.cid = O2.cid AND O2.oid <> O1.oid
JOIN order_detail OD2 ON O2.oid = OD2.oid AND OD1.pid = OD2.pid
   AND OD1.quantity = OD2.quantity;
```

V.

```sql
SELECT C.name
FROM client C
JOIN order O ON C.cid = O.cid
WHERE O.status = 'travelling'
ORDER BY O.o_date DESC
LIMIT 1;
```

Esercizio 3

```
Entità

E1(
  E1_id          PRIMARY KEY,
  A1             NOT NULL,
  A2             NOT NULL
)

E2(
  E2_id          PRIMARY KEY,
  A3             NOT NULL,
  fk_E1          REFERENCES E1(E1_id)     -- partecipazione a R2
)

E3(
  E3_id          PRIMARY KEY,
  A4             NOT NULL,
  A5             NOT NULL
)

E4(
  E4_id          PRIMARY KEY,
  A6             NOT NULL,
  fk_E3          REFERENCES E3(E3_id)     -- partecipazione a R3
)

Relazioni -MN → tabelle associative

R1(
  fk_E1      REFERENCES E1(E1_id),
  fk_E3      REFERENCES E3(E3_id),
  PRIMARY KEY(fk_E1, fk_E3)
)

R4(
  fk_E1      REFERENCES E1(E1_id),
  fk_E3      REFERENCES E3(E3_id),
  PRIMARY KEY(fk_E1, fk_E3)
)
```

Esercizio 4

Esercizio Creato da ChatGPT

Esercizio B

Inventory(I, J, K, L, M, N, O, P)

I -> J,K
K -> L
M -> N,O
I, M -> P

Tupla (I, M)?

I -> J
I -> K
K -> L
M -> N
M -> O
I, M -> P

R1(I, J) chiave I
R2(I, K) chiave I
R3(K, L) chiave K
R4(M, N) chiave M
R5(M, O) chiave M
R6(I, M, P) chiave I, M

BCNF

Esercizio C

Shipment(S, D, T, U, V, W, X, Y)

S -> D, T
T -> U
V -> W, X
S, V -> Y

Tupla (S, V)?

S -> D, T
T -> U
V -> W, X
S, V -> Y

R1(S, D) chiave S
R2(S, T) chiave S
R3(T, U) chiave T
R4(V, W) chiave V
R5(V, X) chiave V
R6(S, V, Y) chiave S, V

BCNF

- Esercizio 1
  Relazione: Studente(Matricola, Nome, Cognome, Corso, Voto)

Dipendenze funzionali:

Matricola → Nome, Cognome
Matricola, Corso → Voto

Decomposizione:

Matricola -> Nome
Matricola -> Cognome

Matricola, Corso -> Voto

PK(Matricola, Corso)

R1(Matricola, Nome) Key matricola
R2(Matricola, Cognome) Key cognome
R3(Matricola, Corso, Voto) Key Matricola, Corso

Forma Normale: BCNF

- Esercizio 2
  Relazione: Corso(ID_Corso, Nome_Corso, Docente, Aula)

Dipendenze funzionali:

ID_Corso → Nome_Corso, Docente
Docente → Aula

Decomposizione:

ID_Corso -> Nome_Corso
ID_Corso -> Docente
Docente -> Aula

PK(ID_Corso)

R1(ID_Corso, Nome_Corso) Key Id_Corso
R2(ID_Corso, Docente) Key ID_Corso
R3(Docente, Aula) KEY Docente

Forma Normale: BCNF

- Esercizio 3
  Relazione: Ordine(ID_Ordine, ID_Prodotto, Nome_Prodotto, Quantità, PrezzoUnitario)

Dipendenze funzionali:

ID_Ordine, ID_Prodotto → Quantità, PrezzoUnitario
ID_Prodotto → Nome_Prodotto

Decomposizione:

PK(ID_ordine, ID_Prodotto)

R1(ID_Ordine, ID_Prodotto, Quantità, PrezzoUnitario) Key ID_Ordine, ID_prodotto
R2(ID_Prodotto, Nome_Prodotto) Key ID_Prodotto

Forma normale: BCNF

- Esercizio 4 Relazione: Impiegato(CF, Nome, Cognome, Reparto, Sede, CapoReparto)

Dipendenze funzionali:

CF → Nome, Cognome, Reparto
Reparto → Sede, CapoReparto

PK(CF)

BCNF -> No (Reparto -> sede, caporeparto) reparto non è superchiavet
3NF -> No Sede e CapoReparto non sono attributi primi
2NF -> Si
1NF -> Si

Decomposizione

R1(CF, Nome, Cognome, Reparto) Key CF
R2(Reparto, Sede, Caporeparto) Key Reparto

Forma Normale: BCNF

- Esercizio 5 Relazione: Esame(Matricola, CodiceCorso, Voto, Data, NomeCorso, Docente)

Dipendenze funzionali:

Matricola, CodiceCorso → Voto, Data
CodiceCorso → NomeCorso, Docente

PK(Matricola, CodiceCorso)

BCNF -> No
3NF -> No
2NF -> No
1NF -> Si

Decomposizione

R1(Matricola, CodiceCorso, Voto, Data) Key Matricola, CodiceCorso
R2(CodiceCorso, NomeCorso, Docente) Key CodiceCorso

Forma normale: BCNF

Esercizio 5

- Esercizio da risolvere
  Hai la seguente relazione:

Relazione: R(Autore, Libro, Genere)

Dipendenze funzionali:

Autore → Genere
Libro → Genere

Richieste: Trova tutte le chiavi candidate della relazione R.

Determina a quale forma normale appartiene la relazione originale (fino alla più alta possibile).

Se non è in BCNF, decomponila in 3NF:

specifica le nuove relazioni

indica la chiave primaria di ciascuna

Motiva perché non puoi arrivare a BCNF mantenendo sia lossless join sia preservazione delle dipendenze.

1 Autore, Libro 2 2NF, Non è in 3NF perche Autore non è superchiave e Genere non è attributo primo 3

R1(Autore, Libro) Key Autore
R2(Libro, Genere) Key Libro
R3(Autore, Libro) Key Autore, Libro

4 In BCNF non posso mantenere la preservazione delle dipendenze e la loseless join contemporaneamente

10, Jannik: GRANT SELECT, INSERT ON R TO Jannik WITH GRANT OPTION;
20, Jannik: GRANT SELECT, INSERT ON R TO Carlos WITH GRANT OPTION;
25, Jannik: GRANT SELECT, INSERT ON R TO Novak WITH GRANT OPTION;
30, Novak: GRANT SELECT ON R TO Carlos;
30, Novak: GRANT INSERT ON R TO Alexander;

30, Novak: GRANT SELECT, INSERT ON R TO Hubert WITH GRANT OPTION;
40, Hubert: GRANT SELECT, INSERT ON R TO Alexander
45, Carlos: GRANT SELECT, INSERT ON R TO Hubert WITH GRANT OPTION;

id_utente	grantor	T	I	S	D	GO
Jannik	Jannik	R	10	10	0	Y
Carlos	Jannik	R	20	20	0	Y
Novak	Jannik	R	25	25	0	Y
Carlos	Novak	R	0	30	0	N
Alexander	Novak	R	30	0	0	N
Hubert	Novak	R	30	30	0	Y
Alexander	Hubert	R	40	40	0	N
Hubert	Carlos	R	45	45	0	Y

Dopo esecuzione di:

50, Jannik: REVOKE ALL ON R FROM Novak CASCADE;