

IL MODELLO RELAZIONALE

$D_1 \times D_2 \times \dots \times D_n \rightarrow$ PRODOTTO CARTESIANO

GRADO n

La CARDINALITÀ = N° TUPLE CHE APPARTENGONO AL PRODOTTO

UNA RELAZIONE E' UN **insieme** (NO ORDINAMENTO FRA TUPLE)

(NOME ATTRIBUTO, dominio) \rightarrow ATTRIBUTO

Noleggio (colloc, dataNol, codCli, dataRest)

Indica che l'attributo puo' assumere valori nulli

CHIAVI

UNA CHIAVE E' UN INSIEME MINIMALE DI NOMI DI ATTRIBUTI CHE CARATTERIZZANO LE ISTANZE DI UN'ENTITÀ
 \rightarrow UN INSIEME DI ATTRIBUTI DI R E' CHIAVE DI R X VERIFICA ENTROBI LE SEGUENTI PROPRIETÀ

- ① QUALSIASI GIA' L'ISTANZA DI R, NON ESISTONO DUE TUPLE DISTINTE DI R CHE ABBIANO LO STESSO VALORE PER TUTTI GLI ATTRIBUTI IN X
- ② NESSUN SOTTOINSIEME PROPRIO DI X VERIFICA LA PROPRIETÀ ①

• **SUPERCHIAVE**: INSIEME DI NOMI DI ATTRIBUTI CHE VERIFICA LA PROPRIETÀ ①, MA NON LA PROPRIETÀ ②

QUANDO UNA RELAZIONE PRECISA PIÙ DI UN INSIEME DI ATTRIBUTI CHE VERIFICANO LE 2 PROPRIETÀ PRECEDENTI SI E' NEI CASO DELLE **CHIAVI CANDIDATE** $\xrightarrow{\quad}$ **CHIAVE PRIMAVERA** \rightarrow NO VALORI NULLI $\xrightarrow{\quad}$ **CHIAVE ASCENDENTE** \rightarrow SI VALORI NULLI

• UNA RELAZIONE HA SICURAMENTE ALMENO UNA CHIAVE

- UNA TUPLA E' IDENTIFICATA DAL VALORE DI UNA OLTRENCHE' CHIAVE CANDIDATA
- LA CHIAVE PRIMAVERA VIENE UTILIZZATA DAL DBMS PER OTTIMIZZARE LE OPERAZIONI

\rightarrow COME SCEGLIERE LA CHIAVE PRIMAVERA:

- CHIAVE CANDIDATA CON IL MINOR NUMERO DI ATTRIBUTI
- CHIAVE CANDIDATA PIÙ FREQUENTE NEGLI UTILIZZATORI NELL'INTERROGAZIONI

CHIAVI ESTERNE

R (A, B, C, D, E)

(D, E) e (F, G) \rightarrow DOMINI COMPATIBILI

(D, E) PUO' ESSERE DEFINITA COME CHIAVE ESTERNA DI R SU R'

R					R'			
A	B	C	D	E	F	G	H	I
a1	b1	c1	d1	e1	d1	e1	h1	i1
a2	b1	c2	d2	e1	d1	e2	h2	i2
a3	b2	c3	d1	e1	d2	e1	h1	i3
a4	b2	c3	d2	e3	d2	e4	h3	i4

le chiavi esterne permettono di collegare tra loro tuple di relazioni diverse

↳ MECCANISMO PER REALIZZARE LE ASSOCIAZIONI PER VALORE → **VINCOLO DI INTEGRITÀ REFERENZIALE**

ESEMPIO

Video(colloc,titolo,regista,tipos)

Film					
titolo	regista	anno	genere	valutaz	
underground	emir kusturica	1995	drammatico	3.20	
edward mani di forbice	tim burton	1990	fantastico	3.60	
nightmare before christmas	tim burton	1993	animazione	4.00	
ed wood	tim burton	1994	drammatico	4.00	
mars attacks	tim burton	1996	fantascienza	3.00	
il mistero di sleepy hollow	tim burton	1999	horror	3.50	
big fish	tim burton	2003	fantastico	3.10	
la sposa cadavere	tim burton	2005	animazione	3.50	
la fabbrica di cioccolato	tim burton	2006	fantastico	4.00	
io non ho paura	gabriele salvatores	2003	drammatico	3.50	
nirvana	gabriele salvatores	1997	fantascienza	3.00	
mediterraneo	gabriele salvatores	1991	commedia	3.80	
pulp fiction	quentin tarantino	1994	thriller	3.50	
le iene	quentin tarantino	1992	thriller	4.00	

Video

colloc	titolo	regista	tipo
1111	underground	emir kusturica	v
1112	underground	emir kusturica	d
1113	big fish	tim burton	v
1114	big fish	tim burton	d
1115	edward mani di forbice	tim burton	d
1116	nightmare before christmas	tim burton	v
1117	nightmare before christmas	tim burton	d
1118	ed wood	tim burton	d
1119	mars attacks	tim burton	d
1120	il mistero di sleepy hollow	tim burton	d
1121	la sposa cadavere	tim burton	d
1122	la fabbrica di cioccolato	tim burton	d
1123	la fabbrica di cioccolato	tim burton	d
1124	io non ho paura	gabriele salvatores	d
1125	nirvana	gabriele salvatores	d
1126	mediterraneo	gabriele salvatores	d
1127	pulp fiction	quentin tarantino	v
1128	pulp fiction	quentin tarantino	d
1129	le iene	quentin tarantino	d

Vincolo di integrità referenziale soddisfatto

Noleggio

colloc	dataNol	codCli	dataRest
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006

Cliente(codCli, nome, cognome, telefono, dataN, residenza)

Noleggio(colloc, dataNol, codCli, dataRest)

Vincolo di integrità referenziale NON soddisfatto

Cliente

codCli	nome	cognome	telefono	dataN	residenza
6610	anna	rossi	01055664433	05-Ott-1979	via scribanti 16 16131 genova
6635	paola	bianchi	0104647992	12-Apr-1976	via dodecaneso 35 16146 genova
6642	marco	verdi	3336745383	16-Ott-1972	via lagustena 35 16131 genova

1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1112	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1116	21-Mar-2006	6610	?
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1126	22-Mar-2006	6655	?

VINCOLI DI INTEGRITÀ REFERENZIALE

• L'INTEGRITÀ REFERENZIALE PUÒ ESSERE VIOLATA

↳ NELLA RELAZIONE REFERENTE DA:

- INSERIMENTI E MODIFICHE DEL VALORE DEI CUI LIANE ESTERNA

↳ NELLA RELAZIONE REFERITA DA:

- CANCELLAZIONI E MODIFICHE DEL VALORE DEI CUI LIANE

→ I NOMI DEGLI ATTRIBUTI NELLA CLAVELA E NELLA CLAVELA ESTERNA NON DEVONO NECESSARIAMENTE ESSERE GLI STESSI (se lo sono, semplificano alcune operazioni)

• PER INDICARE IL CLAVELA ESTERNO SI USA IL NOME DELLA RELAZIONE REFERITA CON ALTI DEI NOMI DEGLI ATTRIBUTI NELLA TABELLA REFERENTE, QUANDO NON CI SIANO AMBIGUITÀ

• UNA RELAZIONE PUÒ CONTENERE PIÙ CLAVELA ESTERNE ANCORA SULLA STESSA RELAZIONE

• SIA LE CLAVELA CHE LE CLAVELA ESTERNE DEVONO ESSERE ESPlicitAMENTE SPECIFICATE NELLO SCHEMA DI RELAZIONE

↳ ATTRIBUTI CON LO STESSO NOME E DIVERSI COMPARTIMENTI IN RELAZIONI DIVERSE NON IMPONE DI NON CLAVELA ESTERNO

• LE CLAVELA ESTERNE POSSANO ABBINARE VALORE NULLO

Modello CINTA' - Relazione

Modellazione astratta dei dati

- **ENTITA'** =insieme di elementi del mondo reale caratterizzati da caratteristiche comuni [Riferimento]
 - **ASSOCIAZIONE** = corrispondenza tra entita' non necessariamente distinte, composta da ciascuna delle entita' da mettere in corrispondenza [esempio]

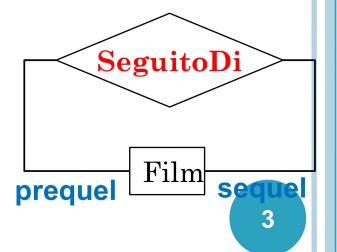


[Rectangular]

~~VERA'S~~ IN FEBR.

- **Ruolo**: funzione che un'istanza di entità esercita nell'ambito di un'associazione

↳ e' indispensabile che la stessa entita' compare piu volte nell'elenco



GRADO DI UN'ASSOCIAZIONE. Numero di entità che partecipano ad una associazione

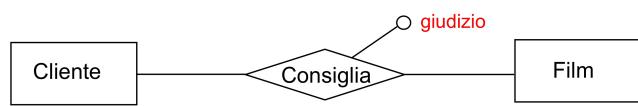
- QUAN TUNE BINARIE
 - GMDO BINARIO NON PUÒ ESSERE ALTO (~ 3)

- **ATTRIBUTI**: per ognuna istanza, un attributo associa un insieme di valori (nella maggior parte dei casi un solo valore)
 - si possono esprimere vincoli di cardinalità di questo insieme

- OPZIONALI \rightarrow CARDINALITÀ MINIMA E' 0
 - OBBLIGATORI \rightarrow CARDINALITÀ MINIMA E' 1
 - MONO - VALORE \rightarrow CARDINALITÀ MASSIMA E' 1
 - MULTI - VALORE \rightarrow CARDINALITÀ MASSIMA > 1

Se si ottengono i vincoli, si definisce c^* $(1,1)$

→ Ainsi le association possède une attribut

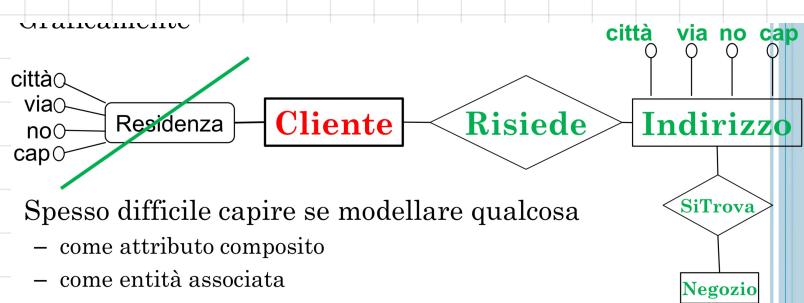


↳ IL GIUDIZIO NON E' SPECIFICO NE' DI UN CLIENTE NE' DI UN FILM, MA DEL LEGANCI Cliente - Film che si crea quando un cliente consiglia un certo film

- ## • Активного Сопротивления

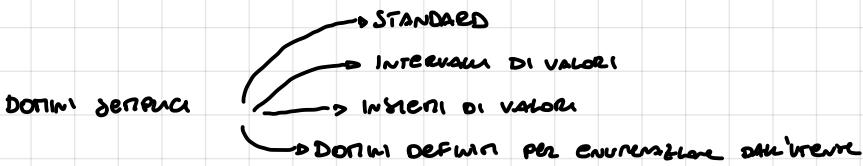
POSSEDE DEL DODICI ATTIVITÀ

→ se lo stesso tipo dell'attivismo si muove negli intrecci di un'entità



DOMINIO DI UN ATTRIBUTO (definisce l'insieme dei valori che un attributo può assumere)

Le informazioni sul dominio di un attributo non sono direttamente rappresentabili in un diagramma ER, ma sono fondamentali per una corretta modellazione



Dichiarazione per gli attributi dell'entità Cliente

codCli: int
nome: string
cognome: string
dataN: date
residenza: string × string × string × int



Dichiarazione per l'attributo dell'associazione Consiglia

giudizio: dom_giudizio
dom_giudizio: [0,4]



- ANCHE I DOMINI DEGLI ATTRIBUTI SONO VINCOLI DI INTEGRITÀ

VINCOLO DI CARDINALITÀ PER ASSOCIAZIONI

Specifica minimo e massimo numero di istanze di un'associazione a cui ogni istanza di quella entità deve partecipare



Se okesso, c'è come aver indicato (0, n)

Un cliente può:

- Non avere in noleggio video ($c_{min} = 0$)
- Averne contemporaneamente in noleggio non più di 3 ($c_{max} = 3$)

Un video può:

- Non essere correntemente in noleggio ($c_{min} = 0$)
- Essere noleggiato da non più di un cliente contemporaneamente ($c_{max} = 1$)

15

VINCOLI DI IDENTIFICAZIONE

- Insieme di attributi e/o identità che identificano le istanze dell'entità (analogo a superchiavi)

- Un identificatore si rinvia se qualcun altro non c'è un identificatore (analogo a chiavi)

- Durante la generalizzazione concettuale per ogni entità si devono identificare tutti gli identificatori minimi

- Gli identificatori hanno senso solo per le entità e non per le associazioni

→ SONO SEMPRE IDENTIFICARE DALLE ISTANZE DI IDENTITÀ CHE METTONO IN COLLEGAMENTO

A VOLTE NON C'È POSSIBILE IDENTIFICARE UNA ISTANZA DI ENTITÀ solo sulla base dei suoi ATTRIBUTI, CHE' DICHIARAZIONE DIVERSA POSSONO CONCIDERE SU TUTTI GLI ATTRIBUTI

→ SI UNISCE ALLORA IL FARLO CHE TALE ISTANZA APPARTI AD UNA PARTICOLARE ISTANZA DI AGGREGAZIONE CON UNA DATA

ISTANZA DI UN'ALTRA ENTRATA

Lo Analogico a usare una curva continua come parte della curva nel modello relazionale

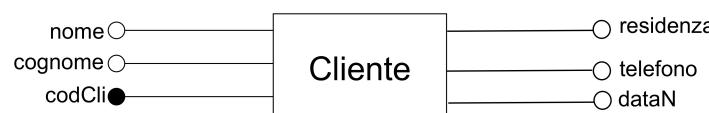
L'ENTRATA IDENTIFICATA IN QUESTO MODO VIENE DETTA ENTRATA DESDE



TIPOLOGIE DI IDENTIFICATORI

- INTERNI** UNO O PIÙ ATTRIBUTI DELL'ENTRATA'
- ESTERNI** UNO O PIÙ ENTRATE COLLEGATE DA UN'ASSOCIAZIONE ALL'ENTRATA A CUI SI RIFERISCONO
- MISTI** COMBINAZIONE DI DUE TIPOLOGIE PRECEDENTI
- SEMPLICI** UN SOLO ELEMENTO
- COMPOSTI** PIÙ DI UN ELEMENTO

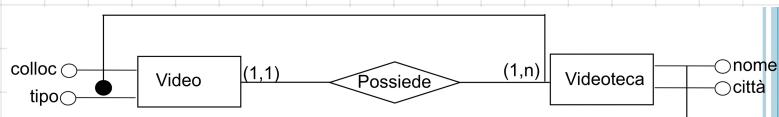
=> DALLE 6 COMBINAZIONI SOLO 5 SONO POSSIBILI



INTERNO
SIMPLICE



INTERNO
COMPOSTO



MISTO
COMPOSTO



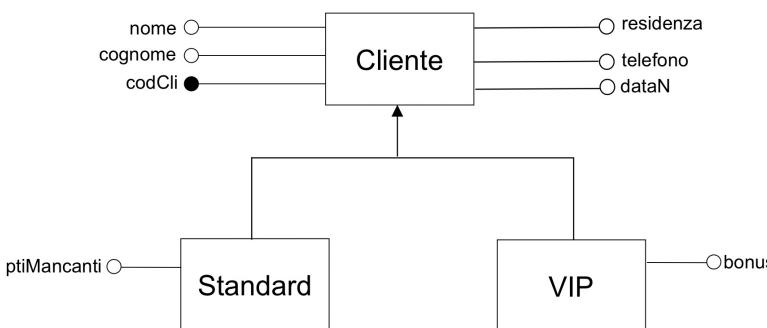
ESTERNO
SIMPLICE

LE ENTRATE DEBONO DENEVRE CALCOLABILITÀ (1,1) RISPETTO ALL'ASSOCIAZIONE ATTROVERSO DI ANCHE L'IDENTIFICAZIONE

- NEL CASO DI IDENTIFICAZIONE ESTERNA L'ASSOCIAZIONE SARÀ UNO A UNO
- NEL CASO DI IDENTIFICAZIONE MISTA L'ASSOCIAZIONE SARÀ UNO A MOLTI

GERARCHIE DI GENERALIZZAZIONE

- TOTALE** = OGNI Istanza di E È Istanza DI ALMENO UN'ENTRATA E.



- PARTIALE** = ESISTE ALMENO UN'ISTANZA DI E CHE NON È Istanza DI ALCUNA ENTRATA E;

LA GENERALIZZAZIONE PUÒ ANCHE ESSERE WOLTRRE:

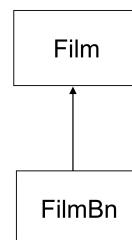
- **ESCLUSIVA**: OGNI ISTANZA DI E È UN'ISTANZA DI AL PIÙ UN'ENTITÀ Ei
- **CONDIVISA**: ESSERE ALMENO UN'ISTANZA DI E CHE È UN'ISTANZA DI PIÙ DI UN'ENTITÀ Ei

- TOTALI ESCLUSIVE
- TOTALI CONDIVISE
- ↳ PARTIZIALI ESCLUSIVE
- PARTIZIALI CONDIVISE

Le informazioni sulle tipologie di generalizzazioni presenti in uno schema ER vanno inserite nella documentazione a corredo dello schema.

CASO PARTICOLARE DI GENERALIZZAZIONE: **ASSOCIAZIONE DI SOTTOinsieme**

Componente	Simbolo
Entità	□
Relazione	◇
Attributo	—○
Attributo composito	○—○
Gerarchia di generalizzazione	□↑
Relazione di sottoinsieme	↑
Identificatore	—●
Vincolo di cardinalità	(c_min,c_max)



PROGETTAZIONE LOGICA

• FASE DI RISTRUTTURAZIONE

- Genera lo schema ER ristrutturato, scelta semplificata, in corrispondenza a quei che si preferisce
- Ha lo scopo di semplificare la modellazione successiva, eliminando dallo schema i contenuti non rappresentabili direttamente nel modello relazionale
- LA TRADUZIONE NON È SEMPRE UNIVOCHE: SCHEMI VENGONO FATI SUA BASE DELLE CONIGNE

• FASE DI TRADUZIONE

- TRADUCE LO SCHEMA ER RISTRUTTURATO IN UNO SCHEMA RELAZIONALE CONVENIENTE
 - REGOLE PREDEFINITE PER LA TRADUZIONE
 - SI POSSONO APPLICARE PIÙ REGOLE
- TRADUZIONE NON SEMPRE UNIVOCHE: BIOGNA CONSIDERARE PROGRESSIONI

RISTRUTTURAZIONE

ELIMINAZIONE DALLO SCHEMA ER DEI CONTENUTI NON DIRETTAMENTE RAPPRESENTABILI NEL MODELLO RELAZIONALE

- ATTRIBUTI COMPOSTI
- ATTRIBUTI CON MONTEPLICITA' > 1
- GENERALIZZAZIONI

RISTRUTTURAZIONI PER MIGLIORARE LE PRESTAZIONI, SUBCERTE DALL'ANALISI DEL CARICO DI LAVORO

- ANALISI DELLA RIDONDANZA
- PARTIZIONAMENTO/ACCORPAMENTO DI ENITÀ

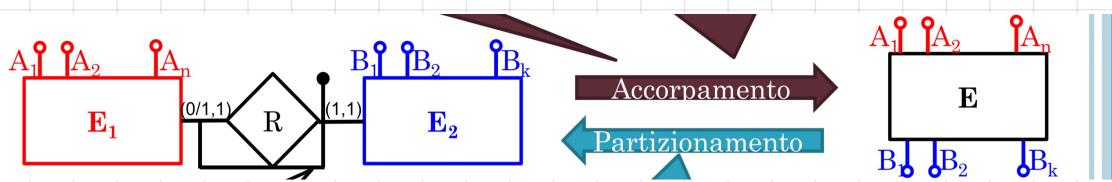
→ LA RIDONDANZA DEVE ESSERE

- LIMITATA AI CASI IN CUI SIA POSSIBILE OTTENERE UN SIGNIFICATIVO BENEFICIO IN TERMINI DI TEMPO DI ESECUZIONE DI INTERROGAZIONI ESEGUITE FREQUENTEMENTE

• ESPlicitata nella documentazione

• GESTITA AUTOMATICAMENTE DAGLI AGGIORNAMENTI

PARTIZIONAMENTO / ACCORPAMENTO DI ENITÀ



• SI POSSONO ESEGUIRE DURANTE LA FASE DI RISTRUTTURAZIONE

COSTRUZIONE DEGLI ATTRIBUTI COMPOSTI

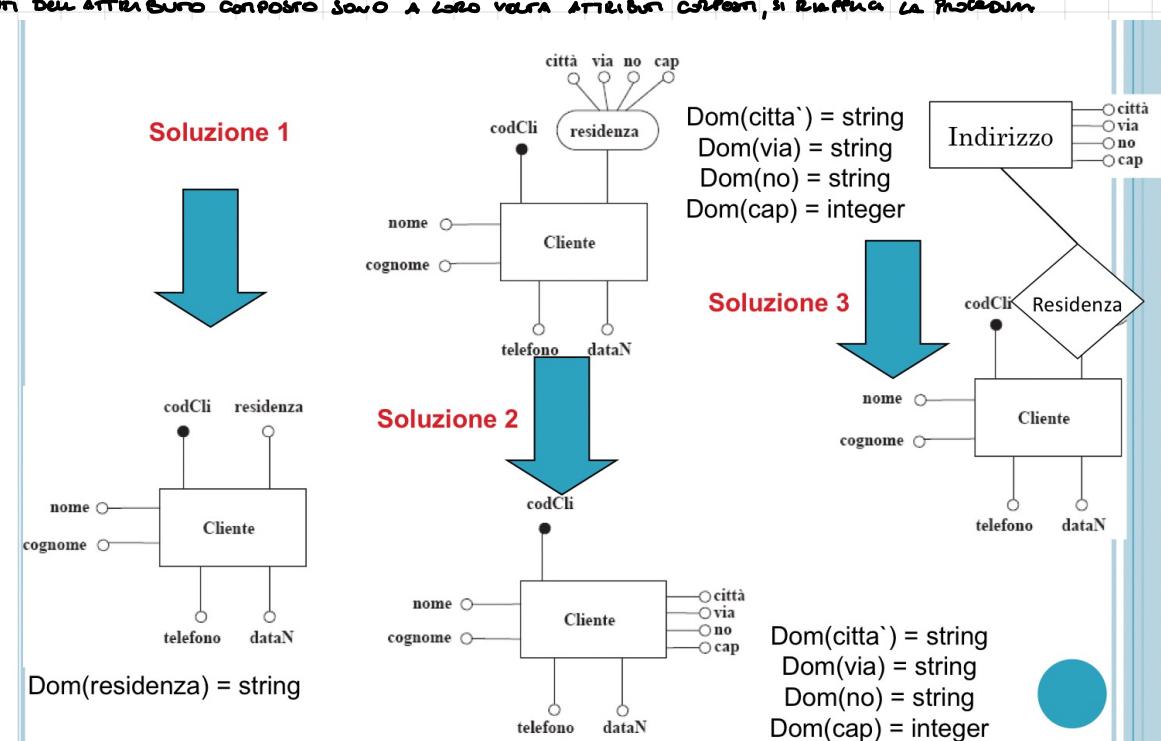
① Mettere dei sotto-attributi di A in un unico attributo composito

② Considerare tutti i sottodominanze di A come attributi di E

③ Introduire un'entità per rappresentare il tipo di A e associarla ad E

• EVENTUALI VINCOLI DI CARDINALITÀ esistenti per l'attributo composto vengono associati a ciascuno dei nuovi attributi all'associazione generata tramite la ristrutturazione

• Se le componenti dell'attributo composto sono a loro volta attributi composti, si ripete la procedura

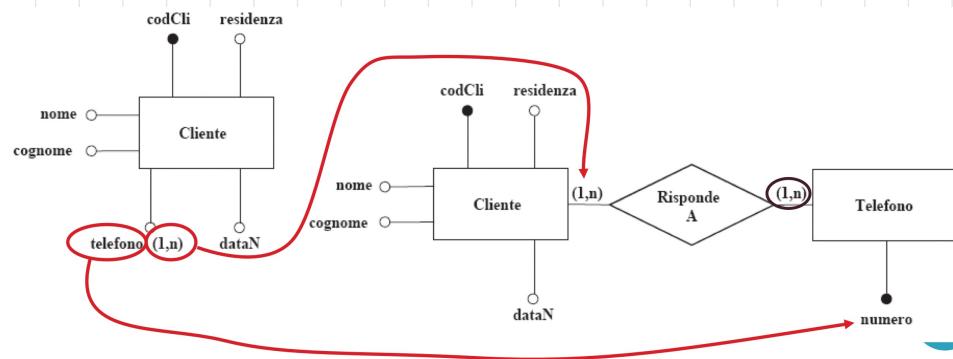


→ ELIMINAZIONE DEGLI ATTRIBUTI MULTIVALORE

- DATA UN'ENTITA' E CON UN ATTRIBUTO MULTIVALORE A
 - SI DEFINISCE UNA NUOVA ENTITA' E_A CON UN ATTRIBUTO MONOVALORE A
 - SI COLLEGANO E ed E_A TRAMITE UN'ASSOCIAZIONE R_A

• VINCOLO DI CARDINALITÀ RISPETTO ALLA NUOVA ASSOCIAZIONE

- PER E STESO VINCOLO DI CARDINALITÀ DELL'ATTRIBUTO MULTIVALORE
- PER E_A PUÒ ESSERE IN GENERALE POSTO UGUALE A $(1, n)$



→ ELIMINAZIONE DELLE GENERALIZZAZIONI

- SI ESTRAGGONO DALLA DOCUMENTAZIONE INFORMAZIONI SU TIPO DI GENERALIZZAZIONE (T.O.P, e.c.)
- SI SCEGLIE LA SOLUZIONE DI RISTRUTTURAZIONE, SULLA BASE DI:
 - TIPO DI GENERALIZZAZIONE
 - CAMPO DI LAVORO

→ ELIMINAZIONE ENITA' FIGLIE

→ ELIMINAZIONE ENITA' PADRE

→ SOSTITUZIONE DELLA GENERALIZZAZIONE CON ASSOCIAZIONI

→ ELIMINAZIONE ENITA' FIGLIE

- ATTRIBUTO ENITA' FIGLIE IN E CORRE OZIONARE

- SI AGGIUNGE AD E UN ATTRIBUTO TIPO CHE SPECIFICA DA QUALE ENITA' FIGLIA Ei POSSERRE L'ISTANZA DELL'ENITA' PADRE NELLO SCHEMA RISTRUTTURATO

→ NEL CASO DI GENERALIZZAZIONI TOTALI, NON PUÒESSERE NULLO

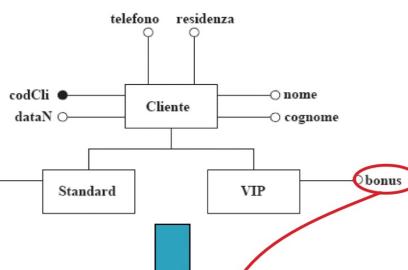
→ NEL CASO DI GENERALIZZAZIONI PARZIALI, UN VALORE NULLO VOLTE IN Istanza DI E CORRISPONDENTI A NIENTE DI NESSUNA Ei;

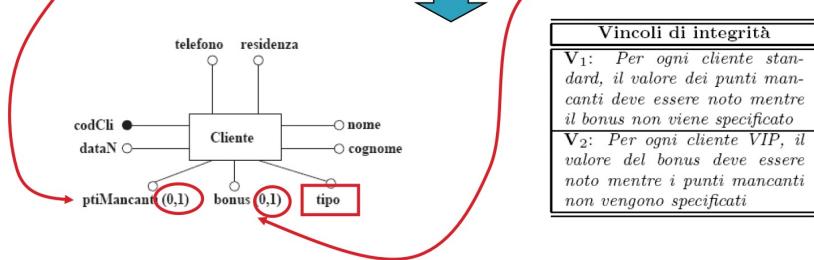
→ NEL CASO DI GENERALIZZAZIONI CARDINALI, Ei È UNO SET DI VALORI ASSOCIATI AL TIPO

• BISOGNA AGGIUNGERE UN VINCOLO DI INTEGRITÀ PER GARANTIRE CHE SE TIPO = Ei, GLI ATTRIBUTI OBBLIGATORI DI Ei SONO NULLI

• BISOGNA AGGIUNGERE UN VINCOLO DI INTEGRITÀ PER GARANTIRE CHE SE UN ATTRIBUTO DI Ei È NON NULLO, ALLORA TIPO = Ei (SE LA GENERALIZZAZIONE È CARDINALE, Ei È UNO SET DI VALORI ASSOCIATI AL TIPO)

Generalizzazione totale ed esclusiva





21

→ ELIMINAZIONE ENTITA' FIGLIE - ASSOCIAZIONI

- LA PARTECIPAZIONE DI UN'ENTITA' FIGLIA AD UN'ASSOCIAZIONE DIVENTA LA PARTECIPAZIONE OZIOMALE DELL'ENTITA' PADRE ALLA STESSA ASSOCIAZIONE

- PER OGNI ASSOCIAZIONE, BISOGNA AGGIUNGERE UN VINCULO DI INTEGRITA' CHE INDICHI COME TIPI DI Istanze dell'entita' Padre POSSANO ESSERE CONVOLTI NELL'ASSOCIAZIONE

→ ELIMINAZIONE ENTITA' PADRE (APPLICABILE SOLO NEL CASO DI GENERALIZZAZIONE TOTALE)

- IDENTIFICATO DEGLI ATTRIBUTI DI E IN Ogni UNA DELL'ENTITA' FIGLIE

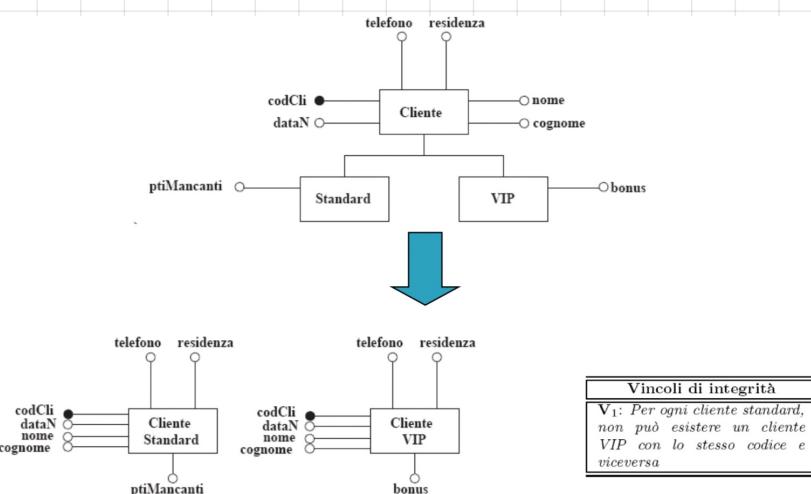
- Ogni ASSOCIAZIONE A CUI PARTECIPAVA E viene SOSTITUITA CON NOVE ASSOCIAZIONI, UNA PER OGNI Ei

VINCOLI
DI
INTEGRITA'

- SE GENERALIZZAZIONE ESCLUSIVA, VINCULO PER INDICARE CHE NEGLI ATTRIBUTI RISTRITTIVI NON POSSONO ESISTERE ISTANZE DI DUE ENTITA' FIGLIE DISTINTE AVANTI LO STESSO VALORE PER GLI IDENTIFICATORI (EXCLAMATION DELL'ENTITA' PADRE)

- IL VINCULO DI CARDINALITA' DI Ogni UNA DELL'ENTITA' FIGLIE RISPETTO ALLA NUOVA ASSOCIAZIONE CONDIVISA CON IL VINCULO DI CARDINALITA' DELL'ENTITA' PADRE RISPETTO ALL'ASSOCIAZIONE SOSTITUITA

- I VINCULI DI CARDINALITA' DELL'ENTITA' FIGLIE DIVERTERANNO INVECE OPERATORI



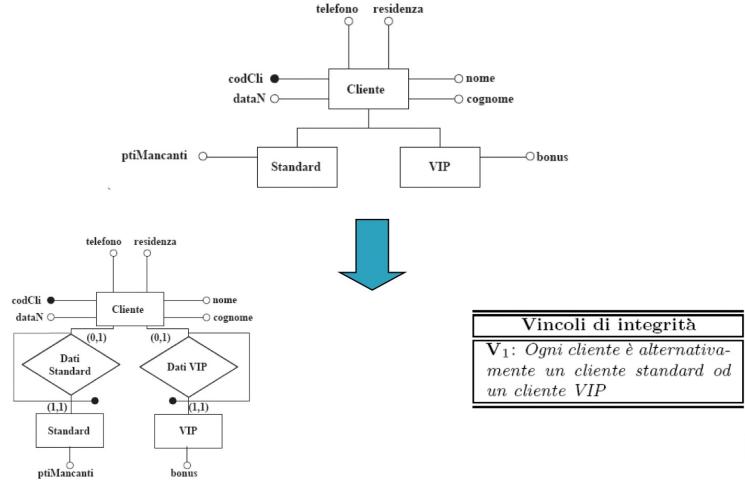
→ SOSTITUZIONE DELLA GENERALIZZAZIONE CON ASSOCIAZIONI

- LA GENERALIZZAZIONE viene SOSTITUITA DA N ASSOCIAZIONI Ri, UNO A UNO, CHE COLLEGANO Ei CON Ei
- CIASCUN Ei E' IDENTIFICATO ESCLUSIVAMENTE DA Ei E PARTECIPA OBBLIGATORIAMENTE A Ri
- LA PARTECIPAZIONE DI Ei A Ogni Ri E' OZIOMALE

VINCOLI
DI
INTEGRITA'

- GENERALIZZAZIONE ESCLUSIVA: UN'ISTANZA DI E NON PUO' PARTECIPARE CONTEMPORANEAMENTE A DUE O PIU' ASSOCIAZIONI Ri

- GENERALIZZAZIONE TOTALE: OGNI Istanza di E deve PARTECIPARE OBBLIGATORIAMENTE AD ALmeno UN'ASSOCIAZIONE Ri



2

OSSERVAZIONI

- ELIMINAZIONE ENTITA' FIGLIE
 - SPERCO DI MIGRARE PER LA PRESENZA DI VALORI NULL
 - CONVENIENTE SOLO NEL CASO IN CUI LE OPERAZIONI NON FANNO DISTINZIONE TRA LE VARIABILI SOTTO-ENTITA'
- ELIMINAZIONE ENTITA' PADRE
 - SOLO PER GENERALIZZAZIONE TOTALE
 - CONVENIENTE SOVRINTENDUTO NEL CASO IN CUI ESISTANO OPERAZIONI CHE SI RIFERISCONO A ISTRUZIONI DI UNA SPECIFICA ENTITA' FIGLIA
- SOSTITUZIONE CON ASSOCIAZIONI
 - PREFERIBILE ALLA SOLUZIONE DI SOSTituIRE LE ENTITA' FIGLIE PER QUANTO RIGUARDA LA QUANTITA' DI PERSONA UTILIZZATA
 - CONVENIENTE QUANDO CI SONO DIVERSE OPERAZIONI CHE DISCRIMINANO TRA ENTITA' PADRE E ENTITA' FIGLIE
- IN ALCUNE SITUAZIONI, PUO' ESSERE CONVENIENTE ADOTTARE SOLUZIONI IBIDE: GENERALIZZAZIONE DI UN SOTTO-UNIONE DELLE ENTITA' FIGLIE, MANTENENDO LE ALTRE PIENO SCHERZO
- GENERALIZZAZIONE A PIU' LIVELLI:
 - SI ATTENGONO LE STRATEGIE PROPOSTE PARTENDO DALLE FOGGIE DELLA GENERALIZZAZIONE SUCCESSIVA
 - LO SCHEMA RISULTANTE DIPENDERÀ DAL TIPO DELLA RESTITUTTURAZIONE APPLICATA A OGNI LIVELLO

TRADUZIONE

TRAUDUZIONE LO SCHEMA RESTITUITO IN UNO SCHEMA RELAZIONALE EQUIVALENTE

- TRADUZIONE ENTITA'
- TRADUZIONE ASSOCIAZIONI
- TRADUZIONE VINCOLI DI INTEGRITÀ
- OTTIMIZZAZIONI FINALI

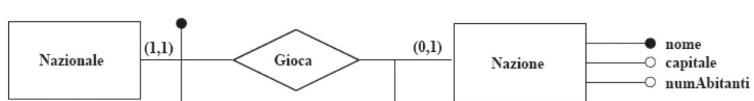
ENTITA' → RELAZIONE

ASSOCIAZIONE → RELAZIONE O CLAVIA ESTERNA

TRADUZIONE ENTITA':



Videoteca(nome, città)
Video(colloc, nome Videoteca, città Videoteca, tipo)

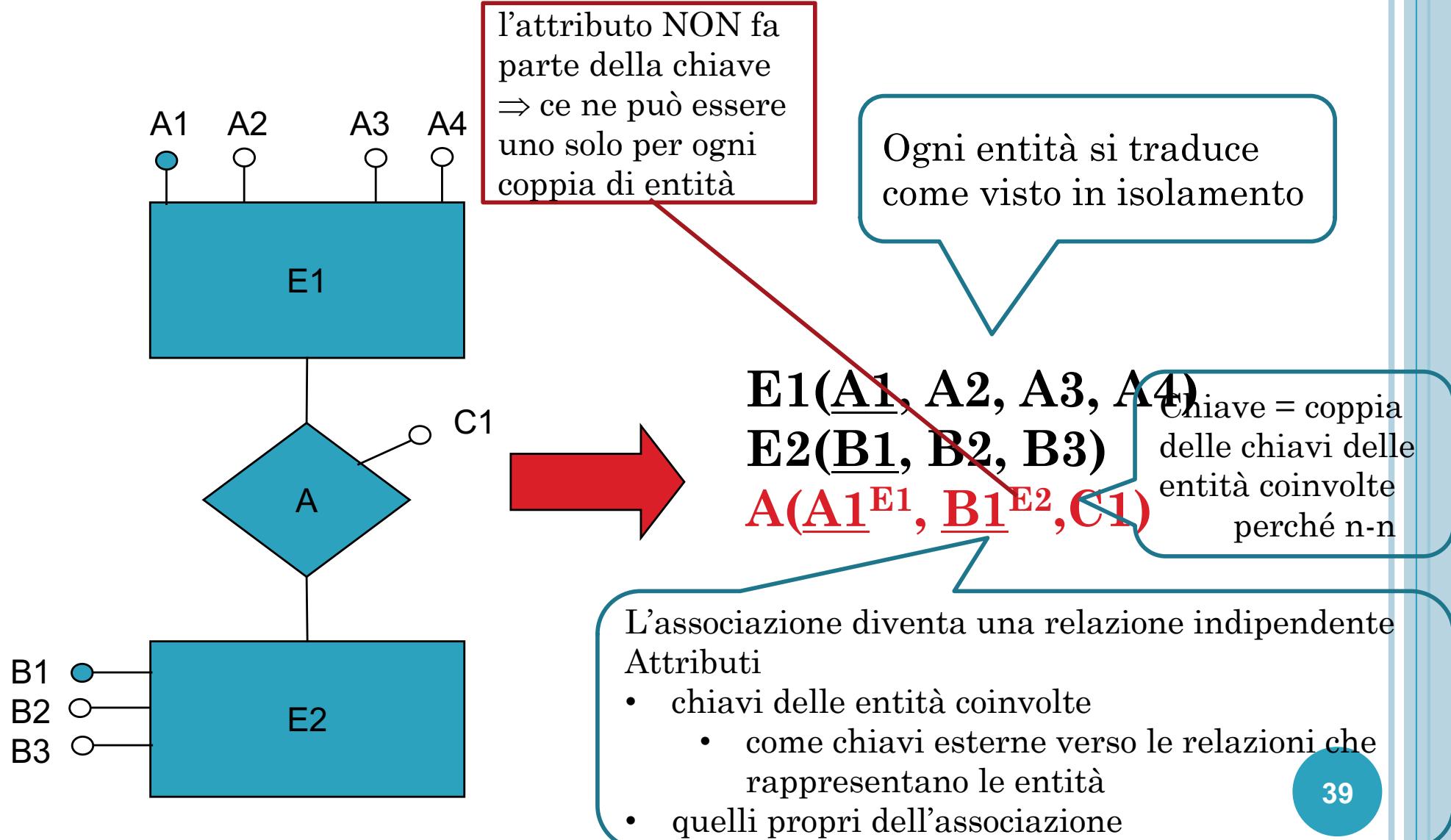


Nazione(nome, capitale, numAbitanti)
Nazionale(nome^{Nazione})

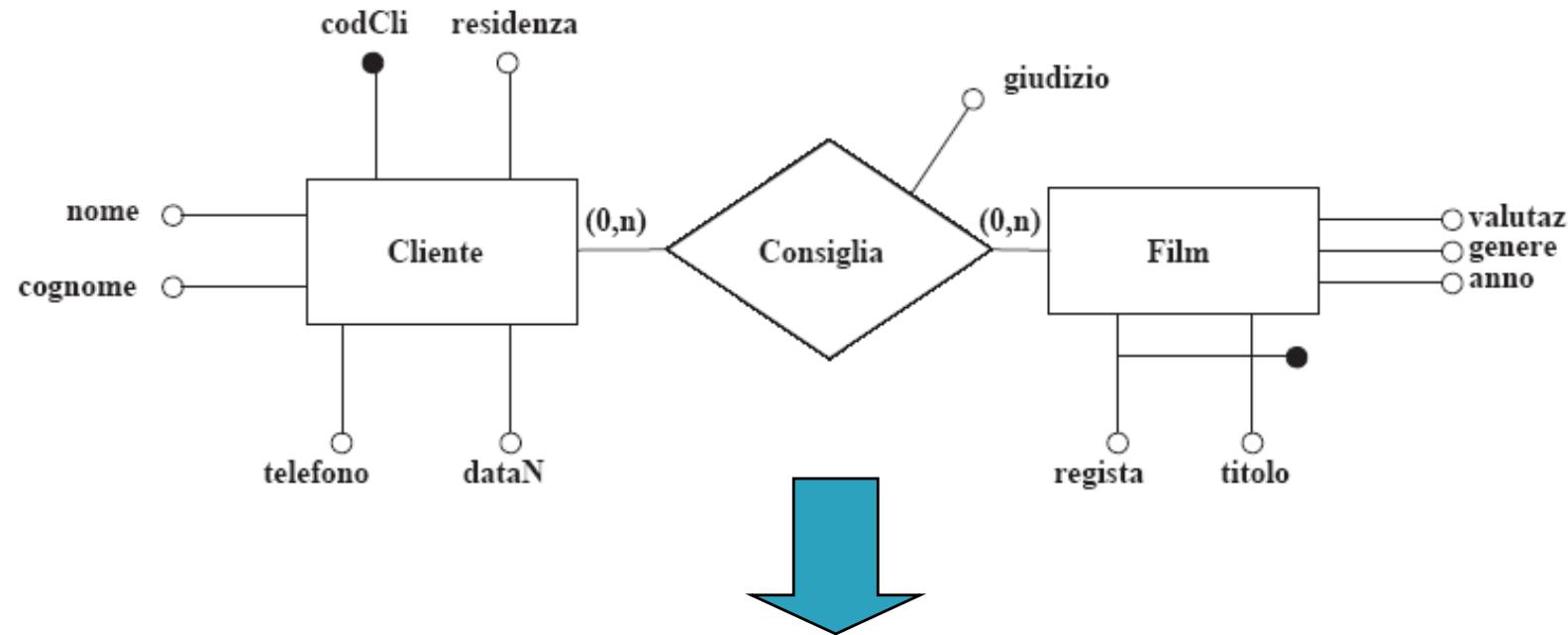
→ Scelta chiave primaria

- CIASCONA RELAZIONE POTREBBE ESSERE CONTENUTA DA PIÙ DI UNA CHIAVE
 - È NECESSARIO SELEZIONARE UNA DI QUESTE COME CHIAVE PRIMARIA
 - LA SCELTA DIPENDE DA CRITERI DI EFFICIENZA
- È PREFERIBILE DEFINIRE UNA CHIAVE CIECA SULLA BASE DI UNA CHIAVE PRIMARIA
- GLI IDENTIFICATORI CHE CONTENGONO ATTRIBUTI OPTIONALI NON POSSANO ESSERE SELEZIONATI COME CHIAVE PRIMARIA
- PREFERIBILI IDENTIFICATORI COMPOSTI DA POCCHI ATTRIBUTI O CHE SI STIMA VENGANO UTILIZZATI DA TANTE OPERAZIONI PER ACCEDERL' ALLA ENTRÀ
- SE nessuna chiave candidata SODDISFA i REQUISITI PRECEDENTI, SI AGGIUNGE UN ULTERIORE ATTRIBUTO "CODICE" COME CHIAVE PRIMARIA E GLI SI ATTRIBUISCONO I VALORI DEFATI GENERICI

TRADUZIONE ASSOCIAZIONE BINARIA MOLTI A MOLTI



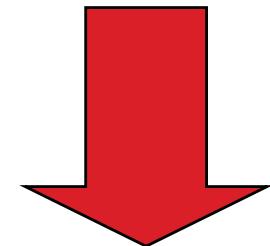
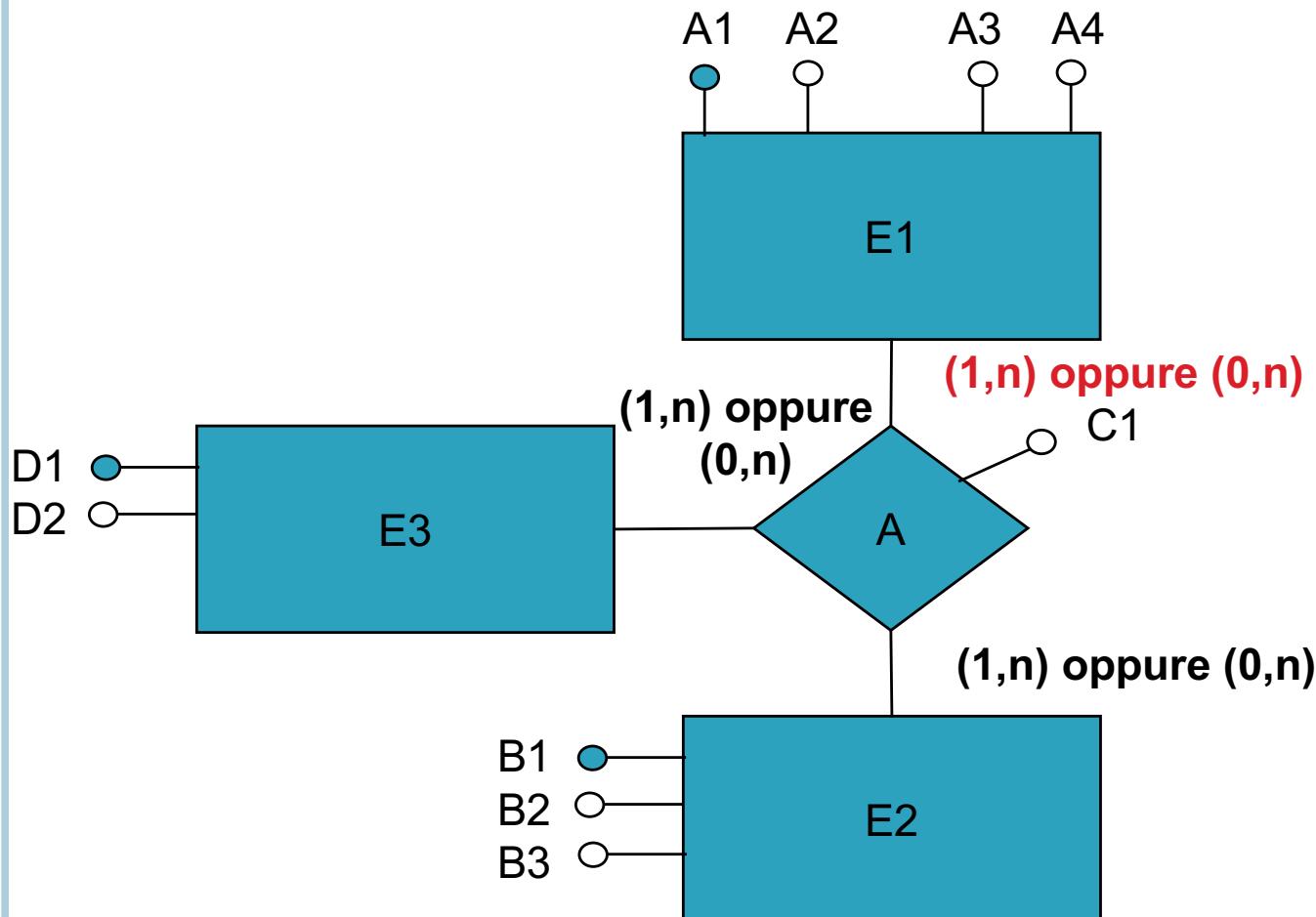
TRADUZIONE ASSOCIAZIONE BINARIA MOLTI A MOLTI ESEMPIO



Cliente(codCli, nome, cognome, telefono, dataN, residenza)
Film(titolo, regista, anno, genere, valutaz)
Consiglia(codCli^{Cliente}, titolo^{Film}, regista^{Film}, giudizio)

TRADUZIONE ASSOCIAZIONE N-ARIA MOLTI A MOLTI

- In modo analogo ad associazioni binarie
- Caso molto comune



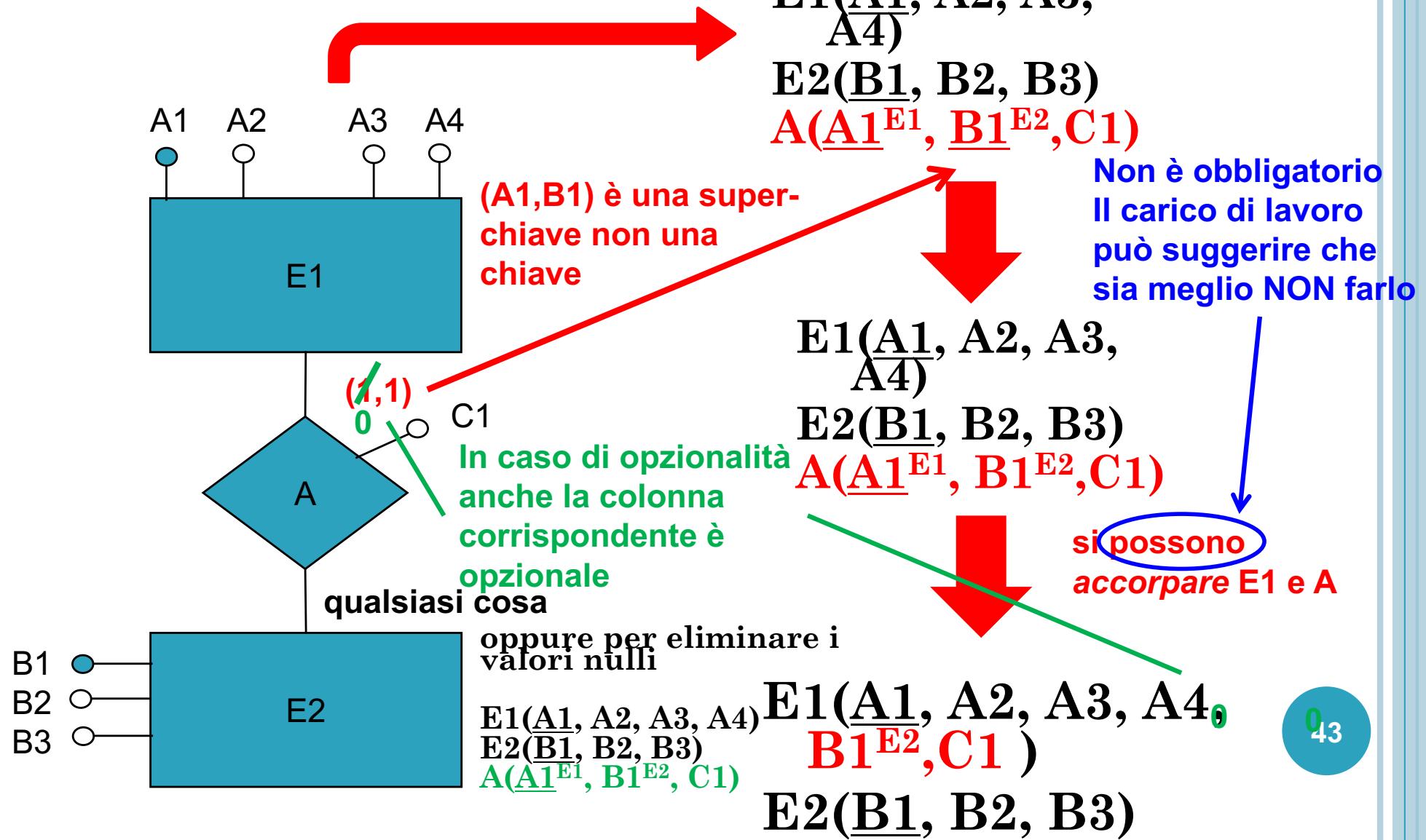
E1(A1, A2, A3, A4)
E2(B1, B2, B3)
E3(D1, D2)
A(A1^{E1}, B1^{E2}, D1^{E3}, C1)

TRADUZIONE GENERICA: PROBLEMATICHE

- Ogni navigazione di un'associazione richiede due join
 - computazionalmente pesante
- In alcuni casi (A1, B1, D1) è una super-chiave per A
 - La determinazione della chiave può avvenire dall'analisi di particolari vincoli di integrità
 - es. se la molteplicità massima di partecipazione di una delle entità è 1
- In casi di relazioni 1-1 l'informazione memorizzata è ridondante
 - può lo stesso essere utile mantenere questa traduzione per le stesse ragioni per cui si fa partizionamento
- In casi di relazioni 1-1 non si cattura il vincolo di unicità

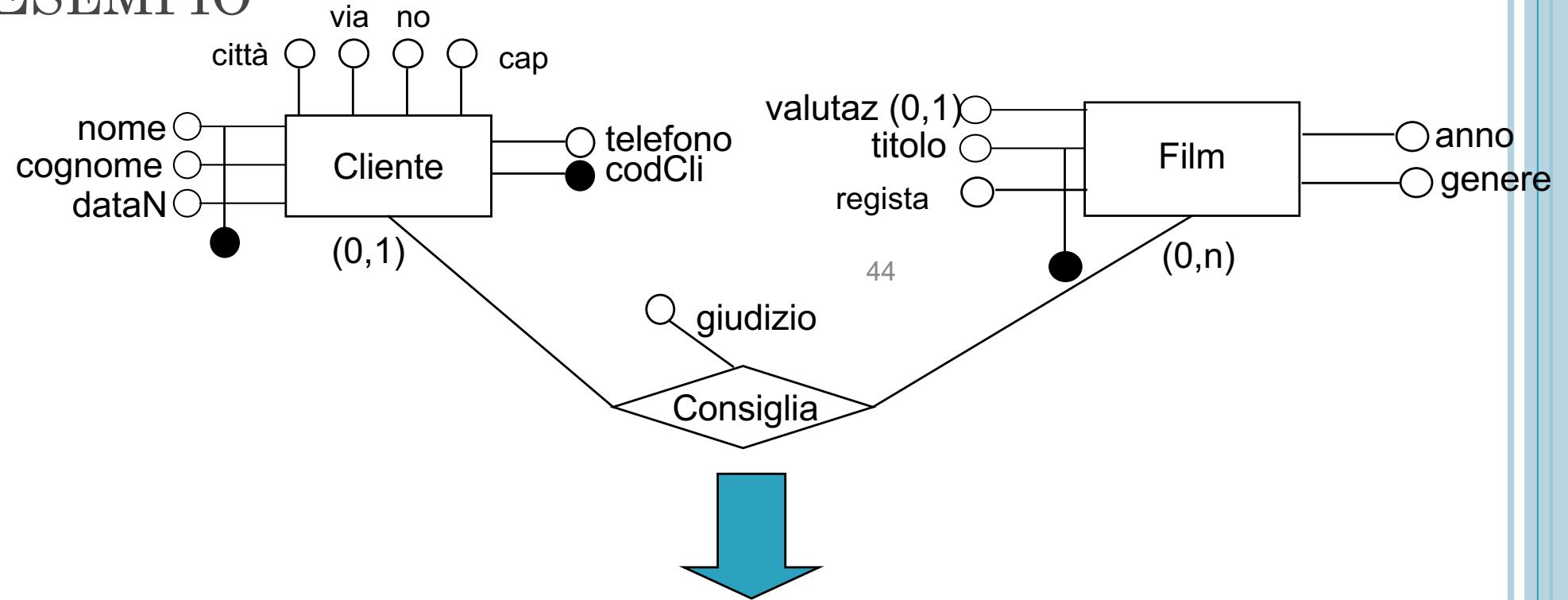
TRADUZIONE ASSOCIAZIONE BINARIA

UNO A *



TRADUZIONE ASSOCIAZIONE BINARIA UNO A *

ESEMPIO

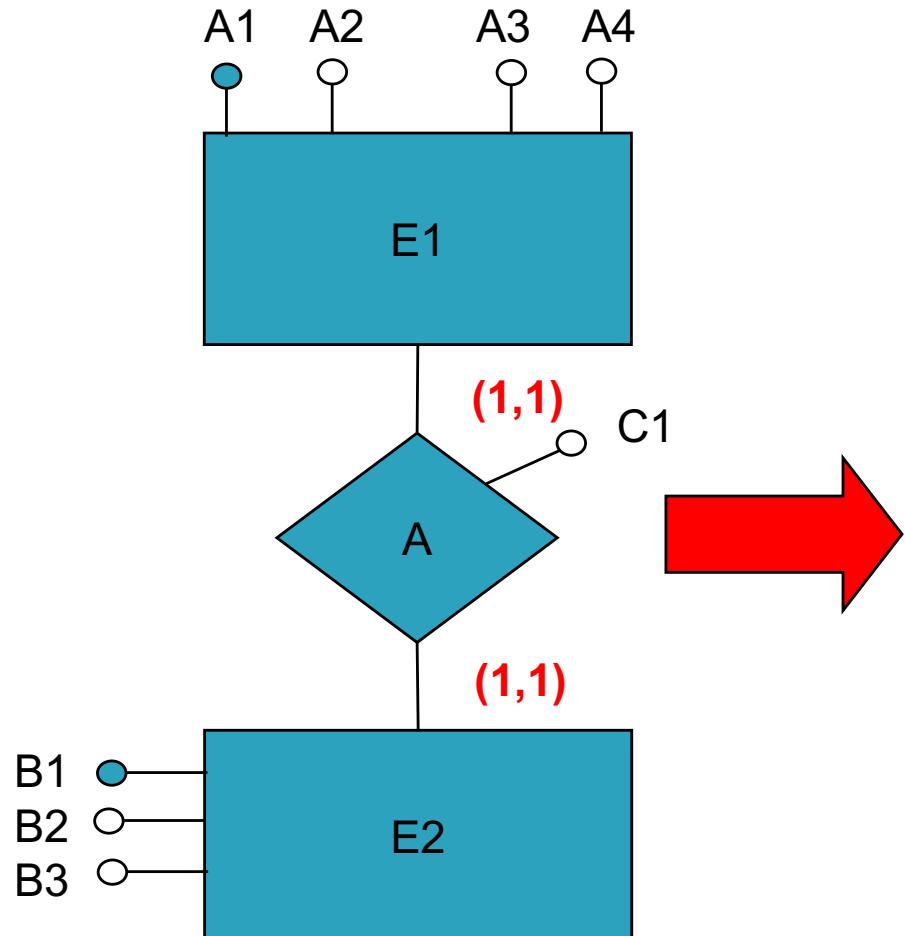


Film(titolo,regista,anno,genere,valutaz)

Cliente(codCli,nome,cognome,telefono,dataN,città,via,no,cap,
titolo₀^{Film},regista₀^{Film},giudizio₀)

TRADUZIONE ASSOCIAZIONE BINARIA UNO A UNO

Se entrambe partecipano univocamente si può scegliere con quale accorpate l'associazione sulla base del carico di lavoro

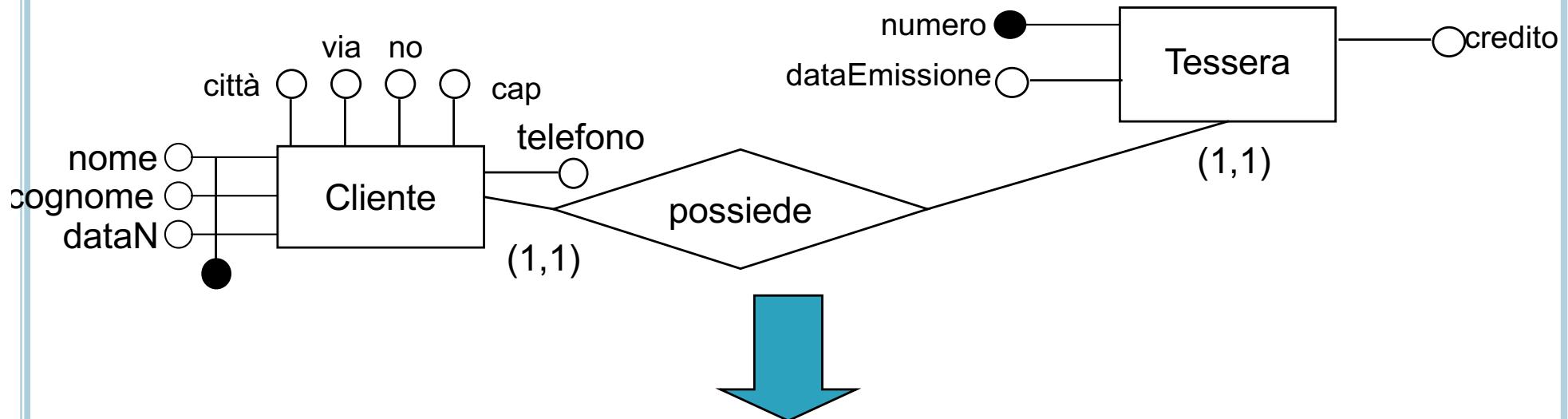


$E1(\underline{A1}, A2, A3, A4, \underline{B1}^{E2}, C1)$
 $E2(\underline{B1}, B2, B3)$

Oppure

$E1(\underline{A1}, A2, A3, A4)$
 $E2(\underline{B1}, B2, B3, \underline{A1}^{E1}, C1)$

TRADUZIONE ASSOCIAZIONE BINARIA UNO A UNO ESEMPIO



Tessera(numero,dataEmissione,credito)

Cliente(nome,cognome,telefono,dataN,città, via, no, cap,numero^{Tessera})

Tessera(numero,dataEmissione,credito, nome^{Cliente},cognome^{Cliente},dataN^{Cliente})

Cliente(nome,cognome,telefono,dataN,città, via, no, cap)

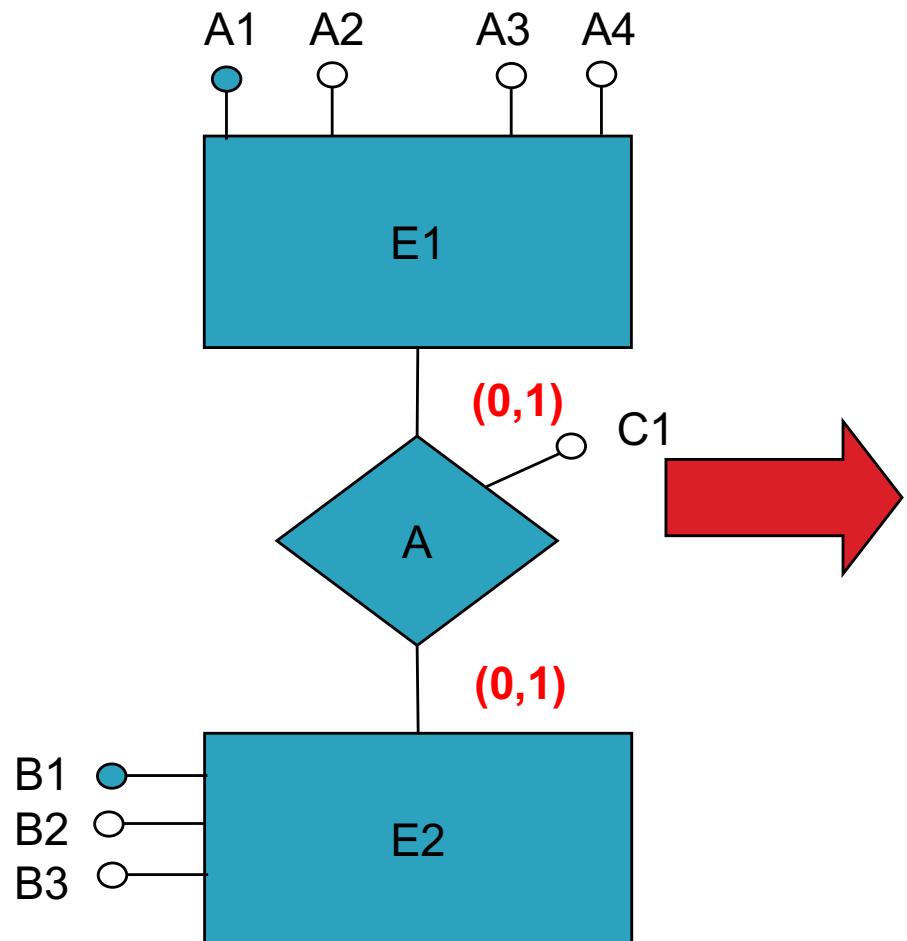
Tessera(numero,dataEmissione,credito)

Cliente(nome,cognome,telefono,dataN,città, via, no, cap)

Possiede(nome^{Cliente},cognome^{Cliente},dataN^{Cliente},numero^{Tessera}) oppure

Possiede(nome^{Cliente},cognome^{Cliente},dataN^{Cliente},numero^{Tessera})

TRADUZIONE ASSOCIAZIONE BINARIA UNO A UNO PARTECIPAZIONE OPZIONALE DI ENTRAMBE LE ENTITÀ



$E1(\underline{A1}, A2, A3, A4, \textcolor{red}{B1_o^{E2}}, \textcolor{red}{C1_o})$
 $E2(\underline{B1}, B2, B3)$

oppure

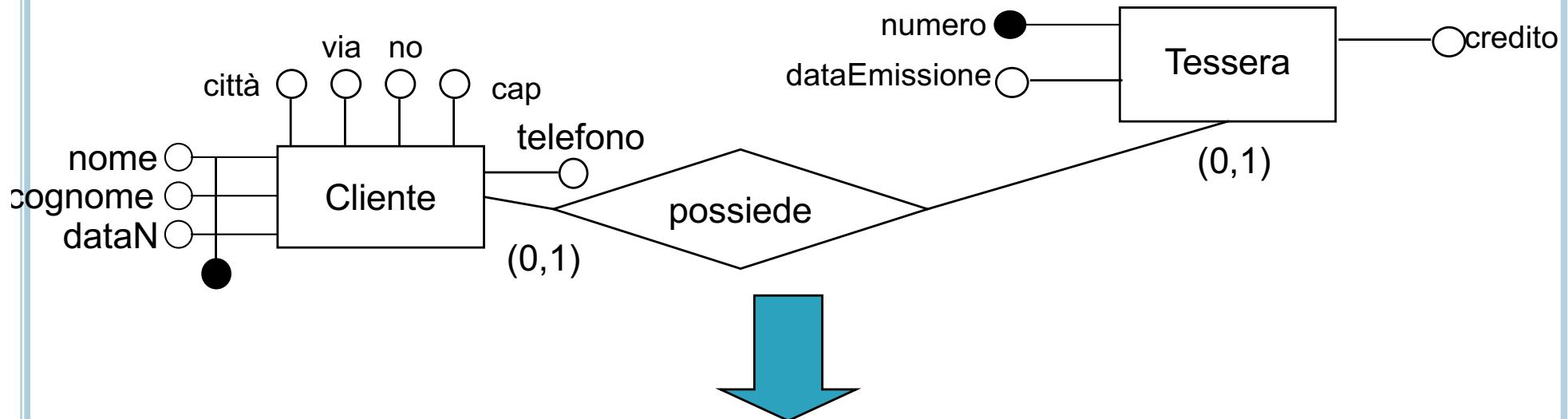
$E1(\underline{A1}, A2, A3, A4)$
 $E2(\underline{B1}, B2, B3, \textcolor{red}{A1_o^{E1}}, \textcolor{red}{C1_o})$

alternativa per eliminare i
valori nulli

$E1(\underline{A1}, A2, A3, A4)$
 $E2(\underline{B1}, B2, B3)$
 $\textcolor{red}{A(A1^{E1}, B1^{E2}, C1)} \text{ oppure}$
 $\textcolor{red}{A(A1^{E1}, \underline{B1^{E2}}, C1)}$

= si ritorna alla traduzione
standard

TRADUZIONE ASSOCIAZIONE BINARIA UNO A UNO ESEMPIO



Tessera(numero,dataEmissione,credito)

Cliente(nome,cognome,telefono,dataN,città, **via, **no**,**cap**,numero₀^{Tessera})**

Tessera(numero,dataEmissione,credito, **nome₀^{Cliente},**cognome**₀^{Cliente},**dataN**₀^{Cliente})**

Cliente(nome,cognome,telefono,dataN,città, **via, **no**,**cap**)**

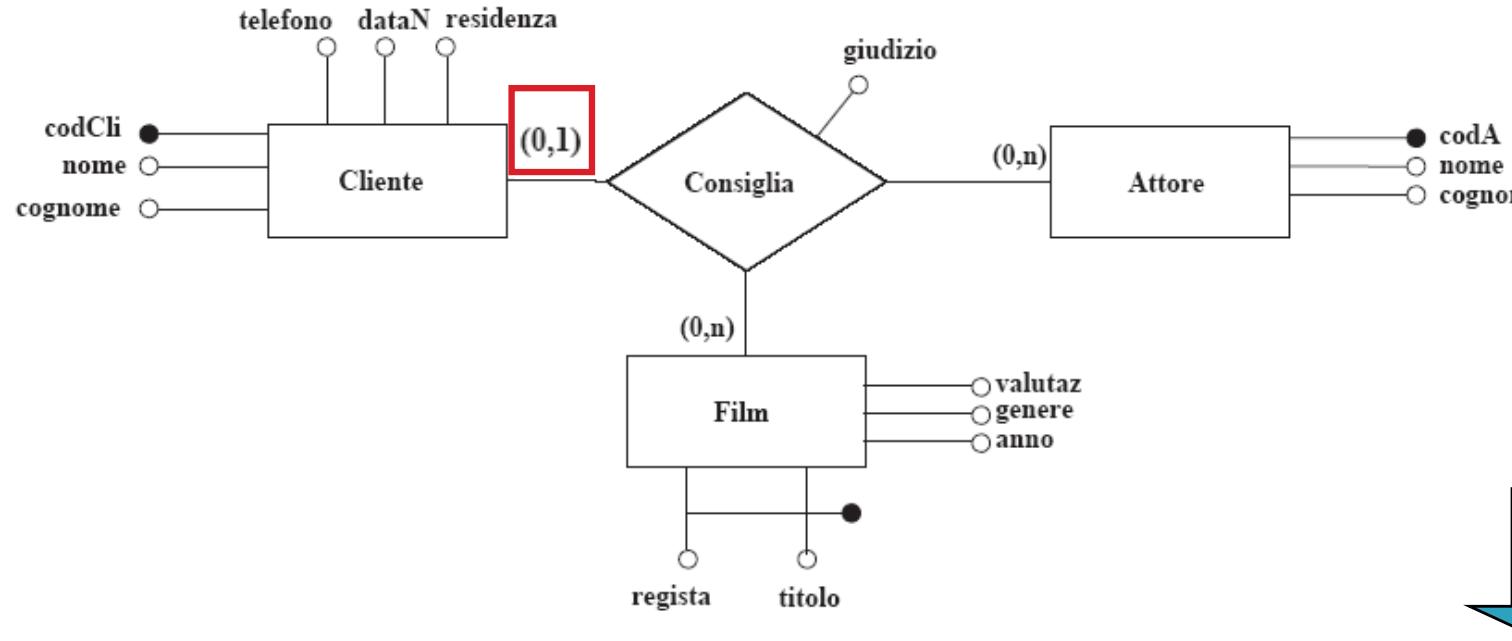
Tessera(numero,dataEmissione,credito)

Cliente(nome,cognome,telefono,dataN,città, **via, **no**,**cap**)**

Possiede(nome**₀^{Cliente},**cognome**₀^{Cliente},**dataN**₀^{Cliente},numero₀^{Tessera}) oppure**

Possiede(nome**₀^{Cliente},**cognome**₀^{Cliente},**dataN**₀^{Cliente},numero₀^{Tessera})**

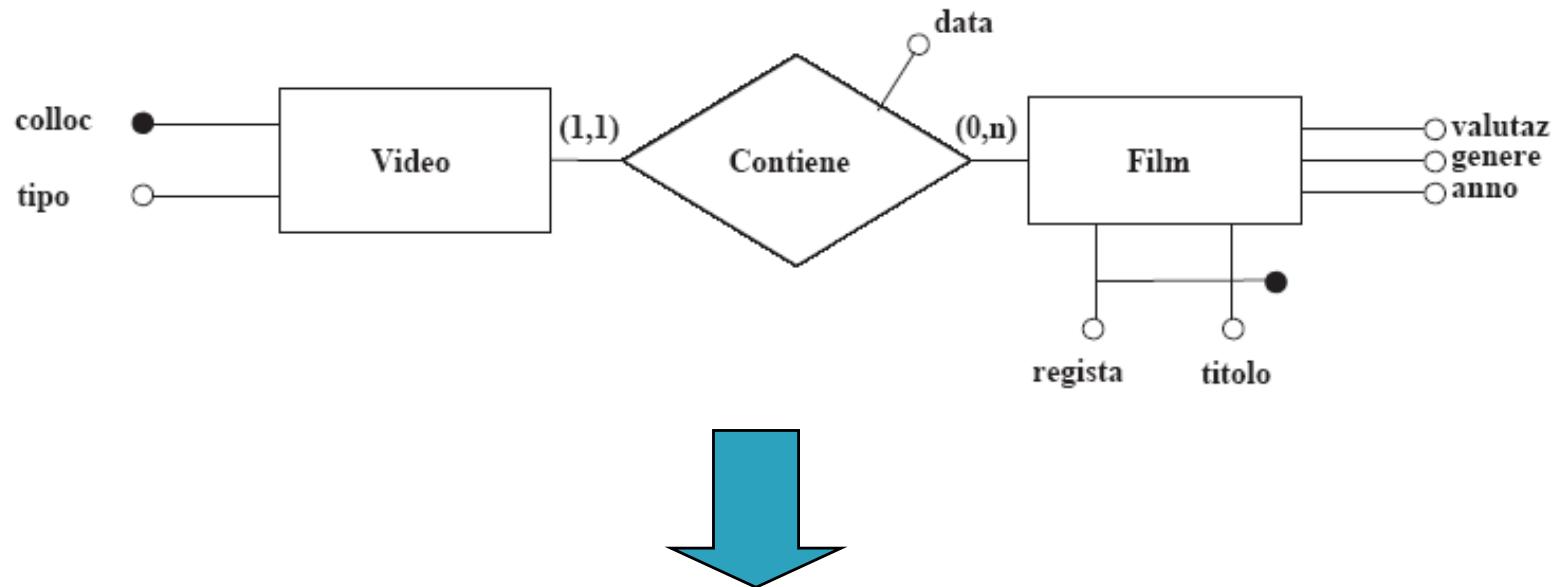
TRADUZIONE ASSOCIAZIONE N-ARIA 1 A MOLTI ESEMPIO



Cliente(codCli,nome,cognome,telefono,dataN,residenza,
titolo_o^{Film},regista_o^{Film},codA_o^{Attore},giudizio_o)
Film(titolo,regista,anno,genere,valutaz)
Attore(codA,nome,cognome)

Come si può ridurre la presenza di valori nulli?

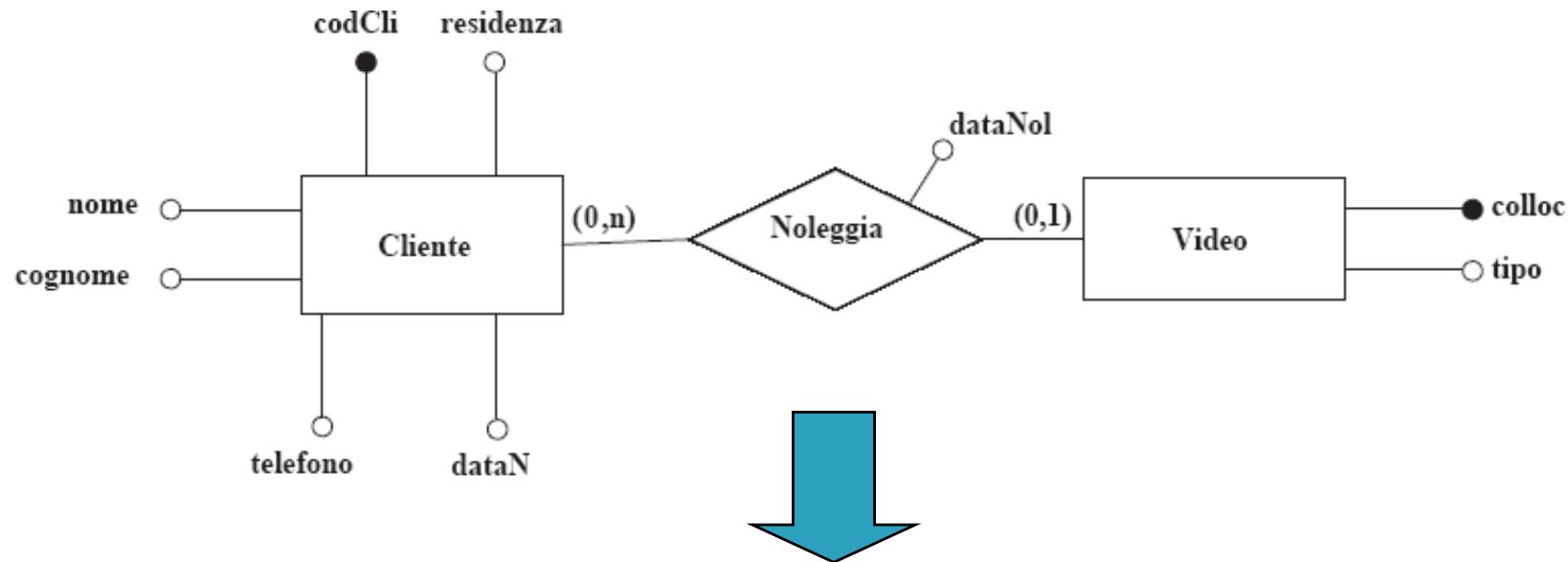
TRADUZIONE ASSOCIAZIONE: PALESTRA ESEMPIO 1



Film(titolo,regista,anno,genere,valutaz)

Video(colloc,tipo,titolo^{Film},regista^{Film},data)

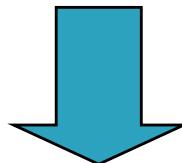
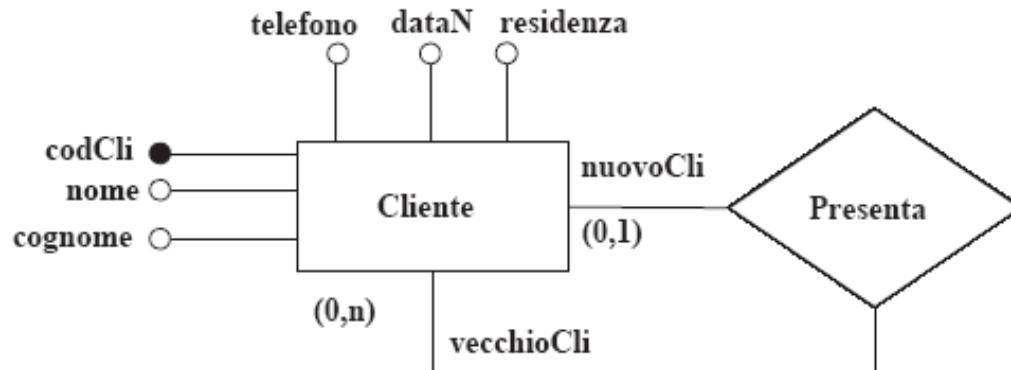
TRADUZIONE ASSOCIAZIONE: PALESTRA ESEMPIO 2



Video(colloc,tipo,**codCli_o**^{Cliente},**dataNol_o**)
Cliente(**codCli**,nome,cognome,telefono,dataN,residenza)

Video(colloc,tipo)
Cliente(**codCli**,nome,cognome,telefono,dataN,residenza)
Noleggia(colloc^{Video},**codCli**^{Cliente},**dataNol**)

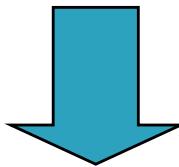
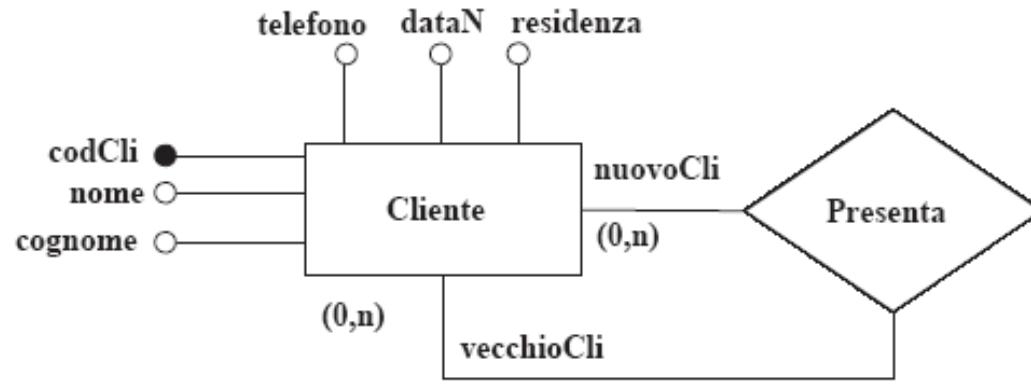
TRADUZIONE ASSOCIAZIONE: PALESTRA ESEMPIO 3



Cliente(codCli, nome, cognome, telefono, dataN, residenza, **vecchioCli**^{Cliente})
oppure

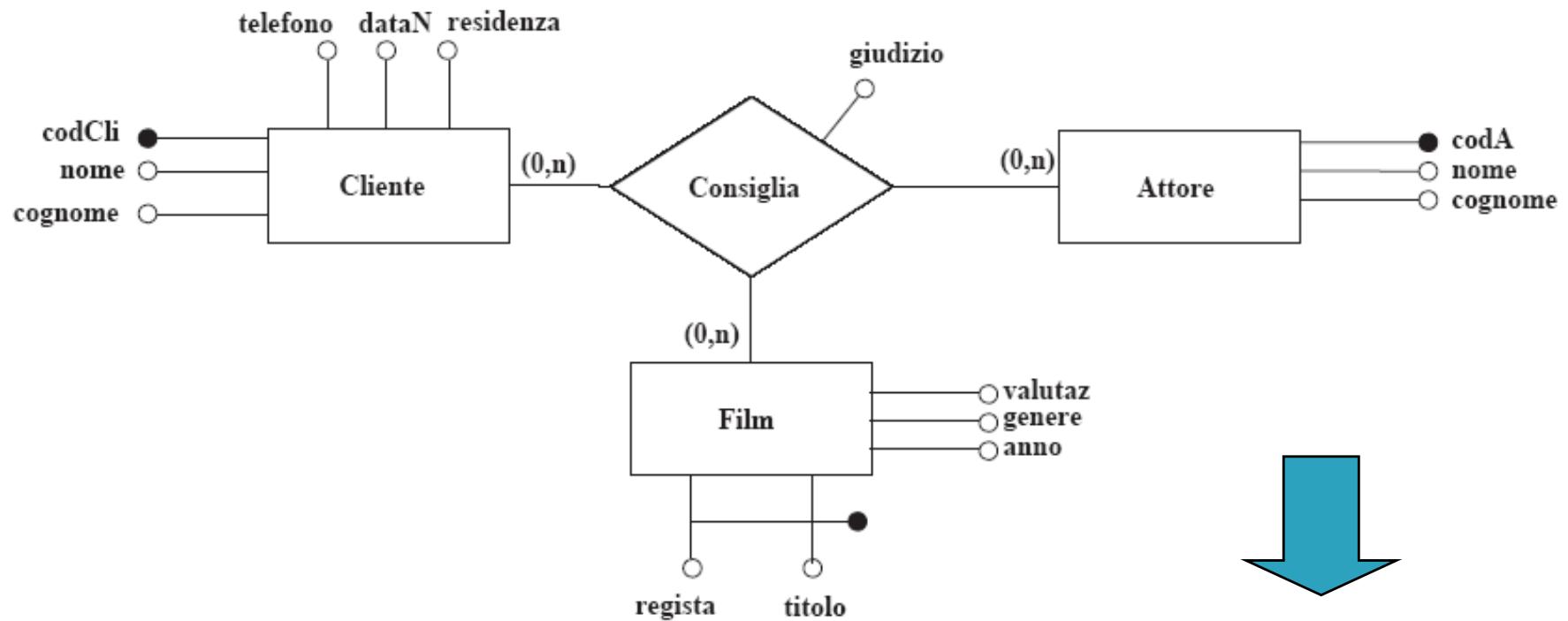
Cliente(codCli, nome, cognome, telefono, dataN, residenza)
Presenta(nuovoCli^{Cliente}, **vecchioCli**^{Cliente})

TRADUZIONE ASSOCIAZIONE: PALESTRA ESEMPIO 4



Cliente(codCli, nome, cognome, telefono, dataN, residenza)
Presenta(nuovoCli^{Cliente}, vecchioCli^{Cliente})

TRADUZIONE ASSOCIAZIONE: PALESTRA ESEMPIO 5



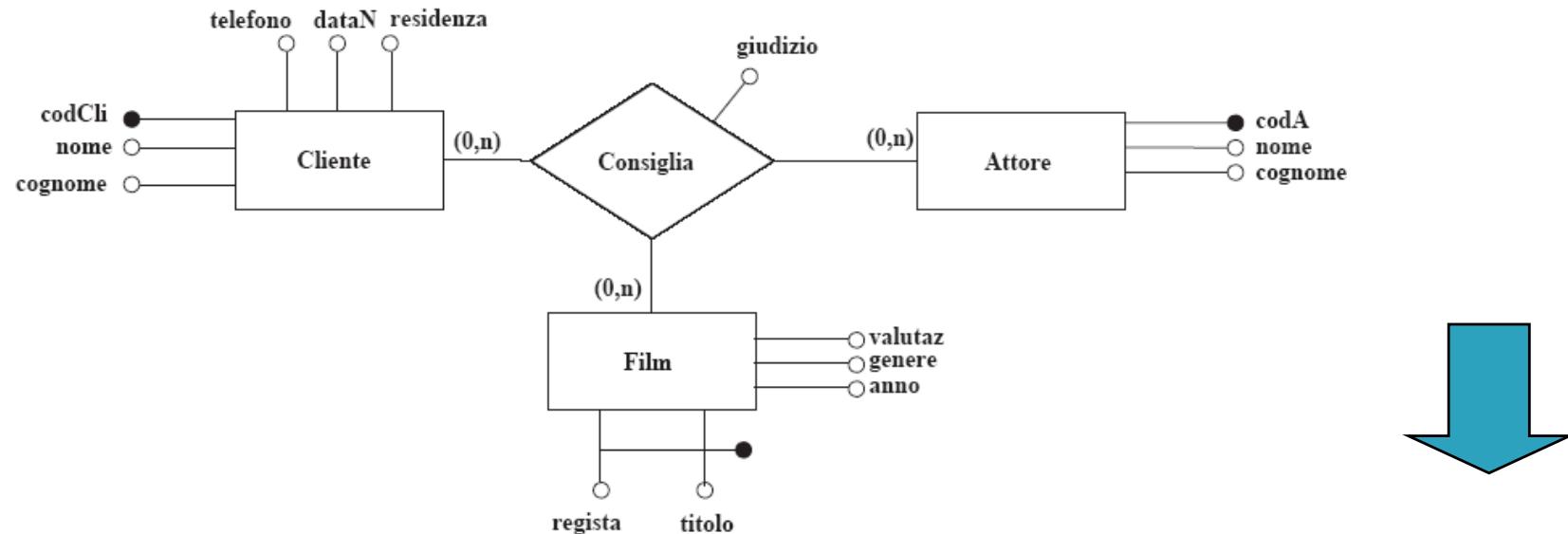
Cliente(codCli,nome,cognome,telefono,dataN,residenza)

Film(titolo,regista,anno,genere,valutaz)

Attore(codA,nome,cognome)

Consiglia(codCli^{Cliente},titolo^{Film},regista^{Film},codA^{Attore},giudizio)

TRADUZIONE ASSOCIAZIONE: PALESTRA ESEMPIO 6



Vincolo di integrità: un cliente può consigliare attori in un numero arbitrario di film, ma al più un film per ogni attore

Cliente(codCli, nome, cognome, telefono, dataN, residenza)

Film(titolo, regista, anno, genere, valutaz)

Attore(codA, nome, cognome)

Consiglia(codCli^{Cliente}, titolo^{Film}, regista^{Film}, codA^{Attore}, giudizio)

ALGEBRA RELAZIONALE

Le operazioni usano solo relazioni come argomento e producono relazioni

5 OPERAZIONI DI BASE:

- PROIEZIONE
- SELEZIONE
- PRODOTTO CARTESIANO
- UNIONE
- DIFFERENZA

+ OPERAZIONI DERIVATE (JOIN)

RIDENOMINAZIONE

$P_{\text{DATAHOL}, \text{DATAREST} \leftarrow \text{DATAHOL}, \text{DATAFINE}} (\text{Noleggio})$

DATAHOL viene ridenominato in DATAFINE
DATAHOL viene ridenominato in DATAW

SQL: `SELECT colloc, dataHol AS dataW, codChi, dataRest AS dataFine
FROM Noleggio`

PROIEZIONE

$\Pi_{A_1 \dots A_n}(R)$ tiene solo le colonne A_i

ELIMINAZIONE DEI DUPLICATI

SQL: `SELECT DISTINCT FROM R`

SELEZIONE

$G_F(R)$ tiene solo le tuple di R su cui vale F

SQL: `SELECT DISTINCT * FROM R
WHERE B = b`

(esempio) $G_{B=b}(R)$

NON INTRODUCCE DUPLICATI, MA PUÒ PROPAGARE

PRODOTTO CARTESIANO

$R = R_1 \times R_2$

A_1	A_2	A_3
a	b	3
d	a	5
c	b	27

R_1
 R_2
In SQL come si esprimono?

B_1	B_2
1	g
6	k

B_1	B_2	A_1	A_2	A_3
1	g	a	b	3
6	k	a	b	3
1	g	d	a	5
6	k	d	a	5
1	g	c	b	27
6	k	c	b	27

$R_2 \times R_1$
SELECT
DISTINCT *
FROM R2, R1

$R_1 \times R_2$

A_1	A_2	A_3	B_1	B_2
a	b	3	1	g
a	b	3	6	k
d	a	5	1	g
d	a	5	6	k
c	b	27	1	g
c	b	27	6	k

SELECT
DISTINCT *
FROM R1, R2

• UNIONE

$$R = R_1 \cup R_2$$

A ₁	A ₂	A ₃
a	b	3
d	a	5
c	b	27

R₁

A ₁	A ₂	A ₃
a	b	3
j	a	5

R₂

A ₁	A ₂	A ₃
j	a	5
a	b	3
d	a	5
c	b	27

R₂ \cup R₁

perché le relazioni sono insiemi anche se le rappresentiamo come tabelle le righe non hanno un ordine

R₁ \cup R₂

A ₁	A ₂	A ₃
a	b	3
d	a	5
c	b	27
j	a	5

In SQL come si esprimono?
Con UNION

23

SONO UGUALI

• DIFFERENZA

le tuple che compaiono in entrambi vengono eliminare

A ₁	A ₂	A ₃
a	b	3
d	a	5
c	b	27

R₁

A ₁	A ₂	A ₃
j	a	5

R₂

A ₁	A ₂	A ₃
d	a	5
c	b	27

In SQL come si esprimono?
Con EXCEPT

25

A	B	C
a	b	c
d	a	f
c	b	d

A	B	C
d	a	f
b	g	a

A	B	C
a	b	c
c	b	d

In SQL come si esprimono?
Con INTERSECT

R - S

A	B	C
d	a	f

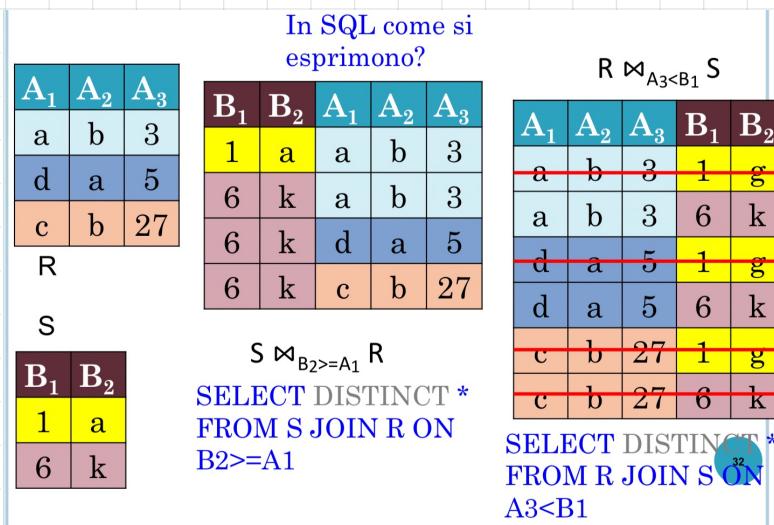
R \cap S = R - (R-S)

27

• THETA - JOIN

$$R \bowtie_{A \theta A'} S = \sigma_{A \theta A'} (R \times S)$$

\hookrightarrow se θ è una uscita binaria, prese le colonne di esclusione



- Determinare il **titolo** ed il **regista** dei film noleggiati il 15 Marzo 2006 dal cliente di codice 6635

$\sigma_{codCli = 6635 \wedge dataNol = '15-Mar-2006'} (\text{Noleggio}) \bowtie_{colloc = c} \rho_{colloc \leftarrow c} (\text{Video})$

c	titolo	regista	tipo	colloc	dataNol	codCli	dataRest
1111	underground	emir kusturica	v	1121	15-Mar-2006	6635	18-Mar-2006
1112	underground	emir kusturica	d	1122	15-Mar-2006	6635	18-Mar-2006
1113	big fish	tim burton	v	1113	15-Mar-2006	6635	18-Mar-2006
1114	big fish	tim burton	d	1129	15-Mar-2006	6635	20-Mar-2006
1115	edward mani di forbice	tim burton	d				
1116	nightmare before christmas	tim burton	v				
1117	nightmare before christmas	tim burton	d				
1118	ed wood	tim burton	d				
1119	mars attacks	tim burton	d				
1120	il mistero di sleepy hollow	tim burton	d				
1121	la sposa cadavere	tim burton	d				
1122	la fabbrica di cioccolato	tim burton	d				
1123	la fabbrica di cioccolato	tim burton	d				
1124	io non ho paura	gabriele salvatores	d				
1125	nirvana	gabriele salvatores	d				
1126	mediterraneo	gabriele salvatores	d				
1127	pulp fiction	quentin tarantino	v				
1128	pulp fiction	quentin tarantino	d				
1129	le iene	quentin tarantino	d				

35

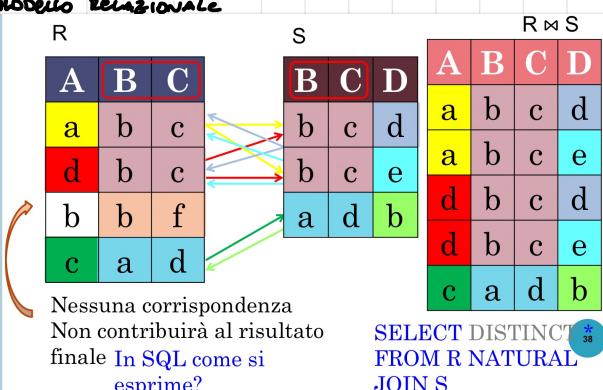
• JOIN NATURALE

è una semplificazione del JOIN:

- si cancellano le colonne con lo stesso nome
- non serve ridefinire le colonne
- non bisogna specificare il predicato di JOIN
- non si hanno colonne duplicates nel risultato

→ ha senso solo nella notazione con nome del modello relazionale

→ corrisponde ad una serie di EQUI-JOIN



- Individuare codice e nome dei clienti e titolo e anno dei film di Quentin Tarantino da lui noleggiati

$\Pi_{\text{codCli}, \text{nome}, \text{titolo}, \text{anno}}(\text{Cliente} \bowtie \text{Noleggio} \bowtie \text{Video} \bowtie \sigma_{\text{regista}=\text{'quentin tarantino'}}(\text{Film}))$

codCli	nome	titolo	anno
6635	paola	le iene	1992
6635	paola	pulp fiction	1994
6642	marco	pulp fiction	1994

43

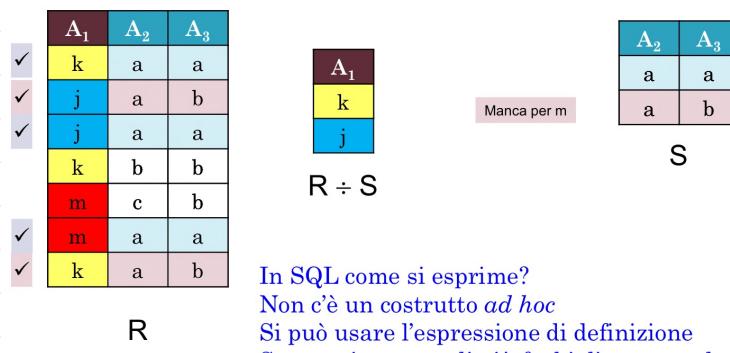
• DIVISIONE

→ INVERSA DEL PRODOTTO CARTESIANO

$$R = (R \times S) \div S$$

→ Scorrere a selezionare le tuple che su un sottoinsieme degli attributi (il divisore) comparano con tutte le

POSSIBILI COMBINAZIONI DI VALORI



In SQL come si esprime?

Non c'è un costrutto *ad hoc*

Si può usare l'espressione di definizione

Spesso ci sono modi più furbi di ottenere lo stesso risultato usando il potere espressivo ulteriore di SQL (operazioni sui dati, count, group by...)

47

Determinare i codici dei clienti che hanno noleggiato tutti i film di Tim Burton

1. Film di Tim Burton

$TB = \Pi_{\text{titolo}, \text{regista}}(\sigma_{\text{regista}=\text{'tim burton'}}(\text{Film}))$

titolo	regista
edward mani di forbice	tim burton
nightmare before christmas	tim burton
ed wood	tim burton
mars attacks	tim burton
il mistero di sleepy hollow	tim burton
big fish	tim burton
la sposa cadavere	tim burton
la fabbrica di cioccolato	tim burton

48

codCli	titolo	regista
6635	underground	emir kusturica
6635	edward mani di forbice	tim burton
6635	nightmare before christmas	tim burton
6635	ed wood	tim burton
6642	underground	emir kusturica
6635	mars attacks	tim burton
6635	il mistero di sleepy hollow	tim burton
6642	nightmare before christmas	tim burton
6642	ed wood	tim burton
6635	la sposa cadavere	tim burton
6635	la fabbrica di cioccolato	tim burton
6635	big fish	tim burton
6635	le iene	quentin tarantino
6642	mars attacks	tim burton
6610	mediterraneo	gabriele salvatores
6610	underground	emir kusturica
6610	big fish	tim burton
6642	pulp fiction	quentin tarantino
6610	io non ho paura	gabriele salvatores
6610	edward mani	gabriele salvatores
6642	io non ho paura	tim burton
6610	nightmare before christmas	tim burton
6635	pulp fiction	quentin tarantino
6635	nirvana	gabriele salvatores
6642	la fabbrica di cioccolato	tim burton
6642	big fish	tim burton

→ 6642 non sarà nel risultato

→ 6610 non sarà nel risultato

NC ÷ TB

2. Film noleggiati dai clienti

$NC = \Pi_{\text{codCli}, \text{titolo}, \text{regista}}(\text{Noleggio} \bowtie \text{Video})$

codCli	titolo	regista
6635	underground	emir kusturica
6635	edward mani di forbice	tim burton
6635	nightmare before christmas	tim burton
6635	ed wood	tim burton
6642	underground	emir kusturica
6635	mars attacks	tim burton
6635	il mistero di sleepy hollow	tim burton
6642	nightmare before christmas	tim burton
6642	ed wood	tim burton
6635	la sposa cadavere	tim burton
6635	la fabbrica di cioccolato	tim burton
6635	big fish	tim burton
6635	le iene	quentin tarantino
6642	mars attacks	tim burton
6610	mediterraneo	gabriele salvatores
6610	underground	emir kusturica
6610	big fish	tim burton
6642	pulp fiction	quentin tarantino
6610	io non ho paura	gabriele salvatores
6610	edward mani	gabriele salvatores
6642	io non ho paura	tim burton
6610	nightmare before christmas	tim burton
6635	pulp fiction	quentin tarantino
6635	nirvana	gabriele salvatores
6642	la fabbrica di cioccolato	tim burton
6642	big fish	tim burton

47

ALGEBRA RELAZIONALE – SINTESI

Op.	Funzionalità	Cond.	Semantica
Π_A	$\mathcal{R}(U) \rightarrow \mathcal{R}(A)$	$A \subseteq U$	$\Pi_A(R) = \{t[A] \mid t \in R\}$
σ_F	$\mathcal{R}(U) \rightarrow \mathcal{R}(U)$	$A(F) \subseteq U$	$\sigma_F(R) = \{t \mid t \in R \wedge F(t)\}$
\times	$\mathcal{R}(U) \times \mathcal{R}(V) \rightarrow \mathcal{R}(U \cup V)$	$U \cap V = \emptyset$	$R_1 \times R_2 = \{t_1 \cdot t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2\}$
\cup	$\mathcal{R}(U) \times \mathcal{R}(U) \rightarrow \mathcal{R}(U)$		$R_1 \cup R_2 = \{t \mid t \in R_1 \vee t \in R_2\}$
$-$	$\mathcal{R}(U) \times \mathcal{R}(U) \rightarrow \mathcal{R}(U)$		$R_1 - R_2 = \{t \mid t \in R_1 \wedge t \notin R_2\}$
\cap	$\mathcal{R}(U) \times \mathcal{R}(U) \rightarrow \mathcal{R}(U)$		$R_1 \cap R_2 = \{t \mid t \in R_1 \wedge t \in R_2\}$
\bowtie_F	$\mathcal{R}(U) \times \mathcal{R}(V) \rightarrow \mathcal{R}(U \cup V)$	$U \cap V = \emptyset$	$R_1 \bowtie_F R_2 = \{t_1 \cdot t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2\}$ $\wedge F(t_1 \cdot t_2)\}$
\bowtie	$\mathcal{R}(U) \times \mathcal{R}(V) \rightarrow \mathcal{R}(U \cup V)$		$R_1 \bowtie R_2 = \{t \mid t[U] \in R_1 \wedge t[V] \in R_2\}$
\div	$\mathcal{R}(U) \times \mathcal{R}(V) \rightarrow \mathcal{R}(U \setminus V)$	$V \subset U$	$R_1 \div R_2 = \{t \mid \forall t_2 \in R_2 \exists t_1 \in R_1 \text{ t.c.}$ $t_1[U \setminus V] = t, t_1[V] = t_2\}$

LINGUAGGIO SQL - Query 1

select {[DISTINCT]} { }

↳ OPZIONALE, cancella le tuple → la omessa non è più un'intera in risultato

FROM

WHERE

↳ CLAUSOLA OPZIONALE

CLAUSOLA DISTINCT

SELECT DISTINCT
genere
FROM Film;

genere
drammatico
fantastico
animazione
fantascienza
horror
commedia
thriller

SELECT
genere
FROM Film;

genere
drammatico
fantastico
animazione
fantascienza
horror
fantastico
animazione
fantastico
drammatico
fantascienza
commedia
thriller
thriller

QUA NON SERVE, HO GIÀ SELEZIONATO LE CITTÀ DELL'ATTORI E NON AVO' DIPENDENTI



SELECT DISTINCT titolo, regista

FROM Film

WHERE anno < 1999 AND

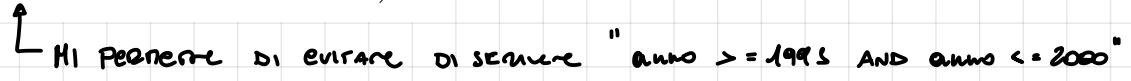
genere = 'fantascienza';

operatori di confronto

- operatori standard matematici $=, >, <, \geq, \leq, \neq$
- BETWEEN (due valori min e max) equivale a $\leq \text{max}$ AND $\geq \text{min}$
- IN (insieme $\{v_1, \dots, v_n\}$) equivalente a $= v_1 \text{ OR } \dots \text{ OR } = v_n$
- LIKE (pattern) seleziona stringhe che si adeguano al pattern

SELECT * FROM Film

WHERE anno BETWEEN 1995 AND 2000;


Mi permette di evitare di scrivere "anno ≥ 1995 AND anno ≤ 2000 "

SELECT * FROM Film

WHERE genere IN ('horror', 'fantascienza');


EVITO DI SCRIVERE OR IN GENERE

CONFRONTO TRA STRINGHE DI CARATTERI: LIKE

- semplici operazioni di *pattern-matching* su valori di tipo stringa
- formato: e [NOT] LIKE *pattern*
- un *pattern* è una stringa di caratteri che può contenere i caratteri speciali % e _
 - il carattere % denota una sequenza di caratteri arbitrari di lunghezza qualsiasi (anche zero)
 - il carattere _ denota esattamente un carattere

- Determinare tutti i film che hanno 'd' come terza lettera del titolo

SELECT * FROM Film
WHERE titolo LIKE '_ _ d%';

• ASSEGNAME UN NOME AD UNA COLONNA:

AS <NOME COLONNA>

• FUNZIONI PER DATE E TEMPI

→ COSTANTI: • CURRENT_DATE

• CURRENT_TIME

• CURRENT_TIMESTAMP

→ SELETTORI DA VALORI TEMPORALI AD INTERI

• EXTRACT (FROM c)

→ EXTRACT (DAY FROM DATE '08-Ott-1969')

Ritrovare la collocazione del video noleggiato e la durata (in giorni) di ogni noleggio **di durata superiore a due giorni** del cliente di codice 6635

```
SELECT colloc, (dataRest - dataNol) DAY AS durata
FROM Noleggio WHERE codCli = 6635 AND
(dataRest - dataNol) DAY > INTERVAL '2' DAY;
```

Risultato

colloc	durata
1117	4
1118	4
1121	3
1122	3
1113	3
1129	5

qui non si può usare durata

nel risultato viene associato il nome **durata** alla colonna virtuale

53

• OPERAZIONI INSIEMISTICHE

$Q_1 \text{ op } Q_2$

- UNION
- MINUS { **SONO SINONIMI IN SQL**
- EXCEPT
- INTERSECT

↳ Q_1 e Q_2 Devono avere lo stesso numero di colonne ed i dati corrispondenti devono essere compatibili

→ UNION

$Q_1 \text{ UNION } Q_2$ Restituisce tutte le tuple restituite da Q_1 e/o da Q_2 **ELIMINANDO EVENTUALI DUPLICATI**

↳ UNION ALL NON LI ELIMINA

- Esempio: determinare i nomi ed i cognomi di registi o di clienti della nostra videoteca

$Q_1 \left[\begin{array}{l} \text{SELECT regista} \\ \text{FROM Film} \end{array} \right]$ nome della colonna in Q_1 → $\begin{array}{l} \text{regista} \\ \text{emir kusturica} \\ \text{tim burton} \\ \text{gabriele salvatores} \\ \text{quentin tarantino} \\ \text{anna rossi} \\ \text{paola bianchi} \\ \text{marco verdi} \end{array}$

$Q_2 \left[\begin{array}{l} \text{UNION} \\ \text{SELECT nome || ' ' || cognome} \\ \text{FROM Cliente;} \end{array} \right]$ colonna calcolata non si chiama regista

→ INTERSECT

$Q_1 \text{ INTERSECT } Q_2$ Restituisce tutte le tuple restituite sia da Q_1 che da Q_2 senza duplicati

- Esempio: determinare gli anni in cui sono usciti sia film di Tim Burton sia film di Quentin Tarantino

$Q_1 \left[\begin{array}{l} \text{SELECT anno FROM Film WHERE regista = 'tim burton'} \\ \text{INTERSECT} \end{array} \right]$

$Q_2 \left[\begin{array}{l} \text{SELECT anno FROM Film WHERE regista = 'quentin} \\ \text{tarantino';} \end{array} \right]$

- il risultato è il solo anno 1994

→ EXCEPT

$Q_1 \text{ EXCEPT } Q_2$ restituisce tutte le tuple da Q_1 , ma non da Q_2 senza doppioni

- Esempio: determinare gli anni in cui sono usciti film di Tim Burton ma non film di Quentin Tarantino

$Q_1 \text{ [SELECT anno FROM Film WHERE regista = 'tim burton']}$
EXCEPT

$Q_2 \text{ [SELECT anno FROM Film WHERE regista = 'quentin tarantino']}$;

anno
1990
1993
1996
1999
2003
2005
2005

ORDINAMENTO DEL RISULTATO

per specificare un determinato ordinamento alla fine dell'interrogazione:

$\text{ORDER BY } <\text{nome colonna} \text{ [ASC | DESC]}> \{, <\text{nome colonna} \text{ [ASC | DESC]}> \}^*$

elencare la collocazione del video, la data di noleggio e la data di restituzione di tutti i noleggi del cliente 6635, in ordine crescente in base alla data di inizio del noleggio

$\text{SELECT colloc, dataNol, dataRest}$
FROM Noleggio
WHERE codCli = 6635
 ORDER BY dataNol;

colloc	dataNol	dataRest
1111	01-Mar-2006	02-Mar-2006
1115	01-Mar-2006	02-Mar-2006
1117	02-Mar-2006	06-Mar-2006
1118	02-Mar-2006	06-Mar-2006
1119	08-Mar-2006	10-Mar-2006
1120	08-Mar-2006	10-Mar-2006
1121	15-Mar-2006	18-Mar-2006
1122	15-Mar-2006	18-Mar-2006
1113	15-Mar-2006	18-Mar-2006
1129	15-Mar-2006	20-Mar-2006
1127	22-Mar-2006	?
1125	22-Mar-2006	?

UNION + ORDER BY

• VA USATA SOLO ALLA FINE DELL'INTERROGAZIONE

• NON SI POSSONO USARE I NOMI PER SPECIFICARE LE COLONNE SU CUI ESEGUIRE L'ORDINAMENTO

($\text{SELECT regista FROM Film}$
UNION

Senza le tonde
 • ha precedenza order by su union
 • il parser produce
 $Q_1 \text{ union } (Q_2 \text{ ORDER BY 1})$
 ⇒ errore perché ORDER BY non è in fondo

$\text{SELECT nome || ' ' || cognome FROM Cliente) ORDER BY 1;}$

TIPI DI JOIN

<NOME RELAZIONE> CROSS JOIN <NOME RELAZIONE>

- PRODOTTO CARTESIANO PURO E SENZA F
- FROM R CROSS JOIN R' equivale a FROM R, R'

<NOME RELAZIONE> JOIN <NOME RELAZIONE> ON <PREDICATO>

- PRODOTTO CARTESIANO FILTRATO: restituisce solo le tuple che soddisfano il predicato
- FROM R JOIN R' ON F equivale a FROM R, R' WHERE F

<Nome Relazione> NATURAL JOIN <Nome Relazione>

- I VALORI DELLE COLONNE CON LO STESSO NOME NELLE DUE RELAZIONI DEVONO ESSERE uguali
- NELLO SSETTORE DEL RISULTATO LE COLONNE CON LO STESSO NOME NELLE DUE RELAZIONI CORRISPONDANO UNA VERA SOLO

<Nome Relazione> JOIN <Nome Relazione> USING (<lista nomi colonne>)

- Come il NATURAL JOIN MA SOLO PER LE COLONNE ELENcate

→ EVENTUALI ALTRI NOMI DI COLONNE CON LO STESSO NOME NELLE DUE RELAZIONI RESTANO DUPLICATI NEL RISULTATO E NON SOFFRONO A CONDITIONI

OUTER JOIN

- IN R JOIN S NON SI HA TRACCIA DELLE TUPLE DI R CHE NON CORRISPONDONO AD ALCUNA TUPLA DI S
- L'OPERATORE DI OUTER JOIN AGGIUNGE AL RISULTATO LE TUPLE DI R E S CHE NON SONO PARTECIPANTI AL JOIN, COMPLETANDOLE CON NULL
- L'OPERATORE DI JOIN ORIGINARIO È ANCHE DETTO INNER JOIN
- LA VARIANTE OUTER PUÒ ESSERE UTILIZZATA PER TUTTI I TIPI DI JOIN (TRAMME PER IL CROSS, NON HA SENSO)

VARIANTI

- **FULL**: SIA LE TUPLE DI R CHE QUELLE DI S CHE NON PARTECIPANO AL JOIN VENGONO COMPLETATE ED INSERITE NEL RISULTATO
- **LEFT**: LE TUPLE DI R CHE NON PARTECIPANO AL JOIN VENGONO COMPLETATE ED INSERITE NEL RISULTATO
- **RIGHT**: LE TUPLE DI S CHE NON PARTECIPANO AL JOIN VENGONO COMPLETATE ED INSERITE NEL RISULTATO

Per ogni video contenente un film di Tim Burton di genere fantastico vogliamo visualizzare la sua collocazione, il titolo ed i codici dei clienti che l'hanno eventualmente noleggiato

SELECT colloc, titolo, codCli

FROM Film NATURAL JOIN Video

NATURAL LEFT OUTER JOIN Noleggio

WHERE regista = 'tim burton' AND genere = 'fantastico';

colloc	titolo	codCli
1113	big fish	6635
1113	big fish	6642
1114	big fish	6610
1115	edward mani di forbice	6635
1115	edward mani di forbice	6610
1122	la fabbrica di cioccolato	6635
1122	la fabbrica di cioccolato	6642
1123	la fabbrica di cioccolato	?

senza l'utilizzo dell'outer join i video, quali il 1123, che non sono mai stati noleggiati non avrebbero fatto parte del risultato

SQL - DDL

operazione	DDL	DML
creazione	CREATE	INSERT
modifica	ALTER	UPDATE
cancellazione	DROP	DELETE
interrogazione		SELECT

TIPI DI DATO CARATTERE

• CHAR [(n)]
 → DEFALUT 1
 → STRINGS DI ESISTENZE n CARATTERI

• VARCHAR (n) → STRINGS CON PAROLE DI LUNGHEZZA MASSIMA n

I VALORI DI TIPO CHAR / VARCHAR VANNO TRACCUSI TRA SINGOLI APICI

TIPI DI DATO TEMPORALI : ISTANTANEE

• DATE → RISOLUZIONE DI UN GIORNO

DATE << STRINGA CHE RAPPRESENTA DATA>>

⑥ Date '27-ott-1969'. Date '1969-10-27'

• TIME [(p)]

 → p è l'eventuale numero di cifre frazionarie a cui si c' interessa

 → FORMATO DEI VALORI TIME 'hh:mm:ss[.nnnnnn]'

• TIMESTAMP [(p)] = DATE + TIME [(p)]

→ TIMESTAMP << STRINGA CHE RAPPRESENTA DATA>> 'hh:mm:ss[.nnnnnn]'

TIPI DI DATO TEMPORALI : INTERVALLI

• INTERVAL << STRINGA CHE ESPRIME DURATA >> << UNITÀ DI MISURA>>

⑥ INTERVAL '3' YEAR

• INTERVAL << STRINGA CHE ESPRIME DURATA >> << UNITÀ DI MISURA>> TO << UNITÀ DI MISURA>>

TIPI BOOLEANI $\begin{cases} \text{AND} \\ \text{NOT} \\ \text{OR} \end{cases}$ → TRUE FALSE UNKNOWN

CREAZIONE DI RELAZIONI

CREATE TABLE Video

```
(colloc DECIMAL(4),
titolo VARCHAR(30),
regista VARCHAR(20),
tipoo CHAR DEFAULT 'd');
```

↳ Il valore di default deve appartenere al dominio

• OBBLIGATORIETÀ DI COLONNE (NOT NULL)

• CHIAVI (UNIQUE e PRIMARY KEY) → chiavi primarie con primary key, chiavi alternative con unique

In una tabella è possibile specificare più colonne unique, ma una sola primary key

• CHIAVI ESTERNE (FOREIGN KEY)

• VINCOLI CHECK SU COLONNA O SU TUTTA

• ESEMPIO.

```
CREATE TABLE Noleggio
```

```
(colloc DECIMAL(4),
dataNol DATE DEFAULT CURRENT_DATE,
codCli DECIMAL(4) NOT NULL,
dataRest DATE,
```

PRIMARY KEY (colloc,dataNol),

UNIQUE (colloc,dataRest);

ci possono due i
lo stesso video

{ METTO IN FONDO
SI HA PIÙ COLONNE

→ se sono primary key, e' tutto NOT NULL

Clausola opzionale FOREIGN KEY del comando CREATE TABLE

chiave esterna della relazione che si sta definendo (relazione referente)

devono corrispondere ad una chiave della relazione riferita

FOREIGN KEY (<lista nomi colonne>)
REFERENCES <nome relazione>(<lista nomi colonne riferite>)

[ON DELETE { NO ACTION |
CASCADE | SET NULL | SET
DEFAULT }]
[ON UPDATE { NO ACTION |
CASCADE | SET NULL | SET
DEFAULT }]

CREATE TABLE Film

```
(titolo VARCHAR(30),
regista VARCHAR(20),
anno DECIMAL(4) NOT NULL,
genere CHAR(15) NOT NULL,
valutaz NUMERIC(3,2),
PRIMARY KEY  
(titolo,regista))
```

CREATE TABLE Video

```
(colloc DECIMAL(4) PRIMARY KEY,  
titolo VARCHAR(30) NOT NULL,  
regista VARCHAR(20) NOT NULL,  
tipo CHAR NOT NULL DEFAULT 'd',  
FOREIGN KEY (titolo,regista)  
REFERENCES Film);
```

- non necessariamente gli stessi nomi
- i domini degli attributi corrispondenti devono essere compatibili
- si può (deve in caso di ambiguità) indicare quali sono le colonne riferite

Se la chiave esterna è costituita da un solo attributo, si può far seguire la specifica della colonna da REFERENCES <nome relazione>

Esempio videoteca (segue)

CREATE TABLE Noleggio

(colloc DECIMAL(4) REFERENCES Video
ON DELETE CASCADE
ON UPDATE CASCADE,

dataNol DATE DEFAULT CURRENT_DATE,

codCli DECIMAL(4) NOT NULL REFERENCES Cliente

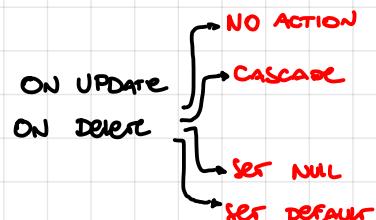
ON DELETE CASCADE
ON UPDATE CASCADE,

dataRest DATE,

PRIMARY KEY (colloc,dataNol),

UNIQUE (colloc,dataRest));

→ per specificare le azioni da eseguire nel caso di cancellazione/modifica di una tupla della tabella referita mantenere come esterna della tabella referente



MODIFICHE

• **NO ACTION**: La tupla viene cancellata solo se non esiste alcuna tupla in referente che vi fa riferimento

• **CASCADE**: La cancellazione della tupla implica la cancellazione di tutte le tuple in referente che vi fanno riferimento
→ per la modifica, l'opzione CASCADE implica che ogni chiavi esterna ottenga il nuovo valore di chiave della tupla

• **SET NULL**: La cancellazione/modifica della tupla implica che in tutte le tuple nell'insieme referente che vi fanno riferimento la chiave esterna viene posta a NULL (se ammesso)

• **SET DEFAULT**: La cancellazione/modifica della tupla implica che in tutte le tuple nell'insieme referente che vi fanno riferimento la chiave esterna viene posta al valore di default specificato per la colonna che costituiscono la chiave esterna

→ L'OPZIONE DI DEFAULT è NO ACTION

CANCELLAZIONI DI RELAZIONI

DROP TABLE <nome relazione> {RESTRICT | CASCADE}

• La relazione viene cancellata solo se non c'è riferimento da altri elementi dello schema della base di dati

• La relazione e tutti gli elementi dello schema della base di dati che la informano vengono cancellati

ALTER TABLE <nome relazione> <modifica>;

aggiunta

ADD [COLUMN]
<colonna>

ALTER TABLE Film

ADD COLUMN studio
VARCHAR(20);

- aggiunge in fondo alla relazione Film la colonna studio
- manca specifica del default ⇒ a tutte le tuple di Film viene assegnato il valore NULL per tale colonna

modifica del default

ALTER [COLUMN]
<colonna>

{SET DEFAULT
<valore default>

| DROP DEFAULT}

ALTER TABLE Video

ALTER COLUMN tipo
SET DEFAULT 'v';

ALTER TABLE Video

ALTER COLUMN tipo
DROP DEFAULT;

eliminazione

DROP [COLUMN]
<colonna>

{RESTRICT | CASCADE}

ALTER TABLE Film

DROP COLUMN
RESTRICT valutaz;

VI SONO ANCHE

ADD CONSTRAINT

C DROP CONSTRAINT



ALTER TABLE Film
ADD CONSTRAINT UNIQUE (studio, titolo, anno)

LINGUAGGI SQL - Query 2

Select COUNT (*), MIN(valutaz), AVG(valutaz), MAX(valutaz)

FROM Film;

↳ Valutazione più alta
↳ recupera valutazioni
↳ visualizza la tupla con valutazione più bassa
↳ Conta il numero di tuple

- SI USANO SOLO PER CONTARE LE EXPRESSION NELLA CLAUSOLA DI PROIEZIONE DI UN'INTERROGAZIONE

Select COUNT(DISTINCT *)

↳ Conta il numero di tuple senza doppiioni

VALORI NULL ESCLUSI

• MAX e MIN APPLICABILI SU VALORI ORDINATI

• SUM e AVG APPLICABILI SU VALORI NUMERICI

SELECT titolo, regista

FROM Film

WHERE genere = 'drammatico' AND valutaz = MIN(valutaz);

↳ SBAGLIATO, POSSO USARE LE FUNZIONI DI GRUPPO SOLO DOPO SELECT, PER FILTRARE LE TUPLE INVECE HO BISOGNO DI SOTTOINTERROGAZIONI

⇒ FUNZIONI DI GRUPPO SONO NELLA CLAUSOLA DI PROIEZIONE +

OPERATORE DI RAGGRUPPAMENTO - GROUP BY

determinare per ogni regista quanti suoi film con valutazione superiore a 3 sono presenti in catalogo

```
SELECT regista,  
COUNT(*) AS numeroFilmBuoni  
FROM Film  
WHERE valutaz >= 3  
GROUP BY regista;
```

IL RISULTATO DI UN'INTERROGAZIONE CHE CONTIENE UNA CLAUSOLA GROUP BY CONTIENE TANTE TUPLE

- QUANTI SONO I GRUPPI DI TUPLE RISULTANTI DAL PARTECIPANTE
- QUANTI SONO I VALORI DISTINTI DELLE PROIEZIONI SULLE CUIANE USATE PER RAGGRUPPARE

LA CLAUSOLA DI PROIEZIONE DI UN'INTERROGAZIONE CONTENENTE LA CLAUSOLA GROUP BY PUÒ SOLO INCLUDERE:

- UNA O PIÙ TRA LE COLONNE CHE COMPOSTO NELLA CLAUSOLA GROUP BY
- FUNZIONI DI GRUPPO

IL RISULTATO CONTIENE UNA SOLA TUPLA PER OGNI GRUPPO

→ LE COLONNE SU CUI NON SI RAGGRUPPA DUE TUPLE IN UNO STESSO GRUPPO POSSONO ASSUNGERE DIVERSI VALORI

- Partizionare i noleggi in base al cliente che li ha effettuati ed al regista del film noleggiato
- per ogni gruppo, determinare il numero di noleggi e la durata massima (in giorni) di tali noleggi

```
SELECT codCli, regista, COUNT(*) AS NumN,  
       MAX((dataRest-dataNol) DAY) AS durataM
```

```
FROM Noleggio NATURAL JOIN Video
```

```
GROUP BY codCli, regista;
```

Risultato

codCli	regista	numN	durataM
6635	emir kusturica	1	1
6635	tim burton	8	4
6635	gabriele salvatores	1	?
6635	quentin tarantino	2	5
6642	emir kusturica	1	1
6642	tim burton	5	1
6642	gabriele salvatores	1	1
6642	quentin tarantino	1	2
6610	emir kusturica	1	2
6610	tim burton	4	1
6610	gabriele salvatores	2	1

Se volessi anche il nome e cognome del cliente

```
SELECT codCli, nome, cognome, regista, COUNT(*) AS NumN,
```

```
       MAX((dataRest-dataNol) DAY) AS durataM
```

```
FROM Noleggio NATURAL JOIN Video NATURAL JOIN Cliente
```

```
GROUP BY codCli, nome, cognome, regista;
```

Anche se nome, cognome non influenzano i gruppi (perché sono determinati a codCli) per poterli restituire bisogna includerli nel group by

CLAUSOLA HAVING

- SPECIFICA UN FILTRO SUI GRUPPI OTTENUTI DAL PARTIZIONAMENTO
- E' UNA COMBINAZIONE BOOLEANA DI PREDICHI (o un AND)
- → CIASCUN ATTORE DEVE CONFERIRE FUNZIONI DI GRUPPO
- → NON PUÒ CONTENERE CONDIZIONI SUI VALORI DELLE SINGOLE COLONNE
- NON VI E' ALQUA RELAZIONE TRA LE FUNZIONI DI GRUPPO EVENTUALMENTE USATE NELLA CLAUSOLA SELECT E' ANCHE USARE NELLA CLAUSOLA HAVING

Per ogni regista **che ha girato almeno due film prima del 2000**, determinare quanti sono tali film, di quanti generi diversi e la valutazione minima, media e massima di tali film

```
SELECT regista, COUNT(*) AS numF,  
       COUNT(DISTINCT genere) AS numG,  
       MIN(valutaz) AS minV, AVG(valutaz) AS avgV, MAX(valutaz) AS maxV  
FROM Film  
WHERE anno < 2000  
GROUP BY regista  
HAVING COUNT(*) >= 2;
```

Risultato

regista	numF	numG	minV	avgV	maxV
tim burton	5	5	3.00	3.62	4.00
gabriele salvatores	2	2	3.00	3.40	3.80
quentin tarantino	2	1	3.50	3.75	4.00

WHERE



GROUP BY



HAVING



Risultato

- TRUE
- FALSE
- UNKNOWN (?) → le tuple UNKNOWN non vengono restituite dalle interrogazioni

SOTTO - INTERROGAZIONI

Determinare i film la cui valutazione è superiore alla media

Select * FROM Film

Where valutaz > (Select AVG(valutaz) FROM Film);

- Esempio: determinare il titolo ed il regista del film drammatico di valutazione minima

SELECT titolo, regista FROM Film

WHERE genere = 'drammatico' AND

valutaz = (SELECT MIN(valutaz)

FROM Film

WHERE genere = 'drammatico');

Le sotto-interrogazioni che restituiscono un solo valore sono dette SCALARI

L'errore RUN-TIME in caso contrario

Se nessuna tupla verifica la sotto-interrogazione, viene restituito il valore NULL

SOTTO - INTERROGAZIONI TABLE - SUBQUERY

Sono sotto-interrogazioni che restituiscono una tabella

In SQL si usano i quantificatori ANY e ALL insieme all'operatore di confronto e la sotto-interrogazione

Determinare titolo, regista ed anno dei film più vecchi di tutti i film di Quentin Tarantino (si può fare anche con MIN)

SELECT titolo, regista, anno

FROM Film

WHERE anno < ANY (SELECT anno FROM Film

 WHERE regista = 'quentin tarantino');

Determinare titolo, regista ed anno dei film più vecchi di almeno un film di Quentin Tarantino (si può fare anche con MAX)

SELECT titolo, regista, anno

FROM Film

WHERE anno < ANY (SELECT anno FROM Film

 WHERE regista = 'quentin tarantino');

SOTTO-INTERROGAZIONI – ALL ⇔ MAX/MIN

- Le interrogazioni
SELECT ... FROM ... WHERE exp >= ALL SELECT exp'
e
SELECT ... FROM ... WHERE exp >= SELECT MAX(exp')
Sono equivalenti?
- Esempio: per determinare il titolo ed il regista del film drammatico di valutazione minima
SELECT titolo, regista FROM Film
WHERE genere = 'drammatico' AND
valutaz >= ALL (SELECT valutaz FROM Film WHERE genere = 'drammatico');
equivale a
SELECT titolo, regista FROM Film
WHERE genere = 'drammatico' AND
valutaz = SELECT MAX(valutaz) FROM Film WHERE genere = 'drammatico';
????
- In assenza di valori nulli sì
 - altrimenti la prima si valuta in FALSE, la seconda potrebbe ritornare TRUE

APPARTENENZA INFERIRE

IN e NOT IN

Esempio: elencare il titolo dei film di Quentin Tarantino usciti nello stesso anno di un film di Tim Burton

SELECT titolo
FROM Film
WHERE regista = 'quentin tarantino'
AND anno IN (SELECT anno FROM Film WHERE regista = 'tim burton');
Risultato: 'pulp fiction' (uscito nel 1994)

33

Restituire il titolo dei film di Quentin Tarantino usciti in un anno in cui non sono usciti film di Tim Burton

SELECT titolo FROM Film
WHERE regista = 'quentin tarantino'
AND anno NOT IN (SELECT anno FROM Film
WHERE regista = 'tim burton');

e' possibile selezionare più di una colonna tramite una sottointerrogazione

↳ BISOGNA RACCHIUDERE FRA PARENTESI LA LISTA DELLE COLONNE A SINISTRA DELL'OPERATORE DI CONFRONTO

- Esempio: elencare il titolo dei film presenti nella videoteca con lo stesso anno di uscita e genere di un film di Tim Burton
SELECT titolo FROM Film
WHERE regista <> 'tim burton' AND
(anno,genere) IN (SELECT anno, genere FROM Film
WHERE regista = 'tim burton');

- Una sotto-interrogazione può avere al suo interno un'altra sotto-interrogazione, predicati di join, clausole group by e tutti i predicati visti
- la clausola di qualificazione di una interrogazione può contenere una qualsiasi combinazione di condizioni normali e condizioni con sotto-interrogazioni

SQL - query 3

SOTTO-INTERROGAZIONI CORRELATE

Esempio: si vogliono determinare titolo, regista ed anno dei film la cui valutazione è superiore alla media delle valutazioni dei film **dello stesso regista**

↳ BISOGNA RICORRERE AGLI ALIAS DI RELAZIONE!

SI DEFINISCE NELLA CLAUSOLA FROM FACCENDO SEGUIRE IL NOME DELLA RELAZIONE DA:

- UN IDENTIFICATORE
- AL SEGUITO DA UN IDENTIFICATORE

SELECT titolo, regista, anno FROM Film **X**
 WHERE valutaz > (SELECT AVG(valutaz)
 FROM Film
 WHERE regista = **X**.regista);

UTILE ANCHE PER:

→ ABBREVIARE LA SCRITTURA DI QUERY

SELECT **X**.a FROM
 LaMiaTabellaConNomeSignificativoQuindiLungo **X**

→ FAR RIFERIMENTO A DUE DIVERSE TUPLE DELLA STESSA RELAZIONE

SELECT DISTINCT **X**.regista FROM Film **X**, Film **Y**
 WHERE **X**.anno = **Y**.anno AND **X**.regista = **Y**.regista
 AND **X**.titolo <> **Y**.titolo;

EXISTS e **NOT EXISTS**

→ RESTITUISCONO TRUE e FALSE, NON RESTITUISCONO MAI UNKNOWN

Esempio: i registi di cui sono usciti (almeno) due film diversi lo stesso anno

```
SELECT DISTINCT regista FROM Film X  
WHERE EXISTS ( SELECT * FROM Film  
    WHERE regista = X.regista  
    AND anno = X.anno  
    AND titolo <> X.titolo);
```

INTERSEZIONE e **DIFERENZA** SI POSSONO DEFINIRE IN SQL ANCHE TRAMITE [NOT] EXISTS

Esempio: determinare gli anni in cui sono usciti sia film di Tim Burton sia film di Quentin Tarantino ma non

```
SELECT anno FROM Film WHERE regista = 'tim burton'
```

~~INTERSECT MINUS~~

```
SELECT anno FROM Film WHERE regista = 'quentin tarantino'  
diventa
```

```
SELECT DISTINCT anno FROM Film
```

```
WHERE regista = 'tim burton' AND
```

```
NOT EXISTS ( SELECT * FROM Film F  
    WHERE regista = 'quentin tarantino'  
    AND anno = F.anno);
```

In generale

```
SELECT C1,...,CK FROM R1 WHERE P1
```

INTERSECT

```
SELECT C1,...,CK FROM R2 WHERE P2
```

si può esprimere anche come

```
SELECT C1,...,CK FROM R1 WHERE P1 AND  
EXISTS ( SELECT * FROM R2  
    WHERE P2 AND R1.C1=R2.C1 AND ...  
    AND R1.CK = R2.CK)
```

```
SELECT C1,...,CK FROM R1 WHERE P1
```

MINUS

```
SELECT C1,...,CK FROM R2 WHERE P2
```

si può esprimere anche come

```
SELECT C1,...,CK FROM R1 WHERE P1 AND  
NOT EXISTS (SELECT * FROM R2  
    WHERE P2 AND R1.C1=R2.C1 AND...  
    AND R1.CK = R2.CK)
```

le sono - interrogazioni correlate e l'operatore di NOT EXISTS permettono di esprimere l'operazione di **Divisione**, per cui SQL non prevede un operatore apposito

esempio: determinare i codici dei clienti che hanno noleggiato tutti i film di Tim Burton

viene "riformulata" come: determinare i codici dei clienti per cui non è possibile determinare un film di Tim Burton che il cliente non ha mai noleggiato

```
SELECT DISTINCT codCli FROM Noleggio X  
WHERE NOT EXISTS (SELECT * FROM Film F  
WHERE regista = 'tim burton' AND  
NOT EXISTS (SELECT *  
FROM Noleggio NATURAL JOIN Video  
WHERE codCli = X.codCli  
AND titolo = F.titolo  
AND regista = F.regista));
```

2

UN MODO ALTERNATIVO DI ESEMPLIFICARE LA DIVISIONE E' MEDIANTE L'USO DI FUNZIONI DI GRUPPO

per determinare i clienti che hanno noleggiato tutti i film di Tim Burton confrontiamo il numero di film di Tim Burton con il numero dei film di Tim Burton noleggiati dal cliente

```
SELECT codCli FROM Noleggio NATURAL JOIN Video  
WHERE regista = 'tim burton'  
GROUP BY codCli  
HAVING COUNT(DISTINCT titolo) =  
(SELECT COUNT(DISTINCT titolo)  
FROM Film  
WHERE regista = 'tim burton');
```

3

LINGUAGGIO SQL - MODIFICATE ALL'ISTANZA DI UNA RICHIESTA

Inserire un nuovo film, con

- titolo → "La tigre e la neve"
- regista → Roberto Benigni
- anno di produzione → 2005
- genere → commedia
- valutaz → 3

INSERT INTO Film(titolo, regista, anno, genere, valutaz)

VALUES ('la tigre e la neve', 'roberto benigni', 2005, 'commedia', 3);

oppure

INSERT INTO Film(anno, titolo, genere, regista, valutaz)

VALUES (2005, 'la tigre e la neve', 'commedia', 'roberto benigni', 3);



Se la lista delle colonne manca, escludere dalla lista di tutte le colonne di R nell'ordine dato dal comando CREATE TABLE

INSERT INTO Film

VALUES ('la tigre e la neve', 'roberto benigni', 2005, 'commedia', 3);

Se una colonna di R non compare nella lista, l'attributo corrispondente viene initializzato con il valore di default se specificato nel comando di creazione di R, con NULL altrimenti

Se non c'è default e una colonna ha vincolo di obbligatorietà ma non abbiano nello la colonna → errore

INSERIMENTO MEDIANTE QUERY

Inserire in data odierna noleggi relativi al cliente 6635 e a tutti i video contenenti film di Gabriele Salvatores che non aveva ancora noleggiato

INSERT INTO Noleggio(colloc, codCli)

SELECT colloc, 6635 FROM Video

WHERE regista = 'gabriele salvatores' AND (colloc, 6635)

NOT IN (SELECT colloc, codCli FROM Noleggio);

CANCELLAZIONE

Cancellare il film "La tigre e la neve"

DELETE FROM Film

WHERE titolo = 'la tigre e la neve'

AND regista = 'roberto benigni';

Cancellare i clienti che non hanno effettuato noleggi nell'ultimo anno

DELETE FROM Cliente

WHERE codCli NOT IN (

SELECT codCli FROM Noleggio

WHERE dataNol > (CURRENT_DATE - 1 YEAR));

MODIFICA

Raddoppiare le valutazioni dei film (ad esempio in corrispondenza ad un cambio di scala da 0-5 a 0-10)

UPDATE Film

SET valutaz = valutaz * 2;

Il cliente 6635 restituisce tutti i video che ha attualmente in noleggio e segnala che il noleggio è iniziato ieri (e non come erroneamente diversamente registrato)

UPDATE Noleggio

SET dataRest = CURRENT_DATE,
dataNol = (CURRENT_DATE - 1 DAY)

WHERE codCli = 6635 AND dataRest IS NULL;

14

Aumentare del 10% la valutazione dei film horror usciti dopo il 1999 che sono stati noleggiati da almeno due clienti

UPDATE Film

SET valutaz = valutaz * 1.1

WHERE genere = 'horror' AND anno > 1999

AND (titolo, regista) IN

(SELECT titolo, regista

FROM Noleggio NATURAL JOIN Video

GROUP BY titolo, regista

HAVING COUNT(DISTINCT codCli) >= 2);

Assegnare ad ogni film che è stato noleggiato almeno una volta nell'ultimo mese, una valutazione pari al 110% della valutazione media dei film dello stesso anno e genere

UPDATE Film X

SET valutaz = (SELECT 1.1 * AVG(valutaz)
FROM Film
WHERE anno = X.anno AND genere = X.genere)

WHERE EXISTS (

SELECT *
FROM Noleggio NATURAL JOIN Video
WHERE dataNol > CURRENT_DATE - 1 MONTH
AND titolo = X.titolo AND regista = X.regista
)

Vincoli statici

Su singola relazione

Su relazioni multiple

Su singola tupla

Su tuple multiple

Su singolo attributo

Su attributi multipli

vincoli di aggregazione

vincoli di dipendenza funzionale

NOT NULL

Data_restituzione > Data_noleggio

non ci sono più di 3 noleggi attivi per lo stesso cliente

il nome di un cliente è determinato dal suo codice

Vincoli CHECK su colonna

Vincoli CHECK su relazione/tabella

ASSERZIONI in SQL
(non sempre implementati)

5

```
CREATE TABLE Film ( ...
    valutaz DECIMAL(3,2),
    ...);
```

- **Vincolo su valutazione** (singolo attributo): la valutazione è un numero compreso tra 0 e 5
→ Vincolo su singolo attributo

Condizione dopo CHECK = formula che potrebbe essere inserita in clausola WHERE e che si riferisce all'attributo

Può includere sotto-interrogazioni

```
CREATE TABLE Film ( ...
    valutaz DECIMAL(3,2)
        CHECK (valutaz BETWEEN 0.00 AND 5.00),
    ...);
```

8

- CREATE TABLE Suggerimento(user...,voto....,titolo...,regista...);

```
CREATE TABLE Film ( ...
    valutaz DECIMAL(3,2)
        CHECK (valutaz =
            SELECT AVG(voto)
            FROM Suggerimento S
            WHERE titolo = S.titolo
            AND regista = S.regista),
    ...);
```

SELECT AVG(voto)
FROM Suggerimento S
WHERE titolo = S.titolo
AND regista = S.regista),

IL VINCOLO VENE VERIFICATO SOLO QUANDO SI INSERISCE O MODIFICA UNA TUPLA DELL'OGGETTO
(CHECK SU RELAZIONE) O UNA COLONNA (CHECK IN COLONNE)

• MODIFICHE AD ELEMENTI DIVERSI NON ATTIVANO LA VERIFICA → IL VINCOLO PUÒ ESSERE VIDIATO

• LA RELAZIONE VUOTA SODDISFA SEMPRE TUTTI I VINCOLI CHECK

• UN VINCOLO CHECK NON PUÒ IMPORRE AD UNA RELAZIONE DI CONTENERE (ALMENO) UN CERTO NUMERO DI TUPLE
(CHE SODDISFANO UNA CONDIZIONE)

ASSOLUTAMENTE DA EVITARE ESEGUIRE VINCOLI CHECK CHE VERIFICANO CONDIZIONI ARBITRARIE
ANCHE SU ALTRE TABELLE

• I VINCOLI CHECK DEVONO USARE SOLO CONDIZIONI CHE SI RIESCONO A VERIFICARE ESAMINANDO
LA SINGOLA TUPLA CUI ASSOCIAZIO IL VINCOLO
→ SCHEMA PIÙ COMPRENSIBILE
→ VERIFICA DEI VINCOLI CON MAGGIORE EFFICIENZA

SE VOGLIO ESEGUIRE CONDIZIONI CHE RICHIEDANO DI ESAMINARE PIÙ TUPLE, RICORRO ALL'UTILIZZO
DI:

- TRIGGER
- ASSEGNAZIONI → NON SUPPORTATE DAL DBMS

CONSTRAINT <nomeC> CHECK (<condizione>)

il nome dei vincoli serve a farvi riferimento per

- modificareli
ALTER TABLE <nomeT> ADD CONSTRAINT <nomeC>
CHECK (<condizione>)
- eliminarli
ALTER TABLE <nomeT> DROP CONSTRAINT <nomeC>

```

CREATE TABLE Video(
    colloc DECIMAL(4) CONSTRAINT PKey PRIMARY KEY,
    titolo VARCHAR(30) CONSTRAINT Tnn NOT NULL,
    regista VARCHAR(20) CONSTRAINT Rnn NOT NULL,
    tipo CHAR      CONSTRAINT Snn NOT NULL
                           DEFAULT 'd'
                           CONSTRAINT Tok CHECK (tipo IN ('d','v'))
CONSTRAINT FK FOREIGN KEY (titolo,regista)
REFERENCES Film ON DELETE RESTRICT);

```

Modifica vincolo Tok

```

ALTER TABLE Video DROP CONSTRAINT Tok;
ALTER TABLE Video ADD CONSTRAINT Tok CHECK (tipo IN
('d','v','x'));

```

Cancellazione vincolo Rnn

```

ALTER TABLE Video DROP CONSTRAINT Rnn;

```

20

ASSEZIONI

- SI POSSONO SPECIFICARE VINCOLI SU TUPLE MULTIPLE O REAZIONI MULTIPLE

CREATE ASSERTION <nome asserzione>

CHECK (<condizione>)

- **<nome asserzione>** è il nome che viene assegnato all'asserzione
- **<condizione>** è un predicato o una combinazione booleana di prediciati (come nel caso dei vincoli CHECK)
- **<condizione>** in genere specifica che l'insieme delle tuple che NON soddisfano il vincolo è vuoto
→ uso del predicato NOT EXISTS

IN POSTGRES I VINCOLI CHECK NON POSSONO CONTENERE SOTTO-INTERROGATIONI
NON E' NEPPURE POSSIBILE CREARE ASSEZIONI

TRANSAZIONI

SEQUENZA DI OPERAZIONI DI LETTURA E SCRITTURA SULLA BASE DI DATI A CUI VIENE GARANTITA UN'ESECUZIONE CHE SOGGETTA ALKEVNE BUONE PROPRIETA'

IL DBMS GARANTISCE CHE LE TRANSAZIONI VENGANO ESEGUITE NEL RISPETTO DELLA PROFESSIONE **ACID**

• **ATOMICITA'**: UNA TRANSAZIONE E' UN'UNITA' DI ELABORAZIONE.

TUTTE LE OPERAZIONI DI UNA TRANSAZIONE DOVONO ESSERE TRATTATE COME UNA SINGOLA UNITA': O VENGONO ESEGUITE TUTTE, OPPURE NON NE VENGONO ESEGUITA alcuna

• **CONSISTENZA**: UNA TRANSAZIONE LASCA IL DB IN UNO STATO CONSISTENTE

LA TRANSAZIONE TRASFORMA LA BASE DI DATI DA UNO STATO CONSISTENTE AD UN ALTRO ANCHE CONSISTENTE

• **ISOLAMENTO**: UNA TRANSAZIONE SI ESEGUE INDIPENDENTEMENTE DALLE ALTRE

Ogni TRANSAZIONE DEVE SEMPRE OSSERVARE UNA BASE DI DATI CONSISTENTE E IL SUO ESITO NON DEVE ESSERE INFUENZATO DA ESECUZIONI CONCORSANTI DI ALTRE TRANSAZIONI.

• **DURABILITA'**: GLI EFFETTI DI UNA TRANSAZIONE CHE HA TERMINATO CORRETTAMENTE LA SUA ESECUZIONE DOVONO ESSERE PERMANENTI AL TEMPO

ATOMICITA' E DURABILITA' → GESTORE DEL RIPRISTINO

CONSISTENZA E ISOLAMENTO → GESTORE DELLA CONCORRENZA

TRANSAZIONI FLAT

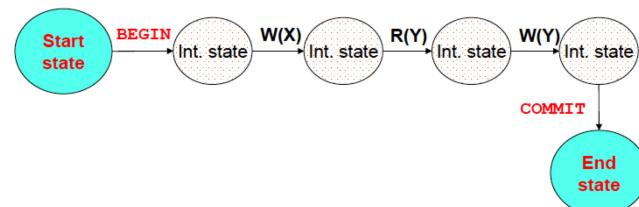
INIZIATA
→ IMPLICATAMENTE ALL'ESECUZIONE DEL PRIMO COMANDO SQL
→ ESPlicitamente con il comando BEGIN [TRANSACTION] o START TRANSACTION

BEGIN

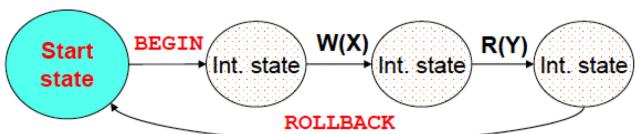
```
UPDATE CC
SET Saldo = Saldo - 50
WHERE Conto = 123
```

```
UPDATE CC
SET Saldo = Saldo + 50
WHERE Conto = 235
```

COMMIT → COMMUNICA AL TRANSACTION MANAGER IL TERMINE DELLE OPERAZIONI



ROLLBACK



Aviene quando la transazione ABORTisce
(Terminazione non corretta esplicita)

(Terminazione non corretta IMPULSA)

↳ AVVIENE PER UN GUASTO O PER LA VIOLAZIONE DI UN VINCOLO

Se per qualche motivo la transazione non puo' terminare correttamente la sua esecuzione il DBMS deve disfare (**UNDO**) le eventuali modifiche da essa apportate al DB

→ **VALUTAZIONE IMMEDIATA**: viene effettuata dopo ogni comando di manipolazione dei dati
→ TANTI controlli all'interno della stessa transazione

→ **VALUTAZIONE DIFFERITA**: il controllo viene effettuato al termine dell'esecuzione della transazione

Al momento della dichiarazione del vincolo, è possibile specificare se la sua valutazione:

• **NOT DEFERRABLE** la valutazione deve essere immediata

• **DEFERRABLE** (si può specificare una modalità di controllo iniziale per le transazioni)

↳ **INITIALLY IMMEDIATE** (default) → vincolo valutato in modo immediato
↳ **INITIALLY DEFERRED** → vincolo valutato in modo differito

SET CONSTRAINTS {<lista nomi vincoli> | ALL}

{DEFERRED | IMMEDIATE}

La modalità di terminazione è gestita dalla proprietà di **AUTOCOMMIT**

• **TRUE** → ogni comando comincia una transazione a sé stante

• **FALSE** → tutti i comandi eseguiti all'interno di una sessione costituiscono immediatamente una transazione

CONCORSO e RIPRISTINO

2 TIPI DI ESECUZIONE:

- SERIALE**: DIVERSE TRANSAZIONI CHE ACCEDONO A DATI CONDIVISI VENGONO ESEGUITE IN SEQUENZA

T1	T2
R(X)	
W(X)	
Commit	
	R(Y)
	W(Y)
	Commit

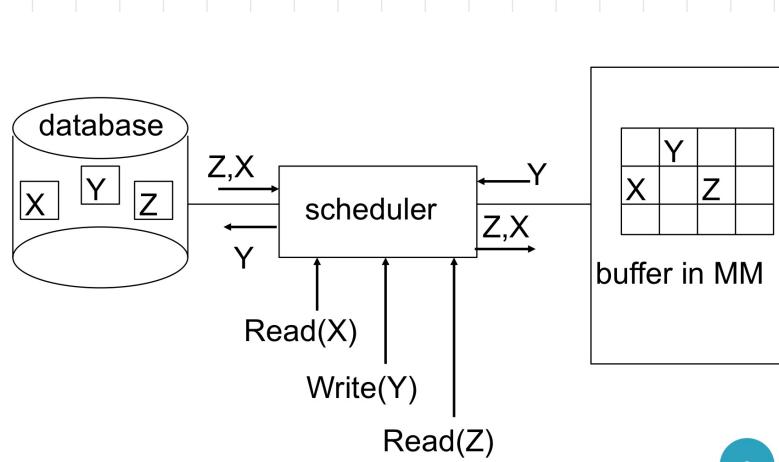
- CONCORSANTE**: IL DBMS PUÒ ESEGUIRE PIÙ TRASAZIONI IN CONCORSO, AVERZANDO L'ESECUZIONE DI OPERAZIONI DI ALTRE TRASAZIONI - interleaved execution

T1	T2
R(X)	
	R(Y)
	W(Y)
	Commit
W(X)	
Commit	

MIGLIORA IL TEMPO MEDIO DI RISPOSTA, MA PUÒ GENERARE ANOMALIE

IL GESTORE DELLA CONCORSO DEL GARANTIRE CONSISTENZA E ISOLAMENTO DELLE TRASAZIONI ESEGUIRE CONCORSANTEMENTE EVITANDO POSSIBILI ANOMALIE

Le trasazioni sono sequenze di operazioni di INPUT/OUTPUT in cui ogni operazione legge blocchi di memoria secondaria in pagine di buffer o scrive pagine di buffer in blocchi di memoria secondaria



ANOMALIE

IL TRANSACTION MANAGER DEVE GARANTIRE CHE TRASAZIONI CHE ESEGUISCONO IN CONCORSO NON INTERFERISCANO TRA LORO.

SE CIÒ NON AVVIENE, SI POSSONO AVERE 4 TIPI BAX DI PROBLEMI:

- LOST UPDATE**:

	T1	X	T2
	R(X)	1	
	X=X-1	1	
		1	R(X)
		1	X=X-1
	W(X)	0	
	Commit	0	
		0	W(X)
		0	Commit

T2 LEGGE IL VALORE DI X PRIMA CHE T1 (CHE LO HA GIÀ LETTO)
LO MODIFICA

T1 E T2 CONDIVISCONO ENTROBI L'ULTIMO BLOCCO DEL CONCORSO

• DIRTY READ :

T1	X	T2
R(X)	0	
X=X+1	0	
W(X)	1	
	1	R(X)
Rollback	0	
	0	...
	0	Commit

T2 legge un dato che non c'è

"DATA NON ANCORA FINITO ANCHE SE COMPARE NELLE TAPE DEL CONCERNO"

• UNREPEATABLE READ : 2 LETTURE INCONSISTENTI DELLO STESSO DATO

T1	X	T2
R(X) 1oE	0	
	0	R(X)
	0	X=X+1
	1	W(X)
	1	Commit
R(X) SoE	1	
Commit	1	

• PHANTOM ROW : AVVIENE QUANDO VENGONO INSERIRE O CANCELLARE TUPLE CHE UN'ALTRA TRANSAZIONE DOVREBBE LOCALMENTE CONSIDERARLE

T1	T2
R(t1)	
R(t2)	
...	
W(t1)	
W(t2)	
	Insert(t3)
...	
Commit	
	Commit

Schedule → **SERIALE**: SCHEDULE IN CUI LE TRANSAZIONI VENGONO ESEGUITE IN SEQUENZA
 -- l'isolamento è totale
 -- POTREBBE PORTARE A TEMPI DI ESECUZIONE PIÙ ALTI

Schedule

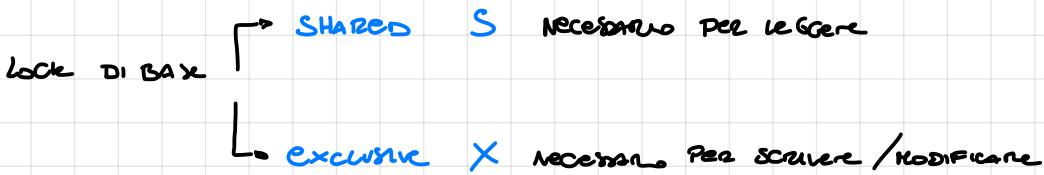
→ **SERIALIZZABILE**: SCHEDULE INDIVIDUALI, EVITA ANOMALIE E GARantisce l'isolamento totale
 → Richiede notevole di concordanza tra schedule

IL GESTORE DELLE TRANSAZIONI ESEGUE UN PROTOCOLLO DI CONTROLLO DELLA CONCERNZA. I PIÙ NOTI SONO:

- PROTOCOLLI BASATI SU LOCK
- PROTOCOLLI BASATI SU TIMESLICE

Protocollo di Locking

I Lock sono blocchi usati per disciplinare l'accesso a risorse condivise
→ La richiesta di lock è impedita, non è visibile a livello SQL



Il modulo del DBMS per tenere traccia di chi ha le risorse coinvolte in uso e di quale transazione le stanno usando e in che modo c'è il **LOCK MANAGER**

Su Y un'altra transazione ha un lock di tipo

compatibilità tra lock

		S	X
S	OK	NO	
X	NO	NO	

T richiede un lock di tipo

STRONG 2 PHASE LOCKING

Per risolvere i problemi di concorrenza bisogna focalizzarsi sul modo in cui le transazioni guadagnano i lock acquisiti

- UNA TRANSAZIONE PRIMA ACQUISISCE TUTTI I LOCK NECESSARI
- RILASCIÀ I LOCK SOLO AL TERMINE DELL'ESECUZIONE (COMMIT o ROLLBACK)

{ ISOLATION GARANTITA

→ Si possono verificare DEADLOCK, risolti con l'abbandono della transazione

Per quanto riguarda le anomalie, ovvero dei phantom row c'è la più difficile da risolvere.
SOLUZIONI ADOTTABILI:

- Acquisire un S-lock su tutta la table e richiedere X-lock sulle tuple da modificare
- "Predicate lock": riguarda le tuple che soddisfano un predicato

Il protocollo STRONG 2-PHASE LOCKING GARANTISCE SCHEMATICHE SERIALIZZABILI E QUINDI TRANSAZIONI CONPLICAMENTE ISOLATE → MAGGIOR GIUSTIZIA DEL SISTEMA



LIVELLI DI ISOLAMENTO

LIVELLO DI ISOLAMENTO

IN SQL PUO' ESSERE SCELTO TRA NELLE CORANTI:

Livello di isolamento	Lost update	Dirty read	Unrepeatable read	Phantom problem
READ UNCOMMITTED	NO	YES	YES	YES
READ COMMITTED	NO	NO	YES	YES
REPEATABLE READ	NO	NO	NO	YES
SERIALIZABLE	NO	NO	NO	NO

• SERIALIZABILE

STRONG 2PL
+
LOCK SU
TABELLE
e INDICI
PER EVITARE
PHANTOM ROW

- SI LEGGONO SOLO MODIFICHE EFFETTUATE DA TRANSAZIONI CHE HANNO EFFETTUATO IL COMMIT
- NON SI LEGGE / SCRIVE NESSUN VALORE FINCHÉ T NON TERMINA L'ESECUZIONE
- SE UN INSERIMENTO DI VALORI VENNE LETTO, NON PUO' ESSERE MODIFICATO DA ALTRI TRANSAZIONI FINCHÉ T NON HA TERMINATO DI LEGGERE

Table V			
T1	A	B	...
R(t1)	3	5	...
R(t2)	4	8	...
...			
R(t1)			
t1.A = t1.A+1			
W(t1)			
R(t2)			
Commit			

```
BEGIN TRANSACTION
SELECT *
FROM V
WHERE A > 3;

UPDATE V
SET A = A + 1
WHERE A = 3

COMMIT;
```

T1
S-lock(V)
R(t1)
R(t2)
...
X-lock(V)
R(t1)
t1.A = t1.A+1
W(t1)
R(t2)
...
Unlock(V)
Commit

• REPEATABLE READ

STRONG 2 PL

- SI LEGGONO SOLO MODIFICHE EFFETTUATE DA TRANSAZIONI CHE HANNO EFFETTUATO IL COMMIT
- NON SI LEGGE / SCRIVE NESSUN VALORE FINCHÉ T NON TERMINA L'ESECUZIONE

ESEMPIO ANNOTAZIONE REPEATABLE READ

Table V			
T1	A	B	...
R(t1)	3	5	...
R(t2)	4	8	...
...			
R(t1)			
t1.A = t1.A+1			
W(t1)			
R(t2)			
Commit			

```
BEGIN TRANSACTION
SELECT *
FROM V
WHERE A > 3;

UPDATE V
SET A = A + 1
WHERE A = 3

COMMIT;
```

T1
S-lock(t1)
R(t1)
S-lock(t2)
R(t2)
...
X-lock(t1)
R(t1)
t1.A = t1.A+1
W(t1)
X-lock(t2)
R(t2)
...
Unlock(t1)
Unlock(t2)
Commit

• **READ COMMITTED** → SI LEGGONO SOLO MODIFICHE EFFETTUATE DA TRANSACTIONI CHE HANNO EFFETTUATO IL COMMIT

→ NON SI SCRIVE NESSUN VALORE FINCHE' T NON TERMINA L'ESECUZIONE

→ UN VALORE SCRITO DA T PUO' ESSERE MODIFICATO PRIMA CHE TERMINI L'ESECUZIONE

I LOCK ESCLUSIVI VENGONO MANTENUTI FINO AL TERMINE DELLA TRANSACTIONE
QUELLI CONDIVISI VENGONO ACQUISITI E RILASCIATI APPENA POSSIBILE

ESEMPIO ANNOTAZIONE READ COMMITTED

Table V

T1	A	B	...
R(t1)	3	5	...
R(t2)			
...			
R(t1)			
t1.A = t1.A+1			
W(t1)			
R(t2)			
Commit			

T1
S-lock(t1)
R(t1)
Unlock(t1)
S-lock(t2)
R(t2)
Unlock(t2)
...
X-lock(t1)
R(t1)
t1.A = t1.A+1
W(t1)
X-lock(t2)
R(t2)
...
Unlock(t1)
Unlock(t2)
Commit

• **READ UNCOMMITTED** UNA TRANSACTIONE PUO' VEDERE MODIFICHE EFFETTUATE DA TRANSACTIONI CONCERNENTI ANCHE SE NON HANNO ANCORA EFFETTUATO IL COMMIT

→ NUOVO VALORE SCRITTO DA T PUO' ESSERE MODIFICATO DA ALTRE TRANSACTIONI PRIMA CHE T ABbia TERMINATO

→ NON VENGONO ACQUISITI LOCK CONDIVISI PRIMA DI EFFETTUARE LE LETTURE

LOCK ESCLUSIVI MANTENUTI FINO AL TERMINE DELLA TRANSACTIONE

LOCK CONDIVISI NON VENGONO RICHIESTI

NO LOCK SU LETTURE

ESEMPIO ANNOTAZIONE READ UNCOMMITTED

Table V

T1	A	B	...
R(t1)	3	5	...
R(t2)			
...			
R(t1)			
t1.A = t1.A+1			
W(t1)			
R(t2)			
Commit			

T1
R(t1)
R(t2)
...
X-lock(t1)
R(t1)
t1.A = t1.A+1
W(t1)
X-lock(t2)
R(t2)
...
Unlock(t1)
Unlock(t2)
Commit

LIVELLI DI ISOLAMENTO IN SQL

```
BEGIN [ TRANSACTION ]  
[ ISOLATION LEVEL { SERIALIZABLE | REPEATABLE  
READ | READ COMMITTED | READ UNCOMMITTED } ] )
```

OPPURE

```
SET TRANSACTION ISOLATION LEVEL  
{ SERIALIZABLE | REPEATABLE READ | READ  
COMMITTED | READ UNCOMMITTED }
```

SPIEGO READ COMMITTED È IL PIÙ ADEGUATO

Controllo della Concorrenza

- PROTOCOLLO DI GESTIONE DELLA CONCORRENZA **Multiverso**: QUANDO UN ELEMENTO VIENE SCRITTO DA UNA TRANSAZIONE, TENGONO TRACCA DEL SUO VECCHIO VALORE

↳ LE VENGONO MANTENUTE VERSIONI MULTIPLE DELL'ELEMENTO

- IN QUESTO MODO ALCUNE OPERAZIONI IN LETTURA CHE VERREBBERO MESSE IN ATTESA DAL 2PL STRONG SEDIANO INVECE ESSERE ESEGUITE SIEGUENDO LA VERSONE OPPERA

↳ MINOR NUMERO DI TRANSAZIONI IN ATTESA A CAUSA DI OPERAZIONI DI LETTURA

AUMENTO DEL COSTO DI GESTIONE DELLA CONCORRENZA DAVUTO ALLA NECESSITÀ DI GESTIRE LE VERSIONI

→ 2PL Multiverso

- UNA TRANSAZIONE DESIRE IL LOCK X PER IL SCRUTUM, Mentre le altre leggono la versione corrente precedente → lettura e scrutum non solo in conflitto

- QUANDO UNA TRANSAZIONE VOLLE EFFETTUARE IL COMMIT, DEVE OTTENERE IL CERTIFY LOCK SU OGNI ELEMENTO SU CUI HA LOCK IN SCRUTUM

↳ NUOVA VERSIONE COMMITTED CHE VERRÀ VISTA DA TRANSAZIONI CHE VOLGANO LEGGERE

	S	X
S	OK	NO
X	NO	NO

Matrice di compatibilità 2PL

	S	X	C
S	OK	OK	NO
X	OK	NO	NO
C	NO	NO	NO

Matrice di compatibilità 2PL multiverso

- MAGGIOR GRADO DI CONCORRENZA, MA THROTTLING MINORE A CAUSA DEL COSTO DI GESTIONE

- POSSONO GENERARE DEADLOCK

VARIANTE DEL CONTROLLO DELLA CONCERNZA MULTIVISUALIZZARE 2PL → **SNAPSHOT ISOLATION**

- LE TRANSAZIONI NON DEVONO CHIEDERE LOCK PER LA LETTURA
- LE SCRITTURE VENGONO GESTITE IN ACCORDO AL 2PL STRONG

LOCK ESPATI IN SQL

→ LA RICHIESTA DI LOCK E' IMPLICITA

SE UNA TRANSAZIONE SA DI DOVER ELABORARE MOLTE TUPLE DI UNA RELAZIONE PUÒ PARTECIPARE ESPATIAMENTE DI POSSERE UN LOCK (SHARE O EXCLUSIVE MODE) SULL'INTERA RELAZIONE

```
LOCK TABLE Studenti IN SHARE MODE ;  
SELECT *  
FROM Studenti  
WHERE DataNascita < '11/07/1982' ;
```

LOCK SU RELAZIONI

→ SERVONO POCCHI LOCK

→ BASSO GRADO DI CONCERNZA

LOCK SU TUPLE

→ SERVONO PIÙ LOCK

→ MAGGIOR GRADO DI CONCERNZA

LOCK ESCALATION. SE AD UN CERTO LIVELLO DI GRANULARITÀ IL NUMERO DI LOCK E' ELEVATO, IL DBMS ADESSO AUMENTA IL LIVELLO DI GRANULARITÀ PER ABBASSARE IL NUMERO DI LOCK.

↳ SI RIDUCE LA CONCERNZA E AUMENTA IL RISCHIO DI DEADLOCKS

GESTIONE DEL RIPURTINO

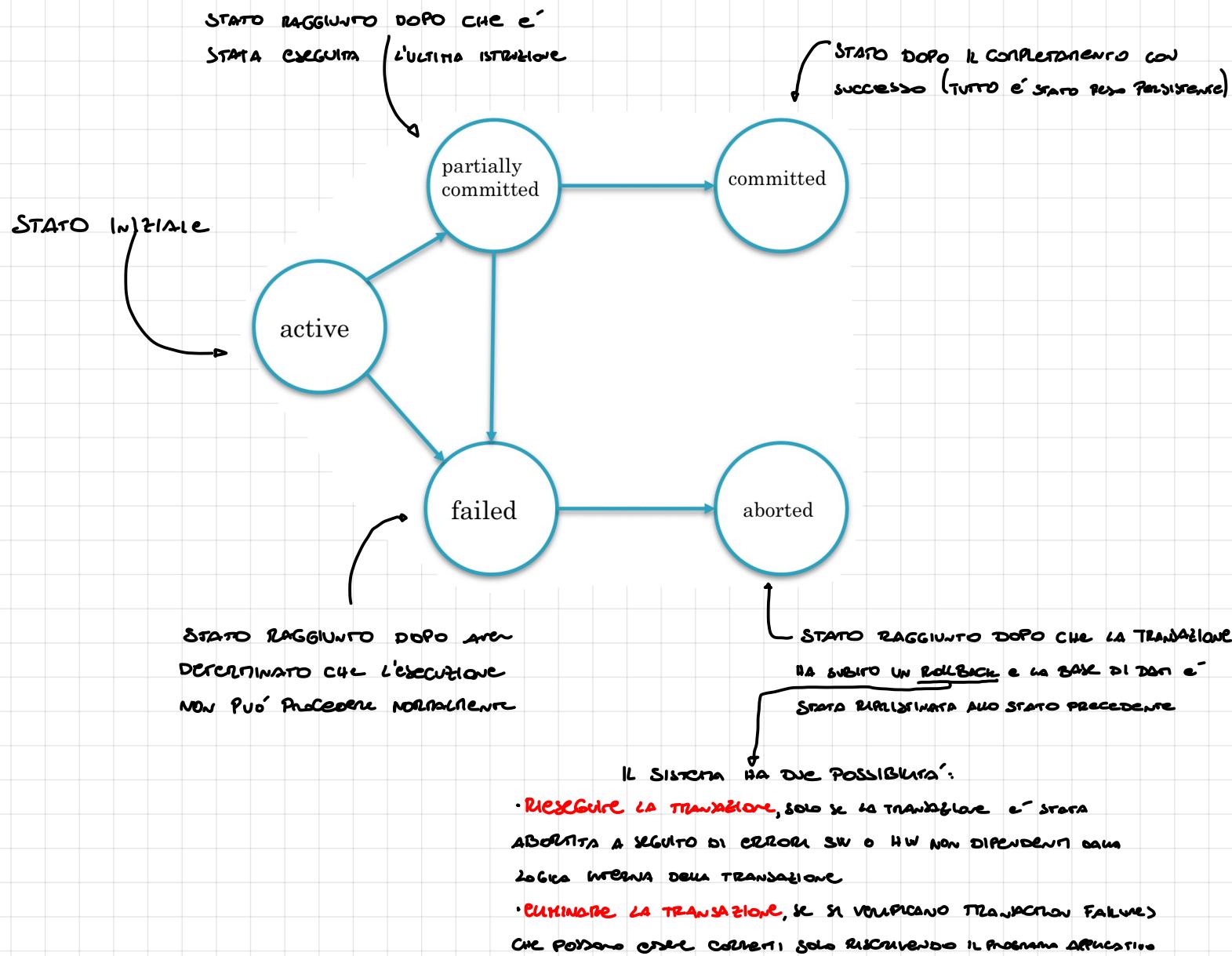
GUERRA DUE TRANSAZIONI COMMITTED DEVONO ESSERE PERMANENTI (DURABILITY) E GUERRA DUE TRANSAZIONI ABORTED NON DEVONO LASCIARE TRACCE (ATOMICITY)

ASSICURATI DAL GESTORE DEL RIPURTINO

→ DEVE IDENTIFICARE I MALFUNZIONAMENTI E RIPURTINARE LA BASE DI DATI ALLO STATO PRECEDENTE IL MALFUNZIONAMENTO

- **TRANSACTION FAILURE**: QUANDO UNA TRANSAZIONE ABORTE, GLI OPERI DEL MANAGGERE SUL DB DEVONO ESSERE ANNULLATI.
- **SYSTEM FAILURE**: GUARDO HW O SW CHE PROVOCÀ L'INTERROGAZIONE DI TUTTE LE TRANSAZIONI IN EXECUTION, SENZA DANNEGGIARE LA MEMORIA PERMANENTE → SPESO DOVUTO A PROBLEMI DI MEMORIA VOLATILE (cache)
- **HARD FAILURE**: CONTENUTO PERSISTENTE DEL DB VIENE DANNEGGIATO

- Memoria Volatile \rightarrow MM e Cache
- Memoria Non Volatile \rightarrow Disco e nastri magnetici
- Memoria Stabile \rightarrow Non Possono Errore perché le informazioni contengono informazioni duplicate in diverse memorie non volatile con probabilità di fallimento indipendente



Le attività di ripristino eseguite a valle del verificarsi di un transaction o di un system failure vengono genericamente indicate con il termine **RIPRESA A CADDO**

→ Recovery con Log

- Durante l'esecuzione di una transazione tutte le operazioni di scrittura sono registrate in un parziale file gestito dal sistema \rightarrow **FIRE DI LOG**

\hookrightarrow Memorizzato sulla memoria stabile

\hookrightarrow Il sistema con il log può gestire system e transaction failure

I log permettono di eseguire operazioni di REDO / UNDO di transazioni in caso di failure

- Se una pagina P del DB viene modificata dalla transazione T, il Log contiene un record del tipo

(LSN, T, PID, before(P), after(P), prevLSN)

- LSN = Log Sequence Number (n. progressivo del record nel Log)
- T = identificatore della transazione
- PID = identificatore della pagina modificata
- before(P) = è la cosiddetta "before image" di P, ovvero il contenuto di P prima della modifica
- after(P) = è l'"after image" di P, ossia il contenuto di P dopo la modifica
- prevLSN = LSN del precedente record del Log relativo a T

PROTOCOLLO WRITE-AHEAD LOGGING WAL

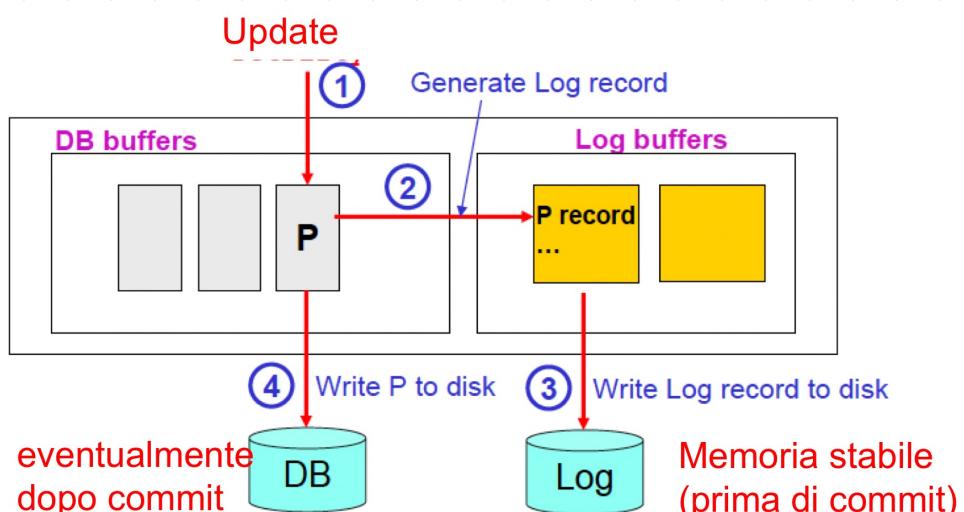
PRIMA DI SCRIVERE SU DISCO UNA PAGINA MODIFICATA, IL LOG RECORD CORRISPONDENTE DEVE ESSERE GIÀ SCRITTO NEL LOG

→ IL RECORD DI LOG DEVE ESSERE FORZATO SU MEMORIA STABILE PRIMA CHE LA PAGINA DIA ARRETI SU DISCO
• ATOMICITÀ GARANTITA

→ TUTTI I RECORD DI LOG DI UNA TRANSAZIONE DEVONO ESSERE FORZATI PRIMA DEL COMMIT
• PERSISTENZA GARANTITA

DI ESEGUIRE IL PROTOCOLLO E GARANTIRLO C'È CARICO DEL BUFFER MANAGER

↳ GESTISCE I BUFFER DEL DB E DEL LOG



GESTIONE DEL BUFFER : QUANDO UNA TRANSAZIONE MODIFICA UNA PAGINA, IL BUFFER MANAGER HA 2 POSSIBILITÀ .

• **POLITICA NO STEAL** Maneggiare la pagina nel buffer e attendere che T ABbia eseguito COMMIT prima di scriverla su disco

• **POLITICA STEAL** Scrivere la pagina quando più conviene, anche prima della terminazione di T
↳ per necessitare di liberare spazio nei buffer oppure per ottimizzare le prestazioni di I/O

Lo c'è il rischio di esaurire lo spazio a disposizione in memoria centrale, poiché una transazione non può rubare memoria buffer ad altre transazioni

ESECUZIONE DEL COMMIT

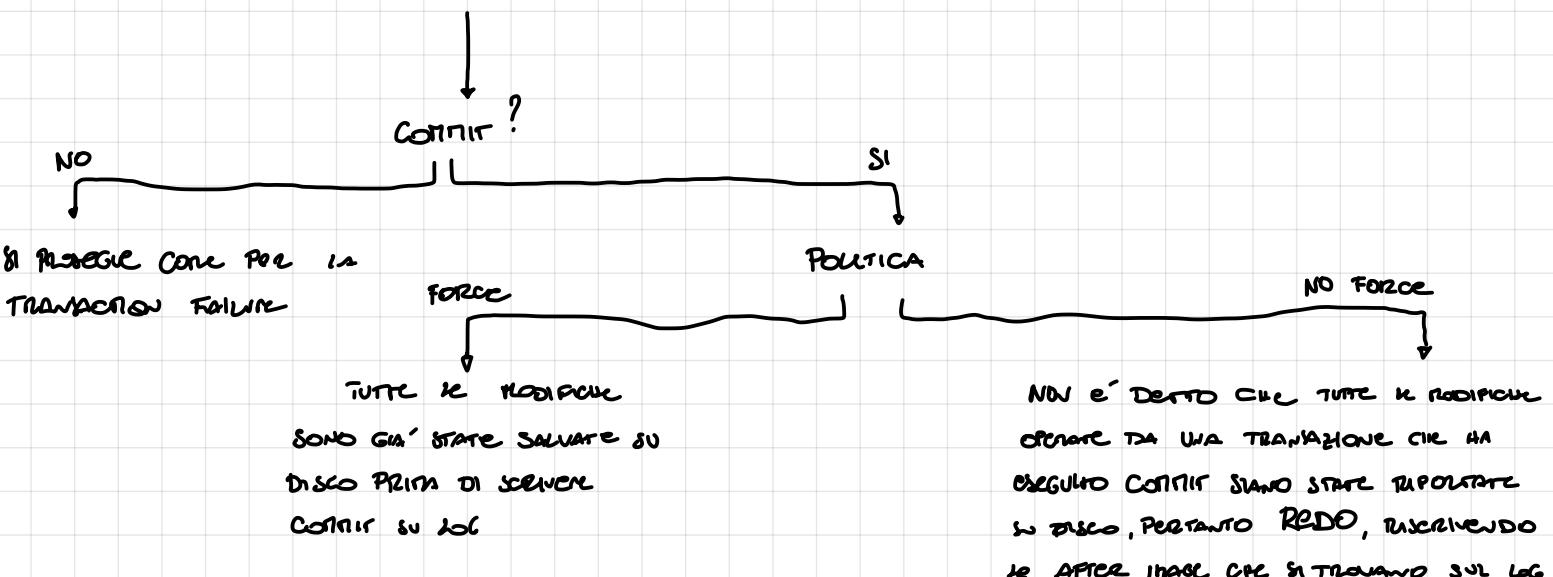
- **POLITICA FORCE** Si forza la scrittura di tutte le pagine modificate dalla transazione prima di scrivere il record di commit sul log **può generare molti I/O inutili**

- **POLITICA NO FORCE** Si scrive subito il record di commit sul log → alcune modifiche della transazione possono non essere ancora state真的很 persistenti sul disco

TRANSACTION FAILURE

- **STEAL**: si scandisce il log a ritroso e si ripristinano nel DB le **before image** delle pagine modificate da T, poiché al momento dell'aborto alcune pagine sono già state scritte su disco
- **NO STEAL**: non ci sono operazioni da eseguire, nel suono blocco è stato aggiornato su disco

SISTEM FAILURE



- **STEAL**: → log a modifiche immediate
→ necessaria della procedura UNDO

- **NO STEAL**: → log a modifiche differite
→ gli aggiornamenti non dovranno mai essere disfatti

- **NO FORCE**: → serve procedura di redo

- **Force**: → non serve procedura di redo poiché al commit i dati saranno già tutti salvati su disco (e se una transazione non c'è arrivata al commit deve solo essere disfatta)

LA POLITICA PIÙ UTILIZZATA È LA STEAL POICHÉ NON RICHIESTE DI MANTENERE NEL BUFFER NECESSARIAMENTE TUTTI I BLOCCHI MODIFICATI DA TRANSAZIONI CHE NON HANNO EFFETTUATO IL COMMIT

→ **STEAL - NO FORCE** COMBINAZIONE PIÙ COMUNE, MA HA PURÒ LO SVANTAGGIO DI ESEGUIRE UNA PROCEDURA DI RECOVERY PER OGNI TRANSAZIONE ESEGUITA DURANTE LA FORCE DURA DI RESTART IN CASO DI FAILURE

- ↳ UNDO SE NON È ARRIVATA AL COMMIT
- ↳ REDO SE È ARRIVATA AL COMMIT

CHECKPOINT

LA PROCEDURA DI RESTART È QUELLA CHE SI OCCUPA DI RIPORTARE IL DB IN UNO STATO CONSISTENTE A FRONTE DI SYSTEM FAILURE; PER RIDURRE I TEMPI DI RESTART PERIODICAMENTE SI PUÒ ESEGUIRE UN "CHECKPOINT" OVVERO UNA SCRITTURA FORZATA SU DISCO DELLE PAGINE MODIFICATE

→ REGISTRATA SCRIVENDO UN RECORD CKP SUL LOG

IN QUESTO MODO SE T HA ESEGUITO COMMIT PRIMA DI CHECKPOINT N C'È CERTO CHE NON DESSA CHERE RELEASED → SALVATI GLI EFFETTI DI T IN MODO PERMANENTE NEL BD E I BLOCCHI BUFFERED POSSONO CHERE RELEASED

IL SISTEMA PERIODICAMENTE AL CHECKPOINT :

- FORZA LE PAGINE DI LOG PRESENTI IN MEMORIA PRINCIPALE SU MEMORIA STABILE
- FORZA TUTTE LE PAGINE DATI DI BUFFER SUL DISCO
- FORZA IL RECORD <CKP> SUL LOG IN MEMORIA STABILE

MEDIA FAILURE

LA PROCEDURA DI RESTART VIENE DETTA RIPRESA A FREDDO

→ **DUMP**: COPIA COMPLETA DELLA BASE DI DATI, NORMALMENTE CREATA QUANDO IL SISTEMA NON È OPERATIVO

BACKUP: COPIA PARZIALE IN MEMORIA STABILE

→ UN RECORD <DUMP> NEL LOG SEGNALA LA PRESENZA DI UN BACKUP ESEGUITO IN UN CERTO MOMENTO ED IDENTIFICA IL FILE O IL DEVICE SU CUI È STATO ESEGUITO IL DUMP

RIPRESA A FREDDO, ACCEDO AL DUMP AL RESTART E SI RIPRISTINA IL CONTENUTO DELLA BASE DI DATI SU MEMORIA NON VOLATILE

- ↳ SI ESEGUE UNA RIPRESA A CALDO: SI ACCADE AL FILE DI LOG E SI ESEGUE IL RIPRISTINO COME DISCUSSO IN PRECEDENZA

LINGUAGGIO SQL - DATI DERIVATI

c'è possibile creare una colonna con contenuto derivato

creare una relazione con schema e contenuto iniziale derivato

creare una relazione derivata (vista) → vista

COLONNE DERIVATE

ALTER TABLE Noleggio ADD COLUMN durata INTEGER

GENERATED ALWAYS AS ((dataRest-dataNol) DAY);

→ TUTTE le colonne che compaiono nell'espressione devono APPARTENERE ALLA RELAZIONE IN CUI VIENE DEFINITA LA COLONNA E NON POSSONO ESSERE CALCOLATE

RELAZIONI DERIVATE

→ VOGLIANO CREARE UNA NUOVA RELAZIONE ClienteV CON LO STESSO SCHEMA DI Cliente PIÙ UNA COLONNA Bonus

CREATE TABLE ClienteV

(**LIKE** Cliente,

bonus NUMERIC(4,2));

LA NUOVA TABEla È VUOTA E NON HA CONNESSIONI PERMANENTI A QUELLI IN CUI È BANATA

→ NESSUNA RELAZIONE FRA LE ISTANZE

→ NESSUNA PROPAGAZIONE DI MODIFICA

• COPIA IN ClienteV LA STRUTTURA COMPLETA (NON COLONNE, DOMINI E VINCOLI DI OBBLIGATORIETÀ) DI Cliente

{ NON VENGONO COPIATI I VINCOLI GENERALI
EVENTUALI DATI NON VENGONO COPIATI

RELAZIONI DERIVATE DA QUERY

→ DECIDIAMO DI MATERIALIZZARE IN DUE RELAZIONI SEPARATE I NOLEGGI CONCLUSI ED I NOLEGGI IN CORSO

CREATE TABLE NoleggioA **AS**

SELECT colloc, dataNol, codCli

FROM Noleggio

WHERE dataRest IS NULL

WITH NO DATA;

→ CREA UNA RELAZIONE DERIVATA VUOTA

CREATE TABLE NoleggioC **AS**

SELECT * FROM Noleggio

WHERE dataRest IS NOT NULL

WITH DATA;

→ RELAZIONE DERIVATA E POPOLATA CON LE TUPLE RISULTATO DELLA VALORIZZAZIONE DELL'INTERROGAZIONE

→ USO L'INTERROGAZIONE SOLO PER LA CREAZIONE E PER LA POPOLAZIONE INIZIALE DELLA RELAZIONE

→ MODIFICHE ALLE TABELLE USATE NELLA QUERY NON SI PROPAGANO ALLA NUOVA TABEla

VISTE

Sono un meccanismo per legare l'istanza e la query che la definisce visualmente.

→ SUPponiamo che un nuovo solo della videoteca ABbia necessita' di gestire solo i noleggi dei clienti over 65

CREATE TABLE NoleggioOver65 AS

SELECT *

FROM Noleggio NATURAL JOIN Cliente
WHERE (CURRENT_DATE - dataN) YEAR >= 65

WITH DATA;

Così facendo però si aggiungono nuovi clienti over 65 in Noleggio, la relazione non la contiene e si aggiunge una tupla a NoleggioOver65, la tupla non viene invece inserita in Noleggio
↓

DENO QUNDI Creare una vista

↳ c'è una relazione virtuale che permette di accedere diversamente ai dati
realizzando delle relazioni di base e si definisce trarre un'interrogazione
in relazioni di base e Viste.
NON CORRISPONDE A DATI REALIZZATI secondo il suo schema ma al risultato
dell'interrogazione e può essere vista a suon torn gli effetti come una
relazione di base

- Le Viste sono una PEr:
- semplificare l'accesso ai dati
 - fornire indipendenza logica
 - garantire la privacità dei dati

CREATE VIEW Over65 AS

SELECT *
FROM Noleggio NATURAL JOIN Cliente
WHERE (CURRENT_DATE - dataN) YEAR >= 65

↳ posso usare la vista come normale relazione →

SELECT * FROM Over65

CREATE VIEW <Nome Vista> [<Lista nomi colonne>]

AS <Interrogazione>

[WITH [{Local | cascaded}] CHECK OPTION]

CREATE VIEW Noleggio AS
SELECT codCli, dataNol, colloc
FROM Noleggio
WHERE dataRest IS NULL AND
(CURRENT_DATE - dataNol) DAY
> INTERVAL '3' DAY;

CREATE VIEW Noleggio
(codiceCliente, dataNoleggio, Video)

AS
SELECT codCli, dataNol, colloc
FROM Noleggio
WHERE dataRest IS NULL AND
(CURRENT_DATE - dataNol) DAY
> INTERVAL '3' DAY;

Le colonne della vista
corrispondono in numero e ordine
a avere una clausola di proiezione
dell'interrogazione

I nomi delle colonne della vista sono
codCli, dataNol e colloc

I nomi delle colonne della vista sono
codiceCliente, dataNoleggio, Video

Per semplificare l'accesso agli utenti che lavorano spesso con particolari associazioni fra entità

- Una sola tabella invece di un gran effluvio
- Si possono preferire eventuali dati superflui

Per astrarre i dati e presentarli agli utenti in modo agevole

- Garantisce riservatezza

Su una vista si possono eseguire:

- Interrogazioni
- Inserimenti
- Aggiornamenti

Interrogazioni su viste

Su una vista è possibile:

- Effettuare proiezioni
- Specificare condizioni di terza
- Effettuare dei join con altre relazioni o viste
- Effettuare raggruppamenti e calcolare funzioni di gruppo
- Definire altre viste

NON È AMMESO L'USO DI FUNZIONI DI GRUPPO SU COLONNE DI VISTE CHE SONO A LORO VOLTA DEFINITE TRAMITE FUNZIONI DI GRUPPO

→ PERÒ LA TRASPORTA POICHE' PER AVERE RESTRIZIONE L'UTENTE DEVE ESSERE CONSCIO DI STARE USANDO UNA VISTA

Aggiornamenti su viste

Se possibile, l'esecuzione di un'operazione di aggiornamento su una vista viene propagata alla relazione su cui la vista è definita

DELETE SU VISTE · RESTRIZIONI

È POSSIBILE ESEGUIRE IL COMANDO DELETE SU V SE LA QUERY DI DEFINIZIONE Q

- C'è UNA SINGOLA RELAZIONE R
- NON CONTIENE
 - GROUP BY
 - DISTINCT
 - OPERATORI INSIEMISTICI
 - FUNZIONI DI GRUPPO
- EVENTUALI SOTTOINTERROGAZIONI IN Q NON FANNO RIFERIMENTO AD R

UPDATE SU VISTE: RESTRIZIONI

È POSSIBILE ESEGUIRE IL COMANDO UPDATE

- LA QUERY DI DEFINIZIONE Q SODDISFA TUTTE LE RICHIESTE PER IL DELETE
- C'È UNA DEFINITA TRATTIRE UN'ESPRESSIONE O FUNZIONE

INSERT IN VISTE: RESTRIZIONI

È POSSIBILE ESEGUIRE IL COMANDO INSERT IN V SE

- LA QUERY DI DEFINIZIONE Q SODDISFA TUTTE LE RESTRIZIONI RICHIESTE PER IL DELETE
- TUTTE LE COLONNE DI V SODDISFANO LE RESTRIZIONI RICHIESTE PER L'UPDATE
- TUTTE LE COLONNE DI R SU CUI VALE IL VINCOLO DI NOT NULL E PER CUI NON È SPECIFICATA VALORE DI DEFAULT SONO INCLUSE IN V

Check option

Per assicurare che le tuple siano inserite / modificare tramite una vista solo se verificano la condizione nella sua interrogazione di definizione, si usa la clausola **Check option** del comando Create view

La Check option può essere specificata con due possibili alternative:

- LOCAL
- CASCADING → DEFAULT

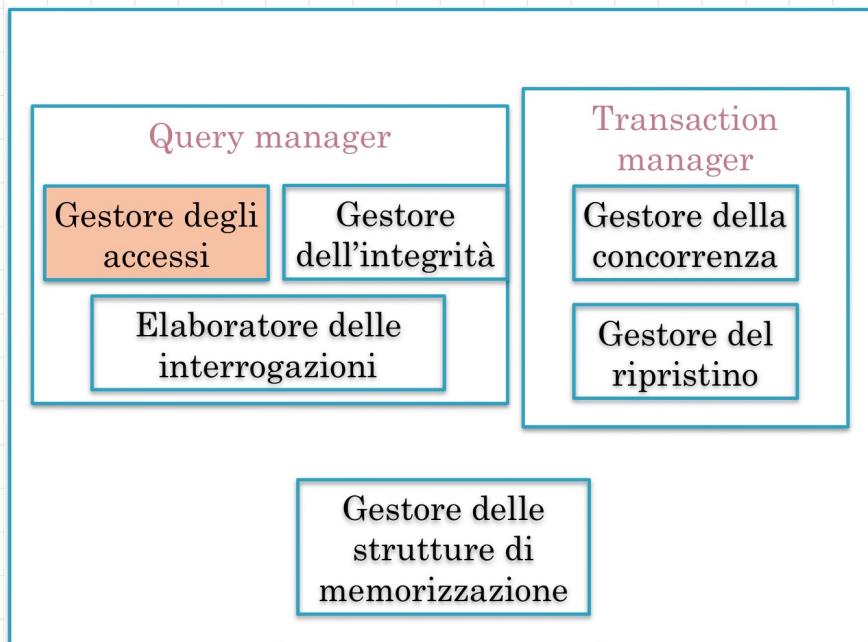
La differenza tra Local e Cascading è rilevante nei casi in cui una vista è definita in termini di un'altra vista.

Sia V1 una vista definita in termini di un'altra vista V2

- Se V1 è definita WITH LOCAL CHECK OPTION, gli inserimenti eseguiti su V1 devono verificare
 - L'interrogazione di definizione di V1
 - L'interrogazione di definizione di V2 solo se V2 è a sua volta definita con Check option
- Se V1 è definita WITH CASCADING CHECK OPTION, gli inserimenti eseguiti su V1 devono verificare
 - L'interrogazione di definizione di V1
 - L'interrogazione di definizione di V2, indipendentemente dal fatto che V2 sia definita o meno con Check option

Controllo dell'accesso

COMPONENTI DBMS



Il controllo dell'accesso regola le operazioni che si possono compiere sui dati in una base di dati, allo scopo di:

- Limitare e controllare le operazioni che gli utenti effettuano
- Prevenire azioni accidentali o deleziose che potessero compromettere la coerenza e la sicurezza dei dati

Il DBA fornisce gli strumenti per realizzare le protezioni

Lo DBA (Amministratore del DB)

Il DBA infatti ha il compito di conferire agli utenti i giusti privilegi tramite particolari comandi SQL

→ **DATA CONTROL LANGUAGE** estende l'SQL con nuove colonne + gestione privilegi

Il controllo dell'accesso si basa sulla specifica di Politiche di Sicurezza

• BASATA SU 3 ENTI FONDAMENTALI:

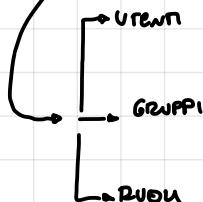
Qualsiasi componente di base dello schermo (utente, utile, ecc..)

• OGGETTO → UTILE A VOGLIANO GARANTIRE PROTEZIONE

• SOGGETTO → ENTITÀ ATTIVA (effettua richiesta per accedere ad un oggetto)

• PRIVILEGI → DETERMINANO LE OPERAZIONI CHE I SOGGETTI POSSONO FAR SUGLI OGGETTI

Lettura, Scrittura, Aggiornamento, Eliminazione...



POLITICHE DI SICUREZZA TRADIZIONALI E AUTOMATIZZATI CHE STABILISCONO SPECIFICI DIRITTI CHE I VARI SOGGETTI POSSANO ESEGUIRE SUGLI OGGETTI

→ RAPPRESENTATE MEDIANTE UNA TUPLA (S, O, P)

**Controllo dell'accesso
DISCREZIONALE**

S	O	P
o	c	R
e	c	v
r	r	l
t	o	c
o	c	i
		g

Reference Monitor - Gestore degli accessi

→ INTERCETTA OGNI CONSOLO INVIATO AL DBMS

→ Controlla se il soggetto è autorizzato

→ La concessione può essere

→ TOTALE: Accesso sicuro può essere totalmente eseguito

→ PARZIALE: Solo parte dell'accesso può essere eseguito

→ NEGATA: L'accesso richiesto non può essere eseguito



DBA
Politiche
di sicurezza



Gestore degli accessi

Accesso totalmente o
parzialmente
accordato

Accesso negato



utente

PRINCIPI GENERALI

Il Modello realizza una politica di tipo discrezionale adottando il **paradigma di sistema aperto**

↳ Accesso consentito solo se esiste un'esplicita autorizzazione per esso

L'amministrazione dei privilegi è **decentralizzata mediante ownership**

Lo s'utente che crea una relazione, riceve tutti i privilegi su di essa ed anche la possibilità di delegare ad altri suoi privilegi

GRANT OPTION

Il soggetto che lo riceve può concederlo ad altri

- SQL :
- **GRANT** → inserisce una nuova autorizzazione nel sistema
 - **REVOKE**

GRANT {<lista privilegi>} | ALL PRIVILEGES p

ON <nome oggetto> o

TO {<lista utenti> | PUBLIC} s

[WITH GRANT OPTION];

↳ Consente la delega

dell'autorizzazione sui
privilegi

↳ PUBLIC Consente di specificare le autorizzazioni implicite dal comando per tutti gli utenti del sistema

PRIVILEGI SPECIALI

- INSERT
- UPDATE Possono essere concessi selettivamente solo su alcune colonne di una relazione
- DELETE
- SELECT

Il DBA possiede tutti i privilegi su tutte le risorse

PRIVILEGI

- ↳ Delegabili
- ↳ Non delegabili

REVOKE

Un utente può revocare solo i privilegi da lui stesso concesso

c'è possibile revocare più privilegi con un unico comando di revoke

In unico comando di revoke può essere utilizzato per revocare gli stessi privilegi sulla stessa relazione ad utenti diversi

REVOKE [GRANT OPTION FOR] <lista privilegi>
 ON <nome oggetto>
 FROM <lista utenti>
 {RESTRICT | CASCADE};

Serve per revocare la sola GRANT OPTION,
 mantenendo il diritto ad esercitare i privilegi
 oggetto del comando di revoca

LA REVOCÀ PUÒ ESSERE RICHIESTA CON O SENZA CASCATA:

- **SENZA CASCATA (RESTRICT)** l'esecuzione non viene concessa se questo comporta la revoca di altri privilegi - DEFAULT
- **CON CASCATA (CASCADE)** revoca tutti i privilegi che erano stati propagati, generando una reazione a catena ed eventuali elementi della base di dati che erano stati creati sfruttando questi privilegi

GESTIONE DELL'ACCESSO

INFORMAZIONI SULLE AUTORIZZAZIONI PRESENTI NEL SISTEMA SONO RICORDATE IN CATALOGHI DEL SISTEMA
 OGNI SISTEMA HA UN PROPRIO SCHEMA PER LE TABELE

IL GESTORE DELL'ACCESSO MODIFICA IL CONTENUTO DEI CATALOGHI DURANTE L'ESECUZIONE DI GRANT O REVOCARE
 LO SVOLGIMENTO TUTTE AUTORIZZAZIONI

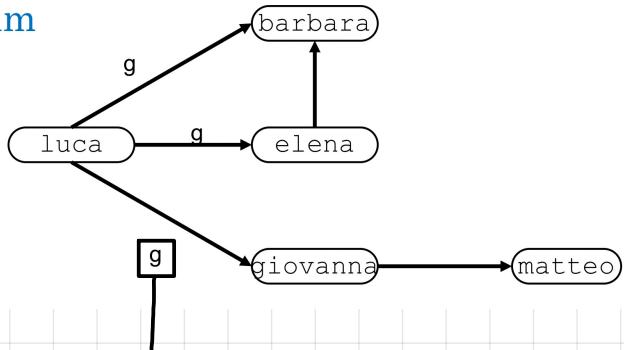
DURANTE L'ESECUZIONE DI ALTRI COMANDI SQL LEGGE IL CONTENUTO DEI CATALOGHI PER
 CONTROLLARE SE IL COMANDO PUÒ ESSERE ESEGUITO

Le informazioni contenute nei cataloghi possono essere rappresentate in astratto come un insieme di grafi

GRAFI DELLE AUTORIZZAZIONI

- ESISTE UN GRAFO PER OGNI PRIVILEGIO SU UNA CERTA TABELE

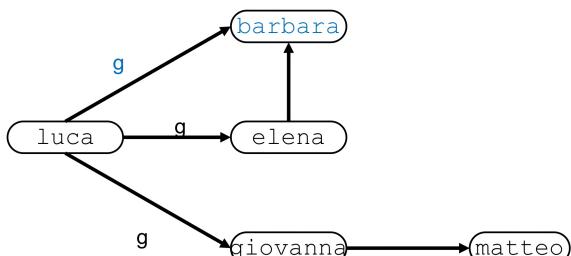
Grafo delle autorizzazioni relativo al privilegio **SELECT** e alla relazione **Film**



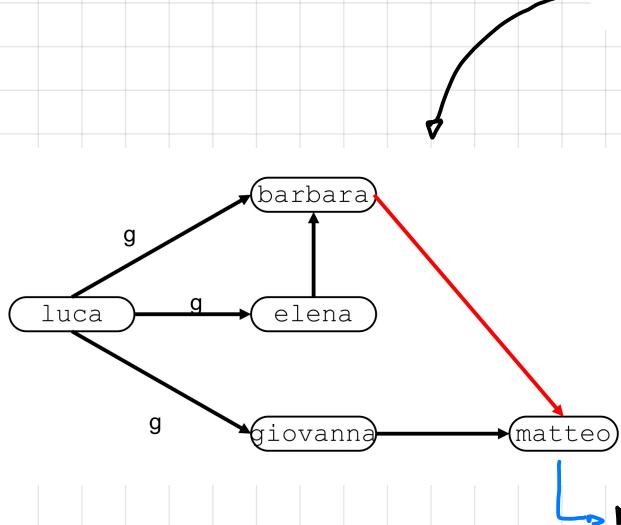
L'arco è etichettato con la lettera **g** se il privilegio è delegabile (concesso con GRANT OPTION)

DURANTE L'ESECUZIONE DI UN COMANDO GRANT, IL GESTORE DELL'ACCESSO:

- VERIFICA CHE L'UTENTE CHE ESEGUE IL COMANDO ABbia I DIRITTI PER POTERLO ESEGUIRE, LEGGENDO I CATALOGhi → COMANDO SELECT
- SE L'UTENTE PUO' ESEGIRE GRANT, I NUovi PRIVILEGI CONCERNENTI VENGONO INSERITI NEI CATALOGhi (INSERT), MODIFICANDOUNI
- IL COMANDO IN ALGUNI CASI PUO' ESSERE ESEGUITO SOLO PARZIALMENTE, INSERENDO SOLI SOLO ALCUNE AUTORIZZAZIONI



barbara: GRANT select ON Film TO matteo;



Matteo now ha acquisito il privilegio con Grant option
→ now puo' delegarlo

DURANTE L'ESECUZIONE DI UN COMANDO REVOKE, IL GESTORE DELL'ACCESSO:

- LEGGE I CATALOGhi PER VERIFICARE SE L'UTENTE DEBBE I DIRITTI PER POTER ESEGIRE IL COMANDO (SELECT)
- SE PUO' ESEGUIRSI, LE AUTORIZZAZIONI VENGONO CANCELLATE DAL CATALOGhO (DELETE o UPDATE)
- IL COMANDO IN ALGUN CASO PUO' ESSERE ESEGUITO PARZIALMENTE
 - SOLO ALCUNE TRA LE AUTORIZZAZIONI RECETTE VENGONO MODIFICATE O CANCELLATE
 - DIPENDE ANCHE DALLA MODALITA' DI REVOKE (PERMUTE o CASCADE)

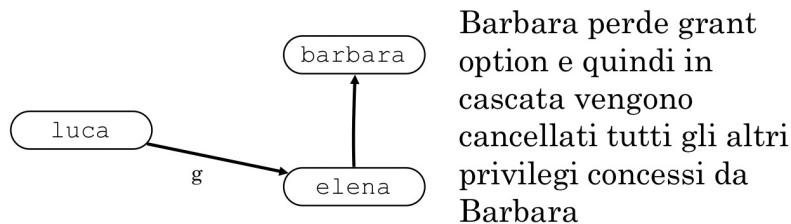
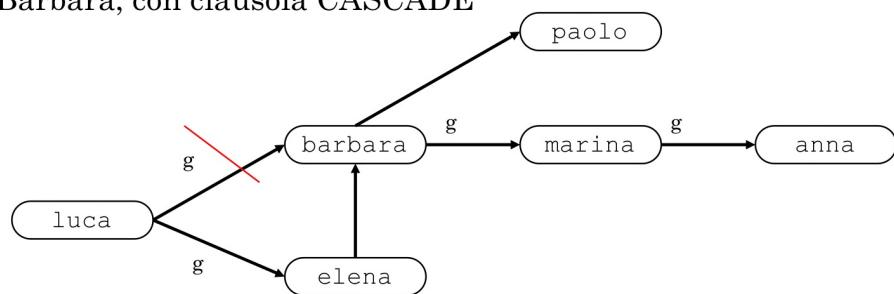
SI PUO' RINNOVARE L'ARCO CON RESTRICT IC: IL NODO SUCCESSIVO NON HA IL PRIVILEGIO DI DEREGLI OPPURE LO HA, MA NON LO HA CONCESSO

ALTRIMENTI SCHEDE CASCADE

ATTENZIONE: • UN UTENTE POTREBBE AVER ACQUISITO IL PRIVILEGIO DA PARTE DI PIÙ' UTENTI E PUÒ QUINDI ESEGUIRE LA REVOCÀ ANCHE DOPO

Supponiamo che Luca revochi il privilegio select concesso a Barbara, con clausola CASCADE

REVOCÀ IN CASCATA



Barbara perde grant option e quindi in cascata vengono cancellati tutti gli altri privilegi concessi da Barbara

CONTROLLO DEI' ACCESSO BASATO SUL RUOLO

RBAC Role-Based Access Control

UN RUOLO RAPPRESENTA UNA FUNZIONE CHE GLI UTENTI RICOPRANO ALL'INTERNO DELLA REALTA' ORGANIZZATIVA IN CUI OPERANO

GLI UTENTI SONO ABILITATI A RICOPRIRE UNO O PIÙ RUOLO IN BASE ALLE ATTIVITA' CHE DEVONO SVOLGERE

- I PRIVILEGI POSSONO ESSERE CONCESSI AI SINGOLI UTENTI, MA ANCHE AI RUOLO
- OGNI UTENTE CHE RICOPRE UN RUOLO ACCORDA TUTTE LE AUTORIZZAZIONI AD ELLO CONFERITE

QUESTO COMPORTA

{
PIÙ' FLESSIBILITÀ DA PARTE DI CONTROLLO DEI' ACCESSI
POSSIBILITÀ CHE UN UTENTE RICOPRA RUOLO DIVERSI IN DIVERSI PIANI
ATTIVITÀ DI AMMINISTRAZIONE SEMPLIFICATA
FACILITÀ NELLA DEFINIZIONE DEL PROFILO DI NUOVI UTENTI}

CREAZIONE DI RUOLO

- CREATE ROLE <Nome Ruolo>
- DROP ROLE <Nome Ruolo>

UN UTENTE IN PIEMONTE DIVERSI PUÒ' RICOPRIRE RUOLO DIVERSI

↳ ASSOCIAZIONE DINAMICA DI UN RUOLO ALL'UTENTE DURA SESSIONE ATTIVA

- SET ROLE <Nome Ruolo>

COMANDO GRANT ESTESO AI RUOLI

GRANT MI PERMETTE DI CONCEDERE PRIVILEGI ANCHE AI RUOLI

DI AUTORIZZARE UN UTENTE A RACOPPIRE UNO O PIÙ RUOLI

DI DEFINIRE GERARCHIE TRA I RUOLI, AUTORIZZANDO UN RUOLO A RACOPPIARE UN ALTRO

GRANT {<lista privilegi> | ALL PRIVILEGES}

ON <nome oggetto>

TO {<lista utenti> | <lista ruoli> | PUBLIC}

[WITH GRANT OPTION];

CONSEGNA DI SPECIFICARE RUOLI
Gli utenti/ruoli del sistema

• CONCESSIONE PRIVILEGI AI RUOLI

CREATE ROLE direttoreVideoteca;

GRANT delete ON Clienti TO **direttoreVideoteca**
WITH GRANT OPTION;

• AUTORIZZARE UN UTENTE A RACOPPIARE UN RUOLO

GRANT <lista ruoli concessi>

TO {<lista utenti> | PUBLIC}

[WITH ADMIN OPTION];

ANALOGO PER I RUOLI DELLA GRANT OPTION

- VENGONO EREDITATE TUTTE LE AUTORIZZAZIONI SPECIFICATE PER QUESTO RUOLO
- È ANCHE POSSIBILE CONCEDERE A QUESTO RUOLO LA POSSIBILITÀ DI RACOPPIARE UN RUOLO

GERARCHIA TRA RUOLI

SOL PERMETTE DI DEFINIRE RUOLI GERARCHICAMENTE

UNA GERARCHIA SUI RUOLI DEFINISCE UN ORDINAMENTO PARZIALE \geq TRA DI essi

- INDUCE UN'EREDITARIETÀ DEI PRIVILEGI TRA RUOLI NELLA GERARCHIA
- STABILISCE UNA RELAZIONE TRA GLI UTENTI ABILITATI A RACOPPIARE I VARI RUOLI

SE R1 \geq R2 POSSANO ANCHE DIRE CHE GLI UTENTI CON RUOLO R1 SONO ABILITATI A RICEVERE ANCHE IL RUOLO R2

• GERARCHIE TRA RUOLI

GRANT <lista ruoli concessi>

TO {<lista ruoli> | PUBLIC}

[WITH ADMIN OPTION];

CREATE ROLE direttoreVideoteca;

CREATE ROLE commesso;

GRANT commesso to direttoreVideoteca;

Quindi $\text{direttoreVideoteca} \geq \text{commesso}$

IL COMANDO REVOKE E' ESTESO AI RUOLI

- REVOCA DEI PRIVILEGI AI RUOLI
- REVOCA DI RUOLI ED ERINNAGLIO RELAZIONI GERARCHICHE
- REVOCA DI PRIVILEGI A RUOLI

REVOKE [GRANT OPTION FOR] <lista privilegi>

ON <nome oggetto>

FROM {<lista utenti> | <lista ruoli>}

{RESTRICT | CASCADE};

REVOKE delete ON Clienti FROM
direttoreVideoteca ;

• REVOCA DI RUOLI A UTENTI O RUOLI

REVOKE [ADMIN OPTION FOR] <lista ruoli revocati>

FROM {<lista utenti> | <lista ruoli>}

{RESTRICT | CASCADE};

AUTORIZZAZIONI SU VISTE

LA SINTASSI DEL COMANDO GRANT NON CONSENTE DI CONCEDERE PRIVILEGI SOLO SU ALCUNE TUPLE DI UNA TABELLA

↳ E' PERO' SUFFICIENTE DEFINIRE UNA VISTA CHE SELEZIONI LE TUPLE DI INTERESSE E AUTORIZZARE L'ACCESSO ALLA VISTA INVECE CHE ALLA RELAZIONE DI BASE

LE VISTE PERMETTONO DI REALIZZARE IL COSÌ DETTO **CONTROLLI DELL'ACCESSO IN BASE AL CONTENUTO**, OSSIA PERMETTONO DI AUTORIZZARE L'ACCESSO SOLO A SPECIFICHE TUPLE DI UNA RELAZIONE, SUCA BASE DEI VALORI DEI LORO ATTRIBUTI

- LA VISTA PUO' ESSERE CREATA SOLO SE L'UTENTE HA IL PRIVILEGIO SELECT SULLA RELAZIONE/VISTA SU CUI E' DEFINITA

- IL PROPRIETARIO DELLA VISTA HA UN'INTERSEZIONE DI AUTORIZZAZIONI: E' L'INTERSEZIONE DI

- ① LE AUTORIZZAZIONI CHE L'UTENTE POSSIDE SULLA RELAZIONE/VISTA SU CUI LA VISTA E' DEFINITA

- ② LA SEMANTICA DELLA VISTA, OSSIA LA SUA DEFINIZIONE IN TERMINI DELLE RELAZIONI O VISTE COMPONENTI

SE LA VISTA E' DEFINITA SU PIU' RELAZIONI O VISTE, I PRIVILEGI CHE L'UTENTE CHE CREA LA VISTA HA SU DI QUESTE SONO OTTENUTI INTERSECCANDO I PRIVILEGI CHE L'UTENTE HA SU TUTTE LE RELAZIONI/VISTE COMPOSTE CON LE OPERAZIONI ESEGUITIBILI SULLA VISTA

UN PRIVILEGIO SULLA VISTA E' DELEGABILE SOLO SE IL CREATORE DELLA VISTA HA IL DIRITTO DI DELEGARE TALE PRIVILEGIO SU TUTTE LE RELAZIONI O VISTE COMPOSTE

SQL PONE ALCUNE RESTRIZIONI SULLE OPERAZIONI CHE POSSONO ESSERE ESEGUITE SU UNA VISTA

↳ UNA VISTA CHE COMPUTA STATISTICHE NON PUO' ESSERE AGGIORNATA

↳ SI RIPIETONO ANCHE SULLE AUTORIZZAZIONI CHE UN UTENTE CHE CREA UNA VISTA HA SULLA VISTA STessa

Elena ha i privilegi SELECT, INSERT e UPDATE sulla relazione Noleggi, con GRANT OPTION

Elena esegue il comando:

```
CREATE VIEW NumNoleggi AS  
SELECT codCli, COUNT(*) AS NumNol  
FROM Noleggi  
GROUP BY codCli;
```

L'esecuzione del comando viene autorizzata e la vista creata

Privilegi esercitabili da Elena sulla vista

- (i) SELECT, INSERT, UPDATE
- (ii) SELECT
- L'intersezione contiene solo SELECT, quindi questo è l'unico privilegio di Elena sulla vista
- Elena può concedere a terzi tale privilegio, in quanto possiede GRANT OPTION

LA CONCESSIONE DI PRIVILEGI SU UNA VISTA E' RUOLO SISTEMA A OMESSO SU RELAZIONI DI BASE.

LE OPERAZIONI DI REVOCARE SONO INVECE PIU' COMPLICATE

↳ E' NECESSARIO STABILIRE COSA SUCCIDE AD UNA VISTA SE UN PRIVILEGIO SELECT SU UNA DELLE RELAZIONI O VISTE COMPONENTI E' REVOCATO

↳ SI CANCELLA LA VISTA SE IL PRIVILEGIO REVOCATO ERA L'UNICO UTILE PER LA SUA DEFINIZIONE

```
barbara>
CREATE VIEW Commedie AS
SELECT * FROM Film
WHERE genere = 'commedia';
```

```
elena>
CREATE VIEW NumNoleggi AS
SELECT codCli,
       COUNT(*) AS NumNol
FROM Noleggi
GROUP BY codCli;
```

(GUARDA GRAFI PRECEDENTI.)

- Se Luca revoca ad Elena (Barbara) il privilegio SELECT sulla relazione Noleggi e Elena (Barbara) non ha acquisito da altri utenti lo stesso privilegio, la vista NumNoleggi (Commedie) viene a sua volta cancellata
- In caso contrario la vista viene mantenuta

Sviluppo di applicazioni per basi di dati

LA MAGGIOR PARTE DEI DBMS SUPPORTA UN'ESTENSIONE PROCEDURALE DI SQL

I PROGRAMMI SONO IN GENERE ORGANIZZATI IN ROUTINE (PROCEDURE O FUNZIONI) CHE VENGONO

- ESEGUITE DIRETTAMENTE DAL DBMS - ACCOPPIAMENTO INTERNO
- CHIAMATE DA APPLICAZIONI ESTERNE IN ACCOPPIAMENTO ESTERNO

→ CREAZIONE DI ROUTINE

```
CREATE [OR REPLACE] FUNCTION <nome routine>
  ([<lista parametri>])
[ RETURNS <tipo ritorno> ]
AS
$$ <corpo routine>$$
LANGUAGE plpgsql;
```

<lista parametri> = <parametro>[, <lista parametri>]
<parametro>= [<modo>] [<nome parametro>]<tipo parametro>

Creatazione di Procedure o Funzioni,
Perche' non c'e' differenza in PostgreSQL
esteso

Possono essere scritte in più linguaggi,

- ↳ RAPPRESENTA IL TIPO DI PARAMETRO
- INPUT IN (DEFAULT)
 - OUTPUT OUT
 - INPUT/OUTPUT INOUT

→ POSSO ANCHE INDICARE IL TIPO DI UNA COLONNA
DI UNA TABELLA ESISTENTE
<Nome Tabella>.<Nome Colonna>%TYPE

→ TIPO DEL VALORE RESTITUITO DALLA FUNZIONE

- STESSI TIPI AMMISSEI RISPETTO AI PARAMETRI
- VOID SE LA FUNZIONE NON RESTITUISCE NULLA
- SE ALTRIMENTI UN PARAMETRO È OUT o INOUT, LA CLAUSOLA RETURNS PUÒ ESSERE OMessa
- SE È PRESENTE, DEVE CONCORDARE CON IL TIPO DEI PARAMETRI DI OUTPUT
 - SE CI SONO PIÙ PARAMETRI DI OUT o INOUT, <TIPO DI RITORNO> = RECORD

- RETURNS presente → funzione
(oppure se ci sono parametri OUT o INOUT)
- RETURNS assente → procedura
(se non ci sono parametri OUT o INOUT)

• BODY

→ CORPO DELLA PROVOCHE, ORGANIZZATO IN BLOCCHI

→ TUTTO IL CODICE DEVE ESSERE RACCHIUSO TRA \$\$ E \$\$

• CHIAMATA DI FUNZIONE

<NOME FUNZIONE> (<LISTA ARGOMENTI>)

NOME DELLA FUNZIONE

DA ESEGUIRE

LISTÀ DI ARGOMENTI CIASCUÑO DEI QUALI CORRISPONDE A:

- UN'ESPRESSIONE PER OGNI PARAMETRO DI INPUT ALLA FUNZIONE
- UNA VARIABILE INIZIALIZZATA, PER OGNI PARAMETRO DI INPUT-OUTPUT
- NON SONO PRESENTI I PARAMETRI DI OUTPUT

Lo servono solo a

- DARE UN NOME ALLE COLONNE DEL RISULTATO
- DEFINIRE UN TIPO DI RITORNO COMPLESSO IN FORMA ANONIMA

LA CHIAMATA DI FUNZIONE RESTITUISCE UN VALORE PER <TIPO DI RITORNO>

→ SE LA CLAUSOLA RETURNS NON È PRESENTE (O CI INDICANO RETURNS RECORD), VIENE RESTITUITO UN RECORD

- COSTITUITO DA VALORI ASSEGNATI AI PARAMETRI OUT E INOUT
- NEGL'ORDINE IN CUI SONO DEFINITI

→ IL RISULTATO DI UNA FUNZIONE È QUINDI SEMPRE UNA TUPLA FORMATA DA:

- UN SOLO ELEMENTO, RAPPRESENTATO DAL VALORE DI RITORNO, SE NON CI SONO PARAMETRI OUT E INOUT
- I VALORI ASSEGNATI A PARAMETRI OUT E INOUT, NEGL'ORDINE CON CUI CORRISPONDONO NELLA SEQUENZA

CREATE FUNCTION AggiornaVal2 (IN ilGenere CHAR(15))
RETURNS NUMERIC(3,2) AS

\$\$

<aggiorna la valutazione dei film di genere uguale ad ilGenere e restituisce valutazione media dei film con quel genere>

\$\$ LANGUAGE plpgsql;

Restituisce tupla formata da un unico elemento, che rappresenta la valutazione media

CREATE FUNCTION

AggiornaVal3 (IN ilGenere CHAR(15), **OUT** val NUMERIC(3,2))

AS

\$\$

<aggiorna la valutazione dei film di genere uguale ad ilGenere e restituisce valutazione media dei film con quel genere **nel parametro di output val**>

\$\$ LANGUAGE plpgsql;

Restituisce tupla formata da un unico elemento, che rappresenta la valutazione media

CREATE FUNCTION

AggiornaVal4 (IN ilGenere CHAR(15),

OUT val NUMERIC(3,2), **OUT** minAnno INTEGER)

AS

\$\$

<aggiorna la valutazione dei film di genere uguale ad ilGenere e restituisce:

(1) valutazione media dei film con quel genere nel parametro di output val

(2) l'anno di produzione del film piu` vecchio, con genere ilGenere, **nel parametro di output minAnno**>

\$\$ LANGUAGE plpgsql;

27

Restituisce tupla formata da due valori

UNA CHIAMATA DI FUNZIONE PUO' COMPARIRE

- IN UN COMANDO SQL
- IN UNA ISTRUZIONE PL / pgSQL

SELECT AggiornaVal1('drammatico');

aggiorna la valutazione dei film di genere drammatico e restituisce tabella vuota;

SELECT AggiornaVal2('drammatico');

aggiorna la valutazione dei film di genere drammatico e restituisce una tabella di grado 1, costituita da un'unica tupla contenente la valutazione media dei film di genere 'drammatico'

SELECT AggiornaVal3('drammatico');

aggiorna la valutazione dei film di genere drammatico e restituisce una tabella di grado 1, costituita da un'unica tupla contenente la valutazione media dei film di genere 'drammatico'

manca il parametro out nella chiamata
il risultato ha una colonna denominata
val

CORPO ROUTINE

è composto da due sezioni:

- **SEZIONE DI DICHIARAZIONE**: dichiarazione di tutte le variabili del programma
è opzionale

- **SEZIONE DI ESECUZIONE**: codice da eseguire istruzioni + statement SQL
è obbligatorio

```
[ DECLARE <dichiarazioni> ]
  BEGIN
    <istruzioni>
  END;
```

SEZIONE DI DICHIARAZIONE

→ se indicata, non può essere mancata

```
<nome> [ CONSTANT ] <tipo>
[ NOT NULL ]
[ { DEFAULT | := } <espressione> ];
```

↳ si indica DEFAULT o :=
→ viene assegnato il valore ottenuto dalla valutazione di <espressione>
altrimenti NULL

DECLARE

```
valutaz NUMERIC(3,2) DEFAULT 2.50;
regista VARCHAR(20) := 'quentin tarantino';
genere CONSTANT CHAR(15) := 'drammatico';
codCli DECIMAL(4) NOT NULL :=0000;
```

SEZIONE DI ESECUZIONE

- Contiene costrutti procedurale e statement SQL → collegano fra loro usando le variabili precedentemente dichiarate
- In SQL le variabili possono essere usate in:
 - clausola WHERE di statement Select, Delete, UPDATE
 - clausola SET di uno statement di UPDATE
 - clausola VALUES di uno statement di INSERT
 - clausola INTO di uno statement SELECT
- Nella parte procedurale ad una variabile può essere assegnato un valore ottenuto da una query

ASSEGNAZIONE

`<Nome_VAR> := <espressione>`

DEVONO AVERE LO STESSO TIPO

```
DECLARE ilGenere CHAR(15);
BEGIN
    ilGenere := 'comico';
    ...
    ilGenere := (SELECT genere FROM Film
    WHERE titolo = 'pulp fiction' AND regista =
    'quentin tarantino');
    ...
END;
```

Assegnando alla variabile un valore calcolato da una query si collegano i costrutti imperativi e quelli di SQL

OUTPUT

`RAISE NOTICE <stringa> [, <espressione>]`

Se contiene %, a video % viene sostituito con il valore di espressione

→ LA STRINGA VENDE VISUALIZZATA NELL'AREA 'MESSAGGI' DELL'OUTPUT

```
DECLARE ilGenere CHAR(15);
BEGIN
    ilGenere := 'comico';
    RAISE NOTICE 'Il genere considerato e`` %', ilGenere;
    ...
END;
```

I COMANDI CHE NON RESTITUISCONO TUPLE SONO ISTRUZIONI (INSERT, DELETE, UPDATE, CREATE TABLE)

```
DECLARE
    ilTitolo VARCHAR(30);
    ilRegista VARCHAR(20);
BEGIN
    ilTitolo := 'Pulp Fiction';
    ilRegista := 'quentin tarantino';
    UPDATE Film
    SET valutaz = valutaz * 1.1
    WHERE titolo = ilTitolo AND regista = ilRegista;
END;
```

Per i comandi **SELECT** che restituiscono esattamente una tupla, i valori degli attributi della tupla possono essere inseriti direttamente in variabili utilizzando lo statement **Select ... INTO**

```
DECLARE  
    laValutaz NUMERIC(3,2);  
BEGIN  
    [...]  
    SELECT valutaz INTO laValutaz  
    FROM Film  
    WHERE titolo = 'Mediterraneo' AND regista =  
        'gabriele salvatores';  
END;
```

laValutaz :=
(SELECT valutaz
FROM Film
WHERE titolo =
'Mediterraneo' AND regista =
'gabriele salvatores');

equivalente a

- Se la query restituisce più di una tupla, nelle variabili vengono inseriti i valori degli attributi della prima tupla restituita
- Se la query restituisce un insieme vuoto, viene assegnato NULL alle variabili
- Se si vuole che in entrambi i casi precedenti venga generato un errore, è necessario utilizzare la parola chiave **STRICT**

```
DECLARE  
    laValutaz NUMERIC(3,2);  
BEGIN  
    [...]  
    SELECT valutaz INTO STRICT laValutaz  
    FROM Film  
    WHERE titolo = 'Mediterraneo';  
END;
```

→ IN QUESTO MODO SE ESISTE PIÙ DI UN FILM 'Mediterraneo' O NON NE ESISTE, SI HA UN ERRORE

```
DECLARE  
    infoNoleggi INTEGER;  
    laValutaz NUMERIC(3,2);  
BEGIN  
    [...]  
    IF infoNoleggi > 5000 THEN laValutaz := 3.00;  
    ELSEIF infoNoleggi > 3000 THEN laValutaz := 2.00;  
    ELSE laValutaz := 1.00;  
END IF;  
INSERT INTO Film  
VALUES ('kill bill I', 'quentin tarantino', 2003, 'thriller', laValutaz);  
END;
```

POSso anche usare
costanti di scelta
(if else then)

Nelle strutture di controllo ho anche a disposizione i **CICLI DI ITERAZIONE INCERCA**

[WHILE <expr booleana>] LOOP <istruzioni> END LOOP;

↳ E' OPZIONALE:

- se assente continua a while TRUE
- se presente il ciclo termina quando la condizione diventa FALSE

↳ POSSO USARE:

• EXIT [WHEN <expr booleana>] per uscire dal ciclo

• CONTINUE [WHEN <expr booleana>] per passare all'esecuzione dell'iterazione successiva

• IN CASO DI CICLI ANNIDATI EXIT / CONTINUE fanno riferimento al ciclo più interno

↳ USANDO LE LABEL POSSO FARE REFERENZE ANCHE DA UN ALTRO CICLO PIÙ ESTERNO

ED I **CICLI DI ITERAZIONE CERTA** (perche' le due espressioni sono valutate una volta sola prima di iniziare l'esecuzione)

FOR <var indice> IN [Reverse] <expr> .. <expr>

[BY <expr>] LOOP <istruzioni> END LOOP;

→ VARIABILE DI TIPO INTERO, se specificata Reverse, la variabile decreterà in automatico ambiente incrementare

→ PERMETTE DI DEFINIRE IL PASSO (DI DEFAULT E' 1)

ESECUZIONE COMANDI SQL

Se con SELECT ho un insieme di tuple, spesso devo iterare sul risultato utilizzando i **CURSORI**

→ Puntatori a tuple nel risultato SQL, sono associati a valutazioni di interrogazioni

→ POSSO LEGGERE IN OGNI MOMENTO I VALORI ATTRIBUITI ALLA TUPLA A CUI PUNTA IL CURSOR

4 OPERAZIONI SUI CURSORI:

• **DICHIARAZIONE** ASSOCIA IL CURSOR ALL'INTERROGAZIONE → DA INSERIRE NELLA SEZIONE DI DICHIARAZIONE

<nome cursore> CURSOR FOR <select statement>

• **APERTURA** Esegue l'interrogazione associata al cursor e lo inizializza prima della prima tupla
OPEN <nome cursore> DEL RISULTATO

• CHIUSURA DISABILITA IL CURSOR (DOPO AVERLO CHIUSO E' NECESSARIO RIAPRIRLO PER USARLO → RISOLVE QUERY)

CLOSE <nome cursore>

• POSIZIONAMENTO CI SONO MOLTE POSSIBILI OPERAZIONI

LA PIÙ COMUNE È L'**AVANZAMENTO**, CHE POSIZIONA IL CURSOR SULLA TUPLA DEL RISULTATO SUCCESSIVA A QUELLA

FETCH <nome cursore> [INTO <lista variabili>] CORRENTE

↳ È POSSIBILE INSERIRE IN VARIABILI I VALORI DEGLI ATTRIBUTI DELLA TUPLA PRESA DAL CURSOR Dopo L'AVANZAMENTO

- SE LA TUPLA NON ESISTE VIENE INVESTITO NULL
- N° VARIABILI DEVE CORRISPONDERE AL N° COLONNE DELLA QUERY ASSOCIAVA AL CURSOR
- AD OGNI VARIABILE DELL'ELenco, DA SINISTRA A DESTRA, E' ASSEGNATO IL VALORE DELLA COLONNA CORRISPONDENTE CORRE POSIZIONE
- TIPO DI DATO DELLA VARIABILE DEVE ESSERE LO STESSO / COMPATIBILE CON QUELLO DELLA COLONNA CORRISPONDENTE

STATO DI UN'OPERAZIONE → LO VERIFICO CON IL COMANDO FOUND

• INIZIO FALSE

• A TRUE VIENE PODUTO DALLE OPERAZIONI:

- SELECT INTO SE VIENE SELETTA UNA TUPLA
- UPDATE, INSERT, DELETE SE NUOVO UNA TUPLA VIENE AGGIORNATA
- FETCH, SE DOPO LO SPOSTAMENTO IL CURSOR PUÒVA AD UNA TUPLA
- CYCLE FOR, SE VIENE ESEGUITA ALTRUI UN'ITERAZIONE

Attenzione!!!

Non «esageriamo» con i cursori: vanno usati solo se l'elaborazione che vogliamo fare su titolo e regista non si riesce a fare in modo set-oriented con comando SQL

ESEMPIO

DECLARE

 ilTitolo VARCHAR(30);
 ilRegista VARCHAR(20);
 valElevata CURSOR FOR

 SELECT titolo, regista FROM Film WHERE valutaz > 3.00;

BEGIN

 [...]

 OPEN valElevata;

 FETCH valElevata INTO ilTitolo, ilRegista;

 WHILE FOUND LOOP

 BEGIN

 ... elaborazione di ilTitolo e ilRegista ...

 FETCH valElevata INTO ilTitolo, ilRegista;

 END;

 END LOOP;

 CLOSE valElevata;

END;

SQL DINAMICO → CONANDI CREAMO DURANTE L'ESECUZIONE DEL'APPlicazione

EXECUTE <espressione di tipo stringa>

[**INTO [STRICT] <lista variabili>**]

NON PUO' CONTENERE Select INTO

DECLARE

```
ilTitolo VARCHAR(30);  
ilRegista VARCHAR(20);  
ilComando VARCHAR(100) := 'SELECT AVG(valutaz) FROM Film';
```

BEGIN

[...]

```
IF (ilRegista IS NOT NULL) THEN  
    ilComando := ilComando || ' WHERE regista=ilRegista';  
    IF (ilTitolo IS NOT NULL) THEN  
        ilComando := ilComando || ' AND titolo=ilTitolo';  
    END IF;  
ELSEIF (ilTitolo IS NOT NULL) THEN  
    ilComando := ilComando || ' WHERE titolo=ilTitolo';  
END IF;  
EXECUTE ilComando;
```

END;

GESTIONE DEGLI ERROREI

→ MECCANISMO DEGLI ECCEZIONI

→ POSSONO ESSERE CATTURATE ALL'INTERNO DI UN
qualsiasi BLOCCO

2 TIPI DI ECCEZIONI → DI SISTEMA (DEFINIRE E SOLLEVARE DA ESSO)

→ UTENSILE (DEFINIRE E SOLLEVARE DAL PROGRAMMA APPLICATIVO)

BEGIN

<istruzioni>

EXCEPTION

WHEN <condizione> [OR <condizione> ...] THEN

<handler_statements>

[**WHEN <condizione> [OR <condizione> ...] THEN**

<handler_statements> ...]

END;

Analogo a

try

{<istruzioni>}

catch (<exception>)

{<handler_statements>}

[**catch (<exception>)**

{<handler_statements>} ...]

Se viene sollevata un'eccezione a causa di un errore:

- Il controllo passa alla nostra delle eccezioni
- Si determina la prima condizione soddisfatta dall'errore rilevato e l'handler - Statement corrispondente viene eseguito
- Si passa all'esecuzione della prima istruzione dopo il blocco

Se nessuna condizione viene soddisfatta dall'errore, l'errore viene propagato al blocco più esterno.

Se un'eccezione non viene gestita a nessun livello, la funzione viene abortita.

La gestione delle eccezioni è costosa, va fatta solo quando è ragionevole aspettarsi che un errore si possa verificare.

→ le eccezioni non sono tipate

→ sulla base del valore dell'eccezione, si sceglie quale handler usare

→ eccezione in SQL: SQLSTATE, variabili globali e predefinita. A seconda del valore in SQL qual è l'errore

↳ OGNI CONDIZIONE CONTROLLA LO STATO DI SQLSTATE: 00000 → NO ERROR
oltre ai codici numerici si possono definire delle costanti standard

CREATE FUNCTION AggiornaVal(ilGenere CHAR(15)) AS
DECLARE

```
    laVal NUMERIC(3,2);
    ilTitolo VARCHAR(30);
    ilRegista VARCHAR(20);
    valCr CURSOR FOR SELECT titolo, regista, valutaz FROM Film WHERE genere = ilGenere;
BEGIN
    SELECT AVG(valutaz) INTO laVal
    FROM Film WHERE genere = ilGenere;
    IF (laVal < 4) THEN UPDATE Film SET valutaz = valutaz * 1.05;
    ELSE UPDATE Film SET valutaz = valutaz * 0.95;
    END IF;
    OPEN valCr;
    FETCH valCr INTO ilTitolo, ilRegista, laVal;
    WHILE FOUND LOOP
        BEGIN
            RAISE NOTICE 'Titolo: %', ilTitolo;
            RAISE NOTICE 'Regista: %', ilRegista;
            RAISE NOTICE 'Valutazione: %', laVal;
            FETCH valCr INTO ilTitolo, ilRegista, laVal;
        END;
    END LOOP;
    CLOSE valCr;
EXCEPTION
WHEN no_data THEN RAISE NOTICE 'Errore, dati non trovati';
END;
```

INTEGRAZIONE DI SQL + LINGUAGGIO DI PROGRAMMAZIONE GENERICO

• SQL STATICO

- COMANDI GIA' NOTI A TEMPO DI COMPILAZIONE
- PERMETTE PREPARAZIONE DEL COMANDO → MAGGIOR EFFICIENZA
- L'APPLICAZIONE HA PLENO CONTROLLO DEL COMANDO → MAGGIOR SICUREZZA / CORREZZIONE

• SQL DINAMICO

- COMANDI NOTI SOLO A TEMPO DI COMPILAZIONE
- FAVORISCE CREEZIONE ED ATTACCHI
- FACILITA LO SVILUPPO DI APPLICAZIONI ONLINE
- MAGGIOR TEMPO DI IMPOSTA, POICHÉ UN COMANDO SQL DINAMICO DEVE ESSERE PREPARATO OGNI VOLTA (STATICO LO SI FA UNA VOLTA PER ESSERE UTILIZZATO PIÙ VOLTE IN ESECUZIONI DISTINTE)

IMPEDIMENTO MISMATCH → DOVUTO ALLE DIFFERENZE TRA I DUE LINGUAGGI

- DIFFERENTE NEI TIPI DI DATO
- DIFFERENTE NELLA MODALITÀ DI ELABORAZIONE (SQL → SET-ORIENTED, ALM → TUPLE-ORIENTED)
- SCARSA INTEGRAZIONE IN TECNICHE DI PROGETTAZIONE E SVILUPPO

↳ SOLUZIONI

• TRADIZIONALI (RECEPITE DALLO STANDARD SQL)

- ESTENSIONI OPERATORIALI PER ARMONIZZARE LE DIFFERENZE (TIPO I CURSORI)
- I COMANDI SQL POSSONO ESSERE INVOCATI DA LINGUAGGI GENERAL PURPOSE
- I COMANDI SQL SONO OPACI AL LINGUAGGIO → INTEGRAZIONE DEBOLE

• INNOVATIVE (ESTERNE A SQL, A LIVELLO DI LINGUAGGIO / PIATTAFORMA)

- INFRASTRUTTURA RUN-TIME PER GESTIRE DATI RELAZIONALI COME OGNI

! → POSSIBILITÀ DI ESEGUIRE TIPICHE OPERAZIONI DI INTERROGAZIONE E MANIPOLAZIONE ALL'INTERNO DI LINGUAGGI GENERAL PURPOSE → INTEGRAZIONE FORTE

GESTIONE DEI RISULTATI

IL RISULTATO DI UN'INTERROGAZIONE SQL DEVE ESSERE INTRODOTTO IN UNA STRUTTURA DATI DEL LINGUAGGIO DI PROGRAMMAZIONE :

→ SE RESTITUISCE UNA SOLO TUPLA: DEFINISCO UNA VARIABILE DI COMUNICAZIONE IN CUI INSERIRE I VALORI RESTORNATI DALLA TUPLA RESTITUITA

→ SE RESTITUISCE PIÙ TUPLE: LA DIMENSIONE DEL RISULTATO NON È NOTA A COMPILE-TIME E PUÒ ESSERE TROPPO GRANDE PER ESSERE CARICATA IN MEMORIA.
→ NECESSARIO USO DI CURSORI PER CAMBIARE LE TUPLE UNA AD UNA

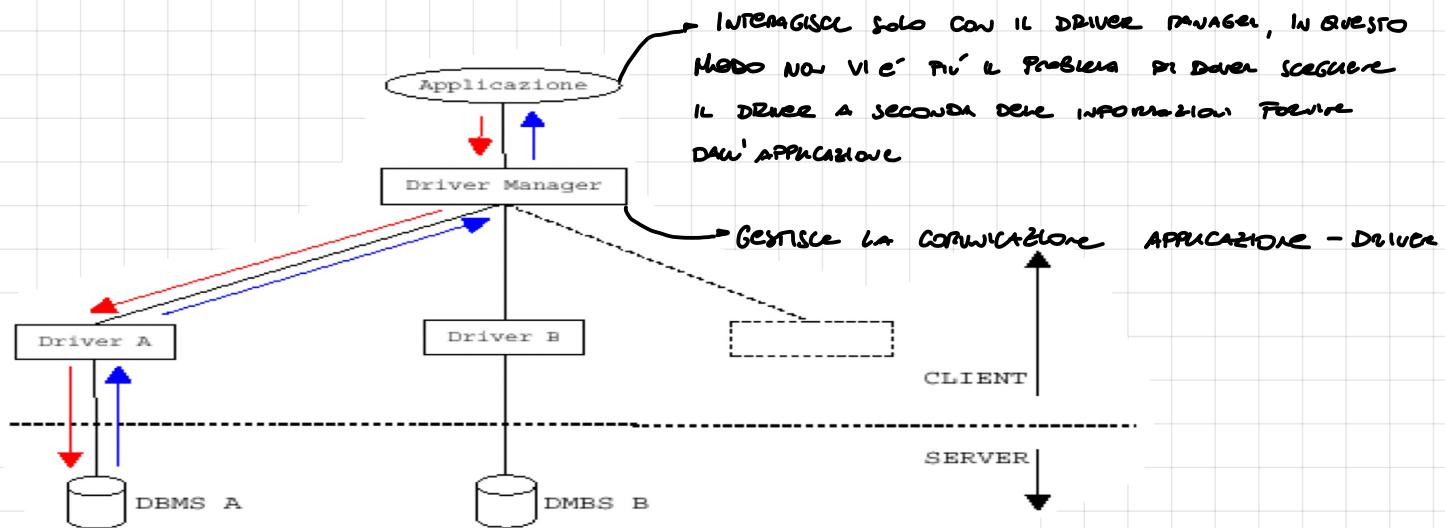
CALL LEVEL INTERFACE: INTERFAZIA AUTOMATICA CON IL QUALE AVREMO L'ACCESSO AL BLOCCHETTO DI DATI

LO STANDARD SQL PREvede SQL/CLI

STANDARD DE FACTO: JDBC/ODBC

DRIVER: PROGRAMMA CHE TRADUCE TUTTE LE CHIAMATE ODBC/JDBC IN CHIAMATE SPECIFICHE PER UN DBMS

JDBC



Ricevono sempre e solo richieste nel linguaggio SUPPORTATO

JDBC è UNA API JAVA STANDARD PER INTERAGIRE CON BAN DI DATI

PER CIASCUN DBMS ESISTONO 4 TIPI DI DRIVER: SI DIFFERENZIANO PER IL LIVELLO A CUI INTERAGISCONO CON LA BAN DI DATI

Il più semplice è JDBC-ODBC che un ODBC per convertire al BD

ESEMPIO

apertura connessione

esecuzione (unica
possibilità per SQL
dinamico)

risultato restituito una
tupla alla volta, stile
cursor

SQL
statico

preparazione

esecuzione

chiusura connessione

```

import java.sql.*;
import java.io.*;
class exampleJDBC
{
    public static void main (String args [])
    {
        Connection con = null;
        try {
            String ilGenere = "comico";
            Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection("jdbc:odbc:ilMioDB",
                                            "laMiaLogin",
                                            "laMiaPassword");
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("SELECT AVG(valutaz)
                                         FROM Film
                                         WHERE genere = '" + ilGenere + "'");
            rs.next();
            if (rs.getDouble(1) < 4)
                st.executeUpdate("UPDATE Film
                                 SET valutaz = valutaz * 1.05");
            else
                st.executeUpdate("UPDATE Film
                                 SET valutaz = valutaz * 0.95");
            PreparedStatement pst =
                con.prepareStatement("SELECT titolo, regista, valutaz
                                    FROM Film
                                    WHERE genere = ?");
            pst.setString(1,ilGenere);
            rs = pst.executeQuery();
            while (rs.next())
                System.out.println("Titolo: " + rs.getString(1) +
                                   " Regista: " + rs.getString(2) +
                                   " Valutazione: " + rs.getDouble(3));
            con.close();
        }
        catch(java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }
        catch (SQLException e) {
            while( e!=null){
                System.out.println("SQLState: " + e.getSQLState());
                System.out.println("Code: " + e.getErrorCode());
                System.out.println("Message: " + e.getMessage());
                e = e.getNextException();
            }
        }
    }
}
  
```

ESTENDERE DEL LINGUAGGIO APPLICATIVO PER SQL IN 2 MODE:

- INTEGRAZIONE FORTE : COSTRUM DEL LINGUAGGIO DA TRASDURRE IN COMANDI SQL
- INTEGRAZIONE DEBOLE : COMANDI SQL INIETTATI NEL CORPO DI UN BLOCCO SEPARATO

↳ SQL OSPITATO

SQL OSPITATO

Dopo lo sviluppo di JDBC c'è stata proposta una specifica ANSI/ISO per incapsulare comandi SQL in programmi Java → SQLJ

I COMANDI SQL OSPITATO POSSONO APPARIRE IN OGNI PUNTO IN CUI PUO' COMPARE UN'ISTRUZIONE DEL LINGUAGGIO OSPITANTE

→ OGNI ISTRUZIONE DELLE ESSERE CHIARAMENTE IDENTIFICABILE NEL TESMO DEL PROGRAMMA :

- PRECEDUTA DA UN PREFIXO E SEGUITA DA UN TERMINATORE

EXEC SQL ↗
 ↗;
 ↗;
 ↗;
 ↗;
 ↗;

ESEMPIO

apertura connessione

SQL statement
uso simile a
plpg/SQL

sql iterator ~
cursore

```
import java.sql.*;
import java.io.*;
import java.math.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import oracle.sqlj.runtime.*;

class exampleSQLj
{
    #sql iterator FilmIter(String titolo, String regista);
    public static void main (String args [])
    {
        try{
            Double valMedia;
            String ilGenere = "comico";
            oracle.connect("jdbc:odbc:ilMioDB",
                          "laMiaLogin",
                          "laMiaPassword");
            #sql{SELECT AVG(valutaz) INTO :valMedia
                  FROM Film
                  WHERE genere = :ilGenere};
            if (valMedia > 4)
                #sql{UPDATE Film SET valutaz = valutaz * 1.05};
            else
                #sql{UPDATE Video SET valutaz = valutaz * 0.95};
            FilmIter ilFilmIter = null;
            #sql ilFilmIter ={SELECT titolo, regista, valutaz
                               FROM Film
                               WHERE genere = :ilGenere};
            while (ilFilmIter.next())
                System.out.println("Titolo: " + ilFilmIter.titolo() +
                                   " Regista: " + ilFilmIter.regista() +
                                   " Valutazione: " + ilFilmIter.valutaz());
        }
        catch(java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }
        catch (SQLException e) {
            while(e!=null){
                System.out.println("SQLState: " + e.getSQLState());
                System.out.println("Code: " + e.getErrorCode());
                System.out.println("Message: " + e.getMessage());
                e = e.getNextException();
            }
        }
    }
}
```

TRANSAZIONI E PROGRAMMI APPLICATIVI

A seconda di come e' settato l'autocommit ogni comando costituisce implicitamente una transazione
a se' stante e tutti i comandi eseguiti all'interno di una sessione costituiscono implicitamente una transazione.

SET AUTOCOMMIT

In PostgreSQL il default e' false

START TRANSACTION

COMMIT

ROLLBACK

TRANSAZIONI IN JDBC

In JDBC Autocommit e' su true → disabilita modificando `setAutoCommit` della classe Connection
ogni comando SQL viene eseguito come una transazione, se Autocommit intero e' disabilitato, una transazione viene creata all'inizio dell'esecuzione dell'applicazione

→ termina invocando il metodo Commit o Rollback
della classe Connection o in maniera implicita al termine dell'esecuzione

TRANSAZIONI SQL

SQL per default disabilita l'autocommit

termina la transazione con:

- `*sql COMMIT;`
- `*sql ROLLBACK;`

L'autocommit puo' comunque essere modificato al momento della connessione con il DBMS

TRIGGER

- DBMS TRADIZIONALI → PASSIVI
 - ADBMS → ATTIVI, OSSIA IN GRADO DI REACTIONS AD EVENTI ESEGUEndo OPERAZIONI DEFINITE DAL PROGRAMMA
- ↳ SI POSSONO DEFINIRE REGOLE ATTIVE O TRIGGER

I TRIGGER FORNISCONO UNA NUOVA REATTIVA FUNZIONALITA' AUTONOMA DELLE GATE AI PROGRAMMI APPLICATIVI

- EFFICIENTI
- COSTI DI MANUTENZIONE
- UNIFORMITA' DI GESTIONE DEI DATI
- INTEGRAZIONE CON LE ALTRE COMPONENTI DEL DBMS

- MONITORAGGIO
 - VINCOLI DI INTEGRITA'
 - ALERTING
 - AUDITING
 - SICUREZZA
 - STATISTICHE
 - ECCEZIONI
- } USI DEI DBMS ATTIVI CON TRIGGER

Alcune OPERAZIONI SONO AUTOMATICAMENTE ESEGUITE QUANDO SI VERIFICA UNA CERTA PRIMA SITUAZIONE
INTERNA O ESTERNA ALLA BASE DI DATI

- Eventi specifici
- particolari condizioni o particolari stati o transizioni di stato

UN TRIGGER C'È IL CODISTRO SINTATTICO PER DEFINIRE LA REACTIONS DEL SISTEMA ED È SPECIFICATO NEL DDL DEL DBMS

PARADIGMA ECA

EVENTO - CONDIZIONE - AZIONE

- ON evento
 - IF condizione
 - THEN azione
- AL VERIFICARSI DELL'EVENTO SI VALUTA LA CONDIZIONE E SE È SODDISFAVILE
SI ESEGUE L'AZIONE
- ↳ ULTERIORE CONTROLLO CHE VIENE ESEGUITO QUANDO IL TRIGGER È CONSIDERATO E PRIMA
CHE L'AZIONE SIA ESEGUITA

EVENTI

POSSIBILITA' DI DEFINIRE TRIGGER CHE POSSONO ESSERE ATTIVATI PRIMA O DOPO UN EVENTO

- BEFORE
 - UTILE PER TRIGGER CHE VERIFICANO PRE-CONDIZIONI
 - SE NON SONO VERE → ABORT, SI PRECIDE L'ESECUZIONE
DEL'EVENTO

- AFTER
 - IL TRIGGER VIENE ESEGUITO DOPO
L'ESECUZIONE DELL'EVENTO

POSSO COMBINARE GLI EVENTI (EVENTI CONNESSI) USANDO OPERATORI LOGICI OPPURE UNA SEQUENZA

È VANTAGGIOSO AVERE L'EVENTO POICHE' VALUTARE UNA CONDIZIONE È COSTOSO, mentre RILEVARE L'ACCADERE DI UN EVENTO È IMMEDIATO.

INOLTRE SI POSSONO SPECIFICARE AZIONI DIVERSE PER EVENTI DIVERSI E SICSA CONDIZIONE.

CREAZIONE DI UN TRIGGER IN SQL : 2000

```
CREATE TRIGGER <nome trigger>
{BEFORE | AFTER} <evento> ON <tabella soggetto>
[WHEN <condizione>]
{<comando SQL> |
BEGIN ATOMIC <sequenza di comandi SQL> END} ;
```

- POSSIBILI EVENTI:
per la tabella
SOGGETTO
- Insert
 - Delete
 - Update

• Update [OF <lista attributi>] → se specificato, il trigger viene attivato solo da un evento che modifica tutti e solo gli attributi a1, ..., an

a1, ..., an

CONDIZIONE

ESPRESSIONE BOOLEANA SQL ARBITRARIA

AZIONE

- UN SINGOLO COMANDO SQL
- UNA SEQUENZA DI COMANDI SQL
- NON POSSONO CONTENERE PARAMETRI DI CONNESSIONE
- NEL CASO DI TRIGGER DI TIPO BEFORE, SQL SCONSIGLIA L'ESECUZIONE DI COMANDI DI AGGIORNAMENTO DEI DATI NEL CONTESTO DELL'AZIONE, MA NON LO VIENE

↳ POTREBBERO AGGIORNARE ALCUNI DATI PRIMA DELL'ESECUZIONE DELL'EVENTO CHE HA ATTIVATO IL TRIGGER, GENERANDO CORPORETTA PIENO ANONIMI (SOPRATTUTTO IN PRESENZA DI PIÙ' TRIGGERS PER LO STESSO EVENTO)

VI SONO DUE MODALITÀ DI ESECUZIONE:

• **ORIENTATA ALL'ISTANZA**: IL TRIGGER VIENE ESEGUITO UNA VOLTA PER OGNI TUPLA COINVOLTA NELL'EVENTO CHE ATTIVA IL TRIGGER E SODDISFA LA CONDIZIONE
FOR EACH ROW

• **ORIENTATA ALL'INSIEME**: IL TRIGGER VIENE ESEGUITO UNA SOLA VOLTA PER TUTTE LE TUPLE COINVOLTE NELL'EVENTO

FOR EACH STATEMENT

ESECUZIONE ORIENTATA ALL'INSIEME: VALUTAZIONE CONDIZIONE ED ESECUZIONE AZIONE VENGONO ESEGUITI UNA SOLO VOLTA, PER L'INSIEME DI TUPLE COINVOLTE NELL'EVENTO CHE HA ATTIVATO IL TRIGGER

→ Insieme delle tuple aggiornate dall'evento si chiama **TABELLA DI TRANSIZIONE**

ESECUZIONE ORIENTATA ALL'ISTANZA: VALUTAZIONE CONDIZIONE ED ESECUZIONE AZIONE VENGONO ESEGUITI UNA VOLTA PER OGNI TUPLA COINVOLTA NELL'EVENTO CHE HA ATTIVATO IL TRIGGER

→ La tupla coinvolta nell'evento viene chiamata **VARIABILE o TUPLA DI TRANSIZIONE**

TUPLE E TABELLE DI TRANSIZIONE

- OLD o OLD ROW **Before**
 - New o New Row **AFTER**
 - OLD TABLE **Before**
 - NEW TABLE **AFTER**
- } SI POSSONO USARE NELLA CONDIZIONE E/O NEW/ACTION
SI POSSONO ASSEGNARE ALIAS ALLE TUPLE/TABELLE DI TRANSIZIONE

CREATE TRIGGER <nome trigger>

{BEFORE | AFTER} <evento> ON <tabella soggetto>

[REFERENCING { OLD [ROW] AS <variabile> |
NEW [ROW] AS <variabile> |
OLD TABLE AS <variabile> |
NEW TABLE AS <variabile> }]
di specificano alias a livello di
tabella o tupla di transizione

[FOR EACH {ROW | STATEMENT}]

[WHEN <condizione>]

{<comando SQL> |

BEGIN ATOMIC <sequenza di comandi SQL> END};

26

• LA PAROLA CHIARE OLD/NEW
SPECIFICA ALIAS PER LA TABELLA/TUPLA
DI TRANSIZIONE PRIMA/DOPPO
DELL'ESECUZIONE DELL'EVENTO

Trigger T

- **Evento**: inserimento nella relazione Film
- **Condizione**: valutaz IS NULL
- **Azione**: calcolo del valor medio di valutaz ed assegnazione di tale valore moltiplicato per 1.1 all'attributo valutaz delle tuple inserite

• **Operazione scatenante**: Comando SQL che inserisce 5 tuple in Film

ESEMPIO DI ESECUZIONE

- **Orientata all'insieme**: LA CONDIZIONE VIENE VALUTATA E L'AZIONE VIENE ESEGUITA UNA SOLA VOLTA, INDIPENDENTEMENTE DAL NUMERO DI FILM INSERITI

TUTTI I 5 FILM INSERITI AVRANNO LO STESSO VALORE PER L'ATTRIBUTO VALUTAZ

• **ORIENTATA ALL'ISTANZA**: LA CONDIZIONE VENE VALUTATA E L'AZIONE VENE ESEGUITA UNA VOLTA PER OGNI FILT INSERITO (quindi s volte)

I S FILT INSERITI AVRANNO VALORI DELL'ATTRIBUTO VALUTAT POTENTIALMENTE DIVERSI
→ LA MEDIA E' CALCOLATA SU INSTANTI DIFFERENTI DI VALORI

LA VISIBILITA' DELLE TUPLE / TABELLE DI TRANSIZIONE DURANTE LA VALUTAZIONE DELLA CONDIZIONE E L'ESECUZIONE DELL'AZIONE DIPENDE DA:

- TIPO DI TRIGGER
- TIPO DI ESECUZIONE
- EVENTO CHE HA ARRIVATO IL TRIGGER

tupla su cui l'evento dovrà essere eseguito

	FOR EACH ROW		FOR EACH STATEMENT	
BEFORE	tuple sì	tabelle no	tuple no	tabelle no
AFTER	tuple sì	tabelle sì	tuple no	tabelle sì

Si esegue il trigger **prima** di completare l'esecuzione dell'evento

⇒

la tabella di transizione non esiste ancora

33

VISIBILITA' DI TUPLE / TABELLE DI TRANSIZIONE

• INSERT

• NON SI POSSONO SPECIFICARE CLAUSOLE REFERENCING OLD, POICHÉ LE TUPLE INSERITE DALL'EVENTO NON ESISTEVANO PRIMA DELLA SUA ESECUZIONE

• SE IL TRIGGER E' DI TIPO BEFORE LE TUPLE INIZIALI NON SONO VISIBILI NELLA TABEGLIA SOGGETTO, MA POSSONO ESSERE ACCEDUTE UNA ALLA VOLTA USANDO LA TUPLA DI TRANSIZIONE NEW

• SE INVECE E' DI TIPO AFTER LE TUPLE INIZIALI SONO VISIBILI NELLA TABEGLIA SOGGETTO E POSSONO ESSERE ACCEDUTE MEDIANTE LA TUPLA O LA TABEGLIA DI TRANSIZIONE NEW

• DELETE

• NON SI POSSONO SPECIFICARE CLAUSOLE REFERENCING NEW, POICHÉ LE TUPLE CANCELLATE DALL'EVENTO NON ESISTONO PIÙ DOPO LA SUA ESECUZIONE

• SE IL TRIGGER E' DI TIPO BEFORE LE TUPLE CANCELLATE SONO VISIBILI NELLA TABEGLIA SOGGETTO E POSSONO ESSERE ACCEDUTE USANDO LA TUPLA DI TRANSIZIONE OLD

• SE INVECE E' DI TIPO AFTER LE TUPLE CANCELLATE SONO VISIBILI NELLA TABEGLIA SOGGETTO, MA POSSONO ESSERE ACCEDUTE USANDO LA TUPLA O LA TABEGLIA DI TRANSIZIONE OLD

• UPDATE

- I VALORI PRECEDENTI E CORRENTI DELLE TUPLE POSSONO ESSERE ACCEDUTI USANDO LE CLAUSOLE REFERENCING OLD e NEW, A LIVELLO DI TUPA NEI TRIGGER DI TIPO BEFORE e A LIVELLO DI TABELLA NEI TRIGGER DI TIPO AFTER

L'EFFETTO DELLE MODIFICA E' VISIBILE ANCHE NEI TABELLI SOGLIOLO

Tipo trigger e modalità esecuzione	Evento	Tabelle/tuple di transizione
BEFORE ROW	INSERT DELETE UPDATE	NEW OLD NEW, OLD
AFTER ROW	INSERT DELETE UPDATE	NEW, NEW TABLE OLD, OLD TABLE TUTTE
BEFORE STATEMENT	INSERT DELETE UPDATE	- - -
AFTER STATEMENT	INSERT DELETE UPDATE	NEW TABLE OLD TABLE NEW TABLE, OLD TABLE

ATTIVITA' FONDAMENTALI IN UN ADBMS:

- RILEVARE GLI EVENTI ED ATTIVARE I TRIGGER CORRESPONDENTI
- PROCESSO REATTIVO:** SELEZIONARE ED ESEGUIRE I TRIGGER

} POSSANO ESSERE ESEGUITE CONCURRENTEMENTE

CRITERI DI SCELTA DEL TRIGGER

- TIPO DI TRIGGER (BEFORE / AFTER)
 - MODALITA' DI ESECUZIONE (ROW / STATEMENT)
 - PRIORITA' ASSEGNAVA NELLO STANDARO DAL TEMPO DI CREAZIONE DEL TRIGGER
- PostgreSQL lo assegna in base al nome (ORDINE ALFABETICO)
- è necessario considerare anche il CONTROLLO DEI VINCOLI

- O
R
D
I
N
E
- TRIGGER BEFORE, FOR EACH STATEMENT

- PER OGNI TUPLA OGGETTO DEL COMANDO CHE RAPPRESENTA L'EVENTO:

- TRIGGER BEFORE, FOR EACH ROW: ESECUZIONE EVENTO PER LA SINGOLA TUPLA E VERIFICA DEI VINCOLI DI INTEGRITÀ SUA TUPLA, CON VALIDAZIONE IMMEDIATA

• TRIGGER AFTER, FOR EACH ROW

• VERIFICA DEI VINCOLI CON VALIDAZIONE IMMEDIATA SULLA TABELLA

• TRIGGER AFTER, FOR EACH STATEMENT

L'esecuzione dell'azione di un trigger può provocare nuovi eventi e a loro volta attivare altri (esecuzione a cascata)

→ QUESTI TRIGGERS POSSONO ESSERE AGGIUNTI ALL'INTERNO DI UN FILE DA CONSIDERARE (**MODALITA' ITERATIVA**) OPPURE DARE ORIGINE AD UNA NUOVA ESECUZIONE DELL'ALGORITMO DURANTE L'ESECUZIONE DELL'AZIONE DEL TRIGGER CORRENTEMENTE ATTIVATO (**MODALITA' RECURSIVA**)

TERMINAZIONE

- IL PROCESSO REATTIVO POTREBBE NON TERMINARE
- NON VENGONO POSTE RESTRIZIONI SINTATTICHE PER EVITARE LA NON TERMINAZIONE
- DI SOLITO I DBMS HANNO UNA LINEA SUPERIORE AL NUMERO DI TRIGGERS ATTIVABILI CONSECUTIVAMENTE

PER CANCELLARE UN TRIGGER:

DROP TRIGGER <nome trigger>;

TRIGGERS E VINCOLI

I TRIGGERS SONO PIÙ FLESSIBILI DEI VINCOLI D'INTEGRITÀ, POICHÉ PERMETTONO DI STABILIRE COME REAGIRE AD UNA VIOLAZIONE DEL VINCULO

I TRIGGERS POSSANO OPERARE ANCHE VINCOLI DI TRANSIZIONE

ALCUNE VOLTE PERO', DEFINIRE DEI VINCOLI È PIÙ VANTAGGIOSO:

Ogni cliente non può noleggiare più di tre video contemporaneamente

Invece di abortire la transazione se il vincolo è violato si vuole annullare l'inserimento

- MIGLIORE OTIMIZZAZIONE
- MENO ERRORE DI PROGRAMMAZIONE
- I VINCOLI SONO PIÙ DEI STANDARD DA LUNGO TEMPO.
- I TRIGGERS NO

CREATE TRIGGER

VerificaNoleggi

AFTER INSERT ON Noleggio

REFERENCING NEW ROW AS NR

FOR EACH ROW

WHEN (SELECT COUNT(*)

FROM Noleggio

WHERE dataRest IS NULL AND
codCli = NR.codCli) > 3

DELETE FROM Noleggio

WHERE colloc = NR.colloc AND

dataNol = NR.dataNol;

CREATE ASSERTION

VerificaNoleggi

CHECK (NOT EXISTS

(SELECT * FROM Noleggio
WHERE dataRest IS NULL
GROUP BY codCli
HAVING COUNT(*) > 3));

Aggiornamento automatico attributo ptiMancanti nella tabella Standard ad ogni nuovo noleggio:

```
CREATE TRIGGER CalcolaPtiMancanti
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
BEGIN ATOMIC
UPDATE Standard
SET ptiMancanti = ptiMancanti - 1
WHERE codCli = NR.codCLI AND
    NR.colloc IN (SELECT colloc FROM Video
                  WHERE tipo = 'v');
UPDATE Standard
SET ptiMancanti = ptiMancanti - 2
WHERE codCli = NR.codCLI AND
    NR.colloc IN (SELECT colloc FROM Video
                  WHERE tipo = 'd');
END;
```

ogni VHS 1 punto
ogni DVD 2 punti

5

Quando il valore dell'attributo ptiMancanti per un cliente standard diventa 0, il cliente è rimosso dalla tabella Standard ed inserito nella tabella VIP con bonus pari a 5 euro

```
CREATE TRIGGER OrganizzaClienti
AFTER UPDATE OF ptiMancanti ON Standard
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN NR.ptiMancanti <= 0
BEGIN ATOMIC
    INSERT INTO VIP
        VALUES (NR.codCli,5.00);
    DELETE FROM Standard
        WHERE codCli = NR.codCli;
END;
```

51

TRIGGER IN POSTGRESQL

```
CREATE TRIGGER Nome
{ BEFORE | AFTER } {Evento [OR Evento] *}
ON Relazione
[FOR EACH {ROW | STATEMENT} ]
[WHEN (Condizione)]
EXECUTE PROCEDURE NomeFunzione ( Argomenti )
```

il default è **FOR EACH STATEMENT**

- **CREATE CONSTRAINT TRIGGER**
{DEFERRABLE {INITIALLY IMMEDIATE | INITIALLY DEFERRED} | NOT DEFERRABLE}
come visto per i vincoli (momento di attivazione si modifica con **SET CONSTRAINTS**)
- **INSTEAD OF** *Evento* solo per trigger definiti su viste

TRIGGER POSTGRESQL - EVENTI

- Comandi di **INSERT, DELETE, UPDATE** su relazione (di base, no viste)
- è possibile specificare **UPDATE OF** *Lista attributi*
- è possibile specificare più di un evento (in **OR**)
- trigger attivato **BEFORE** o **AFTER**
- **INSTEAD OF** solo per trigger su viste

Altri comandi:

- **DROP TRIGGER**
- **ALTER TRIGGER** (permette solo di modificare il nome)
- Clausole **DISABLE TRIGGER** [*NomeTrigger* | **ALL** | **USER**] e **ENABLE TRIGGER** [*NomeTrigger* | **ALL** | **USER**] del comando di **ALTER TABLE**

TRIGGER POSTGRESQL – AZIONE

- L'azione deve essere l'invocazione di una *funzione* definita dall'utente
 - deve essere definita senza parametri
 - deve restituire tipo **trigger**
- A tale funzione si può passare una lista di *argomenti* (stringhe, separate da virgola)
- La stessa trigger function può essere utilizzata come azione di più trigger, l'uso dei parametri permette di specializzarne l'utilizzo
- Vedremo solo trigger function specificate in PL/pgSQL

TRIGGER FUNCTION IN PL/PGSQL

- PL/pgSQL può essere utilizzato per definire trigger function
- Una trigger function è una funzione senza parametri e con tipo di ritorno **trigger** creata con il comando **CREATE FUNCTION**
- La funzione deve essere dichiarata senza parametri anche se ci si aspetta che riceva gli argomenti specificati nel **CREATE TRIGGER**
- Gli argomenti del trigger sono passati alla funzione attraverso l'array **TG_ARGV**

TRIGGER FUNCTION IN PL/PGSQL

- Quando una funzione PL/pgSQL è dichiarata come trigger function vengono automaticamente create diverse variabili speciali nel top-level block
- Queste variabili sono:
 - **NEW**: contiene la nuova versione della tupla per le operazioni INSERT/UPDATE nei trigger row-level, NULL nei trigger statement-level
 - **OLD**: contiene la vecchia versione della tupla per le operazioni DELETE/UPDATE nei trigger row-level, NULL nei trigger statement-level

TRIGGER FUNCTION IN PL/PGSQL

- **TG_NAME:** contiene il nome del trigger correntemente attivato
- **TG_WHEN:** contiene la stringa AFTER o BEFORE a seconda della definizione del trigger
- **TG_LEVEL:** contiene la stringa ROW o STATEMENT a seconda della definizione del trigger
- **TG_OP:** contiene la stringa INSERT, UPDATE o DELETE a seconda dell'operazione che ha attivato il trigger
- **TG_TABLE_NAME:** contiene il nome della relazione su cui è definito il trigger
- **TG_NARGS:** contiene il numero degli argomenti passati alla trigger function nel comando CREATE TRIGGER
- **TG_ARGV[]:** array di stringhe, che contiene gli argomenti passati alla trigger function nel comando CREATE TRIGGER
 - l'indice parte da 0
 - accessi all'array con indici invalidi risultano in valore NULL

TRIGGER FUNCTION IN PL/PGSQL

- Una trigger function deve restituire NULL oppure una tupla con la stessa struttura della tabella su cui è stato attivato il trigger
- Il valore di ritorno è utilizzato solo dai trigger *before row*
 - valore di ritorno NULL segnala al trigger manager di non eseguire il resto dell'operazione per tale tupla
 - i trigger successivi non sono eseguiti
 - l'operazione corrispondente all'evento non viene eseguita
 - valore di ritorno diverso dal valore NEW originale modifica la tupla che verrà inserita o aggiornata
 - per modificare tale riga è possibile modificare i campi di NEW e restituire la NEW modificata o costruire una nuova tupla e restituirla
- Per tutti gli altri tipi di trigger il valore di ritorno è ignorato (e può essere o meno NULL)

POSTGRESQL - MODALITÀ DI ESECUZIONE

- Scelta regola:
 - dipende dal tipo di trigger come in SQL200N
 - trigger BEFORE STATEMENT
 - per ogni tupla oggetto del comando
 - trigger BEFORE ROW
 - comando e verifica dei vincoli di integrità
 - trigger AFTER ROW
 - verifica dei vincoli che richiedono di aver completato il comando
 - trigger AFTER STATEMENT
 - se esistono più trigger dello stesso tipo: ordine alfabetico
- Possibilità di esecuzione differita per CONSTRAINT TRIGGER

POSTGRESQL – TABELLA RIASSUNTIVA

Eventi primitivi	Operazioni sulla base di dati
Eventi compositi	OR
Transition tuple/table	tuple
Constraint trigger	deferrable, solo for each row
Terminazione	Nessuna condizione
Ordinamento regole	Tipo + ordine alfabetico

POSTGRESQL VS SQL200N

	PostgreSQL	SQL200N
Evento	OR di insert, delete, update	Singolo insert, delete, update, update of
Tuple e tabelle transizione	Solo per ROW trigger, solo tupla di transizione No clausola REFERENCING	ROW e STATEMENT, tupla e tabella
Condizione	Senza sottoquery ROW e STATEMENT (ma senza tabelle transiz.)	ROW e STATEMENT
Azione	Invocazione di trigger function	Sequenza di comandi
Ordinamento	Alfabetico su nome	Tempo creazione

TRIGGER PostgreSQL – ESEMPIO 1

- Trigger che implementa vincolo ‘non più di tre video contemporaneamente’

```
CREATE OR REPLACE FUNCTION non_tre() RETURNS trigger AS
$non_tre$
BEGIN
    IF (SELECT COUNT(*) FROM Noleggio
        WHERE dataRest IS NULL AND codCli = NEW.codCli) > 3
    THEN
        RAISE EXCEPTION '%ne ha già tre', NEW.codCli;
        -- nb nn viene effettuato return new, la tupla nn è inserita
    ELSE
        RETURN NEW;
    END IF;
END;
$non_tre$ LANGUAGE plpgsql;
```

TRIGGER POSTGRESQL – ESEMPIO 1

- Trigger che implementa vincolo ‘non più di tre video contemporaneamente’

```
CREATE TRIGGER non_più_di_tre
BEFORE INSERT OR UPDATE ON Noleggio
FOR EACH ROW
EXECUTE PROCEDURE non_tre();
```

TRIGGER POSTGRESQL – ESEMPIO 2

- Trigger che implementa dato derivato punti mancanti

```
CREATE OR REPLACE FUNCTION agg_pti() RETURNS trigger AS
$agg_pti$
BEGIN
  IF (NEW.codCli IN (SELECT codCli FROM Standard))
  THEN
    UPDATE Standard
    SET ptimancanti = ptimancanti - 1
    WHERE codcli = NEW.codcli AND NEW.colloc IN
          (SELECT colloc
           FROM VIDEO
           WHERE tipo = 'v');
```

TRIGGER POSTGRESQL – ESEMPIO 2

- Trigger che implementa dato derivato punti mancanti

```
UPDATE Standard
SET ptiMancanti = ptiMancanti - 2
WHERE codCli = NEW.codCli AND NEW.colloc IN
      (SELECT colloc
       FROM VIDEO
       WHERE tipo = 'd');
END IF;
RETURN NEW;
END;
$agg_pti$ LANGUAGE plpgsql;
```

TRIGGER POSTGRESQL – ESEMPIO 2

- Trigger che implementa dato derivato punti mancanti

```
CREATE TRIGGER agg_pti  
AFTER INSERT ON Noleggio  
FOR EACH ROW  
EXECUTE PROCEDURE agg_pti();
```

TRIGGER POSTGRESQL – ESEMPIO 3

- Trigger che implementa regola operativa migrazione clienti

```
CREATE OR REPLACE FUNCTION diventa_vip() RETURNS trigger
AS $diventa_vip$
BEGIN
    IF (NEW.ptiMancanti <= 0)
    THEN
        INSERT INTO Vip
        VALUES (NEW.codCli,5.00);
        DELETE FROM Standard
        WHERE codCli = NEW.codCli;
    END IF;
    RETURN NEW;
END;
$diventa_vip$ LANGUAGE plpgsql;
```

TRIGGER POSTGRESQL – ESEMPIO 3

- Trigger che implementa regola operativa migrazione clienti

```
CREATE TRIGGER diventa_vip  
AFTER UPDATE ON Standard  
FOR EACH ROW  
EXECUTE PROCEDURE diventa_vip();
```

REGOLE PostgreSQL

- PostgreSQL prevede anche una nozione di regola che consente di dire al query executor come riscrivere una determinata operazione
- Quelle che coinvolgono le operazioni di aggiornamento prendono il nome di *update rule*

```
CREATE [ OR REPLACE ] RULE Nome AS
ON Evento TO Relazione
[ WHERE Condizione ]
DO [ ALSO | INSTEAD ]
{ NOTHING | Comando | (Comando [;Comando]*) }
```

POSGRESQL: REGOLE VS TRIGGER

- Le regole manipolano il comando o generano un nuovo comando da eseguire
- L'effetto dell'esecuzione di una regola è specificato in termini del *query tree*
- Molte cose che si possono fare con i trigger si possono fare anche con le regole
 - l'approccio dei trigger è più aderente allo standard e concettualmente più semplice dell'approccio delle regole
 - i trigger sono in generale più adatti per gestire i vincoli di integrità
 - le regole permettono di gestire in maniera flessibile le modifiche attraverso le viste

REGOLE POSGRESQL

- Esempio: aggiornamenti su vista Nolegg

```
CREATE VIEW Nolegg AS  
SELECT colloc, dataNol, codCli  
FROM Noleggio  
WHERE dataRest IS NULL AND  
(CURRENT_DATE - dataNol) > INTERVAL '3' DAY;
```

REGOLE POSGRESQL

- Esempio: aggiornamenti su vista Noleggio

```
CREATE RULE nol3gg_ins AS ON INSERT
TO Noleggio
DO INSTEAD
INSERT INTO Noleggio VALUES (
    NEW.colloc,
    NEW.dataNol,
    NEW.codCli);
```

REGOLE POSGRESQL

- Esempio: aggiornamenti su vista Nol3gg

```
CREATE RULE nol3gg_del AS
  ON DELETE TO Nol3gg
  DO INSTEAD
  DELETE FROM Noleggio
  WHERE colloc = OLD.Collect
        AND dataNol = OLD.dataNol;
```

REGOLE POSGRESQL

- Esempio: aggiornamenti su vista Noleggio

```
CREATE RULE nol3gg_upd AS ON UPDATE TO Noleggio
DO INSTEAD
UPDATE Noleggio
SET codCli = NEW.codCli,
    colloc = NEW.colloc,
    dataNol = NEW.dataNol
WHERE colloc = OLD.Collect AND dataNol = OLD.dataNol;
```

UNA FORMA NORMALE E' UNA PROPRIETA' DI UNA BASE DI DATI RELAZIONALE CHE NE GARANTISCE LA "QUALITA'", CIOE' L'ASSUNTA DI DETERMINATI DIFETTI

→ QUANDO UNA RELAZIONE NON E' NORMALIZZATA, PRESENTA RIDONDANZE E SI PRESTA AD ANOMALIE

LA NORMALIZZAZIONE E' LA PROCEDURA CHE PERMETTE DI TRASFORMARE SCHEMI NON NORMALIZZATI IN SCHEMI CHE SODDISFANO UNA FORMA NORMALE

→ VA UTILIZZATA COME UNA TECNICA DI VERIFICA DEI RISULTATI DELLA PROGETTAZIONE
DI UNA BASE DI DATI. **NON COSTRUISE UNA METODOLOGIA DI PROGETTAZIONE**

- ANOMALIE**
- RIDONDANZA
 - DI AGGIORNAMENTO
 - DI INSERIMENTO
 - DI CANCELLAZIONE

VINCOLO D'INTEGRITA': DIPENDENZA FUNZIONALE

• Relazione r su $R(X)$

• Due sottoinsiemi non vuoti Y e Z di X

• Esiste in r una dipendenza funzionale (FD) da Y a Z se, per ogni coppia di tuple t_1 e t_2 di r con gli stessi valori su Y , risulta che t_1 e t_2 hanno gli stessi valori anche su Z

$X \rightarrow Y$

○ Esempi:

MatricolaStudente → NomeStudente
MatricolaStudente → CognomeStudente

MatricolaStudente → Anno

CodiceCorso → SiglaCorso

CodiceCorso → NomeCorso

CodiceCorso → Crediti

MatricolaStudente CodiceCorso → Voto

TERZA FORMA NORMALE DI BOYCE E COOD (BCNF)

UNA RELAZIONE r E' IN FORMA NORMALE DI BOYCE E COOD K , PER Ogni DIPENDENZA FUNZIONALE (NON BANALE) $X \rightarrow Y$ DEFINITA SU DI essa, X CONTIENE UNA CHIAVE K DI r

LA FORMA NORMALE RICHIEDE CHE I CONCERNI IN UNA RELAZIONE SIANO ONTOGENI (SOLO PROPRIMENTE CORRISPONDENTI ASSOCIARE ALLA CHIAVE)

- DIPENDENZE FUNZIONALI E CHIAVI SONO COLLEGATE: PER OGNI COPPIA DI TUPLE t_1 E t_2 DI r CON GLI STESSI VALORI SU K , RISULTA CHE t_1 E t_2 HANNO GLI STESSI VALORI ANCHE SU TUTTI GLI ATTRIBUTI IN X

TERZA FORMA NORMALE

UNA RELAZIONE r E' IN TERZA FORMA NORMALE SE, PER OGNI FD (NON BANALE) $X \rightarrow Y$ DEFINITA SU r , E' VERIFICATA ALMENO UNA DELLE SEGUENTI CONDIZIONI:

- X CONTIENE UNA CHIAVE K DI r
- OGNI ATTRIBUTO IN Y E' CONTENUTO IN ALMENO UNA CHIAVE DI r

LA TERZA FORMA NORMALE E' PIENO RESTRITTIVA DELLA FORMA NORMALE DI BOYCE E COOD (e ANTERIE RELAZIONI CON ALCUNE ANOMALIE)

→ HA IL VANTAGGIO DI ESSERE SEMPRE RAGGIUNGIBILE

SCOMPOSIZIONE:

- SI CREA UNA RELAZIONE PER OGNI GRUPPO DI ATTRIBUTI COINVOLTI IN UNA DIPENDENZA FUNZIONALE
 - SI VERIFICA CHE ALLA FINE UNA RELAZIONE CONTENGA UNA CHIAVE DELLA RELAZIONE ORIGINARIA
- PRODUCE UNA DECOMPOSIZIONE IN TERZA FORMA NORMALE

SE FUNZIONA O MELO DIPENDE DALLE DIPENDENZE INDIVIDUATE, SE SONO IN FORMA RIWINALE, SI

- Non ci sono dipendenze inutili
 - $A \rightarrow B, B \rightarrow C$
 - $A \rightarrow C$ è inutile
- Le dipendenze non contengono attributi inutili a sinistra
 - $A \rightarrow B, AB \rightarrow C$
 - B in $AB \rightarrow C$ è inutile perché se in due tuple A coincide, allora coincide anche B
- Le dipendenze non contengono attributi inutili a destra
 - $A \rightarrow B, A \rightarrow C, A \rightarrow CD$
 - In $A \rightarrow CD$, C è ridondante

SE LA RELAZIONE NON È NORMALIZZATA SI DECOPONE IN TERZA FORMA NORMALE E ALLA FINE SI VERIFICA SE LO SCHHEMA OTTENUTO È ANCHE IN BCNF

→ SE UNA RELAZIONE HA UNA SOLO CHIAVE ALLORA LE DUE FORME NORMALI CONCLUDONO

PROGETTAZIONE E NORMALIZZAZIONE

LA TEORIA DELLA NORMALIZZAZIONE PUÒ ESSERE USATA NELLA PROGETTAZIONE LOGICA PER VERIFICARE LO SCHEMA RELAZIONALE FINALE

LA METODOLOGIA DI PROGETTAZIONE APPENA VISTA AIUTA A PRODURRE SCHEMI NORMALIZZATI, MA NON LO GARANTISCE

IN ALCUNI CASI PUÒ ESSERE ACCETTABILE UTILIZZARE SCHEMI NON NORMALIZZATI MA È IMPORTANTE CHE LA SCHEMA SIA CONSAPEVOLI E DOCUMENTATA

QUALITÀ DEL SERVIZIO

LE PRESTAZIONI VENGONO ANALIZZATE IN RIFERIMENTO A UN CERTO CARICO DI LAVORO / WORKLOAD

• **TUNING:** ATTIVITÀ CON CUI SI ANALIZZANO LE PRESTAZIONI DEL SISTEMA E SI CERCANO DI ADOTTARE CORRETTIVI

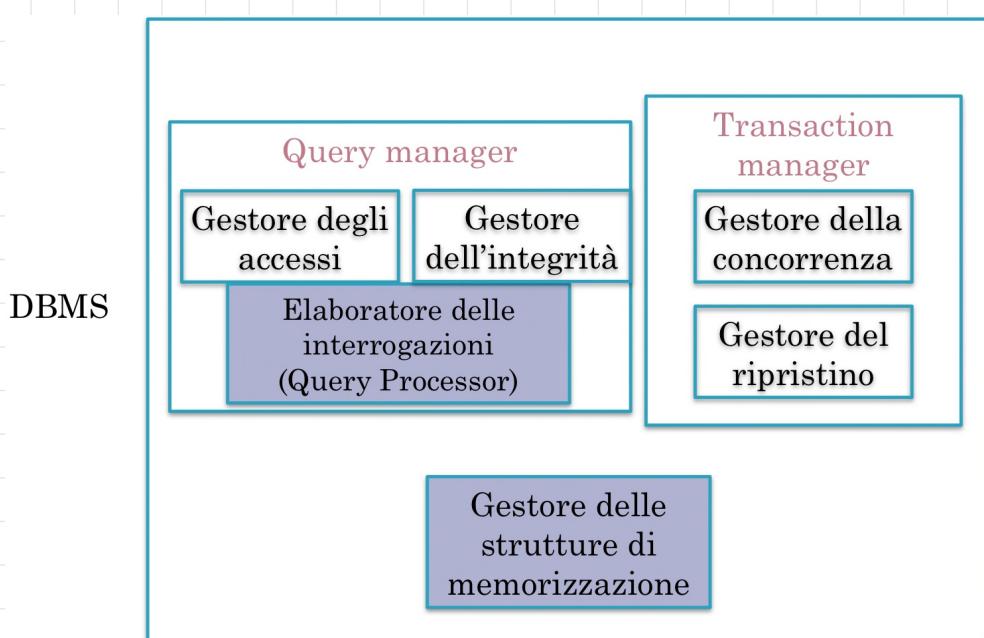
QUANDO IL SISTEMA NON E' ANDATO IN PRODUZIONE → TUNING = PROGETTAZIONE

WORKLOAD

• QUALI SONO LE INTERROGAZIONI PIÙ IMPORTANTI E CON CHE FREQUENZA VENGONO ESEGUGI

• QUALI SONO GLI AGGIORNAMENTI PIÙ IMPORTANTI E CON CHE FREQUENZA VENGONO ESEGUITI

• QUALI SONO LE PRESTAZIONI DESIDERATE / NECESSARIE PER TALE INTERROGAZIONI E AGGIORNAMENTI E PER IL SISTEMA IN GENERALE



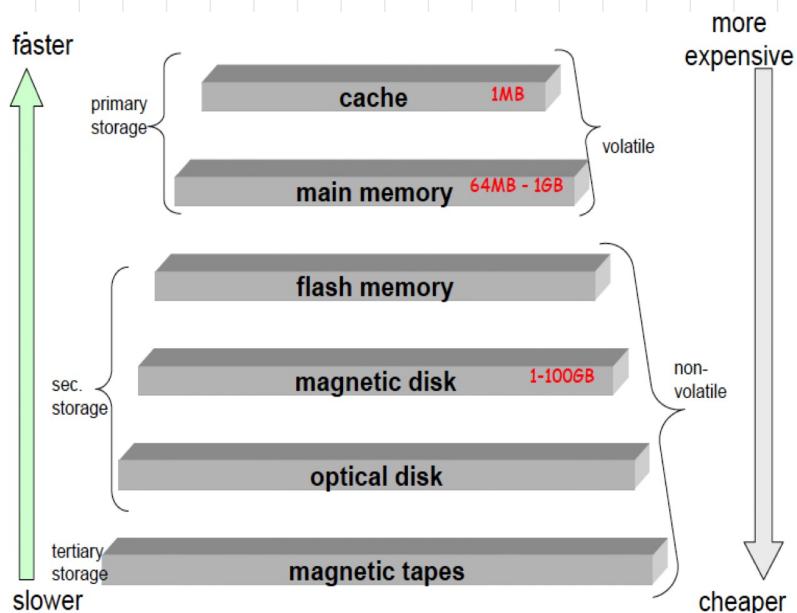
GESTORE DELLE STRUTTURE DI MEMORIZZAZIONE

PRESTAZIONI DI UNA MEMORIA: DATO UN INDIVIDUO DI MEMORIA, le prestazioni si misurano in termini di tempo di accesso, determinato dalla somma della

- LATENZA
- TEMPO DI TRASFERIMENTO

TEMPO DI ACCESSO : LATENZA + DIREZIONE DATI DA TRASFERIRE
VELOCITÀ DI TRASFERIMENTO

LEGGE DI MOORE → Come conseguenza ha che diventa sempre più oneroso muovere i dati lungo i livelli della gerarchia

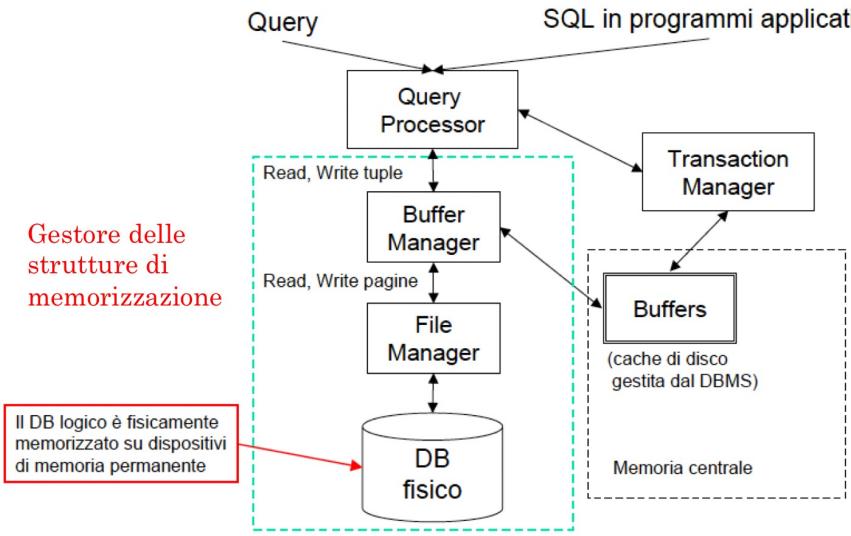


UN DB A CAUSA DELLA SUA DIMENSIONE RISOLVE NORMALMENTE IN DISCHI E I DATI PER ENERE ELABORATI DAL DBMS DEVONO ESSERE TRASFERITI IN MEMORIA CENTRALE

↳ NON AVviENE IN TERMINI DI SINGOLE TUPLE, BENSÌ DI BLOCCI

IL BOTTLENECK È CAUSATO PRINCIPALMENTE DALLE OPERAZIONI DI I/O, BISOGNA QUINDI OTTIMIZZARE L'IMPLEMENTAZIONE FISICA DEL DB, ATTENDENDO:

- GESTORE DELLE STRUTTURE DI MEMORIZZAZIONE:
 - ORGANIZZAZIONE EFFICIENTE DEI TUPLE SU DISCO
 - GESTIONE EFFICIENTE DEI BUFFER IN MEMORIA
 - STRUTTURE DI ACCESSO EFFICIENTI
- ELABORATORE DEVE INTERROGARSI:
 - STRATEGIE DI EXECUZIONE EFFICIENTI PER LE QUERY



Gestione delle strutture di memorizzazione

Si preoccupa di decidere come gestire e migliorare la lettura dei dati su disco e del buffer

Ha una componente a stretto contatto con una base di dati fisica (dati memorizzati su disco)

Ha due sottocomponenti:

- FILE MANAGER → Gestire lettura e scrittura di dati sul disco
- BUFFER MANAGER → Gestire il trasferimento delle pagine tra Buffer e Disco

FILE MANAGER

DATI MEMORIZZATI SU DISCHI → TEMPI DI ACCESSO LUNGHETTI

→ DATI SALVATI IN TRACCE DI DISCO, SUDDIVISE IN SEZIONI (100-1000 PER TRACCIA)
SEZIONE È L'UNITÀ MINIMA PER LETTURA E SCRITTURA
DATI MEMORIZZATI SULLO STESSO CIUDORO POSSONO ESSERE RECUPERATI PIÙ VELOCEMENTE DI ALTRI

→ TRACCIE CON LO STESSO DIRETTO SULLE VARIETÀ SUPERFICI

PRESTAZIONI DEI DISCHI MAGNETICI:

- INTERNE: DIPENDONO DA
 - CARATTERISTICHE MECCANICHE
 - TECNICHE DI MEMORIZZAZIONE E CODIFICA DEI DATI
 - DISK CONTROLLER (INTERFAZIA TRA HHD E SISTEMA)
- ESTERNE: DIPENDONO DA
 - TIPO DI INTERFAZIA
 - ARCHITETTURA DEL SOTTOsistEMO DI I/O

MISURARE IN
TEMPO DI LATENZA E
TEMPO DI TRASFERIMENTO

TEMPO DI LATENZA

TEMPO IMPIEGATO PER ACCEDERE AL PRIMO BYTE

SEEK TIME + ROTATIONAL LATENCY + SPIN TIME + COMMAND OVERHEAD TIME

nel caso peggiore richiede qualche decina di milisecondi

TEMPO DI TRASFERIMENTO

Velocità massima alla quale il driver può leggere o scrivere i dati

DIMENSIONE DATI SULLA TRACCIA

TEMPO DI ROTAZIONE

UNITA' DI TRASFERIMENTO → BLOCCHI o PAGINE

UNA PAGINA È 4 - 64 KB, PAGINI TROPPO PICCOLI CAUSANO FREQUENTI I/O, TROPPO GRANDI DURANTE LA FRATTAMENTORE INTERNO E INCHIDONO PIÙ SPAZIO IN MEMORIA PER CUSTODIRE CAMBIARE

IL TEMPO DI TRASFERIMENTO DI UN BLOCCO (T_t) è il tempo impiegato dalla testina per trasferire un blocco nel buffer, una volta posizionata all'interno del blocco

DIMENSIONE DEL BLOCCO

$$T_t = \frac{\text{Dimensione del blocco}}{\text{Velocità di trasferimento}}$$

Il gestore deve strutturare la memorizzazione. Deve cercare di ridurre il tempo di latenza

IL DB FISICO (livello fisico)

Indirizzi di file visti come una collezione di pagine di dimensione fissa

Ogni pagina memorizza più record

↳ consiste in più campi che rappresentano gli attributi

Casi unici:

- OGNI RELAZIONE DEL DB È MEMORIZZATA IN UN PROPRIO FILE
- TUTTO IL DB È MEMORIZZATO IN UN SINGOLO FILE

DB FISICO IN POSTGRESQL

DATABASE CLUSTER: insieme di tutti i database gestiti da un server PostgreSQL

↳ contiene tabele e altri oggetti

↳ OGUNA REROMIZZATA IN UN FILE SEPARATO

PGSQL ORGANIZZA LO SPAZIO FISICO IN TABLESPACE, CIASCUONE CORRISPONDENTE A UNA POSIZIONE SU DISCO IN CUI VENGONO MEMORIZZATI I FILE CORRISPONDENTI A OGGETTI DEL DATABASE

IL FILE CORRISPONDENTE AD UNA TABELLA È MEMORIZZATO IN UN SINGOLO TABLESPACE, MA ESSO PUÒ CONTENERE PIÙ RELAZIONI

TRARRE I TABLESPACE IL SISTEMA DECIDE DOVE MEMORIZZARE I VARI OGGIETTI

↳ PERMETTERE DI RAGGIUNGERE LE MIGLIORI PRESTAZIONI NEI CASI DI DB DI GRANDI DIMENSIONI

LE PRESTAZIONI DI UN DBMS DIPENDONO FORTEMENTE DA COME I DATI SONO ORGANIZZATI SU DISCO
AL FILE SYSTEM NON SONO NOTI I TIPI DI ACCESSO AI DATI, QUASI SONO LE RELAZIONI LOGICHE TRA ELIMI E COME
DOVANNO ESSERE ELABORATE

RECORD

FORMA IN CUI SONO GENERALMENTE MEMORIZZATI I DATI

- OGNI RECORD È COSTITUITO DA UN'INSIEME DI VALORI COLLEGATI
- OGNI VALORE È FORMATO DA UNO O PIÙ BYTES E CORRISPONDE AD UN CAMPO DEL RECORD

TIPO DI RECORD COLLEZIONE DI NODI DI CAMPI A CUI SONO ASSOCIATI I TIPI CORRESPONDENTI

PER OGUNO DI ESSI NEL DB DEVE ESSERE DEFINITO UNO SCHEMA (FISICO) CHE PERMETTA DI
INTERPRETARE CORRETTAMENTE IL SIGNIFICATO DEI BYTES CHE COSTITUISCONO IL RECORD

CHAR (n) : n byte

VARCHAR (n) : m + p byte

n° caratteri ↓
n° byte per rappresentare la stringa
nella struttura

DATE, TIME : STRINGHE DI LUNGHEZZA FISSA

- DATE : 10 caratteri
- TIME : 8 caratteri

LIVELLO LOGICO

Relazione (tabella)

Tupla

LIVELLO FISICO

File

Record

```
CREATE TABLE Film
(titolo VARCHAR(30),
regista VARCHAR(20),
anno DECIMAL(4) NOT NULL,
genere CHAR(15) NOT NULL,
valutaz NUMERIC(3,2),
PRIMARY KEY (titolo,regista))
```

```
Struct Film {char titolo [30],
char regista [20],
int anno,
char genere [15],
real valutaz};
```

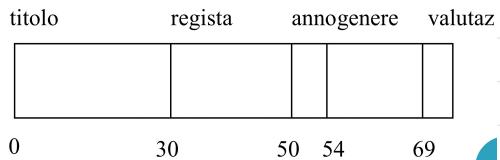
- Ogni record memorizzato in 73 byte
 - $30+20+15+4+4=73$ per le stringhe
 - 4 per l'intero
 - 4 per il reale



FILE CON RECORD A LUNGHEZZA

LUNGHEZZA FISSA

Struct Film {char titolo [30],
char regista [20],
int anno,
char genere [15],
real valutaz};



SI PUO' IDENTIFICARE DOVE INIZIA IL RECORD E DOVE INIZIA OGNI CAMPO DI ESSE

LUNGHEZZA VARIABILE

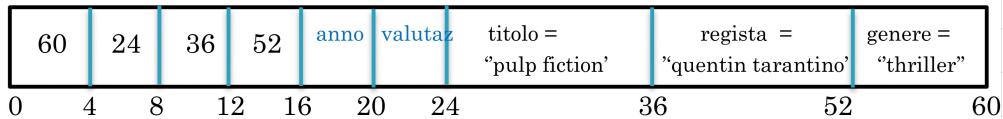
UN FILE CHE CONTIENE UN RECORD DI QUESTO TIPO HA VARIE REGIONI:

- PUO' CONTENERE CAMPI DI LUNGHEZZA VARIABILE
- PUO' CONTENERE CAMPI OZIONALI
- PUO' ESSERE UN FILE HETEROGENEO: Contiene RECORD DI TIPI DIFFERENTI

UNA POSSIBILE SOLUZIONE PER MEMORIZZARE RECORD DI QUESTO TIPO e':

- MEMORIZZARE PRIMA TUTTI I CAMPI A LUNGHEZZA FISSA, E QUINDI TUTTI QUELLI A LUNGHEZZA VARIABILE
- PER OGNI CAMPO A LUNGHEZZA VARIABILE SI HA UN PREFIX POINTER CHE RIPORTA L'INDIRIZZO DEL PRIMO BYTE DEL CAMPO

Struct Film { varchar titolo [30],
varchar regista [20],
int anno,
varchar genere [15],
real valutaz};



ORGANIZZAZIONE DEI RECORD IN BLOCCHI

DI SOLITO LA DIMENSIONE DI UN RECORD e' MINORE DI QUELLA DI UNA PAGINA

L'ORGANIZZAZIONE DI UNA PAGINA DI RECORD A LUNGHEZZA FISSA:

Page header	record 1	record 2	...	record n	
-------------	----------	----------	-----	----------	--

→ MANTIENE LE INFORMAZIONI DELLA PAGINA, UNA SORTA DI METADATI
(ID DELLA PAGINA NEL DB, ULTIMA MODIFICA, RELAZIONE A CUI LE TUPLE APPARTENGONO)

• NORMALMENTE UN RECORD e' CONTENUTO INTERAMENTE IN UNA PAGINA → POSSIBILITA' DI SPAZIO SPERATO

SI PREFERISCE NON MEMORIZZARE RECORD A CAVALLO DI PIÙ BLOCCHI, PERCHE' R/W VAIANO DA BLOCCO A BLOCCO, QUINDI PER LEGGERE UN RECORD A CAVALLO SI OPPORTUNERESSERO DUE LETTURE

ORGANIZZAZIONE DEI BLOCCHI SU DISCO: ALLOCAZIONE CONTINUA

I BLOCCHI DI UNO STESSO FILE VENGONO MEMORIZZATI SU DISCO IN MODO VIZIOSO (UNO Dopo L'altro) PER UN ACCESO PIÙ VELOCE
IN LETTURA E SCRITTURA, SEMPRE SE SI VOGLIONO LEGGERE TUTTE LE TUPLE

→ ESPANSIONE DEL FILE DIFFICILE

ALLOCAZIONE CONCATENATA

OGNI BLOCCO CONTIENE UN PUNTATORE AL SUCCESSIVO BLOCCO DEL FILE

→ ESPANSIONE DEL FILE FACILE

→ lettura lenta

In Generale I SISTEMI USANO UNA VIA DI MEZZO TRA QUESTE DUE ALLOCAZIONI:

- **BUCKER** insieme di blocchi non per forza contigui ma comunque vicini (velocemente raggiungibile) per record dello stesso relazione

ORGANIZZAZIONE DEL RECORD NEI FILE

TUTTI I FILE CHE CONTENGONO I RECORD COSTITUISCONO L'**ORGANIZZAZIONE PRIMARIA** DEI DATI

IL MODO CON CUI I RECORD VENGONO ORGANIZZATI NEI FILE INCIDE SULL'EFFICIENZA DELLE OPERAZIONI E SULL'OCCUPAZIONE DI MEMORIA

- È IMPORTANTE ASSEGNARE I RECORD AI BLOCCHI IN MODO TALE CHE UNO STESSO BLOCCO CONTENGA RECORD TRA LORO INTERRELATI
- PIÙ SONO VICINI, PIÙ IL TEMPO DI SEEK SI RIDUCE

CATALOGHI DI SISTEMI CONTENGONO INFORMAZIONI SULL'ORGANIZZAZIONE PRIMARIA DEI DATI, DIPENDONO DALLO SPECIFICO DBMS CONSIDERATO E IN GENERALE CONTENGONO PER OGNI RELAZIONE

- Numero tuple
- Numero blocchi nel file
- Altre statistiche
- Informazioni su strutture auxiliary per l'accesso ai dati

SELECT *

FROM Noleggio;

- L'ordine di memorizzazione dei record nel file è ininfluente a fini prestazionali perché devono essere restituite tutte le tuple

SELECT *

FROM Noleggio

WHERE dataNol = '15-mar-2006';

- Se si potesse 'ricordare' la posizione del primo blocco contenente record con **dataNol = '15-mar-2006'**, si ridurrebbe il numero di blocchi 'inutili' letti

SELECT *

FROM Noleggio

WHERE dataNol = '15-mar-2006';

- Se i record fossero memorizzati ordinati nel file, si potrebbe procedere sequenzialmente, fino al primo record con **dataNol = '15-mar-2006'** e leggere fino al primo record con **dataNol > '15-mar-2006'**
- Si riduce il numero di blocchi letti
- Si possono comunque leggere blocchi che non contengono risultati
 - I blocchi contenenti i film con **dataNol < '15-mar-2006'** (precedono i blocchi contenenti i record di nostro interesse se il file è ordinato rispetto a dataNol)

I RECORD NEI FILE POSSONO ESSERE ORGANIZZATI IN DIVERSE MODALITÀ

• FILE HEAP: I RECORD DATI VENGONO MESSI UNO DOPPO L'ALTRO IN ORDINE DI INVENTARIO

• FILE ORDINATO: I RECORD DATI SONO MESSI IN ORDINE DI INVENTARIO MA CONSERVANDO L'ORDINAMENTO DI UNO O PIÙ CAMPI

• FILE HASH: I RECORD CHE CONDIVIDONO LO STESSO VALORE PER UNO O PIÙ CAMPI SONO MESSI VICINI, O NUOVO SCHEMA
BUCKET IN GENERE

File heap

File ordinato su dataNol

File hash su dataNol

colloc	dataNol	codCli	dataRest
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006
1118	10-Mar-2006	6642	11-Mar-2006
1117	02-Mar-2006	6635	06-Mar-2006
1118	02-Mar-2006	6635	06-Mar-2006
1111	04-Mar-2006	6642	05-Mar-2006
1119	08-Mar-2006	6635	10-Mar-2006
1120	08-Mar-2006	6635	10-Mar-2006
1116	08-Mar-2006	6642	09-Mar-2006
1121	15-Mar-2006	6635	18-Mar-2006
1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1122	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1113	22-Mar-2006	6642	?
1116	21-Mar-2006	6610	?

colloc	dataNol	codCli	dataRest
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006
1117	02-Mar-2006	6635	06-Mar-2006
1118	02-Mar-2006	6635	06-Mar-2006
1111	04-Mar-2006	6642	05-Mar-2006
1119	08-Mar-2006	6635	10-Mar-2006
1120	08-Mar-2006	6635	10-Mar-2006
1116	08-Mar-2006	6642	09-Mar-2006
1118	10-Mar-2006	6642	11-Mar-2006
1121	15-Mar-2006	6635	18-Mar-2006
1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1112	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1116	21-Mar-2006	6610	?
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1113	22-Mar-2006	6642	?
1116	22-Mar-2006	6642	?

colloc	dataNol	codCli	dataRest
1117	02-Mar-2006	6635	06-Mar-2006
1118	02-Mar-2006	6635	06-Mar-2006
1111	04-Mar-2006	6642	05-Mar-2006
1119	08-Mar-2006	6635	10-Mar-2006
1120	08-Mar-2006	6635	10-Mar-2006
1116	08-Mar-2006	6642	09-Mar-2006
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006
1118	10-Mar-2006	6642	11-Mar-2006
1121	15-Mar-2006	6635	18-Mar-2006
1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1112	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1116	21-Mar-2006	6610	?
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1113	22-Mar-2006	6642	?

Gestione del Buffer

UN ALTRO modo per minimizzare gli accessi al disco consiste nel mantenere più blocchi possibile in memoria principale

Ogni blocco che deve essere letto o scritto deve partire dal buffer (cache del dbms)

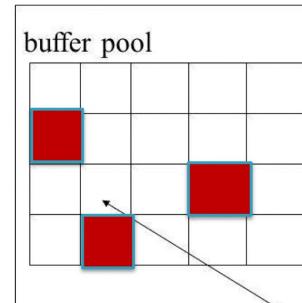
la sua gestione è affidata al gestore del buffer

il buffer è un'area di memoria organizzata in pagine, che hanno la stessa dimensione delle righe / blocchi su disco

TECNICOLOGIA ALTERNATIVA:

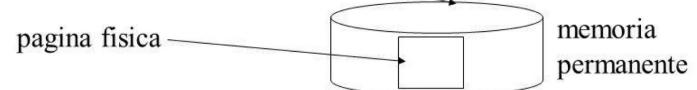
- BUFFER → BUFFER POOL
- PAGINA DEL BUFFER → BUFFER

gestore
buffer



memoria primaria

tabella pag. residenti:
<id_pag, posiz. buffer>



DURANTE UN'OPERAZIONE DI LETTURA (SELECT), IL BUFFER MANAGER APRE CON:

- SE LA PAGINA E' NEL BUFFER → VIENE FORNITO L'INDIRIZZO DELLA PAGINA CORRISPONDENTE
- SE LA PAGINA NON E' IN MEMORIA → IL BUFFER MANAGER SELEZIONA UNA PAGINA NEL BUFFER PER LA PAGINA RICHIESTA. SE E' OCCUPATA VIENE RASCIUTTA SU DISCO SOLO SE E' STATA MODIFICATA E NON ANCORA SALVATA SU DISCO E SE NELL'ISTANTE LA STA USANDO. A QUESTO PUNTO IL B.M. PUO' LEGGERE LA PAGINA DA DISCO E COPIARLA NELLA PAGINA DEL BUFFER SELEZIONATA, RIPPLATZANDO QUELLA PRESENTE

POLITICA LRU NON E' ADOTTATA PER I DBMS

- PER MOTIVI LEGATI ALLA GESTIONE DEL RECOVERY IN ALCUNI CASI UN BLOCCO NON PUO' ESSERE TRASFERITO SU DISCO
- PER MOTIVI LEGATI ALLA GESTIONE DEL RECOVERY IN ALCUNI CASI E' NECESSARIO TENERE UN BLOCCO SU DISCO
→ STEAL/NO STEAL, FORCE/NO FORCE POLICIA WAL
- UN DBMS E' IN GRADO DI PREDIRE MEGLIO DI UN S.O. IL TIPO DEI FUTURI RIFERIMENTI → DAI COTANONI

STRUTTURE AUXILIARIE DI ACCESSO

HEAP, SEQUENZIALE ED HASH PERMETTONO DI MIGLIORARE LE PRESTAZIONI DI RICERCHE RISPECTO A CONDIZIONI SU ATTRIBUTI CHE SONO QUELLI UTILIZZATI PER ORGANIZZARE I RECORD NEL FILE

- POSSONO PERO' PORTARE AD ACCEDERE A BLOCCI INUTILI
- NON SONO EFFICACI PER RICERCHE CON CONDIZIONI SU ALTRI ATTRIBUTI

INDICI

```
SELECT *  
FROM Noleggi  
WHERE dataNol = '15-Mar-2006'
```

File ordinato rispetto a dataNol

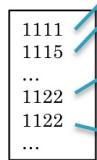
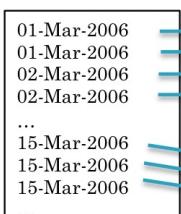
Indice con chiave di ricerca dataNol

```
SELECT *  
FROM Noleggi  
WHERE colloc = 1122
```

File non ordinato rispetto a colloc
Indice con chiave di ricerca colloc

File ordinato su dataNol

colloc	dataNol	codCli	dataRest
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006
1117	02-Mar-2006	6635	06-Mar-2006
1118	02-Mar-2006	6635	06-Mar-2006
1111	04-Mar-2006	6642	05-Mar-2006
1119	08-Mar-2006	6635	10-Mar-2006
1120	08-Mar-2006	6635	10-Mar-2006
1116	08-Mar-2006	6642	09-Mar-2006
1118	10-Mar-2006	6642	11-Mar-2006
1121	15-Mar-2006	6635	18-Mar-2006
1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1112	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1116	21-Mar-2006	6610	?
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1113	22-Mar-2006	6642	?



UN INDICE E' UN MULTI-INSIEME DI COPPIE DEL TIPO (k_i, r_i) DOVE:

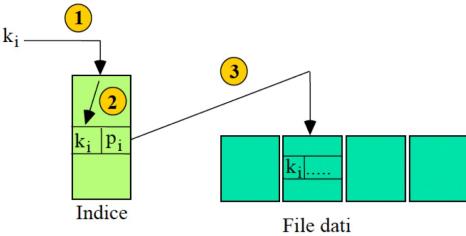
- k_i E' UN VALORE PER LA CHIAVE DI RICERCA SU CUI L'INDICE E' COSTRUITO
- r_i PUO' ASSUNGERE DIVERSE FORME, PER IL MIGLIORAMENTO ASSURGENTI CHE SIA UN PUNTAFOGLIO A UN RECORD CON VALORE DI CHIAVE k_i
→ UN RID O AL LIVELLO UN PID

UN INDICE OCCUPA MENO SPAZIO RISPETTO AL FILE DAT

→ RISOLVE PROBLEMI DI ACCESSO A BLOCCI INUTILI

- Accedere all'indice
- Ricercare la coppia (k_i, p_i)
- Accedere alla pagina dati relativa

```
SELECT *
FROM R
WHERE A = ki
```



A chiave di ricerca

→ CREARE DEGLI INDICI PER RENDERE EFFICIENTI ALCUNE OPERAZIONI

Ha come svantaggio quello di rendere altre altre meno efficienti

UN'INTERROGAZIONE NON C'È SOLO RALENTATA DALLA PRESENZA
DI INDICI

→ RENDERÀ PIÙ COSTOSI GLI AGGIORNAMENTI, POICHÉ
A SEGUIMENTO DI UN AGGIORNAMENTO DEI DATI TUTTI GLI INDICI
DEVONO ESSERE AGGIORNATI

```
DELETE FROM Noleggio
WHERE dataNol = '22-Mar-2006'
AND colloc = 1122
```

ESEMPIO

File ordinato su dataNol

Indice su dataNol

01-Mar-2006
01-Mar-2006
02-Mar-2006
...
22-Mar-2006
...

Indice su colloc

1111
1115
...
1122
...

Indice su codcli

1111 01-Mar-2006 6635 02-Mar-2006
1115 01-Mar-2006 6635 02-Mar-2006
1117 02-Mar-2006 6635 06-Mar-2006
1118 02-Mar-2006 6635 06-Mar-2006
1111 04-Mar-2006 6642 05-Mar-2006
1119 08-Mar-2006 6635 10-Mar-2006
1120 08-Mar-2006 6635 10-Mar-2006
1116 08-Mar-2006 6642 09-Mar-2006
1118 10-Mar-2006 6642 11-Mar-2006
1121 15-Mar-2006 6635 18-Mar-2006
1122 15-Mar-2006 6635 18-Mar-2006
1113 15-Mar-2006 6635 18-Mar-2006
1129 15-Mar-2006 6635 20-Mar-2006
1119 15-Mar-2006 6642 16-Mar-2006
1126 15-Mar-2006 6610 16-Mar-2006
1112 16-Mar-2006 6610 18-Mar-2006
1114 16-Mar-2006 6610 17-Mar-2006
1128 18-Mar-2006 6642 20-Mar-2006
1124 20-Mar-2006 6610 21-Mar-2006
1115 20-Mar-2006 6610 21-Mar-2006
1124 21-Mar-2006 6642 22-Mar-2006
1116 21-Mar-2006 6610 ?
1117 21-Mar-2006 6610 ?
1127 22-Mar-2006 6635 ?
1125 22-Mar-2006 6635 ?
1122 22-Mar-2006 6642 ?
1113 22-Mar-2006 6642 ?

INDICI PRIMARI E SECONDARI

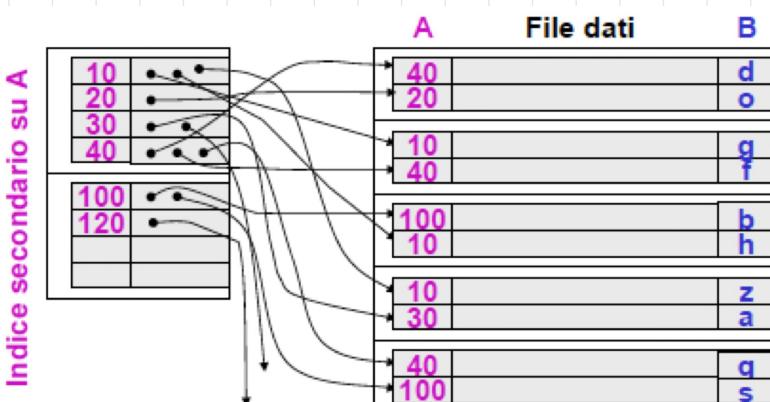
• **PRIMARIO** Non ha coppie che condividono lo stesso valore. Poi c'è un tipo di indice costruito su un attributo chiave.

• **SECONDARIO** Ha coppie che condividono lo stesso valore.

È IMPORTANTE INTRODURRE QUESTA DIFFERENZA DEGLI INDICI CONOGLIOPPI (VALORE CHIAVE DI RICERCA, PUNTATORI). Poi c'è l'organizzazione degli indici secondari pro' essere più efficienti.

→ è utile avere più copie con lo stesso valore di chiave di ricerca.

⇒ RAGGRUPPARE TUTTE LE COPPIE CON LO STESSO VALORE DI CHIAVE IN UNA LISTA DI PUNTATORI.



TIPI DI INDICI

DIFERISCONO NEL MODO IN CUI ORGANIZZANO L'INSIENE DI COPPIE (k_i, r_i)

↳ PUNTATORE O LISTA DI PUNTATORI

• INDICI ORDINATI

LE COPPIE VENGONO MANTENUTE ESPlicitamente, SALVATE SU UN FILE E ORDINATE RISPETTO AI VALORI DI CHIAVE k_i

• INDICI HASH

LE COPPIE NON VENGONO MEMORIZZATE SU FILE, MA VENGONO CALCOLATE ATTRAVERSO GLI USO DI UNA FUNZIONE HASH

$$h(k_i) = r_i$$

INDICI ORDINATI / AD ALBERO

• SE L'INDICE E' PICCOLO, PUO' ESSERE TENUTO IN MEMORIA PRINCIPALE

↳ SPESO PERÒ E' NECESSARIO TENERLO SU DISCO E LA SCANSIONE DELL'INDICE PUO' RICHIEDERE PARECCHI TRASFERIMENTI DI BLOCCHI

• VIENE QUINDI UTILIZZATA UNA STRUTTURA MULTILIVELLO CHE PERMETTE DI ACCEDERE VELOCEMENTE ALLE COPPIE DELL'INDICE

↳ L'INDICE ASSUME UNA STRUTTURA AD ALBERO IN CUI OGNI NODO DELL'ALBERO CORRISPONDE A UN BLOCCO DISCO

→ INDICI AD ALBERO

ORGANIZZAZIONI AD ALBERO BILANCIOATO, UTILIZZATE CORE STRUTTURE DI INDICAZIONE PER DATI IN MEMORIA SECONDARIA

REQUISITI FONDAMENTALI:

• BILANCIMENTO

DEVE ESSERE BILANCIOATO RISPETTO AI BLOCCHI E NON AI SINGOLI NODI

→ IL COSTO DI I/O E' DETERMINATO DAL NUMERO DI BLOCCHI ACCEDUTI

• OCCUPAZIONE MINIMA

PER EVITARE UN SOTTO-UTILIZZO DELLA MEMORIA, BISOGNA cercare di OCCUPARE BENE I BLOCCHI E NON USARE BLOCCHI POCHISSIMI (SPAZIO VUOTO : SPERCO DI MEMORIA) → ottimizzare ratios spazio - disco

• EFFICIENZA DI AGGIORNAMENTO

LA STRUTTURA CHE SI USA DEVE ESSERE EFFICIENTE ANCHE PER LE OPERAZIONI DI AGGIORNAMENTO
DEVONO AVERE UN COSTO LIMITATO ↳

IL PIÙ FAMOSO TIPO DI INDICE AD ALBERO È Quello del **B⁺ TREE**

- OGNI NODO CORRISPONDE AD UN BLOCCO

↳ Memorizzano COPIE di VALORI CHIAVE DI RICERCA - Puntatori si fanno

- LE OPERAZIONI DI RICERCA, INSERIMENTO E CANCELLAZIONE HANNO UN COSTO

↳ Lineare non l'altezza dell'albero è essendo **BILANCIAZIO**

↳ DIPENDENTE DAL NUMERO DI VALORI DELLA CHIAVE DI RICERCA

ALTEZZA LOGARITMICA NEI VALORI DELLA CHIAVE DI RICERCA DELL'INDICE, IL COSTO SARÀ SEMPRE $h = O(\log N)$
 $N = \text{N° VALORI CHIAVE DI RICERCA}$

IL NUMERO MASSIMO $m-1$ DI ELEMENTI MEMORIZZABILI IN UN NODO È L'UNICO PARAMETRO DI PENDENTE DALLE CARATTERISTICHE DI MEMORIA (DIMENSIONE DEL BLOCCO)

OGNI BLOCCO UTILIZZATO PER MEMORIZZARE GLI INDICI È GARANTITO NEL MIGLIORAMENTO DI ALTEZZA A SO%

LE COPPIE (k_i, r_i) SONO CONTENUTE NEI NODI FOGLIA E LE FOGLIE SONO COLLEGATE A LISTA MEDIANTE PUNTATORI (PID) PER FAVORIRE RICERCHE DI TIPO RANGE

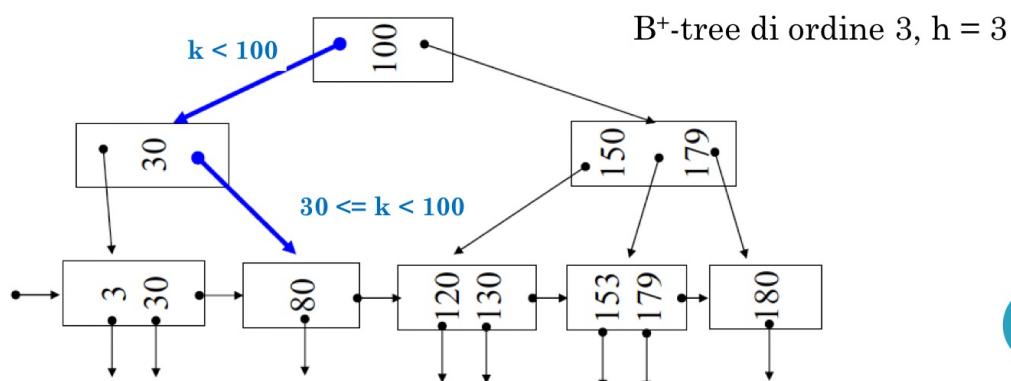
→ LA RICERCA DI UNA CHIAVE DEVE INDIVIDUARE UNA FOGLIA

• I NODI INTERNI MEMORIZZANO DEI SEPARATORI (ANCH'ESSI VALORI DELLA CHIAVE DI RICERCA K_i) LA CUI FUNZIONE È DETERMINARE IL GIUSTO CAMMINO NELLA RICERCA DI UNA CHIAVE

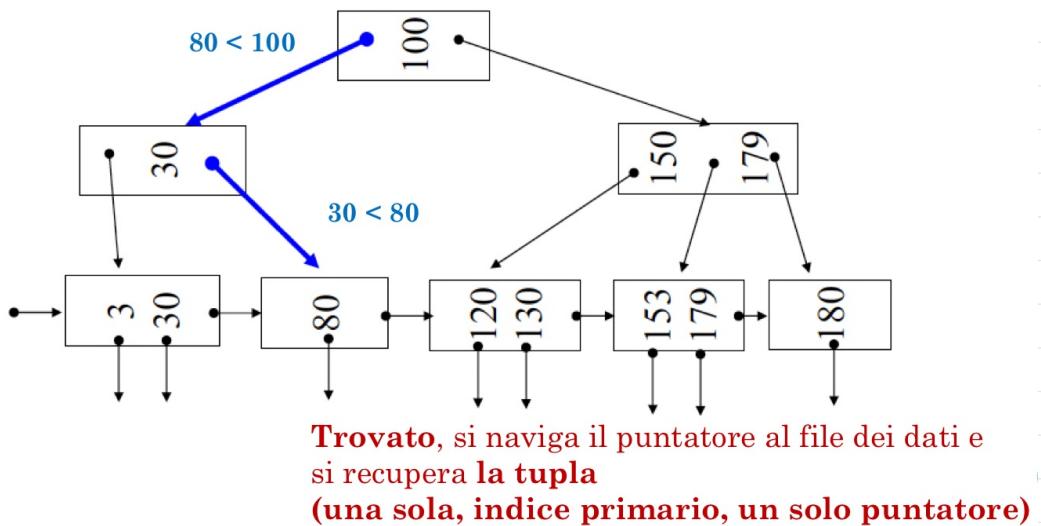
↳ IMPUNGA UNA PARZIALE DUPLICAZIONE DELLE CHIAVI (ALCUNE SONO PRESENTI IN NODI FOGLIA E NODI INTERNI)

Un B⁺-albero di **ordine m** ($m \geq 3$) è un **albero bilanciato** che soddisfa le seguenti proprietà:

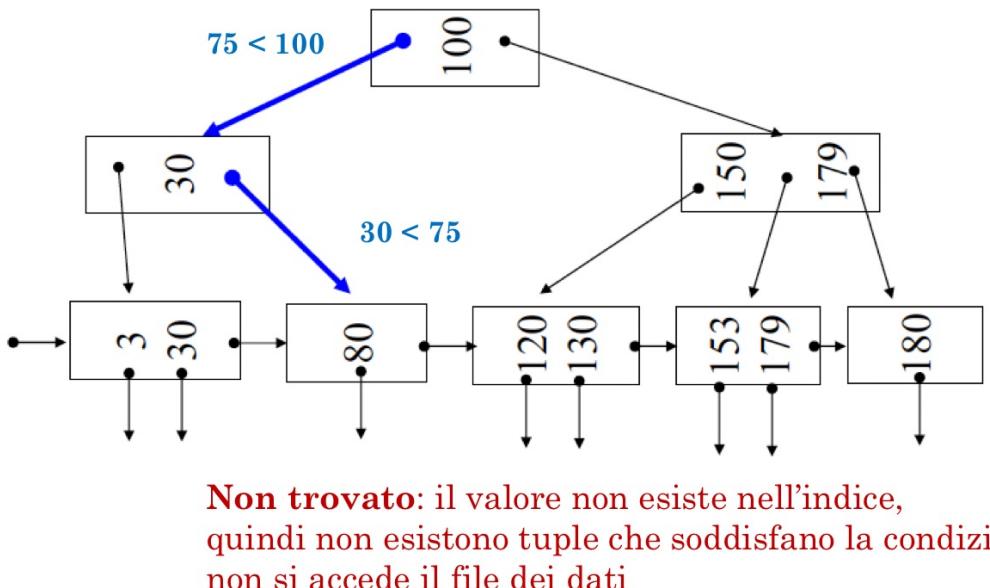
- ogni nodo contiene al più $m - 1$ elementi **OCCUPAZIONE MASSIMA**
- ogni nodo, tranne la radice, contiene almeno $\lceil m/2 \rceil - 1$ elementi, la radice può contenere anche un solo elemento **OCCUPAZIONE MINIMA**
- ogni nodo non foglia contenente j elementi ha $j + 1$ figli



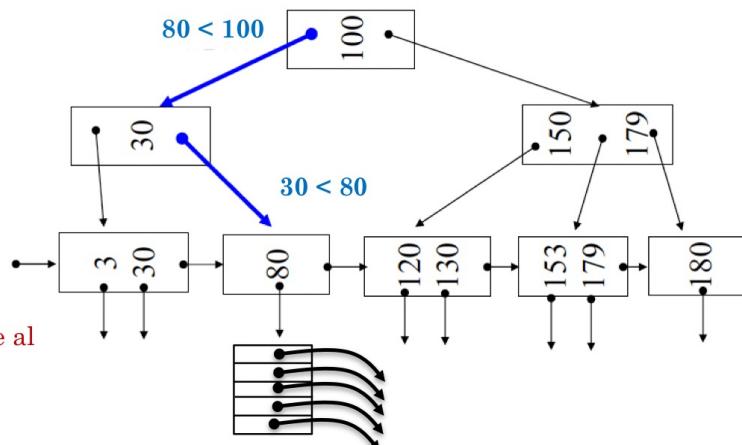
- o $R(\underline{K}, B, C)$
- o Chiave di ricerca per indice primario: **K**
- o Ricerca tuple con **$K = 80$**



- o $R(\underline{K}, B, C)$
- o Chiave di ricerca per indice **primario**: **K**
- o Ricerca tuple con **$K = 75$**



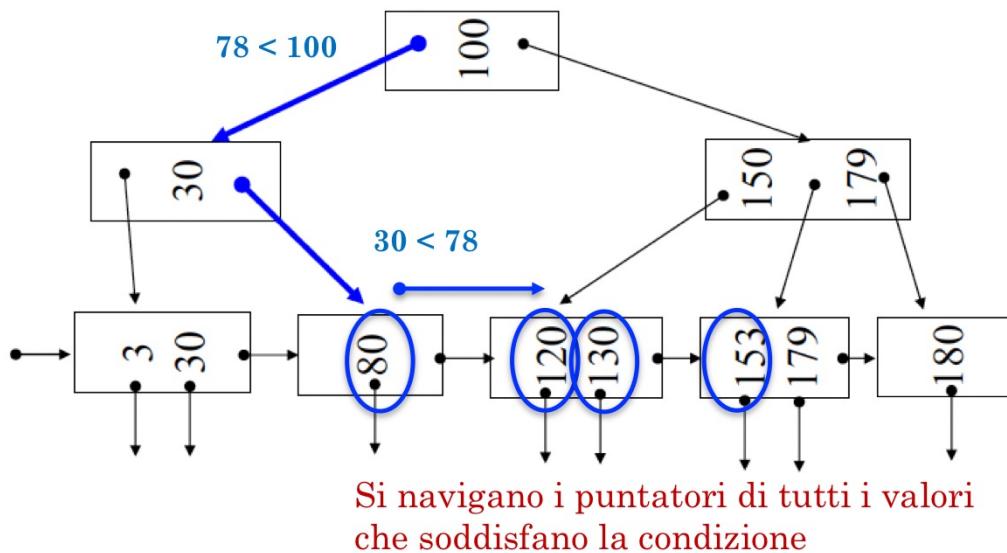
- o $R(K, B, C)$
- o Chiave di ricerca per indice secondario: **K**
- o Ricerca tuple con **K = 80**



Trovato, si naviga il puntatore al file dei dati e si recuperano le tuple (anche più di una, indice secondario, numero arbitrario di puntatori, inseriti in un blocco puntato dal puntatore nell'indice)

Ricerca per Intervallo

- o $R(\underline{K}, B, C)$
- o Chiave di ricerca per indice primario: **K**
- o Ricerca tuple con $78 \leq K \leq 154$



Si navigano i puntatori di tutti i valori che soddisfano la condizione

Vale anche per le operazioni di inserimento e cancellazione, con la differenza che si possono andare ad alterare vincoli di riempimento minimo e massimo di ciascun blocco e questo potrebbe portare ad una reazione a catena
 → prima si effettua la ricerca, dopodiché operazioni ai nodi dal basso verso l'alto per continuare a far valere le proprietà

VARIANTI

- **B-TREE** = MIGLIORA L'OCCUPAZIONE DI MEMORIA E I PUNTATORI AI DATI POSSONO ANCHE ESSERE CONTENUTI NEI NODI INTERNI

INDICI AD ALBERO CLUSTERIZZATI E NON CLUSTERIZZATI

Se il file dei dati è ordinato
rispetto alla chiave di ricerca
In caso contrario

UN FILE DEI DATI ORDINATO È SEMPRE ASSOCIAVO A UN INDICE AD ALBERO CLUSTERIZZATO

→ VIENE SFRUITATO L'ORDINAMENTO DELLE FOGLIE DELL'INDICE PER ORDINARE IL FILE DEI DATI

→ CON LA PRESENZA DELL'INDICE LE OPERAZIONI DI INSERIMENTO, CANCELLAZIONE E AGGIORNAMENTO NEL FILE ORDINATO SONO FACILITATE

• AL PIÙ UN CLUSTER PER TABELLA (Indipendentemente dal tipo)

Indice clusterizzato primario



Indice clusterizzato secondario: è sufficiente associare a ciascun valore della chiave di ricerca un unico puntatore

INDICI AD ALBERO SU SINGOLO ATTRIBUTO E MULTI-ATTRIBUTO

• **SINGOLO ATTRIBUTO** LA CHIAVE DELL'INDICE È FORMATA DA UN SINGOLO ATTRIBUTO

• **MULTIATTRIBUTO** LA CHIAVE DELL'INDICE È COMPOSTA DA PIÙ DI UN ATTRIBUTO

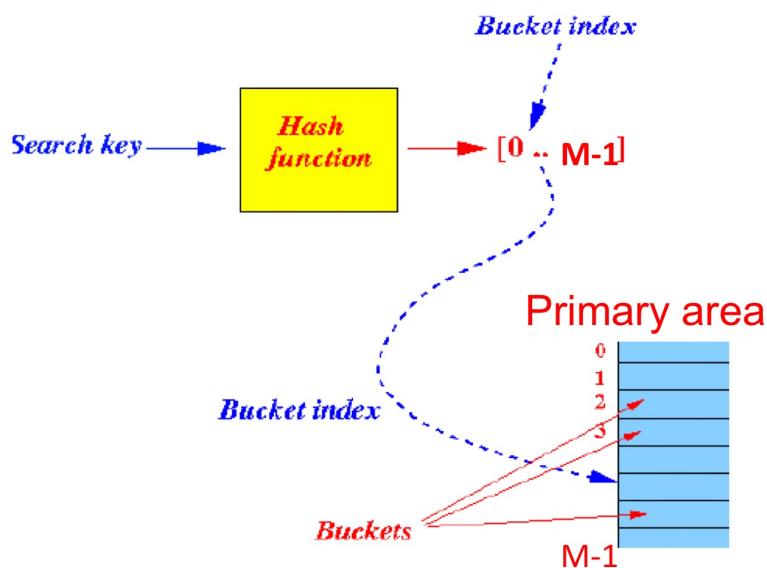
• POSSONO SUPPORTARE PIÙ INTERROGAZIONI DI UN INDICE A SINGOLO ATTRIBUTO

• Deve però essere AGGIORNATO PIÙ FREQUENTEMENTE ED È DI DIMENSIONE MAGGIORE RISPETTO AD UN INDICE SU SINGOLO ATTRIBUTO

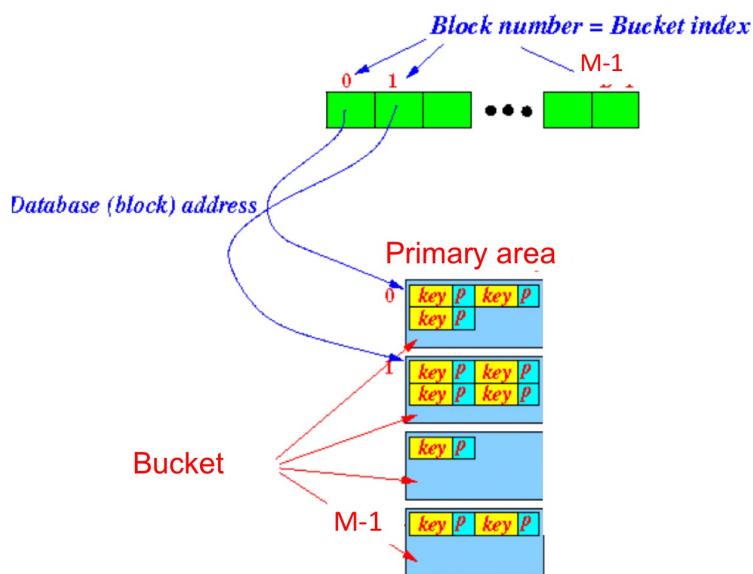
INDICI HASH

INDEX IN CUI LE COUPLE (k_i, v_i) NON VENGONO MEMORIZZATE ESPlicitamente, ma vengono calcolate utilizzando una funzione hash \rightarrow A differenza degli index ad albero che vengono salvati su file

• AD OGNI VALORE DELLA FUNZIONE HASH CORRISPONDE UN INDIRIZZO IN AREA PRIMARIA



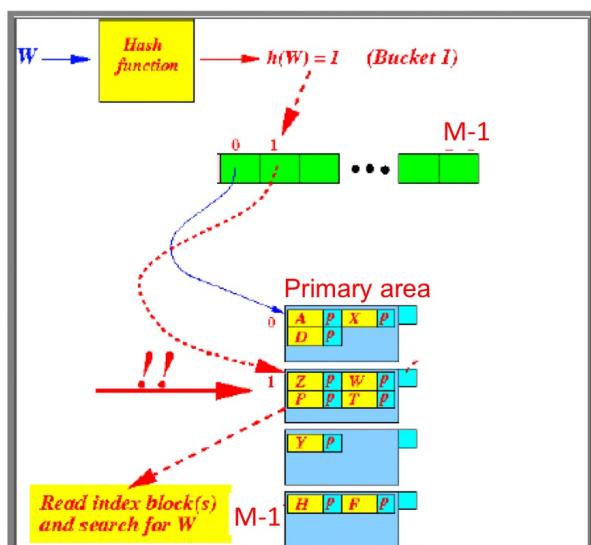
Bucket in blocchi vicini all'interno da cui effettuare la ricerca



L'ASSOCIAZIONE TRA I VALORI DELLA FUNZIONE HASH E GLI INDIRIZZI SU DISCO AVVIENE TRAMITE UN **Bucket index**

INDICI HASH - RICERCA PER UGUAGLIANZA

1. Supponiamo di cercare le tuple della relazione $R(K, B, C)$ con $B = w$, con indice hash su attributo B
2. Si calcola $H(w)$
3. Se $H(w) = 1$, tramite il bucket index, si determina l'indirizzo corrispondente al bucket (supponiamo 1)
4. Si accede il bucket e, record per record, si cercano le tuple t con $t.B = w$



UN INDICE HASH NON SUPPORTA RICERCHE PER INTERVALLO

→ UNA FUNZIONE HASH NON MANTIENE L'ORDINE: TUPLE CON VALORI CONTIGUI PER LA CHIAVE DI RICERCA POSSONO ESSERE MEMORIZZATI IN BUCKETS DIVERSI

INDICI HASH - INSERIMENTO E TRABOCCHI

PER INSERIRE UN RECORD, SI PROCEDE COME PER LA RICERCA PER UGUALANZA, IDENTIFICANDO IL BUCKET IN CUI IL RECORD DEVE ESSERE INSERITO

LA CAPACITÀ c DEI BUCKETS INDICA IL NUMERO DI RECORD CHE POSSONO ESSERE ALLOCATI NELLO stesso BUCKET

→ IN GENERALE I BUCKETS HANNO LA STESSA CAPACITÀ

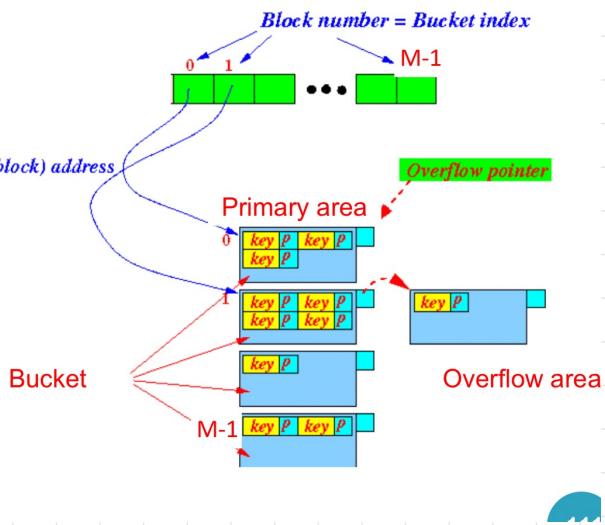
Se il Bucket

- HA ANCORA SPAZIO SI INSERISCE IL RECORD
- NON HA PIÙ SPAZIO SI GENERA UN TRABOCCHIO (OVERFLOW)

- Supponiamo che ogni bucket abbia capacità $c = 4$ (può contenere 4 record)

- Supponiamo adesso di inserire un nuovo record con chiave k tale che $H(k) = 1$

- Il bucket 1 è già pieno, quindi è necessario usare un blocco in area di overflow per memorizzare il nuovo record



→ LA SUA PRESENZA PUÒ RICHIEDERE L'USO DI UN'AREA DI MEMORIA SEPARATA
→ AREA DI OVERFLOW

INDICI HASH - AREA PRIMARIA E AREA DI OVERFLOW

L'AREA PRIMARIA VIENE ALLOCATA AL MOMENTO DELLA CREAZIONE DELL'INDICE HASH, MENTRE QUELLA DI OVERFLOW SOLO QUANDO CE N'E' BISOGNO

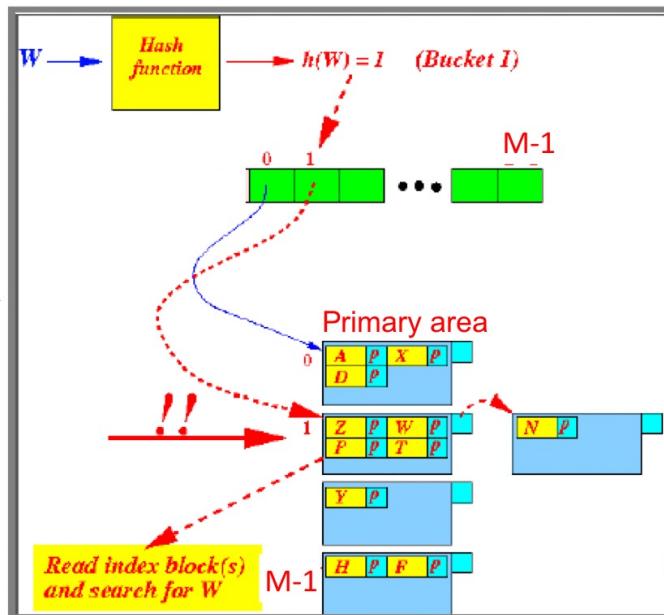
→ LA PRIMA VIENE CREATA IN TIEMPO DA OTTIMIZZARE IL TEMPO DI LATENZA PER ACCESSO (BUCKETS MEMORIZZATI IN MODO CONTIGUO), MENTRE PER QUELLA DI OVERFLOW HA I BLOCCI MEMORIZZATI DOVE SI PUÒ ALLOCARE.
→ MAGGIORE LATENZA DI ACCESSO

INDICI HASH PERFETTI → SE PER UN CERTO NUMERO DI BLOCCI NON GENERA TRABOCCHI

PUÒ SEMPRE ESSERE DEFINITA DISPOSENDO DI UN'AREA PRIMARIA CON CAPACITÀ COMPLESSIVA PARI AL NUMERO DEI RECORD DA MEMORIZZARE

INDICI HASH – DI NUOVO RICERCA PER UGUAGLIANZA

- Supponiamo di cercare le tuple della relazione $R(K, B, C)$ con $B = w$, con indice hash su attributo B
- Si calcola $H(w)$
- Se $H(w) = 1$, tramite il bucket index, si determina l'indirizzo corrispondente al bucket (supponiamo 1)
- Si accede il bucket e, record per record, si cercano le tuple t con $t.B = w$
- Si deve analizzare sia la parte del bucket in area primaria sia quella in area di overflow



CANCELLAZIONE

- CERCO IL RECORD
- LO CANCELLO → SE SI TROVA IN UN'AREA DI OVERFLOW, POTREBBE DETERMINARE LA CANCELLAZIONE DELL'AREA SE ULTRIMO RECORD NELL'AREA

COSTO

- IN ASSENZA DI OVERFLOW, IL COSTO DI ACCESSO A INDICE È COSTANTE
- IN PRESENZA DI OVERFLOW, LE PRESTAZIONI NON SONO FACILMENTE DETERMINABILI. DIPENDE DALLA DIMENSIONE DEL'AREA DI OVERFLOW E DA DOVE SI TROVANO IN MEMORIA I BLOCCHI DEDICATI AD essa

CREAZIONE INDICI DI HASH

BISOGNA SPECIFICARE:

LA FUNZIONE H PER LA TRASFORMAZIONE DELLA CHIAVE

- IL METODO DELLA GESTIONE DEI TRABOCCHI
- IL FATTORE DI CARICA d , UN VALORE TRA 0 E 1 CHE INDICA QUANTO SI INTENDONO MANTENERE PIENI I BUCKER

→ c'è UN'APPLICAZIONE SUGGERITIVA H DELL'INSIEME DELE POSSIBILI CHIAVI ALL'INSIEME $0 \dots M-1$ DEI POSSIBILI INDIRIZZI CHE VERIFICA LE SEGUENTI PROPRIETÀ:

- DISTRIBUZIONE UNIFORME** DELLE CHIAVI NELLO SPAZIO DEGLI INDIRIZZI (OGNI INDIRIZZO DEVE ESSERE GENERATO CON LA STESSA PROBABILITÀ)

- DISTRIBUZIONE CASUALE** DELLE CHIAVI (EVENTUALI CORRELAZIONI TRA I VALORI DELLE CHIAVI NON DEVONO TRADURSI IN CORRELAZIONI TRA GLI INDIRIZZI GENERATI)

TALI PROPRIETÀ DIPENDONO DELL'INSIEME DELLE CHIAVI SU CUI SI OPERA E QUANDO NON ESISTE UNA FUNZIONE UNIVERSALE OTTIMA

→ IN GENERE LE FUNZIONI HASH OPERANO SU **INSIEMI DI CHIAVI INTERE**, SE I VALORI DELLE CHIAVI SONO STRUNGHE ALFANUMERICHE, SI PUÒ ASSOCIARE IN MODO UNIVOCO AD OGNI CHIAVE UN NUMERO INTERO, PUNT DI APPLICARE LA TRASFORMAZIONE

LA FUNZIONE USATA DA TUTTI E' LA FUNZIONE BASETA SUL METODO DELLA DIVISIONE

$$H(k) = k \bmod M$$

Lo resto della divisione intera

→ AFFINCHÉ H DISTINGUISCA BENE, M DEVE ESSERE PRIMO OPPURE NON PRIMO CON NESSUN FATTORE PRIMO MINORE DI 20

INDICI HASH CLUSTERIZZATI E NON CLUSTERIZZATI

CLUSTERIZZATO SE I RECORD CHE CONDIVIDONO LO STESSO VALORE PER LA CHIAVE DI RICERCA SONO MEMORIZZATI IN POSIZIONI ADIACENTI NEL FILE DEI DATI

NON CLUSTERIZZATO IN CASO CONTRARIO

UN FILE DEI DATI DI TIPO HASH E' SEMPRE ASSOCIAZO A UN INDICE HASH CLUSTERIZZATO

Le operazioni di inserimento, cancellazione e aggiornamento nel file ordinato sono facilitate dall'uso dell'organizzazione hash

In presenza di un indice hash clusterizzato, l'organizzazione prima corrisponde ai record memorizzati nell'area primaria + i record memorizzati nell'area di overflow

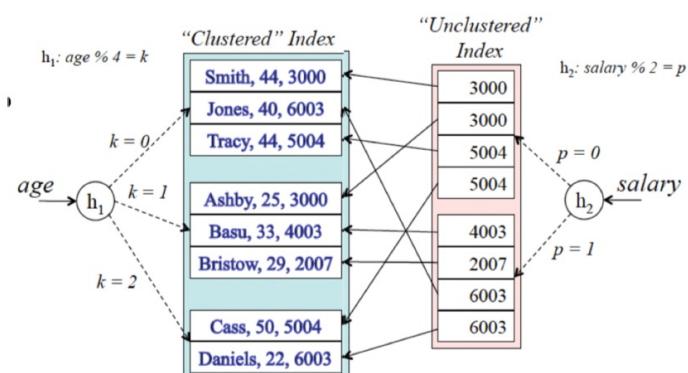
• AL PIÙ UN INDICE CLUSTERIZZATO PER TABELLA (INDIPENDENTEMENTE DAL TIPO)

INDICI HASH CLUSTERIZZATI E NON CLUSTERIZZATI

◦ Employee(Name, age, salary)

- Indice hash rispetto a age clusterizzato
- struttura discussa finora
- tutti i record che corrispondono allo stesso valore hash sono memorizzati vicini, cioè nello stesso bucket
- tutti i record con lo stesso valore per la chiave di ricerca sono memorizzati vicini, cioè nello stesso bucket

- Indice hash rispetto a salary non clusterizzato
- I record con lo stesso valore per la chiave di ricerca o, più in generale, che corrispondono allo stesso valore hash, possono essere memorizzati in bucket diversi
- Serve un livello in più → indice multilivello



CONFRONTO TRA INDICI AD ALBERO E HASH

• PER SELEZIONI CON CONDIZIONI DI UGUAGLIANZA E' PREFERIBILE UN INDICE HASH

• PER SELEZIONI DI TIPO RANGE E' PREFERIBILE UN INDICE AD ALBERO, POICHÉ GLI INDICI AD HASH NON MANTENGONO L'ORDINE

→ INFATTI:

• LA SCANSIONE DI UN INDICE HA UN COSTO PROPORZIONALE AL LOGARITMO DEL NUMERO DI VALORI IN R PER AI

• IN UNA STRUTTURA HASH IL TEMPO DI RICERCA E' INDEPENDENTE DALLA DIMENSIONE DEGLI ELIC DATI

CONDIZIONI DI UGUAGLIANZA

```
SELECT A1, A2, .....An  
FROM R  
WHERE Ai=C
```

SELEZIONE DI TIPO RANGE

```
SELECT A1, A2, .....An  
FROM R  
WHERE C1 < Ai < C2
```