

23/09/2020 ~~de Nitto~~

- SISTEMA INFORMATIVO → gestisce il flusso di informazioni all'interno di un ambito (per esempio, di un'azienda, un comune, ecc.).

È un insieme di ^{e di procedura}insieme per: RACCOLTA, ARCHIVIAZ., ELABORAZ., SCAMBIO delle informazioni necessarie alle attività.

↳ OPERATIVE
↳ PROGRAMMAZ. / CONTROLLO
↳ PIANIFICAZ. STRATEGICA

- SISTEMA INFORMATICO → è il cuore del sistema informativo (è l'insieme delle tecnologie informatiche a supporto delle attività di un'organizzazione).

- SISTEMA INFORMATICO AUTOMATIZZATO ≈ SISTEMA INFORMATICO

↳ la parte del sistema informativo in cui le informazioni sono raccolte, elaborate, archiviate e scambiate usando un sistema informatico

Definizione:

Un DATO è una rappresentazione originaria, non interpretata, di un evento o di un fenomeno effettuata attraverso dei simboli (o di un'altra forma di rappresentazione espressiva) legati a un supporto.

Definizione:

Una BASE DI DATI (BD) ^{de}rappresenta un aspetto del mondo reale che è di interesse per qualche specifico scopo, ed è un insieme di dati coerenti e con un significato preciso per un particolare insieme di utenti.

Definizione:

Un DATABASE MANAGEMENT SYSTEM (DBMS) è un insieme di programmi che permette la creazione di una BD, la manipolazione, l'interrogazione e la consultazione dei dati, garantendo **affidabilità, efficienza, e privatenza ed efficacia**.

→ RIDONDANZA: da la possibilità di gestire i dati in modo + efficiente (replicando le informazioni per eventualmente spezzettarle) → PROBLEMA: possibilità di incerenza, nel caso in cui lo stesso dato scontato + volte presenti + valori differenti (quanto invece non dovrebbe).

→ COLLEZIONE DI DATI GRANDE: ha dimensioni > grandezza della memoria centrale (de t.
t.
t.
N.
at
• 1
,

Vono essere salvati quelli in memoria secondaria).

→ DATI PERSISTENTI: sopravvivono per un tempo > tempo di vita delle applicazioni che li utilizzano.

→ DATI CONDIVISI: possono essere utilizzati da + applicazioni diverse e + utenti diversi, per cui devono essere attivi:

- Meccanismi di AUTORIZZAZIONE

- Controllo della CONCORRENZA

ESEMPIO di AUTORIZZAZIONE: "L'utente A è autorizzato a leggere e modificare tutti i dati X".

"L'utente B è autorizzato a leggere i dati X e modificare i dati Y".

Questo MECCANISMO DI AUTORIZZAZIONE PERMETTE ANCHE LA PRIVATEZZA DEI DATI (i.e. l'utente A non può accedere ai dati Y).

Affinché un DBMS garantisca l'**AFFIDABILITÀ** di una base di dati, essa deve poter funzionare correttamente anche nonostante malfunzionamenti del sistema di tipo hardware o software.

Definizione:

Una TRANSAZIONE è un insieme di operazioni da considerare indivisibile e cioè eseguito anche in presenza di concorrenza e con effetti definitivi.

ESEMPIO: Consideriamo due transazioni t_1, t_2 :

$t_1: R(x), x = x - 200, W(x)$

LETTORE di x

SCRITTURA di x

$t_2: R(x), x = x - 300, W(x)$

LETTURA: passaggio del dato x nella memoria principale

SCRITTURA: scrittura del dato x all'interno della base di dati

Supponiamo che t_1, t_2 vengano effettuate concorrentemente, e supponiamo che inizialmente $x = 1000$.

Se sia t_1 , sia t_2 leggono inizialm. $x = 1000 \dots$

$t_1 \rightarrow x = 800$

} l'ultima transazione che termina determina il risultato
finale di x

$t_2 \rightarrow x = 900$

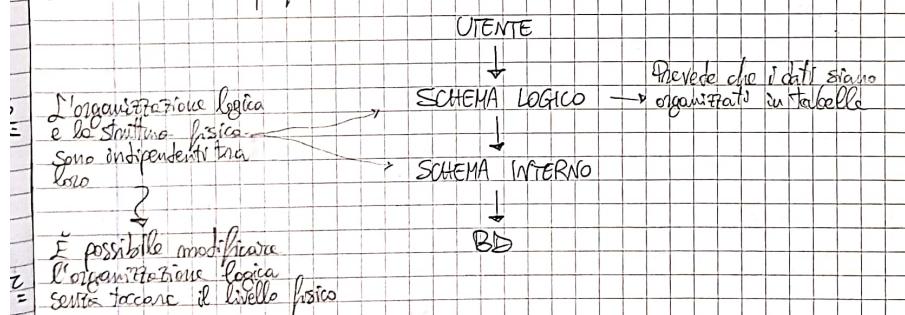
↓

Nell'ipotesi in cui t_2 termina immediatamente dopo, la transazione t_2 non ha alcun effetto !!

- Concetto di COMMIT → consiste in meccanismi che garantiscono il risparmio della base di dati a seguito di un quanto.

Un DBMS deve garantire **EFFICIENZA** sia dal punto di vista **SPAZIALE** che **TEMPORALE**. Un altro fattore chiave è l'**EFFICACIA**, che consiste in un buon modo di riscavare guadagni nella profondità di risorse, e cose analoghe che possono riguardare un'azienda che utilizza basi di dati.

Architettura semplificata di un DBMS:



24/09/2020 scritto

Il MODELLO LOGICO si basa sul MODELLO DI DATI.

Definizione:

- Un MODELLO DEI DATI è un tipo di astrazione dei dati che permette di definire le proprietà degli oggetti d'interesse e le relazioni fra questi, nascondendo i dettagli dell'implementazione.

Ci sono 2 tipi di modello dei dati:

→ MODELLI CONCETTUALI

→ MODELLI LOGICI

Lo scopo dei modelli concettuali è quello di descrivere i concetti alla base del mini-mondo costituito dalla BD e dal dominio di interesse.

Il modello concettuale è relativo al 1° STADIO ed è indipendente da quello logico. Un esempio di modello concettuale è l'^{tipico} ENTITY-RELATIONSHIP.

Il modello logico si basa su un modello RELAZIONALE, in cui i dati vengono RELAZIONATI tramite una tabella (i.e. una riga della tabella mette in relazione i vari dati tra loro).

Nel modello logico si distinguono lo SCHEMA e l'ISTANZA.

MODELLO INTENZIONALE

MODELLO ESTENSIONALE
(mostra i dati)

ESEMPIO DI SCHEMA:

INSEGNANTE	CORSO	AULA	ORA
------------	-------	------	-----

La sola primissima riga di una tabella

SQL
tivo

→
→

JL
dell

Lar
→ LE TRA
Ree

1)

2)

3)

ESEMPIO DI Istanza:

Tutte le righe con i dati veri e propri

→ È variabile nel tempo

→ Gli SCHEMI ESTERNI sono degli schemi logici che sono finalizzati a un determinato gruppo di utenti (cioè magari devono visualizzare solo alcuni dati tra tutti). Sono tecnicamente chiamati VISTE.

Linguaggi per basi di dati:

→ LINGUAGGI TESTUALI INTERATTIVI (SQL) → è un linguaggio di manipolazione dei dati

→ COMANDI IMMERSI IN UN LINGUAGGIO OSPITE (Pascal, Java, C,...)

→ COMANDI SQL IMMERSI IN UN LINGUAGGIO AD ALOC

→ CON INTERFAZIE AMICHEVOLI (senza linguaggio testuale)

→

De
Uu
Inte

→
→

25

NOTA

1

SQL è un linguaggio chiaro nel fornire le informazioni che vogliamo (è molto esplicativo), ma molto meno nel mostrare come internamente ricava i dati.

→ DATA MANIPULATION LANGUAGE (DML) → si occupa della manipolazione di dati

→ DATA DEFINITION LANGUAGE (DDL) → si occupa della definizione dei dati
(i.e. definizione di uno SCHEMA)

(o di un modello logico)

Il termine TRANSAZIONE viene usato anche per indicare programmi che effettuano delle operazioni frequenti e di interesse (QUESTO È IL PUNTO DI VISTA DELL'UTENTE).

La definizione fornita ieri è valida per il PUNTO DI VISTA DEL SISTEMA.

→ Le transazioni sono esattamente implementate in un linguaggio come C/C++ o Ada.

Requisiti della base di dati: → Un'entità di un linguaggio grafico, che permette di avere una rappresentazione non ambigua del minimo dettaglio da rappresentare e gestire.

1) PROGETTAZIONE CONCETTUALE → non tiene conto di come avviene la implementazione della BD, bensì di che cosa deve essere costruito (dei CONCETTI).

2) L'output della progettazione concettuale diventerà l'input della PROGETTAZIONE LOGICA, che invece si occupa dell'"come". → Intesa come rappresentazione del sistema che si sta costruendo.

3) Solo a valle c'è la PROGETTAZIONE FISICA, → i.e. struttura effettiva del database.

→ La progettazione concettuale produce uno SCHEMA CONCETTUALE tramite il modello ENTITY-RELATIONSHIP. Tale schema → diventerà una Tabella logica a Seguito della progettazione logica.

Nella progettazione concettuale emergono tre concetti:

- ENTITÀ
- ASSOCIAZIONI (RELATIONSHIPS)
- ATTRIBUTI

Definizione:

Un'ENTITÀ è una classe di oggetti (fatti, persone, cose...) della realtà di interesse con proprietà comuni e con esistenza "autonoma".

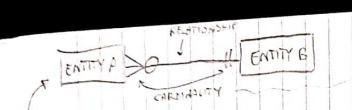
→ Un'entità comprende degli attributi (che sono propriamente le sue caratteristiche).

→ + entità possono essere relazionate tra loro (concetto di associazione/relationship).

25/09/2020 PELEGRI

NOTAZIONE CROW'S FOOT → Le entità vengono rappresentate con dei rettangoli con nome.

Queste notazioni che stiamo elencando sono dei diversi linguaggi grafici in grado di rappresentare un modello CONCETTUALE dei dati (in particolare un modello ER-Entity-Relationship).



0..1
1..M
1..1
0..1

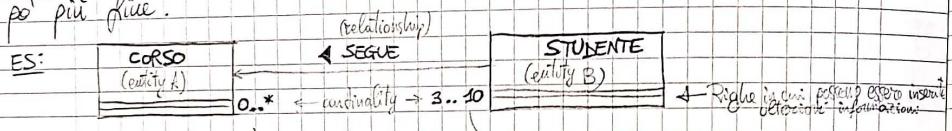
One or zero
One and only One
Zero or Many
One or Many

} Se uniche le possibili per indicare la cardinalità di una relazione

RELAZIONE → Collegamento logico tra due entità.

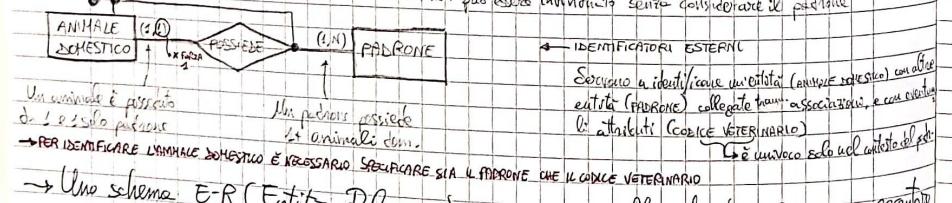
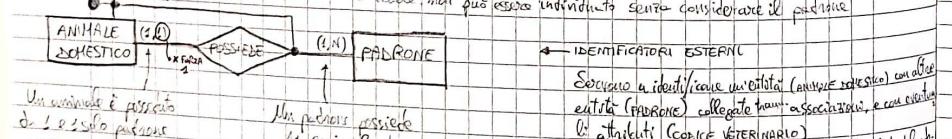
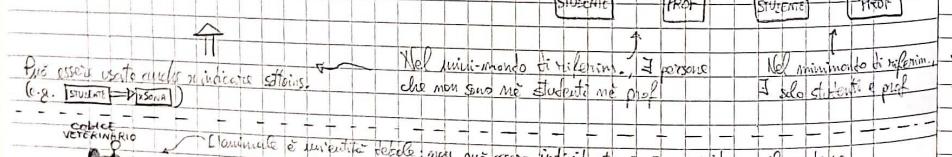
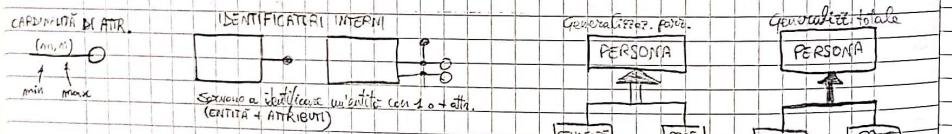
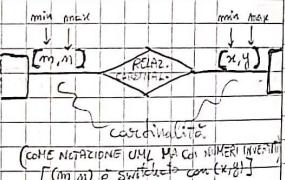
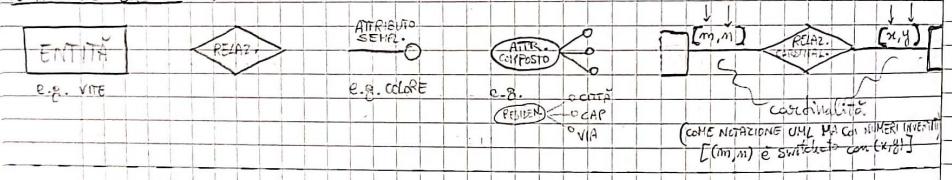
→ Può essere caratterizzato tramite la CARDINALITÀ (0, 1, MOLTI).

NOTAZIONE UML → anche qui le entità possono essere rappresentate con rettangoli e possono essere relazionati tra loro, con cardinalità specificata a grana un po' più fine.



Un studente può seguire 0 oppure + corsi
→ Un corso può essere seguito da 3 a 10 studenti

NOTAZIONE CHEN: → quello che effettivamente utilizziamo



→ Un animale è possibile che lo abbia più padroni.
→ Per identificare l'animale domestico è necessario specificare sia il padrone che il codice Veterinario.

→ Un schema E-R (Entità - Relazione) non è mai sufficiente da solo a rappresentare tutti i dettagli di un'applicazione.

→ UTILIZZO DEL LINGUAGGIO NATURALE PER DESCRIVERE IL PROGETTO

Per esempio tranne il dictionario dei dati

MA RISCHIO AMBIGUITÀ (proprio del linguaggio naturale)

Per rendere il Linguaggio naturale il meno ambiguo possibile, si utilizzano delle regole:

- VINCOLI DI INTEGRITÀ: <concetto> deve/non deve <espressione sui concetti>
- DERIVAZIONI: <concetto> si ottiene <operazione sui concetti>
- DIZIONARIO DEI DATI: è una tabella che descrive le entità con un nome, una definizione in linguaggio naturale, l'elenco di tutti gli attributi, l'elenco delle entità coinvolte con la loro cardinalità.

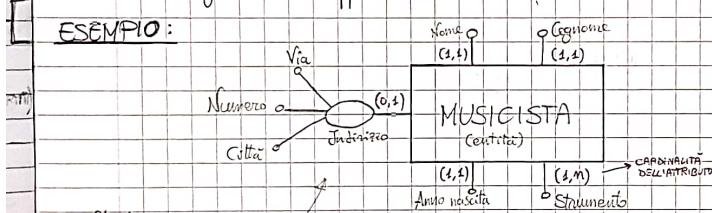
30/09/2020

DEFINIZIONE

Definizione:

Gli ATTRIBUTI sono proprietà elementari di un'entità o di una associazione, di interesse ai fini dell'applicazione.

ESEMPIO:



- CARDINALITÀ MINIMA = 0 → attr. opzionale
- CARDINALITÀ MASSIMA = 1 → attr. semplice
- CARDINALITÀ MASSIMA = n → attr. multivaleore

Le cardinalità semplici e non optionali (1..1) possono essere omesse

Qui l'indirizzo è un ATTRIBUTO COMPOSTO

→ È un attributo che raggruppa un insieme di attributi

Definizione:

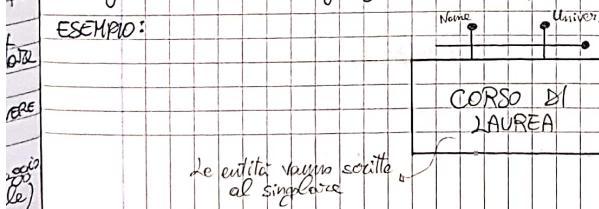
Un IDENTIFICATORE è uno "strumento" per l'identificazione univoca di un'entità.

→ Può consistere in un attributo, un insieme di attributi, ...

• IDENTIFICATORE INTERNO → identificatore che comprende attributi ^{propri} dell'entità in questione.

→ Gli attributi che fungono da identificatori sono rappresentati col pallino nero.

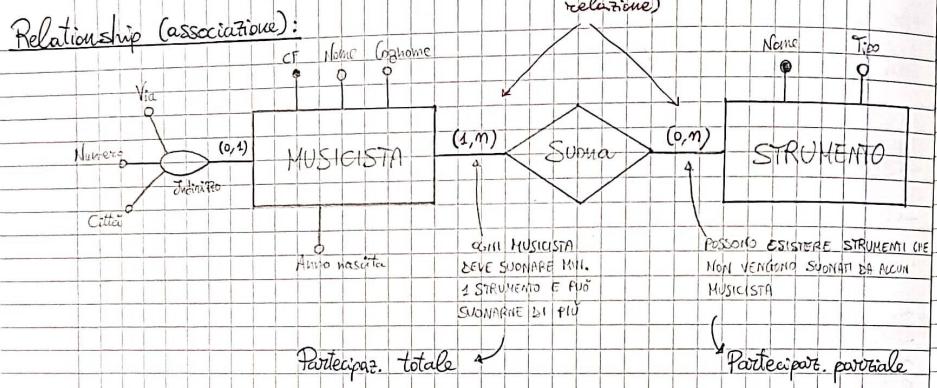
ESEMPIO:



Le entità vanno scritte al singolare

← In questo caso, se identificare univocam. il corso di laurea, bisogna specificare sia il nome, sia l'università: è una chiave composta da due attributi e viene rappresentata con una sbarretta trasvers.

Relationship (associazione):



Assoc

→ Sono

In p

Consider

→ "Suona" è un vero e proprio insieme i cui elementi fungono da associazione tra un elemento dell'insieme "MUSICISTA" e un elemento dell'insieme "STRUMENTO".

Le relationship possono essere
(in base alle cardinalità MASSIME)

MOLTI A MOLTI

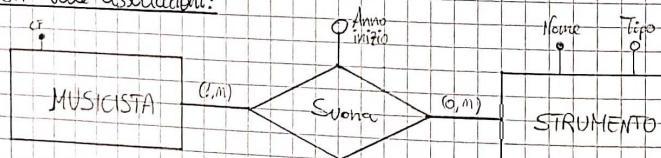
UNO A MOLTI

UNO A UNO

01/10/2020

DE NOTTO

Attributi delle associazioni:



→ Quest

(P₁, P₂

in cui

Associati

→ A differenza degli attributi di entità, non hanno lo scopo di distinguere due elementi differenti relativi alla stessa entità.

ESEMPIO:

$$\text{STRUMENTO} = \{(Chitarra, corda), (Tromba, fiato)\}$$

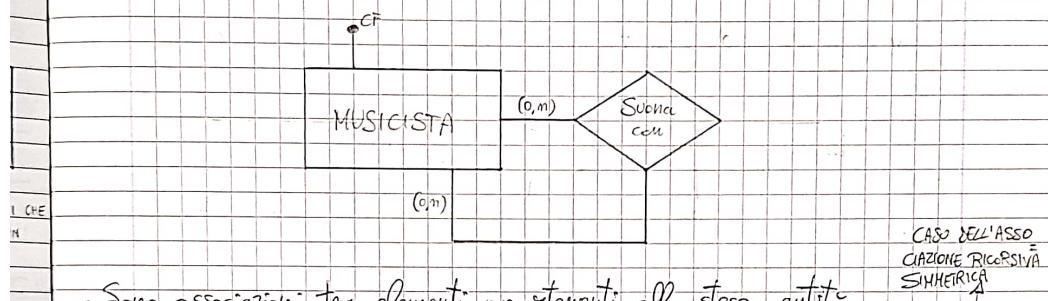
$$\text{SUONA} = \{(\text{Musicista}_1, \text{Strumento}_1, \text{Anno_inizio}_1),$$

$$(\text{Musicista}_2, \text{Strumento}_2, \text{Anno_inizio}_2),$$

$$\text{In un secondo caso: } \rightarrow \{(\text{Musicista}_1, \text{Strumento}_1, \text{Anno_inizio}_3)\}$$

In un secondo
caso
è anche posso
re in un'associa

Associazioni ricorsive:

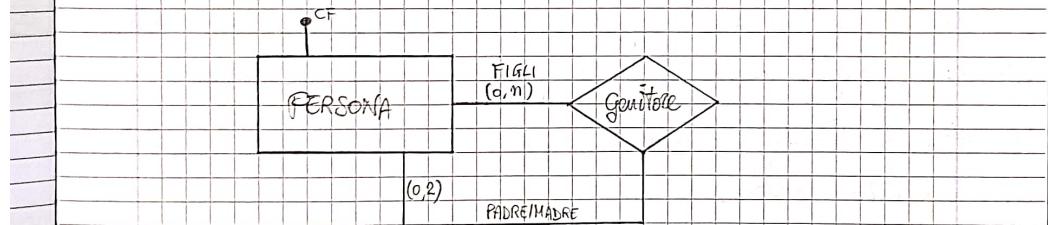


CASO DELL'ASSOCIAZIONE RICORSIVA
SIMMETRICA

→ Sono associazioni tra elementi appartenenti alla stessa entità.

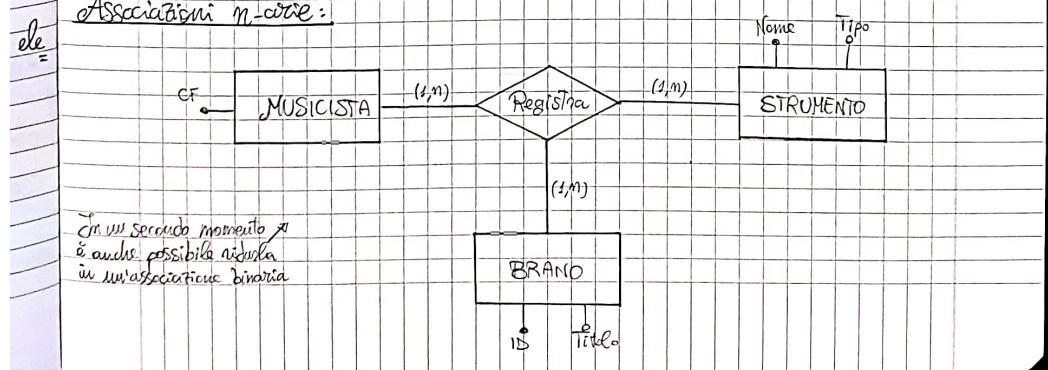
In particolare, l'associazione (m_1, m_2) è uguale all'associazione (m_2, m_1) .

Consideriamo ora quest'altro schema:
VTO.



→ Questa è un'associazione ricorsiva non simmetrica, in cui l'associazione (p_1, p_2) indica che p_1 è il padre/la madre di p_2 , ed è diversa da (p_2, p_1) , in cui p_2 sarebbe il padre/la madre di p_1 .

Associazioni n-arie:

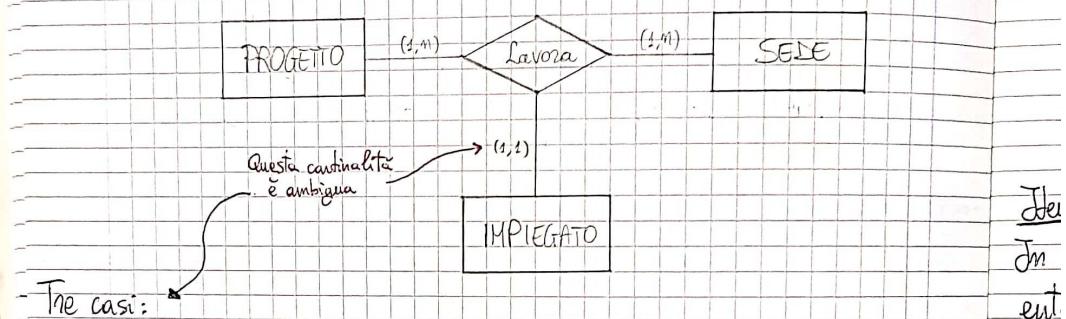


In un secondo momento
è anche possibile ridurla
in un'associazione binaria

07/10/2020 DE NITTO

CASE

Vediamo un altro esempio di associazione ternaria:



- Tre casi:

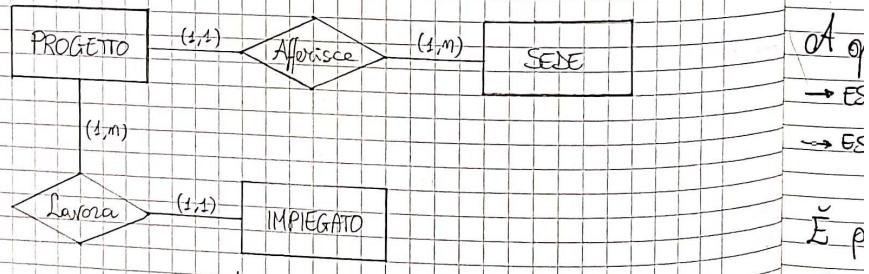
CASO 1: Il progetto ha una sola sede e l'impiegato lavora su un solo progetto.

CASO 2: La sede ha un solo progetto e l'impiegato lavora su una sola sede.

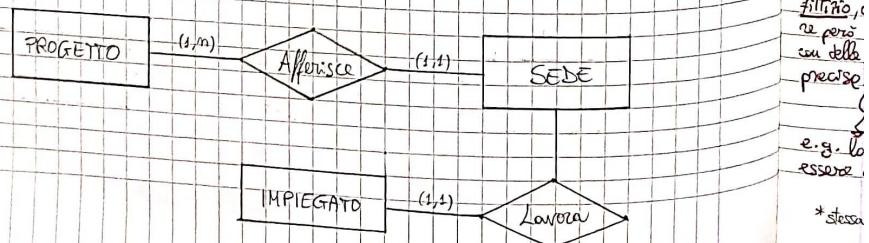
CASO 3: L'impiegato lavora in un solo progetto e in una sola sede.

Per eliminare l'ambiguità, si può convertire l'associazione ternaria in due associazioni binarie.

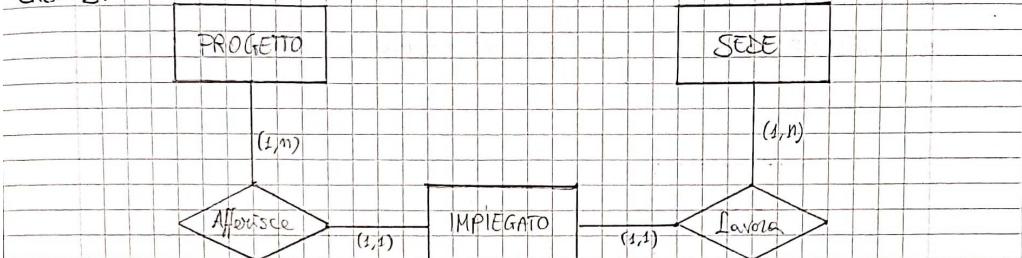
CASO 1:



CASO 2:



CASO 3:

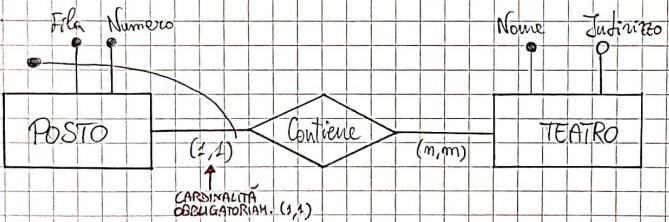


Identificatori esterni:

In alcuni casi è necessario identificare univocamente un'entità tramite un'altra entità. La prima di queste due entità viene definita DEBOLE.

Per rappresentare questo scenario si fa uso degli identificatori esterni.

ESEMPIO:

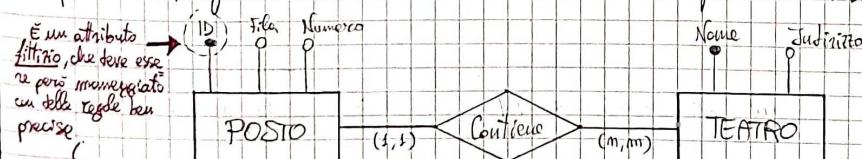


Al questo punto, l'identificatore di un'entità può:

→ ESSERE COSTITUITO DAI SEI ATTRIBUTI DELL'ENTITÀ STESSA (identificatore interno)

→ ESSERE COSTITUITO DA (ATTRIBUTI +) ENTITÀ ESTERNE ATTRAVERSO RELATIONSHIP
↳ non obbligatorio (identificatore esterno)

È possibile trasformare un'entità debole in un'entità forte. Nel nostro esempio:



e.g. lo stesso posto non deve essere associato a + ID diversi

* stessa fila, stesso numero, stesso teatro

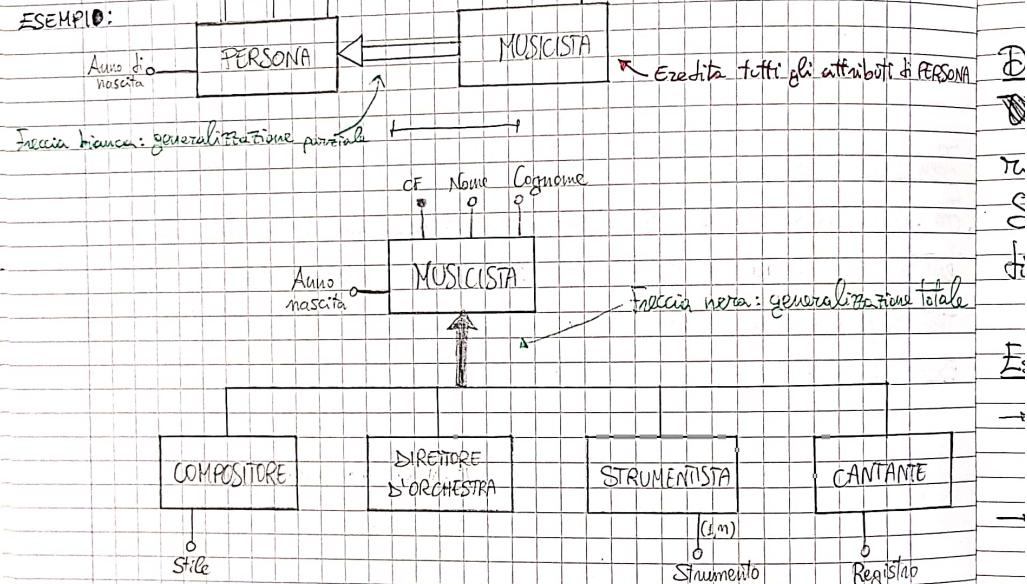
Generalizzazione:

È una caratterizzazione che si stabilisce nei modelli Entity-Relationship estesi

(non di base), cf Nome Cognome

Strumento
(1..n)

ESEMPIO:



→ GENERALIZZAZIONE ESCLUSIVA: ogni occorrenza dell'insieme (dell'entità) padre deve trovarsi al + in un unico insieme figlio.

→ GENERALIZZAZIONE SOVRAPPOSTA: un'occorrenza dell'entità padre può trovarsi anche in + entità figlie differenti.

08/10/2020 DE NUNIO

I modelli Entity-Relationship, inoltre, aggiungono la parte documentativa, che è composta dal DIZIONARIO DEI DATI (che tiene conto di entità e associazioni) e dai VINCOLI NON DIRETTAMENTE ESPRIMIBILI TRAMITE il DIAGRAMMA (che sono sottintesi in vincoli di integrità e regole di derivazione).

- I VINCOLI DI INTEGRITÀ sono asserzioni del tipo:
<concetto> deve / non deve <espressione sui concetti>
- Le REGOLE DI DERIVAZIONE sono asserzioni del tipo:
<concetto> si ottiene <operazione sui concetti>

Dizionario dei dati:

Se basato sull'^{entità} è una tabella composta da 4 colonne che indicano rispettivamente: ENTITÀ - DESCRIZIONE - ATTRIBUTI - IDENTIFICATORE.

Se basato sulle relationship, è una tabella composta da 4 colonne che indicano rispettivamente: ASSOCIAZIONE - DESCRIZIONE - COMPONENTI - ATTRIBUTI
Le entità coinvolte nell'associazione

Esempi di regole di derivazione:

- Il n° di impiegati di un dipartimento si OTTIENE contando gli impiegati che vi appartengono. → (contando il n° di coppie presenti nell'associazione "AFFERISCE").
- Il budget di un progetto si ottiene moltiplicando per 3 la somma degli stipendi degli impiegati che vi partecipano.

Progettazione concettuale:

Prende in input i REQUISITI DELLA BASE DI DATI, perché è un'attività che richiede che siano state ^{più} completate le seguenti attività:

- ACQUISIZIONE DEI REQUISITI
- ANALISI DEI REQUISITI
- COSTRUZIONE DELLO SCHEMA CONCETTUALE
- COSTRUZIONE DEL GLOSSARIO

} Sono attività soggette a feedback e possono riportare indietro con le fasi

Acquisizione dei requisiti:

È un'attività non standardizzabile; le possibili fonti dei requisiti sono:

- UTENTI / COMMITTENTI, attraverso interviste o documentazioni apposite
- DOCUMENTAZIONE GIÀ ESISTENTE, come normative, procedure aziendali
- MODULISTICA

Analisi dei Requisiti:

Regole generali per i Requisiti:

- SCEGLIERE IL LIVELLO CORRETTO DI ASTRAZIONE (→ di dettagliatezza)
- STANDARDIZZARE LA STRUTTURA DELLE FRASI
- SUDDIVIDERE LE FRASI ARTICOLATE
- SEPARARE LE FRASI ~~SUL~~ SUI DATI DA QUELLE SULLE FUNZIONI

Regole generali per organizzare termini e concetti:

- INDIVIDUARE I SINONIMI E UNIFICARE I TERMINI DI CONSEGUENZA
- RENDERE ESPlicito il riferimento FRA TERMINI
- RIORGANIZZARE LE FRASI PER CONCETTI (ovvero in gruppi omogenei)
- COSTRUIRE UN GLOSSARIO DEI TERMINI (è una tabella composta da 4 colonne che indicano rispettivamente: TERMINE - DESCRIZIONE - SINONIMI - COLLEGAMENTI)

09/10/2020 PELEGRINI

REFICAZIONE → trasformazione di una relazione in un'entità; si può assistere nel passaggio da una relazione ternaria a delle relazioni binarie.

14/10/2020 DE MATTEO

Linee guida per la progettazione concettuale:

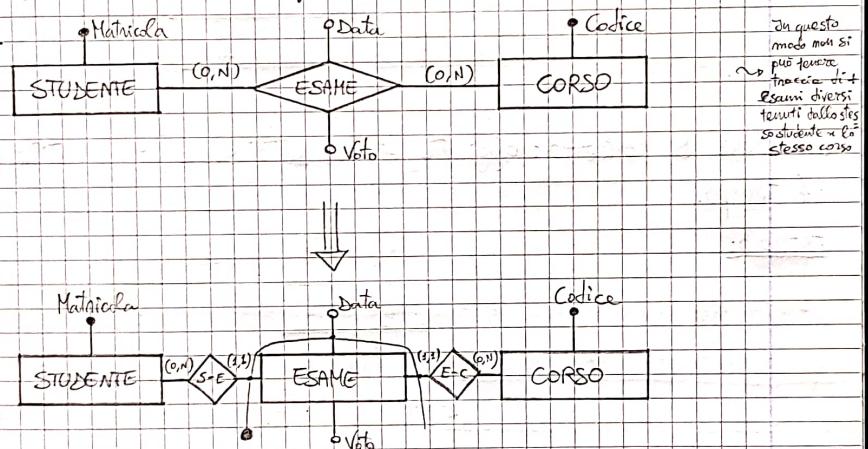
- Se un concetto ha proprietà significative e descrive oggetti con esistenza autonoma ⇒ ENTITÀ
- Se un concetto è semplice e non ha proprietà ⇒ ATTRIBUTO
- Se un concetto connette due o più concetti ⇒ RELATIONSHIP

→ Se un concetto è un caso particolare da un altro \Rightarrow GENERALIZZAZIONE

Design pattern:

- REIFICAZIONE: trasformazione di un concetto astratto in concetto concreto; consiste anche nella trasformazione di un ATTRIBUTO in ENTITÀ con esistenza autonoma, oltre che nella trasformazione di un' ASSOCIAZIONE in ENTITÀ.

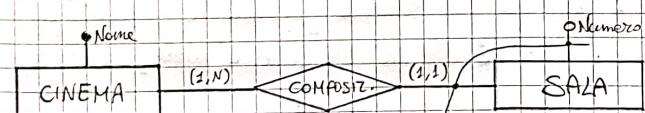
ESEMPIO:



• PART-OF

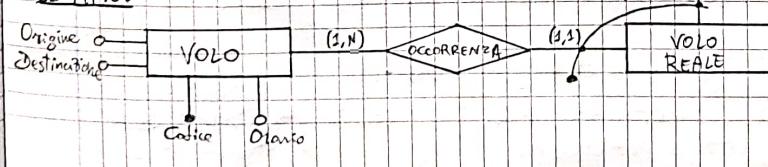
- COMPOSIZIONE: associazione tra due entità (A,B), in cui B fa parte di A.

ESEMPIO:



- INSTANCE-OF: associazione tra un'entità generale e un'istanza reale/specifica di quell'entità.

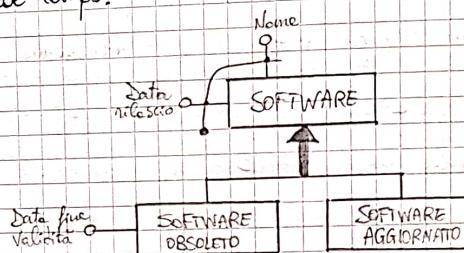
ESEMPIO:



→ TRAMITE LA GENERALIZZAZIONE

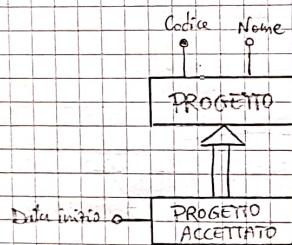
- STORICIZZAZIONE DI CONCETTO: suddivisione di un'entità in due entità specificate che evolvono col tempo.

ESEMPIO:



- EVOLUZIONE DI CONCETTO: aggiunta di un sottoinsieme di un'entità che evolve col tempo.

ESEMPIO:



Strategie di costruzione di un diagramma ER:

→ TOP-DOWN: si parte da uno schema generale e si aggiungono via via sempre più dettaglio, fino ad ottenere uno schema molto specifico e dettagliato.

→ BOTTOM-UP: si parte dalle singole specifiche, le si convertono singolarmente in schemi e si collegano poi gli schemi ottenuti in modo opportuno.

→ INSIDE-OUT: si parte da alcuni concetti importanti e si procede da questi "a macchia d'olio" aggiungendo via via altri concetti.

Nella pratica, si segue una strategia mista:

→ SI INDIVIDUANO I CONCETTI PRINCIPALI e SI REALIZZA UNO SCHEMA SKELETO.

→ SUA BASE DI QUESTO SI PUÒ DECOMPOSERE E Poi SI RAFFINA, SI ESPANDE, SI INTEGRA.

- Uno schema concettuale deve essere: CORRETTO, COMPLETO, LEGGIBILE e MINIMALE.
- La correttezza deve essere sia SINTATTICA che SEMANTICA.
e.g. evitare la generalizzazione fra associazioni.)
- ↳ rappresentare ciò che è effettivamente richiesto nelle specifiche
- La completezza consiste in una condizione in cui TUTTI i concetti sono rappresentati e TUTTE le operazioni sono state eseguite.
- La minimalità è sostituita se tutti i concetti sono ripetuti ~~ma~~ una e una sola volta.

21/10/2020

PROGETTAZIONE LOGICA

→ È una fase in cui si inizia a pensare alle questioni realizzative e di efficienza. Inoltre, bisogna raccogliere le informazioni sul carico applicativo che, insieme allo schema concettuale ER, definisce con più di un modo per la fase della progettazione logica.

La prima cosa da fare è ricostruire lo schema ER per renderlo più efficiente e portarlo allineato agli standard del modello logico (e.g. eliminando le generalizzazioni). Questa è l'unica attività della progettaz. logica che prescinde dal modello logico utilizzato (noi consideriamo il modello RELAZIONALE).

→ CARICO APPLICATIVO: descritto dalla dimensione dei dati e dalle caratteristiche delle operazioni.

↳ Viene rappresentato tramite una tabella (la TAVOLA DEI VOLUMI), che è composta da 3 colonne:

- CONCETTO
- TIPO DI CONCETTO (ENTITÀ / RELAZIONE) ≡ COSTRUITO
- VOLUME (CHE INDICA UNA QUANTITÀ)

Non conoscendo il sistema sottostante, l'efficienza / le prestazioni sono valutate in modo approssimativo. I parametri che descrivono elettano l'efficienza sono 2:

→ SPAZIO: si basa sul volume dei dati.

→ TEMPO: si basa su quante entità (e quante relazioni) devono essere "percorse" (o coinvolte) dalle operazioni più pesanti.

SOLO LE + PESANTI PERCHÉ VIGE LA REGOLA DELL'80-20:

L'80% del carico è determinato (in generale) dal 20% delle per.

Questo si descrive tramite la TAVOLA DEGLI ACCESSI (una per ogni operazione) che ha 4 colonne:

- CONCETTO
- COSTRUTTO → Entità / Relationship
- ACCESSI (num. accessi)
- TIPO → Scrittura / Scrivitura

Attività di ristrutturazione:

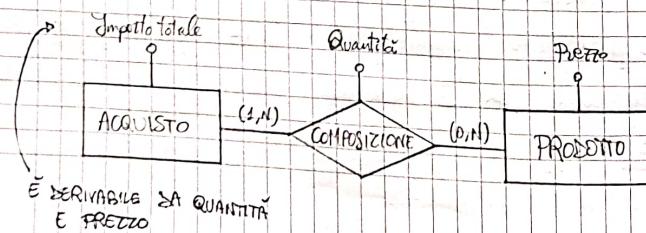
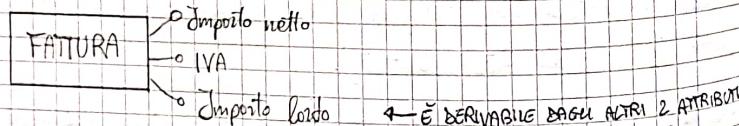
STEP 1: ANALISI DELLE RIDONDANZE

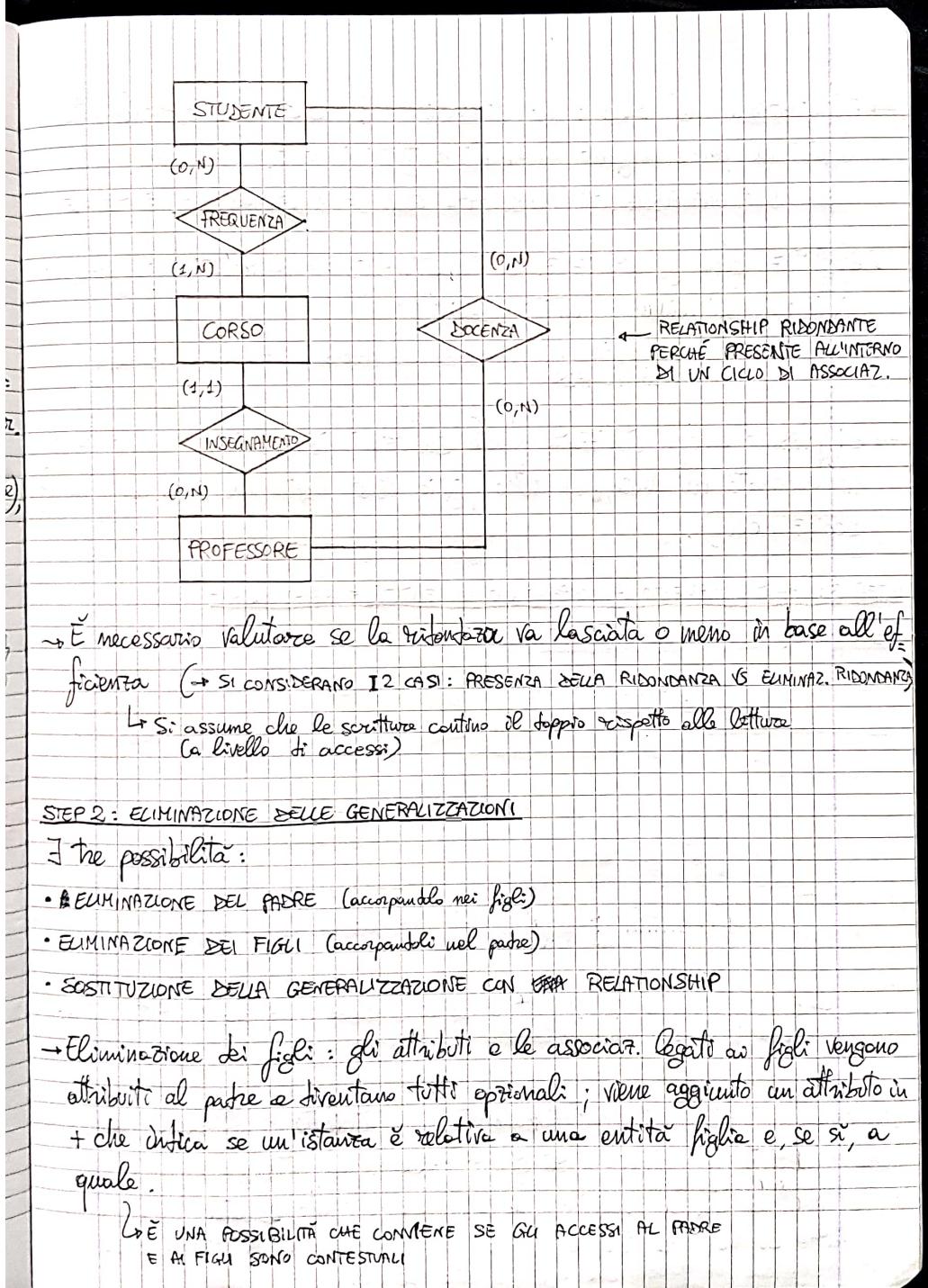
È due forme di ridondanza principali:

- ATTRIBUTI DERIVABILI da altri attributi della stessa entità (o relationship) o di altre entità (o relationship).

- RELATIONSHIP DERIVABILI DALLA COMPOSIZIONE DI ALTRE

Esempi:





→ Eliminazione del padre: gli attributi e le relationship inizialmente legati al padre vengono attribuiti a tutti i figli.

↳ È UNA POSSIBILITÀ CONSENTITA SOLO SE LA GENERALIZZAZ. È TOTALE

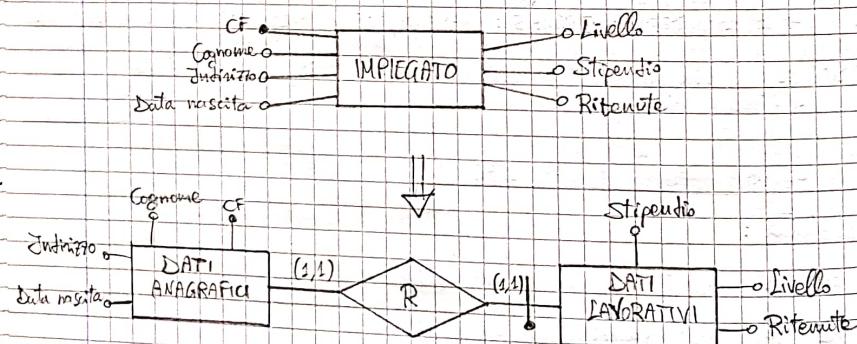
→ Sostituz. della generalizzaz. con relationship: una relationship per ogni figlio; i figli diventano entità deboli.

↳ È UNA POSSIBILITÀ CONVENIENTE SE GIA' ACCESSI AI FIGLI SONO SEMPRE DAGLI ACCESSI AL PADRE

→ Si possono anche utilizzare delle soluzioni ibride (in cui un figlio viene accapato al padre, un altro figlio viene legato al padre con una associaz., ecc.).

STEP 3: PARTIZIONAMENTO / ACCORDAMENTO DI ENTITÀ E RELATIONSHIP

Esempio di partizionamento verticale di entità:



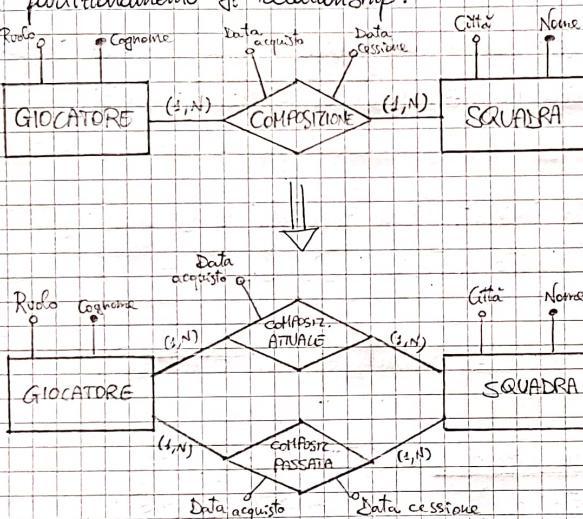
→ È un'operazione che si effettua se ci si rende conto che l'entità partizionata in questo modo aumenta l'efficienza?

Un esempio tipico di partizionamento è l'attributo multivalore (che spesso viene considerato in entità a sé stante): se rimane attributo di un'entità X e, per qualche istanza specifica, la cardinalità è molto elevata, quest'ultima si riflette su TUTTE le istanze di tale entità, portando a un'utilizzo di spazio di molto maggiore di quelli che effettivamente serve.

22/10/2020

Operazione confluente: ACCOGLIIMENTO di concetti in un'unica entità.

Esempio di partitionamento di relationship:



STEP 4: SCELTA DEGLI IDENTIFICATORI PRINCIPALI → nel caso in cui un'entità abbia + identific.

Segue 3 criteri:

- L'attributo non deve essere opzionale
- L'attributo deve essere semplice
- L'attributo deve essere usato molto frequentemente nelle operazioni

→ Se non esiste alcun attributo che rispetta questi criteri, se ne crea uno artificiale appositamente.

Modelli logici dei fatti:

Ne esistono tre tradizionali:

→ MODELLO GERARCHICO: usa una struttura ad albero (→ non può rappresentare relazioni multi-a-multi); non presenta un'indipendenza dalla struttura fisica (è un limite).

→ MODELLO RETICOLARE: usa una struttura a grafo (\rightarrow risolve il vincolo sulle relazioni molti-a-molti); tuttavia, non risolve la dipendenza sulla struttura fisica.

→ MODELLO RELAZIONALE: usa delle tabelle; è un modello indipendente dalla struttura fisica (utilizza solo dei riferimenti logici).

Definizione:

Siano D_1, D_2 due insiemi. Una relazione R su questi due insiemi è un sottoinsieme del prodotto cartesiano $D_1 \times D_2$. \rightarrow Def. generalizzabile sul numero di insiemi

Definizione:

Una TUPLA è una sequenza ordinata di valori di attributi.

Definizione:

Un ATTRIBUTO è una coppia (nome, tipo di dato).

ESEMPI: (Nome, stringa) \rightarrow (Strumento, stringa)

ESEMPIO DI TUPLA: (Alberto, Sax)

Definizione:

Uno SCHEMA DI RELAZIONE $R(A_1, A_2, \dots, A_m)$ è tale che, $\forall j \in \{1, \dots, m\}$

- A_j è il nome dell'attributo

- $\text{dom}(A_j)$ è il tipo di dato dell'attributo (\equiv l'insieme dei valori che l'attr. può assumere)

→ A_1, A_2, \dots, A_m è una lista ordinata di attributi, e può essere identificata con X ($X := A_1, A_2, \dots, A_m$)

→ m è il GRADO DELLA RELAZIONE

ESEMPIO: MUSICISTA (Nome, Strumento)

→ che poi si traducrà in numeri nella tabella

Definizione:

Un'ISTANZA DI RELAZIONE è un insieme di tuple $r = \{t_1, t_2, \dots, t_m\}$,

dove $\forall i \in \{1, \dots, m\}$ $t_i = \langle v_1, \dots, v_j, \dots, v_m \rangle$ è una lista di ~~attributi~~ valori

($\rightarrow v_1, v_2, v_3, \dots, v_m \in \text{dom}(A_j)$)

28/10/2020

Definizione:

Su uno schema $R = \{R_1(X_1), R_2(X_2), \dots, R_n(X_n)\}$, una BASE DI DATI, è un insieme di relazioni ~~sotto forma~~ $\tau = \{\tau_1, \tau_2, \dots, \tau_m\}$, con τ_i relazione su R_i .

RELAZIONE DI GRADO m

R	A_1	...	A_m
t_1	v_{11}	...	v_{1m}
...
t_n	v_{n1}	...	v_{nm}

→ Il modello relazionale ha una struttura NON POSIZIONALE: il significato delle colonne non dipende dalla loro posizione, ma anche l'ordinamento delle righe è irrilevante. Il ruolo delle varie righe e colonne dipende solo dalla loro intestazione.

N.B.: Le righe e le colonne devono essere UNICHE: non possono esistere due uguali.

→ Il modello relazionale impone ai dati una STRUTTURA RIGIDA: le informazioni sono rappresentate per mezzo di tuple, le cui componenti devono essere necessariamente di un certo tipo; può succedere però che alcuni dati che hanno luogo a un'informazione incompleta e, a tal proposito, esistono tre casi diversi:

- VALORE SCONOSCIUTO
- VALORE INESISTENTE
- VALORE SENZA INFORMAZIONE

Possibile soluzione:
ASSEGNAZIONE DEL VALORE NULLO

Definiamo $t[A] :=$ valore $\in \text{dom}(A) \cup \text{NULL}$

Il DBMS non attua questa distinzione ↴ noi si

unicue

* Il vincolo di ENNUPLA (vincolo sull' tupla), insieme a quello di dominio, costituisce la base di una tupla all'interno di un'istanza di relazione.

Carattere vincolante di una tupla

Che è un particolare tipo di vincolo di tupla

→ VALORE SCONOSCIUTO: valore che dovrebbe esistere, ma attualmente non è noto.

ESE

• (

→ VALORE SENZA INFORMAZIONE: valore di cui non si sa se esiste oppure no.

• (

→ VALORE INESISTENTE: valore che non ha senso attribuirlo in uno specifico caso, oppure di cui si sa che non esiste a priori.

Def

Un

tal

In generale, i valori NULL rendono l'intera tupla corrispondente priva di significato, per cui null devono essere troppo numerosi. A tal proposito, vengono in soccorso i VINCOLI DI INTEGRITÀ.

• VINCOLO DI INTEGRITÀ = proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione.

Defi

Una

• Un vincolo è una funzione booleana (un predicato): associa a ogni istanza il valore VERO o FALSO.

ESEM

ESEMPIO DI VINCOLO DI ENNUPLA:

come i vincoli di dominio e di sopravvivenza

→ È UN VINCICO INTRA-RELACIONALE: DÀ UNA REGOLA CHE DEVE VALERE ALL'INTERNO DI UN'UNICA RELAZIONE.

→ ANCHE I VINCICI INTER-RELACIONALI, CHE COINVOLGONO 2 o più RELAZIONI

Stipendi:

IMPIEGATO	LORDO	NETTO	TASSE
Rossi	€ 50.000	€ 30.000	€ 20.000
Bianchi	€ 40.000	€ 27.000	€ 13.000

VINCOLO DI ENNUPLA: lordo = netto + tasse

ESEMPIO DI BASE DI DATI SCORRETTA:

Esami:

STUDENTE	VOTO	LODE	CORSO
276545	32 !		01
287643	30	e late	02
739430	27	e late!	03

Studenti:

MATRICOLA	COGNOME	NOME
276545	Rossi	Mario
276545	Bianchi	Luca

Non comparendo nell'altra tabella: si risolve con un vincolo inter-relat.

IN PARTICOLARE, CON UN VINCICO DI INTEGRITÀ REFERENZIALE (Lo vedremo tra poco)

ESEMPI DI VINCOLI ATTI A CORREGGERE LA BASE DI DATI:

- $(voto \geq 18) \text{ AND } (voto \leq 30)$
- $(voto = 30) \text{ OR NOT } (lode = \text{"e lode"})$

Definizione:

Una SUPERCHIAVE SK di uno schema $R(X)$ è un sottoinsieme $SK \subseteq X$ tale che \forall coppia $t_i, t_e, i \neq e, i, e \in [1, \dots, n]$
 $t_i(SK) \neq t_e(SK)$

Definizione:

Una CHIAVE è una superchiave "minimale".

↳ DA CUI, ELIMINANDO UN QUALUNQUE ATTRIBUTO, NON SI OTTENERE PIÙ UNA SUPERCHIAVE

ESEMPIO:

MUSICISTA	Nome	Cognome	Strumento
t_1	Mario	Rossi	Piano
t_2	Alberto	Bianchi	Sax
t_3	Alessandro	Rossi	Piano

→ In questo caso, COGNOME, STRUMENTO e la coppia COGNOME+STRUMENTO non possono essere delle superchiavi.

↳ Comunque sia, la definizione di chiave e superchiavi non viene definita in base all'istanza (e quindi ai valori dei dati), bensì ^{in base} allo schema e al significato delle varie intestazioni delle colonne.

L'importante della chiavi sta nella possibilità di accedere ai dati in modo univoco. Tuttavia, il valore NULL a un elemento di una tupla che è associato a una chiave, potrebbe essere un problema.

↳ INTRODUZIONE DELLA CHIAVE PRIMARIA, che è una chiave su cui non sono ammessi valori nulli.

→ Fai riferimento al precedente esempio sulla "base di fatti scorretta"

Definizione:

Una CHIAVE ESTERNA FK di $R_1(X_1)$ che fa riferimento a $R_2(X_2)$ è:
1
un insieme di attributi FK $\subset X_1$ tale che $|FK| = |K|$, dove K è la chiave
2
primaria di $R_2(X_2)$.
3

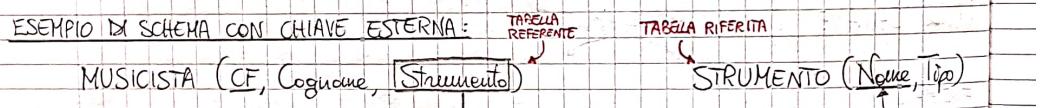
Il valore $t_1^{[i]} \in t_{1[i]}(R_1)$ deve essere presente in una tupla $t_2 \in t_{2[i]}(R_2)$,
1
 t_2 è una tupla dello schema R_2
2

e cioè: $t_1^{[i]}[FK] = t_2^{[i]}[K]$ oppure $t_1^{[i]}[FK] = \text{NULL}$

R_1 VIENE DETTA TABELLA REFERENTE e R_2 VIENE DETTA TABELLA RIFERITA.

29/10/2020

ESEMPIO DI SCHEMA CON CHIAVE ESTERNA:



→ Strumento è FK in MUSICISTA: non possono esserci musicisti che suonano strumenti non presenti all'interno della relazione STRUMENTO.

→ Se viene eliminata una tupla all'interno dello schema riferito, potrebbe accadere a picco una chiave esterna, per cui si potrebbe avere una violazione.

• COMPORTAMENTO STANDARD: Rifiuto dell'operazione.

• AZIONI COMPENSATIVE → Eliminazione in cascata all'interno dello schema referente.

→ Introduzione dei valori nulli all'interno dello schema referente.

Progettazione logica:

Ha come obiettivo quello di ottimizzare la successiva realizzazione fisica (tieni conto dell'efficienza).

IDEA DI BASE: • Le entità diventano relazioni sugli stessi attributi.

→ Gli attributi delle entità diverranno gli ATTR. X DELLA RELAZ. R

• Le relationship diventano relazioni sugli identificatori delle entità coinvolte (più gli attributi propri).

Traduzione di un'entità forte:

1. Creare uno schema relazionale il cui nome è il nome dell'entità.
2. Tutti gli attributi dell'entità sono gli attributi dello schema relazionale.
3. L'attributo che identifica l'entità è la chiave primaria dello schema relazionale.
→ ALL'INTERNO DELLO SCHEMA VIENE SOTTOLINEATA

→ Gli attributi composti vengono convertiti in tanti attributi semplici all'interno dello schema relazionale.

Traduzione di attributi multivalue:

1. Si crea uno schema relazionale che ha come nome l'attributo multivalue.
2. Gli attributi di questo nuovo schema sono:
 - Il nome dell'attributo (o associazione).
 - La chiave primaria dello schema relazionale relativo all'entità.
3. La chiave primaria di questo schema generato sarà la composizione di tutti gli attributi.
4. Si aggiunge un vincolo di integrità referenziale tra lo schema relazionale dell'attributo multivalue e lo schema relazionale principale.

Traduzione di associazioni binarie multi-a-molti:

1. Trasformare le due entità nei due schemi relazionali corrispondenti.
2. Aggiungere uno schema relazionale corrispondente all'associazione che ha come attributo:
→ TABELLA PIVOT
 - Le chiavi primarie dei due schemi relazionali.
 - Eventuali attributi dell'associazione.
3. La chiave primaria di questo nuovo schema relazionale è la composizione delle due chiavi primarie.
4. Aggiungere due vincoli di integrità referenziale tra gli attributi della chiave composta e le chiavi primarie.

$$\rightarrow x, y \in \{0,1\}$$

Traduzione di associazioni binarie $(x, 1) - (y, n)$: \rightarrow TRASFORMAZIONE NON OTTIMA

1. Trasformare le due entità nei due schemi relazionali corrispondenti.

2. Aggiungere uno schema relazionale corrispondente all'associazione che ha come attributi:

- Le chiavi primarie dei due schemi relazionali.

- Eventuali attributi dell'associazione.

3. La chiave primaria di questo nuovo schema relazionale è la chiave primaria che deriva dall'entità che partecipa con cardinalità max pari a 1.

4. Aggiungere due vincoli di integrità referenziale tra gli attributi del nuovo schema e quelli derivati dalle entità forniti.

Traduzione di associazioni binarie $(1, 1) - (y, n)$: \rightarrow Più efficiente della precedente (TRASFORMAZIONE OTTIMA)

1. Trasformare l'entità che partecipa con cardinalità max pari a n nello schema relazionale corrispondente.

2. L'entità che partecipa con cardinalità max pari a 1 viene trasformata allo stesso modo ma ad essa vanno aggiunti:

- Gli attributi della chiave primaria dell'altro schema.

- Gli eventuali attributi dell'associazione coinvolta.

3. La chiave primaria di quest'ultimo schema è la chiave primaria che deriva dall'entità che partecipa con cardinalità max pari a 1.

4. Aggiungere il vincolo di integrità referenziale sugli attributi che sono stati aggiunti all'ultimo schema relazionale.

\rightarrow Nelle associazioni binarie $(0, 1) - (y, n)$ si potrebbe ricorrere a entrambi questi ultimi meccanismi: se nell'ultimo si rischia di avere moltissimi valori null (causati dalla non-partecipazione di molte istanze all'associazione), è preferibile il penultimo, e viceversa.

\rightarrow Se ci ha un'associazione in cui entrambe le entità hanno cardinalità max pari a 1, si può decidere quale delle due entità verrà convertita in uno schema relazionale contenente la foreign key (CHIAVE ESTERNA); la scelta si basa sulle operazioni che dovranno essere eseguite sulla base di dati.

\rightarrow UNA TERZA POSSIBILITÀ È EFFETTUARE UNA TRASFORMAZIONE NON OTTIMA OBTIENENDO COSÌ 3 RELAZIONI DISTINTE (può essere tuttavia presa in considerazione se le entità partecipano con cardinalità max pari a 0).

C

I

1

2.

3.

4.

1.

2.

3.

4..

1.

2.

3.

04/11/2020

Traduzione di associazioni ricorsive:

1. Trasformare l'entità forte nello schema relazionale corrispondente.
2. Aggiungere uno schema relazionale corrispondente all'associazione che ha come attributi:
 - La chiave primaria dell'altro schema, che però deve essere ripetuta due volte (per i due diversi rami), con nomi diversi per evitare ambiguità.
 - Eventuali attributi dell'associazione.
3. La chiave primaria di questo nuovo schema di relazione è la coppia dei due attributi creati a partire dalla chiave primaria della prima relazione.
4. Aggiungere due vincoli di integrità referenziale.

Traduzione di associazioni ternarie:

1. Trasformare le tre entità nei tre schemi relazionali corrispondenti.
2. Aggiungere uno schema relazionale corrispondente all'associazione che ha come attributi:
 - Le chiavi primarie dei tre schemi relazionali ed eventuali attributi dell'associazione.
3. La chiave primaria di questo nuovo schema di relazione è la composizione delle tre chiavi primarie.
4. Aggiungere tre vincoli di integrità referenziali tra gli attributi della chiave composta e le chiavi primarie dei tre schemi corrispondenti alle entità forti.

Traduzione di associazioni ternarie con cardinalità max pari a 1:

1. Trasformare le due entità che partecipano con cardinalità max pari a 1 nei due schemi relazionali corrispondenti.
2. Trasformare l'entità che partecipa con cardinalità max pari a 1 aggiungendo alla lista di attributi:

- Le chiavi primarie degli altri due schemi.
- Gli eventuali attributi dell'associazione.
- 3. Aggiungere due vincoli di integrità referenziale tra gli attributi di quest'ultimo schema e le chiavi primarie dei due schemi corrispondenti.

Trasformazione di un'entità debole:

1. Trasformare l'entità forte nello schema relazionale corrispondente.
2. Trasformare l'entità debole aggiungendo alla lista degli attributi:
 - Gli attributi della chiave primaria dello schema relativo all'entità forte.
 - Gli eventuali attributi dell'associazione coinvolta.
3. La chiave primaria di quest'ultimo schema di relazione è data dalla composizione degli attributi che identificano l'entità debole e quelli che identificano l'entità forte.
4. Aggiungere il vincolo di integrità referenziale necessario.

Trasformazione di uno schema ER in schema relazionale:

1. Trasformare le entità forti che partecipano con cardinalità massima pari a n a tutte le associazioni a esse collegate.
2. Trasformare le entità forti che partecipano con cardinalità (1,1) ad almeno una delle associazioni a esse collegate.
3. Trasformare le entità forti che partecipano con cardinalità (0,1) ad almeno una delle associazioni a esse collegate.
4. Trasformare le entità deboli.
5. Trasformare tutte le associazioni rimaste.

11/11/2020

Linguaggi per basi di dati:

- DDL (Data Definition Language): definisce schemi logici, esterni, fisici e le autorizzazioni agli utenti.

1° OBIETTIVO DEI LINGUAGGI
PER BASI DI DATI

→ 2° OBIETTIVO DEI LINGUAGGI PER EDI DI DATI

→ DML (Data Manipulation Language): interroga e aggiorna le istanze di base di dati.

- Operazione di INTERROGAZIONE = funzione che, data un'istanza di base di dati, produce una relazione su un dato schema (non genera una nuova istanza).
- Operazione di AGGIORNAMENTO = funzione che, data un'istanza di base di dati, produce un'altra istanza sullo stesso schema.

Tipi di linguaggi di interrogazione:

→ DICHIARATIVI: specificano le proprietà del risultato.

Esempi: calcolo relazionale - QBE

→ PROCEDURALI: specificano le modalità di generazione del risultato.

Esempio: algebra relazionale

SQL (Structured Query Language) è una via di mezzo: include sia aspetti dichiarativi, sia aspetti procedurali.

Noi analizzeremo l'algebra relazionale e SQL, che hanno una potenza espressiva paragonabile.

QUERY ↗ ALGEBRA RELAZ.: "task oriented" → È + PRECISA NEL RAPPRESENTARE IL
SUL : "result oriented" → È + PRECISO NEL RAPPRESENTARE LA QUERY IN SE
E NON TANTO IL COME SI GENERA!

Algebra relazionale (AR):

Si articola in 3 gruppi di operazioni

↗ OPERAZIONI DI BASE (monotiche → hanno un solo gerante)
↘ OPERAZIONI INSIEMISTICHE
↙ OPERAZIONI DI CORRELAZ.

OPERAZIONI DI BASE:

• SELEZIONE → $\sigma_F(r) = \{t \mid t \in r, F \text{ è vera su } t\}$

formula proposizionale
(è un'affermazione)
tuple
relaz.

insieme delle tuple della relazione r che soddisfano la formula proposizionale F

- PROIEZIONE; sia una relazione $r(X)$ sugli attributi X , e sia $Y \subseteq X$

$$\hookrightarrow \Pi_Y(r) = \{t[Y] \mid t \in r\}$$

tuple proiettate solo sul
l'insieme di attributi Y

→ E' l'insieme di tutte le tuple di r che mostrano però non tutti gli attributi di r , bensì solo il sottoinsieme Y

→ Se si hanno più $t[Y]$ uguali, se ne lascia solo uno.

→ Una proiezione $\Pi_Y(r)$ contiene AL PIÙ tante esempi quanto l'operando ma può contenere di meno (per il motivo di cui sopra). $\sim |\Pi_Y(r)| \leq |r|$
Se Y è una superclasse, allora $\Pi_Y(r)$ contiene esattamente tante esempi quanto r .

- RIDENOMINAZIONE; sia una relazione $r(X)$ sugli attributi X , e sia Y un insieme di attributi tale che $|Y| = |X|$

A_1, A_2, \dots, A_n

← ORDINAMENTO PER X

B_1, B_2, \dots, B_n

← ORDINAMENTO PER Y

$\tilde{r}(Y)$

Nuovi nomi

Vecchi nomi

→ Si ridenominano gli A_1, A_2, \dots, A_n , os-
sequendo loro i nuovi nomi B_1, B_2, \dots, B_n

OPERAZIONI INSIEMISTICHE:

- UNIONE → può essere applicata solo su relazioni con gli stessi attributi; il risultato è un'unica relazione con tutte le tuple delle relazioni di partenza.
- INTERSEZIONE → può essere applicata solo su relazioni con gli stessi attributi (\equiv omogenee); il risultato è un'unica relazione con le sole tuple presenti in entrambe le relazioni di partenza.
- DIFFERENZA → può essere applicata solo su relazioni omogenee; il risultato è la differenza insiemistica tra le tuple della prima relazione operando e le tuple della seconda relazione operando.

→ In alcuni casi specifici potrebbe essere necessaria una (o più) ridenominazione preliminare

12/11/2020

OPERAZIONI DI CORRELAZIONE:

• JOIN; siano le relazioni $r_1(X_1)$, $r_2(X_2)$

↳ $r_1 \bowtie r_2$ di luogo a una nuova relazione i cui attributi sono $X_1 \cup X_2$

$$r_1 \bowtie r_2 = \{t \text{ su } X_1 X_2 \mid \exists t_1 \in r_1, t_2 \in r_2 : t[X_1] = t_1, t[X_2] = t_2\}$$

• \bowtie È una giustapposizione, in cui le colonne in comune a X_1, X_2 non vanno ripetute

~ Il join non richiede $X_1 \cap X_2 \neq \emptyset$

↳ Se $X_1 \cap X_2 = \emptyset \Rightarrow$ ogni tupla in r_1 viene combinata con tutte le tuple in r_2 (tanto luogo a un PRODOTTO CARTESIANO).

~ Se ogni ennupla sia di r_1 sia di r_2 contribuisce al risultato del join, si tratta di un JOIN COMPLETO.

~ Se NESSUNA ennupla né di r_1 né di r_2 contribuisce al risultato del join, si tratta di un JOIN VUOTO.

$$0 \leq |r_1 \bowtie r_2| \leq |r_1| \times |r_2|$$

SE IL JOIN COINVOLGE UNA CHIAVE DI $r_2 \Rightarrow 0 \leq |r_1 \bowtie r_2| \leq |r_1|$

SE IL JOIN COINVOLGE LA CHIAVE DI r_2 CON VINCOLO DI INTEGRITÀ REFERENZ. DA r_1 A $r_2 \Rightarrow |r_1 \bowtie r_2| = |r_1|$

Fino adesso abbiamo parlato di JOIN INTERNO, in cui, se una ennupla non contribuisce al risultato del join, viene tagliata fuori.

Intrafficiamo il JOIN ESTERNO, che estende, con valori nulli, le ennuple che vorrebbero tagliate fuori da un join interno.

JOIN ESTERNO $\xrightarrow{\text{LEFT JOIN}} \text{tutte le tuple di } r_1 \text{ vengono inserite in } r_1 \bowtie r_2$

$\xrightarrow{\text{RIGHT JOIN}} \text{tutte le tuple di } r_2 \text{ vengono inserite in } r_1 \bowtie r_2$

$\xrightarrow{\text{FULL JOIN}} \text{tutte le tuple vengono inserite in } r_1 \bowtie r_2$

JOIN E PROIEZIONI:

Siano $\tau_1(X_1)$, $\tau_2(X_2)$ due relazioni.

$$\pi_{X_1}(\tau_1 \bowtie \tau_2) \subseteq \tau_1$$

Sia $\tau(X)$ una relazione in cui X è un insieme di attributi partizionabile in modo che $X = X_1 \cup X_2$.

$$\pi_{X_1}(\tau) \bowtie \pi_{X_2}(\tau) \supseteq \tau$$



• PRODOTTO CARTESIANO → è un join naturale su relazioni senza attributi in comune; contiene sempre un numero di esemplificazioni pari al prodotto delle cardinalità degli operandi (le esemplificazioni sono tutte combinabili).

• θ-JOIN; siano $\tau_1(X_1)$, $\tau_2(X_2)$ due relazioni e sia F una formula proposizionale.

$$\hookrightarrow \tau_1 \bowtie_F \tau_2 = O_F(\tau_1 \bowtie \tau_2)$$

18/11/2020

Equivalenza di espressioni:

Due espressioni sono EQUIVALENTE se producono lo stesso risultato qualunque sia l'istante attuale della base di dati.

EQUIVALENZA ASSOLUTA se $E_1 \equiv E_2 \quad \forall R$ (per qualsiasi schema)

DIPENDENTE DALLO SCHEMA se $E_1 \equiv_R E_2 \quad \forall r$ di R (dove R è uno schema specifico)

ESEMPIO DI EQUIVALENZA ASSOLUTA:

$$\pi_{A,B}(O_{A \bowtie B}(R)) \equiv O_{A \bowtie B}(\pi_{A,B}(R))$$

ESEMPIO DI EQUIVALENZA RELATIVA DIPENDENTE DALLO SCHEMA:

$$\pi_{A,B}(R_1) \bowtie \pi_{A,C}(R_2) \equiv_R \pi_{A,B,C}(R_1 \bowtie R_2)$$

↳ Le due espressioni sono equivalenti \Leftrightarrow hanno A come unico attributo in comune ($X_1 \cap X_2 = A$)

Cous
Se

Select
Suppo
conten
Se m
dore
ESEM:
O
O_{eta}
O_{eta}
O_{eta}

Viste
→ RE
→ RE

Cat

Le vi;
1) vis
Vai
Sva

2) REZ

Consideriamo la seguente espressione: $\sigma_F(E_1 \bowtie E_2)$

Se F coinvolge solo attributi di $E_2 \Rightarrow \sigma_F(E_1 \bowtie E_2) \equiv E_1 \bowtie \sigma_F(E_2)$

PUSH SELECTIONS DOWN: è buona norma effettuare la selezione σ_F quanto prima possibile.

Selezione con valori nulli:

Supponiamo di avere un'operazione di selezione $\sigma_{A>0}$ da applicare su uno schema contenente A , in cui alcune tuple hanno un valore NULL in corrispondenza di A .

Se non viene specificato niente, il comportamento di default è scartare o preservare le tuple con l'attributo A con valore NULL.

ESEMPIO:

$$\sigma_{\text{Eta} > 30}(\text{Persone}) \cup \sigma_{\text{Eta} \leq 30}(\text{Persone}) \neq \text{Persone}$$

$$\sigma_{\text{Eta} > 30 \vee \text{Eta} \leq 30}(\text{Persone}) \neq \text{Persone}$$

$$\begin{aligned} \sigma_{\text{Eta} > 30}(\text{Persone}) \cup \sigma_{\text{Eta} \leq 30}(\text{Persone}) \cup \sigma_{\text{Eta IS NULL}}(\text{Persone}) &= \\ = \sigma_{\text{Eta} > 30 \vee \text{Eta} \leq 30 \vee \text{Eta IS NULL}}(\text{Persone}) &= \text{Persone} \end{aligned} \quad \left. \begin{array}{l} \text{Esempio di utilizzo} \\ \text{di IS NULL / IS NOT NULL} \end{array} \right.$$

Viste (= relazioni derivate):

→ RELAZIONI DI BASE: definiscono un contenuto autonomo.

→ RELAZIONI DERIVATE (VISTE): sono relazioni il cui contenuto è funzione del contenuto di altre relazioni (definito per mezzo di interrogazioni).

Le viste si classificano in:

1) VISTE MATERIALIZZATE = relazioni derivate effettivamente presenti nella base di dati

Vantaggi: disponibilità immediata per le interrogazioni

Svantaggi: ridondanza; appesantimento degli aggiornamenti; raramente supportate dai DBMS

2) RELAZIONI VIRTUALI → sono quelle che effettivamente utilizziamo → ci riferiamo a loro quando discutiamo del termine "viste"

Le viste sono otte per catturare dei meccanismi di privatità e delle autorizzazioni, oltre alla possibilità di restringere l'attenzione su una porzione di uno schema (fatto che all'utente non interessa necessariamente tutte le informazioni). Inoltre, le viste possono semplificare la scrittura delle espressioni.

19/11/2020

NB: Non è possibile aggiornare le viste se non nel caso in cui l'aggiornamento sia univoco nelle relazioni di partenza \Rightarrow si ammettono pochissimi aggiornamenti sulle viste.

Normalizzazione:

È quella procedura che porta uno schema relazionale in FORMA NORMALE. FORMA NORMALE = proprietà di uno schema di dati relazionale che ne garantisce la "qualità", cioè l'assenza di difetti o anomalie.

Una relazione è NON normalizzata se presenta ridondanze o si presta a comportamenti poco desiderabili durante gli aggiornamenti.

La normalizzazione va utilizzata come tecnica di verifica dei risultati della progettazione di una base di dati.

NB: Non costituisce una metodologia di progettazione.

Una ridondanza può causare: ANOMALIE DI AGGIORNAMENTO, ANOMALIE DI CANCELLAZIONE, ANOMALIE DI INSERIMENTO.

Ciò è fatto per lo + del raggruppamento di + progetti concorrenti diversi nello stesso schema relazionale.

→ E.g. DIPENDENTE-STIPENDIO-PROGETTO-BUSCO SE PROGETTI
in cui lo stesso dipendente può partecipare a + progetti e a uno stesso progetto partecipano + dipendenti

Definizione:

Sia r un'istanza di relazione su $R(X)$ e siano $Y \subset X$, $Z \subset X$.

\exists una FUNCTIONAL DEPENDENCY FD: $Y \rightarrow\!\!\! \rightarrow Z$ $\forall (t_1, t_2)$ di r se $t_1[Y] = t_2[Y] \Rightarrow t_1[Z] = t_2[Z]$

"TERMINA"

Se $Z \subseteq Y$ è una dipendenza funzionale BANALE

$Y \rightarrow A$ è una FD non banale se $A \notin Y$.

$Y \rightarrow Z$ è una FD non banale se $Y \cap Z = \emptyset$.

→ In generale, le FD causano anomalie, tranne nel caso in cui Y è una superchiave.

Forma normale di Boyce e Codd (BCNF):

Una relazione R è in FORMA NORMALE DI BOYCE E CODD SE, per ogni dipendenza funzionale (non banale) $X \rightarrow Y$ definita su di essa, X contiene una chiave K di R .

PROCEDURA INTUITIVA DI NORMALIZZAZIONE:

• NON VALIDA IN GENERALE, MA SOLO NEI "CASO SEMPLICI":

→ Per ogni dipendenza $X \rightarrow Y$ che viola la BCNF, definire una relazione su XY ed eliminare Y dalla relazione originaria.

a) Decomposizione senza perdita:

Una relazione R si DECOMPONE SENZA PERDITA SU X_1, X_2 SE IL JOIN delle proiezioni di R su X_1 e X_2 è uguale a R stessa (cioè non contiene le cosiddette ENNUPLE SPURIE).

→ La decomposizione senza perdita è garantita se gli attributi comuni contengono una chiave per almeno una delle relazioni decomposte.

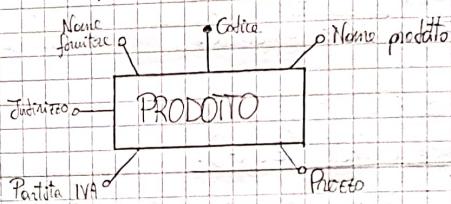
b) Conservazione delle dipendenze:

Una decomposizione CONSERVA LE DIPENDENZE SE Ogni dipendenza funzionale dello schema originario coinvolge attributi che comparevano tutti insieme in uno degli schemi decomposti.

Una decomposizione deve sempre soddisfare sia l'assenza di perdita, sia la conservazione delle dipendenze.

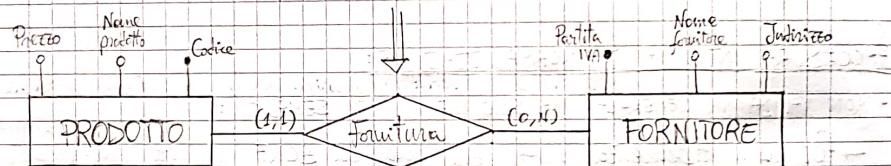
La normalizzazione può essere applicata anche in fase di progettazione concretuale per verificare la qualità del diagramma E-R.

ESEMPIO:



Partita_IVA → Nome_fornitore, Codice

È UNA DIPENDENZA FUNZIONALE CHE VIOLA LA BCNF



SQL:

SQL (originariamente "Structured Query Language") è un linguaggio che contiene sia il DDL, sia il DML.

SQL COME DDL:

→ Istruzione CREATE TABLE: definisce uno schema di relazione (e ne crea un'istanza virtuale); specifica attributi, domini e vincoli.

Per ogni attributo è necessario specificare: NOME - DOMINIO - EVENTUALE

INSIEME DI VINCOLI (e.g. PRIMARY KEY, NOT NULL, ecc.).

• DOMINIO ↗ ELEMENTARE
↘ DEFINITO dall'UTENTE

↗ In entrambi i casi può essere specificata la dimensione del dominio (e.g. CHAR(6) indica una stringa di 6 caratteri.)

• VINCOLO = proprietà che devono essere soddisfatte da ogni istanza

• VINCOLO ↗ INTRARELAZIONALE: PRIMARY KEY - NOT NULL - UNIQUE ↗ indica una superclasse
INTERRELAZIONALE: FOREIGN KEY (Attributi dello schema corrente)
REFERENCES SchemaRif (Attributi di SchemaRif)

"FOREIGN KEY" si può creare solo su un attributo chiaviello della relazione corrente e unico (0..1 solo).

25/11/2020

Azioni compensative:

ESEMPIO: Viene eliminata una tupla causando una violazione.

COMPORTAMENTO "STANDARD": Rifiuto dell'operazione (NO ACTION)

AZIONI COMPENSATIVE: → Eliminazione in cascata (CASCADE) ← APPLICABILE ANCHE IN CASO DI AGGIORNAMENTO
→ Inizializzazione di valori nulli (SET NULL)

Definizione degli indici:

Avviene a livello fisico (non logico) ed è relativa dal punto di vista delle prestazioni.

→ Comando CREATE INDEX

→ ESEMPI: create index Att1_idx on Tabella(Att1);
create unique index Att2_idx on Tabella(Att2);
se specifichiamo UNICO ci assicuriamo che ogni record (tuple) sia associato a un indice diverso

In molti sistemi si utilizzano anche linguaggi diversi da SQL per l'implementazione della base di dati.

Operazioni sui dati: ← SQL come DML

• Interrogazione: SELECT

• Modifica: INSERT - DELETE - UPDATE

ISTRUZIONE SELECT:

SELECT ListaAttributi
FROM Listatabelle

→ La clausola SELECT indica anche TARGET LIST

[WHERE Condizione(J)]

• LE PARENTESI QUADRE INDICANO UN TERMINE OZIONALE (QUI WHERE PUÒ ESSERE O NON ESSERE); SI POSSONO AVERE ANCHE DELLE PARENTESI GRAPPE CHE INDICANO LA POSSIBILITÀ DI RIPETIZIONE DI QUEL TERMINE; INFINE, IL SIMBOLO "(" (CENTRO <...>) INDICA UN'ALTERNATIVA (UN "OR")

Semantica della query in SQL:

1. Prodotto cartesiano delle tabelle in Listatabelle.
2. Selezione delle tuple che soddisfano la Condizione.
3. Proiezione degli attributi in ListaAttributi.

È possibile utilizzare delle abbreviazioni.

ESEMPIO:

```
SELECT Nome, Prezzo  
FROM Persone  
WHERE Età < 30
```

|||

```
SELECT p.Nome AS nome, p.Prezzo AS prezzo  
FROM Persone AS p  
WHERE p.Età < 30
```

QUESTO AS È OBBLIGATORIO

QUESTO AS SI PUÒ OMettere

→

ALTRÒ ESEMPIO: Sia R(A,B) una relazione sugli attributi A,B.

```
SELECT *  
FROM R  
WHERE true
```

```
SELECT X.A AS A, X.B AS B  
FROM R X  
(WHERE true)
```

È anche possibile ricorrere alla condizione "LIKE".

ESEMPIO:

Trovare le persone che hanno un nome che inizia per 'A' e ha una 'J' come terza lettera.

```
SELECT *  
FROM Persone  
WHERE Nome like 'A_J%
```

QUALUNQUE CARATTERE
QUALUNQUE INSIEME DI CARATTERI (ANCHE VUOTO)

Nelle proiezioni, SQL, a differenza dell'AR, non elimina le eventuali ripetizioni di tuple. Per ovviare a questo caratteristico, si usa la parola chiave "DISTINCT" subito dopo SELECT.

→ Una lista di tabella nella clausola FROM porta sempre all'esecuzione di un PRODOTTO CARTESIANO. Se si vuole effettuare un JOIN naturale come in AR, si può ricorrere alla seguente indicazione:

```
SELECT Maternità.Figlio, Matre, Padre  
FROM Maternità JOIN Paternità ON Maternità.Figlio = Paternità.Figlio
```

→ Esempio di scenario in cui si vuole trovare il padre e la madre di ogni persona

→ left join → non viene scartata alcuna tupla della relazione di sinistra

→ Il termine "join" da solo, indica un join interno. Al suo posto, si può utilizzare un "left join" (\equiv "left outer join"), un "right join" (\equiv "right outer join") o un "full join" (\equiv "full outer join") per ottenere un join esterno.

È possibile effettuare un ordinamento del risultato (che può essere ascendente oppure discendente).

ESEMPIO:

Trovare nome e reddito delle persone con meno di trenta anni in ordine alfabetico.

```
SELECT Nome, Reddito  
FROM Persone  
WHERE Età < 30  
ORDER BY Nome
```

OOPERATORE UNION:

```
SELECT A, B  
FROM R  
UNION  
SELECT A, B  
FROM S  
+  
Elimina i duplicati
```

SELECT (A, B)

FROM R

UNION ALL

SELECT (A, B)

FROM S

+
Non elimina i duplicati

Maneggiare l'ordine
è fondamentale

26/11/2020

OOPERATORE DIFFERENZA:

```
SELECT Nome  
FROM Dipiegato  
EXCEPT  
SELECT Cognome AS Nome  
FROM Dipiegato
```

OPERATORE INTERSEZIONE

```
SELECT Nome  
FROM Impiegato  
intersect  
SELECT Cognome as Nome  
FROM Impiegato
```

≡

```
SELECT J.Nome  
FROM Impiegato I, Impiegato J  
WHERE J.Nome = J.Cognome
```

Se,

di

→ R

►

►

→ J

si

f

• E

• N

27/1

TRANS

to l

Select

avvi

portò

recati

Trans

INPUT

La forma ridotta è "meno dichiarativa" ma talvolta + leggibile (richiede meno variabili) rispetto a quella piena.

La forma piena e quella ridotta possono essere ~~ridotte~~ combinate.

NB: le sottointerrogazioni non possono contenere operatori insiemistici ("l'uno ne si fa solo al livello esterno").

ESEMPIO:

Trovare nome e reddito dei padri delle persone che guadagnano > di 20.

```
SELECT distinct P.Nome, P.Reddito  
FROM Persone P, Famiglia F, Persone F  
WHERE P.Nome = Padre and Figlio=F.Nome  
and F.Reddito > 20
```

```
SELECT Nome, Reddito  
FROM Persone  
WHERE Nome in (SELECT Padre  
FROM Famiglia  
WHERE Figlio = any (SELECT Nome  
FROM Persone  
WHERE Reddito > 20))
```

Forma combinata:

```
SELECT Nome, Reddito  
FROM Persone  
WHERE Nome in (SELECT Padre  
FROM Famiglia, Persone  
WHERE Figlio=Nome and Reddito > 20)
```

Se, oltre a nome e reddito dei padri, avessimo dovuto indicare anche il reddito dei figli, come avremmo potuto fare nella forma normalizzata?

Lo nella prima select non possiamo ripetere l'attributo Reddito due volte!

→ Regole di visibilità:

- NON è possibile fare riferimenti a variabili definite in blocchi più interni.
- Se un nome di variabile è ammesso, si assume riferimento alla variabile "più vicina".

→ In un blocco si può fare riferimento a variabili definite in blocchi più esterni; la semantica di base (prodotto cartesiano, selezioni, proiezioni) non funziona più.

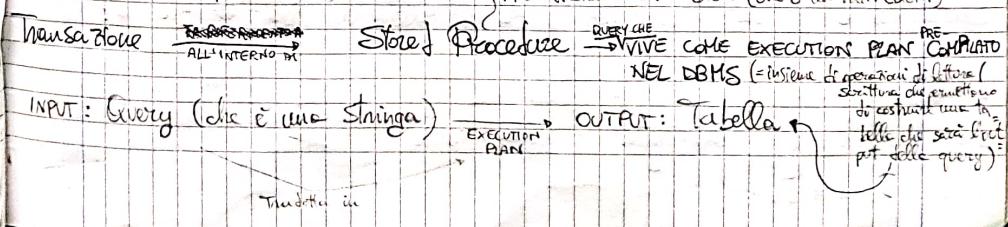
QUANTIFICAZIONE ESISTENZIALE:

- EXISTS (Sottoespressione) \equiv (true if Sottoespressione \neq empty).
- NOT EXISTS (Sottoespressione) \equiv (true if Sottoespressione = empty)

→ Si può utilizzare con una doppia negazione per indicare una quantificazione universale

27/11/2020

PELLEGRINI
insieme di primitive di lettura e scrittura
TRANSAZIONE → operazione da eseguire come atomiche; è nel DBMS non è codice
→ l'utilizzo di spinlock / semafori per la gestione della concorrenza;
bisogna specificare la porzione di codice interessata e il tipo di problema che si può specificare. In caso di errore dovuto alla concorrenza all'interno di quella porzione di codice, tutte le istruzioni vengono semplicemente abortite ed si ricomincia da capo.



02/12/2020

Operatori aggregati:

Sono: CONTEGGIO, CALCOLO DEL MINIMO, CALCOLO DEL MASSIMO,
CALCOLO DELLA MEDIA, CALCOLO DEL TOTALE.

Vengono espressi nel seguente modo:

funzione ([DISTINCT] *)

OPPURE

funzione ([DISTINCT] attributo)

CONTEGGIO (COUNT):

Supponiamo di voler contare il numero di figli di Franco:

```
select count(*) as NumFigliDiFranco  
from Paternita  
where Padre='Franco'
```

→ COUNT RESTITUISCE IL NUMERO DI TUPLE. In particolare:

- select count(*) from R → restituisce la cardinalità di R
- select count(A) from R → restituisce il numero di tuple di R che hanno * un valore nell'attributo A diverso da NULL.
- select count(DISTINCT A) from R → restituisce il numero di valori diversi tra loro e diversi da NULL assunti dall'attributo A nelle tuple di R.

MEDIA (AVG):

Supponiamo di voler trovare la media dei redditi dei figli di Franco:

```
select avg(razziffo)  
from persone join paternita on nome=figlio  
where Padre='Franco'
```

MASSIMO (MAX):

Supponiamo di voler trovare la/le persone/i reftito massimo:

```
select *  
from persone  
where reddito = (select max(reddito)  
                  from persone )
```

—————

Le funzioni possono anche essere applicate a sottosinsiemi di tuple (partizioni di relazioni). A tal proposito si ricorre alla clausola GROUP BY.

Supponiamo per esempio di voler contare il numero di figli di ciascun pa=che (non di uno in particolare):

```
select Padre, count(*) as NumFigli  
from paternita  
group by Padre
```

PASSI DA SEGUIRE PER UTILIZZARE OPERATORI AGGREGATI E RAGGRUPPAMENTI:

1. L'interrogazione senza \rightarrow group by \rightarrow su tutti gli attributi in *.
operatore aggregato)

2. Si raggruppa e si applica l'operatore aggregato a ciascun gruppo.

Condizioni sui gruppi:

Utilizzano l'operatore HAVING al posto della clausola WHERE, che invece è re=lativa alle condizioni sulle single tuple.

03/12/2020

Operazioni di aggiornamento:

INSERIMENTO:

```
INSERT INTO Tabella [(Attributi)]  
VALUES (Valori)
```

\rightarrow Se ne aggiunge
dati nuovi

oppose

```
INSERT INTO Tabella [(Attributi)]
SELECT ...
```

→ Serie x aggiungere dati a una relazione che sono già presenti in un'altra relazione.

→ Se l'inserimento porta alla violazione di uno o più vincoli, viene rifiutato dal sistema.

ELIMINAZIONE:

```
DELETE FROM Tabella
[WHERE Condizione]
```

→ L'eliminazione può causare (se i vincoli di integrità referenziale sono definiti con politiche di reazione CASCADE) eliminazioni da altre relazioni.

MODIFICA DI ENTRATE:

```
UPDATE NomeTabella
SET Attributo =
<Espressione | SELECT... | NULL | DEFAULT>
[WHERE Condizione]
```

Caratteristiche evolute di SQL:

CHECK:

È una Keyword che consente di esprimere dei vincoli (e.g. di entrata).

SINTASSI: check (Condizione)

ESEMPI:

- CHECK (sesso in ('M', 'F'))
- CHECK (Netto = Stipendio - Ritenute)

La Keyword check è utile anche per definire dei meccanismi di protezione per la definizione di viste:

```
create view NomeVista [(ListaAttributi)] as
Select SQL [with [local
cascaded] check option]
```

es. costrutto SELECT
FROM WHERE

→ Si tratta di un comando a metà tra il Data Definition Language e il Data Manipulation Language.

→ Ci ricordiamo che gli aggiornamenti sulle viste non corrispondono sempre ad aggiornamenti univoci sulle relazioni di base.

Perciò, di solito, gli aggiornamenti sono ammessi solo su viste definite su un'unica relazione. Inoltre, alcune verifiche aggiuntive possono essere impostate tramite la Keyword ^{WITH} CHECK OPTION.

WITH CHECK OPTION → WITH LOCAL CHECK OPTION: verranno effettuati controlli solo locali (solo nell'ambito dell'ultima vista definita) → IN CASO DI MANCATA DI VALORI NELLO CHECK
[default] → WITH ~~CASCADE~~ CHECK OPTION: verranno effettuati controlli a cascata sulle viste finché non si risalga alla relaz. di base
= WITH LOCAL OPTION

Controllo dell'accesso:

In SQL è possibile specificare chi (utente) e come (lettura, scrittura, ...) può utilizzare la base di dati (o parte di essa).

→ L'amministratore della base di dati (l'utente - system) ha tutti i privilegi.

→ Il creatore di una risorsa ha tutti i privilegi su di essa.

TIPI DI PRIVILEGI OFFERTI DA SQL:

- insert → permette di inserire nuovi oggetti (esempio)
- update → permette di modificare il contenuto
- delete → permette di eliminare oggetti
- select → permette di leggere la risorsa
- references → permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa)
- usage → permette l'utilizzo in una definizione

CONCESSIONE DI PRIVILEGI (GRANT):

grant <Privileges | all privileges> on Resource To Users [with grant option]

REVOCA DI PRIVILEGI (REVOKE):

revoke Privileges on Resource from Users [restrict | cascade]

→ Per autorizzare un utente a vedere solo alcune entità di una relazione, si utilizza una vista.

TRANSAZIONE:

Sappiamo essere un insieme di operazioni da considerare invisibile.

Deve soddisfare le cosiddette PROPRIETÀ ACIDE:

→ ATOMICITÀ → le transazioni devono essere eseguite o per intero o per niente

→ CONSISTENZA → i risultati delle transazioni devono soddisfare i vincoli di integrità

→ ISOLAMENTO → le transazioni concorrenti devono avere tutte effetto come se avvenissero in modo isolato

→ DURABILITÀ (PERSISTENZA) → i risultati delle transazioni devono persistere a questi e sim.

• In alcuni sistemi le transazioni iniziano col comando START TRANSACTION.

• Le transazioni ~~si~~ si concludono con:

→ COMMIT [WORK] (transazione eseguita)

→ ROLLBACK [WORK] (transazione abortita)

SIGNIFICATO DI
AUTOCOMMIT:
1 operazione = 1 transaz.

ESEMPIO:

start transaction (opzionale)

update Conto Corrente

set Saldo = Saldo - 10

where NumeroConto = 12345;

update Conto Corrente

set Saldo = Saldo + 10

where NumeroConto = 55555;

commit work;

08/12/2020 PELLEGRINI

Creatazione di una Stored Procedure:

CREATE

[OR REPLACE] ← comando che permette di aggiornare una stored procedure se già esistente

[DEFINER = {user | CURRENT_USER | role | CURRENT_ROLE}]
← definisce il creatore della procedura
← come si fissa un certo utente "sp"

PROCEDURE sp-name ([proc-parameter [, ...]])

[characteristic ...] routine-body
↓ formatt. applicativa & tira al cors

↳ una stored procedure può avere 0..N
valori di ritorno, che vengono specificati
qui, tra i parametri

proc-parameter:

[IN | OUT | INOUT] param-name type

I PARAMETRI OUT / INOUT VERRANNO INCARICATI IN UN'APPENA APERTURA TABELLA CONFERMATA DA UNA SINGOLA RIGA

type:

Any valid MariaDB data type

characteristic:

LANGUAGE SQL

[NOT] [DETERMINISTIC] → Output deterministico: se bisogna chiamare + volte la stessa stored procedure con gli stessi parametri, la procedura deve ricevere ogni volta, se non si usa sempre lo stesso output preceduto

{CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}

| SQL SECURITY {DEFINER | INVOKER}

↳ Le stored procedure modifica dei dati

| COMMENT 'string'

↳ DETAIL:

La stored procedure non è deterministica,
contiene statement SQL e modifica dei dati

routine-body:

Valid SQL procedure statement

Esempio: VARIABILE DI SESSIONE → All'interno delle transazioni viene settata a 0 automaticamente ES

set autocommit = 0;

è un nome, non una variabile (per i nomi si usa mettere gli apici all'interno)

create procedure `assign` (in var_mathicda Varchar(45), out var_token Varchar(128))

→ alcune tipi di test di dimensione variabile (e.g. test, blob) → da utilizzare se nessun

begin

declare exit handler for sqlexception

begin

rollback; -- rollback any changes made in the transaction

resignal; -- raise again the sql exception to the caller

end;

→ il segnale serve per informare il client dell'errore (serca il resignal, l'errore sarebbe considerato come gestito con successo)

→ il gestore è utile per quando in abort la transazione senza attendere il timeout in caso di errore applicativo

(gli errori applicativi non portano a un abort automatico)

set transaction isolation level repeatable read;

start transaction;

if... then condizione di errore applicativo: vera ovunque a seguito di errori di esecuzione del dbms

→ es. e' stata inserita una tupla con un valore inatteso in corrispondenza di un tupla

signal solstate '45001' set message_text = "Si è verificato un errore";

end if;

commit;

→ contiene che rende le operazioni della transazione persistenti e visibili agli altri client

end

E.g.: Elapsed time = 25 mil,
movement of cursor = 1000
SQL statements = 50

Livelli di isolamento:

1) READ UNCOMMITTED

È il livello di isolamento più basso, in cui possono avvenire letture su dati aggiornati ma non comitati (ancora) in commit → Gli statement select sono eseguiti in modalità non bloccante.

Di conseguenza, si possono avere delle letture non coerenti (DIRTY READ).

ES

3)

ESEMPIO:

TRANSAZIONE T1

SELECT age FROM users WHERE id = 1;

TRANSAZIONE T2

UPDATE users SET age = 21 WHERE id = 1;

SELECT age FROM users WHERE id = 1; \rightarrow lettura di un dato non corretto (dato aggiornato e istantaneo)

ROLLBACK;

2) READ COMMITTED

È un livello di isolamento che richiede che le letture avvengano solo sui dati aggiornati da transazioni che sono andate in commit \Rightarrow Il DBMS acquisisce un lock per ogni dato che viene letto o scritto. I lock associati ai dati che sono stati aggiornati in scrittura sono mantenuti fino alla fine della transazione, mentre i lock associati ai dati accediti in lettura sono rilasciati alla fine della singola lettura.

In tal modo, si possono verificare delle anomalie di tipo UNREPEATABLE READS.

ESEMPIO:

TRANSAZIONE T1

SELECT * FROM users WHERE id = 1;

TRANSAZIONE T2

UPDATE users SET age = 21 WHERE id = 1;

COMMIT;

SELECT * FROM users WHERE id = 1; \rightarrow lettura di un dato corretto ma modificato rispetto alla lettura precedente
COMMIT;

3) REPEATABLE READ

\rightarrow Livello di isolamento di default in MySQL, MariaDB, ecc.

È un livello di isolamento che garantisce che leggere delle stesse tuple nella stessa transazione restituiscano sempre dati col medesimo valore \Rightarrow
Come in READ COMMITTED, il DBMS acquisisce un lock per ogni dato che viene

letto o scritto; la differenza sta nel fatto che anche i lock associati ai dati acceduti in lettura sono mantenuti fino alla fine della transazione.
Tuttavia, non vengono gestiti i RANGE LOCK, pertanto possono verificarsi anomalie di tipo PHANTOM READ.

ESEMPIO:

TRANSAZIONE T₁

SELECT count(*) FROM users
WHERE age >= 17 and age <= 21; \leftarrow Questa query restituisce un valore x
INSERT INTO users (age) VALUES (20);

SELECT count(*) FROM users
WHERE age >= 17 and age <= 21; \leftarrow Questa query restituisce x+1

TRANSAZIONE T₂

Effettivamente i lock ricoprono le singole tuple, non le intere tabelle, per cui la transazione T₂ non ha dovuto rimanere in attesa

4) SERIALIZZABILE

È il livello di isolamento + alto, in cui, ogni volta che si effettua una query che lavora su un intervallo di valori, si acquisisce un RANGE LOCK, che è una struttura dati aggiuntiva che permette di bloccare un intervallo di valori su uno o + attributi (piuttosto che singole righe).

L'utilizzo dei range lock inficia sulle performance del sistema \Rightarrow

Serializable, più degli altri livelli di isolamento, va usato con parsimonia.

\rightarrow PROCEDURA CON UNA SOLA INSERT: non ha bisogno della definizione di una transaz.

\rightarrow PROCEDURA CON DUE INSERT: ha bisogno di una transazione se le due insert sono logicamente correlate; in tal caso, poiché non ci sono bottine, conviene ricorrere al livello di isolamento READ UNCOMMITTED, in modo tale da non far schiantare le performance invitilmente.

Soff.
• AF

Mobilità di accesso transazionali:

- READ ONLY: la transazione si limita alla lettura di dati.
- READ WRITE: la transazione legge e scrive dati.

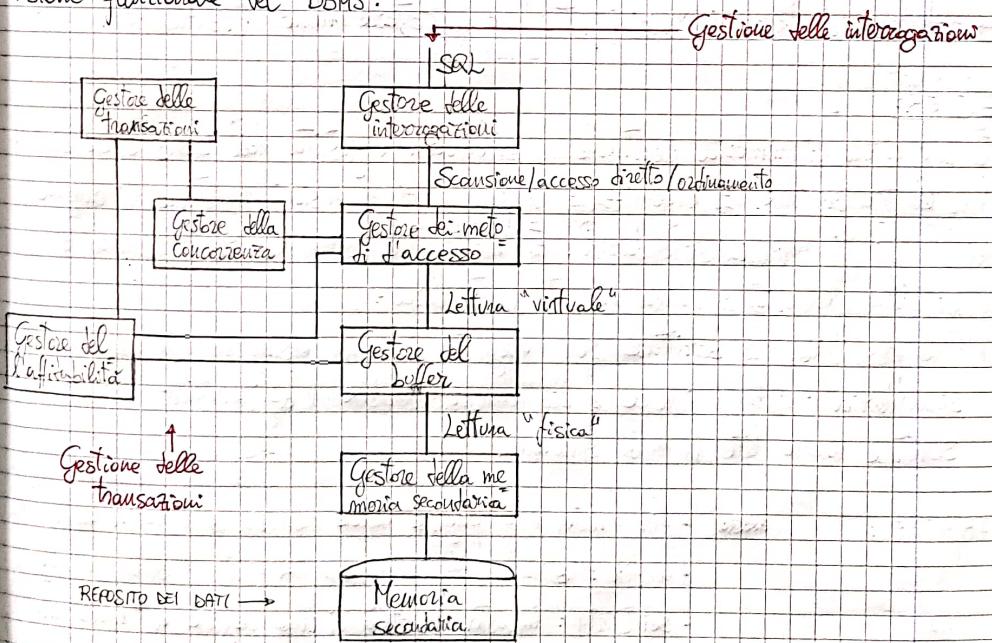
Dalle queste informazioni al DBMS consente di generare dei piani di esecuzione ottimizzati dal punto di vista dell'acquisizione dei lock.

↳ Sono informazioni ortogonali al livello di isolamento richiesto.

09/12/2020

DBMS:

Visione funzionale del DBMS:



Sappiamo che le basi di dati sono:

► AFFIDABILI

► CONDIVISE

↳ gestione della concordanza

① Nei consideriamo buffer con dim pagine = dim blocchi.

Gestione della memoria secondaria e del buffer;

- I dispositivi di memoria secondaria sono organizzati in blocchi di lunghezza fissa (ordine di grandezza: alcuni KB).
- Le uniche operazioni sui dispositivi sono la lettura e la scrittura di una PAGINA, cioè dei dati di un blocco.

- → Il costo di un accesso ⁱⁿ memoria secondaria è almeno lo stesso di gran
forse maggiore di quello per le operazioni in memoria centrale. Però, nel
le applicazioni I/O Bound, il costo dipende esclusivamente dal numero di
accessi in memoria secondaria.
- → Accessi a blocchi "vicini" tendono a costare meno (contiguità).

BUFFER:

È un'area di memoria centrale, preallocata e gestita dal DBMS e condivisa
fra le transazioni. È organizzato in pagine di dimensioni multiple di quelle
dei blocchi di memoria secondaria. È molto utile per ridurre il numero di ac-
cessi alla memoria secondaria. In particolare:

- Ove possibile, le scritture sono eseguite in modo asincrono in memoria
secondaria; quando invece ciò comprometterebbe l'affidabilità dei dati, le
scritture vengono fatte in modo sincrono.

PRIMITIVE:

eseguite dal
buffer manager
MA

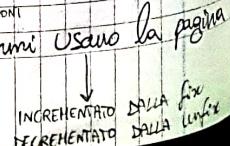
richieste dalle
applicazioni sovrastanti
(con livelli sovrastanti)

PICHIESTE DAL GESTORE DELL'AFFIDABILITÀ,
NON DAL LETTORE REGGIA ACCESSA

- fix: porta una pagina dalla memoria secondaria al buffer
- unfix: indica che la transazione ha finito di usare una pagina
- set Dirty: imposta lo stato di una pagina a "modificata"
- force: esegue una scrittura sincrona in memoria secondaria
- flush: esegue una scrittura asincrona in memoria secondaria

VARIABILI DI STATO:

- contatore che indica quanti programmi usano la pagina
- dirty bit



- * → informazione di controllo relativa alla struttura fisica
- * → informazione di controllo del file system

ESECUZIONE DELLA FIX:

- Cerca la pagina nel buffer
- Se c'è, restituisce l'indirizzo
- Altrimenti, cerca una pagina libera nel buffer (il contatore a zero)
 - Se la trova, restituisce l'indirizzo
 - Altrimenti, si hanno 2 alternative:
 - STEAL: selezione di una vittima tra le pagine nel buffer
 - NO-STEAL: l'operazione viene posta in attesa

FILE SYSTEM:

I DBMS utilizzano le funzionalità del file system, ma in misura limitata, per CREARE ed ELIMINARE file e per LEGGERE E SCRIVERE singoli blocchi o sequenze di blocchi contigui.

FATORE DI BLOCCO:

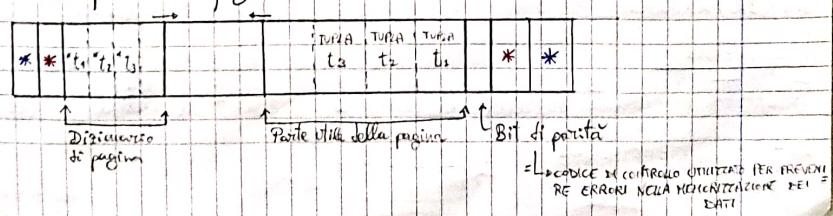
È il numero di record logici che possono trovarsi in un blocco.

- $L_R :=$ dimensione di un record
- $L_B :=$ dimensione di un blocco
- Se $L_B > L_R$, possiamo avere + record in un blocco: $\lfloor L_B/L_R \rfloor$

Lo spazio residuo (se c'è) può essere:

- Utilizzato (record "spanned" o impaccati) → RECORD SPANNED = record suddiviso tra più blocchi
- Non utilizzato (record "unspanned")

Organizzazione delle tuple nelle pagine:



STRUCTURE:

→ **SEQUENZIALI**: le tuple si inseriscono rispettando una sequenza.

- **Serial** → ordinamento fisico ma non logico → LA PIÙ UTILIZZATA; NON RICHIESTA RI-ORGANIZZAZIONE PERIODICA

- **Array** → posizioni individuate attraverso indici → APPLICABILE SOLO SE LE TUPLE HANNO GRANDEZZA FISSA

- **Ordinata** → ordinamento mediante un campo (clustered) → RICHIESTA RI-ORGANIZZAZIONE PERIODICA PER MANTENERE LE PEGGIO DI SPERIMENTARE

→ **AD ACCESSO CALCOLATO (HASH)**: l'accesso diretto è molto efficiente, ma non per accessi ~~casuali~~ basati su range di valori. Le collisioni sono solitamente gestite con blocchi collegati.

→ **AD ALBERO**: si tratta di strutture efficienti sia per accessi basati su range. Vengono utilizzati sia per **STRUTTURE PRIMARIE** (che contengono i dati), sia per **STRUTTURE SECONDARIE** (che contengono gli indici e sono utili per l'accesso ai dati).

, **FUNZIONE HASH**: $val(\text{chiave}) \rightarrow \text{indirizzo}$

→ Si ha una collisione se, per due valori chiavi diversi, la funzione hash restituisce lo stesso indirizzo.

↳ In caso di collisione tra n indirizzi, solo uno di questi viene riportato direttamente nella tabella hash, gli altri $n-1$ indirizzi vengono salvati nella struttura di overflow.

Parliamo sempre di record coincidenti nella stessa collisione

↳ Costo di accesso: 1 per l'indirizzo nella tabella hash;
2 per il primo indirizzo nella struttura di overflow;
...
 n per l'ennesimo indirizzo nella struttura di overflow

Numeri medi di accessi per leggere i dati =
= somma dei numeri di accesso per tutti i dati / numero dei dati

10/12/2020

Definizione:

Sia T il numero di record di un file e sia F il fattore di blocco.

FATTURE DI RIEMPIMENTO

$$0 < f \leq 1$$

$$f := \frac{T}{B}$$

(Dim. spazio utilizzato)

→ Numeri record che possono essere contenuti in un blocco

→ E.g. può essere la dimensione di una traccia hash in termini di numero di record

$$\text{NUMERO DI BLOCCHI NECESSARI PER CONTENERE I RECORD} = \frac{T}{fF} \rightarrow \frac{T}{fF} = \frac{\# \text{ RECORD SPAZIO UTILIZZATO}}{\# \text{ BLOCCO}} \cdot \frac{\text{DIM. RECORD}}{\text{DIM. BLOCCO}} = \frac{\text{DIM. TOTALE IN BYTE}}{\text{DIM. BLOCCO}}$$

Questa è l'organizzazione del FILE HASH: all'interno di ogni blocco vengono inseriti tutti i record legati allo stesso valore dato da una particolare funzione hash a $\frac{T}{fF}$ valori.

Il file hash, rispetto alla funzione hash applicata a una normale tabella hash, reduce anche di molto il numero di collisioni; tuttavia, è comunque utile solo se i dati in questione sono soggetti a una bassa dinamicità (ovvero se la dimensione del file non varia molto nel tempo), altrimenti il numero di overflow (di collisioni) cresce molto.

IN PARTICOLARE IN CASO DI AUMENTO DELLE COLLISIONI

Supponiamo di avere una tabella i cui record sono ordinati rispetto a un campo A e di voler effettuare una ricerca basata su un altro campo B. Si ha il problema per cui i record, rispetto al campo B, sono in ordine casuale, per cui sarebbe necessario esaminare TUTTI i record della tabella \Rightarrow COSTO TEMPORALE ECCESSIVO.

Non è conveniente effettuare una copia della tabella e ordinare i record rispetto al campo B, perché potrebbe a un uso eccessivo dello spazio, a tempo in più per la sincronizzazione e a una potenziale inconsistenza.

Possibili soluzioni:

► STRUTTURE PRIMARIE ORGANIZZATE AD ALBERO

► INDICI

JUSTICI: \rightarrow Sono delle strutture ad albero che permettono di accedere al record sia in modo puntuale, sia a intervalli di valori efficientemente.

► PRIMARI: sono applicati su un campo sul cui ordinamento è basata la memorizzazione.

► SECONDARI: sono applicati su un campo con ordinamento diverso da quello di memorizzazione.

► Densi: contengono tutti i valori della chiave.

► Sparsi: contengono solo alcuni valori della chiave e quindi un numero di riferimenti inferiore rispetto ai record del file.

Un indice primario:

→ Di solito è sparso.

→ Se denso, permette di eseguire operazioni sugli interi, senza accedere ai record.

Un indice secondario deve essere denso.

→ Ogni file può avere al più un indice primario e un qualunque numero di indici secondari (su campi diversi).

DIMENSIONI DELL'INDICE:

Siamo:

- L = # record nel file
- B = dim. blocchi
- R = lunghezza dei record
- K = lunghezza del campo chiave
- P = lunghezza degli indirizzi (ai blocchi)

$$\rightarrow \# \text{ BLOCCHI PER FILE} = N_F \approx L / (B/R)$$

$$\rightarrow \# \text{ BLOCCHI PER UN INDICE DENSO} = N_d = L / (B/(K+P))$$

$$\rightarrow \# \text{ BLOCCHI PER UN INDICE SPARSO} = N_s = N_F / (B/(K+P))$$

* Dove:

$$\begin{aligned} B/R &= \# \text{ record del file in un blocco} \\ B/(K+P) &= \# \text{ elementi di indice in un blocco} \end{aligned}$$

È chiaro come in un blocco

si possono mettere + elementi di indice che record interi

Gli indici sono utili perché $K+P < R$, per cui un blocco può contenere molti elementi di indice

$$N_s < N_d < N_F$$

→ È anche possibile utilizzare indici a + livelli (indici multilivello).

IB: Un indice sull'indice è sempre sperso. → altrimenti non avrebbe senso di esistere comunque sia, gli indici utilizzati dai DBMS sono sofisticati per avere una maggiore flessibilità in presenza di elevata dinamicità.

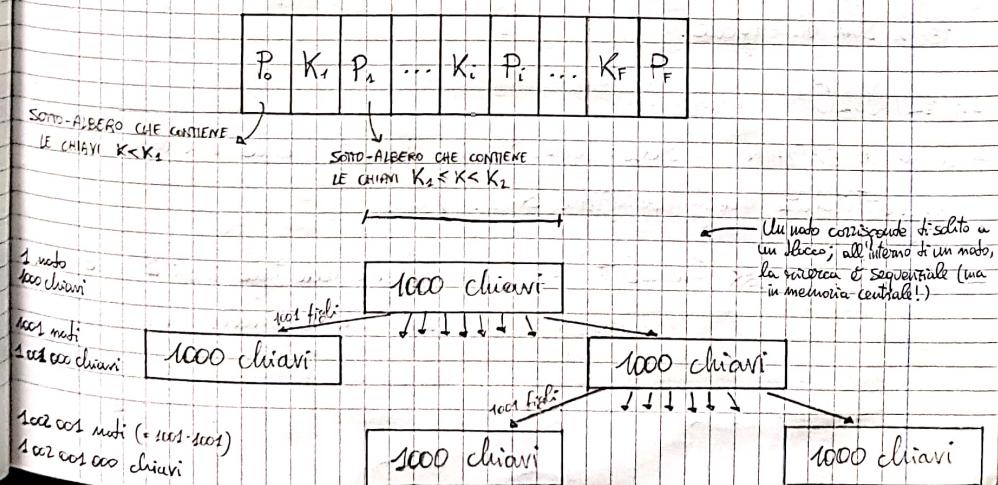
In particolare, introduciamo gli indici dinamici multilivello tramite i B-tree (intuitivamente: alberi di ricerca bilanciati).

ALBERI DI RICERCA DI ORDINE F+1:

- Ogni nodo ha fino a $F+1$ figli e fino a F etichette ordinate.
- Si hanno 2 vincoli fondamentali:
 - (1) \forall nodo $K_1 < K_2 < \dots < K_F$
 - (2) \forall valore K presente nel sottoalbero puntato da P_i , vale la seguente relazione:

$$\begin{array}{ll} K_i \leq K < K_{i+1} & \text{per } 0 \leq i < F \\ K < K_1 & \text{per } i = 0 \\ K_F \leq K & \text{per } i = F \end{array} \quad \left. \begin{array}{l} \text{ } \\ \text{ } \\ \text{ } \end{array} \right\} \begin{array}{l} i \text{ è l'indice} \\ \text{di } P_i \end{array}$$

• Nell' $i+1$ -esimo sottoalbero abbiamo tutte le etichette maggiori della i -esima etichetta e minori della $i+1$ -esima.



Per inserire (o cancellare) valori di ricerca nell'albero di ricerca, sono necessari specifici algoritmi che garantiscono il rispetto dei 2 vincoli fondamentali (ma non garantiscono che un albero di ricerca risulti sempre bilanciato).

In particolare, la cancellazione di record crea nell'albero noti scarsamente popolati, aumentando la dimensione dello spazio utilizzato e, di conseguenza, il numero di blocchi.

→ Un albero di ricerca bilanciato evita che alcuni record richiedano molti accessi a blocco di altri.

B-albero:

È un albero di ricerca con alcuni vincoli aggiornati che garantiscono che l'albero sia bilanciato e lo spazio di memoria che viene utilizzato per effetto della cancellazione di record non diventi troppo grande.

Per soddisfare tali vincoli aggiornati, occorrono algoritmi di inserzione e cancellazione di record più complessi, soprattutto nei casi di:

→ Inserzione di un record in un modo pieno.

→ Cancellazione di un record da un modo che diventa pieno per meno del 50% della sua dimensione.

11/12/2020

PELLEGRINI

→ Comando EXPLAIN di SQL: descrive le performance e i dettagli di una particolare query / operazione SQL.

→ where DAYOFWEEK(GiornoSett) = 5 L'ordinatore riserva del MySQL (non è standard) ⇒ selezione delle tuple in cui il giorno della settimana è giovedì.

→ Vedere la documentazione di MySQL per saperne di più su come manipolare i tipi di dato relativi alle date.

→ group by YEARWEEK (GiornoSett)

Lo comando che enumera le settimane in un anno

→ having count(case (A2.Nazione <> 'Italia') when true then 1 else NULL end)

= 0

← Come selezionava solo i valori che apparivano in Italia tramite l'operatore count.

16/12/2020

Un B-tree è un albero di ricerca che viene mantenuto bilanciato, grazie a:

- Riempiimento parziale dei nodi (mediamente al 70%)
- Riorganizzazioni (locali) in caso di sbilanciamento

SPLIT e MERGE:

I inserimenti ed eliminazioni dall'albero sono preceduti da una ricerca fino a una foglia.

► INSERIMENTO:
• Se c'è posto nella foglia \Rightarrow OK

• Altrimenti il nodo va sovraccarico, con necessità di un puntatore in più per il nodo genitore; se non c'è posto, si sale ancora, eventualmente fino alla radice.

• In ogni caso, il riempimento di ciascun nodo è sempre almeno al 50%.

► ELIMINAZIONE:
• Se il riempimento della foglia ~~scende~~ rimane superiore o uguale al 50% \Rightarrow OK

MERGE
• Altrimenti il nodo va accorpato con altri.

► AGGIORNAMENTO: diminuzione inserimento

B+ tree vs B tree:

→ B+ TREE:

Le chiavi compaiono tutte nelle foglie (ripetizioni).

Le foglie sono collegate in una lista.
tra loro

→ B TREE: Le chiavi che compiono nei nodi interni non sono ripetute
nelle foglie.

(e nel nodo radice)

I nodi interni possono avere puntatori direttamente ai dati.
Le porci si hanno complessivamente 2 puntatori Y, X →

Tori
Ric

e
vol

- In un B+ tree primario, le tuple possono essere contenute nelle foglie.
- In un B+ tree secondario, le foglie contengono puntatori alle tuple.
- In un B tree, anche i nodi interni contengono tuple (se primari) o puntatori (se secondari).

• STRUTTURA INDEX-SEQUENTIAL: le foglie contengono le tuple.

• STRUTTURA INDIRETTA: file con indice secondario.

Definizione degli indici in SQL:

→ create [unique] index IndexName on TableName (AttributeList)

→ drop index IndexName

PROGETTAZIONE FISICA

INPUT: schema logico + informazioni sul carico applicativo

OUTPUT: schema fisico, costituito dalla definizione delle relazioni con le relative strutture fisiche (e molti parametri, spesso legati allo specifico DBMS).

La caratteristica comune dei DBMS relativi è la disponibilità degli indici.

→ Le chiavi primarie delle relazioni sono di solito coinvolte in selezioni e join: molti sistemi superiscono di definire indici sulle chiavi primarie.

→ Altri indici vengono definiti con riferimento ad altre selezioni o join "importanti".

→ Se le prestazioni sono insoddisfacenti, si "torna" il sistema aggiungendo o eliminando indici.

→ Comando SQL SHOW PLAN per verificare se e come gli indici sono utilizzati.

Torniamo ancora una volta sulle transazioni:

Ricordiamo che è una parte di programma che inizia con START TRANSACTION e termina con END TRANSACTION, ed è in grado di eseguire una e una sola volta il COMMIT oppure il ROLLBACK.

→ SISTEMA TRANSAZIONALE (OLTP) = sistema in grado di effettuare delle transaz. → carico di un certo numero di transazioni concorrenti

- Le proprietà di ATOMICITÀ e DURABILITÀ sono garantite dal GESTORE DELL'AFFIDABILITÀ.
- La proprietà di ISOLAMENTO è garantita dal GESTORE DELLA CONCORSIONE.
- La proprietà di CONSISTENZA è garantita dal GESTORE DELL'INTEGRITÀ.

Gestore dell'affidabilità:

→ Realizza i comandi transactionali: start transaction (B), commit (C), rollback (A), ricevuti dal gestore delle transazioni.

→ Realizza le primitive di ripristino (recovery) dopo i guasti, dette RIPRESA A CALDO e RIPRESA A FREDDO.

→ Riceve le richieste di accesso alle pag. in lettura e in scrittura, che passa al gestore del buffer, e genera richieste di scrittura/lettura (fix, unfix, force, flush) per garantire robustezza.

→ Salva delle copie di alcune informazioni per poterle recuperare in futuro in caso di guasto (anche tramite il cosiddetto LOG).

PERSISTENZA DELLE MEMORIE:

• MEMORIA CENTRALE: non persistente

• MEMORIA DI MASSA: persistente ma può danneggiarsi

• MEMORIA STABILE: persistente e idealmente non si danneggia (a meno di danni estremamente catastrofici)

→ ESEMPI CHE SI AVVICINANO: dispi replicati, nastri, ...

mirrored

LOG:

È un FILE SEQUENZIALE gestito dal controllore dell'affidabilità e scritto in memoria stabile.

È una sorta di diario di bordo: riporta in ordine tutte le operazioni eseguite sulla base di dati.

→ RECORD DI TRANSAZIONE:

- begin, B(T) → indice della transaz. che è partita
- insert, I(T, O, AS) → oggetto inserito e il suo stato AS
- delete, D(T, O, BS) → stato dell'oggetto prima dell'elim.
- update, U(T, O, BS, AS)
- commit, C(T) before state → after state
- abort, A(T)

→ record di log nel log di transazione
in particolare quando è stata salvata la transazione

→ RECORD DI SISTEMA:

- dump → è una copia completa ("di riserva") della BD
- checkpoint → indica quali sono le transazioni attive in un dato istante di tempo

17/12/2020

AGGIUNTA DI UN

MODALITÀ + SEMPLICE DEL CHECKPOINT:

- Si sospende l'accettazione di richieste di ogni tipo di operazione.
- Si trasferiscono in memoria (tramite force) tutte le pagine sporche relative a transazioni andate in commit.
- Si scrive nel ~~log~~ log in modo sincrono (force) un record di checkpoint (identificatore delle transazioni in corso).
- Si ripete l'accettazione delle operazioni.

ESITO DI UNA TRANSAZIONE:

È determinato irrevocabilmente quando viene scritto il record di commit nel log in modo sincrono con una FORCE.

• GUASTO PRIMA DEL COMMIT → può portare a un UNDO di tutte le azioni, per ricostruire lo stato originario della base di dati.

• GUASTO SUCCESSIVO AL COMMIT → può portare a un REDO, per annullare gli effetti del guasto.

D'altra parte, il record di ABORT può essere scritto in modo asincrono.

UNDO / REDO:

→ UNDO DI UN'AZIONE SU UN OGGETTO O

• Update, delete: $O = \text{val}(\text{BS})$

• Insert: eliminare O

→ REDO DI UN'AZIONE SU UN OGGETTO O

• Insert, update: $O = \text{val}(\text{AS})$

• Delete: eliminare O

Nessuno esclude che, anche durante le azioni di undo e redo, si possa verificare un guasto. Per fortuna, queste due operazioni godono della proprietà di temperanza:

$$\cdot \text{undo}(\text{undo}(A)) = \text{undo}(A)$$

$$\cdot \text{redo}(\text{redo}(A)) = \text{redo}(A)$$

REGOLE FONDAMENTALI PER IL LOG:

• Write-Ahead-Log: si scrive prima nel log e poi nella BD (altrimenti non si potrebbe disfare le azioni).

• Commit-Pacecerenza: si scrive il log (parte AS) prima del commit (altrimenti non si potrebbe riparare le azioni).

Ma esattamente quando scriviamo nella base di dati?

→ 1° POSSIBILITÀ: tutte le pag. della base di dati vengono aggiornate sempre prima del commit (MODALITÀ IMMEDIATA) → un guasto dopo il commit non comporta conseguenze, ma è sempre necessario fare l'UNDO x guasti prima del commit.

→ 2° POSSIBILITÀ: tutte le pag. della base ti fatti vengono aggiornate sempre dopo il commit (MODALITÀ DIFFERITA) ⇒ un guasto prima del commit non comporta conseguenze, ma è spesso necessario fare il REDO e guasti dopo il commit (NON lo è nel caso in cui il commit sia avvenuto prima di un checkpoint).
 ↴ QUESTO PERCHÉ CHECKPOINT ⇒ SCRITTURA SU DISCO

→ 3° POSSIBILITÀ: si lascia che sia il buffer a decidere come gestire le scritture (che possono essere sincrone o asincrone) (MODALITÀ MISTA) ⇒ bisogna ricordare (anche se nel tempo) sia alle operazioni di UNDO, sia alle operazioni di REDO.

RIPRESA

Qual

→ Tra

→ Cos

→ Rip

fra

→ Ri

Si poi

e RE

Si ac

nel

COSTI:

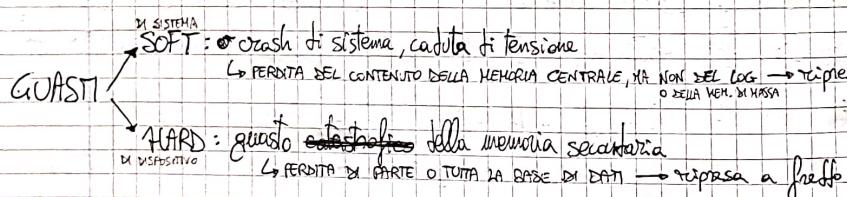
Tutti e tre gli schemi rispettano le tre regole fondamentali e scrivono il record di commit in modo sincrono.

Il costo delle scritture nel log è paragonabile a quello degli aggiornamenti della BD.

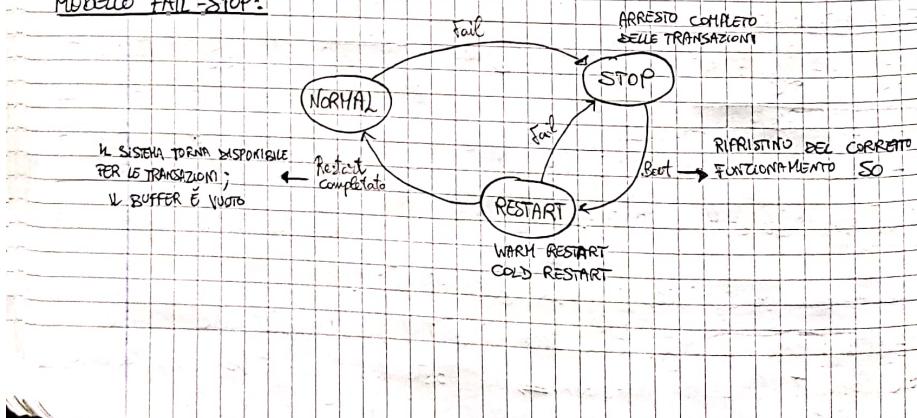
RIPRESA

È un

è stato



MODELLO FAIL-STOPO



18/1

Figg

Due

J +

effett

RIPRESA A CALDO:

Quattro fasi:

- Trovare l'ultimo checkpoint (ripercorrendo il log a ritroso).
- Cognoscere gli insiemi UNDO (transaz. da disfare) e REDO (transaz. da rifare).
- Ripercorrere il log (avanti + indietro), fino alla + vicina azione delle transaz. in UNDO e REDO, disfaccendo tutte le azioni delle transaz. in UNDO.
- Ripercorrere il log in avanti, rifacendo tutte le azioni delle transaz. in REDO.

Si garantisce l'atomicità delle transazioni proprio grazie alle operazioni di UNDO e REDO.

Si garantisce la persistenza delle transazioni grazie alla scrittura del commit nel log in maniera sincrona e del REDO delle transazioni andate in commit.

RIPRESA A FREDDO:

È una ripresa a caldo preceduta dal ripristino della parte della base di dati che è stata deteriorata. Le fasi sono:

- Si accede al + recente record di dump e si ricopia la parte deteriorata del DB.
- Si ripercorre il log in avanti, applicando sia le azioni sulla BD, sia le azioni di commit (relativamente alla parte di BD deteriorata).
- Si esegue una ripresa a caldo.

18/12/2020

FIRECRICK

HIGGERS:

È un insieme di statement SQL che vengono attivati sotto certe condizioni.

Due varianti: → Esegue UN'ASSEGNAZIONE BEFORE: viene lanciato prima di eseguire il comando SQL associato

→ AFTER: viene lanciato dopo l'esecuzione del comando SQL associato → COMANDO DI HIGGER (UPDATE-INSERT-DELETE)

I trigger sono per singola tabella; sono utili per implementare assicurazioni ed effettuare controlli che, altrimenti, non sarebbero supportati da SQL.

`SELECT ... INTO variabili` → Colonna sarà che funziona solo se il risultato della select è una tabella
con una sola riga; inoltre, si troverà avere 1 variabile diversa per ogni colonna

ATTIVAZIONE DELLO SCHEDULER
DEGLI EVENTI TEMPORIZZATI

set global event_scheduler = on; → ATTIVAZIONE valida per la
singola istanza di MySQL

attivazione sul file my.cnf

Ba fai il

MyS

COME

→ COMANDO `on schedule`
`every 2 day`

⇒ l'evento deve avvenire un'unica volta tra 2 giorni

gC

→ COMANDO `on schedule`
`every 2 day`
`on completion preserve`

⇒ l'evento deve avvenire periodicamente ogni
2 giorni

07/

Comh

Uu

utenti

Cam

⇒ NOW(): data/time che indica l'istante corrente.

↳ ESEMPI DI FORMA FA = Now() - interval 2 day

Funzione:

A differenza di una procedura, NON restituisce una tabella; è fa riferirsi in senso
classico: ha un insieme di parametri in input e restituisce un valore di
output.

A differenza delle procedure, non vuole che se i parametri venga specificato
IN/OUT/INOUT.

→ È un'appiglio per calcolare del codice che va eseguito a fine

→ COMANDO 'stringa' `regexp` 'espressione regolare'

→ VERIFICA SE LA STRINGA HA UNA
DETERMINATA STRUTTURA *

DELIMITATORE: è un carattere che indica qual è il carattere di fine riga / fine funz.
(e.g. nel linguaggio C è ';')

COMANDO `delimiter !` → imposta '!' come delimitatore.

→ È utile per non far interrompere a metà un comando (che si vuole eseguire automaticamente) in prossimità di ';

→ CICLO SQL: inizia con loop e termina con end loop → È EQUIV. A WHILE(1)

Cursore:

Serve a estrarre una tupla alla volta dalla tabella puntata dal cursore stesso.

→ ESSERE UTUZZATO, HA BISOGNO DEL COMANDO `open` Dopo LA SUA DICHIARAZ. (dichiaro un cursor for select nome
from tabella)

* ESEMPIO DI ESPRESSIONE REGOLARE: `[a-zA-Z0-9][a-zA-Z0-9.-]*[a-zA-Z0-9.-]@`

`[a-zA-Z0-9][a-zA-Z0-9.-]*[a-zA-Z0-9]\.[a-zA-Z]{2,63}$`

WRITE

X HELL

MEMORI

* ESEMPIO DI UTILIZZO
DEI CURSORI:

```

    raw_loop: loop
        fetch cur into var_name;
        if done then
            leave raw_loop;
        end if;
    end loop;

```

STESO UTILIZZO DELLA CLASSICA SELECT...MA HA CON LA POSSIBILITÀ DI TRATTARE + DI 1 TUPA

RISALICE RIGA DI CODICE + SERA POSSIBILE TROVARE:
delusa continua handles su un found set dove = true
+ gestire di continuare per l'esecuzione "not found set"
NB: qui non viene dichiarata done!

MySQL C Connector:

COME COMPILARE (DOPO AVER INSTALLATO LE LIBRERIE DI MYSQL):

gcc -g *.c -o program `mysql_config --cflags --include --libs`

07/01/2021

Controllo di concorrenza:

Un DBMS deve servire diverse applicazioni e rispondere a + richieste provenienti da utenti diversi \Rightarrow è impensabile un'esecuzione seriale.

Ciononostante, la concorrenza può far luogo ad ANOMALIE.

ESEMPIO:

T ₁	T ₂	→ Transactioni
$X \leftarrow X + X;$	$X = X^{**} 2;$	
$X = X + 2;$	$X = X + 2;$	

ESECUZIONE CONCERENTE:

$$S = [T_1 : x = x + x; \quad T_2 : x = x^{**} 2; \quad T_1 : x = x + 2; \quad T_2 : x = x + 2]$$

\hookrightarrow Per $x=5$ dà come risultato $x=104$ —

È un risultato diverso da tutte le esecuzioni seriali: NON VA BENE

Verifichiamo in dettaglio qualche esempio di anomalia.

PERDITA DI AGGIORNAMENTO:

Avviene quando si hanno due scrittore (w-w) sullo stesso dato. Nella esecuzione concorrente, una delle due viene persa.

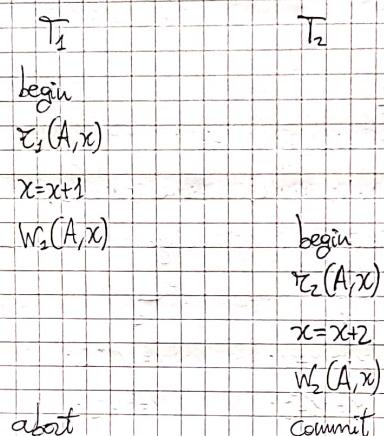
ESEMPIO:

T ₁	T ₂	La transazione t ₁ viene persa!
PER L'OGGETTO A NELLA MEMORIA SECONDARIA, SIENZA IL VALORE NELLE VARIABILI x begin $T_1(A, x)$ $x = x - 1000$ — WRITE LOZ NELLA TABELLA L'OGGETTO A, SOSTANZIALMENTE IN MEMORIA SECONDARIA → $W_1(A, x)$ commit	begin $T_2(A, x)$ $x = x - 1000$ — $W_2(A, x)$ commit	

LETTURA SFORCA:

Avviene quando una transazione legge un dato aggiornato ($r-w$) da una transazione che successivamente avrà in abort.

ESEMPIO:



INSEI

LETTURA INCONSISTENTE (NON-REFEATABLE READ):

Si può avere questo + lettura ($r-w$) dello stesso dato nel contesto della stessa transazione. Per evitare questa anomalia, è opportuno garantire che le transazioni ^{FORM} _{UNI} abbiano esattamente lo stesso valore, non risentendo dell'effetto di altre transazioni. Schi

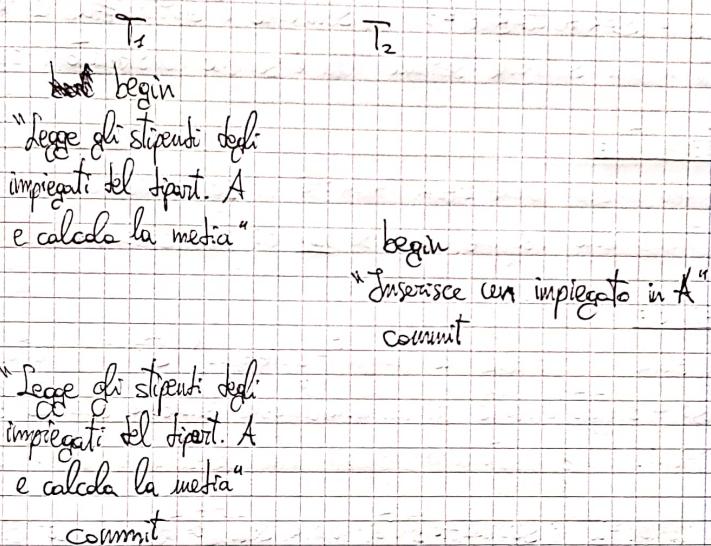
VIOLAZIONE DEI VINCOLI DI INTEGRITÀ / AGGIORNAMENTO FANTASMA:

Si ha, ad esempio, quando due valori A, B sono soggetti a un vincolo di integrità (e.g. $A+B=1000$), una transaz. t_1 legge i valori di A e fa ^{→U} _{FACC} B e una transaz. t_2 , prima che t_1 leggesse A, B ma dopo che t_1 abbia letto A, aggiorna i valori di A, B mantenendo il vincolo di integrità.

→ Dal punto di vista di t_2 il vincolo di integrità è violato! Schi

INSEGNAMENTO FANTASIA:

a Si ha in un caso come il seguente:



sa

TERMINALIZZAZIONE DELLA NOZIONE DI TRANSAZIONE:

l Una TRANSAZIONE è una sequenza di azioni di lettura e scrittura.

Schedule:

È una sequenza di operazioni di I/O relative a un insieme di transazioni.
→ Una schedule è SERIALE se una transazione termina prima che la successiva inizi.

FACCIAHO PER ORA UN'IPOTESI SEMPLIFICATIVA: consideriamo la commit-proiezione:
tutte le transazioni vanno a buon fine e ignoriamo quelle che vanno in abort,
rimuovendo tutte le loro azioni dalla schedule.

Il nostro obiettivo è individuare opportune condizioni da impostare agli schedule per
garantire che l'esecuzione delle corrispondenti transazioni sia corretta
(evitare le anomalie).

→ SCHEDULER = sistema che accetta o rifiuta (o coordina) le operazioni richieste viene
dalle transazioni.

→ SCHEDULE SERIALIZZABILE = uno schedule è serializzabile \Leftrightarrow è equivalente a uno schedule seriale (che è sempre considerato corretto).

Serializzabilità:

RIPRENDIAMO LA DEFINIZIONE DELLA PROPRIETÀ DI ISOLAMENTO:

○ogni transazione esegue come se non ci fosse concorrenza.

Definizione:

L'esecuzione di uno schedule (commit-proiezione) Si è corretta quando produce NB: lo stesso risultato prodotto da un qualsiasi schedule seriale S_j delle stesse N.F. transazioni. In questo caso, diremo che lo schedule Si è serializzabile.

○La garanzia di serializzabilità va a scapito del livello di concorrenza. Di possibile tra transazioni che accedono alle stesse risorse.

○In compenso, sarà possibile definire con precisione i compromessi fra throughput e isolamento.

LIVELLI DI ISOLAMENTO IN SQL:

Read uncommitted - Read committed - Repeatable read - Serializable

RUOLO DELLO SCHEDULER:

- Ha il compito di garantire l'isolamento.
- Accoglie una transazione e le assegna un identificatore unico.
- Chiedere al buffer manager del DBMS di leggere/scrivere sul db secondo il ma particolare sequenza.

IDEA DI BASE: Individuare classi di schedule serializzabili che siano sotto-litici sia verificabile a costo basso.

ANALIZZIAMO LE PRINCIPALI CLASSI DI SCHEDULE SERIALIZZABILI

View-Serializzabilità:

- $R_i(x)$ LEGGE-DA $W_j(x)$ in uno schedule $S \Leftrightarrow W_j(x)$ precede $R_i(x)$ & non vi è $W_k(x)$ fra $W_j(x)$ e $R_i(x)$.
- $W_i(x)$ in uno schedule S è SCRITTURA FINALE se è l'ultima scrittura dell'oggetto x in S .
- Schedule VIEW-EQUIVALENTI ($S_i \approx S_j$): hanno la stessa relazione LEGGE-DA e le stesse SCRITTURE FINALI.
- Una schedule è VIEW-SERIALIZZABILE (VSR) se è VIEW-EQUIVALENTE a uno schedule finale.

NB: Decidere sulla View-serializzabilità di uno schedule è un problema

NP-completo e non è applicabile nella pratica.

Conflict-Serializzabilità:

- Due azioni consecutive in uno schedule sono in conflitto se, cambiando il loro ordine, il comportamento di almeno una delle transazioni coinvolte può cambiare.
→ Un'azione a_i è in CONFLITO con a_j se:
 $\exists t \& a_i, a_j$ operano sullo stesso oggetto $\&$ almeno una delle due azioni è una scrittura.

DUE CASI:
• conflitto read-write (rw / wr) → scambiando l'ordine tra lettura e scrittura
• conflitto write-write (ww) → il valore finale, in generale, cambia
 a seconda dell'ordine delle scritture

08/01/2021

- Schedule CONFLICT-EQUIVALENTI ($S_i \approx S_j$): includono le stesse operazioni e ogni coppia di operazioni e ogni coppia di operazioni in conflitto compare nello stesso ordine in entrambi.
- La CONFLICT-EQUIVALENZA è condizione sufficiente per la serializzabilità.
- Si può essere trasformato in S_j mediante una sequenza di scambi (swaps) tra azioni non in conflitto fra loro.

- Una schedula è **CONFLICT-SERIALIZZABILE** (CSR) se è conflict-equivalente a un qualche schedule seriale.

NB: Schedule come $\tau_1(x) w_2(x) \tau_1(x) w_3(x)$, è **VIEW-SERIALIZZABILE** ma non ~~conflict~~ **CONFLICT-SERIALIZZABILE** \Rightarrow La classe VSR include la classe CSR.

VINCOLI:

- Tutte le
- Tutte le

VERIFICA DI CONFLICT-SERIALIZZABILITÀ:

→ Per mettere del GRAFO DEI CONFLITTI:

• Un nodo per ogni transaz. t_i .

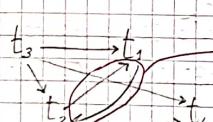
• Un arco orientato da t_i a t_j se c'è almeno un conflitto fra un'azione a_i e un'azione a_j tale che a_i precede a_j .

TEOREMA: Una schedula è in CSR \Leftrightarrow il grafo è aciclico.

ESEMPIO: $S = w_3(A) w_2(C) \tau_1(A) w_3(B) \tau_1(C) w_2(A) \tau_1(A) w_4(E)$

PROPRIETÀ

- 1) Ogni critica
- 2) Ogni



→ C'è un ciclo: il grafo non è conflict-senzabil

NB:

TERZA PRO
In ogni

Nella pratica, la conflict-senzabilità è più rapidamente verificabile (tempo lineare) ma inutilizzabile in pratica, dato che, in un caso medio, si deve ricorrere a grafi di 500 nodi (si hanno +/- 500 transaz. attive in un secondo ciascuna delle quali può effettuare decine di operazioni).

Tra l'altro, la tecnica vista richiede l'ipotesi di commit-proiezione (che non va bene nella pratica).

→ Si utilizzano dunque tecniche che garantiscono la conflict-senzabilità senza dover costruire il grafo e senza richiedere l'ipotesi della commit-proiezione.

: Controllo di concorrenza basato sui lock:

→ Definizione di un protocollo che garantisce la conflict-senzabilità con l'uso dei lock.

mo Ogni
il vi

ESEMI

⇒ La
- 2PL Ra
- 2PL me

VINCOLI:

- Tutte le letture sono precedute da rl -lock (lock condiviso) e seguite da unlock.
- Tutte le scritture sono precedute da wl -lock (lock esclusivo) e seguite da unlock.

↳ QUANDO UNA RISORSA NON È CONCESSA, LA TRANSAZIONE RICHIESTANTE È POSTA IN ATTESA FINO A QUANDO LA RISORSA NON DIVENTA DISPONIBILE

PROPRIETÀ DELLE TRANSAZIONI BEN FORMATE / SCHEDULE LEGALI:

- Ogni azione $p(A)$ [lettura o scrittura di A] deve essere contenuta in una "sezione critica" definita da una coppia di lock:

$T_i: \dots l_i(A) \dots w_i(A) \dots u_i(A) \dots \rightarrow$ Transazione ben formata

- Ogni coppia (lock, unlock) in uno schedule deve essere esclusiva:

$S = \dots l_i(A) \dots u_i(A) \dots \leftarrow$ Schedule legale
Non ci sono altri $l_j(A)$

↳ NB: Queste 2 regole, da sole, non sono sufficienti a garantire serializzabilità.

TERZA PROPRIETÀ:

- In ogni transazione tutte le richieste di lock precedono tutti gli unlock:

$T_i: \dots l_i(A) \dots u_i(A) \dots \rightarrow$ Two-Phase Locking (2PL)

↳ Ogni schedule 2PL è anche conflict-serializzabile ma non vale successivamente il viceversa.

ESEMPIO: $T_1(x) W_1(x) T_2(x) W_2(x) T_3(y) W_3(y)$ NON può essere una schedule 2PL
ma è in CSR



⇒ La classe CSR contiene la classe 2PL.

- 2PL risolve: perdita di aggiornamento, aggiornamento fantasma, letture inconsistenti.
- 2PL non risolve: lettura sporca, inserimento fantasma.

Inoltre, ZPL comporta il rischio di ROLLBACK A CASCATA ("effetto domino").
se ti ha letto un dato scritto da T_k e T_k fallisce \Rightarrow anche tu devi fallire.

- RTM(x)

x: ULT

- WTM(x)

TURA

do schedi

• read(x)

EVITIAMO EFFETTO DOMINO E LETTURE SPORCHE:

- LETTURE SPORCHE: NO commit fino al commit di tutte le transazioni di cui ha letto dati recuperabili (RECOVERABLE).

- ROLLBACK A CASCATA: NO lettura di dati scritti di transazioni che non sono ancora state in commit.

INTRODUCIAMO COSÌ IL LOCKING A DUE FASI STRETTO.

→ Condizione aggiuntiva: i lock possono essere rilasciati solo dopo il commit o abort.

• write (i)

→ Sopra la necessità dell'ipotesi di commit-proiezione ed elimina il rischio di lettura sporche.

Lo G

↳ E l'inserimento fantasma?

→ LOCK DI PREDICATO, che impedisce anche la scrittura di nuovi oggetti che sovrappongono il predicato (e.g. where age >= 17 and age <= 21). perciò commit

- TIMESTAMP

Controllo di concorrenza basato su timestamp:

È una tecnica alternativa al ZPL.

- TIMESTAMP = identificatore che definisce un ordinamento totale sogli: eventi di un sistema.

→ QUERY AV

• mysql

• mysql

• mysql-fil

• mysql-fse

• METABASE

→ Le transaz. sono naturalmente ordinate secondo l'ordine di arrivo ed eseguono liberamente ogni operazione r/w , lo scheduler controlla che i timestamp delle transazioni coinvolte non violino l'ordinamento seriale: se lo violano, le uccide.

→ PROBLEMI:

• mysql

• SELECT

• INSERT

• PASSAGGI PE

! Lo scheduler ha 2 contatori RTM(x), WTM(x) \forall oggetto x:

- $RTM(x)$ uguale al timestamp ts + f grande fra i ts delle transaz. che hanno letto x .

- $WTM(x)$ uguale al timestamp ts della transaz. t che ha eseguito l'ultima scrittura di x .

I scheduler riceve richieste di lettura e scrittura (con indicato il timestamp della transazione).

.read(x, ts): se $ts < WTM(x) \rightarrow$ richiesta respinta e transazione occisa;

altriam. richiesta accolta e $RTM(x) = \max\{RTM(x), ts\}$.

.write(x, ts): se $ts < WTM(x)$ o $ts < RTM(x) \rightarrow$ richiesta respinta e transazione occisa;

altriam. richiesta accolta e $WTM(x) = ts$.

↳ Questo meccanismo funziona sotto l'ipotesi di commit-protezione.

• TIMESTAMP risolve: lettura inconsistenti, aggiornamento fantasma, perdita di aggiornamenti.

• TIMESTAMP non risolve: lettura sporche

 necessità di bufferizzazione

 priz. delle transaz. prima del commit delle transaz. che è ancora in scrittura.

APPENDICE [Capitolo Connector C]

~ QUERY AL DB CEC. COMANDO SELECT

MN: $\text{mysql_stmt_result}()$ → EFFETTUÀ UNA COPIA DELL'ULTIMO RESULT SET; È UTILE QUANDO SI VOGLIONO UTILIZZARE + RESULT SET ALLA VOLTA.
 $\text{mysql_use_result}()$ → UTILIZZA DI RETTAMENTE L'ULTIMO RESULT SET; È UTILE QUANDO SI VOGLIE UTILIZZARE UN RESULT SET ALLA VOLTA.
 $\text{mysql_fetch_row}()$ → PERMETTE DI ESTRARRE LE INFORMAZIONI DI UNA RIGA ALLA VOLTA DEL RESULT SET → UTILIZZO DI UNA VARIABILE DI TIPO MySQL Row.
 $\text{mysql_free_result}()$ → DESALLOCA LA MEMORIA ASSOCIASTA AL RESULT SET.
ERAN: $\text{mysql_use_result}(\text{e.g., mysql_multi_rows})$, $\text{mysql_values_fields}()$: http://www.Kitchin.cc/mysql-look/samples-Enc/MySQL_tut_02.html

ALL'INTERNO DI UNA VARIABILE DI TIPO MySQL_RES*

DE + QUERY

~ PROBLEMI DEDUTI ALL'INPUT DELL'UTENTE

ONO: $\text{mysql_real_escape_string}()$ → CARATTERE DI ESCAPE, come nel seguente caso:
 $\text{SELECT * FROM table WHERE name = 'O\'Malley'}$ $\Rightarrow \text{SELECT * FROM table WHERE name = 'O\'Malley'}$ SQL INJECTION
 $\text{SELECT * FROM table WHERE name = 'O\'Malley'}$ $\Rightarrow \text{SELECT * FROM table WHERE name = 'O\'Malley'}$ SQL INJECTION
 $\text{INSERT INTO table (column1, column2, column3) VALUES (?, ?, ?)}$ → È UNO PREPARED STATEMENT (CHE SERVE ANCHE A PREVENTIRE INJECTION)
MESSAGGIO PER USARE I PREPARED STATEMENTS: $\text{mysql_stmt_init}()$ → AVVIA UN GESTORE DI PREPARED STATEMENT
 $\text{mysql_stmt_prepare}()$ → INVIA LO STATEMENT (QUERY INCOMPLETA) AL SERVER PER PRECOMPILARLO
 $\text{mysql_stmt_bind_param}()$ → FORNISCE I PARAMETRI DA FORNIRE AI POSI DEI PLACEMENTS (GHOST KIDS)
 $\text{mysql_stmt_execute}()$ → INVIA I PARAMETRI AL SERVER E FA ESEGUIRE LO STATEMENT
 $\text{mysql_stmt_result_metadata}()$ → RECUPERA LE INFORMAZIONI SUL RESULT SET
 $\text{mysql_stmt_free_result}()$ → PRECARICA IL NEW RESULT SET
 $\text{mysql_stmt_store_result}()$ → PRECARICA IL NEW RESULT SET

→ STRUTTURA PER LA GESTIONE DEL TEMPO

```
typedef struct MYSQL_TIME {  
    unsigned int year, month, day, hour, minute, second;  
    unsigned long second_part; // microseconds  
    bool neg;  
    enum enum_mysql_timestamp_type time_type;  
} MYSQL_TIME;
```

→ RICORDATI DI INVOCARE

memset(variabile_di_tipo MYSQL_TIME, 0, sizeof(...))
PRIMA DI UTILIZZARE QUESTA STRUTTURA

→ CHIAMATA A UNA STORED PROCEDURE

- CALL nome_stored_procedure(?, ?, ?) → INVOCA UNA STORED PROCEDURE (NB: è un prepared statement)
- mysql_stmt_next_result() → AVANZA (E PUNTA) AL RESULT SET SUCCESSIVO OTTENUTO DALLA STORED PROCEDURE
 - ↳ OUTPUT:
 - 1 se non ci sono più result set
 - 0 se c'è almeno un altro result set
 - >0 se si è verificato un errore (e.g. se i result set dell'eventuale invocazione alla stored procedure precedente non sono stati del tutto consumati).