TEORIA

Progettazione di una base di dati:

- Raccolta ed analisi dei requisiti,
  cioè individuare proprietà e funzionalità del sistema;
- Progettazione,
  ovvero individuare l'organizzazione e la struttura della base di dati
  e schematizzare le operazioni sui dati;
- Implementazione
- Validazione e collaudo
- Funzionamento e manutenzione

La fase di progettazione avviene in tre sottofasi:

1) Progettazione concettuale, (Modello E-R)
   cioè la traduzione dei concetti informali risultati
   dall'analisi dei requisiti in uno schema formale indipendente
   dal DBMS usato (generico ma formale)

2) Progettazione logica, (Tabelle)
   cioè la traduzione dello schema concettuale in un modello logico;

3) Progettazione fisica,
   si completa lo schema logico con la specifica dei parametri fisici di
   memorizzazione dei dati;

Strategie di Progetto:

- Top-down: sono una serie di regole di raffinamento che operano sui singoli concetti dello schema e lo trasformano via via in una struttura più complessa da descrivere il concetto con maggiore dettaglio;

- Bottom-up: si suddividono le specifiche iniziali in sottoprogetti più piccoli e limitati, si sviluppano i sottoschemi separatamente, si fondono i sottoschemi per ottenere lo schema finale;

- Generale;
  - Analisi dei requisiti: glossario dei termini, analisi e raggruppamento requisiti;
  - Passo base: prima costruzione di uno schema scheletrico;
  - Passo di decomposizione: decomposizione dei requisiti;
  - Passo iterativo: raffinazione e aggiunta di nuovi concetti;
  - Passo di integrazione: integrazione dei sottoschemi in uno generale;
  - Analisi di qualità;

Normalizzazione: Anomalie

- Se il telefono dello studente cambia allora questo deve essere aggiornato in tutti i record dello studente (Anomalia di aggiornamento);

- Se vengono annullati gli esami dati non rimane traccia dello studente (Anomalia di cancellazione);

- Similmente se uno studente non ha ancora dato esami non può essere inserito (Anomalia di inserimento);

Funzioni scalari

Funzioni a livello di ennupla che restituiscono singoli valori

Temporali

current_date, extract(year from data) Manipolazione stringhe

char_length, lower

Conversione

CAST(X AS TIPO)

Condizionali

...

Funzioni condizionali : CASE, COALESCE, NULLIF

```
SELECT Nome, Cognome, COALESCE(Dipart,'Ignoto') FROM Impiegato
```

```
SELECT Targa,
   CASE Tipo
      WHEN 'Auto' THEN 2.58 * KWatt
      WHEN 'Moto' THEN (22.00 + 1.00 * KWatt) ELSE NULL
   END AS Tassa
FROM Veicolo
WHERE Anno > 1975
```

Operatori algebra relazionale:

SQL e ALGEBR

- Unione $R \cup S$
- Differenza $R - S$
- Intersezione $R \cap S$

} Attr. comuni;

- Prodotto Cartesiano $R \times S$ } Attr. non comuni;
- Ridenominazione $\delta_{A \to A'}(R)$
- Selezione $\sigma_{A \theta B}(R)$
- Proiezione $\pi_{A,B,\ldots}(R)$
- Natural join $R \bowtie S$ } Uno o più attr. comuni;
- Theta join $R \bowtie_{\sigma_f} S$
- Outer join $R \bowtie_{RIGHT} S$ } Uno o più attr. comuni RIGHT / LEFT / FULL;
- Semi join $R \ltimes S$ } Non commutativo;
- Unione esterna $R \overset{\leftrightarrow}{\cup} S$ } Zero o più attr. comuni;
- Divisione $R \div S$ } $XY \div Y \Rightarrow X$;

Regola Intersezione - Unione:

Trovare gli impiegati che nel 2018 non hanno avuto benefit ma hanno avuto gadget...

CONDIZIONE 1 E CONDIZIONE 2 $\Rightarrow \cap$

CONDIZIONE 1 O CONDIZIONE 2 $\Rightarrow \cup$

Cheatsheet SQL:

SELECT [DISTINCT] [*, R, MAX(R) AS massimo, S ▷, COUNT(*), COALESCE(R, "zero")

FROM Tabella NATURAL JOIN Tabella2 ON .... JOIN Tabella3 ...
RIGHT OUTER JOIN

WHERE Cond θ Cond1 AND, OR NOT Cond3 EXISTS >= ALL (SELECT * ...
ANY
NOT EXISTS

LIKE, IN, BETWEEN, IS NULL, IS NOT NULL

[...] x AND y

GROUP BY ASC [DESC]
DEFAULT

HAVING COUNT MIN MAX SUM AVG

ORDER BY

UNION [ALL] → Mantiene i duplicati

SELECT ...

- - -

INTERSECT [ALL] ""

SELECT ...

EXCEPT [ALL] ""

SELECT ...

current_date

SELECT SUM (CASE classe
WHEN 'prima' THEN ...
WHEN '...' THEN miglia*1,5
ELSE miglia
END)

Vincolo su tabella:

CHECK (Cond θ (SELECT ...))

Creazione view:

CREATE VIEW Nomeview AS (
SELECT ...)
[WITH CHECK OPTION]

Vincolo su schema:

CREATE ASSERTION Nomevincolo
CHECK [Cond] θ (SELECT ...)

Trigger:

CREATE TRIGGER NomeTrigger
AFTER/BEFORE (dopo/prima) INSERT/DELETE/UPDATE [OF Colonna] ON tabella

FOR EACH ROW/STATEMENT
old, new
old_table, new_table

WHEN Cond1 θ Cond2

sql statement
BEGIN
IF() THEN
ENDIF
END

⇒ DECLARE X, Y INT, DATE DEFAULT 0, 1/1/2
⇒ INSERT INTO / UPDATE / DELETE FROM tabella
VALUES tabl SET WHERE WHERE
⇒ SET {-}

Insiemi di Dipendenze Minimali

Un insieme di dipendenze funzionali F è **minimale** se:

1. Ogni lato destro di una dipendenza è un singolo attributo.

2. Per ogni dipendenza $X \to A$ in F , $F \setminus \{X \to A\}$ non è equivalente a F

3. Per ogni $X \to A$ in F e $Z \subset X$, $F \setminus \{X \to A\} \cup \{Z \to A\}$ non è equivalente a F

La regola 2 garantisce che nessuna dipendenza in F è ridondante. La regola 3 garantisce che nessun attributo in qualunque primo membro sia ridondante.

Algoritmo per controllare se una decomposizione preserva i dati

Input:

$R=\{A_1,A_2,...,A_n\}$, F , $d=\{R_1,R_2,...R_k\}$

Output:

Yes/No se d preserva i dati.

INIZIALIZZAZIONE:

Consideriamo una matrice

$M = \{R_1,R_2,...R_k\}X\{A_1,A_2,...,A_n\}$

dove nell'elemento R_iA_j mettiamo a_j se A_j è in R_i

altrimenti mettiamo b_{ij}.

Passo Iterativo

- ITERAZIONE:

  Applichiamo finché è possibile ogni dipendenza $X \to Y$ in F nel seguente modo: se esistono due righe di M che coincidono su X allora facciamole coincidere anche in Y:

  - se ho uno dei due a_j allora cambiamo l'altro (b_{ij}) in a_j;
  - Altrimenti prendiamo uno dei due e lo facciamo uguale all'altro.

Test Finale

- Se durante il passo precedente si produce la riga $\mathbf{a_1a_2...a_n}$ allora rispondi *YES*
  (La decomposizione preserva i dati)

- Altrimenti rispondi *NO*
  (La decomposizione non preserva i dati)

FORME NORMALI

- Si può dimostrare che se tutte le dipendenze di F sono del tipo $X \rightarrow A$, allora basta verificare la suddetta proprietà solo per gli elementi di F e non di F^+ .

BCNF Forma Normale di Boyce-Codd

- Uno schema relazionale R con dipendenze F si dice in **Forma Normale** di **Boyce-Codd** (BCNF) se per ogni $X \rightarrow A$ di F^+ , se A non appartiene ad X allora
  - X è una *superchiave* di R, cioè è o contiene una chiave.
- Lo scopo delle BCNF e' **eliminare ridondanze causate dalle dipendenze**.

Algoritmo per BCNF:alcuni lemmi preliminari

Lemma 1

Sia R uno schema con dipendenze F e sia d={R_1,R_2,...R_k} una decomposizione che preserva i dati rispetto a F, e sia d' ={S_1,S_2}una decomposizione di R_1 che preserva i dati rispetto a $\pi_{R_1}(F)$.Allora la decomposizione di R, d'' ={S_1,S_2,R_2,...,R_k} preserva i dati rispetto a F.

Lemma 2

a) Ogni schema R con due attributi è in BCNF

b) Se R non è in BCNF allora esistono due attributi A,B tali che: $(R - AB) \rightarrow A$.

Lemma 3

Dati (R,F), se proiettiamo su $R_1 \subseteq R$ ottenendo F_1, e successivamente proiettiamo su $R_2 \subseteq R_1$ ottenendo F_2, allora si ha che $F_2 = \pi_{R_2}(F)$.

TERZA FORMA NORMALE

- Uno schema relazionale R con dipendenze F si dice in **Terza Forma Normale** (3NF) se per ogni $X \rightarrow A$ di F^+ , se A non appartiene ad X allora
  - X è una *superchiave* di R oppure A è *primo,* cioè appartiene a qualche chiave.

ci possono essere ridondanze dovute a dipendenze

Preservazione delle dipendenze e 3NF

- **Input**: R,F con F *ricoprimento minimale*
- **Output**: Una decomposizione di R che **conserva le dipendenze** e tale che ogni suo elemento e' in **3NF.**

ALGORITMO

- Se ci sono attributi *non presenti in F* essi possono essere raggruppati in un solo schema ed eliminati.
- Se una dipendenza di F coinvolge *tutti gli attributi* di R , allora ritorna (R).
- Altrimenti ritorna la decomposizione fatta da tutti gli XA tali che $X \rightarrow A$ appartiene ad F.

Preservare dati+dipendenze+3NF

sia X una *chiave* per R

- Allora (R_1,R_2,...R_k,X) preserva i dati e le dipendenze ed ogni suo elemento è in 3NF

- SI per ***conservazione dei dati e BCNF***
- SI per ***conservazione dei dati e delle dipendenze e 3NF.***
- **NO** per ***la conservazione delle dipendenze e BCNF***

Assiomi di Armstrong:

Riflessività: Se $Y \subseteq X \subseteq U$ allora $F \vdash X \to Y$

Aumento:

Se $F \vdash X \to Y$ allora $F \vdash XZ \to YZ$

Transitività:

Se $F \vdash X \to Y$ e $F \vdash Y \to Z$ allora $F \vdash X \to Z$

Lemmi Preliminari:

Decomposizione:

Se $F \vdash X \to Y$ e $Z \subseteq Y$ allora $F \vdash X \to Z$

Unione:

Se $F \vdash X \to Y$ e $F \vdash X \to Z$ allora $F \vdash X \to YZ$

Pseudotransitività:

Se $F \vdash X \to Y$ e $F \vdash WY \to Z$ allora $F \vdash WX \to Z$

NORMALIZZAZIONE

(RICOPRIMENTO MINIMALE)

Dipendenza Funzionale:

È un particolare vincolo di integrità che esprime legami funzionali tra gli attributi di una relazione.

MATRICOLA → NOME, TELEFONO
MATRICOLA, CORSO → VOTO

Chiusura di un insieme di dip. funz.:

Dato un insieme F di dip. funz., la sua chiusura F^+ è l'insieme delle dip. funz. che sono implicate logicamente da F.

Ad esempio, da $F = \{A \to B,\ B \to C\}$ $\quad F^+ = \{A \to B,\ B \to C,\ \underbrace{A \to C}_{\text{prop. trans.}}\}$

Chiave di uno Schema R(...) con F = {...}:

Sia $R(A, B, C, \ldots)$, F il suo insieme di dip. funz e $X \subseteq (A, B, C, D, \ldots)$: si dice che l'insieme di attributi X è una chiave di (R, F) se $X \to A, B, C, D, \ldots \in F^+$;

Ovvero se tramite l'insieme X e la chiusura F^+ è possibile derivare/ricavare tutti gli altri attributi di R;

NORMALIZZAZIONE (GENERICO)

Controllo di concorrenza

- *Obiettivo*: evitare le anomalie
- *Scheduler*: un sistema che accetta o rifiuta (o riordina) le operazioni richieste dalle transazioni
- *Schedule seriale:* le transazioni sono separate, una alla volta

$S_2 : r_0(x)\ r_0(y)\ w_0(x)\ r_1(y)\ r_1(x)\ w_1(y)\ r_2(x)\ r_2(y)\ r_2(z)\ w_2(z)$

- *Schedule serializzabile:* produce lo stesso risultato di uno schedule seriale sulle stesse transazioni
  - Richiede una nozione di equivalenza fra schedule

View-Serializzabilità

- Definizioni prelilminari:
  - $r_i(x)$ ***legge-da*** $w_j(x)$ in uno schedule S se $w_j(x)$ precede $r_i(x)$ in S e non c'è $w_k(x)$ fra $r_i(x)$ e $w_j(x)$ in S
  - $w_i(x)$ in uno schedule S è ***scrittura finale*** *se* è l'ultima scrittura dell'oggetto x in S
- Schedule ***view-equivalenti*** ($S_i \approx_V S_j$): hanno la stessa relazione *legge-da* e le stesse scritture finali
- Uno schedule è ***view-serializzabile*** se è view-equivalente ad un qualche schedule seriale
- L'insieme degli schedule view-serializzabili è indicato con **VSR**

View serializzabilità: esempi

- $S_3 : w_0(x)\ r_2(x)\ r_1(x)\ w_2(x)\ w_2(z)$
  $S_4 : w_0(x)\ r_1(x)\ r_2(x)\ w_2(x)\ w_2(z)$
  $S_5 : w_0(x)\ r_1(x)\ w_1(x)\ r_2(x)\ w_1(z)$
  $S_6 : w_0(x)\ r_1(x)\ w_1(x)\ w_1(z)\ r_2(x)$
  - S_3 è view-equivalente allo schedule seriale S_4 (e quindi è view-serializzabile)
  - S_5 non è view-equivalente a S_4, ma è view-equivalente allo schedule seriale S_6, e quindi è view-serializzabile
- $S_7 : r_1(x)\ r_2(x)\ w_1(x)\ w_2(x)$ (perdita di aggiornamento)
  $S_8 : r_1(x)\ r_2(x)\ w_2(x)\ r_1(x)$ (letture inconsistenti)
  $S_9 : r_1(x)\ r_1(y)\ r_2(z)\ r_2(y)\ w_2(y)\ w_2(z)\ r_1(z)$ (aggiornamento fantasma)
  - S_7, S_8, S_9 non view-serializzabili

Conflict-serializzabilità

- Definizione preliminare:
  - Un'azione a_i è in *conflitto* con a_j ($i \neq j$), se operano sullo stesso oggetto e almeno una di esse è una scrittura. Due casi:
    - conflitto *read-write* (*rw* o *wr*)
    - conflitto *write-write* (*ww*).
- *Schedule conflict-equivalenti* ($S_i \approx_C S_j$): includono le stesse operazioni e ogni coppia di operazioni in conflitto compare nello stesso ordine in entrambi
- Uno schedule è *conflict-serializable* se è conflict-equivalente ad un qualche schedule seriale
- L'insieme degli schedule conflict-serializzabili è indicato con **CSR**

CSR e VSR

- Ogni schedule conflict-serializzabile è view-serializzabile, ma non necessariamente viceversa
- Controesempio per la non necessità:

- r1(x) w2(x) w1(x) w3(x)

  - view-serializzabile: view-equivalente a r1(x) w1(x) w2(x) w3(x)
  - non conflict-serializzabile

Verifica di conflict-serializzabilità

- Per mezzo del **grafo dei conflitti**:
  - un nodo per ogni transazione 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
  - **Uno schedule è in CSR se e solo se il grafo è aciclico**

Lock

- Principio:
  - Tutte le letture sono precedute da *r_lock* (lock condiviso) e seguite da *unlock*
  - Tutte le scritture sono precedute da *w_lock* (lock esclusivo) e seguite da *unlock*
- Quando una transazione prima legge e poi scrive un oggetto, può:
  - richiedere subito un lock esclusivo
  - chiedere prima un lock condiviso e poi uno esclusivo (*lock escalation*)
- Il *lock manager* riceve queste richieste dalle transazioni e le accoglie o rifiuta, sulla base della tavola dei conflitti

Locking a due fasi (2PL)

- Usato da quasi tutti i sistemi
- Garantisce "a priori" la conflict-serializzabilità
- Basata su due regole:
  - "proteggere" tutte le letture e scritture con lock
  - un vincolo sulle richieste e i rilasci dei lock:
    - **una transazione, dopo aver rilasciato un lock, non può acquisirne altri**

2PL e CSR

- Ogni schedule 2PL e' anche conflict serializzabile, ma non necessariamente viceversa
- Controesempio per la non necessita':

$$r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ r_3(y)\ w_1(y)$$

  - Viola 2PL
  - Conflict-serializzabile

Locking a due fasi stretto

- Condizione aggiuntiva:
  - **I lock possono essere rilasciati solo dopo il commit o abort**
- Supera la necessità dell'ipotesi di commit-proiezione (ed elimina il rischio di letture sporche)

Stallo (deadlock)

- Attese incrociate: due transazioni detengono ciascuna una risorsa e aspettano la risorsa detenuta dall'altra
- Esempio:
  - t_1: *read*(x), *write*(y)
  - t_2: *read*(y), *write*(x)
  - Schedule:
    $r\_lock_1(x)$, $r\_lock_2(y)$, $read_1(x)$, $read_2(y)$ $w\_lock_1(y)$, $w\_lock_2(x)$

Risoluzione dello stallo

- Uno stallo corrisponde ad un ciclo nel grafo delle attese (nodo=transazione, arco=attesa)
- Tre tecniche
  1. Timeout (problema: scelta dell'intervallo, con trade-off)
  2. Rilevamento dello stallo
  3. Prevenzione dello stallo
- Rilevamento: ricerca di cicli nel grafo delle attese
- Prevenzione: uccisione di transazioni "sospette" (può esagerare)