

Progetto Basi di Dati 2023-24


“UNIGE SOCIAL SPORT”

Parte III

[GRUPPO 36]

 Virginia Passalacqua 5473700

 Flavio Venturini 5667103

 Denis Muceku 4801139

1. PROGETTAZIONE FISICA

1A+1C CARICO DI LAVORO

Q1 - QUERY CON SINGOLA SELEZIONE E NESSUN JOIN

LINGUAGGIO NATURALE

[Riportare in questa sezione l'interrogazione del carico di lavoro con singola selezione e nessun join, in linguaggio naturale]

- *Determinare nome e cognome di tutti gli utenti della piattaforma che frequentano il corso di 'Informatica'*

SQL

[Riportare in questa sezione l'interrogazione del carico di lavoro con singola selezione e nessun join, in SQL]

```
SELECT nome || ' ' || cognome AS nominativo
FROM cl_utenti
WHERE corso='Informatica';
```

Q2 - QUERY CON CONDIZIONE DI SELEZIONE COMPLESSA E NESSUN JOIN

LINGUAGGIO NATURALE

[Riportare in questa sezione l'interrogazione del carico di lavoro con condizione di selezione complessa e nessun join, in linguaggio naturale]

- *Determinare nome e cognome di tutti gli studenti della piattaforma che frequentano il corso di informatica o economia e sono nati tra il 1970 e il 1990*

SQL

[Riportare in questa sezione l'interrogazione del carico di lavoro con condizione di selezione complessa e nessun join, in SQL]

```
SELECT nome || ' ' || cognome
FROM cl_utenti
WHERE (corso='Informatica' OR corso='Economia') AND anno_nascita BETWEEN 1970 AND 1990;
```

Q3 - QUERY CON ALMENO UN JOIN E ALMENO UNA CONDIZIONE DI SELEZIONE

LINGUAGGIO NATURALE

[Riportare in questa sezione l'interrogazione del carico di lavoro con almeno un join e almeno una condizione di selezione, in linguaggio naturale]

- *Determinare tutti gli utenti della piattaforma che frequentano il corso di Ingegneria e gli eventi in cui hanno fatto richiesta di iscrizione in qualità di 'ARBITRO'*

SQL

[Riportare in questa sezione l'interrogazione del carico di lavoro con almeno un join e almeno una condizione di selezione, in SQL]

```

SELECT nome || ' ' || cognome, id_evento
FROM cl_utenti U JOIN cl_iscrizioni I ON U.matricola=I.matricola
WHERE corso='Ingegneria' AND qualità='ARBITRO'

```

1D-PROGETTO FISICO

[Riportare nella seguente tabella l'elenco degli indici che si intendono creare per: (1) ciascuna query del carico di lavoro individualmente; (2) l'insieme delle query del carico di lavoro, motivando opportunamente, in modo sintetico, le scelte effettuate]

<i>Id query</i>	<i>Relazione</i>	<i>Chiave di ricerca</i>	<i>Tipo (ordinato/hash, clusterizzato/non clusterizzato)</i>	<i>Motivazione</i>
1	cl_utenti	(l _{cl_utenti} (corso), corso = 'Informatica')	hash, secondario e non clusterizzato	corso = 'Informatica' è un fattore booleano in quanto è l'unico presente nella condizione di selezione, la scelta del tipo hash è dettata dal fatto che abbiamo 8 diversi corsi che quindi presumibilmente verranno salvati in bucket senza generare overflow (dato che la selezione è di uguaglianza è consentito l'hash). Avendo quindi una stima del risultato di selezione di 0,125 (1/8) essendo molto vicina a 0 conviene creare l'indice (preferibile alla scansione sequenziale). Non lo clusterizziamo in quanto non possiamo raggruppare il file in base all'indice e in generale non porta a molti benefici per il carico di lavoro Q1 lasciando spazio eventualmente ad un indice clusterizzato diverso

2	<i>cl_utenti</i>	<i>l_{cl_utenti} (anno_nascita), 1970 < anno_nascita < 1990</i>	<i>Ordinato, secondario e clusterizzato</i>	<p>1970 < anno_nascita < 1990 è un fattore booleano perché ‘governa’ la condizione di selezione (se falsa rende falsa tutta l’espressione), la scelta del tipo ordinato è obbligatoria perché gli indice hash non possono lavorare su intervalli. Avendo una stima del risultato di 0,233 (20/86 → (F(1970 < anno_nascita < 1970) = (Max(anno_nascita,R) - 1970)/(Max(anno_nascita,R) - Min(anno_nascita,R) = (1990-1970)/(2006-1920)) essendo più vicino a 0 conviene creare l’indice (preferibile alla scansione sequenziale). Lo clusterizziamo in quanto è secondario e se consideriamo solo Q2 è l’unico che possiamo creare e che verrà preso in considerazione per tale carico di lavoro (valido anche per Q1+Q2)</p>
3	<i>cl_utenti, cl_iscrizioni</i>	<ol style="list-style-type: none"> 1. <i>(l_{cl_utenti} (corso), corso='Ingegneria')</i> 2. <i>(l_{cl_iscrizioni} (matricola))</i> 	<ol style="list-style-type: none"> 1. <i>Hash, secondario e non clusterizzato</i> 2. <i>Ordinato, secondario e clusterizzato</i> 	<p>Il costo più grande è quello del join quindi dobbiamo basarci su quello, la tecnica utilizzata dalla creazione di questi indici è quella dell’index nested loop (in quanto abbiamo un indice sulla condizione di join di uguaglianza nella relazione inner). Evitiamo di adottare la strategia merge join perché ci obbliga a creare un indice primario su matricola nella relazione cl_utenti che però non viene utilizzata. Siamo anche obbligati a scegliere un indice sulla relazione outer per la condizione di selettività in quanto con la realizzazione del index nested loop gli indici relativi alla relazione inner non vengono</p>

				<p>considerati (se non quello per eseguire il tipo di join), perciò usiamo l'indice su corso con lo stesso funzionamento nella Q1. In questo modo applichiamo la condizione di selezione alla relazione outer e poi filtriamo grazie all'indice sulla relazione inner per eseguire il join. Solo l'indice su matricola è clusterizzato poiché secondario e efficiente quando si usa l'index nested loop mentre per l'indice su corso vale lo stesso discorso della Q1.</p>
--	--	--	--	--

Schema fisico complessivo per il carico di lavoro	Motivazione
<ul style="list-style-type: none"> • <i>l_{cl_utenti}</i> (corso) -- hash, non clusterizzato • <i>l_{cl_utenti}</i> (anno_nascita) -- ordinato, clusterizzato • <i>l_{cl_iscrizioni}</i> (matricola) – ordinato, clusterizzato 	<p>La fusione dei 3 non porta ad avere conflitti in quanto c'è al più un indice clusterizzato per tabella. Volendo si potrebbe aggiungere un indice primario su matricola di <i>cl_utenti</i> per implementare il merge join (però non utilizzando più l'indice su corso nella Q3)</p>

1G-ANALISI PIANI DI ESECUZIONE SCELTI DAL SISTEMA

Q1 - QUERY CON SINGOLA SELEZIONE E NESSUN JOIN

PIANO DI ESECUZIONE SCELTO DAL SISTEMA PRIMA DELLA CREAZIONE DELLO SCHEMA FISICO

[Riportare in questa sezione il piano di esecuzione scelto da PostgreSQL prima della creazione dello schema fisico, in formato testuale]

Seq Scan on cl_utenti (cost=0.00..17.44 rows=38 width=32)

Filter: ((corso)::text = 'Informatica'::text)

PIANO DI ESECUZIONE SCELTO DAL SISTEMA DOPO DELLA CREAZIONE DELLO SCHEMA FISICO

[Riportare in questa sezione il piano di esecuzione scelto da PostgreSQL dopo la creazione dello schema fisico, in formato testuale]

Bitmap Heap Scan on cl_utenti (cost=4.44..16.11 rows=38 width=32)

Recheck Cond: ((corso)::text = 'Informatica'::text)

-> Bitmap Index Scan on idx_corso (cost=0.00..4.43 rows=38 width=0)

Index Cond: ((corso)::text = 'Informatica'::text)

CONFRONTO TRA I DUE PIANI

[Riportare nella seguente tabella i tempi di esecuzione per i piani ottenuti prima e dopo la creazione dello schema fisico, giustificando i piani e i tempi ottenuti]

Tempo esecuzione, PRIMA		Tempo esecuzione DOPO	Motivazione
Planning time	0.092 ms	0.118 ms	<ul style="list-style-type: none">- Il Planning time è più alto con la creazione degli indici perché giustamente il sistema avrà più piani da considerare e verificare- L'Execution time invece è a beneficio post indice in quanto esso permette di raggiungere le tuple in modo più efficiente <p>L'aumento della prestazione è circa del 26,5%</p>
Execution time	0.229 ms	0.108 ms	
TOT	0.321 ms	0.236 ms	

Q2 - QUERY CON CONDIZIONE DI SELEZIONE COMPLESSA E NESSUN JOIN

PIANO DI ESECUZIONE SCELTO DAL SISTEMA PRIMA DELLA CREAZIONE DELLO SCHEMA FISICO

[Riportare in questa sezione il piano di esecuzione scelto da PostgreSQL prima della creazione dello schema fisico, in formato testuale]

Seq Scan on cl_utenti (cost=0.00..19.93 rows=36 width=32)

Filter: ((anno_nascita > 1970) AND (anno_nascita < 1990) AND (((corso)::text = 'Informatica'::text) OR (corso)::text = 'Economia'::text)))

PIANO DI ESECUZIONE SCELTO DAL SISTEMA DOPO DELLA CREAZIONE DELLO SCHEMA FISICO

[Riportare in questa sezione il piano di esecuzione scelto da PostgreSQL dopo la creazione dello schema fisico, in formato testuale]

Bitmap Heap Scan on cl_utenti (cost=5.36..18.88 rows=20 width=32)

Recheck Cond: ((anno_nascita >= 1970) AND (anno_nascita <= 1990))

Filter: (((corso)::text = 'Informatica'::text) OR ((corso)::text = 'Economia'::text))

-> Bitmap Index Scan on idx_anno_n (cost=0.00..5.36 rows=121 width=0)

Index Cond: ((anno_nascita >= 1970) AND (anno_nascita <= 1990))

CONFRONTO TRA I DUE PIANI

[Riportare nella seguente tabella i tempi di esecuzione per i piani ottenuti prima e dopo la creazione dello schema fisico, giustificando i piani e i tempi ottenuti]

Tempo esecuzione PRIMA		Tempo esecuzione DOPO	Motivazione
Planning time	0.112 ms	0.201 ms	<ul style="list-style-type: none">- Il Planning time è più alto con la creazione degli indici perché giustamente il sistema avrà più piani da considerare e verificare- L'Execution time invece è a beneficio post indice in quanto esso permette di raggiungere le tuple in modo più efficiente
Execution time	0.233 ms	0.124 ms	
TOT	0.345 ms	0.325 ms	
			L'aumento della prestazione è circa del 5,8% quindi relativamente basso, ciò è dovuto al fatto che rispetto alla Q1 abbiamo un indice con condizione di selettività abbastanza alta ma molto inferiore rispetto a quello della Q1

Q3 - QUERY CON ALMENO UN JOIN E ALMENO UNA CONDIZIONE DI SELEZIONE

PIANO DI ESECUZIONE SCELTO DAL SISTEMA PRIMA DELLA CREAZIONE DELLO SCHEMA FISICO

[Riportare in questa sezione il piano di esecuzione scelto da PostgreSQL prima della creazione dello schema fisico, in formato testuale]

Hash Join (cost=17.89..36.14 rows=25 width=36)

Hash Cond: (i.matricola = u.matricola)

-> Seq Scan on cl_iscrizioni i (cost=0.00..16.88 rows=268 width=8)

Filter: ((""qualità"")::text = 'ARBITRO'::text)

-> Hash (cost=17.25..17.25 rows=51 width=18)

-> Seq Scan on cl_utenti u (cost=0.00..17.25 rows=51 width=18)

Filter: ((corso)::text = 'Ingegneria'::text)

PIANO DI ESECUZIONE SCELTO DAL SISTEMA DOPO DELLA CREAZIONE DELLO SCHEMA FISICO

[Riportare in questa sezione il piano di esecuzione scelto da PostgreSQL dopo la creazione dello schema fisico, in formato testuale]

Hash Join (cost=15.67..28.93 rows=25 width=36)

Hash Cond: (i.matricola = u.matricola)

-> Seq Scan on cl_iscrizioni i (cost=0.00..11.88 rows=268 width=8)

Filter: ((""qualità"")::text = 'ARBITRO'::text)

-> Hash (cost=15.03..15.03 rows=51 width=18)

-> Bitmap Heap Scan on cl_utenti u (cost=4.40..15.03 rows=51 width=18)

Recheck Cond: ((corso)::text = 'Ingegneria'::text)

-> Bitmap Index Scan on idx_corso (cost=0.00..4.38 rows=51 width=0)

Index Cond: ((corso)::text = 'Ingegneria'::text)

CONFRONTO TRA I DUE PIANI

[Riportare nella seguente tabella i tempi di esecuzione per i piani ottenuti prima e dopo la creazione dello schema fisico, giustificando i piani e i tempi ottenuti]

Tempo esecuzione PRIMA		Tempo esecuzione DOPO	Motivazione
Planning time	0.189 ms	0.288 ms	- Il Planning time è più alto con la creazione degli indici perché giustamente il sistema avrà più piani da considerare e verificare

<i>Execution time</i>	<i>0.468 ms</i>	<i>0.381 ms</i>	<p>- <i>L'Execution time invece è a beneficio post indice in quanto esso permette di raggiungere le tuple in modo più efficiente</i></p> <p><i>L'aumento della prestazione è circa del -1,8% quindi addirittura peggio che senza indici, questo perché nella parte più costosa ovvero il join il sistema in entrambi i casi considera l'approccio hash join (uno dei più efficienti) senza considerare il nostro indice creato apposta per l'approccio index nested loop, l'esecuzione è migliore perché abbiamo l'indice su corso che ottimizza la ricerca ma aggiungendo due nuovi indici si aumenta il tempo di planning e l'ottimizzazione della ricerca non riesce coprire questo costo (ne tanto meno migliorarlo)</i></p>
<i>TOT</i>	<i>0.657 ms</i>	<i>0.669 ms</i>	

2. CONTROLLO DELL'ACCESSO

GERARCHIA TRA I RUOLI

GERARCHIA

[Riportare in questa sezione la gerarchia che si intende definire tra i quattro ruoli]

- ❖ $ADMIN \geq GESTORE\ IMPIANTI, UTENTI\ PREMIUM$
- ❖ $UTENTI\ PREMIUM \geq UTENTI\ SEMPLICI$

MOTIVAZIONE GERARCHIA

[Riportare in questa sezione una motivazione per la gerarchia proposta]

- Sicuramente l'Amministratore ha pieni poteri su tutto il database per mantenerlo ed aggiornarlo.
- Il gestore ha poteri sulla tabella impianti ma potrebbe non avere privilegi su tabelle come iscrizioni che invece hanno gli utenti premium e semplici perciò non ha gerarchie sottostanti
- L'utente premium ha dei privilegi in più rispetto a quello semplice come creare eventi, squadre, approvare iscrizioni ecc... Ma non ha dei privilegi che ha il gestore degli impianti

ASSEGNAZIONE PRIVILEGI SPECIFICI AI RUOLI

[Riportare nella prima colonna della seguente tabella le relazioni considerate e in ciascuna altra cella (i,j) i privilegi specifici (quindi non acquisiti tramite la gerarchia) che si intendono assegnare al ruolo j sulla tabella i].

RELAZIONE	Amministratore di UniGE social sport	Utente premium	Gestore impianto	Utente semplice
CATEGORIE	ALL	SELECT	SELECT	SELECT
UTENTI	ALL	SELECT	✗	✗
UTENTI_PREMIUM	ALL	SELECT	✗	✗
IMPIANTI	ALL	SELECT	ALL	SELECT
EVENTI	ALL	ALL	SELECT	SELECT
ISCRIZIONI	ALL	SELECT, UPDATE, INSERT	✗	SELECT, INSERT
SQUADRE	ALL	ALL	SELECT	SELECT
CANDIDATURE	ALL	SELECT, UPDATE, INSERT	✗	SELECT, INSERT

TORNEI	ALL	ALL	SELECT	SELECT
EVENTO_A_SQUADRE	ALL	ALL	SELECT	SELECT
ESITI	ALL	ALL	✗	SELECT
VALUTAZIONI	ALL	ALL	✗	ALL
STATISTICHE	ALL	ALL	✗	SELECT

* eventuali controlli come in valutazione che un utente può esprimere una valutazione solo se indica la propria matricola sono gestite da funzioni/trigger

FUNZIONI	Amministratore di UniGE social sport	Utente premium	Gestore impianto	Utente semplice
appr_iscriz()	✓	✓	✗	✗
Disiscriz()	✓	✓	✗	✓
Appr_cand()	✓	✓	✗	✗
level()	✓	✓	✗	✓
Delete_old_events()	✓	✗	✗	✗

VISTE	Amministratore di UniGE social sport	Utente premium	Gestore impianto	Utente semplice
Programma	ALL	✗	SELECT	✗