UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea in Informatica

Un giorno al Museo

Basi di Dati
05/06/2021 - anno accademico 2020/2021

Prof. Giorgio Ghelli

Salvatore Correnti
584136 - Corso A
Domenico Iiripino
599248 - Corso A
Giacomo Trapani
600124 - Corso A

Esercizio 1

Scopo di questo testo è descrivere la specifica di **Un giorno al museo**, un software di gestione del sistema museale della regione Toscana.

Ogni **museo** è composto da diversi **ambienti** collegati a coppie tramite **varchi**, distinti in **sale** e **ambienti di servizio**.

Ogni **sala** può essere o una **sala museale** o una **sala per mostre temporanee** o una **sala esclusiva** per l'esposizione di opere esclusive.

Per ogni museo è possibile acquistare un **abbonamento generale**, che permette di visitare in totale un certo numero di sale museali e per mostre temporanee, oppure un **abbonamento speciale** che consente di visitare anche le sale esclusive.

Di ogni **museo** interessano il *nome*, gli *ambienti che lo compongono*, *come sono collegati* fra di loro e gli *utenti che hanno effettuato almeno una visita* in quel museo.

Di ogni **ambiente** interessano l'*identificatore*, il *nome* e i *varchi* con cui è collegato ad altri ambienti.

Di ogni **sala** interessa inoltre il *tipo* (sala museale, sala per mostre o sala esclusiva) e gli *utenti* che l'hanno visitata.

Di ogni **ambiente di servizio** interessa inoltre il *tipo* (bar, biglietteria, ristorante, bagno).

Di ogni **varco** interessano l'*identificatore*, la *coppia di ambienti* che collega e il *tipo* di varco (scala, porta, apertura).

Di ogni **utente** interessano *nome*, *cognome*, *indirizzo fisico*, *indirizzo email*, *visite effettuate*, *riviste ricevute* e *abbonamenti acquistati*.

Di ogni **biglietto** interessano il *codice*, il *tipo* (tariffa bianca o tariffa verde), e le *sale visitate* con quel biglietto.

Di ogni **visita** (di sala) interessano la *sala visitata*, l'*utente visitatore*, se la visita al museo di cui fa parte è stata *pagata con* un abbonamento o con un biglietto e l'*ora di inizio e di fine*.

Di ogni **abbonamento** interessano il *numero di sale che permette di visitare*, il *tempo in minuti* che permette per le visite, gli *utenti che lo hanno acquistato*, le *visite effettuate* con quell'abbonamento, se l'abbonamento è *ancora valido* e se è *speciale*.

Di ogni **rivista** interessano il *nome* ("Il museo scientifico" o "Il museo d'arte"), gli *utenti iscritti* e le *spedizioni effettuate*.

Di ogni **spedizione** interessano la *rivista spedita* e l'*utente* che l'ha ricevuta.

Esercizio 2

Vincoli non catturati dallo schema

Vincoli interrelazionali

$(\forall v \in Visite.\ (v.Abbonamento \neq NULL \oplus v.Biglietto \neq NULL))$;
$(\forall v \in Visite\ con\ v.Abbonamento \neq NULL.\ (v.Abbonamento \in v.Utente.Abbonamenti) \wedge (v.Abbonamento.valido = 1))$;
$(\forall v1, v2 \in Visite\ con\ v1.Biglietto,\ v2.Biglietto \neq NULL.\ (v1.Biglietto = v2.Biglietto) \Rightarrow (v1.Sala.Museo = v2.Sala.Museo))$;
$(\forall v \in Visite.\ (v.Abbonamento \neq NULL) \Rightarrow ((v.Sala.tipo =' esclusiva') \Leftrightarrow (v.Abbonamento.Speciale = 1)))$;
$(\forall s \in Spedizioni.\ s.Rivista \in s.Utente.Riviste)$.

Vincoli intrarelazionali

$(\forall v \in Varchi.\ v.Ambiente1 \neq v.Ambiente2)$;
$(\forall a \in Abbonamenti.\ (a.minuti \geq 0) \wedge (a.numeroSale \geq 0))$;
$(\forall a1 \in Ambienti.\ \neg(\exists a2 \in Ambienti.\ (a1 \neq a2) \wedge (a1.codiceAmbiente = a2.codiceAmbiente))$;
$(\forall v1 \in Varchi.\ \neg(\exists v2 \in Varchi.\ (v1 \neq v2) \wedge (v1.codiceVarco = v2.codiceVarco))$;
$(\forall a1 \in Abbonamenti.\ \neg(\exists a2 \in Abbonamenti.\ (a1 \neq a2) \wedge (a1.codiceAbbonamento = a2.codiceAbbonamento))$;
$(\forall u1 \in Utenti.\ \neg(\exists u2 \in Utenti.\ (u1 \neq u2) \wedge (u1.email = u2.email))$;
$(\forall c\ campo\ dello\ schema.\ c \neq NULL)$.

Esercizio 3

Si pone che oraInizio, oraFine sono dei timestamp, per cui identificano un momento preciso nel tempo.

Dipendenze funzionali

Il simbolo (*) a destra di una dipendenza funzionale indica tutti gli attributi della relazione corrispondente.

e.g. data R(IdR,A,B) vale che (IdR $\rightarrow$ $*$) $\Leftrightarrow$ (IdR $\rightarrow$ IdR, A, B).

Si ricorda che non vale IdAmbiente1, IdAmbiente2 $\rightarrow$ *, in quanto nel testo non è specificato che una coppia di ambienti sia sempre collegata da uno e un solo varco.

Si ha che la relazione Visite *non* è in forma normale di Boyce-Codd in quanto IdBiglietto,IdAbbonamento$\rightarrow$IdUtente. Tutte le altre relazioni sono invece in **BCNF** in quanto tutte le dipendenze funzionali individuate sono scomponibili nella forma (X$\rightarrow$A) con A attributo e X superchiave.

Si riportano di seguito le dipendenze funzionali; da queste risulta possibile ricavare tutte le altre dipendenze non banali.

Musei

$IdMuseo \rightarrow *$.

Ambienti

$codiceAmbiente \rightarrow *$;
$IdAmbiente \rightarrow *$.

Varchi

$codiceVarco \rightarrow *$;
$IdVarco \rightarrow *$.

AmbientiDiServizio

$IdAmbiente \rightarrow *$.

Sale

$IdAmbiente \rightarrow *$.

Visite

$IdBiglietto,\ IdAbbonamento,\ oraInizio,\ oraFine \rightarrow *;$
$IdBiglietto,\ IdAbbonamento,\ oraInizio \rightarrow *;$
$IdBiglietto,\ IdAbbonamento,\ oraFine \rightarrow *;$
$IdBiglietto,\ IdAbbonamento \rightarrow IdUtente;$
$IdUtente,\ oraInizio,\ oraFine \rightarrow *;$
$IdUtente,\ oraInizio \rightarrow *;$
$IdUtente,\ oraFine \rightarrow *;$
$IdVisita \rightarrow *.$

Biglietti

$codiceBiglietto \rightarrow *;$
$IdBiglietto \rightarrow *.$

Abbonamenti

$codiceAbbonamento \rightarrow *;$
$IdAbbonamento \rightarrow *.$

Sottoscrizioni

$IdAbbonamento,\ IdUtente \rightarrow *$

Utenti

.

$IdUtente \rightarrow *;$
$email \rightarrow *.$

Iscrizioni

$IdUtente,\ IdRivista \rightarrow *.$

Spedizioni

$IdUtente,\ IdRivista \rightarrow *.$

Riviste

$IdRivista \rightarrow *.$

Esercizio 4

Esercizio 4.a

La query restituisce Id, nome e cognome di tutti e soli gli utenti a cui è stata spedita una rivista dopo il 2 Apr. 2021.

```
SELECT DISTINCT
    u.IdUtente, u.nome, u.cognome
FROM
    Utenti u
    JOIN Spedizioni s USING (IdUtente)
WHERE s.data > '2021-04-02'
```

Esercizio 4.b

La query restituisce Id, Nome, Cognome e numero di sale museali visitate di tutti e soli gli utenti che hanno visitato almeno 3 sale museali ordinandoli sul numero di sale museali visitate (in ordine crescente).

```
SELECT
    u.IdUtente, MAX(u.nome) as NomeUtente,
    MAX(u.cognome) as CognomeUtente,
    COUNT(*) AS nVisite
FROM
    Utenti u
    JOIN Visite v USING (IdUtente)
    JOIN Sale s USING (IdAmbiente)
WHERE s.tipo = 'museali'
GROUP BY u.IdUtente
HAVING COUNT(*) >= 3
ORDER BY nVisite
```

Esercizio 4.c

La query restituisce Id, Nome, Cognome e numero di abbonamenti sottoscritti di tutti e soli gli utenti che hanno sottoscritto almeno 2 abbonamenti speciali.

```
SELECT
    u.IdUtente, MAX(u.nome) AS NomeUtente, MAX(u.cognome)
    AS CognomeUtente, COUNT(*) as nSottoscrizioni
FROM
    Utenti u
    JOIN Sottoscrizioni s USING (IdUtente)
    JOIN Abbonamenti a USING (IdAbbonamento)
WHERE (a.speciale = 1)
GROUP BY u.IdUtente
HAVING COUNT(*) >= 2
```

Esercizio 4.d

La query restitituisce tutti e soli gli utenti che hanno effettuato almeno una visita in una sala per mostre.

```
SELECT
    u.IdUtente, u.nome, u.cognome
FROM Utenti u
WHERE
    EXISTS
    (
        SELECT *
        FROM
            Visite v
            JOIN Sale s USING (IdAmbiente)
        WHERE
            (v.IdUtente = u.IdUtente) AND (s.tipo = 'per␣mostre')
    )
```

Esercizio 4.e

La query restituisce tutti e soli i musei per cui tutte le sale sono museali o per mostre.

```
SELECT
    m.IdMuseo, m.nome
FROM Musei m
WHERE
    NOT EXISTS
    (
        SELECT *
        FROM
            Sale s
            JOIN (Ambienti a) USING (IdAmbiente)
        WHERE
            (a.IdMuseo = m.IdMuseo) AND (s.tipo = 'esclusiva')
    )
```

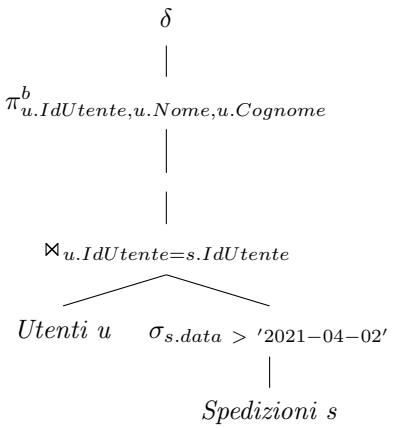
Esercizio 4.f

La query restituisce ID e nome degli di tutti e soli gli utenti che hanno visitato almeno 100 sale.

```
SELECT
    u.IdUtente, u.nome
FROM Utenti u
WHERE
    100 <=
    (
        SELECT COUNT(*)
        FROM Visite v
        WHERE (v.IdUtente = u.IdUtente)
    )
```

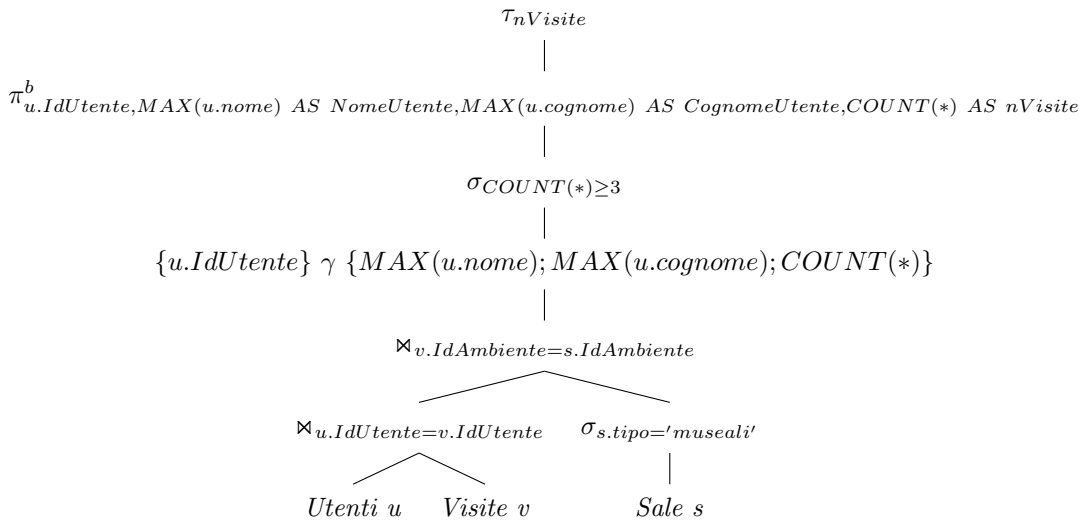
Esercizio 5

Esercizio 5.1.a



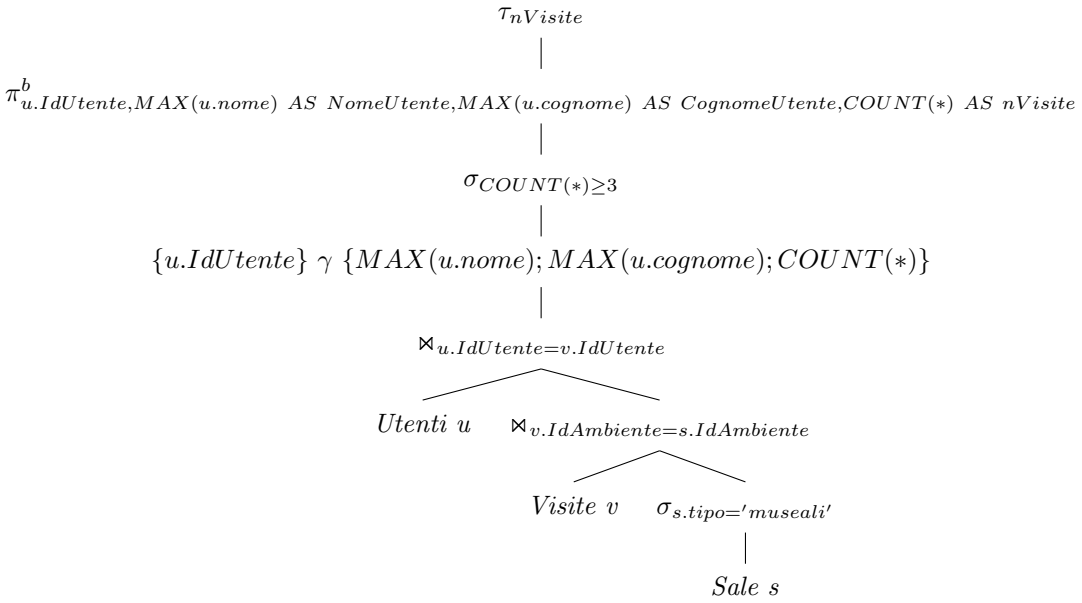
Esercizio 5.1.b

Versione 1



Il calcolo degli attributi (*NomeUtente*, *CognomeUtente*) come (MAX(u.nome), MAX(u.cognome)) - che risulta possibile in quanto il raggruppamento avviene sulla chiave primaria *IdUtente* - permette di avere un piano di accesso fisico leggermente più efficiente.

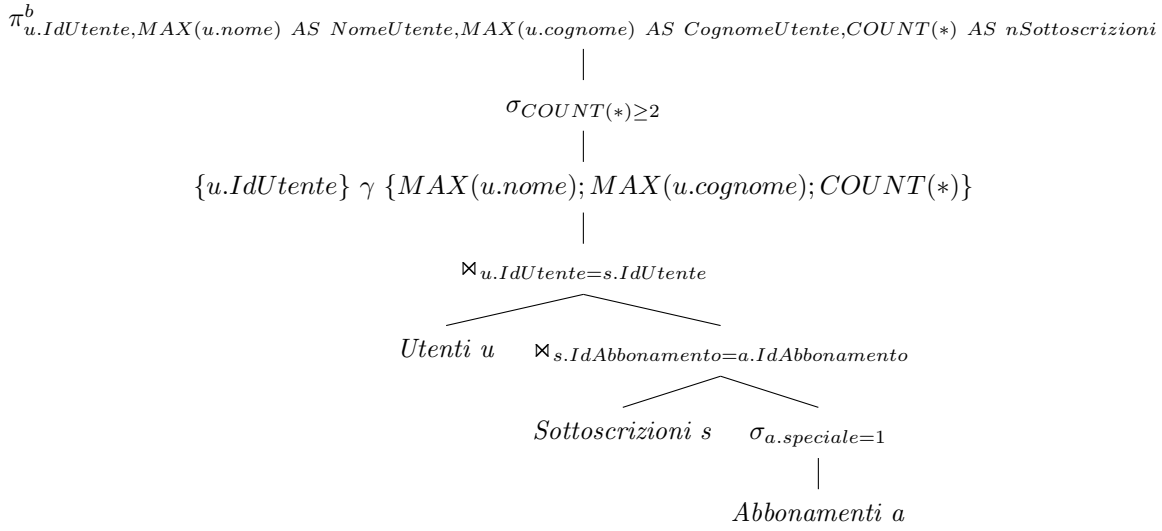
Versione 2



Questo piano logico può essere tradotto in un piano fisico in cui la Sort prima della GroupBy può essere eliminata.

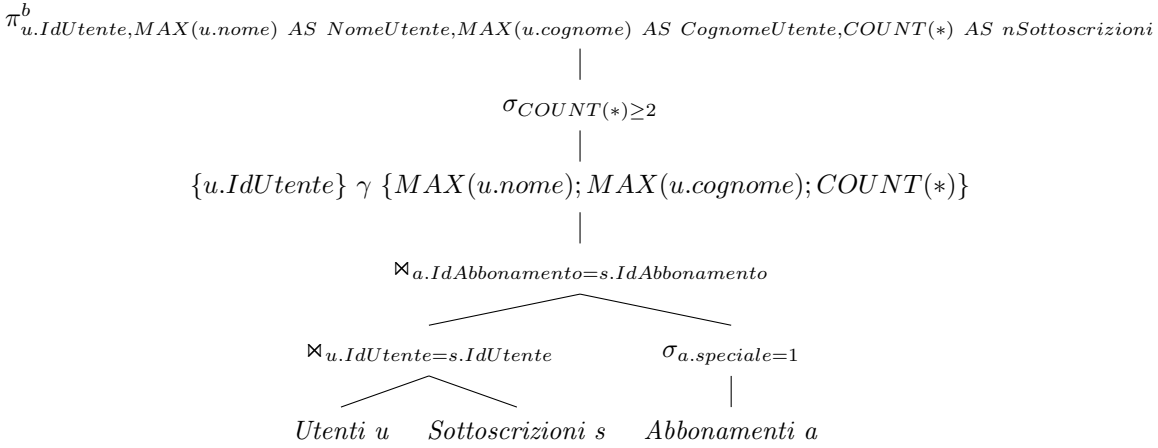
Esercizio 5.1.c

Versione 1



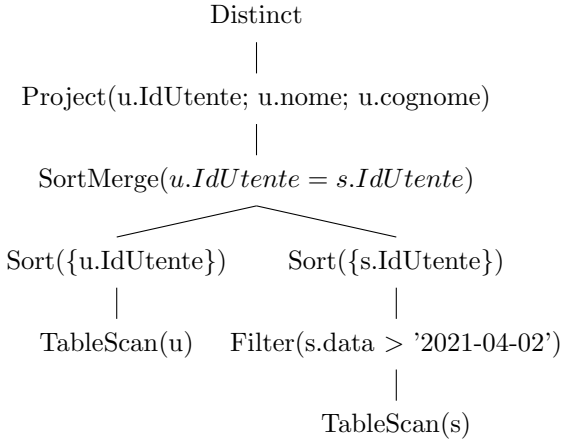
Anche in questo caso risulta possibile scrivere un piano di accesso fisico in cui la Sort prima della GroupBy viene eliminata.

Versione 2


$\pi^{b}_{u.IdUtente,MAX(u.nome)\ AS\ NomeUtente,MAX(u.cognome)\ AS\ CognomeUtente,COUNT(*)\ AS\ nSottoscrizioni}$
$\sigma_{COUNT(*)\geq 2}$
$\{u.IdUtente\}\ \gamma\ \{MAX(u.nome); MAX(u.cognome); COUNT(*)\}$
$\bowtie_{a.IdAbbonamento=s.IdAbbonamento}$
$\bowtie_{u.IdUtente=s.IdUtente}$
$\sigma_{a.speciale=1}$
Utenti u
Sottoscrizioni s
Abbonamenti a

Esercizio 5.2.a

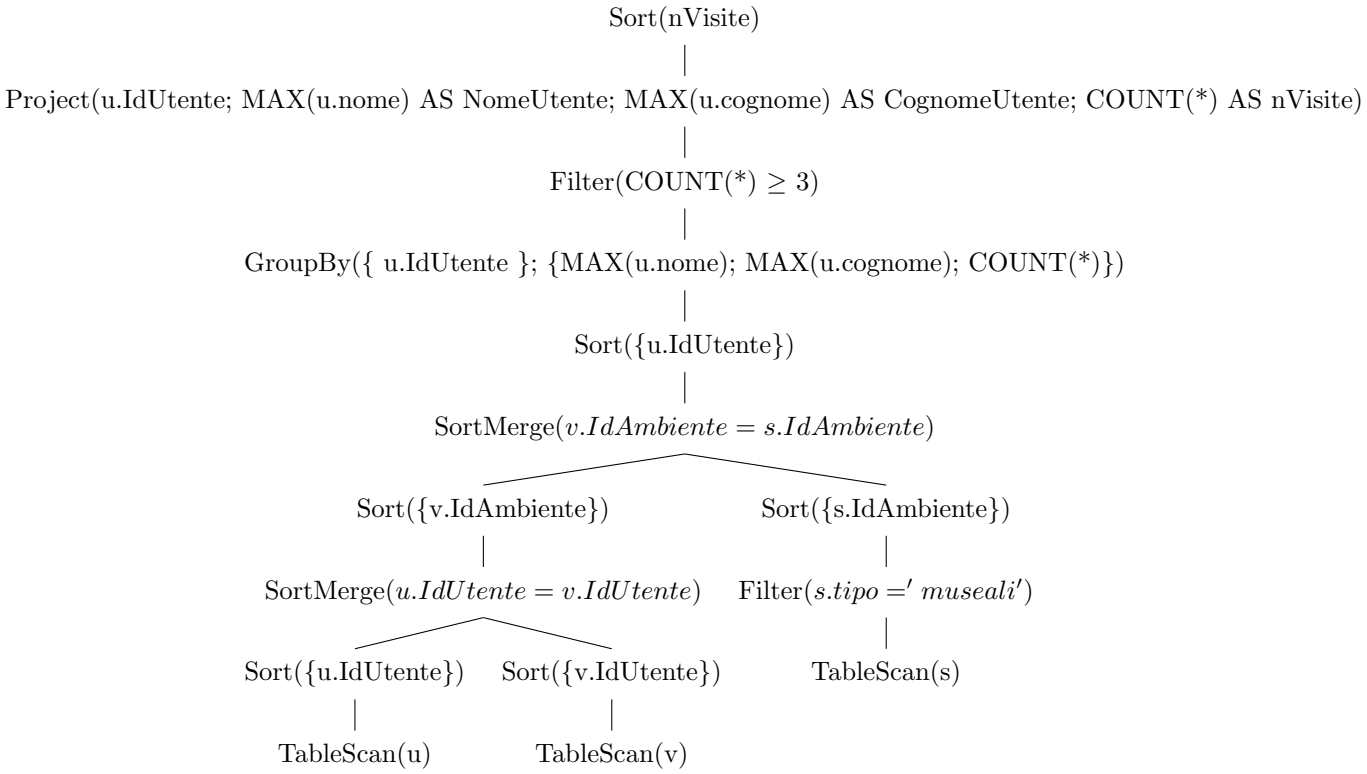
Previa ridenominazione per semplificare la scrittura: Utenti → u, Spedizioni → s.



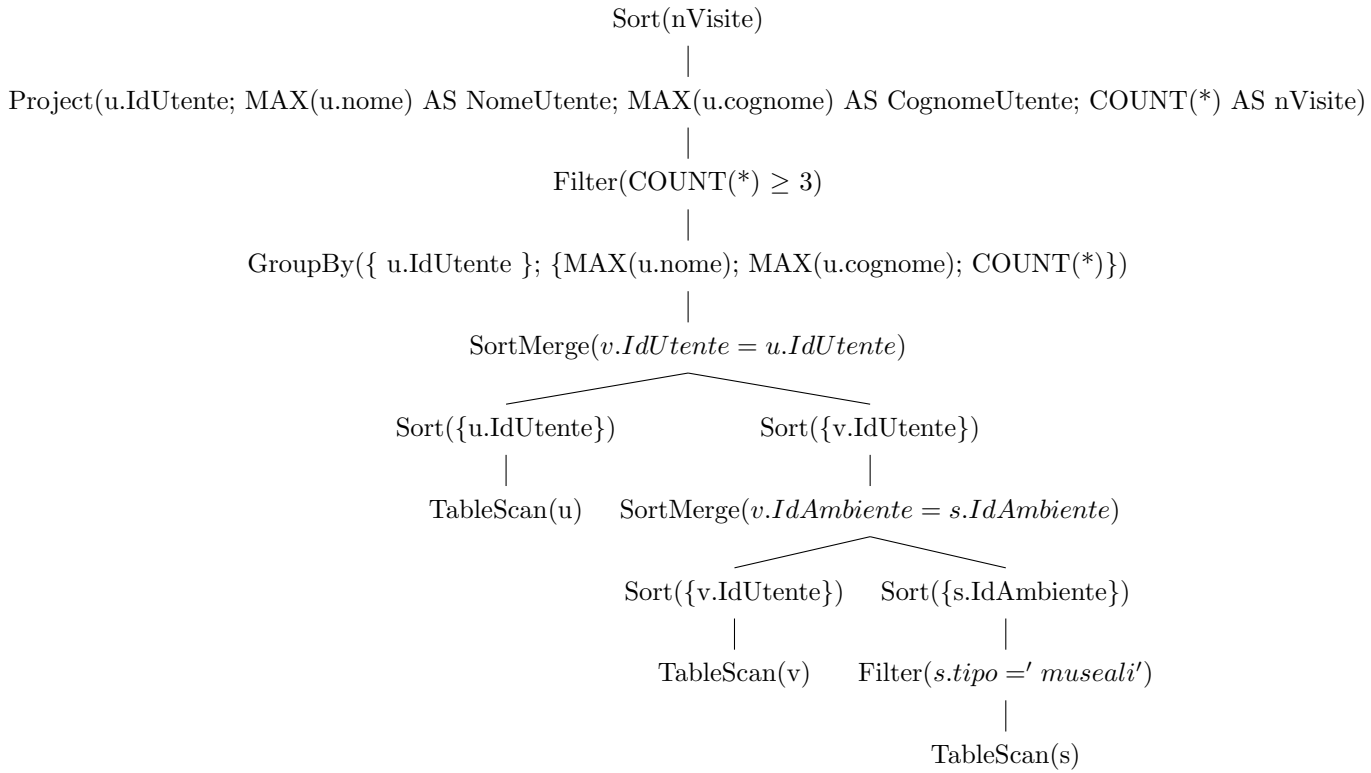
Esercizio 5.2.b

Previa ridenominazione per semplificare la scrittura: Visite → v, Utenti → u, Sale → s.

Versione 1



Versione 2

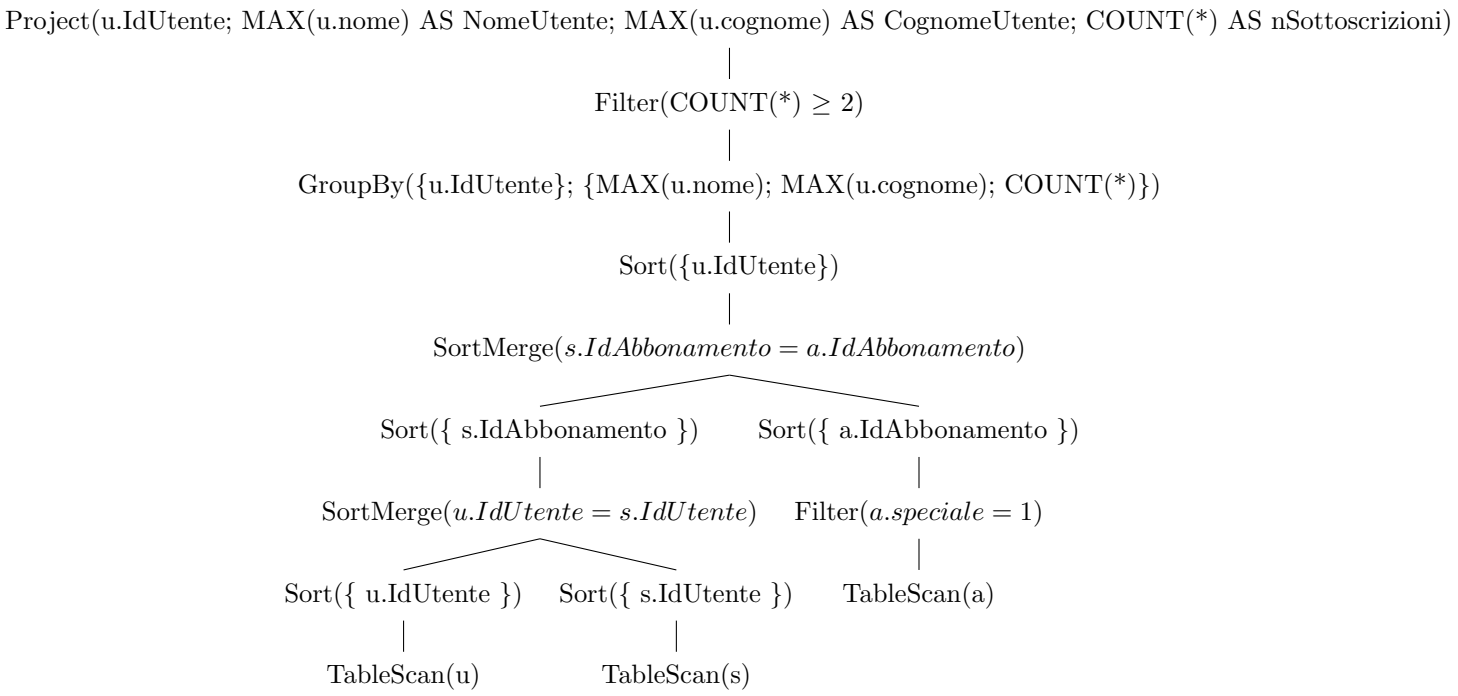


Questa versione contiene 14 operatori al posto dei 15 dell'altra; si è potuto eliminare la Sort prima della GroupBy in quanto l'output di SortMerge al quarto livello dell'albero è già ordinato su u.IdUtente per definizione (dell'operatore).

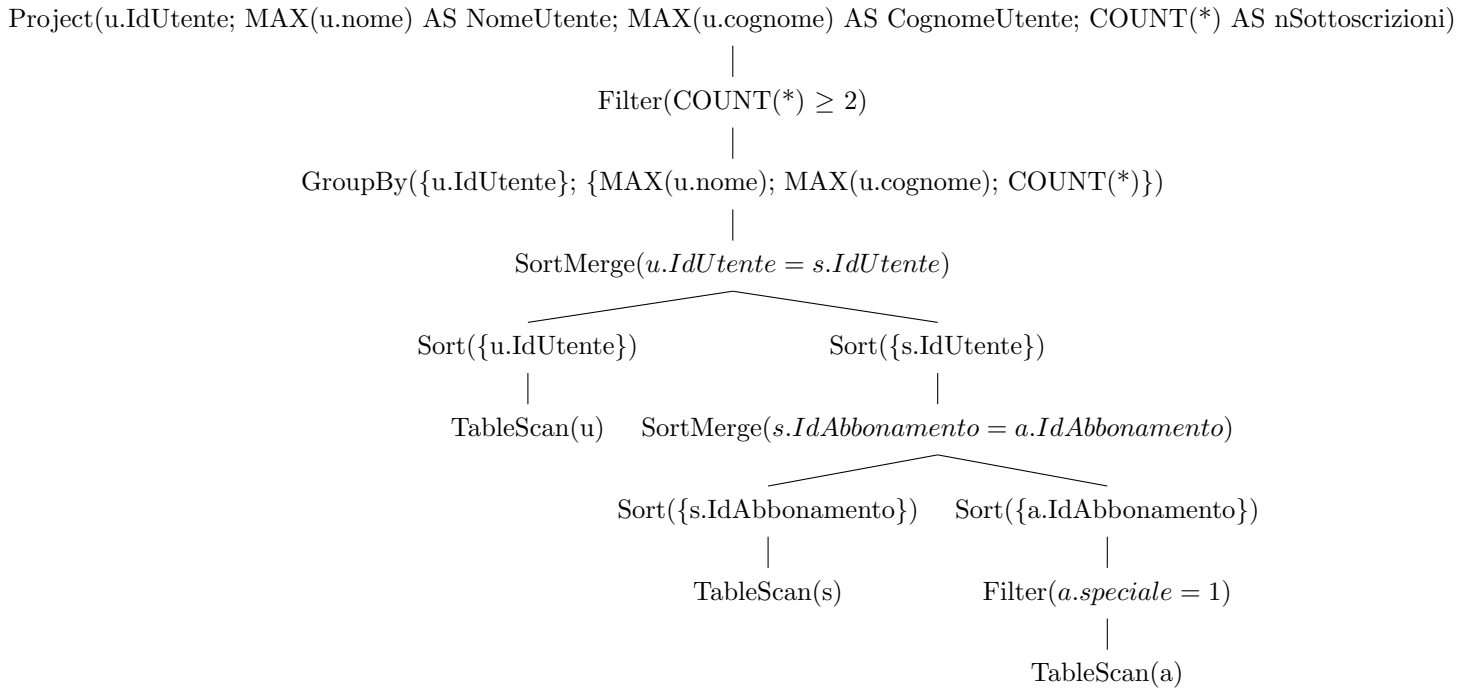
Esercizio 5.2.c

Previa ridenominazione per semplificare la scrittura: Abbonamenti → a, Utenti → u, Sottoscrizioni → s.

Versione 1



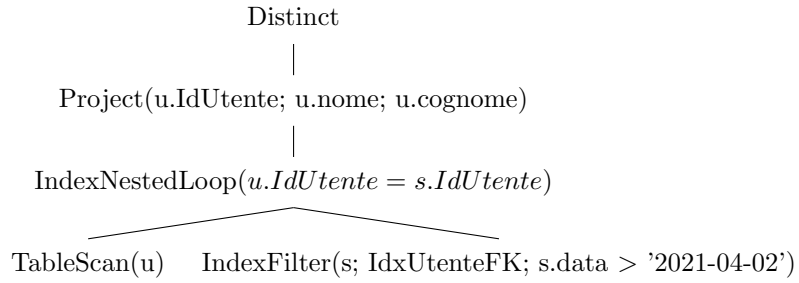
Versione 2



Questa versione contiene 13 operatori al posto dei 14 dell'altra; anche in questo caso la Sort prima della GroupBy può essere eliminata poiché l'output di SortMerge al quarto livello dell'albero risulta ordinato su u.IdUtente.

Esercizio 5.3.a

Previa ridenominazione per semplificare la scrittura: Utenti → u, Spedizioni → s; si assumono inoltre gli indici IdxUtentePK su u.IdUtente, IdxUtenteFK su s.IdUtente.

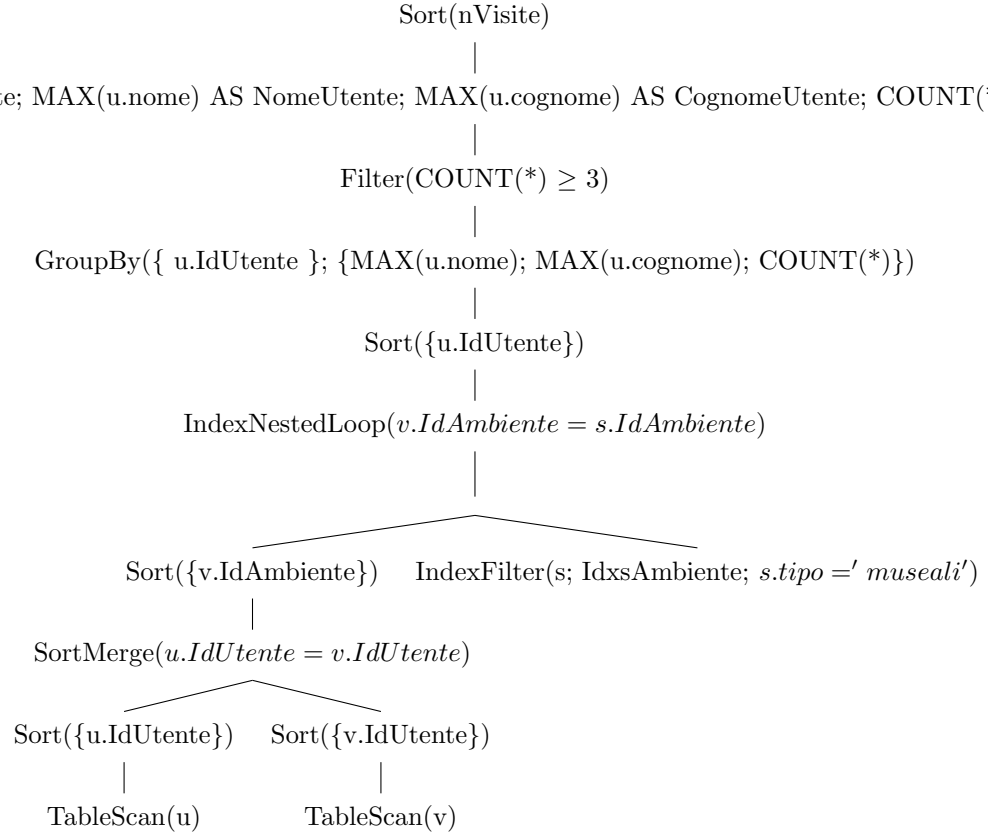


Non si fa uso di IdxUtentePK in quanto un operatore IndexFilter(u, ...) avrebbe condizione *true* e sarebbe "equivalente" a IndexScan.

Esercizio 5.3.b

Versione 1

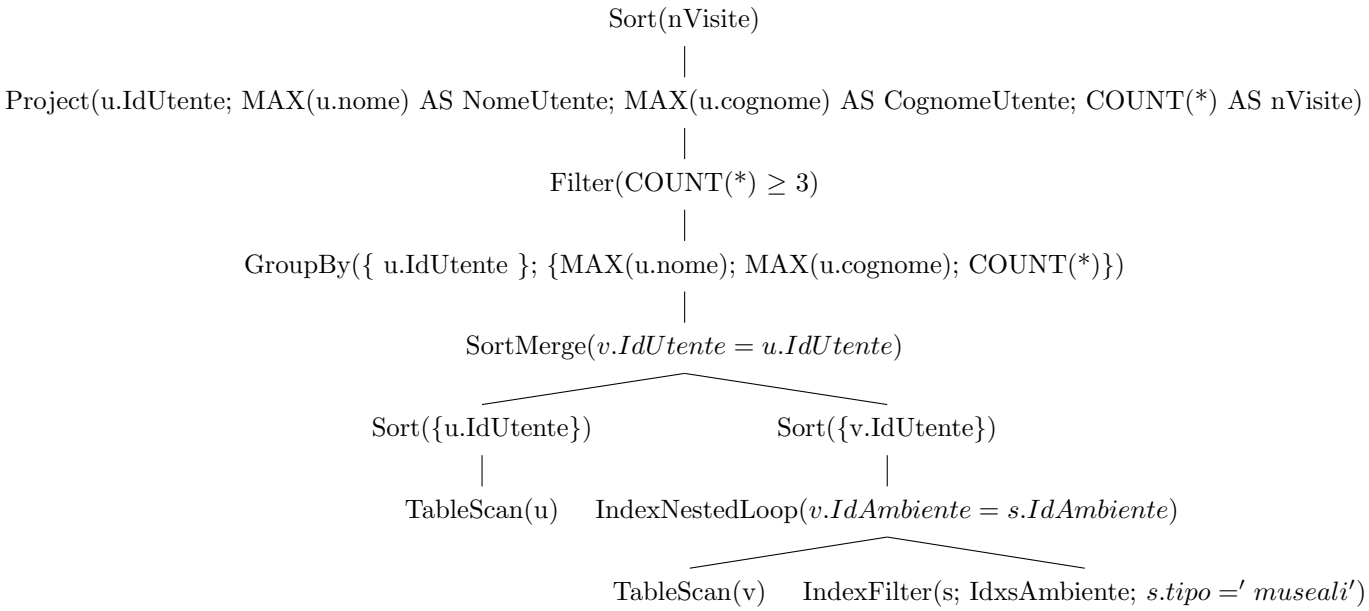
Previa ridenominazione per semplificare la scrittura: Visite → v, Utenti → u, Sale → s; si assumono inoltre gli indici IdxvAmbiente su v.IdAmbiente, IdxsAmbiente su s.IdAmbiente.



Per lo stesso motivo che al punto precedente, non risulta necessario fare uso di IdxvAmbiente.

Versione 2

Previa ridenominazione per semplificare la scrittura: Visite → v, Utenti → u, Sale → s; si assumono inoltre gli indici IdxvAmbiente su v.IdAmbiente, IdxsAmbiente su s.IdAmbiente, IdxuUtente su u.IdUtente.

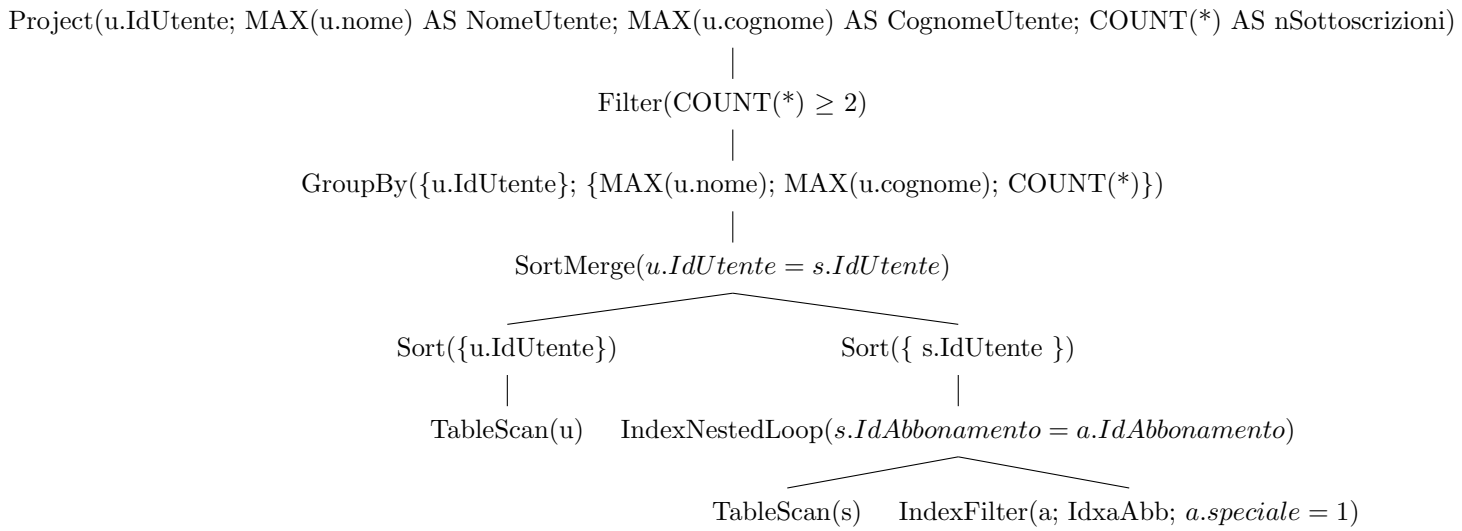


La Sort prima della GroupBy può essere eliminata e - come prima - non si fa uso di IdxvAmbiente per lo stesso motivo. Non si ricorre neanche a IdxuUtente in quanto *non* è sulla relazione interna; anche invertendo le due metà del sottoalbero si ripresenterebbe il problema di non avere condizioni su u: non risulta dunque possibile convertire SortMerge in una IndexNestedLoop.

Esercizio 5.3.c

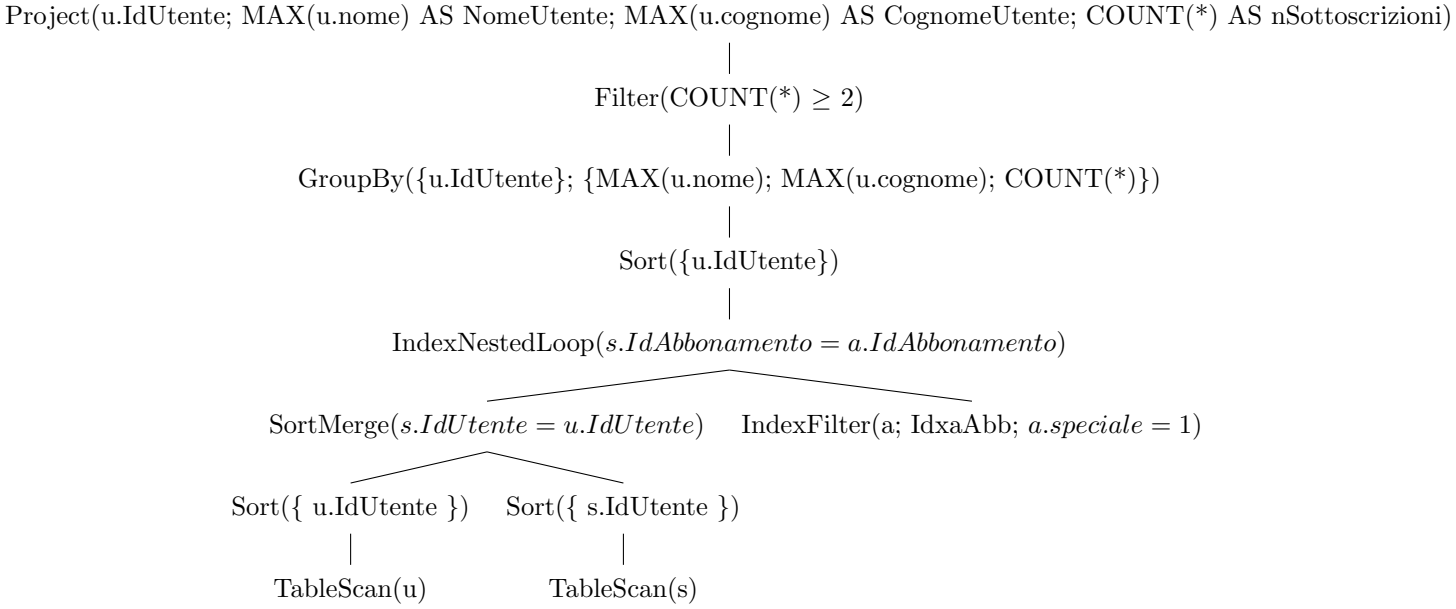
Previa ridenominazione per semplificare la scrittura: Abbonamenti → a, Utenti → u, Sottoscrizioni → s; si assumono inoltre gli indici IdxuUtente su u.IdUtente, IdxsAbb su s.IdAbbonamento e IdxaAbb su a.IdAbbonamento.

Versione 1



La Sort prima della GroupBy viene eliminata, si fa uso di un solo indice per gli stessi motivi che al punto (b); SortMerge non risulta convertibile in IndexNestedLoop.

Versione 2



Si faccia riferimento ai piani di accesso precedenti per il mancato uso di IdxuUtente e IdxsAbb.