SOMMARIO

3 / 10 / 2024

AVVISI INIZIALI

Lezioni

Definizione dei dati in Sql

Ogni giovedì, tranne impedimenti miei o di uno dei miei colleghi.

12 lezioni * 2h= 24 h

Eventuali recupero: mediante scambio di lezioni con gli altri prof., se è possibile; altrimenti, dopo Natale.

Ricevimento

Dopo la lezione in aula Ke (se è libera) oppure a distanza mediante Zoom, con prenotazione.

Il link è su Moodle.

Contatti con me

Contatti per motivi personali: Messaggi mediante il Moodle di STEM.

Contatti per motivi didattici o di interesse generale: Messaggi mediante il Forum del corso.

Moodle

Trovate una sezione dedicata alla mia parte del corso.

Vi trovate:

- Il link al mio URL di ricevimento mediante Zoom.
- Una cartella contenente materiale didattico complementare.
- Il forum “Esercizi”.
  Sarà usato per discutere le vostre soluzioni agli esercizi che vi assegnerò.
  NB: Per rendere leggibile il forum, sarà aperto esattamente un unico argomento di discussione per ogni esercizio che vi proporrò, seguendo la regola seguente:
  - Prima, io aprirò il nuovo argomento di discussione, dandogli, come nome, l’identificatore dell’esercizio (ad es. ESE.6.1.2)
  - Poi, voi inserirete i vostri interventi in risposta al mio intervento di apertura.
- Il forum “Lezioni”.
  Sarà usato per chiedere chiarimenti sugli argomenti svolti a lezione. Potrete anche rispondere alle domande dei vostri compagni.

In futuro, potrò aggiungere altri elementi in questa sezione (sondaggi, ecc.)

Modalità d’esame

Ve ne parleranno i prof. Di Nunzio o Marchesin.

Sussidi e libri

~ Seguirò il libro Atzeni.
Il programma della mia parte del corso è contenuto nel cap. 4, ed eventualmente nel cap. 5.
Indicherò il riferimento a questo libro mediante il simbolo ATZE.

~ Ho postato un documento che descrive come potete scaricare e installare PostgreSQL (d'ora in poi, Postgres), che è un DBMS open source. (Database Management System)

~ Posterò uno o più documenti contenenti integrazioni, chiarimenti, esercizi risolti ed esercizi da risolvere.
Indicherò il riferimento a questo documento mediante il simbolo INTE.
Indicherò il riferimento agli esercizi di questo documento mediante il simbolo ESE.
Indicherò il riferimento ai file sorgenti che ho postato mediante il simbolo SORGE.

~ Potete cercare approfondimenti su Internet.

~ Scriverò eventuali integrazioni mediante un word processor (come adesso), anziché mediante un programma di presentazione. Posterò questo file (dove sto scrivendo adesso) su Moodle dopo ogni lezione.

Programma della mia parte del corso

Io insegnerò Sql, che è il linguaggio standard per la programmazione dei database (d'ora in poi, db) relazionali. Userò soltanto le istruzioni basiche.

Le altre due parti del corso sono:

~ Progettazione concettuale mediante il modello E-R.

~ Progettazione logica mediante il modello relazionale.

Adesso faccio un disegnino di raccordo tra l'Sql e gli altri argomenti del corso.

Cliente	Progettista del modello dei dati (voi)	Progettista del database (voi)	Cliente
Scrive un documento contenente le specifiche del progetto.	Coerentemente con il documento precedente, progetta un db relazionale.	Coerentemente con il db relazionale progettato, progetta le istruzioni Sql per: ~ Creare il db. ~ Modificare i dati del db. ~ Cercare quel sottoinsieme di dati che soddisfa una data proprietà richiesta dal cliente.	Mediante le istruzioni progettate: ~ Modifica i dati del db. ~ Cerca i sottoinsieme di dati che gli interessano.

Per chi vuole programmare mediante il computer

Nel corso farò riferimento all'interprete Sql contenuto in Postgres.

Il Postgres è funzionale al corso per voi per programmare in Sql usando i file che vi posterò.

Prerequisiti matematici / INTE.50

Prerequisiti informatici

Programmazione:

~ Individuazione degli input e degli output di un problema.
NB In Sql, per default l'output è il video.

~ Tipi

~ Costrutti della programmazione: selezioni, cicli, ricorsione.

~ Notazione O grande.

~ Algoritmi di ordinamento / Fondamenti.

INTRODUZIONE AL LINGUAGGIO SQL / ATZE.4.1, INTE.52.1

DEFINIZIONE DEI DATI IN SQL / ATZE.4.2

I domini elementari

Stringhe

~ `CHARACTER` si usa per gestire un singolo carattere, cioè una stringa di lunghezza costante = 1.
Es: `CHARACTER Canale /* Canale può valere 'A' o 'B. */`

~ `CHARACTER(lunghezzaMassima)` si usa per gestire una stringa di lunghezza costante > 1.
Il parametro `lunghezzaMassima` indica la lunghezza massima della stringa.
Le costanti di tipo stringa si mettono tra apici semplici.
Es `'Stringa'`
Nelle versioni Sql più recenti si possono mettere alternativamente tra apici doppi.
Es `"Stringa"`.
Potete usare il delimitatore che vi piace di più.

~ `VARCHAR(lunghezzaMassima)` si usa per gestire una stringa di lunghezza variabile.
Il parametro `lunghezzaMassima` indica la lunghezza massima della stringa.

Tipi numerici esatti

~ `DECIMAL` si usa per gestire i reali esatti.
Precisione = N. totale di cifre destinate a memorizzare il valore dell'attributo
Scala = N di cifre destinate a memorizzare la parte frazionaria del valore dell'attributo
NB: Se le quantità di cifre indicate dal programmatore non sono sufficienti, SQL aggiunge automaticamente altre cifre.

~ `INTEGER` si usa per gestire gli interi esatti.

~ `SMALLINT` si usa per gestire gli interi esatti piccoli
Vi ricordo che un tipo intero memorizzato su `nB` bit può gestire 2 ^ `nB` valori differenti, distribuiti nell'intervallo [2 ^ (`nB` – 1), (2 ^ (`nB` – 1) – 1)]

Tipi reali approssimati

~ `FLOAT` si usa per gestire i reali approssimati
Vi ricordo che la memorizzazione di un valore reale approssimato può avere un errore di arrotondamento.
Es

```
d1 = (1 / 7) * 7;
d2 = 1;
```

Se confronto i due valori suddetti mediante l'operatore == (previsto nei linguaggi di C o Java), l'esito del confronto può essere false.
Un confronto corretto deve essere fatto mediante un'istruzione del tipo;

```
if (abs(d1 – d2) < errore) /* Indica se $d1 è circa uguale a $d2 */
```

Istanti temporali

~ `DATE` si usa per gestire un istante temporale di tipo "data"
Possiede i campi: `YEAR`, `MONTH`, `DAY`.
Es: L'attributo `Oggi` è di tipo `DATE`.
Voglio sapere qual è il mese contenuto in `Oggi`. Lo ottengo mediante `Oggi.MONTH`.

~ `TIME` si usa per gestire un istante temporale di tipo "orario"
Possiede i campi: `HOUR`, `MINUTE`, `SECOND`.

~ `TIMESTAMP` si usa per gestire un istante temporale contenente un tipo "data" insieme a un tipo "orario"
Possiede i campi di `DATE` e quelli di `TIME`.

NB Questo tipo non possiede l'operazione di addizione.

Intervalli temporali

~ `INTERVAL` si usa per gestire un intervallo temporale

Nota

I tipi suddetti possiedono:

~ le operazioni di confronto

~ l'operazione di differenza.

Invece, l'operazione di addizione non è posseduta dai tipi "stringa" e "istante temporale".

Booelani

~ `BOOLEAN` si usa per gestire un booleano
Possiede le costanti `false` e `true`.
`IsEsterno = 'true'` è errato
`IsEsterno = true` è corretto

10 / 10 / 2024

AVVISI INIZIALI

Vostra opinione sulla possibilità di usare Postgres in aula durante le lezioni

Il feedback è ancora aperto fino a tutto venerdì.

Vi ringrazio per le risposte fornite da voi; è un segnale che ci tenete al corso ☺.

Vi ricordo che:

~ La password è

~ L'uso del computer in aula non sarà "assistito". (Non posso passare tra i banchi per controllare i vostri lavori.)

~ È un esperimento didattico (che voi e io ci auguriamo positivo). E quindi, i vostri suggerimenti (che potrete fare anche quando il feedback sarà chiuso) saranno molto importanti.

DEFINIZIONE DEI DATI IN SQL (continuazione)

Altri domini che useremo / INTE.52.3.1

Intero autoincrementante

Definizione di (schema o) database

NB: La sintassi di ATZE non è standard.

Trovate la sintassi standard nel file sorgenti che posterò.

Definizione dei domini / ATZE.4.2.4, / INTE.52.3.2

Definizione delle tabelle / ATZE.4.2.3

Specifica dei valori di default / ATZE.4.2.5

Definizione dei vincoli / ATZE.4.2.6, INTE.52.3.4

Prerequisiti

Definisco in maniera intuitiva e informale alcuni concetti necessari per questa parte del corso.

Voi vedrete questi concetti in maniera formale e più dettagliata con altri docenti.

Attributo opzionale

Un attributo è detto **opzionale** se non è necessario che contenga un valore.

Se un attributo non è opzionale è detto **obbligatorio**.

Nelle relazioni un attributo opzionale è indicato mediante un *

Per indicare l'assenza di valore si dice che il valore è `NULL` (la parola è riconosciuta da Sql).

Es:

```
ABITANTI(..., DataMorte*)
    DataMorte = NULL => L'abitante è ancora vivo.
```

`DataMorte` è opzionale.

Un attributo può essere opzionale per due motivi molto differenti tra loro:

1 Il valore non esiste. (Es. La persona è ancora viva.)

2 Il valore esiste ma non è noto. (Es. La data di morte di una persona è ignota.)

NB Nei miei esempi, esercizi, ecc. un attributo sarà opzionale soltanto per il motivo 1. Le righe che non possiedono un valore hanno una qualità che le distingue dalle altre. (Nel nostro es., `DataMorte` non esiste quando l'abitante è ancora vivo.) Questa qualità dovrebbe essere sempre espressa.

Chiave candidata

Sono dati:

~ una relazione R

~ un suo sottoinsieme non vuoto K di attributi

K è detta **chiave candidata** di R se possiede entrambe queste proprietà:

1 K consente di individuare univocamente ogni tupla di R

2 Non ci sono attributi "inutili" in K.

Es Gestisco gli ospiti di un albergo che devono esibire un documento di tipo: passaporto (PP), patente automobilistica (PA), carta d'identità (CI).

```
OSPITI(IdeOspi, TipoDocu, IdeDocu, Cognome)
```

Un suo possibile insieme di valori è:

IdeOspi	TipoDocu	IdeDocu	Cognome
1	PA	10	ROSSI
2	PA	20	COLOMBO
3	CI	10	RUSSO

I seguenti insiemi sono CC?

```
{IdeOspi, TipoDocu}?    1? Sì       2? No       CC? No
{IdeOspi}?              1? Sì       2? Sì       CC? Sì
{TipoDocu, IdeDocu}?    1? Sì       2? Sì       CC? Sì
```

Chiave primaria (o chiave)

In generale una relazione può avere più chiavi candidate (anche se in verità è abbastanza raro che ne abbia più di 1). Nell'esempio precedente ci sono due CC.

Tra tutte le chiavi candidate, ne scelgo una i cui attributi sono tutti obbligatori, e la chiamo **chiave primaria**.

NB Almeno una chiave candidata deve essere composta da attributi tutti obbligatori. Eventualmente, il programmatore "aggiunge a posteriori" una chiave candidata con la proprietà suddetta.

Nel nostro esempio ho due scelte possibili; scelgo `IdeOspi`.

Quando scrivo la struttura di una relazione, devo esplicitare:

~ la chiave primaria mediante una sottolinea continua;

~ ogni chiave candidata, tranne la chiave primaria, mediante l'abbreviazione CC.

Quindi, la scrittura corretta del nostro esempio è:

```
OSPITI(IdeOspi, TipoDocu, IdeDocu, Cognome)
    TipoDocu € {PP, PA, CI}
    CC: {TipoDocu, IdeDocu}
```

Integrità referenziale

Es.: Consideriamo queste due relazioni:

`CITTA(NomeCitta, NAbitanti)`

Un suo possibile insieme di valori è:

NomeCitta	NAbitanti
RM	2000000
MI	1500000
NA	1200000

`STUDENTI(IdeMatriStu, NomeCittaNasci, NomeCittaResi)`

Un suo possibile insieme di valori è:

IdeMatriStu	NomeCittaNasci	NomeCittaResi
1	RM	NA
2	MI	MI
3	NA	RM
~~4~~	~~TO~~	~~RM~~

Insensato

Se adesso voglio indicare che la matricola 4 è nata a TO, ciò è insensato, perché TO non è elencata tra le città esistenti.

Se voglio inserire la matricola 4 in `STUDENTI`:

~ Prima devo inserire la citta TO in `CITTA`

~ Dopo posso inserire la matricola 4 (nata a Torino) in `STUDENTI`

Finché non ho inserito TO, il DBMS dovrà proibirmi l'inserimento della matricola 4.

Definisco questa proibizione mediante un vincolo di IR da `STUDENTI.NomeCittaNasci` a (oppure verso) `CITTA.NomeCitta`.

L'insieme degli attributi sui quali sussiste un vincolo di IR è detto **chiave esterna**. (Nel nostro esempio, una chiave esterna è l'attributo `STUDENTI.NomeCittaNasci`).

Io metterò una sottolinea ondulata sotto ogni chiave esterna.

Quindi, la relazione `STUDENTI` va scritta in questo modo:

```
STUDENTI(IdeMatriStu, NomeCittaNasci, NomeCittaResi)
    IR: NomeCittaNasci -> CITTA.NomeCitta
    IR: NomeCittaResi -> CITTA.NomeCitta
```

Osservate che il vincolo di IR proibisce all'utente di inserire uno studente nato in una città non ancora inserita.

Però, l'utente potrebbe comunque commettere un inserimento errato; ad es, potrebbe inserire che lo stu-

dente è nato a MI, anziché a NA. Questo errore è possibile perché MI compare nella tabella CITTA.

Integrità referenziale su due o più attributi

Il vincolo di IR può sussistere anche tra insiemi composti da due o più attributi.

Nel problema degli ospiti dell'albergo, voglio gestire anche i servizi (aggiuntivi) che gli ospiti hanno usato. Lo gestisco mediante una nuova relazione

SERVIZI_USATI(IdeSeUsa, Descrizione, TipoDocu, IdeDocu)

Un suo possibile insieme di valori è:

IdeSeUsa	Descrizione	IdeDocu	IdeDocu
1	Piscina	PA	10
2	Caffè	PA	10
3	Piscina	CI	20

Insensato

Il servizio associato a TipoDocu = CI e a IdeDocu = 20 è insensato, perché non esiste un ospite (nell'albergo) con TipoDocu = CI e IdeDocu = 20.

Il problema si risolve definendo un unico vincolo che coinvolge due attributi per ognuna delle due relazioni. Nel nostro es, il vincolo coinvolge due attributi di SERVIZI_USATI verso due attributi di OSPITI.

Quindi, la soluzione corretta è la seguente, che richiede un vincolo di IR.

SERVIZI_USATI(IdeSeUsa, Descrizione, TipoDocu, IdeDocu)
    IR: {TipoDocu, IdeDocu} -> OSPITI.{TipoDocu, IdeDocu}

La seguente soluzione, che prevede due vincoli di IR, è errata:

SERVIZI_USATI(IdeSeUsa, Descrizione, TipoDocu, IdeDocu)
    IR: SERVIZI_USATI.TipoDocu -> OSPITI.TipoDocu
    IR: SERVIZI_USATI.IdeDocu -> OSPITI.IdeDocu

NB I due vincoli suddetti sono errati, perché consentono l'inserimento di un servizio richiesto dall'inesistente ospite (CI, 2).

Da quanto detto prima segue che:

~ Ci possono essere 0, 1 2 o più chiavi esterne.
~ Una chiave esterna può essere composta da 1 o più attributi.

16 / 10 / 2024

AVVISI INIZIALI

Vostra opinione sulla possibilità di usare PostgreSql in aula durante le lezioni

Vi ringrazio per le vostre risposte.

Tra poco entreremo nella parte più viva del corso, in cui programmeremo “in continuazione”.

Correzioni / INTE.52.3.1

La sintassi standard per definire per un intero autoincrementante quando è chiave primaria è:

```
NomeAttributo INTEGER ~~PRIMARY KEY~~ GENERATED BY DEFAULT AS IDENTITY
```

Definizione dei vincoli (continuazione) / ATZE.4.2.6, INTE.52.3.3

~ Vincoli di chiave primaria

~ Vincoli di chiave candidata

Definizione dei vincoli (continuazione) / ATZE.4.2.7, INTE.52.3.3

~ Vincoli di integrità referenziale

Sintassi

Ricordo che uso i seguenti termini per gestire un’IR:

~ Tabella vincolata (o interna), abbreviata in `TABE_VINCO`:
È la tabella che ha il vincolo di IR.

~ Tabella principale (o esterna), abbreviata in `TABE_PRINCI`:
È la tabella che vincola `TABE_VINCO`, perché contiene i valori che `TABE_VINCO` può contenere.

~ Simbologia per gestire un’IR:

  ~ IR: `Attributo -> TABE_PRINCI.Attributo`, se il vincolo è associato a un singolo attributo;

  ~ IR: `{Attributo1, Attributo1} -> {TABE_PRINCI.Attributo}`, se il vincolo è associato a due attributi.

Possiamo definire ognuno dei vincoli di chiave primaria, di chiave candidata, di integrità referenziale, mediante una sintassi che è sempre applicabile.

Esiste una seconda sintassi (alternativa alla prima) che è applicabile soltanto quando il vincolo è associato a un singolo attributo.

Sintetizzo le varie situazioni nello schema seguente.

Tipo di vincolo	N. di attributi	Sintassi
Chiave (primaria)	Qualsiasi	Dopo avere definito ogni attributo, aggiungiamo `PRIMARY KEY (Attributo1, Attributo2)`
Chiave (primaria)	1	Quando definiamo l'attributo, specifichiamo `Attributo Dominio PRIMARY KEY`
Chiave candidata (non primaria)	Qualsiasi	Dopo avere definito ogni attributo, aggiungiamo `UNIQUE (Attributo1, Attributo2)`
Chiave candidata (non primaria)	1	Quando definiamo l'attributo, specifichiamo `Attributo Dominio UNIQUE`

Integrità referenziale	Qualsiasi	Dopo avere definito ogni attributo, aggiungiamo
		`FOREIGN KEY (Attributo1, Attributo2)`
		`REFERENCES TABE_PRINCI(Attributo1, Attributo2)`

Integrità referenziale	1	Quando definiamo l'attributo, specifichiamo `Attributo Dominio REFERENCES TABE_PRINCI(Attributo)`

Esempi su gestione di: Db, tabelle, domini / SORGE.{Prova0, Prova1, Prova2}

Trovate degli esempi di guida nei file suddetti (postati su Moodle).

Potete facilmente eseguire le istruzioni contenute nei file suddetti mediante i seguenti comandi di Postgres:

```
\i 'H:\\Linguaggi_1\\PostgreSQL\\Prove0_DominiEChiaviVuote.sql'

\i 'H:\\Linguaggi_1\\PostgreSQL\\ Prove1_InseriErrati.sql'

\i 'H:\\Linguaggi_1\\PostgreSQL\\Prove2_DominiVari.sql'
```

Politiche di reazione in caso di modifiche / Esempio

CITTA

NomeCitta	NAbitanti
RM	2000000
MI	1500000
NA	1200000

STUDENTI

IdeMatriStu	NomeCitta*
1	RM
2	MI
3	RM
4	NULL

Naturalmente, sussiste la seguente IR: `STUDENTI.NomeCitta -> CITTA.NomeCitta`

Consideriamo le due operazioni di aggiornamento e di cancellazione.

Aggiornamento

Supponiamo che un utente, nella tabella `CITTA`, aggiorni il nome RM in XX. Di conseguenza, la tabella

STUDENTI non può più contenere il valore RM.

Sintetizzo nello schema seguente le conseguenze di ogni politica di reazione.

<table>
<tr><th>Politica di reazione</th><th>Note</th><th>Modifica effettuata dal DBMS sulla tabella STUDENTI</th></tr>
<tr><td>CASCADE</td><td></td><td>Se il valore RM compare in STUDENTI.NomeCitta, il DBMS lo aggiorna in XX:
<table><tr><th>IdeMatriStu</th><th>NomeCitta</th></tr><tr><td>1</td><td>XX</td></tr><tr><td>2</td><td>MI</td></tr><tr><td>3</td><td>XX</td></tr><tr><td>4</td><td>NULL</td></tr></table></td></tr>
<tr><td>SET NULL</td><td>L’attributo STUDENTI.NomeCitta deve essere opzionale; altrimenti, ci sarà un errore di sintassi.
(Infatti, se l’attributo fosse obbligatorio, il DBMS non potrebbe assegnargli il valore NULL.)</td><td>Se il valore RM compare in STUDENTI.NomeCitta, il DBMS lo aggiorna in NULL:
<table><tr><th>IdeMatriStu</th><th>NomeCitta</th></tr><tr><td>1</td><td>NULL</td></tr><tr><td>2</td><td>MI</td></tr><tr><td>3</td><td>NULL</td></tr><tr><td>4</td><td>NULL</td></tr></table></td></tr>
<tr><td>SET DEFAULT</td><td>Se il valore di default di STUDENTI.NomeCitta non compare nell’attributo CITTA.NomeCitta, allora il DBMS rifiuta la modifica effettuata dall’utente.
Invece, se non compare, il DBMS rifiuta la modifica; quindi, entrambe le tabelle CITTA e STUDENTI rimangono immutate.
(Infatti, se il DBMS accettasse la modifica, la tabella STUDENTI avrebbe un valore che non c’è nella CITTA, e questo violerebbe l’integrità referenziale.)</td><td>Se il valore di default di STUDENTI.NomeCitta compare in CITTA, il DBMS lo aggiorna nel valore di default di STUDENTI.NomeCitta:
<table><tr><th>IdeMatriStu</th><th>NomeCitta</th></tr><tr><td>1</td><td>ValoreDiDefault</td></tr><tr><td>2</td><td>MI</td></tr><tr><td>3</td><td>ValoreDiDefault</td></tr><tr><td>4</td><td>NULL</td></tr></table></td></tr>
<tr><td>NO ACTION
NB: È la politica di default.</td><td>Se il valore di CITTA.NomeCitta che l’utente vuole modificare compare nell’attributo STUDENTI.NomeCitta, allora il DBMS rifiuta la modifica effettuata dall’utente; quindi, entrambe le tabelle CITTA e STUDENTI rimangono immutate.
(Infatti, se il DBMS accettasse la modifica, la tabella STUDENTI avrebbe un valore che non c’è nella tabella CITTA, e questo violerebbe l’integrità referenziale.)</td><td></td></tr>
</table>

Cancellazione

Supponiamo che un utente, nella tabella `CITTA`, cancelli il nome RM dalla tabella CITTA.

Anche in questo caso, la tabella STUDENTI non può più contenere il valore RM.

I risultati sono identici alla situazione di modifica (che ho presentato precedentemente) con l'unica eccezione di `CASCADE`.

Politica di reazione	Modifica effettuata dal DBMS sulla tabella STUDENTI
`CASCADE`	Se il valore RM compare in `STUDENTI.NomeCitta`, il DBMS cancella ogni riga dove RM compare:

IdeMatriStu	NomeCitta
2	MI
4	NULL

Esempi su gestione di IR / SORGE.Prova3

Trovate degli esempi di guida nel file suddetto (postato su Moodle).

Potete facilmente eseguire le istruzioni contenute nei file suddetti mediante il seguente comando di Postgres:

```
\i 'H:\\Linguaggi_1\\PostgreSQL\\Prove3_IntegriRefere.sql'
```

Cataloghi relazionali (cenni) / ATZE.4.2.9

In Postgres il catalogo relazionale è visibile mediante il comando `\l` (il carattere dopo il backslash è una elle).

MODIFICA DEI DATI IN SQL (CENNI) / ATZE.4.4

Alla fine del corso potremo vedere altre forme di modifica. Adesso mi limito alla forma più semplice.

Inserimento / ATZE.4.4.1

Vediamo soltanto la forma di inserimento diretto di valori, mediante la parola chiave `VALUES`.

Gli esempi SORGE.{Prova0, Prova1, Prova2, Prova3} inseriscono dati soltanto mediante questa forma.

INTERROGAZIONI IN SQL / ATZE.4.3, INTE.52.4

Dichiaratività di SQL (cenni) / ATZE.4.3.1

Database di riferimento / INTE.52.2

In molti esempi farò riferimento al db contenuto in INTE.52.2.

17 / 10 / 2024

INTERROGAZIONI IN SQL (CONTINUAZIONE)

Interrogazioni semplici / ATZE.4.3.2

Clausole

Nel linguaggio Sql le interrogazioni (o ricerche) si fanno mediante il comando `SELECT`. Questo comando (nella versione basica) prevede sei clausole:

`SELECT`, `FROM`, `WHERE` e altre 3.

Il comando `SELECT` più semplice possibile è:

```
SELECT Attributo
FROM Tabella;
```

Le clausole devono essere scritte secondo un certo ordine fissato dalla sintassi:

```
1   SELECT
2   FROM
3   WHERE
```

L'interprete Sql elabora le clausole secondo un certo ordine fissato (dalla logica):

```
1   FROM
2   WHERE
5   SELECT
```

Clausola Select / ATZE.4.3.2

Seguita da un attributo

Il comando

```
SELECT NomeDipa
FROM DIPARTIMENTI;
```

visualizza

NomeDipa
HW
SW
CE

Seguita da due o più attributi

```
SELECT NomeDipa, Citta
FROM DIPARTIMENTI;
```

Seguito da * (al posto dell'elenco degli attributi)

Mostra ogni attributo.

```
SELECT *
FROM DIPARTIMENTI;
```

Ridenominazione

Possiamo rinominare il nome dell'attributo che comparirà a video. Tipicamente, lo facciamo per due motivi:

1 Per visualizzare l'attributo mediante un nome più esplicativo.

2 Per intestare un'espressione che stiamo visualizzando.

Adesso vediamo un es. del motivo 1.

Vedremo tra poco un esempio del motivo 2.

Es Visualizzate, per ogni impiegato, la sua città di residenza esplicitando più chiaramente che è la città di residenza.

Il comando

```
SELECT MatriImpie, Citta AS CittaResidenza
FROM IMPIEGATI;
```

visualizza

MatriImpie	CittaResidenza
1	PD
2	VI
3	MI
4	MI
5	PD
6	PD
7	VI

Espressione

È possibile inserire un'espressione, ottenuta elaborando uno o più valori provenienti da ciascuna riga.

Es Visualizzate, per ogni impiegato, il suo stipendio mensile.

In questo es. vediamo, oltre l'uso di un'espressione, anche una ridenominazione di attributo causata dal motivo 2 suddetto.

```
SELECT MatriImpie, Stipendio / 12 AS StipendioMensile
FROM IMPIEGATI;
```

visualizza

MatriImpie	StipendioMensile
1	4
2	7
..	..

Es Durata dei prestiti di una biblioteca

```
PRESTITI_CONCLUSI
```

IdePrestito	DataInizio	DataFine
1	2024-01-02	2024-01-05
2	2024-01-02	2024-02-02

```
SELECT IdePrestito, DataFine - DataInizio AS Durata
FROM PRESTITI_CONCLUSI;
```

visualizza

IdePrestito	Durata
1	3
2	31

Clausola Where / ATZE.4.3.2

La clausola `WHERE` deve essere seguita da un'espressione booleana.

L'espressione booleana più semplice confronta due valori appartenenti allo stesso dominio oppure a domini compatibili. (Ad es., un reale è confrontabile con un intero.)

Gli operatori di confronto sono:

`>` `>=` `<` `<=` `<>` `=`

`LIKE` (definito soltanto sul tipo stringa)

Gli operatori booleani sono:

`AND` `OR` `NOT`

I due operandi che combino mediante un operatore BOOLEANO devono essere entrambi booleani.

I due operandi che combino mediante un operatore DI CONFRONTO devono essere dello stesso tipo (qualsiasi) o di tipi compatibili.

Es: Supponiamo che la tabella `IMPIEGATI` possieda un attributo opzionale `AnnoDimissione`.

Usando l'ipotesi suddetta, visualizzate gli impiegati che lavorano attualmente nell'azienda.

```
SELECT *
FROM IMPIEGATI
WHERE AnnoDimissione IS NULL;
```

visualizza ogni dipendente che non ha un valore nell'attributo `AnnoDimissione`, cioè che lavora ancora nell'impresa.

Analogamente

```
SELECT *
FROM IMPIEGATI
WHERE AnnoDimissione IS NOT NULL;
```

visualizza ogni dipendente che ha un valore nell'attributo `AnnoDimissione`, cioè che non lavora più nell'impresa.

Clausola From

Prodotto cartesiano / INTE.52.4.1

Equi join interno / INTE.52.4.2

CITTA (= T1)

NomeCitta	NAbitanti
RM	100
MI	80

STUDENTI (= T2)

MatriStu	NomeCitta
10	RM
20	RM
30	MI

IR: NomeCitta -> CITTA.NomeCitta

Il prodotto cartesiano crea la tabella seguente, avente 4 colonne (= 2 + 2) e 6 (= 2 * 3) righe.

CITTA. NomeCitta	CITTA. NAbitanti	STUDENTI. MatriStu	STUDENTI. NomeCitta
RM	100	10	RM
RM	100	20	RM
RM	100	30	MI
MI	80	10	RM
MI	80	20	RM
MI	80	30	MI

NOTE	
Concatenazione ..	Interes-sante?
tra la riga 1 di T1 e la riga 1 di T2	SÌ
tra la riga 1 di T1 e la riga 2 di T2	SÌ
tra la riga 1 di T1e la riga 3 di T2	NO
tra la riga 2 di T1e la riga 1 di T2	NO
tra la riga 2 di T1 e la riga 2 di T2	NO
tra la riga 2 di T1 e la riga 3 di T2	SÌ

Es Visualizzate, per ogni studente, la sua matricola, la città in cui risiede e il numero di concittadini.

```
SELECT MatriStude, STUDENTI.NomeCitta, NAbitanti - 1
FROM CITTA JOIN STUDENTI ON CITTA.NomeCitta = STUDENTI.NomeCitta;
```

24 / 10 / 2024

SOLUZIONE DEGLI ESERCIZI PER CASA

SOLU.Azienda1

MODIFICHE / ATZE.4.4

Cancellazione

Aggiornamento

DB AEROPORTI / INTE.52.8.1, SORGE.AEROPORTI1

NB: È differente da quello di ATZE.

ESE.Aula: aeroporti.{B, C}

JOIN

Ripasso / INTE.52.5.2

ESE: Studenti pendolari

CITTA (= T1)

NomeCitta	NAbitanti
RM	100
MI	80

SCUOLE (= T1)

IdeScuola	NomeCitta
1	RM
2	RM
3	MI

IR: NomeCitta -> CITTA.NomeCitta

STUDENTI (= T2)

MatriStu	IdeScuola	NomeCittaResi
10	1	RM
20	3	RM
30	3	MI

IR: IdeScuola -> SCUOLE.IdeScuola

IR: NomeCittaResi -> CITTA.NomeCitta

Elencate gli studenti che risiedono in una città differente da quella in cui vanno a scuola.

Soluzione 1

```
SELECT STUDENTI.*
FROM SCUOLE, STUDENTI
WHERE NomeCitta <> NomeCittaResi AND SCUOLE.IdeScuola = STUDENTI.IdeScuola
```

Soluzione 2

Inserisco le opportune (e non obbligatorie) ridenominazioni delle tabelle.

```
SELECT ST.*
FROM SCUOLE AS SC, STUDENTI AS ST
WHERE SC.NomeCitta <> ST.NomeCittaResi AND SC.IdeScuola = ST.IdeScuola
```

Soluzione 3

Riscrivo la stessa soluzione mediante JOIN ON.

```
SELECT ST.*
FROM SCUOLE AS SC JOIN STUDENTI AS ST ON
  SC.NomeCitta <> ST.NomeCittaResi AND SC.IdeScuola = ST.IdeScuola;
```

Es. di espressione booleana nel WHERE

Modifico la consegna, in modo da visualizzare soltanto gli studenti con la matricola > 100;

```
SELECT ST.*
FROM SCUOLE AS SC JOIN STUDENTI AS ST ON
  SC.NomeCitta <> ST.NomeCittaResi AND SC.IdeScuola = ST.IdeScuola
WHERE ST.MatriStude > 100;
```

Note importanti / INTE.52.3.3

ESE.Riso: INTE.52.5.3.1

ESE.Casa: aeroporti.D.

Il join gode della proprietà commutativa e di quella associativa.

Il join su tre tabelle è:

```
FROM (T1 JOIN T2 ON T1.X = T2.X) JOIN T3 ON T2.Y = T3.Y
```

Per la proprietà associativa,. il suo risultato equivale a:

```
FROM T1 JOIN (T2 JOIN T3 ON T2.Y = T3.Y) ON T1.X = T2.V
```

CLAUSOLA SELECT / ARGOMENTO DISTINCT / ATZE.4.3.2

31 / 10 / 2024

CLAUSOLA SELECT / ARGOMENTO DISTINCT (CONCLUSIONE) / ATZE.4.3.2

DISTINCT è sicuramente inutile quando gli attributi da visualizzare contengono una chiave candidata.

```
T(K, CC1A, CC1B, CC2, A)
    CC: {CC1A, CC1B}
    CC: CC2

SELECT K /* DISTINCT inutile */
FROM T;

SELECT CC1A /* DISTINCT utile */
FROM T;

SELECT CC2 /* DISTINCT inutile */
FROM T;

SELECT A /* DISTINCT utile */
FROM T;
```

SOLUZIONE DEGLI ESERCIZI PER CASA

SOLU.Aeroporti.D

SELF JOIN

Il self join è un caso particolare del join, che avviene quando congiungiamo una tabella `T` con sé stessa. Ciò succede perché dobbiamo usare `T` con due differenti finalità.

Di conseguenza, dobbiamo ridenominare entrambe le tabelle. Naturalmente:

~ Commenteremo la finalità di ognuna di queste due tabelle.

~ Ridenomineremo mediante un nome evocativo.

NB: Il self join non è un'operazione differente dal join; è, semplicemente, un modo particolare di usare un join. Quindi, non necessita di una sintassi opportuna.

Esercizi risolti

1. Considerate il db di riferimento.

Elencate le coppie (Matricola di ogni impiegato che lavora nel dipartimento `'SW'`, Matricola di ogni impiegato con lo stesso nome di un impiegato di `'SW'`).

Soluzione

Riporto soltanto la porzione del db che è funzionale a questa consegna; quindi, escludo la tabella `DIPARTIMENTI` e alcuni attributi della tabella `IMPIEGATI`.

Devo congiungere due tabelle:

1) la selezione della tabella `IMPIEGATI` contenente gli impiegati che lavorano in `'SW'`:

`IMPIEGATI`

MatriImpie	Cognome	NomeDipa
1	A	SW
2	A	SW
~~3~~	~~B~~	~~HW~~
~~4~~	~~C~~	~~NULL~~
5	C	SW
~~6~~	~~D~~	~~NULL~~
7	E	SW

2) la tabella `IMPIEGATI` completa, per valutare quali di questi impiegati hanno lo stesso cognome degli impiegati della tabella precedente:

`IMPIEGATI`

MatriImpie	Cognome	NomeDipa
1	A	SW
2	A	SW
3	B	HW
4	C	NULL
5	C	SW
6	D	NULL
7	E	SW

L'output corrispondente è:

I_S.MatriImpie	I_O.MatriImpie	I_O.Cognome
1	2	A
2	1	A
5	4	C

```
/* I_S: Impiegati di Software */
/* I_O: Impiegati di qualsiasi dipartimento e
        che sono omonimi (rispetto al cognome) di un I_S
*/
SELECT I_S.MatriImpie, I_O.MatriImpie, I_O.Cognome
FROM IMPIEGATI AS I_S JOIN IMPIEGATI AS I_O
  ON I_S.Cognome = I_O.Cognome AND I_S.MatriImpie <> I_O.MatriImpie
  /* L'ultimo confronto elimina le coppie composte dallo stesso impiegato di 'SW'. */
WHERE I_S.NomeDipa = 'SW';
```

2. Considerate il db di riferimento.

Visualizzate, in ogni riga:

~ gli attributi `MatriImpie`, `Stipendio` degli impiegati che guadagnano più di 55;

~ gli attributi `MatriImpie`, `Stipendio` dei capi degli impiegati suddetti.

Soluzione

Riporto soltanto la porzione del db che è funzionale a questa consegna.

Devo congiungere due tabelle:

1) la selezione della tabella `IMPIEGATI` contenente gli impiegati che guadagnano più di 55:

MatriImpie	Stipendio	MatriImpieCapo
~~1~~	~~50~~	~~NULL~~
2	80	NULL
~~3~~	~~15~~	~~1~~
4	60	1
~~5~~	~~20~~	~~2~~
6	90	NULL
7	90	4

2) la tabella `IMPIEGATI` completa, per trovare le informazioni dei capi degli impiegati della tabella precedente:

MatriImpie	Stipendio	MatriImpieCapo
1	50	NULL
2	80	NULL
3	15	1
4	60	1
5	20	2
6	90	NULL
7	90	4

```
/* I_55: Impiegati che guadagnano più di 55 */
/* I_C: Capi di I_55 */
SELECT I_55.MatriImpie, I_55.Stipendio, I_C.MatriImpie, I_C.Stipendio
FROM IMPIEGATI AS I_55 JOIN IMPIEGATI AS I_C
  ON I_55.MatriImpieCapo = I_C.MatriImpie
WHERE I_55.Stipendio > 55;
```

L'output corrispondente è:

I_55.MatriImpie	I_55.Stipendio	I_C.MatriImpie	I_C.Stipendio
4	60	1	50
7	90	4	60

NB: Osservate che, nella condizione contenuta nel join dell'istruzione suddetta, non posso scambiare gli attributi `MatriImpieCapo` e `MatriImpie`.

Se lo facessi, otterrei un risultato errato.

Ad es., la seguente istruzione è errata:

```
SELECT I_55.MatriImpie, I_55.Stipendio, I_C.MatriImpie, I_C.Stipendio
FROM IMPIEGATI AS I_55 JOIN IMPIEGATI AS I_C
  ON I_55.MatriImpie = I_C.MatriImpieCapo
WHERE I_55.Stipendio > 55;
```

Infatti, genera l'output:

I_55.MatriImpie	I_55.Stipendio	I_C.MatriImpie	I_C.Stipendio
2	80	5	20
4	60	7	90

che visualizza ogni capo che guadagna più di 55, e ogni suo subalterno.

In altre parole, I_C non contiene i capi di I_55. Invece, contiene gli impiegati che hanno I_55 come capi, cioè contiene i subalterni di I_55.

3. Considerate il db di riferimento.

Visualizzate, in ogni riga:

~ gli attributi `MatriImpie`, `Stipendio` degli impiegati che guadagnano più del loro capo;

~ gli attributi `MatriImpie`, `Stipendio` dei capi degli impiegati suddetti.

Soluzione

Devo congiungere due tabelle:

1) la selezione della tabella `IMPIEGATI` contenente gli impiegati che guadagnano più del loro capo;

2) la tabella `IMPIEGATI` completa, per trovare le informazioni dei capi degli impiegati della tabella precedente:

Riporto soltanto la porzione del db che è funzionale a questa consegna:

`IMPIEGATI`

MatriImpie	Stipendio	MatriImpieCapo
1	50	NULL
2	80	NULL
3	15	1
4	60	1
5	20	2
6	90	NULL
7	90	4

```
/* I_P: Impiegati che guadagnano più del loro capo */
/* I_C: Capi di I_P */
SELECT I_P.MatriImpie, I_P.Stipendio, I_C.MatriImpie, I_C.Stipendio
FROM IMPIEGATI AS I_P JOIN IMPIEGATI AS I_C
  ON I_P.MatriImpieCapo = I_C.MatriImpie AND I_P.Stipendio > I_C.Stipendio;
```

L'output corrispondente è:

I_P.MatriImpie	I_P.Stipendio	I_C.MatriImpie	I_C.Stipendio
4	60	1	50
7	90	4	60

ESE.Aula: aeroporti.E.I

OPERATORI AGGREGATI / ATZE.4.3.3, INTE.52.5.5

Argomento COUNT: ATZE.Inte.20

Differenza tra DISTINCT e ALL

INTE.52.5.5.1

Ese Consideriamo l'attributo Stipendio della seguente tabella IMPIEGATI:

Stipendio
3
4
3
NULL

L'interrogazione seguente mostra il valore 2:

```
SELECT COUNT(DISTINCT Stipendio)
FROM IMPIEGATI;
```

L'interrogazione seguente mostra il valore 3:

```
SELECT COUNT(ALL Stipendio)
FROM IMPIEGATI;
```

INTE.52.5.5.1

Argomenti SUM; AVG, MAX, MIN

ATZE.Inte.{23, 24, 25, 26}

INTE.52.5.5.2

CLAUSOLA GROUP BY / ATZE.4.3.4, INTE.52.5.6

Posizione sintattica	Ordine di elaborazione	Clausola
1	5	SELECT
2	1	FROM
3	2	WHERE
4	3	GROUP BY
5	4	..
6	6	..

L'esempio di fig. ATZE.4.21 (nel par.4.3.4) contiene 4 diversi valori nell'attributo Dipart. Quindi, il GROUP BY sul suddetto attributo (Interrogazione.27) genera 4 gruppi differenti (uno per ogni valore dif-

ferente di Dipart), come si vede in fig. ATZE.4.23.

Successivamente, Sql applica l'operatore di aggregazione (che nell'interrogazione 27 è SUM) sul singolo gruppo (e non su tutta la tabella).

7 / 11 / 2024

SOLUZIONE DEGLI ESERCIZI PER CASA

SOLU.Aeroporti.F.1

Elencate le coppie di città collegate da voli internazionali.

Risolvete mediante Sql basico.

SOLU.Aeroporti.G

Indicate il numero di voli internazionali che partono la domenica da ‘Milano’.

La domenica è identificata mediante la stringa 'DO'.

L’Italia è identificata mediante la stringa 'ITA'.

CLAUSOLA GROUP BY (continuazione) / ATZE.4.3.4

Il GROUP BY raggruppa le righe di una tabella in uno o più gruppi, per consentire (mediante altre clausole dell’interrogazione) di elaborare ogni singolo gruppo prodotto dal GROUP BY.

Posizione sintattica	Ordine di elaborazione	Clausola
1	5	SELECT
2	1	FROM
3	2	WHERE
4	3	GROUP BY
5	4	..
6	6	..

Nota 1

Il GROUP BY può essere seguito da due o più attributi.

Es.

Modifico il GROUP BY dell’interrogazione 27 nel modo seguente:

```
GROUP BY Dipart, Ufficio
```

Adesso, il GROUP BY genera un gruppo per ogni valore differente nella coppia (Dipart, Ufficio).

In questo esempio, i gruppi sono 7.

Infatti le coppie differenti sono 7, perché c’è un’unica coppia (Dipart, Ufficio) ripetuta, che ha i valori (‘Produzione’, 20).

Nota 2

In presenza di un GROUP BY, nel SELECT possiamo inserire soltanto:

~ Operatori di aggregazione
~ Attributi che compaiono nel GROUP BY.

Nota 3

L'espressione sulla quale vogliamo applicare un operatore di aggregazione non può essere ottenuta (a sua volta) mediante un operatore di aggregazione.

Quindi, è proibito annidare gli operatori di aggregazione.

Ec: Considerate il db di riferimento.

Ogni dipartimento spende una certa somma in stipendi.

Trovate il massimo tra tutte queste spese dei dipartimenti.

Supponiamo che il db contenga i valori:

IMPIEGATI

MatriImpie	Stipendio	NomeDipa
1	50	SW
2	80	SW
3	105	HW

I valori suddetti indicano che la spesa massima in stipendi del singolo dipartimento è 130 (che è speso dal dipartimento 'SW').

La soluzione seguente è sintatticamente errata.

```
SELECT MAX(SUM(I.Stipendio)) /* Errore di sintassi */
FROM IMPIEGATI AS I
GROUP BY I.NomeDipa
```

Non possiamo risolvere questo problema usando soltanto le istruzioni viste finora. Lo potremo risolvere mediante i select annidati (che vedremo più avanti).

CLAUSOLA HAVING / ATZE.4.3.4

L'HAVING seleziona (mediante un opportuno criterio) i gruppi ottenuti mediante il GROUP BY.

In altre parole, elimina i gruppi che non soddisfano il criterio suddetto.

CLAUSOLA ORDER BY / ATZE.4.3.2

Adesso che abbiamo visto tutte le sei clausole, riassumo l'ordine con cui dobbiamo posizionarle e l'ordine con cui l'Sql le elabora.

Posizione sintattica	Ordine di elaborazione	Clausola
1	5	`SELECT`
2	1	`FROM`
3	2	`WHERE`
4	3	`GROUP BY`
5	4	`HAVING`
6	6	`ORDER BY`

L'`ORDER BY` si usa quando:
~ È richiesto esplicitamente un ordine nel risultato.
~ È opportuna un'esposizione non mescolata dei dati.

Non bisogna abusare dell'`ORDER BY`, perché questo può aumentare il tempo di esecuzione.

Ordinamento / Esercizi risolti

Uso di un intero positivo come argomento di ORDER BY

1. Considerate il db di riferimento.

Elencate la matricola e lo stipendio di ogni impiegato.

Ordinate l'elenco rispetto allo stipendio.

Soluzione

```
SELECT MatriImpie, Stipendio
FROM IMPIEGATI
ORDER BY 2;
```

Uso di un alias come argomento di ORDER BY

2. Considerate il db di riferimento.

Elencate, per ogni dipartimento, la sua spesa in stipendi.

Ordinate l'elenco rispetto alla somma spesa per gli stipendi.

Soluzione

```
SELECT NomeDipa, SUM(Stipendio) AS SommaStipendi
FROM IMPIEGATI
GROUP BY NomeDipa
ORDER BY SommaStipendi;
```

Uso di ORDER BY senza richiesta esplicita di ordinamento

3. Considerate il db di riferimento.

Elencate, per ogni dipartimento, i dipendenti che ci lavorano e il loro stipendio.

Soluzione

In questo caso è opportuno ordinare rispetto ai dipartimenti.

```
SELECT NomeDipa, MatriImpie, Stipendio
FROM IMPIEGATI
WHERE NomeDipa IS NOT NULL
ORDER BY NomeDipa;
```

Senza `ORDER BY` avremmo una visualizzazione disordinata, come la seguente:

NomeDipa	MatriImpie	Stipendio
SW	1	50
SW	2	80
HW	3	15
SW	5	20
SW	7	90

OPERAZIONI INSIEMISTICHE / ATZE.4.3.5

14 / 11 / 2024

CLAUSOLA HAVING / ATZE.4.3.4

Esempio di HAVING senza GROUP BY

Indicate la capienza media dei tipi di aerei, soltanto se questa capienza media è < 100.

```
/* HAVING SENZA GROUP BY. */
SELECT AVG(Capienza)
FROM TIPI_AEREI
HAVING AVG(Capienza) < 100;
```

SOLUZIONE DEGLI ESERCIZI PER CASA

SOLU.Aeroporti.H

Per ogni città italiana, indicate il numero di voli internazionali in partenza.

L'elenco deve contenere soltanto le città in cui questo numero è > 0.

Soluzione

Come in altri esercizi precedenti, per ogni volo devo conoscere lo stato di partenza e quello di arrivo. Quindi (come già visto), mi servono questi due JOIN:

```
(VOLI AS V JOIN CITTA AS C_P ON V.NomeCittaParte = C_P.NomeCitta)
  JOIN CITTA AS C_A ON V.NomeCittaArri = C_A.NomeCitta
```

Mi interessano soltanto alcuni voli.

Quindi, mi serve il WHERE con questi confronti:

```
WHERE C_P.Stato = 'ITA'
  AND C_A.Stato <> 'ITA'
```

Adesso, mi serve l'operatore COUNT per contare il numero dei voli "superstiti"; però, non lo devo applicare a tutta la tabella risultante, ma a ogni singola città (superstite). Quindi, mi serve il GROUP BY su ogni città.

NB: La frase "L'elenco deve contenere soltanto le città in cui questo numero e` > 0." è ridondante, perché le città che hanno 0 voli sono state eliminate precedentemente dal WHERE.

SOLU.Aeroporti.I

Elencate le città francesi da cui partono almeno venti voli alla settimana diretti in ITA.

Soluzione

Il FROM è identico a quello dell'esercizio precedente, per il medesimo motivo. Mi interessano soltanto alcuni voli: quelli che partono da una città francese e arrivano in una italiana.

Poi, mi serve il WHERE con questi confronti:

```
WHERE C_P.Stato = 'FRA'
```

```
AND C_A.Stato = 'ITA'
```

Adesso, mi serve l'operatore COUNT per contare il numero dei voli "superstiti", per sapere se questo numero è >= 20; però, non lo devo applicare a tutta la tabella risultante, ma a ogni singola città (superstite). Quindi, mi serve il GROUP BY su ogni città.

Adesso, devo considerare soltanto alcuni dei gruppi, cioè quelli aventi un numero di voli >= 2. Quindi, mi serve l'HAVING sul conteggio dei voli.

NB: Capisco che mi serve l'HAVING perché la condizione richiesta confronta un valore (in questo caso, il numero dei voli) che ottengo mediante un operatore di aggregazione. Quindi, adesso devo selezionare alcuni gruppi, non alcune righe.

SOLU.Aeroporti.J

Elencate le città da cui partono voli diretti a Milano, ordinate alfabeticamente.

Soluzione

In questo caso, l'unica novità è l'uso dell'ORDER BY fatto sulle città di partenza.

Notate che i JOIN non sono necessari perché non dobbiamo gestire né gli stati di partenza, né quelli di arrivo.

OPERAZIONI INSIEMISTICHE (CONTINUAZIONE) / ATZE.4.3.5

Alcune versioni di Sql non possiedono gli operatori `INTERSECT` e `EXCEPT`. Quindi, quando questi operatori non esistono, dovremo progettare le operazioni suddette mediante un select annidato (che vedremo tra poco). Poiché l'uso di questi operatori aumenta la leggibilità del software, io e voi li useremo opportunamente.

Naturalmente, useremo anche l'operatore `UNION`, che esiste in qualsiasi versione di Sql e che non è sostituibile da altre operazioni.

Operazioni insiemistici / Esercizi risolti

NB Inserite un commento per ogni sottoproblema nel quale avete scomposto il problema iniziale.

1. Considerate il db di riferimento.

Elencate le matricole dei capi i cui impiegati guadagnano tutti più di 40, senza usare operatori di aggregazione.

Es: L'input seguente:

MatriImpie	Stipendio	MatriImpieCapo
10	50	1
20	30	1
30	60	2
44	70	2

genera l'output seguente:

MatriImpieCapo
2

Soluzione

La consegna equivale a:

```
Matricole dei capi
EXCEPT
Matricole dei capi di (almeno) un impiegato che guadagna al massimo 40
```

Codifico ognuno dei due sottoproblemi precedenti:

```
/* Matricole dei capi */
SELECT MatriImpieCapo
FROM IMPIEGATI
  EXCEPT
/* Matricole dei capi di (almeno) un impiegato che guadagna al massimo 40 */
SELECT MatriImpieCapo
FROM IMPIEGATI
WHERE Stipendio <= 40;
```

JOIN COMPLETO E INCOMPLETO

Il **risultato** di un join effettuato su due tabelle `T1` e `T2` è detto **completo** se ogni riga di `T1` e ogni riga di `T2` compaiono nel risultato del join; altrimenti, è detto **incompleto**.

NB Il join completo o quello incompleto non sono operazioni differenti. Invece, sono esiti differenti della medesima operazione di join.

Es Considerate il db di riferimento.

Per ogni dipendente, elencate la sua matricola e la città in cui lavora.

L'output richiesto è qualcosa del tipo:

MatriImpie	Citta
1	PD
2	PD
3	MI
5	PD
7	PD

Notate che i dipendenti non assegnati a un dipartimento non compaiono nel risultato. Quindi, questo join è incompleto.

Come potete individuare facilmente, la soluzione è:

```
SELECT I.MatriImpie, D.Citta
FROM DIPARTIMENTI AS D JOIN IMPIEGATI AS I ON D.NomeDipa = I.NomeDipa;
```

NB: Quando il join è completo, si ha:
Cardinalità del join completo >= max {cardinalità di T1, cardinalità di T2}.

JOIN ESTERNO

Sono date:

~ Due tabelle `T1`, `T2`.

~ Una condizione `C` come nel join interno.

Join esterno sinistro

Il **join esterno sinistro** sugli attributi `X1` di `T1` e gli attributi `X2` di `T2` è l'unione tra

~ il join interno (eseguito mediante la condizione `C`)

e

~ la concatenazione tra

  ~ ogni riga di `T1` che non compare nel suddetto join interno

  e

  ~ i valori NULL negli attributi di `T2`.

Quindi, il join interno sinistro contiene sempre ogni riga di `T1`; infatti, contiene anche le righe di `T1` che non compaiono nel join interno).

Join esterno destro

Il join esterno destro è analogo a quello sinistro; l'unica differenza è lo scambio dei ruoli tra le tabelle `T1` e `T2`. La definizione formale è la seguente.

Il **join esterno destro** sugli attributi `X1` di `T1` e gli attributi `X2` di `T2` è l'unione tra

~ il join interno (eseguito mediante la condizione `C`)

e

~ la concatenazione tra

  ~ ogni riga di `T2` che non compare nel suddetto join interno

  e

  ~ i valori NULL negli attributi di `T1`.

Quindi, il join interno destro contiene sempre ogni riga di `T2`; infatti, contiene anche le righe di `T2` che non compaiono nel join interno.

Join esterno completo

Il **join esterno completo** sugli attributi `X1` di `T1` e gli attributi `X2` di `T2` è l'unione tra

~ il join esterno sinistro (eseguito mediante la condizione `C`)

e

~ il join esterno destro (eseguito mediante la condizione `C`).

Quindi, il join esterno completo contiene sempre ogni riga di `T1` e ogni riga di `T2`; infatti, contiene anche le righe di `T1` e quelle di `T2` che non compaiono nel join interno.

Inoltre, non contiene due righe uguali (perché, per default, l'operazione di unione le elimina).

Join esterno / Esercizi risolti

1. Considerate il db di riferimento.

Per ogni impiegato, elencate la sua matricola e l'eventuale città in cui lavora.

L'output richiesto è qualcosa del tipo:

MatriImpie	Citta
1	PD
2	PD
3	MI
4	NULL
5	PD
6	NULL
7	PD

Soluzione

Notate la parola "eventuale". Questa parola implica che dobbiamo visualizzare la matricola anche di un dipendente del quale non è nota la città in cui lavora.

Quindi, dobbiamo usare un join esterno:

```
SELECT I.MatriImpie, D.Citta
FROM DIPARTIMENTI AS D RIGHT JOIN IMPIEGATI AS I ON D.NomeDipa = I.NomeDipa
```

2. Considerate il db di riferimento.

Per ogni dipartimento, elencate il suo nome e le matricola degli eventuali impiegati che lavorano nel dipartimento.

L'output richiesto è qualcosa del tipo:

NomeDipa	MatriImpie
CE	NULL
HW	3
SW	1
SW	2
SW	5
SW	7

Soluzione

```
SELECT D.NomeDipa, I.MatriImpie
FROM DIPARTIMENTI AS D LEFT JOIN IMPIEGATI AS I ON D.NomeDipa = I.NomeDipa
ORDER BY D.NomeDipa;
```

2B. Risolvete l'esercizio precedente senza usare il join esterno.

Soluzione

La consegna equivale a elencare:

1: Per ogni dipartimento, il suo nome e le matricole degli impiegati che lavorano nel dipartimento

UNION

2: Ogni dipartimento senza impiegati

La consegna 2, a sua volta, equivale a elencare:

2.1: Ogni dipartimento

EXCEPT

2.2: Ogni dipartimento in cui lavora qualche impiegato

Risolvo i punti 1, 2.1, 2.2 in Sql, e, contemporaneamente, inserisco gli opportuni operatori insiemistici.

```
/*1: Per ogni dipartimento: nome e le matricole degli impiegati che vi lavorano */
(SELECT D.NomeDipa, I.MatriImpie
FROM DIPARTIMENTI AS D LEFT JOIN IMPIEGATI AS I ON D.NomeDipa = I.NomeDipa
ORDER BY D.NomeDipa
) UNION (
/* 2: Dipartimenti senza impiegati */
/* 2.1: Dipartimenti */
SELECT NomeDipa, NULL
FROM DIPARTIMENTI
) EXCEPT (
/* 2.2: Dipartimenti in cui qualche impiegato lavora */
SELECT NomeDipa, NULL
FROM IMPIEGATI
);
```

SELECT ANNIDATI

Un'interrogazione (o query) contiene:

~ 1 select;

oppure

~ 2 o più select.

Nel caso di due (o più) select, ci sono due situazioni:

~ I due select si "trovano allo stesso livello".
Allora, Sql risolve separatamente ogni select e poi combina i due risultati mediante un operatore insiemistico (`UNION`, `INTERSECT`, `EXCEPT`).

oppure

~ Un select è annidato (cioè, è interno) a un altro.

In questo sottocaso, ci sono due ulteriori situazioni:

~ Il select interno è indipendente da quello esterno.

oppure

~ Il select interno è collegato a quello esterno.

Un select è **annidato** quando è composto da un select esterno e da (almeno) un select interno.

Risultato

Un select annidato confronta (mediante un operatore specificato dal programmatore)

~ alcuni valori individuati dal select interno

con

~ ogni valore elencato del select esterno.

Poiché c'è un confronto, dobbiamo inserirlo in una clausola (del select esterno) che lo consente; ciò si fa, specificatamente, gestendo il confronto:

~ nel `WHERE`, quando dobbiamo confrontare ogni riga (del select esterno);

oppure

~ nell'`HAVING`, quando dobbiamo confrontare ogni gruppo (del select esterno).

Di conseguenza, il select annidato visualizza ogni riga (oppure ogni gruppo) del select esterno che soddisfa il confronto.

Es:

```
SELECT T1.X, COUNT(T1.K)
FROM T1
GROUP BY T1.X
HAVING COUNT(T1.K) < (
    SELECT COUNT(T1.K)
    FROM T1
    WHERE T1.X = 13
);
```

Input

Supponiamo che T1 contenga:

T1

K	X
A	11
C	12
D	13
E	13

Esecuzione della traccia

Il select interno individua:

COUNT(K)
2

Il select esterno "diventa":

```
SELECT T1.X, COUNT(T1.K)
FROM T1
GROUP BY T1.X
HAVING COUNT(T1.K) < 2;
```

Risultato finale:

X	COUNT(K)
11	1
12	1

21 / 11 / 2024

QUESTIONARIO INTERMEDIO SULL'ORGANIZZAZIONE E L'EFFICACIA DELLA DIDATTICA IN PRESENZA

CLAUSOLA GROUP BY (approfondimento)

Nota aggiuntiva

Se il GROUP BY è effettuato su un attributo che contiene il valore NULL, tutte le righe aventi questo valore confluiscono in un unico gruppo, che sarà visualizzato.

Ec1: Considerate il db di riferimento.

Per ogni dipartimento, visualizzate la sua spesa in stipendi.

Visualizzate anche la somma degli stipendi degli impiegati che non appartengono a nessun dipartimento.

Soluzione

```
SELECT NomeDipa, SUM(Stipendio) AS SommaStipendi
FROM IMPIEGATI
GROUP BY NomeDipa;
```

La query suddetta applicata sull'input di riferimento visualizza:

NomeDipa	SommaStipendi
	150
HW	15
SW	240

SOLUZIONE DEGLI ESERCIZI PER CASA

SOLU.Aeroporti.K

Elencate ogni città che è collegata a Roma (mediante un volo che parte da Roma o ci arriva).

L'elenco deve avere un'unica colonna.

SOLU.Aeroporti.M

Per ogni volo che parte da Milano, elencate la città di arrivo e, se sono disponibili, il tipo e la capienza dell'aereo.

SOLU.Azienda.B

Per ogni dipartimento, indicate la sua città e l'eventuale stipendio del suo cassiere.

SOLU.Aeroporti.L

Elencate le città collegate soltanto mediante voli effettuati la domenica.

(Quindi, dovete escludere le città che non sono collegate da nessun volo e le città che sono collegate in almeno un giorno differente dalla domenica.)

L'elenco deve avere un'unica colonna.

Non usate join.

SELECT ANNIDATI

Sintesi di quanto visto finora

Rivediamo il capitolo "SELECT ANNIDATI" della lezione precedente.

Indicazioni sulla sintassi in presenza di operatori di confronto

Tipicamente, un select annidato confronta le righe individuate dal select interno con ogni riga o gruppo del select esterno.

C'è un'eccezione all'affermazione precedente, cioè un select annidato può essere usato (sempre per selezionare) anche senza fare confronti tra valori dei due select. Lo vedremo più avanti.

Consideriamo adesso soltanto il caso più frequente, cioè quello in cui c'è un confronto (gestito mediante un operatore).

Nota 1

La clausola `SELECT` deve avere un unico argomento.

Controesempio

Vediamo cosa potrebbe succedere se il select interno potesse consentire due o più argomenti.

1) Supponiamo che il select interno abbia due argomenti, cioè che generi due valori per ogni riga; chiamo `Inte1` e `Inte2` questi due valori.

2) Supponiamo che il select interno generi queste due righe:

Inte1	Inte2
4	8
2	10

Naturalmente, affinché un confronto tra i valori generati dal select esterno e quelli generati dal select interno sia possibile, occorre che anche la clausola che effettuerà il confronto (WHERE oppure HAVING) generi due valori per ogni riga o per ogni gruppo; chiamo `Este1` e `Este2` questi due valori.

3) Supponiamo che voglia selezionare soltanto le righe del select esterno per cui il confronto:

```
(Este1, Este2) > (Inte1, Inte1)
```

è soddisfatto almeno una volta.

Quindi, dovremo usare la clausola WHERE, perché dobbiamo selezionare alcune righe.

4) Supponiamo, infine, che il select esterno generi questa riga:

Este1	Este2
3	7

Conclusioni

Ora, proviamo a verificare se la riga del punto 4) soddisfa il confronto del punto 3).

Qual è il significato di questo confronto?

Con un po' di "sforzo", potremmo dire che il confronto tra la riga suddetta e la prima riga del select interno vale false:

```
(3, 7) > (4, 8)
```

Ma cosa potremmo dire quando confrontiamo tra la riga suddetta e la seconda riga del select interno?:

```
(3, 7) > (2, 10)
```

Non siamo in grado di rispondere, proprio perché stiamo confrontando due coppie di valori, anziché due valori.

In altre parole, l'operazione di confronto non può essere definita; e questa impossibilità è causata proprio dal fatto che il select interno genera righe composte da due colonne (cioè, da due attributi), anziché da una.

28 / 11 / 2024

Indicazioni sulla sintassi in presenza di operatori di confronto

Nota 1

Ripasso dalla lezione precedente.

Nota 2

In un select annidato confrontiamo due valori. (Uno proviene dal select esterno e l'altro proviene da quello interno.)

In ogni singolo passo, questi due valori:

~ devono appartenere a domini compatibili (es: un valore integer e un valore decimal);

e inoltre

~ devono possedere l'operatore di confronto usato (es: l'operatore <).

Nota 3

In una clausola di confronto (che può essere un `WHERE` oppure un `HAVING`) del select esterno, possiamo inserire due o più confronti, combinandoli mediante i consueti operatori logici.

Es:

```
/* Select Esterno */
..
WHERE Este1 = (
    SELECT Inte1
    ..
    )
  AND Este2 = (
    SELECT Inte2
    ..
    )
/* Eventuale continuazione del Select esterno. */;
```

Select interno indipendente

Un select interno è **indipendente** quando usa soltanto tabelle indicate nel proprio `FROM`.

Quindi:

~ Il risultato del select interno non dipende dalle righe del select esterno.

~ Sql esegue il select interno soltanto una volta, prima del select esterno.

Select interno collegato

Un select interno è **collegato** quando non è indipendente, cioè quando usa almeno una tabella che non compare nel `FROM` del select interno. Quindi:

~ Il risultato del select interno dipende dalla particolare riga del select esterno che sta per essere confrontata con il select interno.

~ Sql esegue il select interno per ogni riga del select esterno.

Es:

Il select interno dell'esempio seguente è collegato perché usa (nella clausola `WHERE`) la tabella `T1` che non compare nel `FROM` del select interno.

```
SELECT T1.K, T1.X
FROM T1
WHERE T1.X = (
    SELECT MAX(T2.X)
    FROM T2
    WHERE T1.Y = T2.Y
);
```

Vedremo il suo effetto più avanti.

Note importanti

Riepilogo le note più importanti tra quelle che ho descritto precedentemente.

1 Un select interno serve a selezionare qualcosa del select esterno.
Quindi deve essere inserito nel `WHERE` o nell'`HAVING` del select esterno.

2 La selezione suddetta avviene mediante un operatore di confronto applicato tra:

~ l'(unica) espressione che si trova nel `WHERE` [o nell'`HAVING`] del select esterno

e

~ l'(unica) espressione che si trova nella clausola `SELECT` del select interno.

Quindi, la clausola `SELECT` del select interno deve avere un unico argomento.

3 Per ogni iterazione del select esterno,
La riga [o il gruppo], composta da una colonna, individuata dal suo `WHERE` [o dal suo `HAVING`]
deve essere confrontata con
la tabella individuata dal select interno, composta da una colonna e da zero, una o più righe.

4 Ci sono due tipologie di select interno:

~ Indipendente,

~ Collegato.

Sintassi e risultato di ogni modalità di confronto

Ci sono tre modalità per confrontare l'espressione del select interno con quella del select esterno.

Modalità 1

Questa modalità può essere usata soltanto se il select interno individua al massimo una riga; altrimenti, Sql darà un errore in esecuzione.

Quindi, per essere certi che l'interrogazione funzioni sempre (cioè, con qualsiasi input), dovremo progettare adeguatamente il select interno.

Sintassi:

```
..
[WHERE | HAVING] EspreEste OperatoreDiConfronto (
    SELECT EspreInte
    ..
);
```

Risultato

Il risultato contiene ogni riga [o gruppo] del select esterno che soddisfa il confronto seguente:

```
    EspreEste OperatoreDiConfronto EspreInte
```

Es.1

```
/* Select Esterno */
..
WHERE T1.X >= ( /* Il select interno individua sicuramente al massimo una riga
    se T2.K e` superchiave. */
    SELECT T2.X
    FROM T2
    WHERE T2.K = 5
);
```

Es.2

```
/* Select Esterno */
..
HAVING COUNT(T1.X) >= ( /* Il select interno individua sempre un'unica riga. */
    SELECT COUNT(T2.Y)
    ..
);
```

Esempio importante

Consideriamo una tabella `T1(K, X)`, dove `X` appartiene a un dominio che possiede l'operatore `>`. (Ad es., `X` è un numero.)

Dobbiamo trovare i valori di `K` che corrispondono al massimo di `X`. Questo è il cosiddetto problema della ricerca dei **punti di massimo**.

Risolviamo il problema mediante il seguente select annidato, dove il select interno è indipendente.

```
SELECT T1.K, T1.X
FROM T1
WHERE T1.X = (
    SELECT MAX(T1.X)
    FROM T1
);
```

SOLU.Aeroporti.N

Elencate i tipi di aereo aventi la massima capienza.

SOLU.Aeroporti.O

Elencate le città che sono servite dai tipi di aereo aventi la massima capienza.

L'elenco deve avere un'unica colonna.

Sintassi e risultato di ogni modalità di confronto

Modalità 2

Questa modalità può essere usata qualunque sia il numero (anche zero) di righe individuate dal select interno.

Sintassi

Tra l'espressione del select esterno e l'inizio del select interno deve dobbiamo inserire una a scelta di queste quattro forme alternative:

```
OperatoreDiConfronto ANY
OperatoreDiConfronto ALL
IN
NOT IN
```

Quindi, il select annidato si presenta in questa forma:

```
/* Select esterno */
..
[WHERE | HAVING] EspreEste UnaDelleQuattroFormeSuddette (
    SELECT EspreInte
    ..
);
```

Risultato

Una riga [o un gruppo] del select esterno compare nel risultato a seconda della parola chiave indicata:

~ con `ANY`:
la riga compare se `EspreEste` soddisfa il confronto con almeno un valore del select interno.

~ con `ALL`:
la riga compare se `EspreEste` soddisfa il confronto con ogni valore del select interno.

~ con `IN`:
la riga compare se `EspreEste` è tra i valori del select interno.
(Quindi, `IN` equivale a `= ANY`).

~ con `NOT IN`:
la riga compare se `EspreEste` non è tra i valori del select interno.
(Quindi, `NOT IN` equivale a `<> ALL`).

Es:

```
/* Select esterno */
..
WHERE T1.X >= ANY (
    SELECT T2.X
    FROM T2
);
```

5 / 12 / 2024

Riepilogo / Situazioni in cui richiedo un commento

Commentate ogni:
~ Select interno.
~ Select che sarà combinato mediante un operatore insiemistico.
~ Tabella che state rinominando perché si trova in un self join.
~ Parametro di un select.
~ Vista.

NB Un select esterno non necessita di commento perché coincide con la consegna iniziale.

Riepilogo / Modalità 1

Questa modalità può essere usata soltanto se il select interno individua al massimo una riga; altrimenti, Sql darà un errore in esecuzione.

Quindi, per essere certi che l'interrogazione funzioni sempre (cioè, con qualsiasi input), dovremo progettare adeguatamente il select interno.

Riepilogo / Modalità 2

Consideriamo la query ATZE.35:

```
SELECT Nome
FROM IMPIEGATI
  INTERSECT
SELECT Cognome
FROM IMPIEGATI;
```

Possiamo sostituire l'`INTERSECT` con l'`IN`:

```
SELECT Nome
FROM IMPIEGATI
WHERE Nome IN ( /* Ogni cognome di impiegati. */
    SELECT Cognome
    FROM IMPIEGATI
);
```

Notate che otteniamo una soluzione equivalente sostituendo:
~ l'operatore `>= ALL` (dentro il select esterno)
con
~ l'operatore `MAX` (dentro il select interno).

SOLU.Aeroporti.P

Elencate le città italiane che hanno la massima "ricezione".

La **ricezione** di una città si ottiene sommando la capienza dei voli che arrivano in quella città.

L'elenco deve avere un'unica colonna.

Testate la soluzione usando questi dati:

TIPI_AEREI

TipoAereo	Capienza
A	100
B	70
C	70
D	200

VOLI

NomeCittaParte	NomeCittaArri	TipoAereo
Milano	Paris	D
Milano	Roma	B
Milano	Roma	C
Milano	Venezia	B
Milano	Venezia	B
Roma	Milano	A

I dati suddetti individuano le ricezioni seguenti:

```
Ricezione(Roma)         = 70 + 70   = 140
Ricezione(Milano)                   = 100
Ricezione(Paris)                    = 200
Ricezione(Venezia)      = 70 + 70   = 140
```

Quindi, l'output corrispondente è:

Roma
Venezia

Ricordo che è sintatticamente errato l'annidamento di due operatori di aggregazione.

a) Risolvete mediante operatori insiemistici.
   Suggerimento:
   1) Risolvete prima (erroneamente) annidando due operatori di aggregazione.
   2) Risolvete la consegna finale sostituendo adeguatamente uno dei due operatori che avete inserito nella vostra soluzione del passo 1).

Modalità 3

Questa modalità, a differenza delle due precedenti, non ha il compito di confrontare un valore del select esterno con i valori di quello interno.

Più semplicemente, deve soltanto valutare se il select interno individua una tabella vuota.

Sintassi:

```
/* Select esterno */
..
WHERE [NOT] EXISTS (
    SELECT *
    ..
);
```

Risultato:

~ In presenza di `NOT EXISTS`:
La riga del select esterno compare nel risultato se non esiste nessuna riga nel select interno, cioè se questo individua un tabella vuota.

~ In presenza di `EXISTS`:
La riga del select esterno compare nel risultato se il select interno individua almeno una riga.

NB Nella clausola `SELECT` del select interno ci conviene mettere sempre l'`*` (invece di un attributo o di un'espressione) perché ci interessa soltanto sapere se questo select individua almeno una riga; quindi, i valori specifici della riga non ci interessano.

Nota sull'utilità di EXISTS

Se il select interno è indipendente, l'`EXISTS` non ha nessuna utilità perché il suo esito è identico su tutte le righe del select esterno (cioè, è sempre vero oppure sempre falso) e quindi non seleziona le righe del select esterno.

Es: Il seguente select interno è indipendente (perché usa una tabella definita nel proprio `FROM`).

Provo a confrontarlo (impropriamente) con quello esterno mediante un `EXISTS`.

```
SELECT T1.K, T1.X
FROM T1
WHERE EXISTS (
    SELECT T2.*
    FROM T2
    WHERE T2.X = 3
);
```

L'interrogazione suddetta ha il seguente effetto.

Se il select interno individua una tabella vuota (cioè, se `T2.X` di ogni riga è sempre `!= 3`),
Allora nessuna riga del select esterno compare nel risultato.

Se, invece, il select interno individua almeno una riga,
Allora ogni riga del select esterno compare nel risultato.

Quindi, l'`EXISTS` in questo caso è inutile, perché il select interno (e indipendente) non visualizza soltanto alcune righe del select esterno, ma le visualizza tutte o nessuna.

L'`EXISTS` è utile soltanto nei select collegati, come vedremo tra poco.

Select collegati

L'effetto di un select collegato è il seguente:

Per ogni riga `r` del select esterno:

Il select interno individua l'insieme `I(r)` di righe del select interno. (Quindi, `I(r)` dipende dalla riga `r`).

Se il confronto (indicato nel `WHERE` o nell'`HAVING`) tra `r` (del select esterno) e `I(r)` è soddisfatto,
La riga `r` compare nel risultato.

Esempio

Traccio l'interrogazione:

```
SELECT *
FROM T1
WHERE T1.A = (
    SELECT COUNT(*)
    FROM T2
    WHERE T1.B = T2.B
);
```

sui seguenti dati:

T1

A	B
3	12
2	14

T2

B	C
12	8
14	7
13	5
14	6

Esecuzione della traccia:

Select esterno			Select interno		
T1.A	T1.B		WHERE T1.B ..	T2.COUNT(*)	WHERE T1.A = (SELECT COUNT(*) ..
3	12		12	1	3 = 1 ? F
2	14		14	2	2 = 2 ? T

Risultato:

A	B
2	14

Useremo un select annidato collegato quando l'interrogazione deve visualizzare le righe che soddisfano un confronto tra:

~ un valore del select esterno, e
~ un insieme di righe che varia al variare della riga del select esterno.

Esercizio risolto / 1

Considerate il db di riferimento.

Elencate ogni matricola che ha lo stesso cognome di un'altra matricola.

Usate un select annidato e l'operatore `EXISTS`.

Soluzione

Abbiamo già risolto un problema simile mediante un self join.

Adesso lo risolviamo mediante un algoritmo del tipo:

Per ogni impiegato `i` del select esterno

Il select interno individua l'insieme `I_O(i)` composto da quegli impiegati omonimi (nel cognome) di `i` (del select esterno) e che hanno inoltre una matricola differente.

Se l'insieme `I_O(i)` contiene almeno un impiegato,
La matricola di `i` compare nel risultato.

Possiamo notare che l'insieme `I_O(i)` dipende dall'impiegato `i`. Ciò implica che:

~ Il select interno deve essere collegato.

~ Il select interno confronta il cognome della sua tabella `IMPIEGATI` con il cognome di `IMPIEGATI` del select esterno. Quindi, usa la tabella `IMPIEGATI` con due finalità differenti; per questo, dobbiamo rinominare `IMPIEGATI` nel select interno.

La query in Sql è:

```
SELECT I.MatriImpie
FROM IMPIEGATI AS I
WHERE EXISTS ( /* Ogni impiegato omonimo della matricola I.MatriImpie */
    SELECT *
    FROM IMPIEGATI AS I_O
    WHERE I.Cognome = I_O.Cognome
      AND I.MatriImpie <> I_O.MatriImpie
);
```

Esecuzione della traccia:

Uso questi valori:

MatriImpie	Cognome
1	A
2	A
3	B
4	A

Select esterno

I.MatriImpie	I.Cognome
1	A
2	A
3	B
4	A

Select interno

WHERE I.Cognome = I_O.Cognome AND I.MatriImpie <> I_O.MatriImpie	WHERE EXISTS
'A' = 'A' AND 1 <> 1 ? F	EXISTS({2, 4}) ? T
'A' = 'A' AND 1 <> 2 ? T	
'A' = 'B' AND 1 <> 3 ? F	
'A' = 'A' AND 1 <> 4 ? T	
È analogo a I.MatriImpie = 1	EXISTS({1, 4}) ? T
'B' = 'A' AND 3 <> 1 ? F	EXISTS({ }) ? F
'B' = 'A' AND 3 <> 2 ? F	
'B' = 'B' AND 31 <> 3 ? F	
'B' = 'A' AND 4 <> 4 ? F	
È analogo a I.MatriImpie = 1	EXISTS({1, 2}) ? T

L'interrogazione visualizzerà questa tabella:

I.MatriImpie
1
2
4

Esercizio risolto / 2

Considerate il db di riferimento.

Elencate, per ogni dipartimento:

~ la matricola con lo stipendio massimo nel dipartimento;

~ il valore di questo stipendio.

Ordinate l'elenco rispetto ai dipartimenti.

Soluzione

Per ogni impiegato (`m, d, s`) del select esterno

/* `m, d, s` sono rispettivamente la matricola, il dipartimento e lo stipendio dell'impiegato. */

Il select interno individua l'insieme `I_DIPA(d)` composto dagli impiegati che lavorano nel dipartimento `d`.

Se lo stipendio `s` (del select esterno) è uguale al massimo degli stipendi di `I_DIPA(d)`,

La matricola `m` (del select esterno) compare nel risultato.

12 / 12 / 2024

Esercizio risolto / 2 / Conclusione

Considerate il db di riferimento.

Elencate, per ogni dipartimento:

~ le matricole che hanno lo stipendio massimo nel dipartimento;

~ il valore di questo stipendio.

Ordinate l'elenco rispetto ai dipartimenti.

Soluzione

Per ogni impiegato (`m, d, s`) del select esterno

/* `m, d, s` sono rispettivamente la matricola, il dipartimento e lo stipendio dell'impiegato. */

Il select interno individua l'insieme `I_DIPA(d)` composto dagli impiegati che lavorano nel dipartimento `d`.

Se lo stipendio `s` (del select esterno) è uguale al massimo degli stipendi di `I_DIPA(d)`,

La matricola `m` (del select esterno) compare nel risultato.

Possiamo notare che l'insieme `I_DIPA(d)` dipende dal dipartimento `d`. Ciò implica che:

~ Il select interno deve essere collegato.

~ Il select interno usa la tabella `IMPIEGATI` con due finalità differenti.
Quindi, dobbiamo rinominare `IMPIEGATI` nel select interno.

La query in Sql è:

```
SELECT I.NomeDipa, I.MatriImpie, I.Stipendio
FROM IMPIEGATI AS I
WHERE I.Stipendio = ( /* Stipendio massimo tra quelli degli impiegati
    che lavorano nel dipartimento I.NomeDipa */
    SELECT MAX(I_DIPA.Stipendio)
    FROM IMPIEGATI AS I_DIPA
    WHERE I.NomeDipa = I_DIPA.NomeDipa
)
ORDER BY I.NomeDipa;
```

Questo esercizio è un esempio molto importante di esercizi aventi la tipologia seguente:

Per ogni sottoinsieme individuato al variare di un attributo `A`,

Trovate le righe che hanno il valore massimo nell'attributo `B`.

In questo caso, `A` è il dipartimento, le righe cercate sono gli impiegati, e `B` è lo stipendio.

Esecuzione della traccia:

Uso questi valori:

MatriImpie	Stipendio	NomeDipa
1	50	SW
3	15	HW
5	15	SW
6	90	SW
7	90	SW

Select esterno | Select interno

I.MatriImpie, I.Stipendio, I.NomeDipa	ON I.NomeDipa = I_DIPA.NomeDipa	I_DIPA.Stipendio	WHERE I.Stipendio = MAX(I_DIPA.Stipendio)
1, 50, SW	SW	50	50 = 90 ? F
		15	
		90	
		90	
3, 15, HW	HW	15	15 = 15 ? T
5, 15, SW	SW	50	15 = 90 ? F
		15	
		90	
		90	
6, 90, SW	SW	50	90 = 90 ? T
		15	
		90	
		90	
7, 90, SW	SW	50	90 = 90 ? T
		15	
		90	
		90	

L'interrogazione visualizzerà questa tabella:

I.NomeDipa	I.MatriImpie	I.Stipendio
HW	2	15
SW	6	90
SW	7	90

SOLU.ESE.3

3. È dato il seguente db:

CITTA(NomeCitta, NomeStato, NAbita)

Elencate, per ogni stato, la città più abitata e la sua popolazione.

Ordinate l'elenco rispetto allo stato.

Testate la soluzione usando questi dati:

NomeCitta	NomeStato	NAbita
A	X	300
B	X	200
C	Y	200
D	Y	100
E	Y	200

L'output corrispondente è:

NomeStato	NomeCitta	NAbita
X	A	300
Y	C	200
Y	E	200

```
/* Risolto da: Pietro Trabuio */
SELECT C.NomeStato, C.NomeCitta, C.NAbita
FROM CITTA AS C
WHERE C.NAbita = ( /* N. di abitanti della città più popolosa tra quelle che sono
    nello stesso stato in cui si trova la città C */
    SELECT MAX(CI_STA.NAbita)
    FROM CITTA AS CI_STA
    WHERE C.NomeStato = CI_STA.NomeStato
)
ORDER BY C.NomeStato;
```

VISTE

Descrizione

Le tabelle che abbiamo visto finora sono soltanto un caso particolare (e il più importante) tra le varie categorie di "tabelle nel senso più ampio del termine" che possiamo trovare in un DB.

Possiamo classificare le "tabelle nel senso più ampio del termine" mediante questo albero:

~ Tabelle "normali" (o tabelle)
Sono le tabelle che abbiamo visto finora.

~ Viste
Le viste sono tabelle che contengono soltanto dati ridondanti, che noi inseriamo mediante istruzioni opportune.
Più precisamente, le viste contengono soltanto

  ~ alcuni dei dati contenuti nelle tabelle "normali",
  oppure

  ~ dati ricavabili da altri dati contenuti nelle tabelle.

Le viste si distinguono nelle seguenti categorie, a seconda della modalità con cui il DBMS le gestisce:

~ Viste “normali” (o viste)
Sono le viste di default.
Rimangono nel db finché non le elimineremo esplicitamente mediante il comando
`DROP NomeVista`.

~ Viste temporanee
Saranno eliminate quando chiuderemo la sessione di accesso al db.

Sintassi / ATZE.5.1.3

Considerare le definizioni delle viste contenute nell’esempio suddetto di Atze.

Dopo averle definite, potrò usarle per definire query come se le viste fossero tabelle.

Ad es. posso definire le query seguenti:

```
SELECT *
FROM IMPIEGATIAMMIN;
```

e

```
SELECT *
FROM IMPIEGATIAMMINPOVER;
```

Esempi

Osservate i file `Prove4_...sql` e leggete i commenti.
Troverete esempi di definizioni di viste sia “normali”, sia temporanee.

Vantaggi delle viste

Ci conviene usare una vista in questi casi:

1 Vogliamo definire soltanto una volta un'espressione che dovremo riusare in più interrogazioni.

In questo caso, definiremo una vista che calcola questa espressione; poi la riuseremo nelle varie interrogazioni.

2a Vogliamo scomporre un’interrogazione complicata.

2b Vogliamo scomporre “un’interrogazione non risolubile direttamente” in due o più sottointerrogazioni più semplici.

In questi due sottocasi (2a e 2b), definiremo una vista che risolve una delle sottointerrogazioni più semplici; poi la useremo, insieme ad altre interrogazioni, per risolvere il problema iniziale.

3 Vogliamo nascondere a un utente alcuni dati che non gli sono necessari.

In questo caso, definiremo una vista che contiene soltanto i dati che vogliamo rendere visibili.
Poi, consentiremo all’utente di accedere a questa vista (e non alla tabella originale).

4 Abbiamo ristrutturato il DB (e quindi stiamo usando tabelle che sono differenti da quelle del DB precedente). Inoltre vogliamo continuare a usare le interrogazioni che esistevano prima della ristrutturazione.
Per rendere compatibili le interrogazioni alla versione originaria, dovremmo disporre di ogni tabella presente nella versione originaria; ma alcune di queste tabelle non sono più disponibili a causa della ri-

strutturazione.

In questo caso, per ogni tabella non più disponibile, definiremo una vista “corrispondente” a questa tabella. Così, le interrogazioni precedentemente definite potranno funzionare accedendo a queste viste, anziché alle tabelle che esistevano prima della ristrutturazione (ma che adesso non esistono più).

Svantaggi delle viste

Il DBMS può gestire le viste in due modi differenti:

a) Può memorizzare i dati contenuti nella vista.

oppure

b) Invece di memorizzare i dati, può inserirli nella vista immediatamente prima di ogni volta che un’interrogazione deve usare questa vista.

In entrambi i casi c'è uno svantaggio.

a) Se il DBMS memorizza la vista,
Il DBMS deve aggiornare la vista ogni volta che l'utente modifica qualche dato da cui la vista dipende.

Quindi, questa situazione richiede un’elaborazione aggiuntiva ogni volta che modifichiamo qualche tabella dalla quale la vista dipende.

b) Se il DBMS inserisce i dati della vista quando deve usarla,
Il DBMS elaborare la vista (e anche che per elaborare l'interrogazione).

Quindi, questa situazione richiede un’elaborazione aggiuntiva ogni volta che usiamo la vista.

Esempi di applicazioni delle viste

Definizione di un'espressione che dovremo usare in più interrogazioni

Nel problema degli aeroporti, consideriamo le consegne E, F:

E: “Per il volo con identificativo 3, elencate il giorno della settimana, la nazione di partenza e quella di arrivo.”

F: “Elencate le coppie di città collegate da voli internazionali.”

Notate che entrambe le consegne richiedono di conoscere, per ogni volo, la nazione di partenza e quella di arrivo; quindi, le consegne suddette usano il medesimo FROM.

Allora. ci conviene definire una vista contenente ogni volo (con ogni suo attributo), al quale aggiungiamo i suoi stati di partenza e di arrivo.

Dopo, risolveremo le consegne E e F mediante la vista suddetta:

Consegna E

Abbiamo visto precedentemente questa soluzione:

```
SELECT V.GiornoSetti, C_P.Stato, C_A.Stato
FROM (VOLI AS V JOIN CITTA AS C_P ON V.NomeCittaParte = C_P.NomeCitta)
  JOIN CITTA AS C_A ON V.NomeCittaArri = C_A.NomeCitta
WHERE V.IdeVolo = 3;
```

Possiamo definire una seconda soluzione, che scomponiamo in due passi:

1 Definiamo una vista contenente ogni volo, al quale aggiungiamo la nazione di partenza e quella di arrivo.

```
CREATE VIEW VOLI_STATI (IdeVolo, GiornoSetti, NomeCittaParte, NomeCittaArri,
  TipoAereo, StatoParte, StatoArri) AS (
    SELECT V.*, C_P.Stato, C_A.Stato
    FROM (VOLI AS V JOIN CITTA AS C_P ON V.NomeCittaParte = C_P.NomeCitta)
      JOIN CITTA AS C_A ON V.NomeCittaArri = C_A.NomeCitta
);
```

2 Adesso, risolviamo la consegna mediante la vista suddetta:

```
SELECT GiornoSetti, NazioneParte, NazioneArri
FROM VOLI_STATI
WHERE IdeVolo = 3;
```

Consegna F

Abbiamo visto precedentemente questa soluzione:

```
SELECT DISTINCT V.NomeCittaParte, V.NomeCittaArri
FROM (VOLI AS V JOIN CITTA AS C_P ON V.NomeCittaParte = C_P.NomeCitta)
  JOIN CITTA AS C_A ON V.NomeCittaArri = C_A.NomeCitta
  AND C_P.Stato <> C_A.Stato;
```

Possiamo definire una seconda soluzione, che scomponiamo in due passi.

1 Questo passo è identico al passo 1 precedente.

2 Adesso, risolviamo la consegna mediante la vista suddetta:

```
SELECT DISTINCT NomeCittaParte, NomeCittaArri
FROM VOLI_STATI
WHERE StatoParte <> StatoArri;
```

Definizione di un'espressione che dovremo usare due volte nella stessa interrogazione

Nel problema degli aeroporti, la consegna P richiede: “Elencate le città italiane che hanno la massima "ricezione".

La ricezione di una città si ottiene sommando la capienza dei voli che arrivano in quella città.

L’elenco deve avere un'unica colonna.”

Abbiamo precedentemente visto questa soluzione:

```
SELECT C_A.NomeCitta AS Nome_Citta_Con_Ricezione_Max
FROM (VOLI AS V JOIN CITTA AS C_A ON V.NomeCittaArri = C_A.NomeCitta)
  JOIN TIPI_AEREI AS TA ON V.TipoAereo = TA.TipoAereo
WHERE C_A.Stato = 'ITA'
GROUP BY C_A.NomeCitta
HAVING SUM(TA.Capienza) >= ALL ( /* Ricezioni di ogni citta` italiana */
    SELECT SUM(TA.Capienza)
    FROM (VOLI AS V JOIN CITTA AS C_A ON V.NomeCittaArri = C_A.NomeCitta)
      JOIN TIPI_AEREI AS TA ON V.TipoAereo = TA.TipoAereo
    WHERE C_A.Stato = 'ITA'
    GROUP BY C_A.NomeCitta
);
```

Notate che la soluzione suddetta usa, nel select interno, istruzioni che sono molto simili a quelle che usa nel select esterno.

Possiamo definire una seconda soluzione, che scomponiamo in due passi:

1 Definiamo una vista contenente ogni città di arrivo e la sua ricezione.

```
CREATE VIEW CI_ITA_CAPIE(NomeCittaArriIta, Ricezione) AS (
    /* Per ogni citta` italiana di arrivo, la sua ricezione. */
    SELECT C_A.NomeCitta, SUM(TA.Capienza)
    FROM (VOLI AS V JOIN CITTA AS C_A ON V.NomeCittaArri = C_A.NomeCitta)
      JOIN TIPI_AEREI AS TA ON V.TipoAereo = TA.TipoAereo
    WHERE C_A.Stato = 'ITA'
    GROUP BY C_A.NomeCitta
);
```

2 Adesso, risolviamo la consegna mediante la vista suddetta:

```
SELECT NomeCittaArriIta AS Nome_Citta_Con_Ricezione_Max
FROM CI_ITA_CAPIE
WHERE Ricezione = ( /* Ricezione massima tra quelle delle citta` italiane */
    SELECT MAX(Ricezione)
    FROM CI_ITA_CAPIE
);
```

Esercizi

2. È dato il DB che gestisce la partecipazione delle squadre ai campionati della massima serie di un dato sport. Il DB è composto da:

SQUADRE(NomeSquadra, AnnoFonda)

PARTECIPAZIONI(NomeSquadra, AnnoCampio, NPunti)

Nessun campionato ha avuto due squadre arrivate prime con lo stesso numero di punti. (Cioè, nessun campionato si è concluso con uno spareggio.)

Trovate la/e squadra/e che ha vinto più campionati, mediante una vista (che dovete definire opportunamente).

Es: Con i seguenti valori, dovete individuare la squadra A, perché ha vinto il campionato due volte (nel 2000 e nel 2010); invece, la squadra B lo ha vinto soltanto una volta (nel 2020).

PARTECIPAZIONI

NomeSquadra	AnnoCampio	NPunti
A	2000	100
B	2000	80
A	2010	200
B	2010	100
A	2020	60
B	2020	80

Prima, definisco una vista che, per ogni anno, determina la squadra vincitrice.

```
/* Risolto da: Nicolò Fioranzato */
CREATE VIEW VINCITRICI(AnnoCampio, NomeSquadra) AS (
    SELECT P1.AnnoCampio, P1.NomeSquadra
    FROM PARTECIPAZIONI AS P1
    WHERE P1.NPunti = (
        /* Punti della squadra vincitrice del campionato P1.AnnoCampio
        SELECT MAX(P2.NPunt)
        FROM PARTECIPAZIONI AS P2
        WHERE P1.AnnoCampio = P2.AnnoCampio
    )
);
```

Dopo, trovo la squadra che ha vinto più campionati.

```
/* Risolto da: Nicolò Fioranzato */
SELECT V.NomeSquadra, COUNT(*)
    FROM VINCITRICI AS V
    GROUP BY V.NomeSquadra
    HAVING COUNT(*) >= ALL (
        # Per ogni squadra, elenca il numero di campionati vinti.
        SELECT COUNT(*)
        FROM VINCITRICI AS V
        GROUP BY V.NomeSquadra
    )
);
```