SQL - Structured Query Language

Lab 05

Alessandro Lori

Università di Pisa

27 Aprile 2012

Riepilogo esercitazione precedente

- Operatori insiemistici (**`UNION`**, **`INTERSECT`**, **`EXCEPT`**)

Riepilogo esercitazione precedente

- Operatori insiemistici (**`UNION`**, **`INTERSECT`**, **`EXCEPT`**)
- Query nidificate nel costrutto **`WHERE`**

Riepilogo esercitazione precedente

- Operatori insiemistici (**`UNION`**, **`INTERSECT`**, **`EXCEPT`**)
- Query nidificate nel costrutto **`WHERE`**
  - Costrutto **`IN`**, **`NOT IN`**

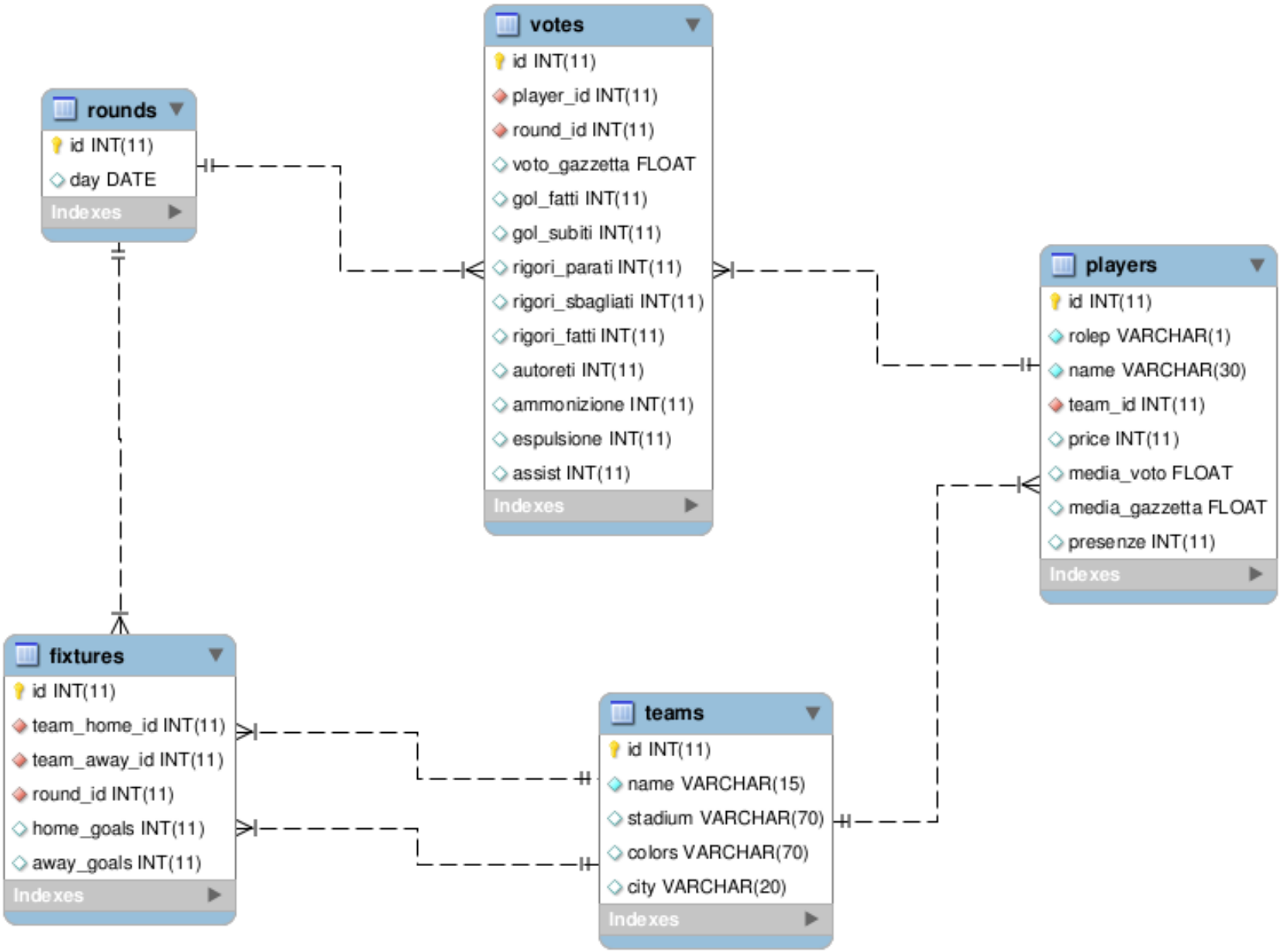
Riepilogo esercitazione precedente

- Operatori insiemistici (**`UNION`**, **`INTERSECT`**, **`EXCEPT`**)
- Query nidificate nel costrutto **`WHERE`**
  - Costrutto **`IN`**, **`NOT IN`**
  - Costrutto **`EXISTS`**, **`NOT EXISTS`**

Riepilogo esercitazione precedente

- Operatori insiemistici (**`UNION`**, **`INTERSECT`**, **`EXCEPT`**)
- Query nidificate nel costrutto **`WHERE`**
  - Costrutto **`IN`**, **`NOT IN`**
  - Costrutto **`EXISTS`**, **`NOT EXISTS`**
  - Costrutto **`ALL`**, **`ANY`**, **`SOME`**

Database seriea



Esercizio

Trovare il risultato dei derby (partite fra squadre della stessa città)

Esercizio

Trovare il risultato dei derby (partite fra squadre della stessa città)

Soluzione

```
SELECT home.name AS home, away.name AS away,
       home_goals, away_goals
FROM fixtures, teams AS home, teams AS away
WHERE home.id = team_home_id
AND away.id = team_away_id
AND home.city = away.city
```

Esercizio

Trovare i difensori del Catania che non hanno mai fatto goal.

Esercizio

Trovare i difensori del Catania che non hanno mai fatto goal.

Soluzione

```
SELECT p.name
FROM players p JOIN teams t ON t.id = p.team_id
WHERE role_p = 'D' AND t.name = 'Catania'
AND p.id NOT IN (SELECT player_id
FROM votes
WHERE gol_fatti <> 0
OR rigori_fatti <> 0)
```

Viste

Sintassi (semplificata)

```
CREATE VIEW NomeVista(NomeAttr1, NomeAttr2, ...)
AS (SELECT ... FROM ...)
```

- Una vista (o **view**) può essere pensata come una tabella di appoggio

Viste

Sintassi (semplificata)

```
CREATE VIEW NomeVista(NomeAttr1, NomeAttr2, ...)
AS (SELECT ... FROM ...)
```

- Una vista (o **view**) può essere pensata come una tabella di appoggio
- Essa viene generata e calcolata ogni volta che la si usa, a partire da una (o più) tabelle o da altre viste.

Viste

Sintassi (semplificata)

```
CREATE VIEW NomeVista(NomeAttr1, NomeAttr2, ...)
AS (SELECT ... FROM ...)
```

- Una vista (o **view**) può essere pensata come una tabella di appoggio
- Essa viene generata e calcolata ogni volta che la si usa, a partire da una (o più) tabelle o da altre viste.
- La lista di attributi è opzionale (corrisponde a effettuare delle ridenominazioni dei campi nella **SELECT**)

Esempio

Creare una vista con la lista dei giocatori titolari. Un giocatore si considera titolare se ha totalizzato più di 20 presenze

Soluzione

```
CREATE VIEW titolari(name, team) AS
(SELECT p.name, t.name FROM players p, teams t
 WHERE t.id = p.team_id
 AND presenze >20)
```

Esempio

Creare una vista con la lista dei giocatori titolari. Un giocatore si considera titolare se ha totalizzato più di 20 presenze

Soluzione

```
CREATE VIEW titolari(name, team) AS
(SELECT p.name, t.name FROM players p, teams t
 WHERE t.id = p.team_id
 AND presenze >20)
```

A questo punto posso riutilizzare la vista come se fosse un'altra tabella. Ad esempio:

```
SELECT team, COUNT(*) FROM titolari GROUP BY team
```

Esercizio

Creare una vista calendario che mostra tutte le partite del campionato, con l'indicazione del giorno in cui si svolge. La vista dovrà avere tre campi: giorno, nome della squadra di casa e nome della squadra in trasferta.

Esercizio

Creare una vista calendario che mostra tutte le partite del campionato, con l'indicazione del giorno in cui si svolge. La vista dovrà avere tre campi: giorno, nome della squadra di casa e nome della squadra in trasferta.

Soluzione

```
CREATE VIEW Calendario(day_r, home, away) AS
( SELECT day_r,h.name,a.name
  FROM rounds r, fixtures, teams h, teams a
  WHERE r.id = round_id AND team_home_id = h.id
  AND team_away_id = a.id)
```

Benefici delle viste

- Query complesse possono essere rese più semplici (e leggibili)
- Colonne calcolate
- Gestione della sicurezza (Viste in sola lettura, diversi livelli di privilegio)
- Possibilità di modificare il database rimanendo compatibili con applicazioni preesistenti

Esercizio

Elencare le squadre che hanno giocato almeno due partite subendo più di 4 ammonizioni

Vista di appoggio

```
CREATE VIEW
AmmPerPartita(round_id, team_id, num_amm) AS
( SELECT round_id, team_id, SUM(ammonizione)
  FROM votes, players p
  WHERE player_id = p.id
  GROUP BY round_id, team_id)
```

Questa vista calcola il numero di ammonizioni per partita. La partita è individuata dalla coppia round_id, team_id.

Soluzione

Squadre che hanno giocato almeno 2 partite subendo più di 4 ammonizioni

Soluzione

```
SELECT name FROM teams
WHERE id IN (SELECT team_id FROM AmmPerPartita
      WHERE num_amm > 4 GROUP BY team_id
      HAVING COUNT(*)>=3)
```

Soluzione

Squadre che hanno giocato almeno 2 partite subendo più di 4 ammonizioni

Soluzione

```
SELECT name FROM teams
WHERE id IN (SELECT team_id FROM AmmPerPartita
    WHERE num_amm > 4 GROUP BY team_id
    HAVING COUNT(*)>=3)
```

Query annidate nel from

```
SELECT name
FROM teams, (SELECT team_id FROM AmmPerPartita
  WHERE num_amm > 4 GROUP BY team_id
  HAVING COUNT(*)>=3) AS t
WHERE id=team_id
```

La ridenominazione **AS** t è obbligatoria quando si vuole utilizzare una query nel **FROM**.

Viste e modifica dei dati - Cenni

In alcuni casi le viste possono essere usate anche per inserimenti e modifica dei dati. Tuttavia ci deve essere una relazione 1-1 con fra le righe della vista e le righe della tabella sottostante. Alcuni dei requisiti per l'aggiornamento:

- La vista non deve contenere i seguenti costrutti
  - Operatori aggregati
  - **`DISTINCT`**
  - **`GROUP BY`**, **`HAVING`**
  - **`UNION`**
- La vista non deve usare viste non aggiornabili nel **`FROM`**
- La vista non deve contenere solo valori costanti
- Non deve avere più riferimenti a colonne della stessa tabella
- Sottoquery nel **`WHERE`** che si riferiscono a tabelle nel **`FROM`**

I requisiti per l'inserimento sono ancora più stringenti

Esercizio

- Si calcoli in quali mesi si è assegnato il numero massimo di rigori
- Il risultato dovrà essere del tipo (Mese, Numero di rigori assegnati)
- Suggerimenti
  - I rigori assegnati si possono ottenere sommando i rigori fatti con i rigori sbagliati
  - Per ottenere il mese da una data si scelga una funzione opportuna di manipolazione delle date

Esercizio

Si calcoli in quali mesi si è assegnato il numero massimo di rigori

Soluzione

```
CREATE VIEW RigoriMese(Mese, Rigori) AS
(SELECT DATE_FORMAT(day_r,"%M") AS Mese,
        SUM(rigori_fatti+rigori_sbagliati)
 FROM rounds r, votes
 WHERE round_id = r.id
 GROUP BY Mese)

SELECT Mese FROM RigoriMese
WHERE Rigori = (SELECT MAX(Rigori) FROM RigoriMese)
```

Creazione/Modifica di tabelle

- SQL permette non solo di interrogare le tabelle, ma anche di definire gli schemi e modificare i dati
- Il comando per creare le tabelle è **CREATE TABLE**
- Per avere degli esempi si veda come sono state create le tabelle della base di dati

```
CREATE TABLE `fixtures` (
  `id` int(11) NOT NULL auto_increment,
  `team_home_id` int(11) NOT NULL,
  `team_away_id` int(11) NOT NULL,
  `round_id` int(11) NOT NULL,
  `home_goals` int(11) default NULL,
  `away_goals` int(11) default NULL,
....
```

Chiavi esterne

```
CREATE TABLE `fixtures` (
...
  PRIMARY KEY (`id`),
  KEY `fk_1` (`team_home_id`),
  KEY `fk_2` (`team_away_id`),
  KEY `fk_3` (`round_id`),
  CONSTRAINT `fk_1` FOREIGN KEY (`team_home_id`)
  REFERENCES `teams` (`id`)
  ON DELETE NO ACTION ON UPDATE NO ACTION,
...
```

Indici

- Strutture dati che permettono di accedere ad alcuni record specifici di una tabella senza leggere sequenzialmente tutte le righe
- Velocizzano (a volte in maniera determinante)
  - La clausola **`WHERE`** se coinvolge indici
  - Il **`JOIN`** di tabelle se la condizione contiene indici (molto utile quindi avere come indici le chiavi esterne)
  - Ordinamento del risultato (**`ORDER BY`**)
- Svantaggi
  - Scritture più lente (può darsi che alcuni indici vadano aggiornati)
  - Maggiore occupazione di memoria (Conviene fare indici piccoli)

Analizzare una query costosa

- `MySQL` permette di capire quanto ci costa una query tramite il costrutto EXPLAIN
- Non fa parte dello standard ma anche altri vendor hanno funzionalità simili (ad esempio `Oracle` ha EXPLAIN PLAN)
- La sintassi è EXPLAIN (**`SELECT`** ...) (`MySQL Query Browser` ha anche una voce di menu)
- Esempio: query fatta la volta scorsa con e senza indici

Il risultato di una EXPLAIN

I campi importanti sono:

- `join type` Indica quanto è costoso il **`JOIN`**. Alcuni possibili valori:
  - `const` Solo una riga al massimo fa corrispondenza (Esempio: controllo per uguaglianza con una chiave primaria)
  - `eq_ref` Controllo di uguaglianza fra un valore indicizzato e un valore già letto (Esempio: join fra due tabelle utilizzando una colonna indicizzata)
  - `ALL` E' necessario leggere tutta la tabella
- `rows` Indica quante righe della tabella è necessario leggere
- `key` Indica quale chiave è stata utilizzata

Aggiornamento di valori nel database

Sintassi

```
UPDATE nome_tabella
SET campo1 = valore1 ,
    campo2 = valore , ...
[WHERE condizione]
```

Aggiornamento di valori nel database

Sintassi

```
UPDATE nome_tabella
SET campo1 = valore1 ,
    campo2 = valore , ...
[WHERE condizione]
```

Incrementare di uno le presenze dei giocatori

```
UPDATE players
SET presenze = presenze+1
```

Aggiornamento di valori nel database (2)

Mettere a zero le presenze dei giocatori che hanno il campo presenze a **`NULL`**

Aggiornamento di valori nel database (2)

Mettere a zero le presenze dei giocatori che hanno il campo presenze a **NULL**

Soluzione

```
UPDATE players
SET presenze = 0
WHERE presenze IS NULL
```

Aggiornamento di valori nel database (2)

Mettere a zero le presenze dei giocatori che hanno il campo presenze a **NULL**

Soluzione

```
UPDATE players
SET presenze = 0
WHERE presenze IS NULL
```

Incrementare di uno le presenze dei giocatori che hanno giocato la giornata 31

Aggiornamento di valori nel database (2)

Mettere a zero le presenze dei giocatori che hanno il campo presenze a **NULL**

Soluzione

```
UPDATE players
SET presenze = 0
WHERE presenze IS NULL
```

Incrementare di uno le presenze dei giocatori che hanno giocato la giornata 31

Soluzione

```
UPDATE players
SET presenze = presenze+1
WHERE id
IN (SELECT player_id
    FROM votes WHERE round_id = 31)
```

Calcolare la media voto dei giocatori

Scrivere una query di aggiornamento che prendendo i dati dalla tabella votes, aggiorna il campo media_gazzetta

Calcolare la media voto dei giocatori

Scrivere una query di aggiornamento che prendendo i dati dalla tabella votes, aggiorna il campo media_gazzetta

Soluzione

```
UPDATE players AS p
SET media_gazzetta =
    (SELECT AVG(voto_gazzetta)
     FROM votes
     WHERE player_id = p.id)
```