

Capitolo 1.5

Database relazionali e SQL

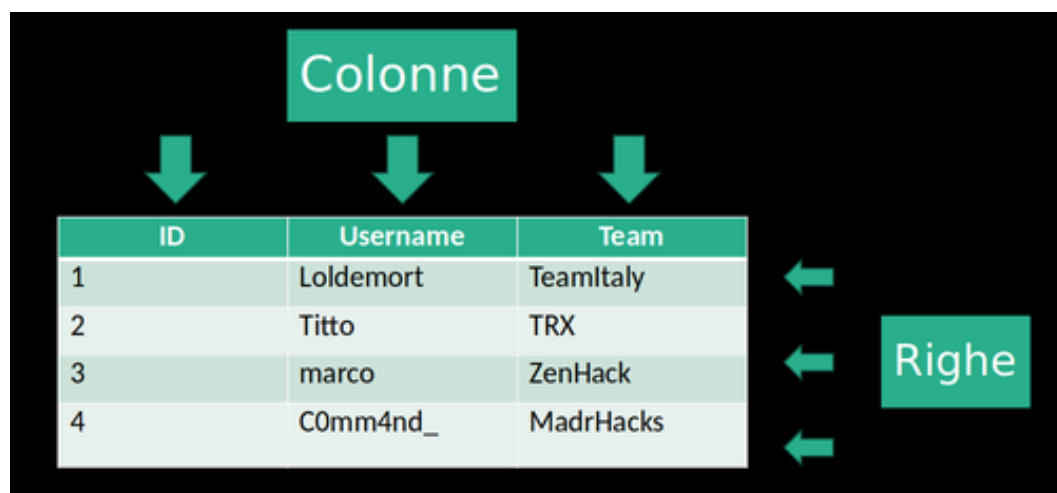
I Relational Database Management System (DBMS) e lo Structured Query Language sono argomenti vastissimi ai quali vengono dedicati interi esami. Tuttavia, per ciò che ci serve, possiamo ottenere risultati soddisfacenti anche solo prendendo dimestichezza con pochi concetti e istruzioni.

Modello relazionale

Nel modello relazionale, le informazioni vengono strutturate in tabelle, righe e colonne.

Un database relazionale è quindi strutturato in modo molto simile ad un foglio di lavoro (esempio: Excel). Ogni foglio di lavoro è una tabella nella quale vengono archiviate informazioni. Le colonne rappresentano i vari attributi, e le righe i “record”, le entità, in un certo senso sono i soggetti dei dati raccolti.

Esempio di tabella:



The diagram shows a table with three columns and four rows. Above the table, a green box labeled 'Colonne' has three green arrows pointing down to the column headers: 'ID', 'Username', and 'Team'. To the right of the table, a green box labeled 'Righe' has three green arrows pointing left to the row numbers: '1', '2', '3', and '4'.

ID	Username	Team
1	Loldemort	TeamItaly
2	Titto	TRX
3	marco	ZenHack
4	C0mm4nd_	MadrHacks

Figure 1: Esempio tabella giocatori CTF

Quindi i record di una tabella condividono la stessa “struttura”: per ognuno di essi abbiamo lo stesso tipo di informazione.

Per ogni tabella è definita una *chiave primaria*, ovvero un dato che identifica univocamente ogni riga o record. La chiave primaria può essere definita da più colonne, o da un’informazione utile sul record, ma per semplicità si tende a definirla su una sola colonna, spesso creata per questo unico scopo, chiamata ID (o simili). Per semplicità potete ricordare la chiave primaria come identificatore univoco. È tuttavia utile ricordare l’eventualità che la chiave primaria possa essere multicolonna o (più plausibile) che essa sia un dato come un codice fiscale o numero di matricola.

Inoltre, ogni riga può essere utilizzata per creare una relazione tra diverse tabelle utilizzando una *chiave esterna*, ovvero la chiave primaria di un’altra tabella.

Esempio riassuntivo:

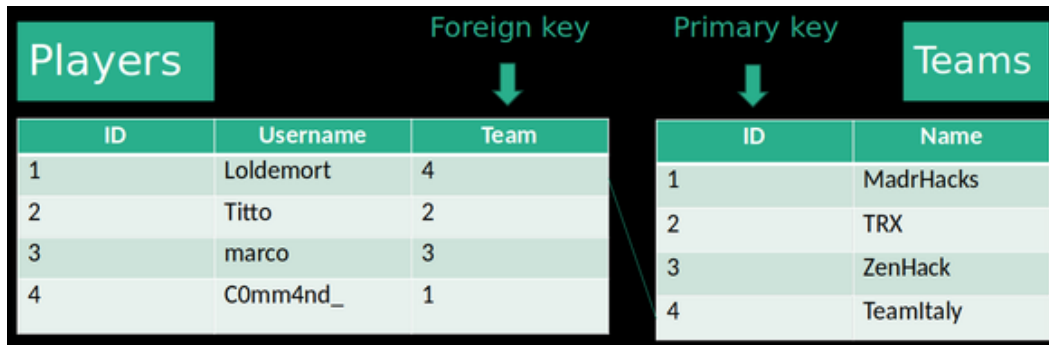


Figure 2: Esempio primary e foreign key

DBMS

Una generica base di dati, o database, è una collezione di dati che viene gestita e organizzata dal DBMS (DataBase Management System). Gli RDBMS gestiscono database relazionali.

L'utente ha in mente uno schema logico di come dovrebbe essere un database (guarda gli esempi di prima), ma i record devono essere memorizzati fisicamente in qualche modo sotto forma di bit. Il DBMS si occupa di gestire i dati in sè, controllando gli accessi concorrenti, assicurando sicurezza e integrità dei dati e permettendo la loro migrazione, tutto questo permettendo all'utente di accedere ai dati attraverso uno schema concettuale piuttosto che ai dati presenti fisicamente in memoria.

tldr: permette l'astrazione assicurando al contempo rapidità di accesso ai dati e la loro integrità.

SQL

SQL è il linguaggio standard per manipolare i database.

Andiamo per esempi. Prima di tutto, vediamo tutti i dati sui quali lavoreremo in questo tutorial:

```
SELECT/FROM SELECT * FROM players;
```

ID	Username	Team
1	Loldemort	4
2	Titto	2
3	marco	3
4	C0mm4nd_	1

```
SELECT * FROM Teams;
```

ID	Name
1	MadrHacks
2	TRX
3	ZenHack
4	TeamItaly

Vediamo i team. L'ID non ci serve a molto... Prendiamo solo i nomi:

```
SELECT Name FROM Teams;
```

Name
MadrHacks
TRX
ZenHack
TeamItaly

E se volessimo vedere solamente il nome della seconda squadra inserita nel database?

```
WHERE SELECT Name FROM Teams WHERE ID = 2;
```

Name
TRX

La struttura della `SELECT` è quindi: `SELECT [colonna/e] FROM [tabella] WHERE [condizione]`, e non è necessario selezionare una colonna per usarla come condizione, come abbiamo visto in quest'ultimo esempio

Ora invece selezioniamo tutti i team tranne i primi due:

```
SELECT * FROM Teams WHERE ID > 2; SELECT * FROM Teams WHERE ID >= 3;
```

ID	Name
3	ZenHack
4	TeamItaly

Però ordiniamoli in ordine alfabetico:

```
SELECT * FROM Teams WHERE ID >= 3 ORDER BY Name;
```

ID	Name
4	TeamItaly
3	ZenHack

Ma la classifica era più bella prima...

ORDER BY SELECT * FROM Teams WHERE ID >= 3 ORDER BY Name DESC;

ID	Name
3	ZenHack
4	TeamItaly

Sintassi completa: SELECT column[s] FROM table[s] WHERE condition[s] ORDER BY column[s] [asc/desc];

Condizioni multiple Se inseriamo più colonne nell'ORDER BY, avrà importanza l'ordine nel quale le elenchiamo. Se per esempio volessimo selezionare dei giocatori in base al punteggio, e in caso di parità dare priorità al più giovane, potremmo usare questa *query*: SELECT name, score FROM players ORDER BY score DESC, age ASC;. Si possono inserire più condizioni in un WHERE usando gli operatori OR e AND.

SQL for exploitation

Ci sono poi altre istruzioni e operatori che tornano particolarmente utili quando si eseguono SQL injection, un tipo di attacco che vedremo nel dettaglio nel prossimo capitolo.

LIKE e wildcards LIKE ci permette di cercare una stringa che “assomiglia” a quella che viene fornita. Questo è possibile grazie alle *wildcards*. Le due wildcards più importanti per i nostri scopi sono l'underscore _, che rappresenta un solo carattere, e il percento %, che rappresenta nessuno o più caratteri. Degli esempi sulla tabella **Players**:

SELECT * FROM Players WHERE Username LIKE "_arco"

ID	Username	Team
3	marco	3

SELECT * FROM Players WHERE Username LIKE "%o"

ID	Username	Team
2	Titto	2
3	marco	3

SELECT * FROM Players WHERE Username LIKE "%o%"

ID	Username	Team
1	Loldemort	4
2	Titto	2
3	marco	3

```
SELECT * FROM Players WHERE Username LIKE "%Titto"
```

ID	Username	Team
2	Titto	2

UNION SELECT Vi ricordate della foreign key e della primary key?

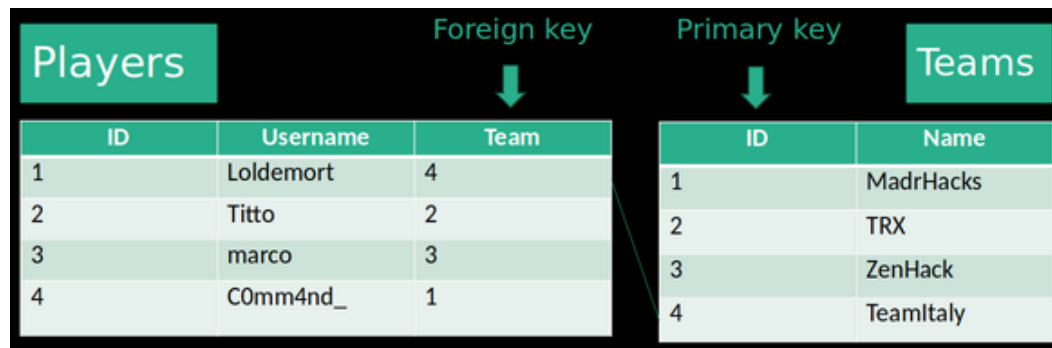


Figure 3: Esempio primary e foreign key

La JOIN è un'istruzione che ci permette di eseguire manipolazioni utili usando queste due informazioni. A noi per il momento interessa solamente ottenere informazioni da due tabelle diverse, indipendentemente dalla presenza o meno di un legame tra di esse, ed in questo ci aiuta la UNION. Per usarla, basta scrivere due SELECT relative a due tabelle diverse, e metterci una UNION in mezzo:

```
SELECT Username FROM Players WHERE Username LIKE "L%" UNION SELECT Name
FROM Teams WHERE ID = 2
```

Username
Loldemort
TRX

Quando eseguiamo una UNION SELECT, dobbiamo tenere a mente che:

- Ogni select deve avere lo stesso numero di colonne. `SELECT ID, Username FROM Players UNION SELECT Name FROM Teams` non è quindi valida.
- Le colonne devono riguardare tipi di dato "simili". Ad esempio, stringhe e varchar, pur non essendo lo stesso tipo, possono far parte della stessa colonna IN UNA QUERY UNION! `SELECT ID FROM Players UNION SELECT Name FROM Teams` restituisce un errore.
- Le colonne generate da una UNION SELECT avranno lo stesso nome delle colonne selezionate dalla prima tabella nominata. Questo non rappresenta di per sè un problema, ma può generare confusione quando ci vengono restituiti i risultati della query (nell'esempio di prima, i TRX appaiono nella colonna Username).

In quanto futuri xHackerZx, non possiamo scoraggiarci alle prime difficoltà. Ci sono delle scorciatoie che possiamo utilizzare, forzando delle funzionalità particolari messe a disposizione dalle query SQL.

Mock columns “Ogni SELECT deve avere lo stesso numero di colonne” In caso l'applicazione con la quale stiamo interagendo ci proponesse una query con troppe colonne (vogliamo sapere solo i nomi dei team tramite una union, ma nella tabella players viene selezionato anche l'ID), possiamo usare le mock columns.

Queste consistono nell'inserire dei valori fissi al posto del nome della colonna in modo che dalla query venga selezionata una colonna finta:

```
SELECT ID, Username FROM Players UNION SELECT 1337, Name FROM Teams
```

ID	Username
1	Loldemort
2	Titto
1337	TeamItaly
1337	TRX

etc...

Possiamo anche usare "carattere" se vogliamo creare una finta colonna di tipo varchar.

Concatenazione Se invece avessimo a disposizione troppe poche colonne, possiamo sfruttare la concatenazione:

```
SELECT Name FROM Teams UNION SELECT CONCAT(Username," ",Fullname) FROM Players
```

Il metodo di concatenazione varia enormemente tra un DBMS e l'altro, quindi sarà necessario fare una nuova ricerca sulla concatenazione ogni qualvolta troveremo un nuovo DBMS.

Nel nostro caso però nella tabella Players non è presente il **Fullname**, oltre l'username ci sono solo ID e l'ID della squadra del giocatore come foreign key. Non avremmo potuto concatenare queste informazioni con **Username**, visto che queste altre sono interi e non varchar. In casi come questi, la scorciatoia presente nel prossimo paragrafo torna particolarmente utile

CASTing “Le colonne devono riguardare tipi di dato simili” In questo caso possiamo fare affidamento al CASTing, che ci permette di trasformare i dati da un tipo all'altro quando possibile. Ad esempio, la query:

```
SELECT Username FROM Players UNION SELECT CAST(ID as varchar) FROM Teams
```

è valida e restituisce

Username
Loldemort
Titto
"1"
"2"

Anche il CASTing, come la concatenazione, può variare molto tra i vari DBMS. In generale questo è vero per quasi tutte le istruzioni che vanno oltre alla soddisfazione delle esigenze

più che basilari del programmatore, come semplici **SELECT**. Per questo è più utile imparare a cercare le informazioni necessarie su internet che imparare a memoria la sintassi dello standard SQL.

AS “Le colonne generate da una **UNION SELECT** avranno lo stesso nome delle colonne selezionate dalla prima tabella nominata.” Come già detto, questo non rappresenta per noi un problema. Se lo si vuole risolvere, basta usare la keyword **AS** sulle prime colonne selezionate:

```
SELECT Username FROM Players AS "UserAndTeamNames" UNION SELECT Name FROM Teams
```

UserAndTeamNames
Loldemort
Titto
TRX
TeamItaly

etc. . .

Come per la **SELECT**, se si devono rinominare più colonne, basta dividere i vari nomi con una virgola.

Pratica: <https://sqlbolt.com/>