

# Introduzione

Questo "manuale" vuole essere una guida utile all'arte della web exploitation per quanto riguarda i programmi seguiti da CyberChallenge negli anni antecedenti al 2025.

Nel mondo sono già disponibili diversi libri che approfondiscono questa materia come merita, e pullulla di write-up e repository da leggere e dalle quali copiare payload da usare e modificare come più ci piace.

La mia intenzione è quella di creare un testo soddisfacente in lingua italiana (nei limiti imposti dalla terminologia tecnica) che copra solamente gli argomenti relativi all'esfiltrazione di dati, così come richiesto dal programma e dagli obiettivi di CyberChallenge.

È ovvio che la sicurezza web non si limiti solo a questo: per un'azienda anche il solo fatto di avere un sito web non disponibile può provocare milioni di euro di danni, per non parlare delle possibili ricadute sulla sua immagine.

Volevo però fare in modo di mettere a disposizione un testo dritto al punto e che non risultasse troppo verboso, toccando solamente i temi necessari al programma in modo da risparmiare tempo a chi deciderà di non approfondire questo ambito per conto proprio, e allo stesso tempo permettere agli appassionati di acquisire una base tecnica sufficiente ad affrontare temi più entusiasmanti e decisamente più complessi.

Mi sembra doveroso sottolineare che questo progetto è legato solamente a me, Tiziano Caruana, e che nè le persone affiliate al progetto [CyberChallenge.IT](#), nè i dipendenti e gli studenti di Sapienza me escluso, nè nessun'altra persona sono legate a questo progetto. Inoltre io, ovvero l'autore di questo "manuale", non ho esperienze dirette nell'ambito della sicurezza informatica nè mi sono posto l'obiettivo di raggiungere con le mie parole professionisti, dilettanti o studenti che abbiano l'intenzione di approcciare applicazioni reali, che sia per motivi professionali o meno. Non mi prendo quindi responsabilità sulle ripercussioni che le conoscenze acquisite durante la lettura di questo "manuale", o i codici inseriti nello stesso, possono avere se utilizzate al di fuori del contesto per cui sono state create, ovvero per la risoluzione di challenge nel contesto di [CyberChallenge.IT](#), che sia negli addestramenti o nelle gare.

Le note sono degli approfondimenti inseriti per completezza che possono tornare utili in challenge particolari e situazioni specifiche, ma che non sono fondamentali per la risoluzione delle challenge proposte nel training interno nè per la comprensione degli argomenti successivi.

Un po' di spazio sarà lasciato per i pre-requisiti necessari a capire le varie tipologie di attacco che altrimenti risulterebbero incomprensibili al lettore. Il numero di partecipanti proveniente dalle superiori o da corsi di laurea non strettamente legati all'informatica non è trascurabile, ed è per questo necessario navigare anche argomenti non direttamente legati all'exploitation.

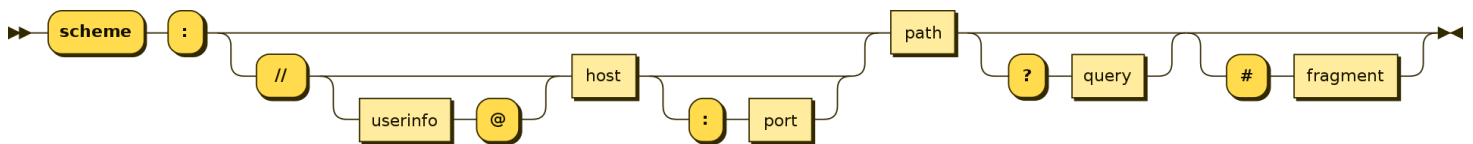
Il capitolo 0 e i capitoli X.5 sono di questa natura, e possono quindi essere saltati da persone che si ritengono sufficientemente preparate sull'argomento. Non abbiate paura di saltare questi capitoli o anche quelli relativi a vulnerabilità che già conoscete. Se vi rendete conto che avete commesso un errore, o se vi serve qualcosa nello specifico in futuro, potrete sempre tornarci successivamente.

# Capitolo 0

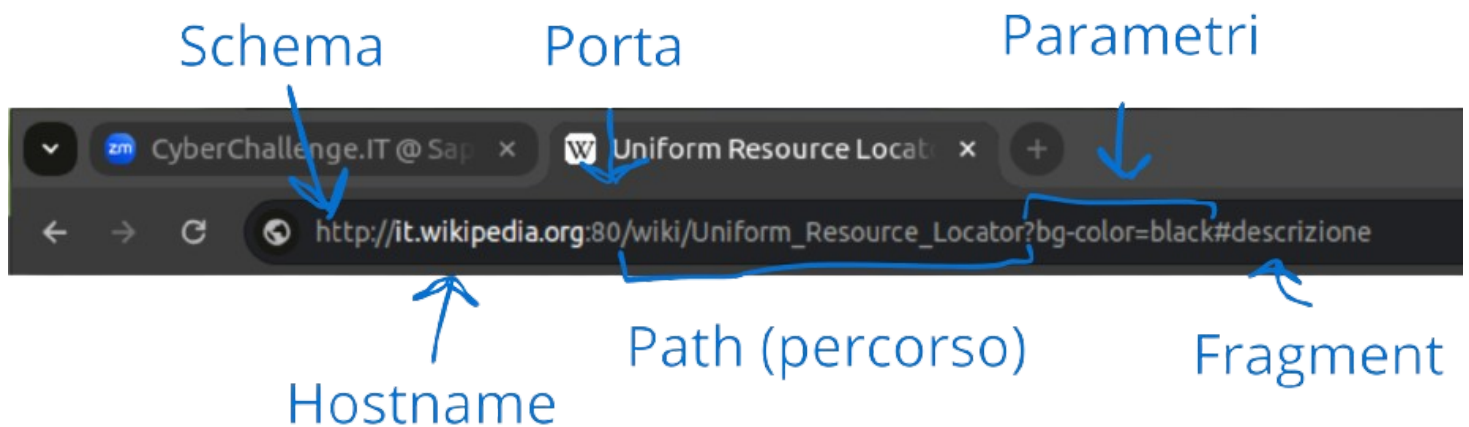
## Internet

Nel World Wide Web ogni risorsa viene identificata univocamente da un **URL** (Uniform Resource Locator)

Illustrazione formale:



Esempio pratico:



*Nota: la porta di default - quindi il valore valido se non specificato - è l'80, il fragment è di default l'inizio pagina. Le query string, o parametri, sono generalmente facoltativi. Le userinfo, ovvero username e password, non mostrati nell'esempio, sono tipici di protocolli diversi da quelli che vediamo usualmente nel browser, come **FTP***

## URL-encoding

Per evitare che nell'URL siano presenti caratteri riservati che potrebbero portare ad un'interpretazione indesiderata da parte del browser, viene usato l'URL encoding, ufficialmente percent-encoding. Se usati per un attacco è quindi utile ricordarsi di codificare gli URL in modo che nessuna parte del "vettore d'attacco"/payload che abbiamo preparato vada persa.

Prima dell'URL-encoding:

Dopo l'URL-encoding (cosa riceve il server):

# Your input was received as:

text=TRX%26gamingClub%21

Da notare come solo il testo che può essere direttamente controllato dall'utente venga URL-encodato.

Ciò che viene fatto è una conversione dal carattere riservato alla sua rappresentazione ASCII esadecimale preceduta da un %.

## HTTP

L'HyperText Transfer Protocol, HTTP, è un protocollo stateless, ovvero ogni richiesta è indipendente dalle richieste precedenti. Le due fasi previste sono l'HTTP request (il client fa una richiesta al server) e l'HTTP response (il server risponde).

Esempio di HTTP request:

```
Method          Path          Version
1 GET /ctf_hscf2023/2023/06/12/flagshop.html HTTP/2
2 Host: theromanxplo.it
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:122.0) Gecko/20100101
  Firefox/122.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: https://theromanxplo.it/
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-User: ?1
13 Te: trailers
14
```

Riga della richiesta

Header(s)

Esempio di HTTP response:

Versione      Status code

1 HTTP/2 200 OK      Riga dello stato

2 Server: GitHub.com

3 Content-Type: text/html; charset=utf-8

4 Last-Modified: Tue, 12 Dec 2023 21:06:45 GMT

5 Access-Control-Allow-Origin: \*

6 Etag: W/"6578cb65-9f1b"

7 Expires: Mon, 19 Feb 2024 13:37:36 GMT

8 Cache-Control: max-age=600

9 X-Proxy-Cache: MISS

10 X-Github-Request-Id: DCE2:0E92:4902CC2:4A55098:65D35748

11 Accept-Ranges: bytes

12 Date: Mon, 19 Feb 2024 13:27:36 GMT

13 Via: 1.1 varnish

14 Age: 0

15 X-Served-By: cache-fco2270027-FCO

16 X-Cache: MISS

17 X-Cache-Hits: 0

18 X-Timer: S1708349257.866990,VSO,VE131

19 Vary: Accept-Encoding

20 X-Fastly-Request-Id: 50d95ffd4ba3bc9b6aec2dc49cdce14babf193ec

21 Content-Length: 40731

22

23 <!DOCTYPE html>      Header(s)

24 <html lang="en-us">      Body message

In caso fosse necessario uno scambio di informazioni al di fuori del contesto degli header e delle query, queste possono essere incluse nel corpo del messaggio (body message). Nell'esempio di richiesta mostrato, apparirebbero "sotto" agli header. La struttura del body message varia a seconda della tecnologia utilizzata dallo specifico sito, ed è comunque facilmente individuabile durante l'esperienza pratica.

## Metodi HTTP comuni:

- **GET** (ottiene la risorsa o informazioni su di essa)
- **POST** (azione/invia dati alla risorsa)
- **HEAD** (GET senza body message)
- **TRACE** (diagnostica)
- **OPTIONS** (visualizza metodi disponibili)
- **PUT** (crea nuova risorsa)
- **DELETE** (elimina risorsa specificata)
- **CONNECT** (crea un tunnel in caso di proxy)
- **PATCH** (modifica la risorsa)

## Header HTTP comuni:

### Request

- **Accept:** Definisce i **MIME type** che il client accetterà dal server, in ordine di preferenza. Ad esempio, `Accept: application/json, text/html` indica che il client preferisce ricevere risposte in JSON, ma le accetta anche in HTML.
- **User-Agent:** Identifica il browser e/o il client che sta effettuando la richiesta.
- **Authorization:** Usato per l'invio di credenziali, utile quando si prova ad accedere ad una risorsa protetta.

- **Cookie:** Usato per inviare al server cookie precedentemente memorizzati. Utile per personalizzare l'esperienza dell'utente e "combattere" i limiti della natura stateless del protocollo HTTP.
- **Content-Type:** Definisce il MIME type del contenuto del request body.

## Response

- **Content-Type:** Come sopra.
- **Server:** La controparte di User-Agent .
- **Set-Cookie:** Comunica al client che dovrebbe memorizzare un cookie con un certo nome, valore, e facoltativamente scadenza, dominio, percorso e flag di sicurezza. Esempio: Set-Cookie: score=127 . Una volta che Set-Cookie viene ricevuto ed accettato, il client invierà al server il cookie ad ogni richiesta eseguita.
- **Content-Length:** Specifica la grandezza in byte del response body. In caso "apparisse" dal lato del richiedente, dobbiamo fare attenzione a specificare la lunghezza giusta in caso volessimo modificare i nostri payload.

Negli esempi mostrati precedentemente potete vedere come questi header vengono utilizzati in una comunicazione reale tra un web browser e un sito web statico.

Generalmente, quando un header inizia per X- , è custom.

È utile notare come il funzionamento di HTTP sia solo una convenzione, ed il server può decidere di implementare qualsiasi metodo e qualsiasi header (custom headers e methods). Questi elementi sono di nostro interesse, essendo implementati direttamente dal gestore del sito e quindi più facilmente soggetti ad errori di implementazione. Inoltre, nulla impedisce al programmatore di usare una GET per modificare dati, o una POST per fornire informazioni, nonostante questo sia ovviamente sconsigliato. Lo stesso vale per gli elementi mostrati successivamente in questo capitolo.

## Status codes:

- **1xx:** Risposte informative
- **2xx:** Successo
- **3xx:** Reindirizzamento
- **4xx:** Errore del client (tipicamente richieste sbagliate)
- **5xx:** Errore del server (errori di un programma ed eccezioni non gestite)

## Riassumendo:

Possiamo pensare all'HTTP request come a una lettera che spediamo tramite posta. Nella riga della richiesta, noi come mittenti specifichiamo cosa vogliamo sia fatto e dove vogliamo sia fatto, come scrivessimo l'indirizzo e il tipo di servizio desiderato su una busta. Gli header della richiesta contengono informazioni su di noi e le nostre preferenze, simili a scrivere il nostro nome e il nostro indirizzo sul retro della busta. Se chi offre il servizio ha bisogno di un materiale o di un oggetto da utilizzare per soddisfare la nostra richiesta, possiamo includerlo nel body message, proprio come inviare un pacco insieme alla busta.

Il server, simile al destinatario della nostra lettera, riceve la richiesta, cerca di soddisfarla e ci invia una lettera di risposta. Nella riga dello stato della risposta, capiamo se tutto è andato bene o se c'è stato un problema, proprio come leggere l'indicazione di consegna sulla nostra busta postale. Negli header della risposta, otteniamo informazioni su chi ha eseguito il lavoro e come vorrebbe che ci comportassimo con il risultato. Infine, nel body message della risposta, riceviamo il prodotto richiesto, come se insieme alla busta ci venisse spedito un pacco contenente ciò che avevamo chiesto.

## HTTP Cookies

I cookie vengono spesso arricchiti da attributi, ed i principali sono:

- **Expires:** Specifica il tempo di scadenza in secondi per il cookie. Se non specificato, il cookie viene eliminato al termine della sessione (session cookie).
- **Secure:** I cookie con questa flag vengono inviati solo in richieste [HTTPS](#) (HTTP crittografato).
- **HttpOnly:** JavaScript non può accedere al cookie.
- **Domain:** Definisce il dominio per il quale il cookie è valido.
- **Path:** Stessa cosa ma col percorso.
- **Same-Site:** Specifica se il cookie può venire incluso in richieste che coinvolgono siti terzi.  
SameSite=Strict indica che il browser si rifiuterà di condividere il cookie con siti web diversi da quello che ci ha "detto" di settare il cookie. È una protezione che può scoraggiare attacchi CSRF.

Un cookie (nel browser: F12 -> Application -> Cookies):

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
__Secure-1PSIDCC	ABTWWhQFjb4ZkHFI4adWEQoNQbtbk	.google.com	/	2025-02-18T16:08:40.000Z	44	✓	✓	

## HTTP Authentication

Secondo gli standard del protocollo HTTP, la struttura della comunicazione tra un client che richiede una risorsa protetta e il server consiste nei seguenti step:

- Il client richiede la risorsa
- Il server risponde con lo status code `401 Unauthorized` (non autorizzato), specificando tramite l'header `WWW-Authenticate` il tipo di autenticazione richiesto. In questa fase possono essere inviate al client diverse informazioni, a seconda del metodo di autenticazione richiesto.
- Il client deve rispondere con l'header `Authorization` contenente le credenziali richieste.
- Il server risponde `200 OK` o `403 Forbidden` (accesso vietato).

`401 Unauthorized` = "non so chi sei", `403 Forbidden` = "non sei un utente che ha accesso alla risorsa".

### Principali tipi di autenticazione:

- **Basic Authentication:** `username:password` vengono inviati encodati in `base64`. L'encoding non offre nessun layer di sicurezza aggiuntivo, di conseguenza un'autenticazione `Basic` in HTTP è completamente insicura, come mandare le informazioni in chiaro.

- **Digest Authentication:** Offusca username e password usando anche altri parametri come `realm` e `nonce`
- **Bearer Authentication:** Usata soprattutto in contesti basati su [OAuth2](#). Di fatto, invece di fornire le credenziali al server che richiede l'autenticazione, ci autenticiamo presso un altro server del quale il server iniziale si fida. Questo è possibile perchè il server autenticante fornisce all'utente un token da usare come "cartellino d'ingresso" quando passiamo all'ultima fase del processo di autenticazione. Spesso i token sono generati come [JWT](#).

*Nota: I tipi di autenticazione potrebbero non essere chiari fin da subito, e non sono necessariamente spendibili in attacchi utili al contesto di CyberChallenge, ma l'utilizzo delle Bearer authentication è in continua crescita ed è un argomento che ritengo particolarmente importante. Ne consiglio caldamente l'approfondimento autonomo.*

## Conclusione del capitolo:

Quando difendiamo o attacchiamo un servizio, è utile ricordare che i cookies, gli header, il body content e il metodo della richiesta possono venire modificati dal client come esso vuole. Esistono strumenti (come BurpSuite) che permettono di modificare facilmente tutte le informazioni possibili che il server è in grado di ricevere. Fidarsi di ciò che viene inviato dal client significa accettare di gestire un servizio vulnerabile. Uno sviluppatore deve fare in modo di limitare per quanto possibile le funzionalità che richiedano una fiducia dell'utente. La natura stateless di HTTP costringe gli sviluppatori ad usare i cookie per l'autenticazione (immaginate di dover loggare ogni volta che cambiate reel), mettendoli in difficoltà e aprendo la possibilità a vari tipi di attacchi cross-site che vedremo più avanti.

*Challenge: Prime 6 di web security a partire da [questa](#)*

# Capitolo 0.5

## Database relazionali e SQL

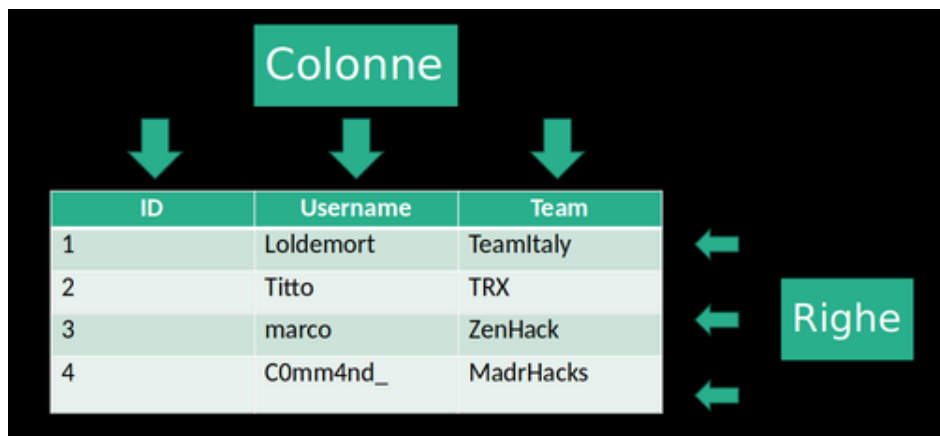
I Relational Database Management System (DBMS) e lo Structured Query Language sono argomenti vastissimi ai quali vengono dedicati interi esami. Tuttavia, per ciò che ci serve, possiamo ottenere risultati soddisfacenti anche solo prendendo dimestichezza con pochi concetti e istruzioni.

### 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 rows of data.

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

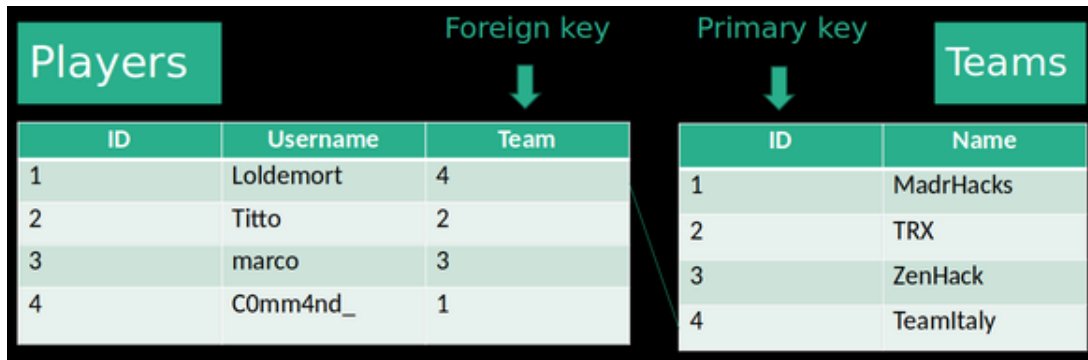
Quindi i record di una tabella condividono la stessa "struttura": per ognuno di essi abbiamo lo stesso tipo di informazione.

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:



## 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 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?

```
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...

```
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];
```

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` 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

ID	Username	Team
3	marco	3

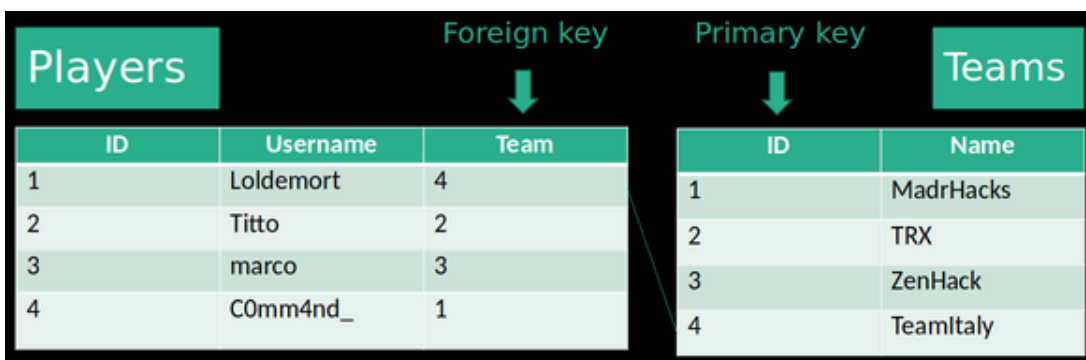
```
SELECT * FROM Players WHERE Username LIKE "%o"
```

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

Vi ricordate della `foreign key` e della `primary 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.

## "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 .

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

## "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.

## "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/>

# Capitolo 1

## SQL injection