1.Banco de dados

Um banco de dados é uma forma de armazenar coleções de dados eletronicamente de maneira organizada. Um banco de dados é controlado por um DBMS, que é a sigla para **Database Management System**. Os DBMSs se dividem em dois grupos: **Relacionais** e **Não Relacionais**. Alguns exemplos comuns são MySQL, Microsoft SQL Server, Access, PostgreSQL e SQLite.

Dentro de um DBMS, você pode ter vários bancos de dados, cada um contendo seu próprio conjunto de dados relacionados. Você armazenaria essas informações separadamente dentro do banco de dados usando algo chamado **tabelas**. As tabelas são identificadas com um nome único cada uma.

Uma tabela é composta por **colunas** e **linhas**. Uma maneira útil de imaginar uma tabela é como uma grade: as colunas vão da esquerda para a direita no topo contendo o nome do campo, e as linhas vão de cima para baixo, cada uma contendo os dados reais.

1.1. Colunas

Cada coluna, mais corretamente chamada de **campo**, tem um nome único por tabela. Ao criar uma coluna, você também define o tipo de dado que ela conterá; alguns tipos comuns são inteiros (números), strings (texto padrão) ou datas. Alguns bancos de dados permitem tipos muito mais complexos, como dados geoespaciais, que contêm informações de localização.

Uma coluna contendo um inteiro também pode ter a função **auto-incremento** ativada. Isso atribui a cada linha um número único que aumenta com cada nova linha inserida. Isso cria o que é chamado de **campo chave**; uma chave deve ser única para cada linha de dados e pode ser usada para localizar exatamente aquela linha em consultas SQL.

1.2. Linhas

Linhas, ou **registros**, contêm linhas individuais de dados. Quando você adiciona dados à tabela, um novo registro é criado; quando você exclui dados, um registro é removido.

2.Bancos de Dados Relacionais vs Não Relacionais

Banco de dados relacional armazena informações em tabelas e, frequentemente, as tabelas compartilham informações entre si. Eles usam colunas para definir os dados armazenados e linhas para armazenar os dados de fato. As tabelas geralmente contêm uma coluna com um ID único (**chave primária**), que é usada em outras tabelas para fazer referência a ela, criando um relacionamento entre as tabelas

Bancos de dados não relacionais, também chamados de **NoSQL**, não utilizam tabelas, colunas e linhas para armazenar dados. Um layout específico de banco de

dados não precisa ser construído, então cada linha de dados pode conter informações diferentes, oferecendo mais flexibilidade do que um banco relacional.

SQL (**Structured Query Language**) é uma linguagem rica em recursos usada para consultar bancos de dados. Essas consultas SQL são melhor referidas como *declarações* (statements).

3.SELECT

O primeiro tipo de consulta que aprenderemos é a consulta **SELECT**, usada para recuperar dados do banco de dados.

```
select * from users;
```

A primeira palavra, **SELECT**, indica ao banco de dados que queremos recuperar dados; o * indica que queremos receber **todas as colunas** da tabela. Por exemplo, a tabela pode conter três colunas (id, username e password).

O trecho **"from users"** diz ao banco de dados que queremos recuperar os dados da tabela chamada **users**. Por fim, o ponto e vírgula indica que a consulta terminou.

A próxima consulta é similar à anterior, mas desta vez, em vez de usar * para retornar todas as colunas, estamos solicitando apenas os campos **username** e **password**:

```
select username, password from users;
```

A consulta seguinte, como a primeira, retorna todas as colunas usando o *, mas a cláusula **LIMIT 1** obriga o banco de dados a retornar **apenas uma linha**:

```
select * from users LIMIT 1;
```

Alterar a consulta para LIMIT 1,1 faz com que ela pule o primeiro resultado, LIMIT 2,1 pula os dois primeiros, e assim por diante.

Lembre-se:

- o **primeiro número** informa quantos resultados devem ser ignorados;
- o **segundo número** informa quantas linhas devem ser retornadas.

A consulta abaixo retorna apenas as linhas onde o username é **igual a 'admin'**:

```
select * from users where username='admin';
```

A próxima consulta retorna as linhas onde o username **não é igual a 'admin'**:

```
select * from users where username != 'admin';
```

A seguinte retorna as linhas onde o username é **admin** ou **jon**:

```
select * from users where username='admin' or username='jon';
```

E esta retorna apenas as linhas onde **username é 'admin' e password é 'p4ssword'**:

```
select * from users where username='admin' and password='p4ssword';
```

Usar a cláusula **LIKE** permite especificar dados que não correspondem exatamente, mas que **começam**, **contêm** ou **terminam** com certos caracteres, usando o caractere curinga %.

```
select * from users where username like 'a%';
```

Retorna qualquer linha cujo username **começa** com a letra *a*.

```
select * from users where username like '%n';
```

Retorna qualquer linha cujo username **termina** com a letra *n*.

```
select * from users where username like '%mi%';
```

Retorna qualquer linha cujo username **contém** a sequência *mi*.

4.UNION

A instrução **UNION** combina os resultados de duas ou mais instruções **SELECT** para recuperar dados de uma ou várias tabelas.
As regras para usar UNION são:

1. Cada SELECT deve retornar **o mesmo número de colunas**;
2. As colunas devem ter **tipos de dados semelhantes**;
3. A **ordem das colunas** deve ser a mesma em cada SELECT.

Isso pode parecer confuso, então vamos usar a seguinte analogia: Imagine que uma empresa quer criar uma lista de endereços de todos os clientes e fornecedores para enviar um novo catálogo. Temos uma tabela chamada **customers** com o conteúdo:

Usando a seguinte instrução SQL, podemos reunir os resultados das duas tabelas e combiná-los em um único conjunto:

```
SELECT name, address, city, postcode FROM customers
UNION
SELECT company, address, city, postcode FROM suppliers;
```

5.INSERT

A instrução **INSERT** informa ao banco de dados que queremos inserir uma nova linha de dados na tabela.

- **"into users"** indica a tabela em que os dados serão inseridos;
- **"(username,password)"** especifica as colunas que receberão dados;

- **"values ('bob','password123');"** fornece os valores correspondentes a essas colunas.

Exemplo:

```
insert into users (username,password) values ('bob','password123');
```

6.INSERT

A instrução **INSERT** informa ao banco de dados que queremos inserir uma nova linha de dados na tabela.

- **"into users"** indica a tabela em que os dados serão inseridos;
- **"(username,password)"** especifica as colunas que receberão dados;
- **"values ('bob','password123');"** fornece os valores correspondentes a essas colunas.

Exemplo:

```
insert into users (username,password) values ('bob','password123');
```

7.UPDATE

A instrução **UPDATE** informa ao banco de dados que queremos atualizar uma ou mais linhas em uma tabela.

Você especifica a tabela com **"update nome_da_tabela SET"**, depois define os campos a atualizar como uma lista separada por vírgulas, por exemplo: **username='root', password='pass123'**

Por fim, assim como no SELECT, você pode especificar exatamente quais linhas serão atualizadas com a cláusula **WHERE**:

```
update users SET username='root', password='pass123' where username='admin';
```

8.DELETE

A instrução **DELETE** informa ao banco de dados que queremos remover uma ou mais linhas.
Tirando o fato de não haver seleção de colunas, o formato é semelhante ao SELECT.

Você pode usar a cláusula **WHERE** para definir quais dados serão apagados e **LIMIT** para determinar quantas linhas serão removidas.

Exemplo:

```
delete from users where username='martin';
```

9.O que é SQL Injection?

O ponto em que uma aplicação web que usa SQL pode se tornar vulnerável a **SQL Injection** é quando dados fornecidos pelo usuário são incluídos diretamente em uma consulta SQL.

9.1. Como isso se parece?

Imagine o seguinte cenário: você está navegando em um blog online onde cada postagem possui um ID único. As postagens podem ser públicas ou privadas, dependendo de estarem prontas ou não para serem exibidas ao público. A URL de cada postagem pode ser algo assim: **https://website.thm/blog?id=1**

A partir da URL acima, podemos ver que a postagem selecionada vem do parâmetro **id** na *query string*. A aplicação precisa recuperar esse artigo do banco de dados e pode usar uma instrução SQL como: SELECT * from blog where id=1 and private=0 LIMIT 1;

Com base no que você aprendeu anteriormente, você pode entender que essa instrução SQL está procurando na tabela **blog** um artigo com:

- **id = 1**
- **private = 0** (ou seja, a postagem é pública)
- E limita o resultado a **apenas um registro**

9.2. Onde o SQL Injection aparece?

Como mencionado no início, SQL Injection ocorre quando **a entrada do usuário** é inserida diretamente na consulta SQL.

Nesse caso, o parâmetro **id** da URL é usado diretamente no comando SQL.

Vamos supor que o artigo de ID **2** ainda é privado e não pode ser visualizado no site. Mas você poderia tentar acessar: https://website.thm/blog?id=2;--

Isso faria com que a aplicação gerasse a seguinte query:

```
SELECT * from blog where id=2;-- and private=0 LIMIT 1;
```

- O **ponto e vírgula (;)** encerra a instrução SQL.
- Os **dois traços (--)** transformam o restante da query em comentário.

Na prática, a consulta real executada seria apenas: **SELECT * from blog where id=2;--**

Isso retornaria o artigo de ID **2**, independentemente de estar público ou privado.

9.3. In-Band SQL Injection

In-Band SQL Injection é o tipo mais fácil de detectar e explorar. “In-Band” significa que o mesmo canal de comunicação é usado **tanto para explorar a vulnerabilidade quanto para receber os resultados**.

Por exemplo: descobrir uma vulnerabilidade de SQL Injection em uma página de um site **e ver os dados do banco de dados sendo exibidos na própria página**.

9.4. Error-Based SQL Injection

Esse tipo de SQL Injection é muito útil para obter informações sobre a **estrutura do banco de dados**, pois mensagens de erro do servidor SQL são exibidas diretamente no navegador.

Isso pode permitir que o invasor **enumere (descubra) toda a estrutura do banco de dados** com bastante facilidade.

9.5. Union-Based SQL Injection

Esse tipo de injeção utiliza o operador **SQL UNION**, junto com instruções **SELECT**, para retornar resultados adicionais na página. É o método mais comum para **extrair grandes quantidades de dados** por meio de uma vulnerabilidade de SQL Injection.

9.6. Blind SQLi (SQL Injection Cega)

Ao contrário da **In-Band SQL Injection**, onde podemos ver diretamente na tela os resultados do ataque, a **Blind SQL Injection** acontece quando recebemos **pouco ou nenhum retorno** que confirme se nossas consultas injetadas realmente funcionam.

Isso ocorre porque as mensagens de erro foram desativadas, porém, **a injeção ainda funciona**. Pode surpreender, mas mesmo com feedback mínimo, ainda é possível **enumerar um banco de dados inteiro**.

9.7. Bypass de Autenticação

Uma das técnicas mais simples de Blind SQL Injection envolve **burlar métodos de autenticação**, como formulários de login. Nesse caso, não estamos interessados em recuperar dados do banco; tudo o que queremos é **passar pelo login**.

Formulários de login que verificam usuários em um banco de dados normalmente funcionam de maneira simples: a aplicação verifica **se o par username/password corresponde a um registro válido**. Em termos simples, a aplicação pergunta ao banco: “Você tem um usuário com o nome bob e a senha bob123?”

O banco responde **sim ou não**. Com base nessa resposta, a aplicação decide se permite ou não que você entre.

Considerando essa lógica, **não é necessário descobrir um par válido de usuário e senha**. Só é preciso construir uma consulta ao banco que resulte em um **retorno verdadeiro**.

9.8 Boolean-Based (Baseado em Booleano)

O *Boolean-based SQL Injection* refere-se ao tipo de resposta obtida a partir das tentativas de injeção. Essa resposta pode ser **verdadeiro/falso**, **sim/não**, **on/off**, **1/0** ou qualquer outro retorno binário.

Esse tipo de comportamento confirma se o payload injetado foi interpretado como verdadeiro ou falso pelo banco de dados.

À primeira vista, pode parecer que respostas tão limitadas fornecem pouca informação, mas, na prática, apenas esses dois possíveis resultados já são suficientes para **descobrir toda a estrutura e conteúdo de um banco de dados**, caractere por caractere.

9.9. Out-of-band SQL Injection

A *Out-of-band SQL Injection* não é tão comum, pois depende de recursos específicos estarem habilitados no servidor de banco de dados ou de a lógica da aplicação web realizar algum tipo de comunicação externa baseada no resultado de uma consulta SQL.

Um ataque *Out-of-Band* é caracterizado pelo uso de **dois canais de comunicação diferentes**:

- um canal para **lançar o ataque**,
- e outro canal separado para **coletar os resultados**.

Por exemplo: O canal de ataque pode ser uma requisição web normal, enquanto o canal de coleta pode envolver o monitoramento de solicitações **HTTP ou DNS** enviadas para um serviço controlado pelo atacante.

Funcionamento geral

1. O atacante envia uma requisição para um site vulnerável a SQL Injection, contendo um payload injetado.
2. O site executa uma consulta SQL no banco de dados, que inclui o payload fornecido pelo atacante.
3. O payload contém um comando que força o servidor a fazer uma requisição HTTP/DNS para a máquina do atacante, enviando nela os dados extraídos do banco.