

Banco de Dados

Explicação sobre SQL, CRUD's, XSS e Sanitização.

Temas

Tópicos

- Conexões.
- Execução de Comandos SQL.
- Operações CRUD.
- Consultas Parametrizadas.
- Sanitização e Prevenção de SQL Injection e XSS (Cross-site Scripting).

• Conexões

- A conexão padrão é composta por propriedades básicas que indicam o servidor, o título do banco ao qual o programa deve se conectar e informações de autenticação (usuário e senha).
- Veja abaixo:
- `Server=myServerAddress;Database=myDataBase;UserId=myUsername;Password=myPassword;`

• Conexões

- podemos ver a estrutura de uma conexão padrão com as propriedades exibidas na ordem em que foram descritas. Esta estrutura é descrita como padrão por ser o suficiente para a maior parte dos projetos. Porém a mesma definição de padrão pode variar de acordo com a tecnologia utilizada para desenvolvimento ou serviço de dados.
- A propriedade Server: É definida aqui através do domínio ao qual o servidor do banco de dados pode estar relacionado assim como pode ser um endereço ip ou nome de uma instância local (veremos mais a frente).
- A propriedade Database: Esta propriedade deve receber o nome do banco de dados.
- A propriedade User Id: também conhecida como nome do usuário ou username deve ser usada para identificar quem ou qual usuário está tentando se conectar através desta conexão.
- A propriedade Password: Deve ser preenchida com a palavra-passe ou senha definida para o mesmo usuário que foi definido na propriedade User Id.

• Conexão a uma Instância

- Conectar à uma instância nada mais é que especificar através do nome a instalação do banco de dados que você deseja utilizar. Podemos ter várias versões do mesmo banco em uma única máquina e ao conectar sem declarar este nome, estamos nos conectando à instância padrão.
- Ao adicionarmos a propriedade Server: Podemos além de declarar o nome ou endereço do servidor, a instância que desejamos nos conectar. Veja no exemplo abaixo:
- `Server=myServerName\myInstanceName;Database=myDataBase;User Id=myUsername;Password=myPassword;`

Execução de Comandos SQL


- **Conexão ao banco de dados é a primeira etapa á estabelecer uma conexão ao banco de dados usado por um cliente, aplicativo ou uma biblioteca de programação. A conexão geralmente é estabelecida por credenciais como nome de usuário e senha, para autenticação.**
- **Depois temos o envio do comando digitado e a inserção, logo apos isso temos a analise do comando para identificar a semantica e a sintaxe.**
- **No final de tudo temos o resultado dos comandos com o Select para mostrar em tela os resultados.**


- **CRUD**


- **Um CRUD é uma abreviação para as quatro operações básicas de manipulação de dados em um sistema de gerenciamento de banco de dados ou aplicação Web; CREATE (criação), READ (leitura), UPDATE (atualização), DELETE (exclusão). Em outras palavras, um CRUD se refere às funções quem permitem criar, ler, atualizar e excluir dados de um banco de dados ou sistemas. Geralmente é utilizado com uma GUI. É uma abordagem comum para operações de bancos de dados em muitos aplicativos e sistemas, permitindo a gestão de dados de forma eficiente.**

- **Exemplo de CRUD;**

My Contacts

 Add Contact

 Remove Contact

 Save All Modifications

NAME	PHONE #	EMAIL	BIRTHDAY
New Guy	(000) 000-0000	new@loianetest.com	01/01/2000
Contact0	(000) 000-0000	Contact0@asdfg.com	01/01/2000
Contact1	(000) 000-0001	Contact1@asdfg.com	01/01/2000
Contact2	(000) 000-0000	Contact2@asdfg.com	01/01/2000
Contact3	(000) 000-0000	Contact3@asdfg.com	01/01/2000
Contact4	(000) 000-0000	Contact4@asdfg.com	01/01/2000
Contact5	(000) 000-0000	Contact5@asdfg.com	01/01/2000
Contact6	(000) 000-0000	Contact6@asdfg.com	01/01/2000
Contact7	(000) 000-0000	Contact7@asdfg.com	01/01/2000
Contact8	(000) 000-0000	Contact8@asdfg.com	01/01/2000

Update

Cancel

January 2000

S	M	T	W	T	F	S
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Today

• Consulta Parametrizada

- Consulta parametrizada é um tipo de consulta SQL que requer pelo menos um parâmetro para execução. Um dos principais motivos para o uso de consultas parametrizadas é que elas tornam as consultas mais legíveis. O segundo e mais convincente motivo é que as consultas parametrizadas ajudam a proteger o banco de dados contra ataques de injeção de SQL.

- **Complexidade da C.P.**

- Normalmente quando pensamos em uma aplicação enviando consultas para execução no banco de dados, imaginamos algo extremamente simples e antiquado, em que a consulta inteira é um texto único. E essa visão simples é o que nos faz injetar os parâmetros da consulta (123) no corpo do comando (SELECT).
- Por exemplo:

© 2011 Blackwell Publishing Ltd *Journal of Internal Medicine* 270: 257–265

Downloaded from <http://ajph.org/> on November 10, 2015

|| Aplicação || ->--->- | SELECT ... WHERE id = '123' | ->--->- || Banco

[illegible]

Aplicando O "SELECT. WHERE"

• **Complexidade da C.P.**

- a realidade é muito mais complexa. Um fator é o contexto da sessão (como o estado da transação, configurações de sessão, objetos temporários...) que afeta tudo que é executado nela. E outro, tão importante quanto, é a estrutura completa da consulta, que pode ser enviada com muito mais informações que são usadas para transmitir parâmetros separados do corpo da consulta, assim como seus tipos de dados e nulidade.

- O exemplo a seguir mostra algumas dessas informações (em uma representação parecida com json por familiaridade, mas a comunicação costuma ser feita por protocolos de rede dedicados a esse propósito):

- Exemplo

```
┌──────────┐ ->--->- ┌{"q": "SELECT ... WHERE id = $1",┐ ->--->-
└──────────┘         └────────────────────────────────┘
┌──────────┐         ┌────────────────────────────────┐
|| Aplicação ||      || Banco
de dados ||         ||
└──────────┘ ->--->- ┌"params": [ { valor: "123",      ┐ ->--->-
└──────────┘         └────────────────────────────────┘
                        tipo: "integer",
                        null: false } ] }
└────────────────────────────────┘
```

• Complexidade da C.P.

- Dessa forma, mesmo que seja usado um parâmetro com valor malicioso (por exemplo, ' or '1' = '1'), ele não corre o risco de ser interpretado como parte do comando, já que a) não foi injetado no corpo do comando; b) o tipo de dados e nulidade podem ser informados e usados como validação extra. Isso é chamado de parametrização de consultas e é a forma mais confiável de evitar injeções de SQL. Em alguns SGBDs, consultas parametrizadas podem ser até mais rápidas que consultas de texto simples por economizar parte do tempo de compilação que antes era desperdiçado com o parse dos parâmetros injetados. Contudo, existem alguns pontos muito importantes que precisamos garantir, mesmo parametrizando as consultas.

• Conclusão da C.P.

Além de consultas parametrizadas, muitas outras estratégias também devem ser usadas, como whitelist de valores, análise de código da aplicação, análise de logs do servidor de banco de dados, auditoria, uso de valores com tipos de dados, entre outros. Concatenação de texto nunca deveria ser usada com valores, mas sim deixada apenas para casos em que for estritamente necessária, como para adicionar cláusulas a mais na estrutura da consulta. Por fim, injeções de dados devem ser tratadas com muita atenção durante o desenvolvimento do software, que deve passar por rigorosas validações de segurança em todos os componentes. Injeções são mais um espectro que nunca deixará de assombrar o mundo da tecnologia.

• SQL Injection

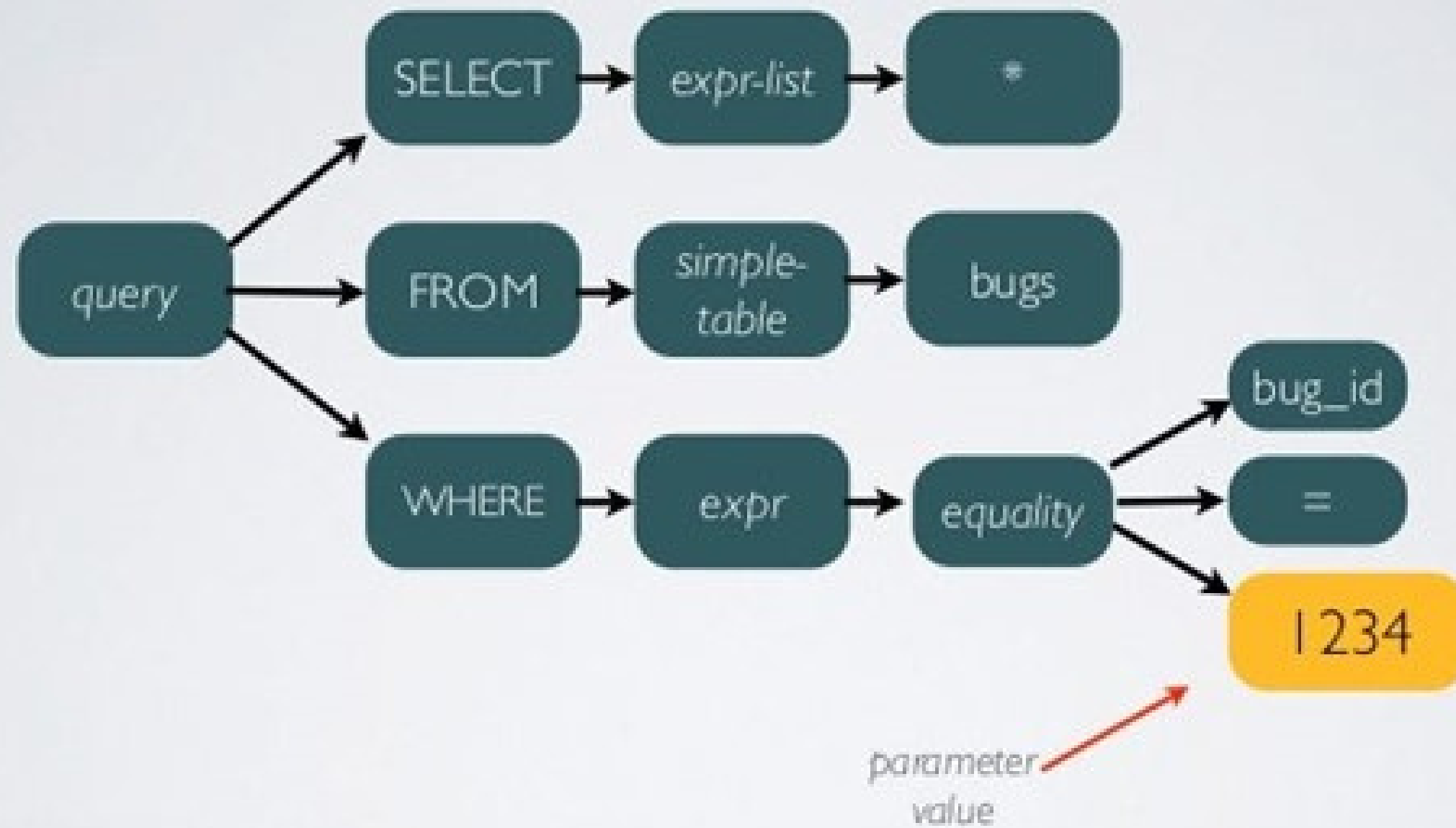
A injeção de dados mais popular e a que mais nos interessa agora é a injeção de SQL. Ela acontece quando um comando SQL é construído concatenando textos, especialmente quando isso inclui dados de origem insegura, causando o vazamento de dados ou a execução de outros comandos inesperados.

Por exemplo, se o endpoint de API `http://example.com/app/tabelaView?id=123` tiver a seguinte consulta no backend:

```
String query = "SELECT * FROM tabela WHERE ID='" + request.getParameter("id") + "'";
```

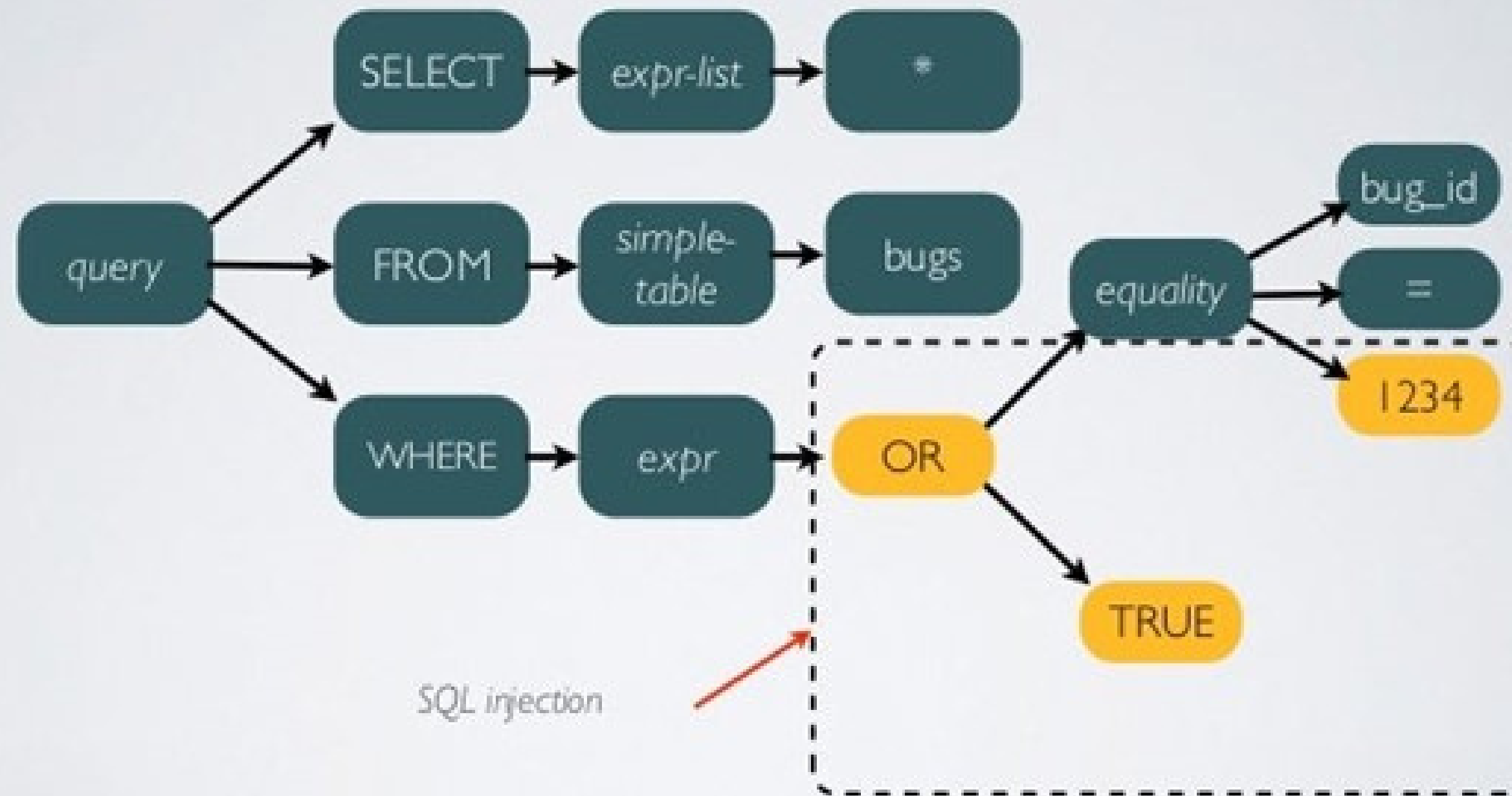
Que gera o plano de execução similar a ->

HOW THE DATABASE EXECUTES IT



- Então essa API é suscetível a requisições maliciosas como `http://example.com/app/tabelaView?id=' or '1'='1` que, pela concatenação acima, gera o comando abaixo:
- `SELECT * FROM tabela WHERE ID=" or '1'='1'";`
- O plano de execução é similar e retornará a tabela completa:

INTERPOLATION



• Sanitização

- A Sanitização refere-se a um processo de segurança que envolve a remoção segura de dados sensíveis ou confidenciais de dispositivos eletrônicos, como computadores, servidores smartphones, dispositivos de armazenamento e entre outros, antes de serem descartados, reutilizados ou transferidos para terceiros. A Sanitização no TI tem como objetivo proteger a privacidade e a segurança das informações armazenadas nos dispositivos, garantindo a seguridade dos dados, e que eles seja apagados e não possam ser recuperados de formas indevidas.
- A Sanitização pode ser realizada por métodos físicos (como a destruição física dos dispositivos) ou por métodos lógicos (como a formatação segura ou a sobrescrição dos dados) em conformidade com as normas e regulamentações específicas, visando minimizar o risco de vazamento de informações sensíveis e proteger a integridade dos dados

- **Formas de evitar sql injection**

- **Mitigações**

Existe uma longa lista de ações de mitigação que são usadas para evitar injeções de SQL. Mas é de vital importância compreender quando usar e quando não usar cada uma delas, já que algumas falham em proteger os sistemas contra injeções de dados, outras aumentam a complexidade do seu código, algumas introduzem bugs tão perigosos quanto as injeções que tentam evitar e ainda outras causam uma falsa sensação de segurança ao mover a responsabilidade para outro componente (normalmente menos confiável). Em resumo, é importante dissolver vários dos mitos que cercam as injeções de SQL. Então vamos ver aqui apenas algumas das mitigações mais comuns.

- **Formas de evitar sql injection**

- **Quoting e Escaping**

Quoting é a ação de adicionar aspas simples, duplas ou outros limitadores de texto em torno do dado que está sendo concatenado, na tentativa de que ele seja interpretado corretamente pelo SGBD. Como visto no exemplo da API, mesmo com aspas simples em torno do campo injetado é possível contornar esse mecanismo quando o dado contém aspas. Portanto, essa estratégia não é uma mitigação válida contra injeção de SQL.

• Formas de evitar sql injection

- Quoting e Escaping

Escaping é a ação de adicionar caracteres especiais para que as aspas do dado não sejam interpretadas como fim do dado, por exemplo tornando ' em \' e \ em \\. Com isso, a consulta anterior não mais retornaria o conteúdo completo da tabela, mas sim um erro de tipo de dados ou uma resposta vazia:

```
SELECT * FROM accounts WHERE custID='\' or \'1\'=\'1'';
```

Então aparentemente o problema foi solucionado através do uso simultâneo de quoting e escaping. Contudo, essa solução não é ideal, por diversas razões.

- **XSS (Cross-site Scripting)**

- **XSS (Cross-site Scripting) é uma vulnerabilidade de segurança na web que permite que um invasor injete scripts maliciosos em paginas web visualizadas por outros usuários. Esses scripts pode ser executados nos navegadores dos usuários vitimas, permitindo que o invasor roube informacoes sensíveis, como senhas, ou realize atividades maliciosas em nome de outros usuários.**

- **XSS (Cross-site Scripting)**

- **O XSS ocorre quando a entrada de um usuário não é devidamente validada ou sanitizada antes de ser exibida em pagina web, permitindo que o código malicioso seja executado no contexto do site vulnerável, levando a possíveis ataques comprometendo a segurança. Para proteger contra XSS, é fundamental aplicar corretamente a validação e sanitização de entrada de usuário e implementar boas práticas seguras de codificação na web.**

Obrigado Pela Atenção

Grupo 3 - Natalia / Renato / Renan / Raul