Triple “C”

Curtiu? Comente e Compartilhe!

Assine nosso canal e siga as novidades!

YouTube

Efésios 2:10 - Porque somos criação de Deus realizada em Cristo Jesus para fazermos boas obras

Hash Join – Dentro do Código do SQL Server

Thiago [TC] Alencar

https://www.linkedin.com/in/alencardba/

Wilson Neto

https://www.linkedin.com/in/wilsonnetobr/

Objetivo

- Cobrir o algoritmo de HASH e sua estrutura de dados.
- Apresentar a estrutura de HASH (internamente) no SQL Server
- Demonstrar como implementar essa estrutura/algoritmo via C#

Agenda

- Algoritmo
- Estrutura de dados
- Hashing
- Joins no SQL Server
- Hash Join
- Demo

O que é algoritmo?

Algoritmos são sequências finitas e bem definidas de instruções para resolver um problema.

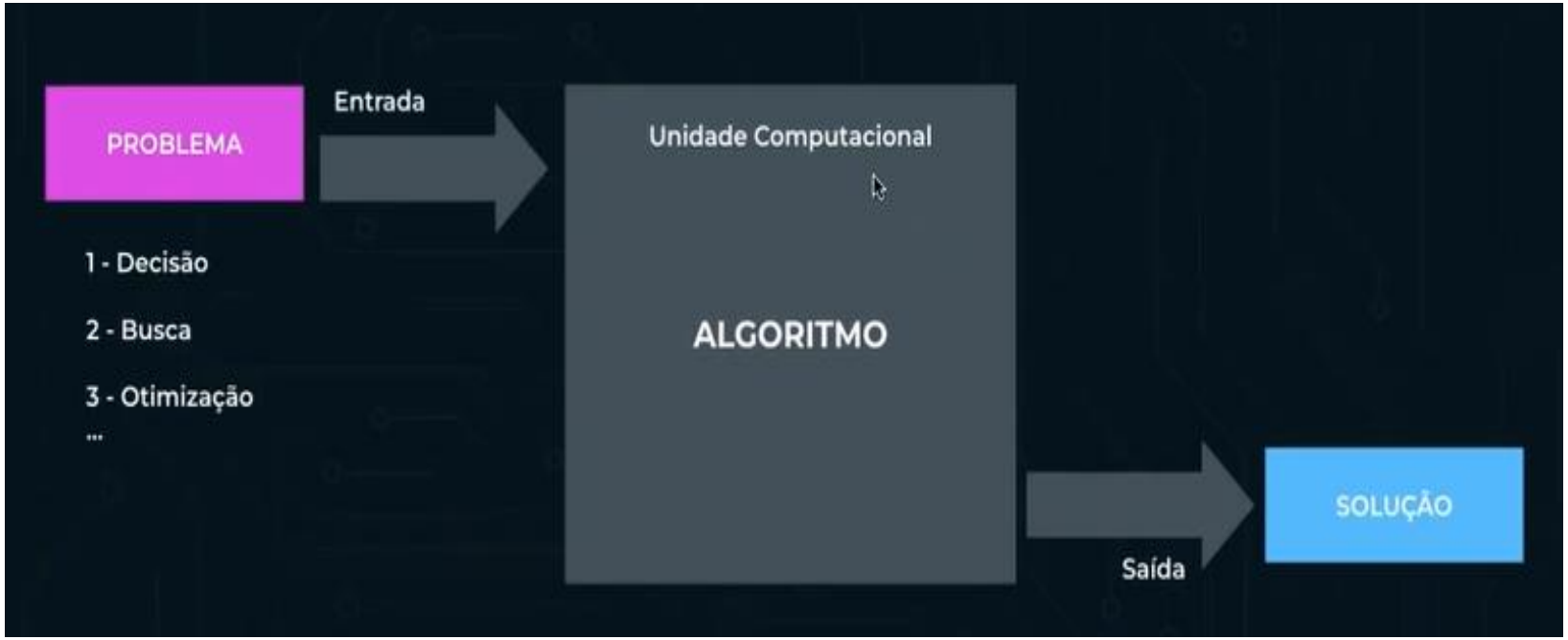
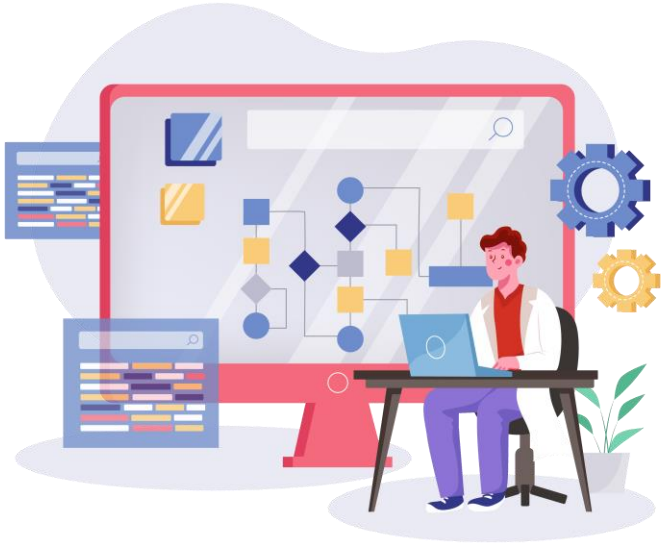


Imagem retirada do conteúdo do Elemar Jr

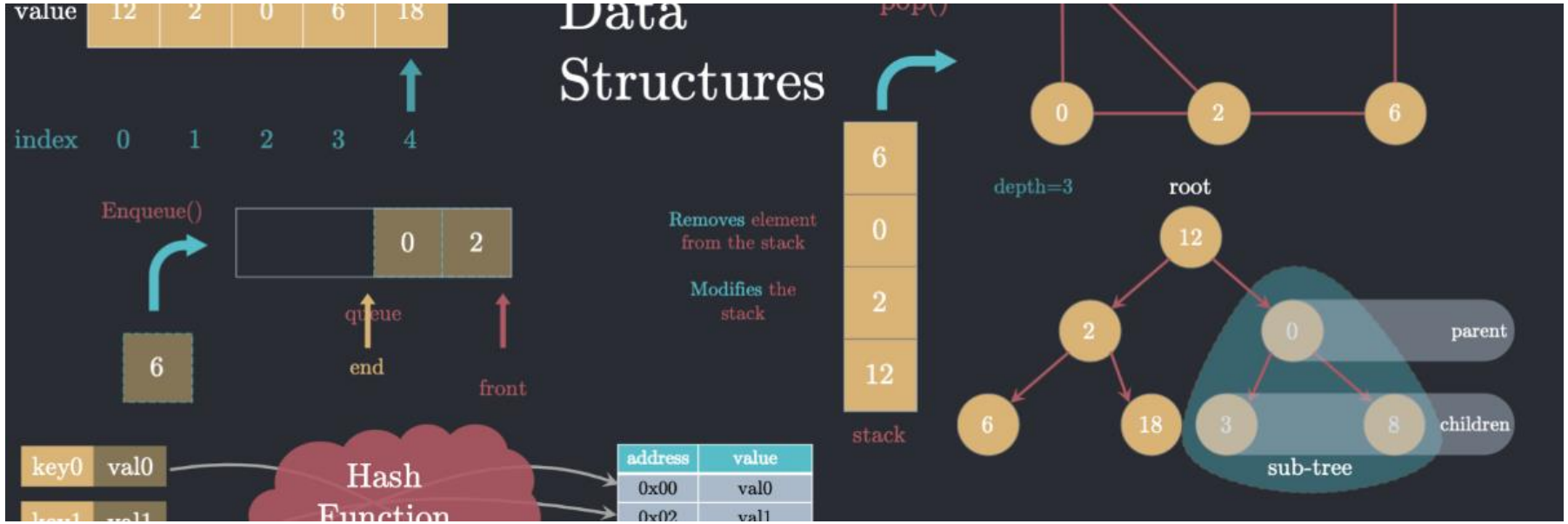
O que é algoritmo?

Algoritmos são fundamentais para a programação e para a Ciência da Computação porque descrevem os passos exatos que um computador deve seguir para alcançar um objetivo específico. A eficiência de um algoritmo pode ser medida em termos de tempo de execução ou espaço utilizado.



Estrutura de dados

Estruturas de dados são formas organizadas e sistemáticas de armazenar e organizar dados para que possam ser acessados e modificados de maneira eficiente. Elas são cruciais porque definem a maneira como os dados são armazenados, e o acesso ou a busca por estes dados depende da estrutura escolhida. Exemplos comuns incluem listas, pilhas, filas e árvores.



Estrutura de dados

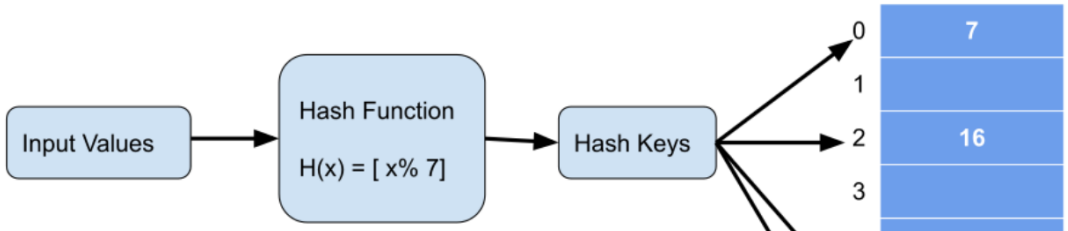
Muito importante conhecer e saber quando usar cada estrutura de dados

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Imagem do site bigocheatsheet.com

Hashing

Hashing é uma técnica de espalhamento de dados, onde um valor é mapeado para uma posição em uma **tabela hash**, fazendo com que a busca deste dado seja extremamente otimizada, sendo realizada em complexidade constante, O(1). Este mapeando é na feito por uma **função de hashing**.

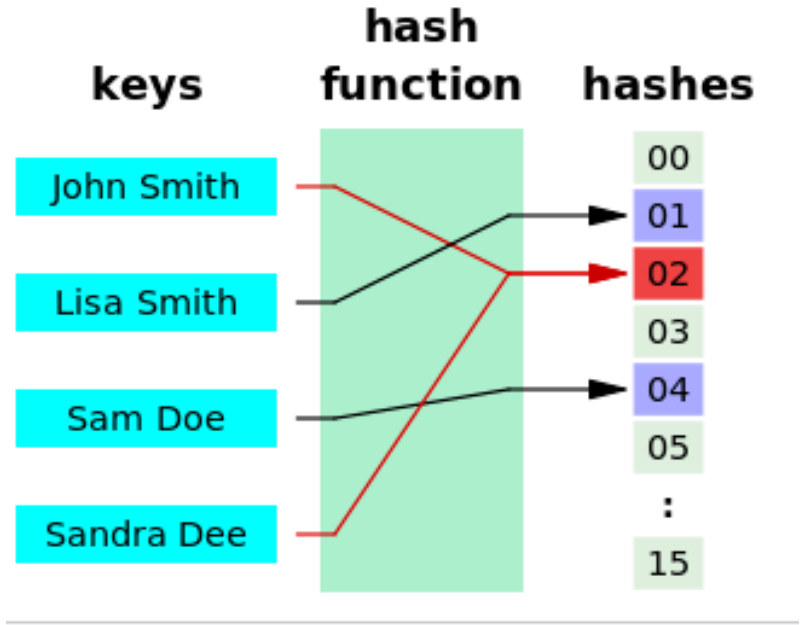


Inputs	Hash Function	Hash Keys
7	=7 % 7	0
16	=16 % 7	2
33	=33 % 7	5
42	=42 % 7	6

Função de Hashing

Uma função de hashing é uma função que converte um dado em um valor de tamanho fixo (geralmente um número). Esse valor resultante é chamado de "hash".

Uma boa função de hashing deve:
- Ser determinística
- Distribuir os valores uniformemente
- Ser rápida
- Minimizar colisões



Hash Table

Uma HashTable, ou Tabela Hash, é uma estrutura de dados que implementa uma array associativa, um tipo de estrutura que pode mapear chaves exclusivas para valores específicos. A principal característica de uma HashTable é sua capacidade de acessar seus elementos de forma extremamente rápida, independentemente do número de itens armazenados.

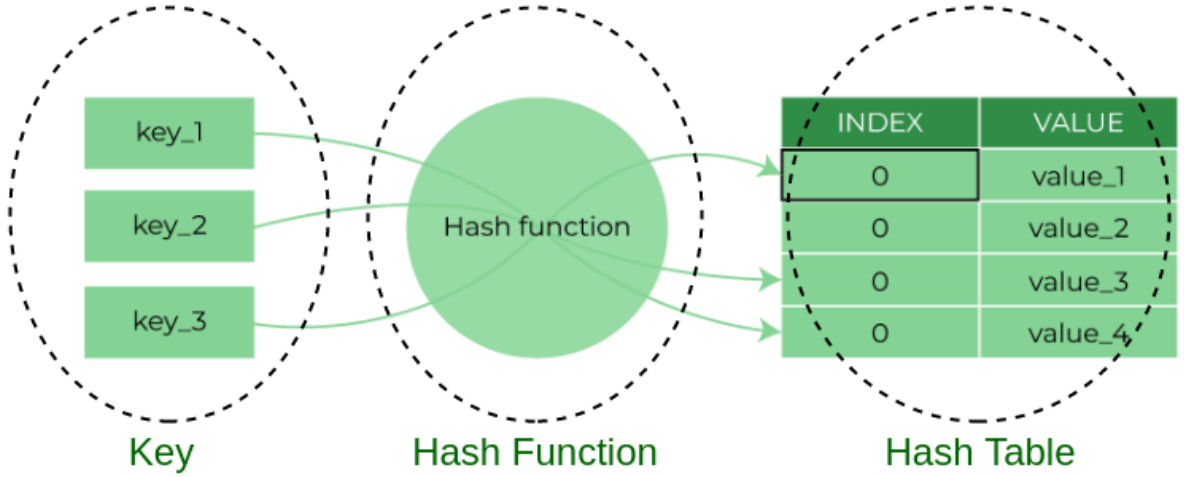


Imagem do site GeeksForGeeks

Tipos de Join no SQL

- Joins físicos, implementam operações lógicas

- Nested loop join - -

- Merge Join –

- Hash Join –

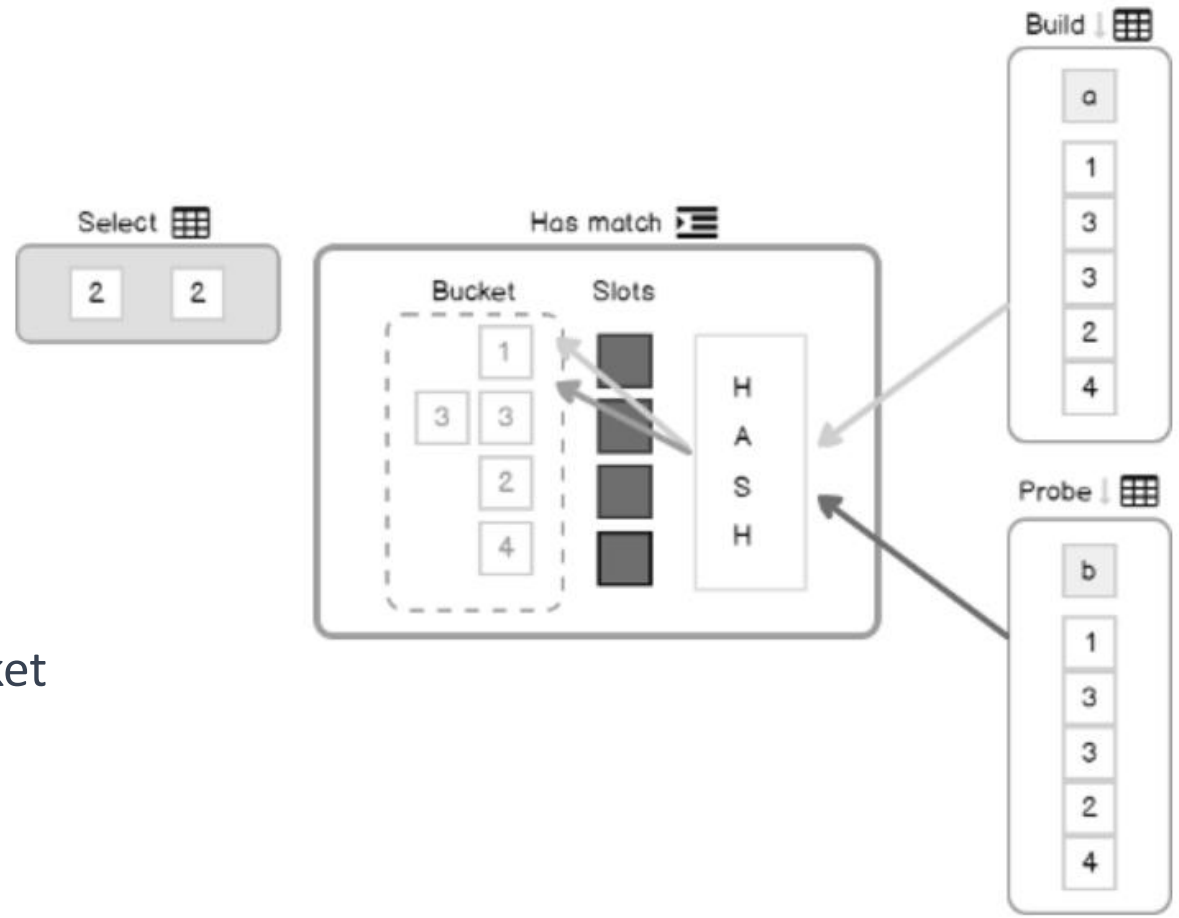
Hash Join – pseudo código

Build Table

```
for each row R1 in the build table
Begin
        calculate hash value on R1 join key(s)
        insert R1 into the appropriate hash bucket
End
```

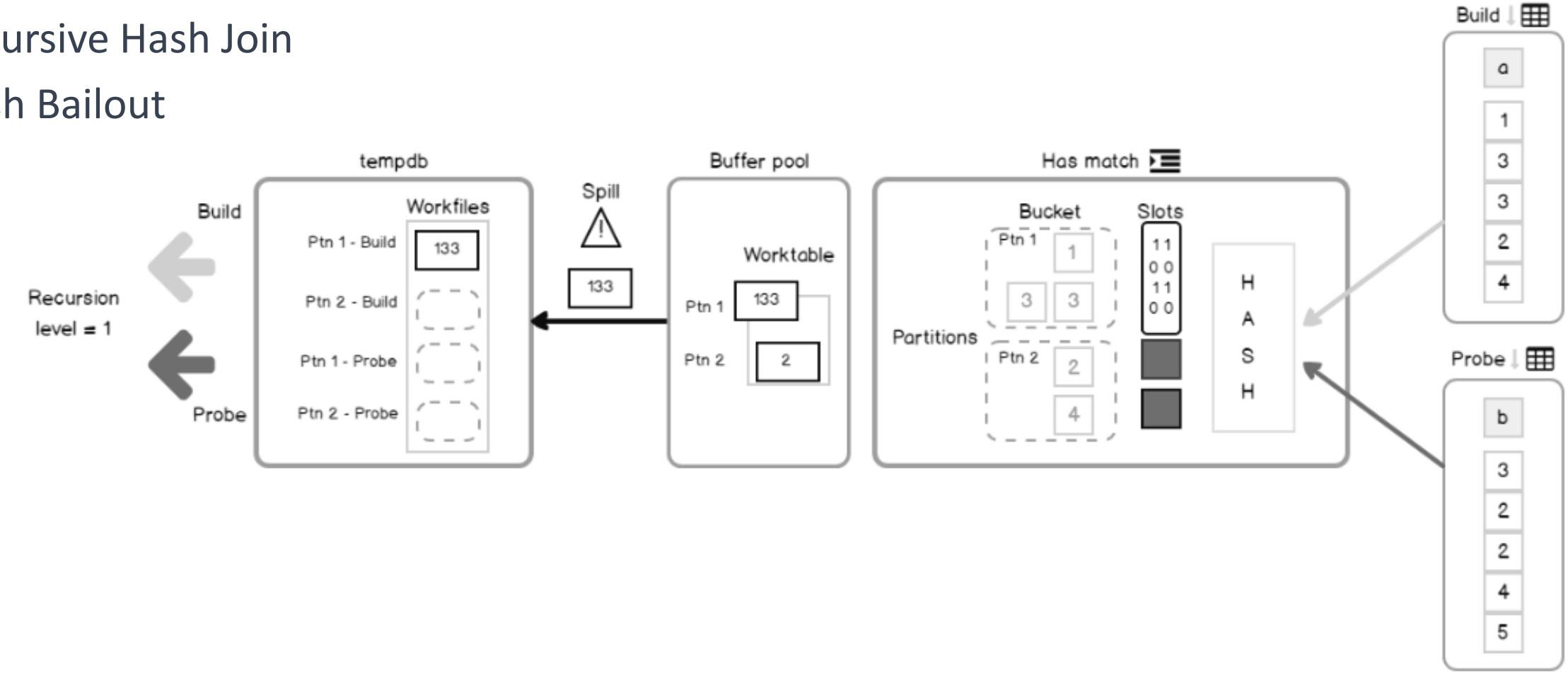
Proble table + Hash Join

```
for each row R2 in the probe table
begin
        calculate hash value on R2 join key(s)
        for each row R1 in the corresponding hash bucket
        if R1 joins with R2
                output (R1, R2)
 end
```



Hash Join

- In-Memory Hash Join
- Grace Hash Join
- Recursive Hash Join
- Hash Bailout



Stack

sqlmin!CQScanHash::Open
sqlmin!CQScanHash::ConsumeProbe
sqlmin!CQScanHash::Open
sqlmin!CQScanSortNew::BuildSortTable
sqlmin!CQScanSortNew::OpenHelper
sqlmin!CQueryScan::StartupQuery
sqllang!CXStmtQuery::SetupQueryScanAndExpression
sqllang!CXStmtQuery::InitForExecute
sqllang!CXStmtQuery::ErsqExecuteQuery
sqllang!CXStmtSelect::XretExecute
sqllang!CMsqlExecContext::ExecuteStmts<1,1>
sqllang!CMsqlExecContext::FExecute
sqllang!CSQLSource::Execute

ntdll!NtMapUserPhysicalPagesScatter
kernel32!MapUserPhysicalPagesScatter
sqldk!SOS_MemoryBlockAllocator::CommitBlockAndCheck NumaLocality
sqldk!SOS_MemoryBlockAllocator::AllocateBlock
sqldk!SOS_MemoryWorkSpace::AllocatePage
sqldk!MemoryNode::AllocatePagesInternal
sqldk!MemoryClerkInternal::AllocatePagesWithFailureMode
sqldk!MemoryClerkInternal::AllocatePages
sqldk!MemoryObjectFactory::CreateMemObjectInt
sqldk!MemoryObjectFactory::CreateMemObject
sqlmin!CHashBitmap::Init
sqlmin!CQScanHash::PickInputAndPrepareOutput
sqlmin!CQScanHash::Open
sqlmin!CQScanHash::ConsumeProbe

DEMO