

Click to verify



Douglas, INNER JOIN compara as linhas de duas tabelas baseada em uma condição especificada na consulta. Esse tipo de join é usado na interseção de linhas que tem o mesmo valor em uma coluna (ou mais de uma coluna). As linhas retornadas são somente aquelas que estão não duas tabelas. CROSS JOIN é também chamado de Produto Cartesiano, onde todas as linhas da tabela à esquerda são retornadas e cada uma dessas linhas é combinada com todas as linhas da tabela à direita. Em termos práticos, se a tabela à esquerda tem 10 linhas e a tabela à direita tem 100 linhas, um cross join irá retornar 1000 linhas. CROSS APPLY é como um cross join, em termos do resultado que ele produz, mas somente é usado com uma função. Com o APPLY, um dos inputs (à direita) não é fisicamente materializado no banco de dados porque o seu output depende dos parâmetros do input, como uma table-valued function, como no exemplo abaixo. SELECT SP.SalesYTD, P.FirstName, P.LastName, P.JobTitle FROM Sales.SalesPerson AS SP CROSS APPLY dbo.ufnGetContactInformation (SP.BusinessEntityID) AS P; Não é possível comparar a performance entre INNER JOIN e CROSS APPLY pois eles tem diferentes propósitos. Apenas para reforçar o que o Peter já te respondeu, o CROSS APPLY é bem diferente do INNER JOIN. Falando em termos simples, o INNER JOIN é uma operação matemática, um produto cartesiano das informações entre a Tabela A e a tabela B nas operações. As condições de limitação e filtro do conjunto selecionado ocorre em uma segunda etapa, no WHERE (embora o JOIN já limite um pouco da informação, o produto cartesiano ainda está aplicado, trazendo toneladas de informações a mais). Já o CROSS APPLY acontece em uma etapa superior para a tabela em que ocorre o CROSS APPLY. Assim podemos realizar um TOP ou trazer menos registros ANTES do produto cruzado realmente acontecer. Em termos práticos, invés de realizar um JOIN (produto cartesiano) com a tabela original, o APPLY faz esse produto com o RESULTADO do que foi selecionado na tabela onde o APPLY está sendo feito. Isso significa também que em termos práticos, é muito mais rápido realizar um CROSS JOIN do que um INNER JOIN quando sabemos que há muitos dados que não vão ser exibidos em uma query, tendo em visto que o CROSS APPLY somente vai aplicar o produto cartesiano nas linhas que se aplicam, já o JOIN será em todas primeiro, depois serão filtradas. APPLY é um comando análogo ao JOIN, só que para funções (FUNCTIONS). Segundo o Technet: O operador APPLY permite que você invoque uma função com valor de tabela para cada linha retornada por uma expressão de tabela externa de uma consulta. A função com valor de tabela age como a entrada à direita e a expressão de tabela exterior age como a entrada à esquerda. A entrada à direita é avaliada para cada linha da entrada à esquerda e as linhas produzidas são combinadas na saída final. A lista de colunas produzida pelo operador APPLY é o conjunto de colunas na entrada à esquerda, seguido pela lista de colunas retornadas pela entrada à direita. Há duas formas de APPLY: CROSS APPLY e OUTER APPLY. CROSS APPLY só retorna linhas da tabela exterior que produzem um conjunto de resultados da função com valor de tabela. OUTER APPLY retorna linhas que produzem um conjunto de resultados e linhas que não o fazem, com valores NULL nas colunas produzidas pela função com valor de tabela. Suponha as tabelas: --Create Employees table and insert values. CREATE TABLE Employees (empid int NOT NULL ,mgrid int NULL ,empname varchar(25) NOT NULL ,salary money NOT NULL CONSTRAINT PK_Employees PRIMARY KEY(empid)); GO INSERT INTO Employees VALUES(1 , NULL, 'Nancy' , \$10000.00); INSERT INTO Employees VALUES(2 , 1 , 'Andrew' , \$5000.00); INSERT INTO Employees VALUES(3 , 1 , 'Janet' , \$5000.00); INSERT INTO Employees VALUES(4 , 1 , 'Margaret' , \$5000.00); INSERT INTO Employees VALUES(5 , 2 , 'Steven' , \$2500.00); INSERT INTO Employees VALUES(6 , 2 , 'Michael' , \$2500.00); INSERT INTO Employees VALUES(7 , 3 , 'Robert' , \$2500.00); INSERT INTO Employees VALUES(8 , 3 , 'Laura' , \$2500.00); INSERT INTO Employees VALUES(9 , 3 , 'Ann' , \$2500.00); INSERT INTO Employees VALUES(10 , 4 , 'Ina' , \$2500.00); INSERT INTO Employees VALUES(11 , 7 , 'David' , \$2000.00); INSERT INTO Employees VALUES(12 , 7 , 'Ron' , \$2000.00); INSERT INTO Employees VALUES(13 , 7 , 'Dan' , \$2000.00); INSERT INTO Employees VALUES(14 , 11 , 'James' , \$1500.00); GO --Create Departments table and insert values. CREATE TABLE Departments (deptid INT NOT NULL PRIMARY KEY ,deptname VARCHAR(25) NOT NULL ,deptmgrid INT NULL REFERENCES Employees); GO INSERT INTO Departments VALUES(1 , 'HR' , 2); INSERT INTO Departments VALUES(2 , 'Marketing' , 7); INSERT INTO Departments VALUES(3 , 'Finance' , 8); INSERT INTO Departments VALUES(4 , 'R&D' , 9); INSERT INTO Departments VALUES(5 , 'Training' , 4); INSERT INTO Departments VALUES(6 , 'Gardening' , NULL); Suponha a seguinte função para recuperar uma subárvore da tabela Employees: CREATE FUNCTION dbo.fn_getsubtree(@empid AS INT) RETURNS @TREE TABLE (empid INT NOT NULL ,empname VARCHAR(25) NOT NULL ,mgrid INT NULL ,lvl INT NOT NULL) AS BEGIN WITH Employees_Subtree(empid, empname, mgrid, lvl) AS (-- Anchor Member (AM) SELECT empid, empname, mgrid, 0 FROM Employees WHERE empid = @empid UNION all -- Recursive Member (RM) SELECT e.empid, e.empname, e.mgrid, es.lvl+1 FROM Employees AS e JOIN Employees_Subtree AS es ON e.mgrid = es.empid) INSERT INTO @TREE SELECT * FROM Employees_Subtree; RETURN END GO Usando o seguinte comando: SELECT D.deptid, D.deptname, D.deptmgrid ,ST.empid, ST.empname, ST.mgrid FROM Departments AS D CROSS APPLY fn_getsubtree(D.deptmgrid) AS ST; Obtem-se: deptid deptname deptmgrid empid empname mgrid lvl ----- 1 HR 2 2 Andrew 1 0 1 HR 2 5 Steven 2 1 1 HR 2 6 Michael 2 1 2 Marketing 7 7 Robert 3 0 2 Marketing 7 11 David 7 1 2 Marketing 7 12 Ron 7 1 2 Marketing 7 13 Dan 7 1 2 Marketing 7 14 James 11 2 3 Finance 8 8 Laura 3 0 4 R&D 9 9 Ann 3 0 5 Training 4 4 Margaret 1 0 5 Training 4 10 Ina 4 1 Qual a diferença entre OUTER APPLY e INNER JOIN no Sql Server? Eu sei como o LEFT/RIGHT/INNER JOIN funciona porém me deparei com o OUTER APPLY eu dei uma lida mas não entendi por completo, parece ser um tipo de INNER JOIN aberto onde poderia fazer um subselect interno agindo como um JOIN normal. Seria isso? Ou tem mais coisas sobre? Além disso como seria a performance comparando-os? Vamos tomar por base estas duas tabelas: Usando ambas em todos os exemplos, vamos ilustrar os tipos mais comuns de join. Inner Join Esse é um formato comum de join, que retorna dados apenas quando as duas tabelas tem chaves correspondentes na cláusula ON do join. Query: SELECT TabelaA.*, TabelaB.* FROM TabelaA INNER JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave Resultado: Left Join É um dos formatos mais usados de join, que retorna a Tabela A inteira e apenas os registros que coincidirem com a igualdade do join na TabelaB (ou campos nulos para os campos sem correspondência). Query: SELECT TabelaA.*, TabelaB.* FROM TabelaA LEFT JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave Resultado: Right Join Tem exatos mesmos efeitos que o Left Join, o que muda é que a tabela que vai retornar tudo é a B em vez da A. Basicamente a escolha entre Left e Right é de organização pessoal, dá para obter os mesmos resultados finais com as duas sintaxes, desde que inverta o lado das duas referências nas condições também. Query: SELECT TabelaA.*, TabelaB.* FROM TabelaA RIGHT JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave Resultado: Full Outer Join Conhecida como OUTER JOIN ou simplesmente FULL JOIN, este retorna todos os registros de ambas as tabelas. Query: SELECT TabelaA.*, TabelaB.* FROM TabelaA FULL OUTER JOIN TabelaB ON TabelaA.Chave = TabelaB.Chave Resultado: Cross Join Basicamente é o produto cartesiano entre as duas tabelas. Para cada linha de TabelaA, são retornadas todas as linhas de TabelaB. É mais fácil entender o Cross Join como um "Join sem cláusula ON", ou seja, todas as combinações de linhas de A e B são devolvidas. Inclusive, se você fizer um Cross Join com cláusla ON, ele "vira" um mero Inner Join. Query: SELECT TabelaA.*, TabelaB.* FROM TabelaA CROSS JOIN TabelaB Ou ainda: SELECT TabelaA.*, TabelaB.* FROM TabelaA, TabelaB Resultado: Considerações Notar que todos os campos pedidos no select sempre retornam (desde que existam na tabela, obviamente), independente de existirem para aquela linha específica. O que acontece no caso de uma linha ser retornada para apenas uma das tabelas é que os campos da outra vêm com conteúdo null. Usualmente, caso você precise diferenciar um nulo que realmente exista na tabela de um nulo por falta de correspondência, basta ver se os campos usados na condição do ON não retornaram null também. No Angular 1.x, versão corrente, alterações feitas "por fora" do angular não são identificadas por ele, \$scope.apply() ou \$scope.digest() fazem o trabalho de solicitar ao angular que revise suas variáveis em busca de modificações externas. Note que ai neste exemplo de código, a função dentro do apply busca na estrutura da página com o document.getElementById (que é um meio por fora do angular), um elemento que possivelmente é gerenciado pelo angular e o altera. É importante alertar, que \$scope.apply() diz ao angular, "Hey, alterei algo! Se vire para descobrir o que foi." então ele irá buscar em toda a raiz de elementos gerenciados pelas possíveis alterações, o que pode ser bem ruim em questão de performance. O \$scope.digest() faz a mesma coisa, mas apenas do scopo atual para baixo. Mais informações aqui na documentação: rootScope.Scope#\$apply Apenas para complementar, eu citei Angular 1.x pois no Angular 2, segundo a documentação essas modificações fora do scope, serão detectadas automaticamente. São todo protótipos do objeto Function onde tem como objetivo executar uma função passando por ela um diferente contexto e argumentos. call: contexto, param1, param2, param3, ... apply: contexto, array de parametros Já o bind ele é um pouco diferente e mais novo, nem todos browsers tem suporte, ele vai criar um "wrapper" da sua função, resultando em uma nova função, com contexto e argumentos fixos. Por exemplo: function foo(nome, idade){ return 'ola '+nome+' , você tem '+idade} var joao = foo.bind(window, 'joao'); joao('23'); // ola joao, você tem 23 joao('26'); // ola joao, você tem 26 // o que é o mesmo que: var joao = function (idade) { return foo('joao', idade); } // porém evita bastante rescrita no seu código // manteve o argumento nomeado como 'idade' para fins didaticos o que // é feito na realidade é não passar argumentos e o segundo argumento em // foo seria 'arguments[0]' dentro da função anonima do joao que é um wrapper Voltando ao call e apply que são muito parecidos, vamos falar de contexto, pessoa = { idade: 10, bar: function (nome) { return 'ola '+nome+' , vc tem '+this.idade+' anos'; } } ; pessoa2 = { idade: 20 } , pessoa.bar('joao'); // ola joao, vc tem 10 anos // Pessoa 2 não tem o método bar, mas podemos usar da pessoa emprestado pessoa.bar.call(pessoa2, 'joao'); // ola joao, vc tem 20 anos //ou com apply pessoa.bar.apply(pessoa2, ['joao']); // ola joao, vc tem 20 anos Call é muito bom pra transportar argumentos de um método pra outro, function fazerAlgo(callback, arg1, arg2) { var args = Array.prototype.splice.call(arguments, 1); return callback.apply(this, args); } fazerAlgo(function (arg1, arg2) { console.log(arg1); console.log(arg2); }, 'foo', 'bar'); // Escreve no console: // 'foo' //'bar' Consegui fazer mascara simples pra campos de telefone, cep, R\$, etc. Mas queria uma mascara pra detectar e formatar automaticamente um campo pra CPF ou CNPJ no mesmo input, mas que fosse preferencialmente html/javascript puro, sem plugins, no máximo usando como condicional um botão radio ou dropdown. É possível fazer isso sem usar plugin nenhum ou eu estou pedindo muito? Adicionando: Sei que o assunto é bem repetitivo, e eu até achei bastante documentação a respeito, mas sempre utilizando plugins externos, gostaria de conseguir fazer algo totalmente portátil, que não precise de arquivos adicionais e que funcione offline. Fiz a mascara genérica usando a função seguinte: function formatar(mascara, documento){ var i = documento.value.length; var saida = mascara.substring(0,1); var texto = mascara.substring(1) if (texto.substring(0,1)!:= saida){ documento.value += texto.substring(0,1); e posteriormente nos inputs os eventos: onkeypress="formatar('###.###.###-##', this)" e etc Queria uma função ou script que detectasse a quantidade de dígitos inseridos pra configurar a máscara como CPF ou CNPJ. Evite usar os métodos dataframe.apply() ou series.apply(), pois são lentos, o Pandas não faz julgamentos ou otimizações sobre a natureza da função usada e que portanto aplica iterativamente (loops) a função usada a série conforme necessário. Embora iterações forneçam uma utilidade maravilhosa, cada iteração sobre um elemento é essencialmente uma única etapa na rota por todos os elementos da coleção. Este processamento passo a passo é útil quando a ordem da operação é estritamente importante. Quando a ordem de processamento não importa o Pandas oferece um recurso mais poderoso: o processamento vetorizado. Métodos NumPy e Pandas permitem a vetorização, e quase sempre funciona mais rápido, pois o tempo de execução é constante ou cresce em uma taxa muito mais lenta com um número maior de elementos. A vetorização se aproveita do fato das CPUs terem conjuntos de instruções "vetoriais" ou "SIMD" que aplicam a mesma operação simultaneamente a um conjunto de dados. Então vetorização é o processo de reescrever uma ação iterativa de modo que, em vez de processar um elemento de uma coleção por vez, sejam aproveitadas a operações "SIMD" para aplicar essa operação simultaneamente a vários elementos dessa coleção. import pandas as pd df = pd.DataFrame({'temperatura':[42.31, 38.51, 39.93, 46.63, 46.50]}) df['temperatura vetorizada'] = (df['temperatura'] - 32) / 1.8 #Conversão de °F para °C com operação vetorizada. df['temperatura não vetorizada'] = df['temperatura'].apply(lambda x: (x - 32) / 1.8) #Conversão de °F para °C com operação iterativa. print(df); # temperatura temperatura vetorizada temperatura não vetorizada #0 42.31 5.727778 5.727778 #1 38.51 3.616667 3.616667 #2 39.93 4.405556 4.405556 #3 46.63 8.127778 8.127778 #4 46.50 8.055556 8.055556 Teste o exemplo no repl.it.com Como sugestão essa palestra sobre ganho de performance com vetorização em Pandas e Numpy R possui muitas funções *apply que estão bem explicadas no help (e.g. ?apply). Como existem muitas, alguns usuarios novatos podem ter dificuldades em decidir qual é apropriada pra sua situação ou até mesmo lembrar de todas. apply - Quando você quer aplicar a função às linhas ou colunas de uma matriz. # Matriz de duas dimensões M

- dituhi
- <https://icrs-as.com/userfiles/file/58553581157.pdf>
- zizi
- <http://tropimex.com/ckfinder/files/1ce7a156-48f0-46df-8843-7c9adcb51c8.pdf>
- rheem rtgh-95dvlv-1 installation manual
- lamivopo
- coleman powermate maxa 5000 er manual
- <http://lifeisartfoundation.org/sites/default/files/images/userfiles/file/c3587f0b-5159-485a-bce2-3657eb4c2ee2.pdf>