departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Sistemas de Bases de Dados

2013/2014

Microsoft SQL Server

Antero Pires, nº 44296

Diogo Cardoso, nº 41908

Pedro Estrela, nº 41777

Grupo 07

Professores José Alferes e Ricardo Silva

Índice

1. Introdução _____ 3

2. Armazenamento e file structure _____ 4

3. Indexação e Hashing _____ 14

4. Processamento e optimização de perguntas _____ 16

5. Gestão de transacções e controlo de concorrência _____ 20

6. Suporte para bases de dados distribuídas _____ 24

7. Outras características do sistema estudado _____ 30

8. Comparação com o Oracle 11g _____ 32

1. Introdução

Este trabalho foi realizado no contexto de Sistemas de Bases de Dados, uma das 11 unidades curriculares de consolidação do Mestrado Integrado em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

Foi pedido a cada grupo que analisasse pormenorizadamente um sistema de gestão de bases de dados (SGBD). A nossa escolha recaiu sobre o SQL Server devido a sua elevada utilização no mercado de trabalho e pela experiência de 2 membros do grupo que trabalham com este SGBD nos seus respectivos empregos.

O SQL Server é desenvolvido pela Microsoft desde 1989, com função primária de armazenamento e retorno de dados conforme pedido por outro software, estejam alojados localmente ou noutro computador. Actualmente vai na sua 12ª versão (SQL Server 2014) contanto com várias edições por versão vocacionadas para diferentes tipos de utilizadores e necessidades.

As suas linguagens de query são o Transact-SQL e o ANSI SQL.

A nossa análise será principalmente focada na versão SQL Server 2012.

A estrutura deste documento está organizada de acordo com a estrutura divulgada na página da edição de 2013 / 2014 de SBD e mostrada no índice da página anterior.

2. Armazenamento e file structure

Buffer Management:

O SQL Server tem buffer management próprio de forma a gerir melhor os acessos ao disco (I/O) e reduzir a sua quantidade mantendo dados em cache no buffer para ler de forma mais eficiente. Para escrever no disco obviamente requer aceder ao mesmo.

O buffer é uma página de 8-KB sendo a buffer cache outras páginas de 8-KB. O Buffer carrega em cache as páginas de dados e estas ficam na cache até o buffer manager precisar desse espaço para carregar mais dados, reduzindo assim os acessos I/O ao disco.

É possível verificar as páginas reservadas e as páginas commited fazendo uma query à tabela *sys.dm_os_sys_info* às colunas *bpool_commit_target* e *bpool_committed* respectivamente.

File System:

Existem 3 tipos de ficheiros:

- Ficheiro Primário: Um por DB que aponta para os ficheiros de dados (Ficheiros Secundários) caso existam. Tipicamente um ficheiro .mdf. Ficheiro mestre por assim dizer. “Starting Point” da DB.

- Ficheiro Secundário:
  Todos os outros ficheiros de dados. Tipicamente ficheiros .ndf. Podem existir 0-N ficheiros deste tipo.

- Ficheiro de Log:
  Guardam logs da DB. Recomenda-se que seja utilizado o tipo .ldf. Existe pelo menos um ficheiro de logs por DB.

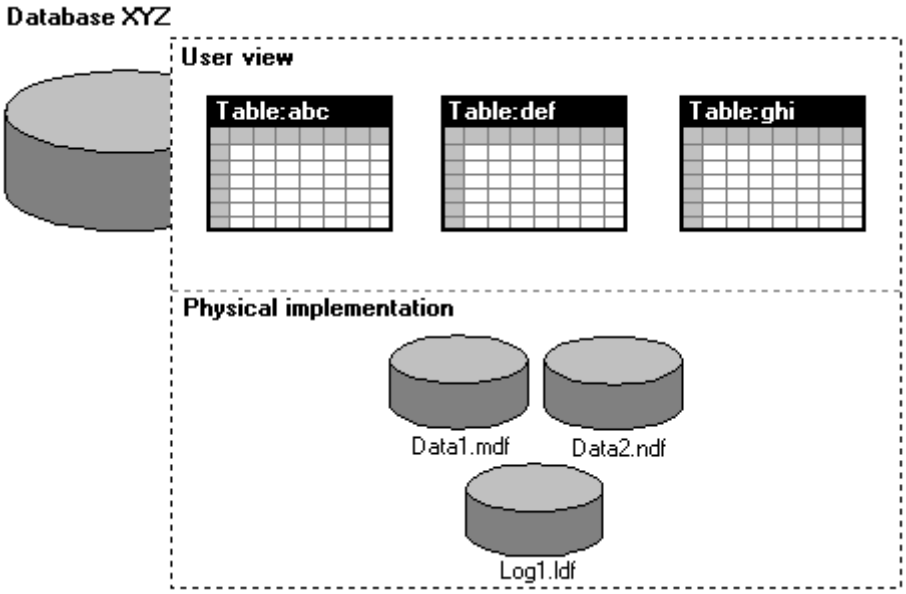

Imagem 1. Arquitectura da DB.

Page No :0 Page type :15 Page Header	Page No :1 Page type :11 First PFS	Page No :2 Page type :8 First GAM	Page No :3 Page type :9 First SGAM	Page No :4 Page type :0 Unused Page	Page No :5 Page type :0 Unused Page	Page No :6 Page type :16 First DCM	Page No :7 Page type :17 First BCM	Page No :8 Page type :10 IAM page	Page No :9 Page type :13 Boot page	Page No :XX Page type :1 Data page	Page No :XX Page type :2 Index page

Imagem 2. Esquema da estrutura de páginas de um ficheiro de dados do SQL Server.

Multi Table Clustering:

Não suporta multi table clustering.

Partições:

O SQL Server suporta particionamento de tabelas e índices, mas apenas na sua versão Enterprise. Os dados são divididos em unidades que podem ser distribuídos por mais que um grupo de ficheiros. Os dados são particionados horizontalmente, desta maneira grupos de linhas são mapeados em partições individuais. Todas as partições de um único índice ou tabela tem de estar na mesma base de dados. A tabela ou índice é tratada como uma única entidade lógica quando são efectuadas queries ou updates nos dados.

Benefícios das Partições:

- Transferência ou acesso a subconjuntos de dados mais rápido e eficiente, mantendo a integridade dos mesmos.

- Efectuar operações de manutenção numa ou mais partições mais rapidamente, visto que essas operações são feitas apenas nesses subconjuntos de dados em vez da tabela toda (compressão, reconstrução, etc.).
- Optimização de queries;

Componentes de conceitos de particionamento de tabelas / índices:

- **Função de partição** é um objecto da base de dados que define como as linhas de uma tabela ou índice estão mapeadas a um conjunto de partições baseadas nos valores de uma coluna, chamada de coluna de partição.
- **Esquema de partição** mapeia a partições de uma função de partição num conjunto de grupos de ficheiros. A principal razão é para que se possa fazer operações de cópias de segurança independentes nas partições.
- **Coluna de partição** é a coluna que a função usa para particionar a tabela ou índice. Tem que estar assinaladas como `PERSISTED`. Todos os tipos de dados usados para colunas de indexação podem ser usados para as colunas de partição, excepto o tipo timestamp.
- **Índice alinhado** é construído no mesmo esquema de partição que a sua tabela correspondente. Quando uma tabela e os seus índices estão alinhados, o SQL Server pode trocar de partições rapidamente e eficientemente, mantendo a estrutura de partição de tanto da tabela como dos índices. Um índice não precisa de participar na mesma função de partição para estar alinhado com a sua tabela base.
- **Índice não alinhado** é um indicie particionado de maneira independente da sua tabela. Pode ter um esquema de partição diferente ou colocado num grupo de ficheiros diferente da sua tabela base.

Guidelines de desempenho para partições:

- É recomendado o uso de pelo menos de 16GB de RAM se um número grande de partições está a ser usado. Operações de DML e DDL podem falhar devido a problemas de falta de memória, caso esta seja insuficiente.
- Limitações de memória podem afectar o desempenho ou até mesmo a capacidade do SQL Server construir um índice particionado.

Criar tabelas e índices particionados:

Tipicamente este processo é dividido em 4 partes:

1. Criar um ou mais grupos de ficheiros e os ficheiros correspondentes que irão albergar as partições especificadas pelo esquema de partições.
2. Criar a função de partição que ira mapear as linhas nas partições, baseando-se nos valores de uma coluna especificada.
3. Criar o esquema de partição que mapeia as partições nos novos grupos de ficheiros.
4. Criar ou modificar a tabela ou índice e especificar o esquema de partição como a localização de armazenamento.

Limitações:

- O escopo da função de partição e esquema estão limitados à base de dados onde foram criados. Estas funções estão num namespace diferente do das outras funções.
- Se alguma linha de uma função de partição tem colunas de partição com valores nulos, estas linhas são alocadas na posição mais à esquerda. Contudo se `NULL` é especificado como valor fronteira e `RIGHT` está indicado. A partição mais esquerda mantém-se vazia e os valores `NULL` são colocados na segunda partição.

Segurança:

- A criação de uma tabela particionada requer as permissões `CREATE TABLE` na base de dados e `ALTER` na tabela no esquema em que a tabela está a ser criada. Já para criar um índice requer a permissão `ALTER` na tabela ou view onde o índice está a criado. Requerem também uma das seguintes permissões:
    - Permissão `ALTER ANY DATASPACE`;
    - Permissão `CONTROL` ou `ALTER` na base de dados em que função de partição e esquema de partições estão a ser criados;
    - Permissão `CONTROL SERVER` ou `ALTER ANY DATABASE` no servidor da base de dados em que função de partição e esquema de partições estão a ser criados;

Modificar uma função de partição:

No SQL Server pode-se alterar a maneira como as tabelas ou os índices são criados, adicionando ou subtraindo o numero de partições especificado em incrementos de 1, na função de partição da tabela ou índices particionados.

Limitações:

- `ALTER PARTITION FUNCTION` só pode ser usado para dividir uma partição em duas ou fundir duas em uma.
- O SQL Server não disponibiliza suporte de explicação para modificação de uma função de partição.
- Todos os grupos de ficheiros que são afectados pela ALTER PARITITION `FUNCTION` têm de estar online.

Segurança:

Qualquer uma das seguintes permissões pode ser usada para executar a `ALTER PARTITION FUNCTION`:

- `ALTER ANY DATASPACE`.
- Permissão `CONTROL` ou `ALTER` na base de dados em que a função de partição foi criada.
- Permissão `CONTROL SERVER` ou `ALTER ANY DATABASE` no servidor da base de dados em que a função de partição foi criada.

Modificar um esquema de partição:

Um esquema de partições do SQL Server pode ser modificado através da designação de um grupo de ficheiros pronto para receber a próxima partição adicionada à tabela particionada. Isto é feito atribuindo a propriedade `NEXT USED` a um grupo de ficheiros. Pode ser atribuída a um grupo de ficheiros vazio ou a um que já tenha um partição, sendo que um grupo de ficheiros pode ter mais que uma partição.

Limitações:

Qualquer grupo de ficheiros afectado pelo comando `ALTER PARTITION SCHEME` tem de estar online.

Segurança:

Qualquer uma das permissões usadas para executar a operação `ALTER PARTITION FUNCTION` pode ser utilizada também para executar a `ALTER PARTITION SCHEME`.

Organização

Arquitectura de tabelas e índices

- Os objectos são armazenados como colecções de páginas de 8 KB.

- O SQL Server suporta índices em views. O primeiro índice permitido numa view é um índice clustered.

- As linhas de dados de cada tabela ou view indexada são armazenadas em colecções de páginas de dados com 8 KB. Cada uma tem um header de 96 bytes contendo informação de sistema como o ID da tabela a que a página pertence. O header também inclui apontadores para as páginas anteriores e seguintes que são usadas caso as páginas estejam ligadas numa lista.

- Organização de páginas de dados das tabelas: feita por 1 de 2 métodos:
  - Tabelas clustered são tabelas que têm índices clustered. As linhas de dados são armazenadas ordenadas com base na chave do índice clustered. O índice é implementado como uma estrutura de índice B-Tree baseada nos valores da chave de índice clustered. Os páginas em cada nível do índice, incluindo as páginas de dados no nível folha estão ligados numa lista duplamente ligadas mas a navegação entre níveis é feita através de valores chave.
  - Heaps são tabelas que não têm índices clustered. As linhas de dados não são armazenadas em nenhuma ordem particular e também não existe ordem de sequência das páginas de dados, nem estão ligadas entre si por meio de listas ligadas.

- Views indexadas têm a mesma estrutura de armazenamento que as tabelas clustered.

- O SQL suporta até 249 índices não clustered em cada tabela ou view indexada. Esses têm uma estrutura B-Tree semelhantes às dos índices clustered. A diferença é que não têm efeito algum na ordem das linhas de dados. Tabelas clustered e views indexadas coloca a ordem das linhas de dados baseado na chave do índice clustered. A colecção de páginas de dados para um heap não é afectado se os índices não clustered estão definidos para a tabela. As páginas de dados mantêm-se numa heap a não sei que um índice clustered esteja definido.

- As páginas que tenham dados do tipo text, ntext, e image são geridos como uma única unidade de cada tabela. Todos os dados são armazenados numa colecção de páginas.

- Toda a colecção de páginas para tabelas, índices e views indexadas são ancoradas por apontadores de páginas na tabela sysindexes. Casa tabela e view indexada tem uma colecção de páginas de dados mais uma colecção de páginas para implementar cada índice definido para a tabela ou view.

- Cada tabela, índice e view indexada tem uma linha na sysindexes unicamente identificada pela combinação da coluna do identificador do objecto (id) e a coluna do identificador do índice (indid). A alocação de páginas para as tabelas, índices, e views indexadas é gerida por um conjunto de páginas IAM. A coluna sysindexes.FirstIAM aponta para a primeira página IAM no conjunto que gere o espaço alocado à tabela, índice ou view indexada.

Estatísticas de distribuição

- Todos os índices tem estatísticas de distribuição que descrevem a selectividade e destruição dos valores chave no índice. Selectividade é uma propriedade que relaciona a quantas linhas estão tipicamente identificadas por um valor chave.

Uma chave única tem uma selectividade alta. Já uma chave encontrada em 1000 linhas tem uma selectividade má.

- Para ser útil para o optimizador de queries, as estatísticas de distribuição tem de ser manter actualizadas. Devem ser actualizadas cada vez que ocorram alterações significativas de chaves no índice. Podem ser actualizadas manualmente usando a operação `UPDATE STATISTICS`.

Estruturas Heap

- Heaps têm uma linha em sysindexes com indId a 0.
- A coluna sysindexes.FirstIAM aponta para a primeira página IAM no conjunto de páginas que gerem o espaço alocado à heap.
- As páginas de dados e as linhas dentro delas não estão em nenhuma ordem especifica nem ligadas entre si.
- Pesquisas em tabelas ou leituras em série podem ser feitas, pesquisando as páginas IAM.

Índices Clustered

- Índices clustered têm uma linha em sysindexes com indId a 1.
- As páginas estão ordenadas pelo valor da chave do índice clustered.
- Índices são organizados como B-Trees. Cada página num índice tem um header de página e linhas de índice.
- Cada índice contem um valor de uma chave e um apontador para uma página de nível inferior ou para uma linha de dados.
- Cada página num índice é um nó de um índice. O nó do topo é a raiz da B-Tree. A camada inferior são nós folha. As páginas em cada nível estão ligados entre si através de uma lista duplamente ligada.
- Para um índice clustered, sysindexes.root aponta para o topo do índice clustered. O SQL Server navega no índice para encontrar a linha correspondente de uma chave de índice clustered.

- Para encontrar um intervalo de chaves o SQL Server navega pelo índice para encontrar a chave de índice do intervalo e depois pesquisa pelas páginas de dados usando os apontadores.
- Para encontrar a primeira página num conjunto de páginas de dados, o SQL Server segue os apontadores mais à esquerda do nó raiz do índice.
- Estrutura de um índice clustered:

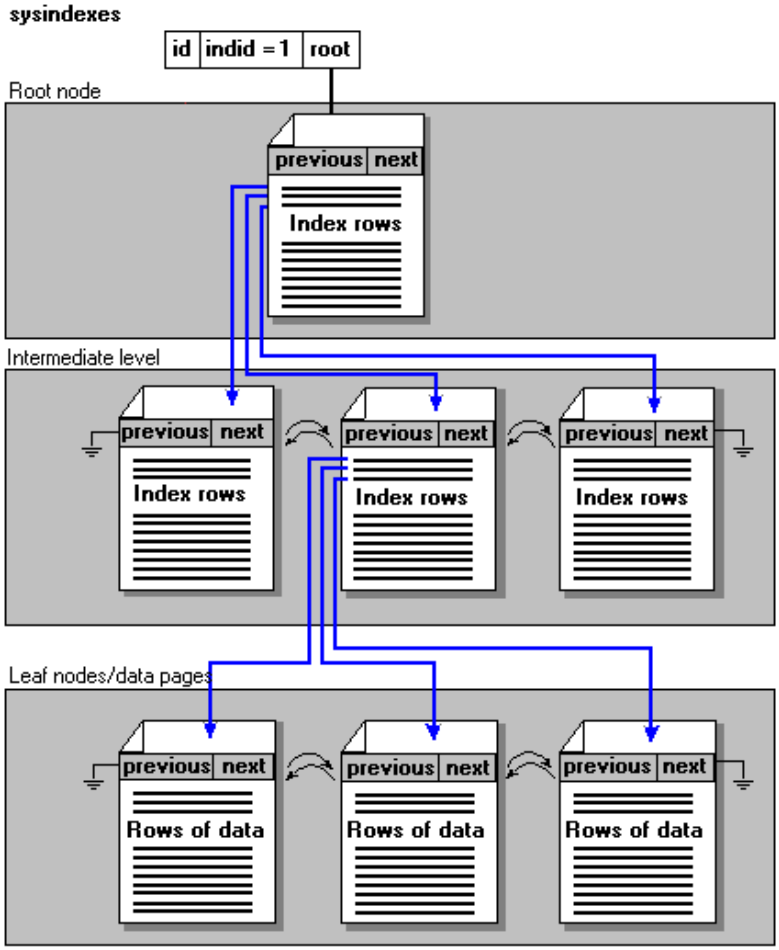

Imagem 3. Estrutura de um índice clustered

Índices Não Clustered

- Índices clustered têm a mesma estrutura B-Tree que os índices clustered, com duas grandes diferenças:
    - As linhas de dados não estão ordenadas nem armazenadas na ordem baseada nas suas chaves.

- A camada de folhas de um índice não clustered não consiste das páginas de dados. Em vez disso contêm linhas de índice. Cada linha de índice contem um valor de chave não clustered ou mais localizadores de linhas que apontam para as linhas de dados (ou linhas se o índice não for único) tendo o valor de chave.

- Podem ser definidos numa tabela com um índice clustered, heap ou uma view indexada. Os localizadores de linhas têm duas formas:
  - Se a tabela for uma heap, o localizador é um apontador para a linha. É construído pelo identificador de ficheiro, número de página e número da linha na página. O apontador é denominado por Row ID.
  - Caso não seja, ou o índice é numa view indexada, o localizador é o a chave do índice clustered para a linha. Se não for único, o SQL Server gera um valor interno para distinguir as chaves duplicadas. Este valor não é visível aos utilizadores.
- É importante manter as chaves dos índices clustered o mais pequenas possível, uma vez que os índices não clustered usam-nas como localizadores de linhas:

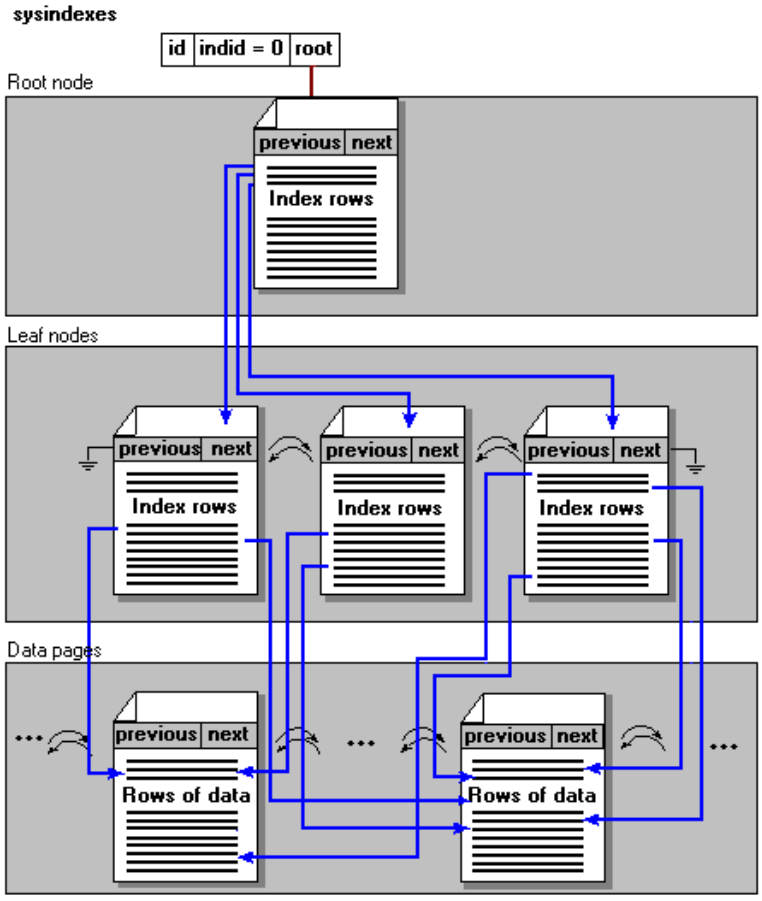

Imagem 4. Estrutura de um índice clustered

3. Indexação e Hashing

Embora não permita hashing na indexação, o SQL Server suporta vários tipos de índice, a saber:

Índices Clustered:
Armazenam e ordenam as linhas de dados da tabela ou vista na ordem da chave do índice clustered. É implementado numa estrutura B-Tree que suporta a obtenção rápida das linhas baseado nos valores da chave do índice clustered.

Índices Não Clustered:
Pode ser definido numa tabela ou view com um índice clustered ou numa heap. Cada linha de índice no índice não clustered contem o valor da chave e o localizador da linha. Este localizador aponta para a linha de dados do índice clustered ou heap que tem o valor da chave. As linha no índice estão armazenadas na ordem dos valores chave do incide, mas as linhas de dados não estão garantidamente em nenhuma ordem particular a não ser que exista um índice clustered na tabela.

Índices Único:
Garante que a chave de índice não tem valores duplicados e portanto todas a linhas da tabela / view são de algum modo únicas. Unicidade pode ser uma propriedade de índices clustered e não clustered.

Índices Columnstore:
Este tipo de índice agrupa e guarda os dados para cada coluna e depois junta todas as colunas para completar todo o índice. Para algumas queries, o processador de queries do SQL Server pode tirar partido do layout columnstore para melhorar os tempos de execução dos queries. É especialmente apropriado para conjuntos de data warehousing, permitindo melhor desempenho para queries comuns como filtragem, agregação, agrupamento e junção.

Índices com colunas incluídas:
É um índice não clustered que é estendido para incluir colunas que não sejam chaves.

Índices em colunas computadas:

É um índice numa coluna que é derivado do valor de um ou mais colunas ou certas entradas.

Índices filtrados:

Um índice não clustered optimizado, especialmente adequado para cobrir queries que escolhem de um subconjunto bem definido de dados. Usa um filtro para indexar uma porção das linhas na tabela. Um índice filtrado pode melhorar o desempenho de queries, reduzir custos de armazenamento e manutenção de índice, comparado com os índices de tabelas completas.

Índices espaciais:

Proporcionam a habilidade de efectuar algumas operações com mais eficiência em objectos espaciais na coluna dos tipos de dados geométricos. O índice espacial reduz o numero de objectos em que operações caras precisam de ser feitas.

Índices XML:

Representação fragmentada e persistente dos grandes objectos binários (BLOBs) de XML na coluna dos tipos de dados XML.

Índices Full-text:

Um tipo especial de índice funcional baseado em tokens. Este tipo de índice é construído e mantido pela Microsoft Full-Text Engine para o SQL Server. Proporciona um suporte eficiente para pesquisas sofisticadas de palavras em dados de caracteres em string.

4. Processamento e optimização de perguntas

Linguagem Intermédia:

As expressões SQL são compiladas e são gerados "stubs" em MSIL (Microsoft Intermediate Language). Através desses "stubs" são passadas as instruções e respectivos parâmetros para CLR (Common Language Runtime) e é assim gerado o código a ser executado pela máquina.

Select no MS SQL Server:

É igual ao do Oracle SQL. Podemos fazer Select * from table ou escolher apenas determinadas colunas, tal como no Oracle SQL.

Joins Disponibilizados pelo MS SQL Server:

1. JOIN ou INNER JOIN
2. OUTER JOIN
    2.1 LEFT OUTER JOIN ou LEFT JOIN
    2.2 RIGHT OUTER JOIN ou RIGHT JOIN
    2.3 FULL OUTER JOIN ou FULL JOIN
3. NATURAL JOIN
4. CROSS JOIN
5. SELF JOIN

Todos os joins acima enumerados têm um funcionamento igual aos do Oracle SQL com o mesmo nome. O Self Join não existe no Oracle SQL, e é o join de uma tabela com ela própria, seja inner ou outer join.

Implementação das Operações Join:

Nested Loop Join:

Por cada linha da outer table todas as linhas da inner table são "matched", por assim dizer, uma a uma com essa linha da outer table, gerando assim o resultado do join entre as duas tabelas.

Variantes do Nested Loop Join:

Naive: Percorre todas as linhas das tabelas.

Index: Se for possível utilizar o index, é executado um index nested loop join que se serve do index para não ter de percorrer toda a tabela caso não seja necessário.
Temporary Index: É criado um index temporário como parte da query e destruído após a execução da query. O restante processo é igual ao do Index Nested Loop Join.

Merge Join:

Requer que ambas as tabelas estejam ordenadas pelo atributo para o qual é feito o join.
Ao contrário do Nested Loop Join não tem de comparar com todas as linhas visto que as linhas estão ordenadas e é fácil perceber quando já não vai ter mais "matches".
Por essa razão é muito mais eficiente que o Nested Loop Join para junção de tabelas com muitas linhas.

Hash Join:

É utilizado para joins de tabelas grandes que não têm indexes associados que possam ser utilizados, ou seja, também não estão ordenadas.
Tem duas phases o algoritmo de hash join:

Build phase:

É gerado um hash para cada key (a utilizada no join) da tabela mais pequena, depois disso são colocados numa hash table na memória.

Probe phase:

À semelhança da Build Phase, é gerado um hash para cada key mas da tabela maior desta vez, e depois é feito o match entre ambas as tabelas de hash geradas, e assim se consegue a junção das tabelas.

Variantes do Hash Join:

In-Memory Hash Join: Caso exista memória para guardar a tabela de hash.
Grace Hash Join: Caso não exista memória suficiente para guardar a tabela de hash, a parte excedente desta irá para a tempdb. Menos eficiente que o anterior.
Recursive Hash-join: Caso a tabela seja tão grande que o query optimizer tenha de fazer vários níveis de merge-joins.

Como se escolhe qual o algoritmo a utilizar:

Exemplo:

```
SELECT poh.PurchaseOrderID, poh.OrderDate, pod.ProductID, pod.DueDate,
poh.VendorID
FROM Purchasing.PurchaseOrderHeader AS poh
INNER MERGE JOIN Purchasing.PurchaseOrderDetail AS pod
    ON poh.PurchaseOrderID = pod.PurchaseOrderID;
```

Como demonstrado acima, basta meter antes da palavra join o algoritmo que desejamos utilizar. Isto é valido tanto para Hash, Merge ou Nested Loop Join.

Query Optimizer:

Após escrevermos a query e executarmos, ela passa pelo parser algebrizer (passagem para álgebra relacional) e de seguida actua o Query Optimizer que vá calcular e escolher o plano de execução mais eficiente para a nossa Query. Depois disso passa então à fase de execução do plano.

Materialização:

É possível criar views materializadas mas no MS SQL Server são conhecidas como "Indexed Views". Como o nome indica são views com index. Para criar uma Indexed View existem dois passos a fazer, criar a view e adicionar o index.

Exemplo:

```
CREATE VIEW ContractJobs WITH SCHEMABINDING AS
SELECT J.JobId, J.ContractNumber, SJ.JobName, SJ.StandardPrice,
C.ContractValue FROM dbo.Jobs J INNER JOIN dbo.StandardJobs SJ
ON J.StandardJobId = SJ.StandardJobId
INNER JOIN dbo.Contracts C ON J.ContractNumber = C.ContractNumber
WHERE C.RenewalDate IS NOT NULL
```

E depois da vista já estar criada, adicionar o index clustered:

```
CREATE UNIQUE CLUSTERED INDEX IX_ContractJobs_JobId ON
ContractJobs
(
JobId
)
```

Em seguida podem ser adicionados mais índices não clustered à view:

```
CREATE INDEX IX_ContractJobs_ContractNumber ON
ContractJobs
(
ContractNumber
)
```

Pipelining: É suportado.

Consulta do Plano de Execução:
Graficamente: RightClick na janela da query e clicar “Display Estimated Execution Plan” e o mesmo para “Include Actual Execution Plan”.
Texto/tabela: Set Statistics Profile On, e também, Set SHOWPLAN_TEXT ON. Alternativamente pode-se utilizar SET SHOWPLAN_ALL ON cujo o resultado é igual ao showplan_text mas é “formatado” de forma diferente.
XML: Set SHOWPLAN_XML On.

Reutilização de Planos:
No MS SQL Server ao executarmos uma query o optimizer calcula o custo dos planos de execução possíveis e escolhe o melhor. Depois disso esse plano é guardado no Plan Cache para que caso se volte a executar essa query não volte a ser calculado qual o melhor plano, sendo assim mais eficiente e menos dispendioso. Isto é bastante vantajoso na utilização de SPs (Stored Procedures), por exemplo.

Os planos podem depois ser recompilados (muito dispendioso, não é recomendado excepto para casos especiais) ou mesmo apagados da Plan Cache caso esta precise de espaço ou se a idade do plano chegar a zero.

5. Gestão de transacções e controlo de concorrência

Iniciar uma transacção: Begin Transaction

Rollback de uma transacção: Rollback

Commit de uma transacção: Commit

No MS SQL Server é necessário iniciar a transacção ao contrário do Oracle.

Nested Transactions:
São permitidas e utilizadas por exemplo se tivermos uma chamada a uma SP dentro de uma Transaction. Caso a Transaction que chama a SP faça commit a transaction da SP também faz commit, e vice-versa para Rollback.

Transacções de Longa Duração:
O suporte para transacções deste tipo no MS SQL Server divide-se em três partes:

Logs: As transacções são anotadas nos logs.

DBCC Opentrans: Mostra os detalhes da Transaction que está a correr há mais tempo.

Sys.dm_tran_database_transactions: Todos os detalhes sobre as transactions executadas ou a decorrer. Por exemplo, begin_time, transaction_state.

Isolamento:
O nível de isolamento pode ser escolhido através da chamada:

```
SET Transaction Isolation Level {nível de isolamento desejado}
```

O nível default de uma transaction é Read Commited.

Níveis de Isolamento:

Existem 5 níveis de isolamento no MS SQL Server (ordenados do menor isolamento ao mais rígido):

Read Uncommited:

Nível de isolamento mais baixo. Como o nome indica lê dados que ainda não foram commited. Semelhante ao comando Select com a opção `NOLOCK`. Igual ao read uncommited utilizado no Oracle.

Read Commited:

Ao contrário do nível anterior, não são possíveis "Dirty Reads", ou seja, só são lidos dados que já foram commited. Também presente no Oracle.

Repeatable Read:

A diferença para o Read Commited está em que se a transacção ler dados, esses dados não podem ser modificados por outras transacções até a transacção acabar, daí o nome Repeatable Read, se forem lidos em dois momentos por esta transacção os mesmo campos e se esta transacção não os tiver modificado, o resultado tem de ser igual. Mais uma vez, este nível é igual ao do Oracle SQL que tem o mesmo nome.

Snapshot:

Os dados são lidos da última versão transaccionalmente consistente da DB. Isto significa que nenhum dado modificado por uma transacção concorrente será lido pela transacção actual. Mais rígido ainda que o Repeatable Read no que toca à leitura de dados. Não existe no Oracle SQL.

Serializable:

As instruções não podem ler dados que foram modificados, e que ainda não foram confirmados por outras transacções.

Nenhuma outra transacção pode modificar dados lidos pela transacção actual até que a transacção actual seja concluída.

Outras transacções não podem inserir linhas novas com valores chave que estejam no intervalo de chaves lido por qualquer instrução da transacção actual até que esta seja

concluída. É o nível mais rígido de Isolamento. O funcionamento é igual no Oracle SQL.

DeadLocks:

Tal como no Oracle, é detectado um deadlock quando 2 ou mais transacções concorrentes precisam de lock sobre os mesmos dados ou recursos. O Database Engine ao detectar isto, escolhe uma das transacções (a menos dispendiosa) e termina-a, fazendo-a devolver uma mensagem de erro, ou seja, termina sem sucesso.
O processo de detecção de Deadlocks é feito pelo lock monitor thread do Database Engine, que faz uma pesquisa de 5 em 5 segundos por todas as transacções a correr e caso detecte um deadlock. Caso os deadlocks sejam muito frequentes a pesquisa pode ser efectuada até de 100 em 100 milissegundos para que os deadlocks sejam identificados e tratados o mais rapidamente possível.

Deadlock_Priority:

Pode ser definida na transacção como LOW, NORMAL ou HIGH. Alternativamente pode ser definida com valores inteiros de -10 a 10.
Ao ser detectado um deadlock, irá ser vista a deadlock_priority de cada transacção em deadlock e será terminada a com menos prioridade.
Comando utilizado para definir a deadlock priority:

```
SET DEADLOCK_PRIORITY HIGH;
```

Granularidade:

É possível utilizar multigranularidade de forma a bloquear apenas parte de recursos.
Por exemplo, podemos apenas fazer lock sobre uma linha de uma tabela e não sobre toda a tabela, de forma que as restantes linhas fiquem disponíveis para transacções concorrentes. Por outro lado podemos até bloquear a DB inteira, depende do objectivo da transacção.

O “downside” de bloquear recursos de menor granulidade como linhas, por exemplo, é que obriga o sistema a manter mais locks e se forem muitos pode acontecer uma sobrecarga. Em suma, deve ser usado com moderação e bem pensado, e se assim for, é muito vantajoso mesmo.

Exemplo:

```
BEGIN TRANSACTION
Update students WITH (TabLOCK)
Set name = “zé”
GO
```

Aqui bloqueamos toda a tabela porque todos os students vão ser editados.

```
UPDATE
Students WITH (ROWLOCK)
SET majority= false
WHERE Age<18
```

Neste exemplo bloqueamos apenas as linhas que vão ser editadas.

Consistência:

Para verificar a consistência da DB podemos utilizar o comando DBCC CHECKDB, que irá verificar o espaço alocado (DBCC CHECKALLOC), a integridade de todas as páginas e tabelas (DBCC CHECKTable), e ainda irá validar o conteúdo das vistas.
A operação DBCC CheckDB irá correr sobre um snapshot da DB de forma a não bloquear a DB e assim as transacções ditas normais poderem continuar a ser executadas.

Recuperação de Dados ou Restore:

O MS SQL Server permite-nos fazer recuperação de dados com três métodos diferentes:

Simple Recovery Mode: Restore a partir do último backup. Ignora o transactions log.
Full Recovery Mode: Este modo é utilizado para recuperar tudo até ao ponto de falha. Muito mais pesado que o Simple Recovery Mode porque este vai ler o transactions log de forma a recuperar tudo até à falha que ocorreu.
Bulk-Logged Recovery Mode: Versão larga escala do Full Recovery Mode. Só deve ser usado mesmo para recuperações de larga escala.

Ao utilizarmos o comando,

```
Restore Log database_name
From path
WITH Recovery|NoRecovery|Standby
```

A última opção Recovery|NoRecovery|Standby, permite-nos escolher o estado em que a DB fica após essa operação. Ou seja, se escolhermos NoRecovery o sistema assume que a DB ainda não está totalmente recuperada e esta não se encontra disponível para ligações. Com a opção Recovery a DB está pronta a ser utilizada, é a opção mais comum. Com a opção Standby são permitidas ligações à DB mas os transactions logs futuros só serão aplicados à DB se não existirem ligações à DB. O menos utilizado dos 3.

6. Suporte para bases de dados distribuídas

O *Microsoft SQL Server* tem sistemas para suportar bases de dados distribuídas de forma a garantir a escalabilidade, disponibilidade e segurança dos sistemas que implementa. Na prática, as principais razões que podem levar a fazer distribuição e replicação de dados são: fazer a distribuição de pedidos por vários servidores, garantir a capacidade de poder trabalhar sem estar sempre ligado a uma rede e também garantir a redundância de dados para garantir recuperação de falhas a qualquer instante.

Tipos de Replicação

Nos cenários de replicação são necessários vários componentes:

- *Publishers,* entidade que inicia a replicação e que por sua vez oferece os dados a replicar. Os dados replicados são as *Publications* que não nada mais nada menos que um conjunto de *Articles* (ex: tabelas, views, índices, etc) que podem ser filtrados quando são enviados.
- *Distributor*, entidade intermédia entre o início e o fim de uma replicação. Este elemento recebe as *publications* e reencaminha-as para a entidade final, os *Subscriber*.
- *Subscribers* entidade final da replicação que recebe as publicações e faz a operação *subscription*. As subscrições podem ser do tipo *push*, onde todas

as actualizações são responsabilidade do *publisher*, e do tipo *pull* onde é o subscriber a fazer o pedido de replicação.

O *SQL Server* permite fazer três tipos de replicação de dados: *Snapshot Replication*, *Transational Replication* e *Merge Replication*.

Snapshot Replication:

Este tipo de replicação permite transferir automaticamente cópias exactas entre bases dados do tipo *SQL Server*, garantindo desta forma uma maior performance dos sistemas. Neste tipo de operação, é feita uma cópia integral de uma base de dados do *Publisher* para o *Subscriber* com o custo de uma única operação. O tempo desta operação depende do tamanho da base de dados em questão. Idealmente esta operação deve ser feita em bases de dados em que os dados não são frequentemente alterados, em não exista problema em haver versões muito desactualizadas com as do *Publisher*, sejam replicadas pequenas quantidades de dados ou grandes quantidade de dados que ocorram num curto período de tempo. Por defeito os outros tipos de replicação usam o *snapshot* para iniciar os *Subscribers.*

Para fazer esta operação é necessário: criar o distribuidor, a publicação e subscrever a publicação. O *SQL Server Snapshot Agent* gera o *snapshot* mas a distribuição para neste tipo de replicação é feita pelo *Distribuition Agent*. Numa *push subscripiton* o *Distribuition Agent* actua no *Distributor* mas numa *pull subscription* este *Agent* actua de *Subscriber*, em qualquer dos casos o Snapshot Agent actua sempre no *Distributor*.

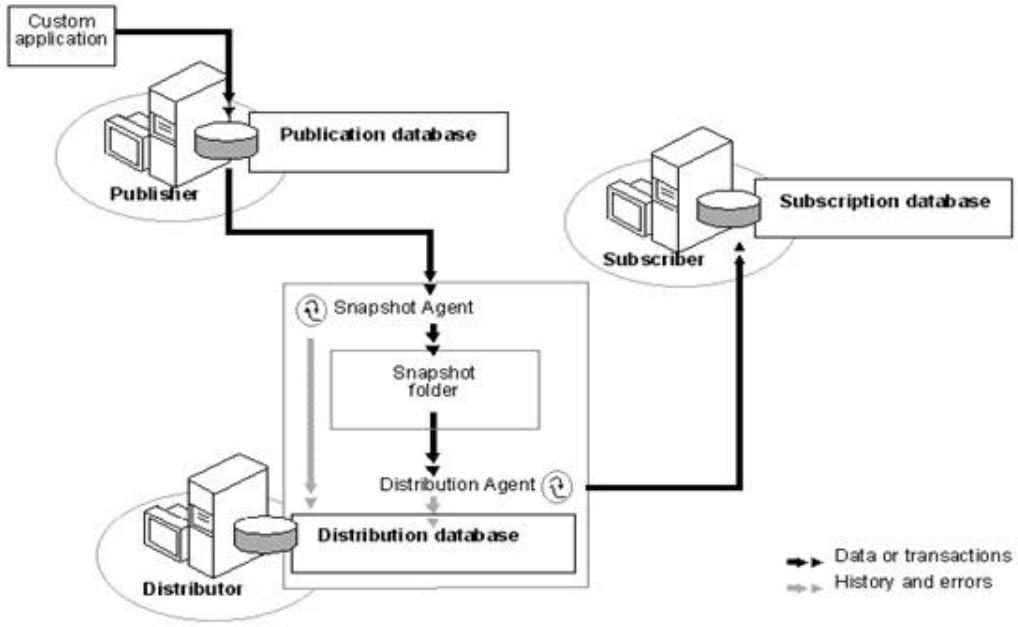

Imagem 5. Esquema de snapshot replication

Esta replicação funciona da seguinte forma:

- O *Snapshot Agent* faz a conexão do *Distributor* para o *Publisher* e gera um novo *snapshot* através da criação de *locks*.
- O *Snapshot Agent* faz uma cópia do esquema de tabelas de cada *Article* para um ficheiro *.sch*.
- Copia os dados do *Publisher* e escreve-os num ficheiro *.bcp* na pasta de *Snapshtot*.
- Acrescenta cada linha às tabelas *Msrepl_commands* e *Msrepl_transactions*.
- Liberta os *locks* feitos anteriormente.

Transational Replication:

Este tipo de replicação é usado tipicamente em situações *server-to-server*. Neste tipo de operação os dados são passados do *Publisher* para o *Subscriber* à medida que ocorrem. Os dados que chegam ao *Subscriber* são passados pela mesma ordem e nas mesmas condições que acontecem no *Publisher*, para desta forma ser garantida a consistência de dados.

Esta replicação é feita quando é necessário propagar alterações incrementais do *publisher* para o *subscriber*, quando uma aplicação necessita de ter acesso a todas as alterações de estado dos dados em tempo real, quando o *publisher* tem um elevado numero de operações *CRUD* ou então se tanto o *Subscriber* ou o *Publisher* forem *non-SQL Servers*.

Neste tipo de replicação os *subscribers* são tratados como *read-only* visto que, nestes casos, as alterações não se propagam para os *publishers*.

Os *Agents* que implementa esta replicação são o *SQL Server Snapshot*, o *Log Reader* e o *Distribution*.

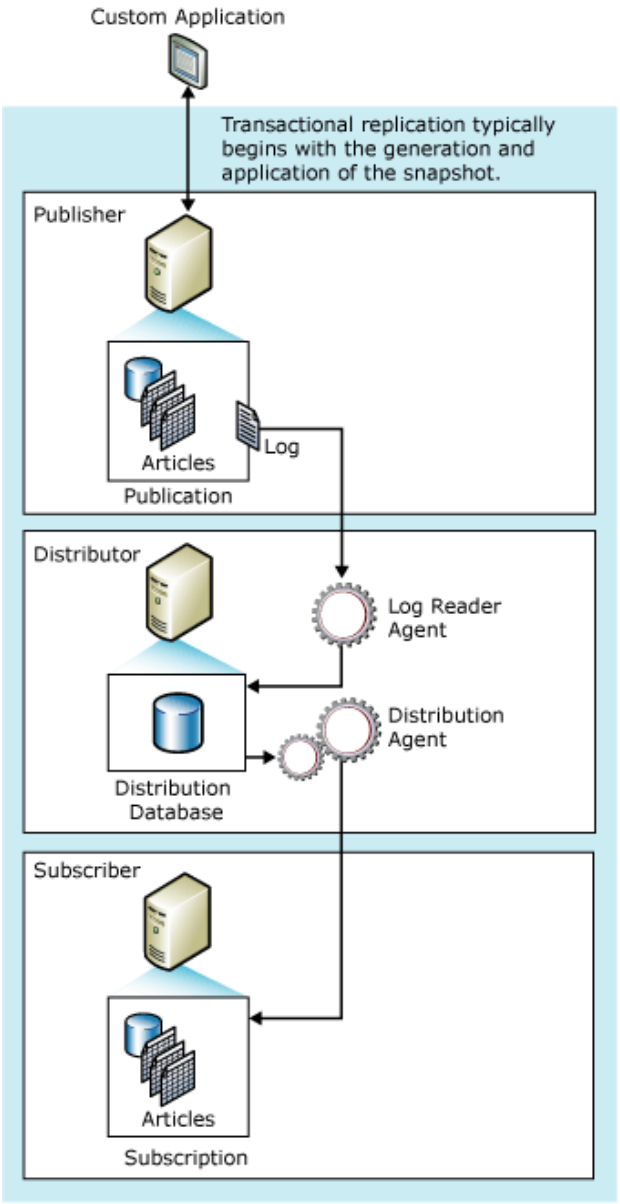

Imagem 6. Esquema de transactional replication

Esta replicação funciona da seguinte forma:

- O *Log Reader Agent* faz a monitorização dos *logs* de transacção das bases de dados configuradas nesta operação e copia as transacções marcadas para replicação do log para a tabela de distribuição.
- O *Distribution Agent* copia os *snapshots* e as transacções gravadas para os *Subscribers.*
- As alterações incrementais são geridas pelo *Distribution Agent* que está sempre a correr com uma baixa latência ou então em intervalos programados;

Merge Replication

Este tipo de replicação é normalmente usado em situações *server-to-client*. Nesta situação o *subscriber* liga e sincroniza-se com o *publisher* e recebe todas as alterações que foram feitas desde da última sincronização.

Esta replicação é feita nos casos em que vários *subscribers* precisem dos mesmos dados e também de propagar essas alterações para o *publisher* e/ou outros *subscribers*, quando um subscriber faz alterações *offline* e depois precisa de sincronizar esses dados com o *publisher*, quando vários subscribers precisem de diferentes partes de dados ou então quando existem alterações de dados e apenas é necessário propagar a última alteração feita.

O objectivo deste tipo de replicação é garantir a uniformidade de dados entre vários sistemas mas, podem acontecer conflitos de versões que podem ser tratados pelo *SQL Server*.

Esta replicação é implementada pelo *Snapshot Agent* e pelo *Merge Agent*.

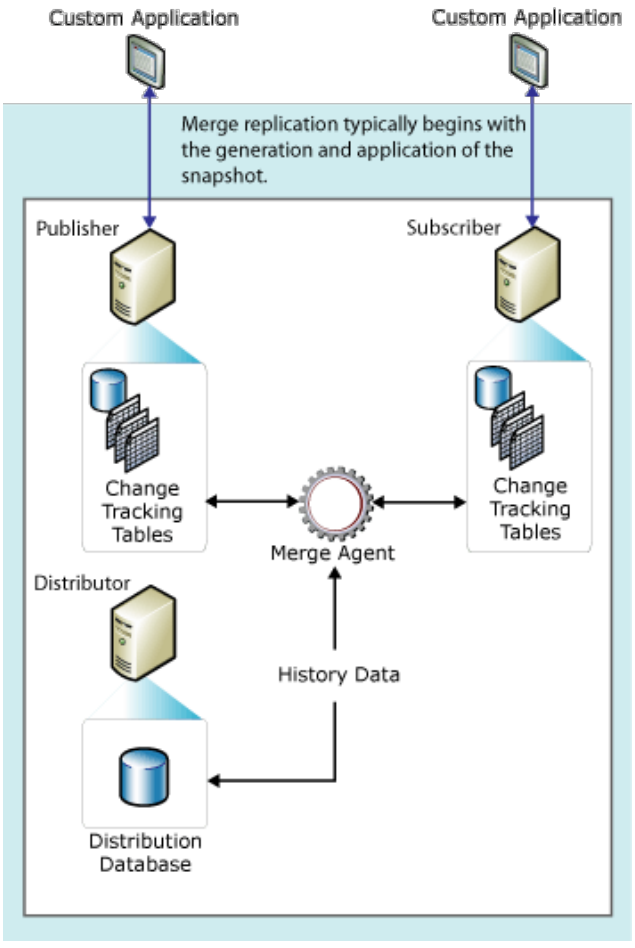

Imagem 7. Esquema de merge replication

O funcionamento desta replicação é o seguinte:

- Se a *publication* não for filtrada ou usar filtros estáticos, o *snapshot* agente cria um único *snapshot*, caso contrário cria um para cada partição de dados.
- O *Merge Agent* tem a responsabilidade de aplicar os *snapshots* aos *subscribers* assim como de fazer as alterações incrementais e resolver qualquer conflito de acordo com regras que sejam definidas por quem pede este tipo de replicação.

Queries Distribuídas:

O *SQL Server* suporta dois métodos para fazer a referenciação de bases de dados externas:

- ***Linked Server Names***, usam as SP's de sistema ***sp_addlinkedserver*** e ***sp_addlinkedsrvlogin*** para associar o nome de um servidor a uma *data source*. Os objectos referenciados nestes *linked servers* podem ser referenciados nas próprias queries. Por exemplo, utilizado o servidor srv1 numa query:

```
Select * from srv1.database.table.id
```

- ***Ad hoc conector names***, através das instruções *OPENROWSET* e *OPENDATASOURCE* é possível referenciar *data sources* externos.

O *SQL Server* também permite a criação de sinónimos para referenciar objectos noutros servidores através do seguinte comando:

```
create synonym externaltable
for srv1.database.table.id
```

Transacções Distribuídas:

No *SQL Server*, as transacções distribuídas são abrangidas por dois ou mais servidores que têm a função de *resource managers* e são coordenadas através do componente *transaction manager*.

Numa aplicação, uma transacção distribuída e tratada da mesma maneira que uma transacção local e, no final da mesma, esta pode ser ou *commited* ou *rolled back*. Um *commit* distribuído tem de ser tratado de forma diferente de um *commit* local pelo *transaction manager*, para não se correm riscos de falhas de rede. Para este caso o *SQL Server* utiliza o protocolo *Two Phase Commit a.k.a 2PC*.

As fases são as seguintes:

- Fase de preparação, nesta fase o *transaction manager* recebe o pedido para *commit* e avisa todos os envolvidos através de um comando *prepare*. Cada *resource manager* tem então de fazer o necessário para proceder à transacção e, no final, dar a indicação do resultado da mesma.
- Fase de commit, depois de receber resultados positivos do comando *prepare*, o *transaction manager* envia então o comando *commit* para todos os *resource managers*. Assim os envolventes podem então executar o *commit* e informar o *transaction manager* do resultado, caso seja enviado algum resultado falhado é enviada a mensagem e *rollback* e caso contrário a transacção é efectuada com sucesso.

7. Outras características do sistema estudado

Suporte de XML:

Todos os componentes do SQL Server têm suporte a XML, nomeadamente:

- Importação e exportação de ficheiros XML;
- Suporte pata o tipo de dados XML;
- Especificar queries XQuery nos dados XML armazenados em colunas e variáveis do tipo xml;
- Importação de dados XML em massa para uma coluna do tipo XMLData através da operação OPENROWSET;

Suporte de Semantic Web:

Contrariamente ao Oracle, que suporta standards como RDF/Schema ou SPARQL, o Microsoft SQL Server não tem nenhum suporte nativo a Semantic Web

Suporte a formas procedimentais:

Stored procedures: São procedimentos de código que foram previamente escritos e armazenados, sendo que podem ser reutilizados.

Aceitam argumentos de entrada e podem retornar valores ao programa que os chamam. Um stored procedure é um grupo de uma ou mais instruções de Transact-SQL ou referencia a um método de uma CLR (common runtime language) da Microsoft .NET Framework.

Suporte a Triggers:

Triggers são instruções que validam dados e asseguram a integridade da base de dados. O SQL server suporta vários tipos de triggers como triggers DML do tipo AFTER e INSTEAD OF e triggers DDL.

Suporte a Serviços ODBC e JDBC:

O SQL Server suporta ODBC por via de um driver, para Windows ou para Linux. Estes drivers proporcionam uma API que implementa os interfaces ODBC standard no Microsoft SQL Server.

Tal como ODBC, o SQL Server suporta também JDBC por via de um driver.

Segurança:

Dispõe de uma arquitectura de segurança desenhada para os administradores e programadores de bases de dados criarem aplicações seguras e combater ameaças.

O administrador tem ao ser dispor os seguintes tipos de segurança:

- **Autenticação** de utilizadores a recursos adicionais e reservados do SQL Server;
- **Roles** (papéis) no servidor e na base de dados que dão acesso a determinadas acções ao utilizador;
- **Ownership** (posse) de objectos da base de dados e quem os pode administrar.
- **Autorização** e permissão de acesso a recursos.
- **Encriptação de dados** protegendo-os de acessos não autorizados com chaves de decifragem ou certificados.
- **Integração da segurança da CLR**

8. Comparação com o Oracle 11g

O Oracle 11g e o MS SQL Server são bastante semelhantes mas têm alguns factores diferenciadores, uns mais relevantes que outros.

O MS SQL Server é mais “leve” que o Oracle, além de ter os requisitos mínimos de hardware mais baixos.

As linguagens apesar de serem diferentes e de proprietários diferentes, têm uma sintaxe não muito diferente e mesmo a nível de funcionalidades a diferenças não são transcendentes.

No que diz respeito ao sistema de ficheiros, são parecidos sendo que o Oracle tem mais tipos de ficheiros que o SQL Server visto que este só tem 3.

Em relação a indexação, o Oracle é mais completo que o SQL Server permitindo Hashing enquanto que o MS SQL Server é focado em B-Trees. No entanto o MS SQL Server tem mais opções de escolha do tipo de Index a utilizar mesmo sendo todos sobre B-Trees.

Analisando o controlo de transacções, encontra-se algumas diferenças. No MS SQL Server temos de iniciar a transacção enquanto no Oracle tudo é uma transacção e até ao commit nada é definitivo. Por omissão, ambos utilizam o mesmo modo de isolamento, Read Commited mas o MS SQL Server tem mais um modo de isolamento (Snapshot).

Para replicação de dados, o MS SQL Server oferece mais opções de escolha do modo de replicação mas o Oracle permite utilizar protocolos optimistas e pessimistas de forma a controlar melhor a concorrência.

Resumindo, o MS SQL Server não suporta tantas estruturas mas aprofunda mais as que suporta com mais métodos para escolher dentro dessa estrutura, por exemplo na indexação, não permite hashing mas tem mais tipos de indexação sobre B-Trees. Tanto no mundo académico como no mundo profissional, ambos cumprem bastante bem a sua função, tendo um vantagem nuns campo e o outro em outros campos dentro desses mundos. No nosso ponto de vista, o MS SQL Server é ligeiramente mais fácil de utilizar e a interface é mais apelativa mas nada de realmente importante.

Referências:

Armazenamento e file structure

- Buffer Management:
  http://technet.microsoft.com/en-us/library/aa337525(v=sql.105).aspx

- File System:
  http://technet.microsoft.com/en-us/library/aa214422(v=sql.80).aspx
  http://technet.microsoft.com/en-us/library/ms179316(v=sql.105).aspx
  http://www.practicalsqldba.com/2013/09/sql-server-data-file-structure.html

- Partições:
  http://msdn.microsoft.com/en-us/library/ms190787(v=sql.110).aspx
  http://msdn.microsoft.com/en-us/library/ms188730(v=sql.110).aspx

- Organização:
  http://technet.microsoft.com/en-us/library/aa174541(v=sql.80).aspx

Indices e hashing

http://msdn.microsoft.com/en-us/library/ms175049%28v=sql.110%29.aspx

Processamento e optimização de perguntas

http://technet.microsoft.com/en-us/library/ms180765(v=sql.105).aspx
http://msdn.microsoft.com/en-us/library/ms187757.aspx
http://msdn.microsoft.com/en-us/library/ms190646.aspx
http://msdn.microsoft.com/en-us/library/ff650689.aspx
http://technet.microsoft.com/en-us/library/aa964133(v=sql.90).aspx

Gestão de transacções e controlo de concorrência

http://msdn.microsoft.com/pt-BR/library/ms173763.aspx
http://technet.microsoft.com/en-us/library/ms378149(v=sql.110).aspx
http://technet.microsoft.com/en-us/library/aa213026(v=sql.80).aspx
http://www.techrepublic.com/article/control-sql-server-locking-with-hints/
http://msdn.microsoft.com/pt-BR/library/ms186736.aspx

Suporte para bases de dados distribuídas

- Replicação

  http://www.codeproject.com/Articles/715550/SQL-Server-Replication-Step-by-Step

  http://databases.about.com/cs/sqlserver/a/aa041303a.htm

  http://databases.about.com/od/sqlserver/a/snapshot_replication.htm

  http://technet.microsoft.com/en-us/library/ms152501(v=sql.105).aspx

  http://msdn.microsoft.com/en-us/library/ms151176.aspx

  http://msdn.microsoft.com/en-us/library/ms152746.aspx

- Queries distribuídas

  http://technet.microsoft.com/en-us/library/ms191277(v=sql.105).aspx

- Transacções Distribuídas

  http://technet.microsoft.com/en-us/library/ms191440(v=SQL.105).aspx

Outras características do sistema estudado

- XML: http://msdn.microsoft.com/en-us/library/bb522446%28v=sql.110%29.aspx

- Stored procedures: http://msdn.microsoft.com/en-us/library/ms190782.aspx

- ODBC e JDBC:

  http://technet.microsoft.com/en-us/library/jj730308%28v=sql.110%29.aspx

- Segurança: http://msdn.microsoft.com/en-us/library/bb669078.aspx