

República Federativa do Brasil
Ministério do Desenvolvimento, Indústria
e do Comércio Exterior
Instituto Nacional da Propriedade Industrial

(21) **PI0708827-2 A2**

(22) Data de Depósito: 16/01/2007
(43) Data da Publicação: 14/06/2011
(RPI 2110)



(51) *Int.Cl.:*
G06F 17/30 2006.01

(54) Título: **LINGUAGEM DE CONSULTA EXTENSÍVEL COM SUPORTE PARA TIPOS DE DADOS RICOS**

(30) Prioridade Unionista: 21/08/2006 US 11/507.864, 20/03/2006 US 60/784.510, 20/03/2006 US 60/784.510, 21/08/2006 US 11/507.864

(73) Titular(es): Microsoft Corporation

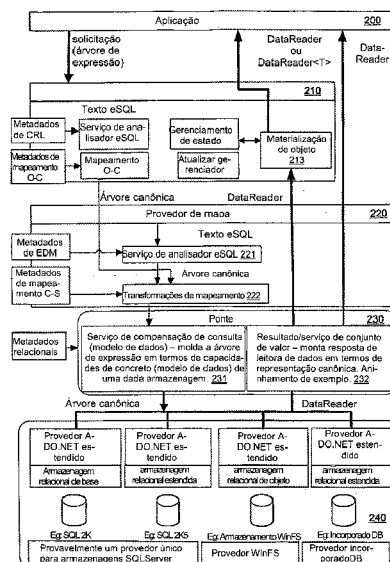
(72) Inventor(es): Fabio Mereles F. Valbuena, Ramesh Nagarajan, Sean B. House, Simon Cavanaugh, Stephen Frederick Starck, Subramanian Muralidhar

(74) Procurador(es): Nellie Anne Daniel Shores

(86) Pedido Internacional: PCT US2007001008 de 16/01/2007

(87) Publicação Internacional: WO 2007/108858 de 27/09/2007

(57) Resumo: LINGUAGEM DE CONSULTA EXTENSÍVEL COM SUPORTE PARA TIPOS DE DADOS RICOS. É fornecida uma linguagem de consulta com suporte para tipos de dados ricos. Um sistema pode compreender uma aplicação configurada para gerar consultas de acordo com a linguagem de consulta. Também é fornecida uma plataforma para tratar de consultas que são geradas de acordo com a linguagem de consulta fornecida. A plataforma compreende genericamente uma camada de serviços de objeto, uma camada de provedor de mapa, e uma camada de ponte que pode realizar interface finalmente com uma variedade de bancos de dados comercialmente disponíveis.





PI0708827-2

0200 801 228 20
22: 22: 00: 2006

"LINGUAGEM DE CONSULTA EXTENSÍVEL COM SUPORTE PARA TIPOS DE DADOS RICOS"

Referência remissiva a pedidos relacionados

Esse pedido reivindica prioridade para o pedido provisional US número 60/784.510,
5 depositado em 20 de março de 2006.

Antecedentes

Dados se tornaram um bem importante em quase toda aplicação, quer seja por exemplo, uma aplicação de Linha de Negócio (LOB) que navega em produtos e gera pedidos ou uma aplicação de Gerenciamento de Informações pessoais (PIM) que programa uma
10 reunião entre pessoas. Aplicações estão se tornando cada vez mais cêntricas em dados – planejam uma porção significativa de sua experiência em run-time e desenho sobre consultar, manipular e apresentar dados. Muitas dessas aplicações lidam com dados que são ricos em semântica, como integridade estrutural, limitações de dados, relações entre dados, e assim por diante. As aplicações hoje em dia gastam esforço significativo em código de pro-
15 cedimento para conservar a semântica de dados.

Considere, por exemplo, uma aplicação LOB. Tipicamente, uma tal aplicação lida com Clientes, Pedidos, Linhas de Pedidos, Fornecedores, Produtos, Funcionários, Transportadores, Faturas, e assim por diante. Cada uma dessas noções representa um tipo de dados ricos separado com uma estrutura específica. Por exemplo, o tipo Cliente tem coisas como
20 ID de Cliente, Nome da companhia, Nome do contato e Endereço; o tipo Pedido tem coisas como ID de Pedido, ID de Cliente, Data do pedido, Linhas de pedido, Data de vencimento, etc. Qualquer um dos acima pode ter exigências adicionais, por exemplo, Endereço pode exigir um Código Postal que, quando nos EUA, deve ser um código postal que tenha cinco caracteres de comprimento, e cada caractere é um dígito entre zero e nove. No Canadá, o
25 Código Postal deve ser da forma "ANA NAN" onde A é uma letra e N é um número. Ao modelar códigos postais, desse modo não é suficiente simplesmente especificar que é uma série; limitações adicionais devem ser colocadas nessa série para limitar a gama de valores possíveis que pode assumir. Além disso, há normalmente relações entre os dados. Por exemplo, um Pedido pode ter sempre um Cliente associado ao mesmo; essa é uma relação
30 de muitos (Pedido)-para-Um (Cliente). Os Produtos e Fornecedores contêm uma relação de muitos-para-muitos porque múltiplos produtos podem ser fornecidos por um único fornecedor, e múltiplos fornecedores podem abranger o mesmo produto.

Um modelo de dados descreve a estrutura e semântica de, e relações entre, os vários pedaços de dados nos quais uma aplicação está interessada. Embora modelos e sistemas de relação tenham sido muito bem sucedidos em gerenciamento de dados, eles falha-
35 ram em capturar os modelos de dados de aplicação. As aplicações cliente-servidor tradicionais relegam operações de persistência e consulta em seus dados para sistemas de banco

de dados. O sistema de banco de dados opera em dados na forma de linhas e tabelas, enquanto a aplicação opera em dados em termos de construções de linguagem de programação de nível mais elevado como classes e tipos de dados ricos. O descasamento de impedância nos serviços de manipulação de dados entre a aplicação e nível de banco de dados era tolerável em sistemas tradicionais. Com o advento de arquiteturas orientadas para serviço (SOA), servidores de aplicação e aplicações de multi-níveis, a necessidade de serviços de manipulação e acesso de dados ricos que sejam bem integrados com ambientes de programação e possam operar em qualquer nível aumentou tremendamente.

A maioria das aplicações e estruturas de aplicação rola seu próprio modelo de dados no topo de sistemas baseados em modelo de dados relacionais para unir o descasamento de impedância entre os dados e o ambiente de programação de aplicação. Isso é porque a maioria das aplicações, quer LOB, PIM, Information Worker, ou de outro modo, exigem conceitos de modelo de dados como estrutura rica, relações, comportamentos e capacidade de extensão. Esses conceitos de modelo de dados não são adequadamente suportados por modelos de dados existentes, e além disso linguagens de consulta adequadas não existem atualmente para acessar dados se fossem para ser organizados de acordo com o modelo de dados mais avançado.

Candidatos modernos exemplares para um meta-modelo de dados incluem a versão de 1999 da Structured Query Language (SQL99), Common Language Runtime (CLR), Unified Modeling Language (UML) e XML Schema Definition (XSD). Entretanto, o CLR é um runtime de programação imperativa, orientada para objeto, e não tem modelo de dados nativos ou noções de limitações de integridade, relações, ou persistência. SQL99 não tem conceitos de modelar dados como relações, e não tem boa integração de linguagem de programação. A especificação XSD não suporta conceitos como chaves, relações e persistência, e é complexa e tem mapeamento incômodo tanto para runtime como para modelos de banco de dados relacionais. A UML é geral em demasia: requer que desenvolvedores de aplicação adicionem semântica precisa, especialmente para persistência.

Há uma necessidade não atendida na indústria para um modelo de dados e estrutura de suporte correspondente que forneça melhor acesso de aplicação a tipos de dados ricos. Há uma necessidade adicional para uma linguagem de consulta extensível com suporte para tipos de dados ricos que possa ser suportada por um tal modelo de dados.

Sumário

Em consideração das desvantagens acima identificadas da técnica, a presente invenção provê uma linguagem de consulta extensível com suporte para tipos de dados ricos. Várias características exemplares da linguagem de consulta fornecida são discutidas aqui. Além disso, o sistema pode compreender uma aplicação configurada para gerar consultas de acordo com a linguagem de consulta. Também é fornecida uma plataforma para tratar de

consultas que sejam geradas de acordo com a linguagem de consulta fornecida. A plataforma compreende genericamente uma camada de serviços de objeto, uma camada de provedor de mapa, e uma camada de ponte que pode finalmente realizar interface com uma variedade de bancos de dados comercialmente disponíveis.

5 Breve descrição dos desenhos

Os sistemas e métodos para uma linguagem de consulta extensível com suporte para tipos de dados ricos, de acordo com a presente invenção, são adicionalmente descritos com referência aos desenhos em anexo, nos quais:

10 A figura 1 ilustra uma arquitetura de suporte de aplicação para aplicações que acessam dados, de acordo com os sistemas e métodos fornecidos aqui.

A figura 2 ilustra os componentes principais de uma arquitetura CDP e em particular, descreve a canalização de consulta como pode conduzir consultas feitas utilizando a linguagem de consulta extensível para tipos de dados ricos, discutidos em maior detalhe abaixo.

15 A figura 3 ilustra uma vista de grupo Outlook por data para uma vista de agrupamento favorável gerada utilizando um método calculado de acordo com a invenção.

Descrição detalhada

20 Certos detalhes específicos são expostos na descrição e figuras a seguir para fornecer compreensão completa de várias modalidades da invenção. Certos detalhes bem conhecidos freqüentemente associados à tecnologia de software e computação não são expostos na revelação a seguir, entretanto, para evitar obscurecer desnecessariamente as várias modalidades da invenção. Além disso, aqueles com conhecimentos comuns na técnica relevante entenderão que podem pôr em prática outras modalidades da invenção sem um ou mais dos detalhes descritos abaixo. Finalmente, embora vários métodos sejam descritos com referência a etapas e seqüências na revelação a seguir, a descrição como tal é para fornecer uma implementação clara de modalidades da invenção, e as etapas e seqüências de etapas não devem ser tomadas como exigido para pôr em prática a presente invenção.

30 Primeiramente, essa descrição provê um modelo de dados exemplar e mecanismos de suporte de dados relacionados como podem ser utilizados em modalidades exemplares de sistemas e métodos que incorporam uma linguagem de consulta com suporte para tipos de dados ricos como descrito aqui. A seguir, uma variedade de características e aspectos novos da linguagem de consulta exemplar exposta aqui é destacada e discutida. Uma especificação detalhada para uma linguagem de consulta exemplar é exposta no Apêndice A e extensões do mesmo são expostas no Apêndice B. As implementações detalhadas incluem
35 um host de características novas e úteis como será reconhecido por aqueles versados na técnica.

Modelo de dados exemplar e mecanismos de suporte de dados relacionados

Um modelo de dados exemplar e os mecanismos de suporte de dados relacionados podem ser incorporados em um conjunto de tecnologias como a plataforma Active X Data Objects para código gerenciado (ADO.NET), que é projetada para fornecer acesso consistente a fontes de dados como o Servidor Structured Query Language (SQL) da MICROSOFT®, bem como fontes de dados expostas através de Object Linking and Embedding for Databases (OLE DB) e Extensible Markup Language (XML). Aplicações de consumidor de partilhar de dados podem utilizar ADO.NET para conectar-se a essas fontes de dados e recuperar, manipular e atualizar dados.

ADO.NET fatora de forma limpa acesso a dados a partir de manipulação de dados em componentes discretos que podem ser utilizados separadamente ou em tandem. ADO.NET inclui provedores de dados de Estrutura .NET para conectar-se a um banco de dados, executar comandos, e recuperar resultados. Esses resultados são processados diretamente, ou colocados em um objeto de Conjunto de dados (DataSet) ADO.NET para serem expostos para o usuário em um modo ad-hoc, combinado com dados a partir de múltiplas fontes, ou remotos entre níveis. O objeto DataSet ADO.NET pode ser utilizado também independentemente de um provedor de dados de Estrutura .NET para gerenciar dados locais para a aplicação ou fornecidos a partir de XML. Desse modo, ADO.NET provê funcionalidade para desenvolvedores que gravam código gerenciado similar à funcionalidade fornecida para desenvolvedores COM nativos pela tecnologia de Active X Data Objects (ADO) familiar para aqueles versados na técnica.

Em uma modalidade, a plataforma ADO.Net pode ser estendida para fornecer um conjunto rico de serviços de dados para aplicações – através de uma variedade de estruturas de aplicação como estruturas PIM e estruturas LOB – para acessar, manipular e gerenciar dados em um modo que é bem integrado com o ambiente de programação de aplicação. A figura 1 ilustra a colocação dessa funcionalidade em uma arquitetura de suporte de aplicação. A Plataforma de Dados Comum (CDP) 100 pode implementar um conjunto de tecnologias como a plataforma ADO.Net. A Plataforma de Dados Comum (CDP) 100 e tecnologias relacionadas são discutidas em detalhe no pedido de patente US 11/171905.

A arquitetura de suporte de aplicação da figura 1 pode incluir, por exemplo, uma fonte de dados 110 como um banco de dados SQL SERVER®, WinFS®, ou ORACLE®; uma CDP 100 que provê serviços de dados ricos para aplicações e estruturas de aplicação; um conjunto de serviços de estrutura, por exemplo, UAF 120 e Estrutura de LOB 130 que estendem e aumentam a funcionalidade de CDP 100; um conjunto de classes de dados, por exemplo, 122, 132, 140 que encapsulam funcionalidade de estrutura e lógica de aplicação geral; e qualquer número de aplicações 150, 160, 170 que consomem funcionalidade fornecida pela CDP 100 e estruturas 120, 130 e/ou classes 122, 132, 140.

Um modelo de dados que é suportado pela CDP 100 pode compreender, por exemplo, o Modelo de dados de entidade (EDM) desenvolvido pela MICROSOFT® Corporation de Redmond, WA. O EDM estende o modelo de dados relacionais para acomodar múltiplas estruturas de aplicação como LOB, PIM, Gerenciamento, e assim por diante. O EDM define abstração de objeto rico para dados, modelos ricos em semântica como relações de dados, minimiza o descasamento entre as estruturas de aplicação e o modelo de dados, suporta certos comportamentos de aplicação, suporta conceitos relacionais básicos, tipos ricos com herança, e relações, e provê genericamente conceitos de modelagem que capturam semântica de dados independente das armazenagens de dados e níveis de uso. O EDM pode ser incorporado em tecnologias como ADO.NET.

A figura 2 ilustra os componentes principais da arquitetura de CDP 100 e em particular descreve a canalização de consulta como pode conduzir consultas feitas utilizando a linguagem de consulta extensível para tipos de dados ricos em maior detalhe abaixo. O termo “eSQL” será utilizado aqui para se referir a uma linguagem de consulta extensível para tipos de dados ricos, cuja modalidade é exposta em detalhe abaixo. A figura 2 também ilustra como uma consulta flui através de diferentes componentes da CDP 100.

Primeiramente, uma aplicação 200 emite uma consulta contra o provedor de serviços de objeto 210 como uma consulta eSQL. O provedor de serviços de objeto 210 pode compreender um serviço analisador 211 que analisa a consulta e converte a mesma em uma árvore canônica, e transformações de mapeamento 212 que executam quaisquer traduções de mapeamento (a partir de um modelo de dados de aplicação para um EDM como fornecido aqui) na árvore canônica. O provedor de serviços de objeto pode passar então a árvore canônica para baixo para o provedor de mapa 220.

O provedor de mapa 220 encapsula a funcionalidade de mapeamento relacional de objeto. Compreende transformações de mapeamento 222 que executam traduções de mapeamento na árvore canônica. O provedor de mapa passa a árvore canônica para a ponte 230 para serviços de mediação.

A ponte 230 pode compreender um componente de serviços de compensação de consulta 231 que fatora a árvore canônica e executa quaisquer compensações de modelo de dados, então passa para baixo uma ou mais árvores canônicas para o provedor de armazenagem 240.

O provedor de armazenagem 240 traduz a árvore canônica em seu dialeto nativo, por exemplo, em um dialeto SQL como SQL 2000 ou SQL 2005, ou um WinFS ou formato de banco de dados incorporado. O provedor de armazenagem 240 executa a consulta e retorna uma mensagem, por exemplo, uma mensagem que pode ser formatada para ser passada para ou a partir de uma interface de DataReader (uma “DataReader” (Leitora de dados)) para a ponte 230.

A ponte 230 compreende um serviço de conjunto de valor/resultado 232 que reúne, se necessário, resultados a partir de potencialmente múltiplas DataReaders retornadas pelo provedor de armazenagem 240. O resultado dessa operação executada por 232 é uma DataReader única em termos do espaço EDM.

O provedor de mapa 220 simplesmente retorna a seguir a DataReader a partir da ponte 230 para o provedor de serviços de objeto 210. Os serviços de objeto 210 traduzem resultados a partir do provedor de mapa 220 em espaço de objeto. O provedor de serviços de objeto 221 compreende um componente 213 que materializa opcionalmente resultados como objetos, e caches os objetos no mapa de identidade. Finalmente, a aplicação 200 consome a DataReader resultante.

Voltando para vários aspectos salientes da EDM mais especificamente, o EDM é genericamente construído de acordo com quatro conceitos de núcleo: tipos, instâncias, entidades e relações. Esses conceitos podem ser ilustrados utilizando o exemplo de uma aplicação LOB típica. Uma tal aplicação lida com tipos diferentes de dados, por exemplo, pedido, cliente, linhas de pedido, endereço, fornecedor, produto, empregado, e assim por diante.

Em um uso exemplar do EDM, dados de Cliente podem ser considerados uma entidade. Uma entidade representa uma unidade de dados de nível superior com a qual uma aplicação trabalha. O Cliente pode ter vários campos: CustomerID, CompanyName, ContactName, Address e Phone. Cada campo tem um tipo, que determina a estrutura dos dados que entram naquele campo. Por exemplo, CustomerID pode ser uma série de um comprimento fixo. Os campos CompanyName e ContactName podem ser também um tipo de série. O próprio Cliente tem também um tipo; como Cliente é uma entidade, esse tipo pode ser mencionado como um tipo de entidade.

Um campo Address pode ser diferente dos outros campos: possui estrutura interna na forma de outros campos como Cidade, Estado e Código postal. Em EDM, o tipo de um campo como Address é denominado tipo complexo. Ao contrário, o tipo de CustomerID, CompanyName e ContactName podem ser todos tipos simples.

O campo Phone pode consistir em vários números de telefone, cada um dos quais é uma série. Isso é denominado um tipo de coleção.

O tipo especifica a estrutura de dados e certas limitações sobre os valores. Dados efetivos são armazenados em instâncias desses tipos. Qualquer pessoa familiarizada com programação orientada para objeto traçará a analogia óbvia: tipos e instâncias são similares a classes e objetos respectivamente.

Tanto Cliente como Endereço são similares no sentido de que ambos têm estrutura interna: são compostos de múltiplos campos. Porém semântica e operacionalmente, um Cliente é diferente de um Endereço. Cliente atua como a unidade para consulta, operações de troca de dados, transações, persistência e partilha. Um Address, por outro lado, sempre

vide em um Cliente e não pode ser mencionado ou de outro modo influenciado independentemente. No EDM, os requerentes denominam essas unidades de dados de nível superior como entidades. Todos os outros dados são considerados como estando em linha com as entidades.

5 Considere agora os dados exemplares de Pedido. Regras comerciais podem exigir que todo pedido tenha um cliente correspondente. Isso é modelado por uma relação entre a entidade Pedido e a entidade Cliente. Há tipos diferentes de relações suportadas pelo EDM. Uma entre um Pedido e um Cliente é denominada como uma associação. Associações são tipicamente utilizadas para modelar relações não hierarquizadas entre entidades.

10 Cada pedido pode ser composto de várias linhas de pedido. Por exemplo, se você pedir cinco livros na AMAZON.COM®, então as informações sobre cada livro são uma linha de pedido. Isso é modelado como outro tipo de relação, uma composição. Cada OrderLine na composição é uma entidade.

Características novas exemplares de eSQL

15 A seguir, uma variedade de características e aspectos novos de uma linguagem de consulta exemplar é destacada e discutida. Em geral, a linguagem eSQL e extensões da mesma, como fornecido abaixo, podem ser implementadas em um sistema tal como aquele descrito acima com referência às figuras 1 e 2, embora aqueles versados na técnica reconhecerão que vários sistemas e métodos novos associados à modalidade exemplar abaixo
20 podem ser implementados em outros contextos também.

A especificação eSQL exposta abaixo compreende uma variedade de características novas, que incluem porém não são limitadas a representar consultas e instruções de Linguagem de Manipulação de Dados (DMLs – o termo “consulta, como utilizado aqui, se refere a consultas e DMLs) em termos de uma linguagem de modelagem de dados com suporte para tipos de dados ricos (uma linguagem de modelagem de dados ricos) como o EDM, árvores de comando canônicas (Ctrees) que representam o modo programático para especificar consultas e DMLs em termos de uma linguagem de modelagem de dados ricos, funcionalidade de ponte que compensa provedores específicos por manipular as consultas canônicas, uso de desdobrar vista para unificar de forma elegante a estratégia de mapeamento OR através de toda uma semântica de linguagem de modelagem de dados ricos com consulta e atualizações. E a capacidade de estender a linguagem de consulta de núcleo através de extensões que são acionadas a partir de metadados. Além disso, aspectos novos exemplares das especificações eSQL expostas abaixo incluem o que se segue.

35 Suporte de primeira classe para coleções: A modalidade eSQL exemplar fornecida aqui é projetada para ser similar a SQL, e fornece vantagens em relação a SQL. Em geral, versões iniciais de SQL (SQL-92 e anterior) eram intensamente cêntricas em tabela. As tabelas foram tratadas como cidadãos de primeira classe, e linhas/colunas foram tratadas co-

mo segunda classe. E evidentemente, não havia noção de coleções. SQL-99 e dialetos posteriores fornecem suporte para coleções, porém esse suporte foi retroencaixado em SQL-99. Witness por exemplo, adições clunky como não aninhar, aplicar e similares.

Ao contrário, em uma modalidade, eSQL trata coleções como entidades de primeira classe. Por exemplo, expressões de coleção são legais na cláusula a partir de. Não há necessidade de utilizar sintaxes não aninhadas. Subconsultas “em” e “existe” foram generalizadas para tratar de quaisquer coleções – uma subconsulta é simplesmente um tipo de coleção. “e1 em e2” e “existe(e)” são as construções eSQL para executar essas operações. Além disso, muitas das operações definidas (união, interseção, exceto) operam em coleções. Junções operam também em coleções.

Tudo é uma expressão: SQL padrão sempre teve um sistema de dois níveis – subconsultas (tabelas) e expressões (linhas, colunas). Em uma modalidade, para fornecer suporte de primeira classe para coleções e coleções aninhadas, eSQL adota um modelo muito mais simples – tudo é uma expressão. Por exemplo, o que segue são todas consultas eSQL legais:

```
1+2 *3
"abc"
row(1 as a, 2 as b)
{1, 3, 5}
e1 union all e2
distinct(e1)
```

Tratamento uniforme de subconsultas. Operando a partir de sua visão do mundo, cêntrica em tabela, SQL tipicamente tendeu a executar interpretação contextual de subconsultas. Por exemplo, em SQL, uma subconsulta na cláusula a partir de é considerada como sendo um multiconjunto (tabela), enquanto a mesma subconsulta utilizada na cláusula selecionar é considerada como sendo uma subconsulta escalar. Ao longo de linhas similares, uma subconsulta utilizada no lado esquerdo de um operador in é considerada como sendo uma subconsulta escalar, enquanto se espera que o lado direito seja uma subconsulta de multiconjuntos.

Em uma modalidade, eSQL elimina essas diferenças. Uma expressão tem uma interpretação uniforme que não depende do contexto no qual é utilizada. eSQL pode considerar todas as subconsultas como sendo subconsultas de multiconjunto. Se um valor escalar for desejado a partir da subconsulta, eSQL pode fornecer o operador elemento que opera em uma coleção (nesse caso, a subconsulta) e extrair o valor singleto a partir da coleção.

Evitar coerções implícitas desnecessárias: um efeito colateral relacionado do problema descrito acima é a noção de conversão implícita de subconsultas em valores escalares. Especificamente, em SQL, um multiconjunto de linhas com um único campo é implíci-

tamente convertido em um valor escalar cujo tipo de dados é aquele do campo. Ao contrário, modalidades de eSQL não suportam essa coerção implícita. ESQL pode fornecer o operador elemento para extrair um valor singlete a partir de uma coleção, e uma cláusula selecionar valor para evitar criar um row-wrapper durante uma expressão de consulta.

- 5 Select_Value – evitar o wrapper de linha implícito: SQL é de um certo modo ambíguo sobre o tratamento do resultado a partir de uma consulta. A cláusula selecionar em uma subconsulta SQL cria implicitamente um row wrapper em torno dos itens na cláusula. Isso, evidentemente, quer dizer que não podemos criar coleções de escalares ou objetos – toda coleção é uma coleção de linhas (com um campo, se necessário). SQL disfarça esse problema permitindo uma coerção implícita entre um tipo de linha com um campo, e um valor
- 10 singlete do mesmo tipo de dados.

- Em uma modalidade, eSQL provê a cláusula selecionar valor para pular a construção de linha implícita. Somente um item pode ser especificado em uma cláusula selecionar valor. Quando uma tal cláusula é utilizada, nenhum wrapper de linha é construído em torno
- 15 dos itens na cláusula selecionar, e uma coleção do formato desejado pode ser produzida. ESQL também provê o construtor row para construir linhas arbitrárias. Assim, uma linha selecionar valor (e1, e2, ...) é extremamente equivalente a selecionar e1, e2, ...

- Em outra modalidade, a semântica de selecionar poderia evitar a construção de linha para o caso de item único, e utilizar o construtor row explicitamente se necessário. Se a
- 20 cláusula selecionar contiver múltiplos itens, continuaria a se comportar como em SQL.

 Uma abordagem alternativa e mais elegante é não permitir construções da forma “select e1, e2 from...” e em vez disso forçar as pessoas a utilizarem “select row(e1, e2) from...” sempre.

- Correlação esquerda e aliasing: em SQL padrão, expressões em um dado escopo
- 25 (uma cláusula única como selecionar, a partir de etc.) não podem referenciar expressões definidas anteriormente no mesmo escopo. Alguns dialetos de SQL, incluindo T-SQL, suportam efetivamente formas limitadas desses na cláusula a partir de, porém a sintaxe para utilizar tais construções é clunky e requer operações aplicar e não aninhar.

- Em uma modalidade, eSQL generaliza correlações esquerdas na cláusula a partir
- 30 de, e trata as mesmas de forma uniforme. Expressão na cláusula a partir de podem referenciar definições anteriores (definições à esquerda) na mesma cláusula sem a necessidade de sintaxe especial. ESQL também impõe restrições adicionais em consultas que envolvem cláusulas agrupar por. Expressões na cláusula selecionar, cláusula ter, etc. dessas consultas podem se referir somente a chaves agrupar por via seus aliases. Construções como as
- 35 que se seguem – que eram legais em SQL – são ilegais em eSQL:

 Select t.x + t.y from T as t group by t.x + t.y

 O modo correto para fazer isso em eSQL é:

Select k from T as t group by (t.x + t.y) as k

Referenciar colunas (propriedades) de tabelas (coleções): Em uma modalidade, todas as referências de coluna em eSQL devem ser qualificadas com a alias de tabela. A seguinte construção (considerando que “a” é uma coluna legal da tabela “T”) é legal em SQL, porém não em eSQL:

Select a from T

A forma aprova eSQL é:

Select t.a as a from T as t

Referências não qualificadas apresentam vários problemas. Podem ser ambíguas. Sofrem de questões de captura interna/externa durante evolução de esquemas, e além disso, complicam a implementação. Consultas bem projetadas em SQL já utilizam referências de coluna qualificadas para trabalhar em torno desses problemas. O que eSQL faz é simplesmente executar essa diretriz, e isso não torna a linguagem mais complexa.

Navegação através de objetos: SQL utiliza a notação “.” Para referenciar colunas de (uma linha de) uma tabela. E uma modalidade, eSQL estende essa notação (novamente tomada emprestada intensamente a partir de linguagens de programação) para suportar navegação através de propriedades de um objeto. Por exemplo, se “p” for uma expressão do tipo Pessoa:

p.Address.City

é a sintaxe eSQL para referenciar a cidade do endereço dessa pessoa. Muitos dialetos de SQL já suportam essa sintaxe.

Navegação através de coleções: SQL não fornece um mecanismo fácil para navegação através de coleções aninhadas. Em uma modalidade, eSQL provê atalhos sintáticos para tratar desses casos. O .. operador em eSQL permite projeção de uma expressão a partir de uma coleção. Por exemplo, “a..b” é realmente ‘syntactic sugar’ para “select value t.b from a as t”. Similarmente, “a..b(c*2)” é ‘syntactic sugar’ para “select value t2.c*2 from a as t1, t1.b as t2”

O operador “.” Em eSQL permite que usuários podem elementos a partir de uma coleção. Isso é similar ao operador “[]” em XPath. Uma construção da forma “a.?p” é realmente um atalho para “select value t from a as t where p”. Por exemplo, “p.?(id = 1)” realmente significa “select value pO from p as pO where pO.id = 1”

Ao trabalhar com herança, é frequentemente útil ser capaz de selecionar instâncias de um subtipo a partir de uma coleção de instâncias de supertipo. O operador oftype em eSQL (similar a oftype em Sequências C#) provê essa capacidade. Logicamente, oftype(c,T) é equivalente a “select value treat(x as T) (from c as x where x is of T)”

Eliminar o *: SQL suporta a sintaxe * não qualificada como uma alias para a linha inteira, e a * syntax (t.*) qualificada como um atalho para os campos daquela tabela. Além

disso, SQL permite um agregado count(*) especial, que inclui nulos. Consultas bem projetadas em SQL não utilizam tais sintaxes (pelo menos as variantes "select*" e "select t.*"). Utilizar "*" é perigoso na presença de evolução de esquemas. Parte do problema é que SQL não tinha meio para os usuários selecionarem a partir de linhas inteiras.

- 5 Em uma modalidade, eSQL não suporta a construção "*". Consultas de SQL da forma "select * from T" e "select T1.* from T1, T2..." podem ser expressas em eSQL como "select value t from T as t" e "select value t1 from T1 as t1, T2 as t2..." respectivamente. Adicionalmente, essas construções tratam de herança (capacidade de substituição de valor) graciosamente, enquanto as variantes "select*" são limitadas a propriedades de nível superior do tipo declarado. As modalidades de eSQL não suportam também o agregado count(*).
10 Em vez disso, suporta a construção count(group) para tratar desse caso.

 Alterações no grupo por: Como descrito anteriormente, em uma modalidade, eSQL suporta aliasing de chaves agrupar por, e correlações esquerdas entre essas chaves. Na realidade, as expressões na cláusula select e cláusula ter devem se referir às chaves agrupar por através dessas aliases. A cláusula agrupar por produz implicitamente um agregado aninhar para cada partição de agrupar por – essa expressão é mencionada como "group". Funções de agregado na lista de seleção etc. devem se referir a essa expressão para executar agregação. Por exemplo:

```

20           Select kl, count(group), sum(group..(t.a))
           From T as t
           Group by t.b + t.c as k1
           É a sintaxe eSQL para a seguinte consulta SQL:
           Select b + c, count(*), sum (a)
           From T
25           Group by b + c

```

 Agregados baseados em coleção: agregados SQL são difíceis de entender. Em uma modalidade, eSQL suporta dois tipos de agregados. Agregados baseados em coleção operam em coleções, e produzem o resultado agregado. Esses podem aparecer em qualquer lugar na consulta, e não exigem a necessidade de uma cláusula agrupar por. Por exemplo, a seguinte consulta eSQL é legal:

```

30           Select t.a as a, count({1,2,3}) as b from T as t

```

 Em uma modalidade, eSQL também suporta agregados do estilo SQL, e converte implicitamente os mesmos em agregados baseados em coleção (com base na coleção de "grupo".) Por exemplo, a seguinte consulta em eSQL é legal:

```

35           Select a, sum(t.b) from T as t group by t.a as a;
           E é internamente traduzido em uma consulta da forma:
           Select a, sum(group..(t.b)) as b from T as t group by t.a as a;

```

Em uma modalidade, eSQL não suporta o agregado count(*). Utilize a construção count(group) em vez disso.

Inserção: em uma modalidade, instrução INSERT..VALUES de eSQL diferente de T-SQL. Ao contrário de T-SQL, eSQL não permite que uma lista de coluna seja especificada em um inserto. Há dois motivos para isso. Primeiramente, o EDM não tem conceito de valores default para colunas; em segundo lugar, a abordagem de lista de coluna não se presta bem a manipular herança (capacidade de substituição de valor).

Deletar, atualizar: ao contrário de T-SQL, em uma modalidade, eSQL não permite uma cláusula from adicional em instruções de deletar e atualizar. Para instruções deletar, isso não apresenta um problema, uma vez que a consulta pode ser gravada com uma subconsulta. Para instruções de atualizar, entretanto, a cláusula a partir de adicional também ajuda a produzir novos valores a serem utilizados na cláusula Set.

Métodos e Propriedades levantados: A Query Language Do WINDOWS® (WinQL) permite navegação através de coleções via o operador “.”, se a própria coleção não tiver uma propriedade do mesmo nome. WinQL também permite filtração de elementos de uma coleção via a construção “[]” – similar a OPath. Em uma modalidade, eSQL utiliza o operador “..” e “.?” para essa finalidade. Novamente, evolução de esquemas (e captura interna) é a principal razão que eSQL escolhe para distinguir entre navegação de coleção e navegação de objeto. E eSQL evita deliberadamente o uso de “[]” para predicados para evitar questões de ambigüidade.

Semântica de ordenar por: WinQL especifica que a cláusula ordenar por é processada antes da cláusula select. Isso é diferente de SQL onde a cláusula ordenar por é logicamente processada após a cláusula select. Em uma modalidade, eSQL pode estar mais próximo a SQL nesse aspecto, enquanto WinQL toma uma abordagem mais semelhante a Xquery. Qualquer abordagem é razoável e em alguns modos, o modelo WinQL é melhor; entretanto, a abordagem WinQL pode não ser suficientemente melhor para justificar uma alteração em comportamento para usuários de SQL.

SQL-92 limita na realidade a cláusula ordenar por para conter somente referências a itens na cláusula select. A maioria das implementações permite que a cláusula ordenar por referencie outros itens que estão no escopo atualmente. Em uma modalidade, eSQL pode seguir o estilo mencionado por último.

Além das implementações específicas explicitamente expostas aqui, outros aspectos e implementações serão evidentes para aqueles versados na arte a partir da consideração do relatório descritivo revelado aqui. Pretende-se que o relatório descritivo e implementações ilustradas sejam considerados somente como exemplos, com um espírito e escopo verdadeiros das reivindicações que se seguem.

Apêndice A: Especificação detalhada para Linguagem de consulta exemplar

O que se segue é uma especificação exemplar, não limitadora para uma linguagem de consulta exemplar como considerado aqui. Após a especificação abaixo se encontra uma especificação exemplar para extensões para uma tal linguagem de consulta.

1. Resumo funcional

1.1 Descrição de espaço problema

O CDP provê uma infra-estrutura de mapeamento relacional de objeto que pode ser consumida por aplicações e estruturas. WinFS provê uma experiência de armazenagem integrada para uma ampla variedade de aplicações de mesa. Tanto WinFS como CDP são construídos sobre o Modelo de Dados de entidade (EDM). O EDM é um modelo de dados relacionais estendido que suporta conceitos relacionais básicos, tipos ricos com herança e relações. Os usuários necessitam a capacidade de emitir consultas ricas contra seus dados expressos em termos do EDM.

Embora SQL tenha sido historicamente a linguagem de consulta de escolha para acesso a banco de dados, apresenta muitas desvantagens. Falta de suporte para tipos ricos, construções não ortogonais etc., são somente algumas dessas. ESQL tenta tratar muitos desses problemas com SQL. ESQL é projetado para ser similar a SQL e fornece alguma infra-estrutura adicional e aperfeiçoamentos em relação a SQL.

Tanto WinSF como CDP alavanca tecnologias de acesso de dados .NET existentes – ADO .NET – para acessar e manipular dados. Um Provedor de Dados .NET provê um modo de um certo modo uniforme para acessar dados a partir de qualquer fonte de dados via Conexões, Comandos, DataReaders e outros tais objetos. Comandos para um Provedor de dados .NET são expressos como séries, e devem estar no dialeto nativo do provedor (mais especificamente, a fonte de dados que é fronteado pelo provedor). Como parte dos esforços de CDP/WinFS, três novos provedores – o Provedor de Objeto, o Provedor EDM, e o Provedor WinFS – serão fornecidos, e todos esses utilizarão eSQL como o dialeto nativo.

2. Visão geral e Princípios de desenho

2.1 Princípios de desenho

Suporte de primeira classe para coleções e objetos

eSQL deve tornar fácil para os usuários consultar continuamente seus objetos e coleções.

Ortogonalidade

Construções em eSQL devem ser ortogonais. Não haverá interpretações contextuais sobre o uso de uma dada construção.

Capacidade de composição

Construções em eSQL devem ser componíveis em outras construções.

Similaridade com SQL

eSQL deve permanecer uma linguagem semelhante a SQL o máximo possível. A

menos que o desvio de SQL seja necessário (por exemplo, quando exigido pelo raciocínio acima), eSQL deve manter fidelidade com construções SQL.

3 Visão geral de linguagem

eSQL inclui a noção de uma expressão. Uma expressão, como em outras linguagens de programação, é uma construção que pode ser avaliada para produzir um valor. Uma consulta eSQL é qualquer expressão legal. eSQL provê os seguintes tipos de expressões.

3.1 Literais

Um literal, como na maioria das linguagens de programação, é simplesmente um valor constante. eSQL provê suporte nativo para literais numéricos e de série, e o literal nulo especial (que tem um tipo de dados de flutuação livre).

3.2 Parâmetros

Parâmetros em eSQL permitem expressões (e consultas) para valores de referência definidos fora da consulta.

3.3 Referências variáveis

Referências variáveis em eSQL permitem que expressões se refiram a outras expressões que foram definidas (e denominadas) anteriormente no mesmo escopo.

3.4 Operadores Bultin

eSQL provê um pequeno conjunto de operadores incorporados. Isso inclui operadores aritméticos como +, -, *, /, operadores de comparação (<, >, ...), operadores lógicos (e, ou, não etc.), e alguns operadores diversos (caso, entre, etc.).

3.5 Funções

eSQL não define nenhuma função incorporada própria. Expressões de função eSQL permitem o uso de qualquer função definida por usuário (que é visível para eSQL).

3.6 Navegação de objeto, Métodos

Essas expressões eSQL permitem que propriedades de objeto e métodos sejam referenciados e manipulados naturalmente.

3.7. Funções agregadas

Funções agregadas em eSQL, como suas cópias SQL, permitem operações de dobrar em coleções. eSQL suporta um agregado aninhar, e qualquer agregado definido por usuário. Funções de agregado eSQL são baseadas em coleção em que podem ser utilizadas em qualquer lugar em uma expressão. eSQL também suporta agregados baseados em grupo como SQL que são somente utilizáveis em expressões de consulta.

3.8 Construções

eSQL provê três tipos de construtores. Construtores de linha podem ser utilizados para criar um valor de tipo estrutural (registro) anônimo. Construtores de objeto (construtores em C# etc.) podem ser utilizados para construir (denominados) objetos definidos por

usuário. Construtores de multiconjunto podem ser utilizados para construir multiconjuntos de outros valores.

3.9 Operações de tipo

5 eSQL provê operações que permitem que o tipo de uma expressão (valor) seja consultado e manipulado. Isso inclui operações de interrogação como `is of` (similar à expressão `is` em C#), casts de subtipo-supertipo como `tratar` (a expressão `as` em C#) e `oftype` (o método `OfType` em Sequências C#), e operadores de conversões como `cast`.

3.10 Operadores set

10 eSQL provê um conjunto rico de operações set. Isso inclui operadores set como SQL como união, interseção, exceto e existe. Além disso, eSQL suporta que operadores dupliquem eliminação (distinto), teste de qualidade de membro (`in`), junções (junção).

3.11 Expressões de consulta

Expressões de consulta representam consultas SQL `select-from-where-...` clássicas.

4. Detalhes de linguagem – expressões eSQL

4.1 Literais

Nulo

O literal nulo é utilizado para representar o valor nulo para qualquer tipo. Um literal nulo é considerado como sendo de flutuação livre (i.e.) é compatível com qualquer tipo. Nulos digitados podem ser criados via um operador `Cast` sobre um literal nulo:

20 `Select Foo (nul.l, v. a + v. b,) from values as v --`
 regular null literal
`select cast (null as Int16) from values as v --`
 typed null

25 Vide a seção 9.6 para regras ou onde literais nulos de flutuação livre podem ser utilizados.

Booleano

Literais Booleanos são representados pelas palavras-chave `verdadeiro` e `falso`.

Número inteiro (Int32, Int64)

30 Literais de número inteiro podem ser do tipo `Int32` ou `Int64`. Um literal `Int32` é uma série de caracteres numéricos. Um literal `Int64` é série de caracteres numéricos seguida por um `L` em letra maiúscula.

Decimal

Um número de ponto fixo (decimal) é uma série de caracteres numéricos, um período `'.'` e outra série de caracteres numéricos seguida por um `M` em letra maiúscula.

35 Flutuar, dobro

Um número de ponto flutuante de precisão dupla é uma série de caracteres numéricos, um período `'.'` e outra série de caracteres numéricos possivelmente seguidos por um

expoente. Um número de ponto flutuante de precisão única (ou flutuante) é uma sintaxe de número de ponto flutuante de precisão dupla seguido pelo f em letra minúscula.

Série

Uma série é uma série de caracteres encerrados em aspas. Aspas podem ser aspas únicas (') ou aspas duplas ("). Identificadores com aspas podem ser representados com [] ou ""'. Por default, somente o estilo [] é permitido, e construções ""' são tratadas como literais de série. O analisador eSQL pode permitir identificadores com aspas de estilo ""' (que serão controlados por opções de analisador – fora do escopo desse documento); nesse caso, literais de série podem utilizar somente aspas únicas. Vide a seção 9.1.2 para mais informações sobre isso:

'hello'

"x"

"This is a sting"

'so is THIS'

Outros literais

eSQL não suporta literais de outros tipos de dados de forma nativa (DateTime, Guid etc.).

4.2 Parâmetros

Parâmetros são variáveis que são definidas fora da linguagem de consulta, normalmente via um API de ligação utilizado por uma linguagem hospedeira. Cada parâmetro tem um nome e um tipo. Nomes de parâmetro são descritos com o símbolo '@' como um prefixo antes do nome para remover ambiguidades dos mesmos a partir dos nomes de propriedades ou outros nomes definidos na consulta. O API de ligação de linguagem hospedeira provê APIs para parâmetros de ligação. O seguinte exemplo mostra o uso de um parâmetro em uma expressão:

Select c from customers as c where c.Name = @name

4.3 Variável

Uma expressão Variável é uma referência a uma expressão (denominada) definida anteriormente no escopo atual. Uma referência de variável deve ser um identificador válido como definido em 6. O seguinte exemplo mostra o uso de uma variável em uma expressão:

Select c from customers as c

4.4 Operadores incorporados

eSQL provê um número de operadores incorporados, como a seguir.

4.4.1 Aritmética

Operação	Símbolo
Soma	+

Subtração	-
multiplicação	*
Divisão	/
Modulo	%
negação	-

Operadores aritméticos são definidos para primitivos numéricos como Int32, Int16, Int64, Dobro, Único e decimal. Para operadores binários, promoção de tipo implícito ocorre para os operandos antes da aplicação do operador. Os tipos de resultado de todos os operadores aritméticos são iguais aos tipos de operando promovido. A soma também é definida para série e atua como concatenação.

4.4.2 Comparação

Operação	Símbolo
Igualdade	=
Desigualdade	<>, !=
Menor do que	<
Menor ou igual	<=
Maior do que	>
Maior ou igual	>=

Operadores de comparação são definidos para primitivos numéricos como Byte, Int32, Int16, Int64, Dobro, Único e Decimal; série e DateTime. A promoção do tipo implícito ocorre para os operandos antes da aplicação do operador de comparação. Os operadores de comparação sempre fornecem Booleano.

Igualdade e desigualdade são definidos para o tipo Booleano e qualquer tipo de objeto que tenha identidade. Objetos não primitivos com identidade são considerados iguais sem partilharem a mesma identidade.

4.4.3 Lógico

Operação	símbolo	Sinônimo
E	E	&&
Ou	Ou	
não	não	!

Operadores lógicos são definidos para tipos Booleanos somente —e sempre retornam um tipo Booleano.

4.4.4 Caso

A expressão caso tem semântica similar à expressão de caso TSQL. A expressão de caso é utilizada para fazer uma série de testes condicionais para determinar qual expressão fornecerá o resultado apropriado.

- 5 Se $b_1, b_2, \dots, b_{n-1}$ forem expressões do tipo Booleano e $e_1, e_2, \dots, e_n$ forem expressões de algum tipo S, então a seguinte expressão é uma expressão de caso válido que fornece um único valor do tipo S.

```

case
  when  $b_1$  then  $e_1$ 
  when  $b_2$  then  $e_2$ 
  ...
  [ else  $v_n$  ]
end

```

- 10 Essa forma da expressão de caso aplica uma série de uma ou mais expressões Booleanas para determinar a expressão resultante correta. A primeira expressão quando a avaliar para o valor verdadeiro fornece a expressão-então correspondente. As expressões restantes não são avaliadas. Se nenhuma das condições quando for satisfeita, o resultado é o resultado de avaliar a expressão else (se uma existir) ou nula (se não houver expressão else). Observe, promoção do tipo implícito pode ocorrer entre expressões de resultado diferentes ($e_1, e_2, \dots, e_n, v_n$) para determinar o tipo de resultado efetivo da expressão de caso.

15 4.4.5 Entre

A expressão entre tem a mesma semântica que a expressão entre SQL. Determina se uma dada expressão resulta em um valor entre um limite inferior e um limite superior:

E_1 between e_2 and e_3

- 20 Se e_1, e_2 e e_3 forem todas expressões de algum tipo T, então a expressão acima é uma expressão válida de operador entre. A expressão entre é uma abreviatura de escrever dois operadores de comparação sobre o mesmo valor. A sintaxe entre é útil se e_1 for uma expressão complexa:

$e_1 \geq e_2$ e $e_1 \leq e_3$

A expressão não entre pode ser utilizada para indicar o inverso da expressão entre:

e_1 não entre e_2 e e_3

->

$e1 < e2$ ou $e1 > e3$

4.4.6 Como

A expressão como tem a mesma semântica como definido para TSQL. Determina se uma série casa com um padrão, resolvendo verdadeiro se a série casa e falso se não casa.

- 5 Se casar, padrão e escape são todas expressões tendo tipo Série, então o que se segue são expressões como válidas que fornecem um valor Booleano:

match like pattern
match not like pattern
match like pattern escape escape
match not like pattern escape escape

A sintaxe de série padrão é igual à sintaxe de série padrão TSQL.

4.4.7 Teste nulo (é nulo)

- 10 O operador é nulo é utilizado para determinar se uma expressão resulta no valor nulo. Se a expressão resultar no valor nulo a expressão resulta no valor verdadeiro. De outro modo a expressão resulta no valor falso. Se e for uma expressão, as seguintes expressões são expressões válidas que fornecem um Booleano:

e is null

e is not null

Use o operador é nulo para determinar se o elemento de uma junção externa não é nulo:

*select c from cs as c left outer join as ds d
 where d is not null and d.x = @x*

- 15 Use o operador é nulo para determinar se um membro tem um valor efetivo ou não:

select c from cs as c where c.x is not null

4.5 Acesso a membro

O acesso a membro também é comumente conhecido como o operador de ponto. Use o operador de acesso a membro para fornecer o valor de uma propriedade ou campo

de uma instância de objeto. Se m for um membro do tipo T que tem tipo M , e t for uma instância do tipo T , então $t.m$ é uma expressão de acesso a membro legal que fornece um valor de tipo M :

```
select p.Name.FirstName from Person as p;
```

4.6 Invocação de método

5 Invocação de método é uma chamada explícita a um método. Fornece o resultado da chamada de método. Os métodos podem ser métodos de instância de um tipo, métodos estágios de um tipo, ou métodos estáticos globais (ie) de função independente.

Se m for um método do tipo T que retorna o tipo M e tem parâmetros de tipos ($P_1, P_2, \dots, P_n$), e t é uma instância de tipo T e e_1 é uma expressão do tipo P_1 e assim por diante, então $t.m(e_1, e_2, \dots, e_n)$ é uma invocação de método válido que fornece um valor de tipo M .

Se m for um método do tipo T que retorna tipo M e tem parâmetros de tipos ($P_1, P_2, \dots, P_n$), e qn é um nome qualificado que se refere ao tipo T e e_1 é uma expressão do tipo P_1 e assim por diante, então $qn.m(e_1, e_2, \dots, e_n)$ é uma invocação de método estágio válido que fornece um valor do tipo M .

Se m é um método T que retorna o tipo M e tem parâmetros de tipos ($P_1, P_2, \dots, P_n$), e qn é um nome qualificado que se refere ao prefixo de espaço de nome do método, e e_1 é uma expressão do tipo P_1 e assim por diante, então $qn.m(e_1, e_2, \dots, e_n)$ é uma invocação de método estágio válido que fornece um valor do tipo M .

20 Um exemplo de uma invocação de método que poderia aparecer em uma consulta eSQL poderia ser o uso de métodos incorporados em tipos de dados primitivos:

```
select tsql.substring(c.Name, 0, 2) from customers c
```

4.6.4 Resolução de sobrecarga

Mais de um método pode ser definido com o mesmo nome. Para determinar qual método é referenciado por uma dada expressão, siga essas regras. A primeira regra aplicada em sucessão que fornece somente um único método é o método resolvido.

1. Um método com o mesmo número de parâmetros existe.
2. um método onde cada tipo de argumento casa exatamente com o tipo de parâmetro ou é o literal nulo.
3. Um método onde cada tipo de argumento casa exatamente ou é um subtipo do tipo de parâmetro ou o argumento é o literal nulo. O método com o número mínimo de conversões de subtipo vence.

4. Um método onde cada tipo de argumento casa exatamente, é um subtipo de, ou pode ser promovido para o tipo de parâmetro ou argumento é o literal nulo. O método com o número mínimo de conversões de subtipo e promoções vence.

5. A expressão de invocação de método é ambígua.

5 Observe que mesmo se um único método puder ser extraído utilizando essas regras, os argumentos ainda podem não casar com os parâmetros. Se esse for o caso, então os argumentos estão em erro.

4.7 Enums

10 Enums são valores numéricos integrais digitados fortemente que são mencionados pelo nome. Um enum é especificado utilizando um nome qualificado para consultar o tipo de enum seguido por um operador de ponto e o nome do enum. Se `qn` for o nome qualificado para algum enum `E`, e `en` for o nome de um valor da enumeração, então `qn.en` é uma referência de enum válida:

`Color.Red`

4.8 Operações de tipo

15 eSQL provê um número de operações para trabalhar com tipos de valores de expressão.

4.8.1 Is Of

A expressão `Is of` verifica se uma expressão é uma instância de um tipo especificado.

```
e is of ( T )
e is not of ( T )
e is of ( only T )
e is not of ( only T )
```

20 Se `e` for de algum tipo de compilar tempo `S`, e `S` é um subtipo/supertipo de `T`, então as expressões acima são expressões válidas, e retornam um resultado do tipo Booleano. Se o tipo de `e` for determinado em runtime como sendo `T` (ou algum subtipo de `T`), então `e is of T` retorna Verdadeiro. Se `e` for nulo em runtime, essa expressão retorna nulo; de outro modo o resultado dessa expressão é falso.

25 Se o modificador único `for` especificado, então a expressão retorna verdadeiro somente se `e` era exatamente do tipo `T`, e não qualquer de seus subtipos. As expressões `e is not of (T)` e `e is not of (only T)` são equivalentes de forma sintática a `not (e is of (T))`, e `not(e is of (only T))` respectivamente. Se `S` não for um subtipo ou um supertipo de `T`, então essa expressão origina um erro de compilar tempo.

4.8.1.1 Tratar

A expressão tratar tenta redigitar uma dada expressão em termos de um supertipo/subtipo:

`treat ( e as T )`

- 5 Se e tiver tipo S e T for um subtipo de S ou S for um subtipo de T, então a expressão acima é uma expressão `treat` válida e fornece um valor de tipo T. Se nem S ou T for um subtipo do outro, a expressão resulta em um erro de compilar tempo. Se e tiver tipo Funcionário e Gerente for um subtipo de Funcionário então a seguinte expressão fornece o mesmo valor de e, porém digitado como Gerente em vez de Funcionário:

`Treat( e as Manager }`

- 10 Se o valor de e não for na realidade do tipo Gerente, então a expressão fornece o valor nulo.

4.8.1.2 Cast

A expressão `cast` tem semântica similar à expressão `converter TSQL`. A expressão `cast` é utilizada para converter um valor de um tipo em um valor de outro tipo. Somente conversões de certos tipos são suportadas pela linguagem:

`Cast( e as T )`

- 15 Se e for de algum tipo S e S for convertível em T então a expressão acima é uma expressão `cast` válida. T deve ser um primitivo (tipo escalar). O uso de uma expressão `cast` é considerada uma conversão explícita. Conversões explícitas podem truncar dados ou perder precisão. A lista de casts válidos suportados por eSQL é descrita na seção 10.5.

4.8.1.3 OfType

- 20 Uma expressão `OfType` especifica uma expressão de tipo que é utilizada para executar um teste de tipo contra cada elemento de uma coleção. A expressão `OfType` produz uma nova coleção do tipo especificado contendo somente aqueles elementos que eram equivalentes àquele tipo ou um subtipo do mesmo.

- 25 Se ts for uma coleção de tipo `collection<S>` e T for um subtipo de S, então `OfType(ts, T)` é uma expressão de tipo `collection <T>`, que fornece uma coleção de todos os elementos e de ts onde o tipo individual de e, Te, é um subtipo de T. Uma expressão `OfType` é uma abreviatura da seguinte expressão de consulta:

`Select value treat (t as T) from ts as t where t is of T`

Dado que um Gerente é um subtipo de Funcionário, a seguinte expressão produz uma coleção somente de gerentes a partir de uma coleção de funcionários:

OfType (employees, Manager)

Também é possível up castt uma coleção utilizando o filtro de tipo:

OfType (executives, Manager)

- 5 Uma vez que todos os executivos são gerentes, a coleção resultante contém ainda todos os executivos originais, embora a coleção seja agora digitada como uma coleção de gerentes. Não é possível fazer um castt de tipo ilegal conhecido. A seguinte expressão originará um erro de compilar tempo. (Considerando evidentemente, que Endereço não é um subtipo/supertipo de Funcionário):

OfType (employees, Address)

10 4.9 Referências

eSQL suporta os seguintes operadores para referências.

4.9.1 Ref

O operador Ref quando aplicado à variável de correlação para um entityset produz uma referência a uma entidade a partir daquela coleção. Por exemplo, a seguinte consulta
15 retorna referências a cada Pedido:

Select ref(o) from Orders as o

O que se segue são usos ilegais do operador ref – a variável de correlação deve resolver para um conjunto de entidade no escopo atual:

Select ref(o) from (select value o from Orders as o)
as o

4.9.2 CreateRef

- 20 O operador CreateRef pode ser utilizado para fabricar referências a uma entidade em um entityset. O primeiro argumento para o operador é o identificador de entityset (não uma literal de série), e o segundo argumento é uma expressão digitada de registro que corresponde às propriedades chave do tipo entidade:

Select ref(BadOrders, row(o.Id)) from Orders as o

A expressão digitada de registro deve ser estruturalmente equivalente ao tipo chave para a entidade (ie) deve ter o mesmo número e tipos de campos na mesma ordem que as chaves de entidade – os nomes dos campos são irrelevantes.

4.9.3 Chave

- 5 O operador Chave é o inverso do operador CreateRef, e pode ser utilizado para extrair a porção chave de uma expressão com ref de tipo. O tipo retorno do operador Chave é um tipo de registro – com um campo para cada chave da entidade, e na mesma ordem:

```
Select key( ref(BadOrders, row(o.Id)) ) from Orders as o
```

4.9.4. DeRef

- 10 O operador Deref desreferencia um valor de referência, e produz o resultado daquela desreferência:

```
Select deref (o.CustRef) from Orders as o
```

Se r for uma referência do tipo ref<T>, então Deref(r) é uma expressão de tipo T que fornece a entidade referenciada por r. Se o valor de referência for nulo, ou estiver pendente (ie) o alvo da referência não existe, o resultado do operador Deref é nulo.

4.9.4 Desreferenciar atributo

- 15 Pode-se navegar através de uma referência via operador ".".
O seguinte snippet extrai a propriedade Id (do Cliente) por navegar através da propriedade CustRef:

```
Select o.CustRef.Id from Orders as o
```

Se o valor de referência for nulo, ou estiver pendente (i.e.) o alvo da referência não existe, o resultado é nulo.

20 4.10 Construtores

eSQL provê 3 tipos de construtores – construtores de linha, construtores de objeto e construtores de multiconjunto.

4.10.1 Construtores de linha

- 25 Construtores de linha podem ser utilizados para construir um registro estruturalmente digitado anônimo de um ou mais valores. O tipo de resultado de um construtor de linha é um tipo de linha – cujos tipos de campo correspondem aos tipos dos valores utilizados para construir a linha. Por exemplo, a expressão abaixo:

row(1 as a, "abc" as b, a+34 as c)

produz um valor de tipo Record(a int, b string, c int).

Todas as expressões no construtor de linha devem ser aliased – se nenhum alias é fornecido, então eSQL tenta gerar um alias via as regras de aliasing especificadas em 0> Expressões no construtor de linha podem não se referir a aliases definidas anteriormente (à esquerda) no mesmo construtor.

t é ilegal para duas expressões no mesmo construtor de linha para ter a mesma alias. O operador dot(.) pode ser utilizado para extrair campos a partir de um registro (similar à extração de propriedades de um objeto).

4.10.2 Construtores NamedType (Inicializadores NamedType)

eSQL permite que construtores do tipo denominado (inicializadores) sejam utilizados para criar instâncias de tipos complexos denominados e/ou tipos de entidade. A expressão abaixo cria uma instância de um tipo Pessoa (considerando que o tipo Pessoa tenha aqueles atributos):

person ("abc", 12)

Os argumentos para o construtor são considerados como estando na mesma ordem que a declaração dos atributos do tipo.

4.10.3 Construtores de coleção

O construtor de multiconjunto cria uma instância de um multiconjunto a partir de uma lista de valores. Todos os valores no construtor devem ser do tipo T mutuamente compatível, e o construtor produz uma coleção do tipo Multiset<T>. As expressões abaixo criam um multiconjunto de números inteiros:

Multiset(1, 2, 3)

{1, 2, 3}

4.11 Operações do conjunto

Essa seção contém uma variedade de operadores de conjunto.

4.11.1 Distinto

A expressão distinta é utilizada para converter uma coleção de objetos em um conjunto fornecendo uma nova coleção com todas as referências duplicatas removidas. Se c for uma coleção do tipo collection<T> então distinct(c) é uma expressão distinta válida que fornece uma coleção do tipo collection<T> que não tem duplicatas. O tipo T deve ser comparável igual. A expressão distinta é uma abreviatura de uma expressão selecionar:

Distinct(c) -> select value distinct c from c

4.11.2 Flatten

A expressão alisar é utilizada para converter uma coleção de coleções em uma coleção alisada com todos os elementos iguais, apenas sem a estrutura aninhada. Se c for uma coleção do tipo collection<collection<T>> então flatten(c) é uma expressão alisar válida que fornece uma nova coleção de tipo collection<T>. A expressão alisar é uma abreviatura de utilizar a expressão select:

Flatten(c) -> select value c2 from c as c1, c1.it as c2

4.11.3 Existe

A expressão existe tem a mesma semântica que a expressão existe TSQL. Determina se uma coleção está vazia ou não. Se a coleção não estiver vazia, a expressão existe retorna verdadeira, de outro modo retorna falsa. Se c for uma coleção do tipo Collection<T>, então exists(c) é uma expressão existe válida que fornece um único valor do tipo Booleano.

4.11.4 In

O operador in é utilizado para testar um valor para qualidade de membro em uma coleção. O valor da expressão no lado esquerdo da palavra chave 'in' é buscado na coleção representada pela expressão no lado direito. Se e for uma expressão com tipo T e ts for uma coleção com tipo Collection<S>, e S e T tiverem uma relação de supertipo/subtipo, então as seguintes expressões são expressões de operador-in válido:

e in ts

e not in ts

No exemplo, a seguir, o nome "Bob" é buscado no conjunto de nomes de clientes:

'Bob' in customerNames

Se o valor for encontrado na coleção, o operador-in fornece o valor verdadeiro. Se o valor for nulo ou a coleção for nula, o operador-in fornece o valor nulo. De outro modo, o operador-in fornece o valor falso. A forma não do operador fornece o oposto.

4.11.5 Union, Union all

As expressões union e union all em eSQL têm a mesma semântica que seus equivalentes TSQL. UnionAll produz uma coleção que contém o conteúdo combinado das duas coleções (com duplicatas). Union produz uma coleção com duplicatas removidas.

Se c_1 for uma coleção do tipo $\text{Collection}\langle T \rangle$, e c_2 for uma coleção do tipo $\text{Collection}\langle S \rangle$, onde M é o supertipo comum de T e S , então $c_1 \text{ union } c_2$ e $c_1 \text{ union all } c_2$ são expressões de união válidas que fornecem uma coleção do tipo $\text{Collection}\langle M \rangle$.

5 O exemplo a seguir identifica o conjunto de todos os clientes que moram em Washington, ou são bons clientes. A primeira expressão elimina quaisquer duplicatas (por exemplo, um bom cliente que também mora em Washington), enquanto a segunda expressão preserva todas as duplicatas:

GoodCustomers union WashingtonCustomers
GoodCustomers union all WashingtonCustomers

4.11.6 Interseção

10 A expressão Interseção tem a mesma semântica que a expressão interseção TSQL. Determina a interação de duas coleções. Se c_1 for uma coleção do tipo $\text{Collection}\langle T \rangle$, e c_2 for uma coleção do tipo $\text{Collection}\langle S \rangle$, onde M é o supertipo comum de T e S , então $c_1 \text{ intersect } c_2$ é uma expressão de interseção válida que fornece uma coleção do tipo $\text{Collection}\langle M \rangle$. O exemplo a seguir identifica o conjunto de todos os clientes que moram em Washington e são bons clientes:

GoodCustomers intersect WashingtonCustomers

4.11.7 Exceto

15 A expressão Exceto tem a mesma semântica que a expressão exceto TSQL. determina a diferença em um sentido de duas coleções. Se c_1 for uma coleção do tipo $\text{Collection}\langle T \rangle$, e c_2 for uma coleção do tipo $\text{Collection}\langle S \rangle$, onde M é o supertipo comum de T e S , então $c_1 \text{ except } c_2$ é uma expressão exceto válida fornecendo uma coleção do tipo $\text{Collection}\langle T \rangle$. No exemplo a seguir, o conjunto de todos os clientes que não são clientes ruins é identificado:

AllCustomers except badCustomers

4.11.8 Sobreposições

25 A expressão sobreposições determina se duas coleções têm um elemento comum. Se c_1 for uma coleção do tipo $\text{Collection}\langle T \rangle$, e c_2 for uma coleção do tipo $\text{Collection}\langle S \rangle$, onde M é o supertipo comum de T e S , então $c_1 \text{ overlaps } c_2$ é uma expressão de sobreposições válida que fornece um resultado do tipo booleano. O exemplo a seguir verifica se algum cliente de Washington é um bom cliente:

WashingtonCustomers overlaps goodCustomers

E é um atalho sintático para:

exists (WashingtonCustomers intersect goodCustomers)

4.11.9 Elemento

A expressão elemento extrai o elemento a partir de uma coleção singleto. Se c for uma coleção do tipo Collection<T>, então element(c) é uma expressão de elemento válida que fornece uma instância do tipo T. O exemplo a seguir tenta extrair o elemento singleto a partir de um conjunto de clientes ruins:

Element (badCustomers)

Se a coleção estiver vazia, ou tiver mais de um elemento, então a expressão elemento retorna um nulo.

4.12 Navegação de coleção

eSQL provê um número de construções sintáticas que permitem manipulação mais fácil de coleções.

4.12.1 Proteção

O operador ..(collection-project) é utilizado para projetar elementos através de uma coleção. Se c for uma coleção do tipo Collection<T>, e p for uma propriedade de T com tipo Q, então c..p produz uma coleção de tipo Collection<Q>. Mais genericamente, se c for uma coleção do tipo Collection<T>, e e for alguma expressão de tipo Q, então c..e produz uma coleção do tipo Collection<Q> contendo o resultado de avaliar e para cada elemento de c. O seguinte exemplo obtém os nomes com letra maiúscula de todos os departamentos:

Departments..(upper(name))

E é um atalho para a seguinte consulta:

select value upper (d.name) from Departments as d

4.12.2 Filtração

O operador .?(collection-filter) é utilizado para filtrar elementos a partir de uma coleção. Se c for uma coleção do tipo Collection<T>, e e é uma expressão booleana com tipo Q, então c.?e produz uma coleção do tipo Collection<T> contendo somente aqueles elementos

de c que satisfaziam o predicado e. O seguinte exemplo obtém o conjunto de todos os departamentos que são baseados em Seattle:

```
Departments.?(location = 'Seattle')
```

E é um atalho para a seguinte consulta:

```
select value d from Departments as d where
d.location = 'Seattle'
```

4.12.3 Projeção alisada

- 5 O operador ...(*flattened collection-project*) é utilizado para projetar elementos através de uma coleção aninhada. Mais genericamente, se c for uma coleção do tipo *Collection<Collection<T>>*, e e é alguma expressão do tipo Q, então c...e produz uma coleção do tipo *Collection<Q>* contendo o resultado do primeiro alisamento c, e então avaliar e para cada elemento de c. O exemplo a seguir obtém os nomes com letra maiúscula de todos os
- 10 funcionários, considerando que *Funcionários* é uma propriedade de valor definido de Departamento:

```
Departments.Employees...(upper(Name))
```

E é um atalho para a seguinte consulta:

```
select value upper(e.name) from Departments as d,
d.Employees as e
```

4.12.4 Filtração alisada

- 15 O operador ..?(*flattened collection-filter*) é utilizado para filtrar elementos a partir de uma coleção aninhada. Mais genericamente, se c for uma coleção de tipo *Collection<Collection<T>>*, e e for alguma expressão booleana, então c...?e produz uma coleção do tipo *Collection<T>* contendo o resultado do primeiro alisamento c, e então filtrando aqueles elementos do resultado que não satisfazem e. O seguinte exemplo obtém o conjunto de todos os funcionários cujo salário seja maior do que 10000, considerando que *Funcionários* é
- 20 uma propriedade de valor definido do Departamento:

```
Departments.Employees..?(salary > 10000)
```

E é um atalho para a seguinte consulta:

```
select value e
  from Departments as d, d.Employees as e
 where e.salary > 10000
```

Regras de abrangência

Em todos os casos acima, ao navegar através de uma coleção do tipo Collection<T>, um novo escopo é criado, e o elemento de coleção atual é ligado à variável it. Para conveniência sintática, as propriedades (e métodos) de T são também consideradas como estando em escopo, porém são consideradas como sendo atalhos para acesso a membro através da variável it. Por exemplo:

Departments ..Name -> Departments ..(it.Name)

4.13 Funções de agregados

Agregados são expressões que reduzem uma série de valores de entrada (normalmente porém não necessariamente em um valor único). São normalmente utilizados em combinação com a cláusula agrupar por da expressão selecionar, e têm limitações sobre onde podem ser na realidade utilizados. Cada operação de agregado é definida para um ou mais tipos. eSQL não define um conjunto de agregados. Simplesmente se baseia em espaço de metadados circundante para tratar funções de agregados.

Agregados à base de coleção

Agregados baseados em coleção são agregados calculados sobre um conjunto específico de valores. Por exemplo, dado que pedidos é uma coleção de todos os pedidos, pode-se calcular a data de transporte mais cedo com a seguinte expressão:

min(orders..ShipDate)

Expressões nos agregados baseados em coleção são avaliadas utilizando o escopo de resolução de nome ambiente atual.

Agregados baseados em grupo

Agregados baseados em grupo são calculados sobre um grupo como definido pela cláusula agrupar por. Para cada grupo no resultado, um agregado separado é calculado utilizando os elementos em cada grupo como entradas para o cálculo agregado. Quando uma cláusula agrupar por é utilizada em uma expressão selecionar, somente nomes de expressão de agrupamento, agregados ou expressões constantes podem estar presentes na projeção ou cláusula ordenar por. O seguinte exemplo calcula a quantidade média enco-

mendada para cada produto:

```
select p, avg(ol.Quantity) from orderLines as ol
group by ol.Product as p
```

Agregados baseados em grupo são simplesmente atalhos (para preservar compatibilidade com SQL) para expressões baseadas em coleção. Os exemplos acima realmente traduzirão em:

```
select p, avg(group..(ol.Quantity)) from
orderLines as ol
group by ol.Product as p
```

- 5 Onde a expressão grupo se refere ao agregado aninhar implicitamente produzido pela cláusula agrupar por. É possível ter um agregado baseado em grupo sem uma cláusula agrupar por explícita na expressão selecionada. Todos os elementos serão tratados como um grupo único, equivalente ao caso de especificar um agrupamento baseado em constante:

```
select avg(ol.Quantity) from orderLines as ol
→ select avg(group..(ol.Quantity)) from orderLines as
ol group by 1
```

- 10 Expressões no agregado baseado em grupo são avaliadas utilizando o escopo de resolução de nome que seria visível para a expressão de cláusula onde.

- Para preservar aparência e sensação de SQL, agregados baseados em grupo também podem especificar um modificador all ou distinct em sua entrada. Se o modificador distinct for especificado, duplicatas são eliminadas a partir da coleção de entrada de agregado, antes do agregado ser computado. Se o modificador all for especificado (ou nenhum modificado for especificado), então nenhuma eliminação de duplicata é executada. O modificador distinct é realmente uma abreviação sintática para o operador distinct:

Avg (distinct ol.quantity)



Avg (distinctc (ol.quantity))

Distinção entre agregados baseados em coleção e baseados em grupo

Agregados baseados em coleção são o modo preferido de especificar agregados

em eSQL. Entretanto, agregados baseados em grupo são também suportados para diminuir as dores de transição para usuários SQL. Similarmente, a especificação de distinct (ou all) como modificadores para a entrada de agregado é suportada para comportamento semelhante a SQL, porém o mecanismo preferido deve utilizar o operador distinct() em vez disso.

- 5 A diferença em programas de resolução de nome entre agregados baseados em grupo e baseados em coleção pode exigir que uma implementação faça tentativas de avaliação sob os dois escopos. A estratégia é favorecer a interpretação como um agregado baseado em coleção primeiramente e um agregado baseado em grupo em segundo lugar.

4.14 Expressões não suportadas

10 4.14.1 Predicados quantificados

SQL permite construções da forma:

```
sal > all (select salary from employees)
sal > any (select salary from employees)
```

eSQL não suporta tais construções. As expressões acima podem ser expressas em eSQL como:

not exists (employees.?(sal > it.salary))

exists (employees.? (sal > it.salary))

ou:

not exists (select 0 from employees as e where sal > e.salary)

exists (select 0 from employees as e where sal > e.salary)

eSQL poderia ser estendido com expressões quantificadas.

15 4.14.2*

SQL suporta o uso do "*" como um atalho sintático na cláusula selecionar para indicar que todas as colunas necessitam ser projetadas para fora.

5. Detalhes de linguagem – expressões de consulta

- 20 A expressão consulta é a expressão eSQL de propósito mais geral e é a mais comum para programadores familiarizados com SQL. Combina juntos muitos operadores de consulta diferentes em uma única sintaxe. Muitos desses operadores podem ser especificados individualmente; entretanto nenhum é tão expressivo como quando combinado na expressão de consulta. A menos que especificado de outro modo, a expressão selecionar comporta similarmente à instrução de selecionar TSQL.

- 25 A expressão consultar é composta de uma série de cláusulas que aplicam operações sucessivas a uma coleção de objetos. São baseados nas mesmas cláusulas encontra-

das em instrução de selecionar SQL padrão; selecionar, a partir de, onde, grupo por, ter e pedido por.

- Em sua forma mais simples, a expressão de consulta é especificada pela palavra-chave `select` seguida por uma lista de projeção, a palavra-chave `from`, uma expressão de fonte, a palavra-chave `where` e finalmente uma condição de filtrar:

`Select m from c where e`

Nesse exemplo, `c` é uma coleção de objetos de algum tipo `T`, `m` é um membro de `T` com tipo `M` e `e` é uma expressão booleana que pode se referir a um ou mais membros de `T`. A consulta produz uma nova coleção de objetos com o tipo `M`.

5.1 Cláusula a partir de

- Para a maior parte, a cláusula-a partir de tem a mesma semântica como descrito para TSQL. Entretanto, é muito menos restritivo em sintaxe, permitindo qualquer expressão de linguagem que forneça uma linguagem para ser uma fonte legal. Uma cláusula-a partir de é uma lista separada por vírgula de um ou mais itens da cláusula-a partir de. A cláusula-a partir de pode ser utilizada para especificar uma ou mais fontes para a expressão `select`. A forma mais simples de uma cláusula-a partir de é uma expressão única que identifica uma coleção e um alias:

`From C as c`

- Genericamente, a expressão selecionar opera em cada elemento da coleção de fonte em ordem. Isso é mencionado como iteração. Expressões na lista selecionar ou cláusula-`onde` podem se referir a propriedades do elemento atual utilizando a alias da coleção como uma variável que se refere ao elemento.

5.1.1 Item da cláusula a partir de

Cada item da cláusula a partir de se refere a uma coleção de fonte na consulta. eSQL suporta as seguintes classes de itens de cláusula a partir de.

5.1.1.1 Item de cláusula a partir de simples

- O item de cláusula a partir de mais simples é uma expressão única que identifica uma coleção e um alias:

`C como c`

A especificação `alias` é opcional – uma especificação alternativa do item de cláusula a partir de acima poderia ser:

C

Se nenhum alias for especificado, então eSQL tenta gerar um alias com base na expressão de coleção. Vide 0 descrito posteriormente.

5.1.1.2 Item da cláusula a partir de Junção

Um Join FromClauseItem representa uma junção entre dois itens da cláusula a partir de. eSQL suporta junções cruzadas, junções internas, junções externas esquerda e direita e junções externas completas – tudo em modo similar a T-SQL. Como em T-SQL, os dois itens de cláusula a partir de envolvidos na junção devem ser independentes – não podem ser correlacionados. Um CrossApply/OuterApply pode ser utilizado para esses casos.

Junções cruzadas

10 Uma expressão junção cruzada produz o produto cartesiano das duas coleções:

C as c cross join D as d

Junção interna

Uma junção interna produz um produto cartesiano limitado a duas coleções:

C as c [inner] join D as d on e

Essa expressão processa uma combinação de todo elemento da coleção do lado esquerdo emparelhada contra todo elemento da coleção do lado direito onde a condição on é verdadeira. Uma condição on deve ser sempre especificada.

15

Junção externa esquerda, Junção externa direita

Uma expressão de junção externa produz um produto cartesiano limitado das duas coleções:

C as c left outer join D as d on e

A expressão processa uma combinação de todo elemento da coleção do lado esquerdo contra todo elemento da coleção do lado direito onde a condição on é verdadeira. Se a condição on for falsa, a expressão ainda processa uma única instância do elemento do lado esquerdo emparelhado contra o elemento do lado direito com o valor nulo. Junções externos direitos podem ser expressos em um modo similar.

20

Junção externa completa

25 Uma junção externa completa explícita produz um produto cartesiano limitado das duas coleções:

C as c full outer join D as d on e

A expressão processa uma combinação de todo elemento da coleção do lado esquerdo emparelhado contra todo elemento da coleção do lado direito onde a condição on é verdadeira. Se a condição on for falsa, a expressão ainda processa uma instância do elemento do lado esquerdo emparelhado contra o elemento do lado direito com o valor nulo, e
5 uma instância do elemento do lado direito emparelhado contra o elemento do lado esquerdo com o valor nulo.

Observações:

para preservar compatibilidade com Sql-92, TSql, a palavra-chave outer é opcional. Assim, “junção esquerda”, “junção direita” e “junção completa” são sinônimos para “junção externa esquerda”, “junção externa direita” e “junção externa completa”.
10

a cláusula ON deve ser utilizada para junções interna e externa; é ilegal para junções cruzadas.

5.1.1.3 Item da Cláusula Apply from

eSQL suporta dois tipos de Apply – CrossApply e OuterApply. Um cross apply produz um emparelhamento exclusivo de cada elemento da coleção do lado esquerdo com um elemento da coleção produzida por avaliação da expressão do lado direito. Com um cross apply a expressão do lado direito é uma função do elemento do lado esquerdo:
15

select s, f from C as c cross apply fn(c) as f

O comportamento de cross apply é similar àquele descrito para a lista junção. Se a expressão do lado direito avaliar para uma coleção vazia, a cross apply não produz emparelhamentos para aquela instância do elemento do lado esquerdo.
20

Um outer apply é similar a um cross apply, exceto um emparelhamento é ainda produzido mesmo quando a expressão do lado direito avalia para uma coleção vazia:

Select c, f from C as c outer apply fn(c) as f

Observação: Diferente de TSql, não há necessidade de uma etapa não aninhada explícita.

5.1.2 Coleções múltiplas na cláusula a partir de
25

A cláusula a partir de pode conter mais de uma especificação de coleção, separada por vírgulas. Em tais casos, as coleções são assumidas como sendo unidas juntas. Pense nessas como uma junção cruzada de n-sentido.

5.1.3 Correlação esquerda

Itens na cláusula a partir de podem se referir a itens especificados anteriormente. No exemplo abaixo, C e D são coleções independentes, enquanto c.Names depende de C:

From C as c, D as d, c.Names as e

Isso é logicamente equivalente a:

From (C as c join D as d) cross apply c.Names as e

5 5.1.4 Semântica

Logicamente, as coleções da cláusula a partir de são assumidas como sendo parte de uma junção cruzada de n-sentido – o caso degenerado sendo uma junção cruzada de 1 sentido. Aliases na cláusula a partir de são processados esquerda para a direita, e são adicionados ao escopo atual para referências posteriores. A cláusula a partir de é assumida como produzindo um multiconjunto de linhas – com um campo para cada item na cláusula a partir de, e representando um único elemento a partir daquele item de coleção.

No exemplo acima, a cláusula a partir de produz logicamente um multiconjunto de linhas de tipo Row(c, d, e) onde os campos c, d e e são assumidos como sendo do tipo de elemento de C, D e c.Names. eSQL introduz um alias para cada item de cláusula a partir de simples em escopo. Por exemplo, na consulta abaixo:

From (C as c join D as d) cross apply c.Names as e

Os nomes introduzidos no escopo são c, d e e. Diferente de SQL, a cláusula a partir de somente introduz as aliases no escopo. Quaisquer referências a colunas (propriedades) dessas coleções devem ser qualificadas com a alias.

5.2 Cláusula Where

20 A cláusula-onde tem a mesma semântica como descrito para TSQL. Restringe os objetos produzidos pela expressão consulta por limitar os elementos das coleções de fonte àquelas que passam a condição:

Select c from cs as c where e

A expressão e deve ter o tipo Booleano. A cláusula-onde é aplicada diretamente após a cláusula a partir de, antes de agrupar, ordenar ou projeção. Todos os nomes de elementos definidos na cláusula a partir de são visíveis para a expressão da cláusula-onde.

25

5.3 Cláusula group by

A cláusula agrupar por tem a semântica similar como descrito para TSQL. Pode-se especificar uma ou mais expressões cujos valores são utilizados para agrupar elementos de fonte juntos para fins de avaliar agregados:

```
Select e1, count(c.d1) from c group by e1, e2, ..., en
```

- 5 Cada expressão na cláusula agrupar por deve avaliar para algum tipo que pode ser comparado para igualdade. Esses tipos são genericamente primitivos escalares como números, séries e datas. É possível utilizar tipos complexos como condições agrupar por, desde que os tipos definam uma noção de igualdade. Não se pode agrupar por uma coleção.

- 10 Após uma cláusula de grupo ser especificada, explícita ou implicitamente (em virtude de de uma cláusula ter na consulta), o escopo atual é oculto, e um novo escopo é introduzido.

- A cláusula selecionar, a cláusula ter e a cláusula ordenar por não mais serão capazes de se referir a nomes de elementos específicos na cláusula a partir de. Pode se referir somente às próprias expressões de agrupamento. Para fazer isso, pode-se atribuir novos
15 nomes (alias) a cada expressão de agrupamento. Se nenhum alias for especificado para uma expressão de agrupamento, eSQL tenta gerar uma via as regras de geração alias em O:

```
select g1, g2, ..., gn from c as c1
      group by e1 as g1, e2 as g2, ..., en as gn
```

Expressões na cláusula agrupar por podem não se referir a nomes definidos anteriormente na mesma cláusula agrupar por.

- 20 Além de agrupar nomes pode-se especificar também agregados na cláusula selecionar, cláusula-ter e cláusula ordenar por. Um agregado contém uma expressão que é avaliada para cada elemento do grupo. O operador de agregado reduz os valores de todas essas expressões (normalmente, porém nem sempre, em um único valor). A expressão de agregado pode fazer referência aos nomes de elementos originais visíveis no escopo de
25 origem, ou a qualquer um dos novos nomes introduzidos pela própria cláusula agrupar-por. Embora os agregados apareçam na cláusula selecionar, cláusula ter e cláusula ordenar por, são na realizado avaliados sob o mesmo escopo que as expressões de agrupamento:

```
select name, sum(o.Price * o.Quantity) as total
      from orderLines as o
      group by o.Product as name
```

Essa consulta utiliza a cláusula agrupar-por para produzir um relatório do custo de todos os produtos encomendados, divididos por produção. Fornece o nome 'nome' ao produto como parte da expressão de agrupamento, e então referencia aquele nome na lista selecionar. Também especifica a 'soma' de agregado na lista selecionar que referencia internamente o preço e quantidade da linha do pedido.

Grupos denominados

Os próprios grupos podem ter nomes. Quando um grupo é nomeado introduz um novo nome no escopo que se refere à coleção de objetos que formam uma instância de grupo. O próprio grupo é um agregado – agregado aninhar – que simplesmente agrega todos os itens que casam os critérios para o grupo em um multiconjunto. Logicamente, a cláusula agrupar por executa agrupamento com base nas chaves, e um agregado aninhar implícito – um para cada grupo – que contém a coleção de colunas não chave para aquele grupo.

Referências subseqüentes a agregados – agregados baseados em grupo – são traduzidas em agregados baseados em coleção, onde a coleção em questão é o agregado aninhar produzido pela cláusula agrupar por:

```
select name, sum(o.Price * o.Quantity) as total,
mygroup
from orderLines o
group mygroup by o.Product as name
```

Nesse exemplo, o grupo mygroup é identificado na cláusula agrupar-por e referenciado na lista selecionar. O nome mygroup se refere a uma coleção de instâncias de Order-Line que formam um único grupo. Cada linha da coleção resultante terá três propriedades; nome, total e mygroup.

É desnecessário sempre definir seu próprio nome para um grupo. Mesmo sem especificar um o grupo pode ser sempre acessado utilizando o nome grupo:

```
select name, sum(o.Price * o.Quantity) as total,
group
from orderLines o
group by o.Product as name
```

5.4 Cláusula ter

A cláusula ter é utilizada para especificar uma condição de filtração adicional no resultado de um agrupamento. Se nenhuma cláusula agrupar por for especificada na consulta, um implícito "group by 1" – grupo de conjunto único – é assumido. A cláusula ter trabalha exatamente como a cláusula where, exceto que é aplicada após a operação agrupar-por. Isso significa que a cláusula ter pode somente fazer referência a aliases de agrupamento e

agregados:

```
select name, sum(o.Price * o.Quantity) as total
  from orderLines o
  group by o.Product as name
  having sum(o.Quantity) > 1
```

Esse exemplo é igual ao exemplo agrupar por, exceto que a cláusula `having` limita os grupos somente aqueles tendo mais de uma unidade de produto nos mesmos.

5.5 Cláusula selecionar

- 5 A lista de uma ou mais expressões após a palavra-chave `selecionar` é conhecida como a lista `selecionar` ou mais formalmente como a projeção. A forma de projeção mais geral é uma única expressão. Se você `selecionar` algum membro `m` a partir de uma coleção `c`, então você produzirá uma nova coleção de todos os valores `m` para cada elemento de `c`:

```
Select c.m. from c
```

- 10 Por exemplo, se `clientes` for uma coleção do tipo `cliente` que tenha uma propriedade `Nome` que é do tipo `série`, então `selecionar Nome` a partir de `clientes` fornecerá uma coleção de séries:

```
Select c.Name from customers as c
```

Cláusulas selecionar valor e linha

- 15 eQL suporta duas variantes da cláusula `selecionar`. A primeira variante – `selecionar linha` – é identificada pela palavra-chave `selecionar`, e pode ser utilizada para especificar um ou mais itens que devem ser projetados para fora. Implicitamente, um wrapper de linha é adicionado em torno dos itens, com o resultado de que o resultado da expressão de consulta é sempre um multiconjunto de linhas – com os campos apropriados. Cada expressão em uma `selecionar linha` deve especificar um alias. Se nenhum alias for especificado, eSQL tenta gerar um alias utilizando as regras de aliasing descritas em 0.
- 20 A outra variante da cláusula `selecionar` – `selecionar valor` – é identificada pela palavra-chave `selecionar valor`, e permite somente que um item seja especificado, e não acrescenta um wrapper de linha.

Um `selecionar linha` é sempre expressável em termos de um `selecionar valor`. Por exemplo:

→ `select 1 as a, "abc" as b, a+34 as c`
`select value row(1 as a, "abc" as b, a+34 as c)`

Modificadores All e Distinct

As duas variantes de selecionar permitem a especificação de um modificador all ou distinct. Se o modificador distinct for especificado, duplicatas são eliminadas a partir da coleção produzida pela expressão consulta (até e incluindo a cláusula selecionar). Se o modificador all for especificado, então nenhuma eliminação de duplicata é executada.

Esses modificadores são somente preservados para serem compatíveis com SQL. eSQL sugere o uso do operador distinct em vez disso, porém suporta essas variantes, e traduz de forma transparente a expressão:

→ `select distinct c.a1, c.a2 from T as a`
`distinct(select c.a1, c.a2 from T as a)`

Semântica

- 10 A cláusula selecionar é avaliada após avaliação das cláusulas a partir de, cláusula, agrupar por e ter. A cláusula selecionar pode se referir somente a itens atualmente em escopo (via a cláusula a partir de, ou a partir de escopos externos). Se uma cláusula agrupar por foi especificada, então a cláusula selecionar é somente permitida para referenciar as aliases para as chaves agrupar por. Com referência aos itens da cláusula a partir de somente é permitido como parte de funções de agregado.

5.5.1 Desvios a partir de SQL

Sem suporte para *

- 20 ESQL não suporta o uso de * ou o * qualificado para indicar que a cláusula- a partir de inteira (ou todas as colunas a partir de uma tabela) deve ser projetada para fora. Mesmo em SQL, consultas bem elaboradas evitam essas construções, uma vez que podem ter efeitos colaterais inesperados quando esquemas se desenvolvem. eSQL permite que consultas projetem para fora de registros inteiros em vez disso por referência à coleção de aliases da cláusula a partir de. A seguinte consulta SQL:

Select * from T1, T2

É muito melhor expressa em eSQL pela construção a seguir:

Select t1, t2 from T1 as t1, T2 as t2

5.6 Cláusula ordenar por

A cláusula ordenar por pode especificar uma ou mais expressões que determinam a ordenação dos elementos no resultado. A cláusula ordenar por é logicamente aplicada ao resultado da cláusula selecionar. Pode referenciar itens na lista selecionar através de seus
5 aliases. Além disso, pode referenciar também outras variáveis que estão atualmente no escopo. Entretanto, se a select_clause foi especificada com um modificador distinct, então a cláusula ordenar-por pode referenciar somente aliases a partir da cláusula selecionar:

Select c as c1 from cs as c order by c1.e1, c.e2, ..., en

Cada expressão na cláusula ordenar por deve avaliar para algum tipo que possa ser comparado para desigualdade ordenada (menor do que ou maior do que, etc.). Esses
10 tipos são genericamente primitivos escalares como números, séries e datas.

Cada expressão na cláusula ordenar-por pode especificar opcionalmente critérios de separação. ASC (ou DSC) pode ser especificado para indicar que um tipo ascendente (ou um tipo descendente) é necessário na expressão específica. Adicionalmente, para expressões do tipo série, uma cláusula COLLATE pode ser especificada para indicar a conferência de série a ser utilizada para ordenação.
15

Especificação posicional, por exemplo, ordenar por 3, pode ser utilizada para indicar o item correspondente (terceiro) na lista selecionar não é suportada.

6 Detalhes de linguagem – comando

6.1 Comando

Um comando eSQL é o termo utilizado para a solicitação de consulta inteira. Um comando pode ser uma expressão de consulta como uma expressão selecionar ou uma instrução utilizada para inserir, deletar ou atualizar objetos. Um comando é logicamente composto de três partes.
20

- . um Prólogo opcional
- 25 . uma cláusula WITH opcional
- . uma consulta ou uma instrução DML

Por exemplo, imagine que haja um tipo denominado Cliente e uma coleção de instâncias de Cliente denominados clientes. O que se segue são todos comandos válidos:

customers.?(Name = 'Bob')

select c.Name, c.Phone from customers c

6.1.1 Prólogos de comando

O Prólogo de comando (opcional) pode especificar um conjunto de espaços de nome para uso. Esses serão descritos posteriormente na seção em espaços de nome.

6.1.2 Cláusula WITH

- 5 eSQL suporta uma cláusula WITH como um prefixo para qualquer consulta (ou instrução DML). A sintaxe da cláusula WITH é como a seguir;

```
withClause ::= WITH <common-table-expr> [, <common-
table-expr>]*

<common-table-expr> ::= <simpleIdentifier> AS
<parenthesizedExpr>
```

Observações

. definições de <common-table-expr> podem se referir a expressões de tabela comum definidas mais cedo na cláusula WITH. Por exemplo, a seguinte definição é legal:

```
WITH tab1 as (...), tab2 as (select t from tab1 as t)
```

10 6.2 Instrução de consulta

Uma Instrução de consulta é simplesmente uma expressão.

6.3 Instrução de DML

Instruções de DML são descritas na próxima seção.

7 DML

- 15 eSQL suporta inserir, atualizar e deletar instruções DML.

7.1.1 Inserir

o comando INSERT em eSQL é muito similar à instrução INSERT em SQL padrão:

```
[Withclause] Insert [into] <container> [<with-parent>] FROM <expr>
[Withclause] Insert [into] <container> [<with-parent>] <query_expr>
[Withclause] Insert [into] <container> [<with-parent>] VALUES <query_expr>
```

Semântica

- 20 . <container> representa qualquer recipiente de dados. EntitySets são o recipiente mais lógico.

. <expr>/<query_expr> é um multiconjunto (nos dois primeiros casos), que produz um conjunto de valores para inserir no recipiente. No terceiro caso (cláusula VALUES), <expr> é do tipo do recipiente de elemento.

. A WithClause é definida em seções subseqüentes.

. a cláusula With-parent é descrita em seções subseqüentes.

Exemplos

Insert into NewCustomers select o from oldCustomers

as o

insert into NewCustomers from Multiset (Customer (...),

Customer (...), ...)

--The follwing statements are all equivalent

inset into NewCustomers from Multiset(Customer(...))

inset into NewCustomers select c from

Multiset(Customer(...)) as c

Insert into NewCustomers from {Customer(...)}

inset into NewCustomers values Customer(...)

inset into NewCustomers values (Customer(...))

Observações:

- 5 . a instrução INSERT..VALUES de eSQL é levemente diferente de T-SQL. Por um motivo, os parênteses em torno do valor são opcionais, e não são interpretados como um construtor de linha quando presentes.

. Ao contrário de T-SQL, eSQL não permite que uma lista de coluna seja especificada para uma instrução Insert.

7.1.2 Deletar

- 10 O comando DELETE em eSQL é similar à instrução DELETE em SQL padrão.

[WithClause] Delete [from] <container> [as <alias>]

[<with-parent-clause>] [where <expr>]

Semântica

. <container> representa qualquer recipiente de dados. EntitySets são o recipiente mais lógico.

. <expr> é um predicado (ie) do tipo booleano.

- 15 . <with-parent-clause> é definido posteriormente.

Exemplos:

Delete from Newcustomers as n where n.id > 10

Delete from NewCustomers

Observações

. Diferente de T-SQL, a instrução DELETE em eSQL não permite cláusulas a partir de múltiplas. Utilize uma subconsulta em vez disso.

7.1.3 Atualizar

- 5 A instrução atualizar em eSQL é similar de forma conceptual a instrução UPDATE em SQL padrão:

```
Update_statement:: = [WithClause]
                    UPDATE <container> [as <alias>]
                    [<with-parent-clause>]
                    {<set_clauses>} [<apply-
clauses>]
                    [where <expr>]
<set_clauses> :: = SET <set_clause> [, <set_clause>]*
<set_clauses> :: = <field_expr> EQUAL <expr>
<field-expr> :: = <expr>
```

Semântica

. <container> representa qualquer recipiente de dados. EntitySets são o recipiente mais lógico.

- 10 . <expr> é um predicado (ie) do tipo booleano.
. <field-expr> é uma expressão de valor-1 (i.e.) pode ser uma referência de coluna, uma referência a uma propriedade de uma coluna, ou uma expressão TREAT que envolve uma tal propriedade.
. <with-parent-clause> é definida posteriormente.

- 15 Exemplos:

```
UPDATE NewCustomers as n
SET n.name.firistName=UPPER (n.name.firstName),
TREAT (n.address as
InternationalAddress).contrycode='US'
WHERE n.name like 'ABC%'
```

Observações

. Diferente de T-SQL, a instrução UPDATE em eSQL não permite múltiplas classes a partir de. Utilize uma subconsulta em vez disso.

. Um atributo pode ser modificado no máximo uma vez na instrução UPDATE.

7.2. DML para Relações de EDM

O EDM introduz a noção de RelationshipSets como coleções de instâncias de relação. Relações tanto de Associação como de Composição são modeladas através desses RelationshipSets.

5 7.2.1 Associações

O padrão de uso geral é que você primeiramente crie instâncias das entidades, e então crie relações entre as entidades pela inserção no RelationshipSet. O exemplo a seguir demonstra DMLs em uma tal relationshipset. Assuma dois tipos de entidade Pedido e Cliente (com EntitySets – Pedidos e Clientes). Assuma ainda uma relação OrderCustomer entre
10 essas duas entidades, e OrderCustomerSet como a relationshipset:

```
-- Insert a relationship between an Order and a
Customer
Insert into OrderCustomerSet
Select OrderCustomer(ref(o), ref(c))
From Customers as c, Orders as o
Where o.Id = 123 and c.Name = 'ABC'

-- Delete a relationship between an Order and a
Customer
Delete from OrderCustomerSet as oc
Where oc.Order.Id = 123
```

7.2.2 Composições

RelationshipSets correspondendo a composições são também atualizáveis. Deve-se primeiramente criar a instância de entidade de pai. Ao contrário de associações, entretanto, não há etapa separada de criar a instância filho. A inserção na RelationshipSet de Com-
15 posição cria automaticamente a instância de filho. Os exemplos desse incluem:

```
-- Inserting new lines into an existing order
Insert into orderLineset,.Line.
with parent (element(select ref(o) from Orders
as o
where o.Id = 20))
select Line(...) from ...

-- modifying an existing line item
Update OrderLineSet.Line as . 1
with parent (element(select ref(o) from Orders
as o
where o.Id = 20))
Set 1.description = ...
```

Where ...

-- deleting an existing line item

Delete OrderLineSet.Line as 1.

with parent (element(select ref(o) from Orders

as o

where o.Id = 20))

where 1.description =

Observação: a cláusula especial “com pai como” para modificar a composição.

8 eSQL Misc

8.1 Consultas recursivas

eSQL permite suporte para consultas recursivas permitindo que a definição de tabela (parenthesizedExpr) referencie a si própria. Por exemplo:

WITH tab1. as (multiset(1, 2, 3, 4, 5)),

tab2 as

multiset(1)

union all

select t2

from tab2 as t2, tab1 as t1

where t2 = t1

)

select t from tab1 as t

eSQL não garante sobre terminação da recursão.

8.2 Suporte para relações EDM

O EDM introduz a noção de RelationshipSets – que são equivalentes lógicos de tabelas de link em SQL. RelationshipSets podem ser consultados exatamente como outras coleções. Adicionalmente, eSQL suporta ajudantes de navegação para navegação mais fácil sobre relações. A forma geral de um ajudante de navegação é a seguinte:

Select o.Id, o-> OrderCustomer.Customer

From Orders as o

Onde OrderCustomer é o nome da relação, e Cliente é o nome do cliente final da relação.

8.3 Comentários

eSQL suporta comentários do estilo. Comentários eSQL podem ser das seguintes

formas:

- this is a comment
- "this is also a comment"

Comentários podem ser utilizados em qualquer local onde um caractere de espaço branco é esperado.

caracteres de comentário do estilo "--" (hífen duplo) podem ser utilizados na mesma linha que o código a ser executado, ou em uma linha sozinhos. Tudo a partir dos hífens duplos até o final da linha faz parte do comentário.

/*...*/ (pares de caractere de asterisco-traço para frente). Esses caracteres de comentário podem ser utilizados na mesma linha como código a ser executado, em linhas sozinhas, ou mesmo no código executável. Tudo a partir do par de comentários aberto (/*) até o par de comentários próximo (*/) é considerado parte do comentário. Esses comentários podem cobrir múltiplas linhas.

9 Resolução de nome eSQL e outras regras

9.1 Identificadores

Identificadores em eSQL são utilizados para representar aliases de expressão, referências variáveis, propriedades de objetos, funções, etc. Identificadores em eSQL podem ser de dois tipos.

9.1.1 Identificador simples

Um identificador simples é simplesmente uma seqüência de caracteres alfanuméricos (e o underRank)

O primeiro caractere do identificador deve ser um alfabeto, a-z ou A-Z.

9.1.2 Identificadores citados

Um identificador citado é qualquer seqüência de caracteres circundada por colchetes, ou aspas duplas. A citação default é um colchete. Aspas duplas podem ser utilizadas como o caractere de citação somente quando o analisador para eSQL permitir isso – as opções de analisador exato para controlar isso são deixados a critério do analisador. Observe também que quando aspas duplas podem ser utilizadas como o caractere de citação, não podem ser utilizados para literais de série – literais de série necessitariam, então sempre utilizar aspas únicas.

Identificadores citados permitem que uma pessoa especifique identificadores com caracteres que não seriam normalmente legais em identificadores. Todos os caracteres entre os colchetes (ou as aspas) são considerados parte do identificador, incluindo todo espaço branco.

Um identificador citado pode não incluir os seguintes caracteres:

- . linha nova
- . retorno do carrinho
- . tabs
- . backspace

5 . [(somente no caso de aspas de estilo [])

Poder-se-ia inventar nomes de propriedade com caracteres de outro modo normalmente ilegais:

```
select c.ContactName as [Contact Name] from customers as c
select c.ContactName as "Contact Name" from customers as c
```

10 Pode-se utilizar um identificador citado para especificar um identificador que normalmente seria considerado uma palavra reservada da linguagem. Por exemplo, se o tipo E-mail tiver uma propriedade denominada 'a partir de', poder-se-ia remover ambiguidade da mesma da palavra reservada 'a partir de' utilizando os colchetes:

```
Select e.[from] from emails as e
```

Pode-se utilizar um identificador citado no lado direito de um operador de ponto:

```
Select t from ts as t where t.[a property] == 2
```

Para utilizar o mesmo char como o caractere de aspa de fechamento em um identificador, utiliza-se o mesmo duas vezes. Por exemplo:

```
Select t from ts as t where t.[a bc]] == 2
```

15 O identificador abc] fica de fora apropriadamente.

9.2 Sensibilidade do caso

Todas as palavras-chave em eSQL são insensíveis ao caso (como em SQL). Identificadores podem ser sensíveis ao caso ou insensíveis ao caso dependendo das opções de analisador especificadas. Palavras-chave em eSQL são sempre insensíveis ao caso.

20 9.3 Regras de aliasing

eSQL recomenda que aliases sejam especificadas em consultas sempre que for necessário. Aliases são necessários para as seguintes construções:

- . campos de um construtor de linha
- . itens na cláusula a partir de, de uma expressão de consulta

- . itens na cláusula selecionar de uma expressão de consulta
- . itens na cláusula agrupar por de uma expressão de consulta
- . argumentos para uma expressão junção

Aliases válidos

- 5 Qualquer identificador simples, ou identificador citado é um alias válido.

Geração de alias

Se nenhum alias for especificado, eSQL tenta gerar um com base em algumas regras simples.

- 10 . Se a expressão (para a qual o alias é não especificado) for um identificador simples ou citado, aquele identificador é utilizado como o alias. Por exemplo:

`row (a, [b/]) -> row (a as a, [b] as [b])`

. se a expressão for uma expressão mais complexa, porém o último componente daquela expressão for um identificador simples, então aquele identificador é utilizado como o alias. Por exemplo:

`row (a.a1, b.[b1]) -> row(a.a1 as a1, b.[b1] as [b1])`

- . de outro modo uma exceção de compilar tempo é originada.

15 9.4 Regras de abrangência

Regras de abrangência definem quando variáveis específicas são visíveis na linguagem de consulta. Algumas expressões ou instruções introduzem novos nomes. As regras de abrangência determinam onde aqueles nomes podem ser utilizados, e quando ou onde uma nova declaração com o mesmo nome que outro pode ocultar seu predecessor.

- 20 Quando nomes são definidos, diz-se que são definidos em um escopo. Um escopo cobre uma região da consulta. Todas as expressões ou referências de nome em um certo escopo podem ver nomes definidos naquele escopo. Antes do início de um escopo e após o mesmo terminar, nomes definidos no escopo não podem ser referenciados.

- 25 Escopos podem ser aninhados. Partes da linguagem introduzem novos escopos que cobrem regiões que podem conter outras expressões de linguagem que também introduzem escopos. Quando escopos são aninhados, referências podem ser feitas a nomes definidos no escopo mais interno em que a referência está bem como quaisquer nomes definidos em quaisquer escopos externos. Quaisquer dois escopos definidos no mesmo escopo são considerados escopos filhos. referências podem não ser feitas a nomes definidos
- 30 nos escopos filhos.

Se um nome for declarado em um escopo interno que casa com um nome declara-

do em um escopo externo, referências naquele escopo ou nos escopos declarados naquele escopo se referem somente ao nome recentemente declarado. O nome no escopo externo é oculto. Mesmo no mesmo escopo, os nomes podem não ser referenciados antes de serem definidos.

5 Nomes globais podem existir como parte do ambiente de execução. Isso pode incluir nomes de coleções ou variáveis de ambiente persistentes. Esse escopo que contém os mesmos é o escopo mais externo. Parâmetros não estão em um escopo. Uma vez que as referências a parâmetros incluem sintaxe especial, os nomes de parâmetros não colidem com outros nomes na consulta.

10 9.4.1 Expressões de consulta

A expressão de consulta introduz um novo escopo. Nomes definidos na cláusula a partir de são introduzidos no escopo a partir em ordem de aparição, da esquerda para a direita. Na lista junção, expressões podem se referir a nomes definidos anteriormente na lista. Propriedades públicas (campos, etc.) de elementos identificados na cláusula a partir de não são adicionadas ao escopo a partir de – devem ser sempre referenciados através do nome qualificado por alias. Normalmente, todas as partes da expressão selecionar são consideradas compreendidas no escopo a partir de.

A cláusula agrupar por também introduz um novo escopo filho. Cada grupo pode ter um nome de grupo que se refere à coleção de elementos no grupo. Cada expressão de agrupamento também introduzirá um novo nome no escopo de grupo. Além disso, o agregado aninhar (ou o grupo denominado) também é adicionado ao escopo. As próprias expressões de agrupamento estão compreendidas no escopo a partir de. Entretanto, quando uma cláusula agrupar por é utilizada, a lista selecionar (projeção), cláusula ter e cláusula ordenar por são consideradas como estando compreendidas no escopo de grupo e não no escopo a partir de. Agregados têm tratamento especial, e esses são descritos abaixo.

A lista selecionar pode introduzir novos nomes no escopo, em ordem. Expressões de projeção à direita podem se referir a nomes projetados à esquerda. A cláusula ordenar por pode se referir a nomes (aliases) especificados na lista selecionar. A ordem de avaliação de cláusulas na expressão selecionar determina a ordem em que os nomes são introduzidos no escopo. A cláusula a partir de é avaliada primeiramente, seguido pela cláusula onde, cláusula de agrupamento, cláusula ter, cláusula selecionar e finalmente a cláusula ordenar por.

Manipulação de agregado

eSQL suporta duas formas de agregados – agregados à base de coleção e agregados baseados em grupo. Agregados baseados em coleção são a construção preferida em eSQL, enquanto os agregados baseados em grupo são simplesmente suportados para compatibilidade de SQL.

Ao resolver um agregado, eSQL primeiramente tenta tratar o mesmo como um agregado baseado em coleção. Se isso falhar, então eSQL transforma a entrada de agregado para ser uma referência ao agregado aninhar, e tenta resolver essa nova expressão. Por exemplo:

`Avg(t.c) -> avg(group..(t.c))`

5 9.4.2 Operações de projetar/filtrar coleção

Para os operadores de projetar/filtrar coleção (... .., .? e ..?) um novo escopo é criado para a projeção/predicado, e o nome it é implicitamente adicionado ao escopo – para representar cada elemento da coleção. Além disso, para simplicidade, os membros públicos do elemento de coleção são também adicionados ao escopo – novamente apenas pela duração da avaliação da projeção/predicado. Por exemplo:

`Departments..Name`

`Departments..(it.Name)`

Produz um multiconjunto de nomes de departamentos – onde cada Departamento é assumido como tendo uma propriedade de Nome.

9.5 espaço de nome

15 eSQL introduz espaços de nome para tratar a questão de conflitos de nome para identificadores globais como nomes de tipo, conjuntos de entidade, funções etc. O suporte de espaço de nome em eSQL é muito similar ao modelo CLR. ESQl provê a cláusula usar que pode ser utilizada no Prólogo Comando. Duas formas da cláusula usar são fornecidas – espaços de nome qualificados (onde um alias mais curto é fornecido para o espaço de nome), e espaços de nome não qualificados:

`using System.Data.TSql;`

`using tsq1=System.Data.TSql;`

20 Regras de resolução de nome

Se um identificador não puder ser resolvido nos escopos locais, eSQL tenta então localizar o nome nos escopos globais (ie) os espaços de nome. eSQL primeiramente tenta casar o identificador (prefixo) com um dos espaços de nome qualificados. Se houver um casamento, então eSQL tenta resolver o resto do identificador no espaço de nome especificado – se nenhum casamento for encontrado, uma exceção é originada:

25

```

using tsq1=System.Data.TSql;
using winfs.=System.Storage.WinFs,
select tsq1.substr (p:Name) from Person as p
select i.ItemId from Items as i where i is of winfs.Contact

```

eSQL então tenta sondar todos os espaços de nome não qualificados (especificados no prólogo) para o identificador. Se o identificador puder ser localizado exatamente em um espaço de nome, então aquele local é retornado. Se mais de um espaço de nome tiver um casamento para aquele identificador, uma exceção é originada. Se nenhum espaço de nome puder ser identificado para o identificador, eSQL então passa o nome sobre o escopo para fora seguinte (o objeto de Conexão/comando):

```

using System.Data.TSql;
select subst r(p.Name) from Person as p

```

Diferenças a partir do modelo CLR

Vale a pena citar uma diferença a partir do modelo CLR. No CLR, você pode utilizar espaços de nome parcialmente qualificados – eSQL não permite isso. Por exemplo, em C#, o que se segue é legal, enquanto o equivalente não é legal em eSQL:

```

using System.Data.TSql;
void main () {
int x = y.z.foo(); -- reference to method Foo of class z in
-- namespace
System.Data.TSql.Y
}

```

Uso de ADO.NET

Consultas/instruções de DML são expressos via Comandos ADO.NET. Os comandos podem ser construídos sobre objetos de Conexão. Espaços de nome podem ser especificados como parte dos objetos de Conexão e comando também. Se eSQL não puder resolver um identificador na própria consulta, então os espaços de nome externos são sondados (via regras similares). A consulta, objetos de Comando e Conexão formam anéis de espaços de nome – onde cada anel é sondado primeiramente.

9.6 Literais de nulo e inferência de tipo

Como descrito anteriormente, literais nulos são compatíveis com qualquer tipo no sistema de tipo eSQL. Entretanto, para o tipo de um literal nulo ser deduzido corretamente, eSQL impõe certas limitações sobre onde um literal nulo pode ser utilizado. As seguintes

regras se aplicam

Nulos digitados

Nulos digitados (i.e.) “cast(null as Int16)” podem ser utilizados em qualquer lugar; não há necessidade adicional de inferência de tipo, uma vez que o tipo é conhecido nesse momento.

Literais nulos de flutuação livre

Literais nulos de flutuação livre podem ser utilizados nos seguintes contextos:

- . como um argumento para uma expressão cast/treat – esse é o mecanismo recomendado para produzir uma expressão nulo digitado.

- . um argumento para um método ou uma função (regras de sobrecarga padrão se aplicam, e se uma sobrecarga não puder ser pega, então um erro de compilação é originado)

- . como um dos argumentos para uma expressão aritmética como +, -, *, /. O(s) outro(s) argumento(s) não podem ser literais nulos; de outro modo inferência de tipo não é possível.

- . Como um dos argumentos para uma expressão lógica (e/ou/não) – todos os argumentos são conhecidos como sendo do tipo booleano.

- . como o argumento para uma expressão “é nula” ou “não é nula” – as expressões “nulo é nulo” e “nulo não é nulo” são legais (porém bobas)/

- . como um ou mais dos argumentos uma expressão como – espera-se que todos os argumentos sejam séries.

- . como um ou mais dos argumentos para um construtor do tipo denominado.

- . como um ou mais dos argumentos para um construtor de multiconjunto. Entretanto, pelo menos um argumento para o construtor de multiconjunto deve ser uma expressão diferente de um literal nulo.

- . como uma ou mais das expressões then/else em uma expressão de caso. Pelo menos uma das expressões then/else na expressão de caso deve ser uma expressão diferente de um literal nulo.

Literais nulos de flutuação livre não podem ser utilizados em outros cenários. Alguns desses cenários (não exaustivos; incluídos somente para ilustração) incluem

- . argumentos para um construtor de linha.

Observações

As expressões é nulo e não é nulo são consideradas como sendo especiais.

10 Sistema do tipo eSQL

eSQL opera em e produz instâncias do Modelo de dados de entidade (EDM). Esse documento não entrará em detalhes sobre o EDM; em vez disso, alguns dos itens de interesse especial são chamados.

10.1 Tipos de linha

Linhas (a.k.a. tuples) são estruturais por qualidade de membro. A estrutura de uma linha depende da seqüência de membros denominados e digitados que compreende. Uma linha é um EDM InlineType – não tem identidade e não pode ser herdada a partir do mesmo.

5 Instâncias do mesmo tipo de linha são equivalentes se os membros são respectivamente equivalentes. Linhas não tem comportamento além de sua equivalência estrutural.

Linhas não tem equivalente no runtime de linguagem comum. As consultas podem resultar em estruturas contendo linhas ou coleções de linhas.

10 A ligação de API entre as consultas de eSQL e a linguagem hospedeira definirá como as linhas são realizadas no resultado da consulta. Em linhas ADO.NET, linhas serão alisadas como DataRecords.

10.2 Coleções

15 Tipos de coleções representam zero ou mais instâncias de outros objetos. O EDM suporta vários tipos de coleções – incluindo multiconjuntos, listas etc. Esse documento somente lida com Multiconjuntos.

10.3. Capacidade de anulação

No EDM, capacidade de anulação é limitação sobre uma propriedade, e não uma faceta de um próprio tipo. Todo tipo no EDM é anulável. Literais Nulos em eSQL são considerados como sendo compatíveis com tipo alternado no EDM.

20 10.3.1 Semântica nula

Nulos em eSQL se comportam muito similarmente com Nulos em TSq1. Especificamente, a maioria das expressões que operam em um nulo retorna nulos. Isso inclui:

- . expressões aritméticas
- . expressões lógicas
- 25 . expressões de tipo

As seguintes expressões podem não retornar nulos quando um ou mais dos seus argumentos são nulos:

- . construtores (tipo, linha, multiconjunto)
- . funções/métodos
- 30 . operadores é nulo e não é nulo
- . outros

10.4 Promoção de tipo implícito

35 eSQL define uma variedade de operadores binários e unários comuns, como adição, multiplicação e negação sobre tipos de dados primitivos como número inteiro, flutuante and string. Normalmente, operadores binários como adição operam sobre duas instâncias do mesmo tipo de dados. Entretanto, às vezes é legal solicitar que um operador trabalhe sobre duas instâncias de tipo diferente; por exemplo, adicionar e número inteiro e um flutu-

ante.

Para que isso funcione, um dos dois operandos deve ser promovido ao tipo do outro. Em geral, esse é o tipo com a maior precisão, de modo que não há perda de dados durante a conversão. Em algumas circunstâncias, os dois operandos são promovidos a um terceiro tipo para assegurar que não haja perda de dados.

A tabela a seguir lista as promoções de tipo implícito incorporadas na linguagem. A primeira coluna lista o tipo base e a segunda coluna lista as promoções mais imediatas que podem ser feitas ao tipo. Por exemplo, um Int16 pode promover para um Int32. Ainda assim, pode promover também para um Int64, uma vez que Int32 pode promover para Int64.

Tipo	Promoções
Booleano	Nenhum
Sbyte	Int16, único, decimal
Int16	Int32, único, decimal
Int32	Int64, dobro, decimal
Int64	Decimal
Byte	Int16, UInt16, único, decimal
UInt16	Int32, UInt32, único, decimal
UInt32	Int64, UInt64, dobro, decimal
UInt64	Decimal
Único	Dobro, decimal
Dobro	Decimal
Decimal	Nenhum
Char	Série
Série	Nenhum
DateTime	Nenhum
Guid	Nenhum

Tipos que não são listados ou são listados como tendo uma promoção de 'nenhum' não podem ser implicitamente promovidos.

Se uma pessoa desejar saber a promoção de tipo correto ao tentar adicionar um Int16 a um flutuante, poderia olhar a tabela para ver que um Int16 realmente promoverá um flutuante, ainda assim um flutuante não promoverá para um Int16. Portanto, o Int16 será promovido a um flutuante, de modo que dois flutuantes podem ser adicionados juntos.

Um exemplo ligeiramente mais complicado teria um Int32 sendo adicionado a um flutuante. Observe que um Int32 não promoverá a um flutuante e um flutuante não promoverá a um Int32. Entretanto, tanto Int32 como flutuante promoverão para um dobro. Portanto,

os dois operandos são promovidos para dobro, e dois dobros são adicionados juntos.

10.5 Conversões do tipo legal

O que segue é uma tabela de conversões legais entre tipos de dados primitivos.

5

N.T. INSERIR A TABELA NA PÁGINA 60

Onde

	Booleano	Sbyte	Int16	Int32	Int64	Byte	UInt16	UInt32	UInt64	Único	Dobro	Decimal	DateTime	Char
Booleano	●													
Sbyte	●	●	●	●	●	●	●	●	●	●	●			
Int16	●	●	●	●	●	●	●	●	●	●	●			
Int32	●	●	●	●	●	●	●	●	●	●	●			
Int64	●	●	●	●	●	●	●	●	●	●	●			
Byte	●	●	●	●	●	●	●	●	●	●	●			
UInt16	●	●	●	●	●	●	●	●	●	●	●			
UInt32	●	●	●	●	●	●	●	●	●	●	●			
UInt64	●	●	●	●	●	●	●	●	●	●	●			
Único	●	●	●	●	●	●	●	●	●	●	●			
Dobro	●	●	●	●	●	●	●	●	●	●	●			
Decimal	●	●	●	●	●	●	●	●	●	●	●			
DateTime												●		
Char													●	●
Série	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Guid													●	

11. Ligações de linguagem hospedeira

A ligação de linguagem hospedeira é a API pela qual a pessoa interage com eSQL. ESQl foi projetado como uma linguagem de consulta que é acessada via uma API a partir de uma linguagem de programação hospedeira bem do mesmo modo que TSQL atual é acessado por programadores utilizando API's de acesso de dados da Microsoft. De forma conceptual, os dados que estão sendo consultados residem em algum lugar fora do domínio da linguagem hospedeira, em um banco de dados ou outra armazenagem persistente, ou uma estrutura de dados que é opaca para operadores de linguagem normal.

15

O mecanismo de acesso de dados preferido para eSQL é ADO.NET. O objeto Comando em ADO.NET tem uma propriedade CommandText (que pode ser definida para uma série de consulta eSQL). O método Executar no objeto Comando retorna uma DataReader

(que suporta uma interface IEnumerator). A DataReader é logicamente uma coleção de DataRecords, onde uma DataRecord representa um único elemento do resultado.

12 Gramática

12.1 Comando

command	:: =	[Prolog][withClause](query dml)
prolog	:: =	{NamespaceDecl;}*
NamespaceDecl	:: =	using (AliasedNamespaceDecl\
		UnaliasedNamespaceDecl)
AliasedNamespaceDecl	:: =	simpleIdentifier = Identifier
UnaliasedNamespaceDecl	:: =	Identifier
Withclause	::	withwithclauseItemList
withClauseItemList	:: =	withClauseItem
		withClauseItemList, withClauseItem
withClauseItem	:: =	simpleIdentifier as parenthesizedExpr

5

1.2.DML

dml	:: =	insert update delete
insert	:: =	insert [into] expr [with-parent-clause] from expr
		insert [into] expr [with-parent-clause] queryExpr
		inset [into] expr [with-parent-clause] values expr
delete	::=	delete [from] aliasedExpr [with-parent-clause]
[where expr]		
update	::	update aliasedExpr [with-parent-clause] set-
clause [where expr].		
with-parent-clause	::=	with parent paparenthesizedExpr
set-clause	:: =	set set-clause-item-list
set-clause-item-list	:: =	set-clause Item
		set-clause-item-list, set-clause-item
set-clause-item	:: =	simple-set-clause-item
simple-set-clause-item	:: =	expr=expr

12.3 Expressões de consulta

query	::=	generalExpr
generalExpr .	::=	expr

		queryExpr
expr	::=	literalExpr
		parameterExpr
		variableRefExpr
		builtinExpr
		dotExpr
		constructorExpr
		methodExpr
		aggregateExpr
		collectionNavigationExpr
		relationshipNavigationExpr
		setOpExpr
		parenthesizedExpr
exprList	::=	expr [,expr]*
aliasedExpr	::=	(generalExpr)
literalExpr	::=	numericLiteral
		stringLiteral
		booleanLiteral
		nullLiteral
parameterExpr	::=	@simpleIdentifier
variableRefExpr	::=	identifier
dotExpr	::=	expr.identifier
methodExpr	::=	[expr.]identifier([all distinct] [exprList])
constructorExpr	::=	rowConstructorExpr
		objectConstructorExpr
		multisetConstructorExpr
rowConstructorExpr	::=	row(aliasedExpr[,aliasedExpr]*)
multisetConstructorExpr	::=	[identifier.]* identifier([exprList])
aggregateExpr	::=	[identifier.]*identifier([all distinct]exprList)
builtinExpr	::=	arithmeticExpr
		comparisonExpr
		logicalExpr
		likeExpr
		isNullExpr
		betweenExpr

		caseExpr
		typeOpExpr
		refExpr
		derefExpr
		keyExpr
		createRefExpr
arithmeticExpr	::=	expr+expr
		expr-expr
		expr*expr
		expr%expr
		+expr
		-expr
comparisonExpr	::=	expr<expr
		expr>expr
		expr=expr
		expr<=expr
		expr>=expr
		expr<>expr
logicalExpr	::=	expr and expr
		expr && expr
		expr or expr
		expr expr
		not expr
		! expr
isNullExpr	::=	expr is [npt] null
likeExpr	::=	expr [not] like expr escape expr
betweenExpr	::=	expr [not] between expr and expr
derefExpr	::=	deref(expr)
refExpr	::=	ref(identifier)
createRefExpr	::=	createref (identifier, expr)
keyExpr	::	key (expr)
caseExpr	::=	case whenlist[elseExpr] end
whenList	::=	whenExpr {whenExpr}
WhenExpr	::=	when expr then expr
ElseExpr	::=	else expr
typeOpExpr	::=	treat (expr as expr)
		cast (expr as expr)

		expr is [not] of ([only] expr)
		oftype (expr, expr)
relationshipNavigationExpr ::=		expr->identifier.identifier
collectionNavigationExpr ::=		expr..expr
		expr.?expr
		expr...expr
		expr..?expr
setOpExpr ::=		expr union expr
		expr union all expr
		expr intersect expr
		expr except expr
		expr overlaps expr
		expr in expr
		exists (generalExpr)
		element (general Expr)
		flatten (generalExpr)
queryExpr ::=		selectClause
		from clause
		[whereClause]
		[groupByClause]
		[havingClause]
		[orderByClause]
SelectClause ::=		rowSelectClause
		valueSelectClause
rowSelectClause ::=		select[all distinct]aliasedExpr
		{,aliasedExpr}
valueSelectClause ::=		Select value [all distinct] expr
fromClause ::=		from fromClauseitem {,fromClauseitem}
fromClauseItem ::=		simpleFromClauseItem
		joinFromClauseItem
		(joinFromClauseItem)
		applyFromClauseItem
		(applyFromClauseItem)
simpleFromClauseItem ::=		expr
		expr as simpleidentifier
Jointype ::=		left [outer] join
		right [outer] join

		[[inner] join
		full [outer] join
joinFromClauseItem	::=	fromClauseItem cross join fromClauseItem
		fromClauseItem joinType fromClauseItem
om expr		
applyType	::=	crossapply
		outer apply
applyfromClauseItem	::=	fromClauseItem applyType fromClauseItem
whereClause	::=	where expr
groupByClause	::=	group [identifier] by aliasedExpr
		{,aliasedExpr}
havingClause	::=	having expr
orderByClause	::=	order by orderByClauseItem
		{,orderByClauseItem}
orderByClauseItem	::=	expr[collate collation] [asc desc]
collation	::=	simpleIdentifier
identifier	::=	quotedIdentifier simpleIdentifier
quotedIdentifier	::=	[UnicodeCharacters]
		"unicodeCharacters"
unicodeCharacters	::=	unicodeCharacter {unicodeCharacter}
SimpleIdentifier	::=	letter [letter digit _]
numericLiteral	::=	int32Literal
		int64Literal
		doubleLiteral
		floatLiteral
		decimalLiteral
nullLiteral	::=	null
booleanLiteral	::=	true false
int32Literal	::=	digit {digit}
int64Literal	::=	integerLiteralL
doubleLiteral	::=	digit {digit} .digit {digit}
		[(E e) [+ -] digit {digit}]
floatLiteral	::=	doubleLiteralf
decimalLiteral	::=	digit{digit}.digit{digit} M
stringLiteral	::=	'{unicodeCharacter}'
		" {unicodeCharacter}"

APÊNDICE B: Especificação detalhada para extensões para a linguagem de consulta exemplar

1. Visão geral

1.1 Finalidade

Esse documento detalha as extensões (funções) para a linguagem de consulta eSQL para WinFS/CDP.

eSQL provê suporte para invocar funções, porém não provê nenhuma função incorporada sozinha. (Operadores como +, -, são tratadas diferentemente, e são assumidas como sendo preparadas na própria linguagem). Os usuários de linguagens como T-SQL e outros dialetos de SQL são acostumados a um conjunto rico de funções incorporadas. A finalidade desse documento é descrever como eSQL pode ser estendida para tratar dessas necessidades.

De modo fundamental, todas as funções em eSQL são tratadas iguais: as funções definidas pelo usuário. eSQL utiliza espaços de nome para definir diferentes recipientes de funções em um modo similar ao uso de espaços de nome para conter tipos. Provedores podem definir tais espaços de nome com o conjunto associado de funções, e tornar os mesmos disponíveis via serviços de metadados para usuários de eSQL. Esse documento focaliza no conjunto específico de funções incorporadas disponíveis a partir do Servidor SQL e WinFS e os espaços de nome através dos quais são tornados disponíveis.

Esse documento não trata da questão de como os provedores fornecem essa lista de funções, e como essas são carregadas pelo resto do sistema.

Essas extensões são agrupadas pelos provedores de armazenagem que suportam os mesmos. A lista de provedores é como a seguir:

1. SQL Server 2000.
2. SQL Server 2005. Esse provedor suporta todas as funções de SQL 2000.
3. WinFS. Esse provedor suporta todas as funções SQL 2005 e SQL 2000.

A seguinte tabela sumariza a funcionalidade:

área	Espaço de nome	descrição	SQL2000	SQL2005	WinFS
Agregados	System.Data.TSql.Aggregates	funções como min(), Max(), sum() etc.	✓	✓	✓
Funções de série	System.Data.TSql.string	Concat(), substring(), Len() etc.	✓	✓	✓
funções de matemática	System.Data.TSql.Math	Abs(), Sqrt(), Sign() etc.	✓	✓	✓
funções DateTime	System.Data.TSql.Date	GetDate(), GetUTCDate() etc.	✓	✓	✓

funções XML	System.Data.TSql.XML	consultas XML		✓	✓
funções WinDS	System.Data.TSql.WinDS	SumString de busca de texto completo WinFS			✓

Por favor vide as seções posteriores do documento para casos listados como suporte parcial na tabela acima.

1.2 Mecanismo de extensão

Detalhes sobre o mecanismo de extensão eSQL que permitem a funcionalidade de extensão podem ser encontrados na seção 9.4 (Espaços de nome) da especificação de Linguagem eSQL. Observe que espaços de nome parciais não são suportados; vide a especificação de linguagem eSQL para mais detalhes.

Espaços de nome podem ser importados em uma consulta utilizando a cláusula utilizar de eSQL. Alternativamente, um ou mais espaços de nome podem ser especificados fora dos limites da consulta, por exemplo, como parte dos objetos de Comando/Conexão.

1.3 Execução de extensão

Execução de consulta (incluindo funções de extensão) sempre ocorre na armazenagem. Adicionalmente, a menos que mencionado de outro modo, espera-se que a semântica das funções de extensão sejam igual a do servidor. Os analisadores de eSQL validam a assinatura das invocações de função; entretanto, qualquer validação semântica é executada na armazenagem subjacente.

2. Desenho detalhado

2.1 agregados

Funções de agregados executam um cálculo sobre um multiconjunto de valores e retornam um único valor. Com a exceção de COUNT, funções de agregado ignoram valores nulos. As seguintes funções SQL disponíveis são suportadas:

função	Tipos de dados	Descrição
Avg(x)	Suporta int e flutuante. Tipo de Retorno int ou flutuante respectivamente.	Retorna a média dos valores em uma coleção. Valores nulos são ignorados.
Count(x)	Suporta todos os tipos de dados CDM. Tipo de retorno int.	Retorna o número de itens em uma coleção.
Max(x)	Suporta tipos numéricos, série e tipos de datetime. Tipo de retorno é igual ao da expressão x.	Retorna o valor máximo na coleção
Min(x)	Suporta tipos numéricos, série e	Retorna o valor mínimo na coleção.

	tipos de datetime. Tipo de retorno é igual ao da expressão x.	
Sum(x)	Suporta int e flutuante. Tipo de retorno int ou flutuante respectivamente.	Retorna a soma de todos os valores em uma coleção. Valores nulos são ignorados.

Funções de agregado ignoram valores nulos ao computar o valor (exceto para contagem). Observe também que count(*) não é suportado. Funções de agregado podem ser utilizadas em um predicado e em uma definição de projeção. Adicionalmente, os agregados suportam um modo “distinto” (similar a suporte de SQL) em tipos de dados que suportam igualdade. O tipo de retorno dessas funções se baseia no tipo da propriedade contra a qual a função é executada. Assim, por exemplo, avg(int) retornará um int. Isso pode não ser esperado visto que a média de 2,3 retornará 2 em vez de 2,5. Para trabalhar isso o usuário necessita cast explicitamente a propriedade para um tipo que tenha a precisão correta, por exemplo, avg(cast(intProperty as double)).

10

2.1.1 Espaço de nome

Essas funções estão no espaço de nome System.Data.Tsql.Aggregates.

2.1.2 Exemplos

```
Using System.Data.TSql.Aggregates;
Select count (i.Children) as Count
From Items as i;
Select Max (i.displayName)
From Items as i
Group by i.gender;
```

2.1.3 Plataformas suportadas

Suportado em: SQL 2000, SQL 2005 e WinFS.

15

2.2 Funções de série

Essas funções escalares executam uma operação em um valor de entrada de série e retornam um valor numérico ou de série.

função	Descrição
Ascii(x)	Retorna o valor de código ASCII do caractere mais a esquerda de uma expressão de caractere
Char(x)	Uma função de série que converte um código ASCII int em um caractere

Replicate(x, y)	Repete uma expressão de caractere x para um número especificado de tempos y
Patindex (x, y)	Retorna a posição de partida da primeira ocorrência de um padrão x em uma expressão especificada y, ou zero se o padrão não é encontrado, em todos os tipos de dados de caractere e texto válido
Stuff (x, y, z, a)	Deleta um comprimento especificado de caracteres z e insere outro conjunto de caracteres a em um ponto de partida especificado y em uma expressão de série x
Space(x)	Retorna uma série de espaços repetidos, onde x é uma expressão de número inteiro
Len(x)	Retorna o número de caracteres, em vez do número de bytes, da expressão de série dada, excluindo lacunas posteriores
Substring(x, y, [, z])	Retorna uma parte da série, iniciando no índice (baseado em um) y e para um número opcional de caracteres z
Lower(x)	Retorna uma cópia dessa série convertida em letra minúscula, utilizando as regras de cobertura da cultura atual
Upper(x)	Retorna uma cópia dessa Série convertida em letra maiúscula, utilizando as regras de cobertura da cultura atual
Charindex(x, y, [,z])	Reporta o índice da primeira ocorrência da série especificada y em x. A busca pode iniciar opcionalmente em uma posição de caractere especificado z
Ltrim(x)	Retorna uma expressão de caractere após remover lacunas dianteiras
Rtrim(x)	Retorna uma série de caracteres após truncar todas as lacunas traseiras
Replace(x,y,z)	Substitui todas as ocorrência da segunda expressão de série dada y na primeira expressão de série x com uma terceira expressão z
Reverse(x)	Retorna o inverso de uma expressão de série.

Por favor observe que todas as funções de série tem índices baseados em um.

2.2.1 Espaço de nome

Essas funções estão no espaço de nome System.Data.Tsql.String.

2.2.2 Exemplos

Using System.Data.String;

Select substring(reverse(i.DisplayName), 1, 5)
From Itens as i

2.2.3 Plataformas suportadas

Suportadas em: SQL 2000, SQL 2005 e WinFS.

2.3 Funções de matemática

Essas funções escalares executam um conjunto comum de operações matemáti-

5 cas.

função	Descrição
Abs(x)	Retorna o valor positivo absoluto da expressão numérica dada
Acos(x)	Retorna o ângulo, em radianos, cujo co-seno é a expressão flutuante dada; também denominado arccosine.
Asin(x)	Retorna o ângulo, em radianos, cujo seno é a expressão flutuante dada (também denominado arcsine)
Atan(x)	Retorna o ângulo em radianos cuja tangente é a expressão flutuante dada (também denominado artangent)
Atn2(x,y)	Retorna o ângulo, em radianos, cuja tangente está entre as duas expressões flutuantes dadas (também denominado arc-tangent)
Ceiling(x)	Retorna o menor número inteiro maior do que, ou igual à expressão numérica dada
Cos(x)	Uma função matemática que retorna o co-seno trigonométrico do ângulo dado (em radianos) na expressão dada
Exp(x)	Retorna o valor exponencial da expressão flutuante dada
Floor(x)	Retorna o maior número inteiro menor ou igual à expressão numérica dada
Log(x,[,y])	Retorna o logaritmo natural da expressão flutuante dada
Log10(x)	Retorna o logaritmo base-10 da expressão flutuante dada
Pow(x,y)	Retorna o valor da expressão dada (x) à potência especificada (y)
Round(x,[,y])	Retorna uma expressão numérica, arredondada para o comprimento ou precisão especificado
Sign(x)	Retorna o sinal positivo (+1), zero (0) ou negativo (-1) da expressão dada
Sqrt(x)	Retorna a raiz quadrada da expressão dada
Tan(x)	Retorna a tangente da expressão de entrada

Degrees(x)	Dado um ângulo em radianos, retorna o ângulo correspondente em graus
Radians(x)	Retorna radianos quando uma expressão numérica, em graus, é entrada
Rand([x])	Retorna um valor flutuante aleatório a partir de 0 até 1
Sin(x)	Retorna o seno trigonométrico do ângulo dado (em radianos) em uma expressão (flutuante) numérica aproximada
Square(x)	Retorna o quadrado da expressão dada
PI()	Retorna o valor constante de PI como um flutuante

2.3.1 Espaço de nome

Essas funções estão no espaço de nome System.Data.TSql.Math.

2.3.2. Exemplos

```
Using System.Data.TSql.////math;
Select PI()
From Items as i
```

2.3.3 Plataformas suportadas

Suportadas em: SQL 2000, SQL 2005 e WinFS.

2.4 Funções de dados

Essas funções escalares executam uma operação em um valor de entrada de data e horário e retornam um valor de série, numérico ou data e horário:

função	descrição
DateAdd(x,y,z)	Retorna um novo valor de datetime com base na adição de um intervalo à data especificada. Onde x é uma parte da data (por exemplo, Ano, Mês, dia) e y é um número e z é uma data
DateDiff(x,y,z)	Retorna o número de limites de data e horário cruzado entre duas datas especificadas. Onde x é uma parte da data (por exemplo, ano, mês, dia) e y é a data de início e z é a data final.
DateName(x,y)	Retorna uma série de caractere representando a parte da data especificada da data especificada. Onde x é uma parte da data (por exemplo, Ano, Mês, dia) e y é a data
DatePart(x,y)	Retorna um número inteiro representando a parte da data especificada da data especificada. Onde x é uma parte da

	data (por exemplo Ano, Mês, dia) e y é a data.
Day(x)	Retorna um número inteiro representando a parte da data de Dia da data especificada. Onde x é a data.
GetDate()	Retorna a data atual e horário (no servidor) como um DateTime. Datetime.Kind será definido para horário local.
GetUtcDate()	Retorna o valor de datetime em formato UTC. DateTime.Kind será definido para UTC. Observe que pode haver uma perda em precisão.
Month(x)	Retorna um número inteiro representando a parte da data do Mês da data especificada. Onde x é a data
Year(x)	Retorna um número inteiro representando a parte da data do Ano da data especificada. Onde x é a data

2.4.1 Espaço de nome

Essas funções estão no espaço de nome System.Data.TSql.Date.

2.4.2 Exemplos

A parte da data necessita ser especificada como um literal de série:

```
Using System.Data.TSql.Date;
Select i
From Items as i
Where i.ModifiedTime > dateadd ('day', -10,
i.ModifiedTime)
```

5

2.4.3 Plataformas suportadas:

Suportadas em: SQL 2000, SQL 2005 e WinFX.

2.5 Funções do sistema

Há um conjunto de funções de Sistema que variam por plataforma que vivem no espaço de nome System.Data.TSql.System. A seguinte tabela resume as mesmas:

função	descrição	SQL 2000	SQL 2005	WinFS
Soundex(x)	Onde x é uma expressão de série	√	√	√
Difference(x,y)	X e y são expressões de série	√	√	√
DataLength(x)	Onde x é uma propriedade	√	√	√

Current_timestamp	Retorna a data e horário atuais. Essa função é equivalente a GET-DATE()	√	√	√
Newid()	Cria um valor exclusivo de Guid de tipo	√	√	√

As seguintes seções fornecem mais detalhes para funções específicas.

2.5.1 Soundex & Diferença

A função Soundex mapeia para a função Soundex T-SQL e é utilizada para converter uma série alfa para um código de quatro caracteres para encontrar palavras ou nomes de som similar. O primeiro caractere do código é o primeiro caractere de character_expression e o segundo até quarto caracteres do código são números. Vogais no character_expression são ignoradas a menos que sejam a primeira letra da série. Funções de série podem ser aninhadas. A sintaxe para isso é como a seguir:

String soundex (string character_expression)

A Diferença retorna a diferença entre os valores SOUNDEX de duas expressões de caractere como um número inteiro. Essa função pode ser útil para separar nomes por pronúncia, por exemplo. A sintaxe para isso é como a seguir:

Strin soundex (string character_expression)

O que se segue são exemplos de utilizar soundex e diferença em uma consulta:

```
Using System.Data.TSql>System;
// get the data length of DisplayName for all items
Select p. DisplayName, p.Fullname.givenName,
p.FullName.Surname
from OfType(Items, Person) as p
order by soundex (p.FullName.Surname) desc
```

2.5.1.1 Plataformas suportadas

Suportadas em: SQL 2000, SQL 2005 e WinFS.

2.5.2 comprimento de dados

A função de comprimento de dados é utilizada para retornar o tamanho de propriedades (as an int). Mapeia para a função de comprimento de dados SQL. A sintaxe para isso é como a seguir:

Int datalength (property)

No caso onde a propriedade não tenha valor (isto é nula) o valor de retorno é zero *not*null. Isso é ao contrário do modo em que a função de comprimento de onda SQL opera. O que se segue são exemplos de utilizar comprimento de dados em uma consulta:

```
Using System.Data.TSql>System;
// get the data length of DisplayName for all items
Select WinFS.Datalength (i.DisplayName)
From Items as i

// get the data length of the message Body
Select sum(datalength(m.Body..Content)),
m.subject
from OfType(Items, Message) as m
order by soundex (p.FullName.Surname) desc
```

2.5.2.1 Plataformas suportadas

5 Suportadas em: SQL 2005 e WinFS.

2.6 Funções XML

Suporte para consultas XML é habilitado via funções de extensão. As seguintes funções de extensão são fornecidas:

função	descrição	Comentários
Consulta	Método query() de tipo de dados XML	Instância xml é primeira arg
Valor	Método value() de tipo de dados XML	Instância xml é primeira arg; (segunda arg é sempre assumida como sendo série SqlType)
existe	Método exist() do tipo de dados XML	Instância xml é primeira arg

XML query (XML xmldoc, string query);

Boolean exist (XML xmldoc, string query);

String value (XML, xmldoc, string query);

2.6.1 Espaço de nome

Essas funções estão no espaço de nome System.Data.TSql.XML.

2.6.2 Exemplos

```
Using System.Data.TSql.XML;
Select i
From Items as i
Where exist (i.XMLProperty, '//text()')
```

2.6.3 Plataformas suportadas:

5 Suportadas em: SQL 2005 e WinFS.

2.7 Funções WinFS

2.7.1 Consultas de texto completo (WinFS somente)

WinFS suportará dois TVFs – Contém e TextoLivre (duas sobrecargas para cada) – para busca de texto completo. Essas funções são mapeadas para funcionalidade em MS-
10 Search:

```
type FTFElementType as row (ItemId Guid, Rank Int);
type FTFTType as Multiset <FTFElementType>;
FTFTType Contains (string containsQuery);
FTFTType Contains (string propName, String containsQuery);
FTFTType FreeText(string freeTextQuery);
FTFTType FreeText (string propName, string freeTextQuery);
```

Todas essas funções retornam um multiconjunto de linhas – cada com dois campos. O campo ItemId é um Guid e é o id de um Item que casou com o predicado de texto completo. O campo Rank é a Classificação para aquele item. O resultado não é separado pela Classificação – um ordenar por explícito deve ser utilizado na consulta para obter isso.

15 Contém

Contém modela o predicado Contains da MSSearch. O argumento containsQuery deve se conformar à sintaxe especificada pelo predicado Contains de MSSearch. O argumento propName deve se conformar à sintaxe utilizada para especificar uma propriedade de busca durante instalação de esquema WinFS.

20 FreeText

FreeText modela predicado FreeText de MSSearch. O argumento freeTextQuery deve se conformar à sintaxe especificada pelo predicado FreeText de MSSearch. O argumento propName deve se conformar à sintaxe utilizada para especificar uma propriedade de

busca durante instalação de esquemas WinFS.

Observações

. eSQL não interpreta nenhum dos argumentos. Cuida somente que eles devam ser séries.

5

2.7.1.1 Mecânica

Instalação de esquemas

Quando um esquema WinFS é instalado, quaisquer propriedades de busca são identificadas e armazenadas no conjunto de cliente. A sintaxe para a especificação de propriedade é como a seguir:

Descrição	Espec	Exemplo
propriedade de nível superior de Item ou um de seus subtipos	<typename> <property-name>	Contact.DisplayName
propriedade aninhada de item ou um de seus subtipos	<typename> <property-name> <propertyname>	Contact. FullName. First-Name
propriedade e um subtipo de um tipo em linha aninhada	Treat(<typename> <propertyname> as type) <propertyname>	Treat (Contact.Adress as UsAddress).Zip
propriedade de uma coleção	<typename> <property-name> <propertyname>	Contact.Phones..AreaCode
todas as propriedades		

10

Processamento de consulta

Como mencionado anteriormente, eSQL não interpreta os argumentos para essas funções. Mais especificamente, o runtime WinFS – que analisa a consulta – não executa nenhuma verificação diferente de assegurar que os parâmetros sejam séries.

O Provedor WinFS é responsável por interpretar o argumento propertyName – possivelmente como parte de SqlGen, consultando o id de propriedade apropriada no conjunto de cliente, e então traduzindo o mesmo para uma chamada TVF equivalente na armazenagem. As TVFs de armazenagem terão assinaturas que se parecem com:

15

```
Create function FullTextSearch(@sqlStr nvarchar (Max))
Returns @rTable table (ItemId
[System.Storage.Storage, 0.1,*, f45b95b0f79f4fbe].ItemId,
Rank int)
```

Create function FulltextSearch(@propId uniqueidentifier,
@sqlStr nvarchar (Max))

Returns @rTable table (ItemId
[System.Storage.Storage, 0.1,*, f45b95b0f79f4fbe].ItemId,
Rank int)

2.7.1.2 Espaço de nome

Essas funções estão no espaço de nome System.Data.TSql.WinFS.

2.7.1.3 Exemplos

Os seguintes exemplos ilustram o uso dessas funções:

```
using System.Data.TSql.WinFS;

select i.DisplayName, c.Rank
from Items as i,
      Contains('foo and bar') as c
where i.ItemId = c.ItemId;

select i.DisplayName, c.Rank
from Items as i,
      Contains('Contact.DisplayName', 'foo and bar')
as c
where i.ItemId = c.ItemId;

select i.DisplayName, c.Rank
from Items as i,
      FreeText('Treat(Contact.EAddress as
UsAddress).Zip',
              '12345') as c
where i.ItemId = c.ItemId
order by c.Rank desc;
```

5 2.7.1.4 Plataforma suportada

Suportada em: WinFS.

2.7.2 SumString (WinFS somente)

10 O agregado SumString (que mapeia para uma UDF de armazenagem denominada SumString) provê a capacidade de concatenar o conteúdo de uma coleção de série em uma única série com um delimitador. Isso é útil para diversos cenários App incluindo exibir os participantes de lista de Mensagem 'para' em um único campo de texto.

15 Essa extensão eSQL é uma função de agregado similar à contagem, soma, max, etc., e pode ser aplicada a propriedades de série em InlineObjectsCollections, itemFragmentCollections, ExtensionCollections e qualquer propriedade de série como resultado de uma operação de Grupo. A sintaxe para esse agregado é como a seguir:

String SumString (Multiset<string> coll)

Onde a coleção tem de ser uma coleção de séries.

2.7.2.1 Espaço de nome

Essas funções estão no espaço de nome System.Data.TSql.WinFS.

2.7.2.2 Exemplos

5 O que se segue é um exemplo do uso de SumString em uma consulta:

```
Using System.Data.TSql.WinFS;
//concatenate the list of participantEAddress together
Select
SumString(m.Participants.?(ParticipantType=Sender)
        ..ParticipantAddress.Address),
        m.subject
from ofType(Itens, Message) as m
```

2.7.2.3 Plataformas suportadas

Suportada em: WinFS.

2.7.3 NormalizeDigits (somenteWinFS)

10 O agregado NormalizeDigits (que mapeia para uma UDF de armazenagem denominada normalizedigits) provê a capacidade de separar séries em um formato similar à separação de armação de janela. A sintaxe para essa função é como a seguir:

String NormalizeDigits(string property, int maxlength)

2.7.3.1 Espaço de nome

Essas funções estão no espaço de nome System.Data.TSql.WinFS.

2.7.3.2 Exemplos

15 O que se segue é um exemplo de utilizar SumString em uma consulta:

```
Using System.Data.TSql.WinFS;;
// order items using the function
Select i.ItemId, i.NamespaceName
From Items as i
Order by NormalizeDigits(i.NamespaceName, 4000)
```

2.7.3.3 Plataformas suportadas

Suportada em: WinFS.

3 Carregamento de Espaço de nome

Todos os espaços de nome descritos nesse documento devem ser automaticamente carregados ao conectar-se a um provedor SqlCenter. Não deve haver necessidade de utilizar uma cláusula "usar" explícita.

4 Propriedades e Métodos calculados

A linguagem de consulta eSQL suporta um mecanismo de extensão pelo qual propriedades que são adicionadas à classe parcial O-Space (isto é propriedades não mapeadas) podem ser referenciadas em consultas. Essas propriedades exigem que uma expressão eSQL seja associada a elas através de um atributo. Essa expressão é então substituída sempre que a propriedade for referenciada em uma consulta.

A expressão de eSQL que é fornecida pode referenciar outras propriedades e métodos calculados, outras propriedades mapeadas e outras funções que são especificadas no manifesto Provedor.

4.1 Propriedades calculadas

O System.Data.Objects.CalculatedAttribute pode ser adicionado a uma Propriedade que é somente definido em uma classe de cliente (sem propriedade mapeada) para permitir referência da propriedade em consultas.

O tipo da Propriedade CLR deve casar com o tipo retornado pela expressão eSQL. Se o tipo de retorno da expressão for um tipo anulável então a propriedade deve ser declarada como um tipo anulável. Essa limitação é executada pela infra-estrutura, e uma exceção será lançada. Se o tipo de retorno da expressão for uma coleção então a Propriedade deve ser do tipo Query<T>.

A propriedade ItemSize no Item é um exemplo efetivo de uma propriedade calculada que existe no Esquema hoje. O que se segue é um exemplo do uso dessa propriedade:

```
// The following retrieves all documents that are greater than 10k (~10000
bytes) in a particular folder
Query<Item> searcher = wd.Items.Source.Where("it.ItemSize > 100000");
```

Essa propriedade será marcada com o seguinte atributo:

```
[Calculated (Expression = "System.Storage.Item.GetItemSize(ItemId)")]
public int ItemSize {get;}
```

A expressão acima se refere a uma função no manifesto do provedor que mapeia para uma função do lado de armazenagem.

PrimaryEmailAddress é um exemplo de uma propriedade que poderia ser calculada a partir do esquema e agora através dessa funcionalidade pode ser predicada e projetada

através de eSQL. Essa propriedade poderia retornar o endereço de E-mail de uma Pessoa marcada com a Palavra-chave Primary. O código que se segue ilustra como definir essa propriedade PrimaryEmailAddress para permitir consultar a mesma através de eSQL:

```
public partial class PersonItem : ...
{
    ...
    // Attribute the PrimaryEmailAddress property
    // the @keyword refers to the calculated parameter
    [Calculated(Expression= @
        "using System.Storage.Contacts;

        element(select value 'e
            from this.EAddresses as e
            where e is of SmtpeMailAddress
            and exists(select 1
                from e.Keywords
                where Value=@keyword)))")]
    [CalculatedParamotor(Name="keyword",
        Value="System.Storage.Contacts.Primary")]
    public SmtpeMailAddress PrimaryEmailAddress
    {
        get
        {
            foreach ( SmtpeMailAddress smtpEA in
                EAddresses.OfType<SmtpeMailAddress>() )
            {
                foreach(Keyword k in smtpEA.Keywords)
                {
                    if(k.Value == "System.Storage.Contacts.Primary")
                        return smtpEA;
                }
            }
        }
    }
}
```

Essa propriedade pode ser então utilizada em uma consulta como a seguir:

```
Query<Person> searcher = wd.Items.Source.OfType<Person>().Where("
PrimaryEmailAddress.Address='rameshn@microsoft.com'");

Person person = searcher.GetSingle(); //Note the GetSingle is being added as
a DCR

//The property can be projected as follows:
Query<DataRecord> projection =
wd.Items.Source.OfType<Person>().Select("it.DisplayName,
it.PrimaryEmailAddress");

// Enumerate the results
foreach (DataRecord records in projection)
{
    string displayName = (string) record["DisplayName"];
    SmtpeMailAddress primaryEmailAddress = (SmtpeMailAddress)
record["PrimaryEmailAddress"];
}
```

5 Observações sobre Expressão eSQL:

- . O prefixo "this" tem de ser utilizado para referenciar outras Propriedades/Métodos (mapeado ou calculado) na instância.
- . a expressão somente é validada em runtime.

- . qualquer expressão eSQL válida pode ser fornecida, incluindo as cláusulas usar.
- . Parâmetros calculados necessitam ser prefixos com @ para referenciar os mesmos na expressão.

4.2 Métodos calculados

- 5 Similar às propriedades calculadas, os métodos definidos em classes parciais também podem ser habilitados para serem utilizáveis em eSQL. O System.Data.Objects.CalculatedAttribute pode ser adicionado ao método em um modo similar para adicionar o mesmo a uma propriedade. As seguintes semântica e restrições se aplicam a métodos calculados:
- 10 . Métodos genéricos (por exemplo X<T>) não são atualmente suportados em eSQL, isto é o parâmetro do tipo genérico não será utilizado na expressão eSQL.
- . a palavra chave params C# que permite um número variável de parâmetros para um método não é suportado nos métodos calculados.
 - . Polimorfismo de métodos calculados não é atualmente suportado.
- 15 . atualmente não há suporte para métodos calculados internos ou privados externamente definidos, isto é a API deve ter acesso à propriedade/método.
- . Em geral chamadas recursivas em métodos calculados não é suportado.
 - . a sobrecarga de métodos calculados por valores de parâmetro (tipos) não é atualmente suportada.
- 20 . o tipo de retorno do método deve casar o tipo retornado pela expressão eSQL.
- ° se o tipo de retorno da expressão for um tipo anulável então o método deve ser declarado como retornando um tipo anulável. Essa limitação é executada pela infraestrutura, e uma exceção será lançada.
 - ° se o tipo de retorno da expressão for uma coleção então o tipo de retorno do método deve ser do tipo Query<T>.
- 25 . Ocultar métodos pela especificação do operador novo não é suportado.
- . Métodos virtuais não são suportados.
 - . Métodos que retornam vazio não são suportados.
 - . Parâmetros de método não exigem um prefixo para referenciar os mesmos na expressão
- 30 O que se segue é um exemplo do uso de um método calculado. O cenário aqui é para gerar uma vista de agrupamento favorável similar à vista agrupar por data Outlook, como ilustrado na figura 3.
- 35 O código a seguir encapsula isso em um método de modo que possa ser reutilizado em outro lugar:

```

public partial class Message : ...
{
    //This method returns the employees of an Organization
    //The employer in the expression refers to the method parameter
    employer
    [Calculated(Expression=@"
        using System.Data.TSql.Date;

        case
            when datediff('hour',this.ReceivedTime, date) < 0
            then 'Future'
            when datediff('hour',this.ReceivedTime, date) < 24
            then 'Today'
            when datediff('hour',this.ReceivedTime, date) between 24 and
            then 'Yesterday'
            when datediff('hour',this.ReceivedTime, date) between 48 and
            (24*(datepart('weekday',date)))
            then datename(weekday,this.ReceivedTime)
            when datediff('hour',this.ReceivedTime, date) between
            (24*(datepart('weekday',date))) and (24*(datepart('weekday',date)))+(7*24)
            then 'Last Week'
            when datediff('hour',this.ReceivedTime, date) between
            (24*(datepart('weekday',date)))+(7*24) and
            (24*(datepart('weekday',date)))+(14*24)
            then 'Two Weeks Ago'
            when datediff('hour',this.ReceivedTime, date) between
            (24*(datepart('weekday',date)))+(14*24) and
            (24*(datepart('weekday',date)))+(21*24)
            then 'Three Weeks Ago'
            when datediff('hour',this.ReceivedTime, date) between
            (24*(datepart('weekday',date)))+(21*24) and
            (24*(datepart('weekday',date)))+(28*24) and
            datediff('month',this.ReceivedTime,date) < 2
            then 'Last Month'
            else 'Older'
        end as RelativeDate"
    )]]
    public string GetReceivedTimeName(DateTime date)
    {
        //The implementation of this method would need to do similar
        logic to the expression for consistency
        ...
    }
}

```

Isso pode ser então utilizado para agrupar os resultados com a seguir:

```

// Create a query over messages to group by the ReceivedTime Name
Query<DataRecord> projection =
context.Items.Source.OfType<Message>().Select("
it.Subject, it.GetReceivedTimeName(@date) as RelativeDate", new
QueryParameter("date", DateTime.Now)).GroupBy("RelativeDate");

// Enumerate the results
foreach (DataRecord record in projection)
{
    ...
}

```

4.3 Atributos Calculado & CalculatedParameter

O System.Data.objects.CalculatedAttribute é definido como a seguir:

```

namespace System.Data.Objects
{
    [System.AttributeUsage(AttributeTargets.Property |
        AttributeTargets.Method)]
    public class CalculatedAttribute : Attribute
    {
        public string Expression {get; set;}
        public CalculatedAttribute(string expression);
    }
}

```

Esse atributo define as seguintes propriedades:

- . Expressão. A propriedade expressão deve conter uma expressão eSQL válida que projeta um valor. Esse valor pode ser então utilizado em outra expressão. Vide abaixo para mais detalhes sobre como especificar parâmetros denominados em expressões.

5 O System.Data.Objects.CalculatedParameterAttribute é definido como a seguir:

```

namespace System.Data.Objects
{
    [AttributeUsage(AttributeTargets.Property | AttributeTargets.Method,
        AllowMultiple = true)]
    public class CalculatedParameterAttribute : Attribute
    {
        public string Name {get; set;}
        public object Value {get; set;}
        public CalculatedParameterAttribute();
    }
}

```

Esse atributo define as seguintes propriedades:

- . Nome. O nome do parâmetro como referenciado na propriedade Expressão de um Atributo calculado.

- . valor. O valor do parâmetro definido pela propriedade Nome.

10 Os parâmetros Denominados definidos em um CalculatedAttribute seguem essas semânticas:

- . referências a parâmetros calculados devem ter um prefixo @

- . a ordem de definir o CalculatedParameterAttribute e CalculatedAttribute não importa

15 . Nomes de parâmetro devem ser exclusivos através dos nomes de parâmetro de método e outros parâmetros denominados definidos pelo CalculatedParameterAttribute no método.

- . Para uma propriedade calculada quaisquer parâmetros denominados na expressão devem ser definidos por CalculatedParameterAttribute.

20 . O exemplo a seguir ilustra o uso das classes de atributo acima:

```
public partial class Person : ...
{
    //The @year in the expression refers to the named parameter year
    provided by
    //the CalculatedParameterAttribute
    [Calculated(Expression=@"
    using System.Data.TSql.Date;

    (@year - year(this.BirthDate))"]
    [CalculatedParameter(Name="year",Value=System.DateTime.Year)]
    public int GetRoughAge()
    { ... }
}
```

REIVINDICAÇÕES

1. Sistema para processar uma solicitação de aplicação para dados, **CARACTERIZADO** por compreender:

5 uma camada de serviços de objeto (210) para aceitar uma solicitação, em que a solicitação compreende uma expressão de coleção em uma cláusula a partir de;

uma camada de provedor de mapa (220) que fornece uma pluralidade de transformações de mapeamento para transformar a solicitação em uma consulta de banco de dados;

10 uma camada de ponte (230) para converter a consulta de banco de dados em uma consulta de banco de dados modificada, a consulta de banco de dados modificada sendo adaptada a pelo menos uma necessidade de um primeiro banco de dados (240);

o primeiro banco de dados (240) compreende um motor de processamento de consulta para processar a consulta de banco de dados modificada, em que o primeiro banco de dados é configurado para retornar um resultado para a camada de ponte.

15 2. Sistema, de acordo com a reivindicação 1, **CARACTERIZADO** pelo fato de que a solicitação compreende ainda uma operação definida selecionada a partir do grupo que compreende união, interseção e exceto, onde a operação definida opera em coleções.

3. Sistema, de acordo com a reivindicação 1, **CARACTERIZADO** pelo fato de que a solicitação compreende ainda uma operação junção que opera em coleções.

20 4. Sistema, de acordo com a reivindicação 1, **CARACTERIZADO** pelo fato de que a camada de ponte (230) é configurada para converter a consulta de banco de dados em qualquer de uma pluralidade de consultas de banco de dados modificada, adaptada a necessidades de qualquer de uma pluralidade de bancos de dados (240).

25 5. Sistema para processar uma solicitação de aplicação para dados, **CARACTERIZADO** por compreender:

uma camada de serviços de objeto (210) para aceitar uma solicitação, em que a solicitação compreende uma operação definida selecionada do grupo que compreende união, interseção e exceto, em que a operação definida opera em coleções;

30 uma camada de provedor de mapa (220) que fornece uma pluralidade de transformações de mapeamento para transformar a solicitação em uma consulta de banco de dados;

uma camada de ponte (230) para converter a consulta de banco de dados em uma consulta de banco de dados modificada, a consulta de banco de dados modificada sendo adaptada a pelo menos uma necessidade de um primeiro banco de dados (240);

35 o primeiro banco de dados (240) compreende um motor de processamento de consulta para processar a consulta de banco de dados modificada, em que o primeiro banco de dados é configurado para retornar um resultado para a camada de ponte.

6. Sistema, de acordo com a reivindicação 5, **CHARACTERIZADO** pelo fato de que a solicitação compreende ainda uma operação junção que opera em coleções.

7. Sistema, de acordo com a reivindicação 5, **CHARACTERIZADO** pelo fato de que a camada de ponte (230) é configurada para converter a consulta de banco de dados em qualquer de uma pluralidade de consultas de banco de dados modificada, adaptada a necessidades de qualquer de uma pluralidade de bancos de dados (240).

8. Sistema para processar uma solicitação de aplicação para dados, **CHARACTERIZADO** por compreender:

uma camada de serviços de objeto (210) para aceitar uma operação junção que opera em coleções;

uma camada de provedor de mapa (220) que fornece uma pluralidade de transformações de mapeamento para transformar a solicitação em uma consulta de banco de dados;

uma camada de ponte (230) para converter a consulta de banco de dados em uma consulta de banco de dados modificada, a consulta de banco de dados modificada sendo adaptada a pelo menos uma necessidade de um primeiro banco de dados (240);

o primeiro banco de dados (240) compreende um motor de processamento de consulta para processar a consulta de banco de dados modificada, em que o primeiro banco de dados é configurado para retornar um resultado para a camada de ponte (230).

9. Sistema, de acordo com a reivindicação 8, **CHARACTERIZADO** pelo fato de que a camada de ponte é configurada para converter a consulta de banco de dados em qualquer de uma pluralidade de consultas de banco de dados modificadas, adaptada a necessidades de qualquer de uma pluralidade de bancos de dados.

10. Sistema de computador **CHARACTERIZADO** por compreender uma aplicação (200) configurada para solicitar dados a partir de um banco de dados (240), em que a aplicação gera uma solicitação de acordo com uma linguagem de consulta, a linguagem de consulta compreendendo pelo menos as seguintes características:

a linguagem de consulta permite uso de uma expressão de coleção em uma cláusula a partir de;

a linguagem de consulta permite uso de uma operação definida selecionada a partir do grupo que compreende união, interseção e exceto, em que a operação definida opera em coleções;

a linguagem de consulta permite o uso de uma operação junção que opera em coleções.

11. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta ainda exige que toda consulta compreenda pelo menos uma expressão.

12. Sistema de computador, de acordo com a reivindicação 11, **CHARACTERIZADO** pelo fato de que cada expressão tem uma interpretação uniforme que não depende do contexto no qual é utilizado.

5 13. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso das seguintes expressões:

$1+2*3$

"abc"

Row (1 as a, 2 as b)

{1, 3, 5}

E1 union all e2

Distinct (e1)

14. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta considera todas as subconsultas como sendo subconsultas de multiconjunto.

10 15. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que se um valor escalar for desejado a partir de uma subconsulta, a linguagem de consulta requer que a aplicação forneça um operador de elemento que opera em uma coleção.

15 16. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de um operador de elemento para extrair um valor singleto a partir de uma coleção, e uma cláusula selecionar valor para evitar a criação de um wrapper de linha durante uma expressão de consulta.

20 17. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de uma cláusula selecionar valor selecionado para pular uma construção de linha implícita.

20 18. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de um construtor de linha para construir linhas arbitrárias de tal modo que uma instrução de linha de selecionar valor (e1, e2, ...) é equivalente exatamente à instrução selecionar e1, e2, ...

25 19. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de uma expressão de consulta que generaliza correlações esquerdas na cláusula a partir de e trata as mesmas uniformemente.

20. Sistema de computador, de acordo com a reivindicação 10, **CHARACTERIZADO** pelo fato de que a linguagem de consulta requer que todas as referências de coluna sejam

qualificadas com uma alias de tabela.

21. Sistema de computador, de acordo com a reivindicação 10, **CARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de um atalho sintático para manipular navegação através de uma coleção, em que o atalho utiliza o operador ... para permitir projeção de uma expressão a partir de uma coleção, de tal modo que "a..b" é equivalente a "select value t.b from a as t".

22. Sistema de computador, de acordo com a reivindicação 10, **CARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de aliasing de chaves de agrupar por, e correlações esquerdas entre tais chaves.

23. Sistema de computador, de acordo com a reivindicação 10, **CARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de dois tipos de agregados, um agregado à base de coleção que opera em coleções, e produz um resultado agregado, e um agregado estilo Linguagem de Consulta estruturada (SQL), em que o mecanismo converte implicitamente agregados do estilo SQL em agregados à base de coleção.

24. Sistema de computador, de acordo com a reivindicação 10, **CARACTERIZADO** pelo fato de que a linguagem de consulta permite ainda o uso de uma instrução INSERT..VALUES, e a instrução não permite que uma lista de coluna seja especificada na instrução INSERT..VALUES.

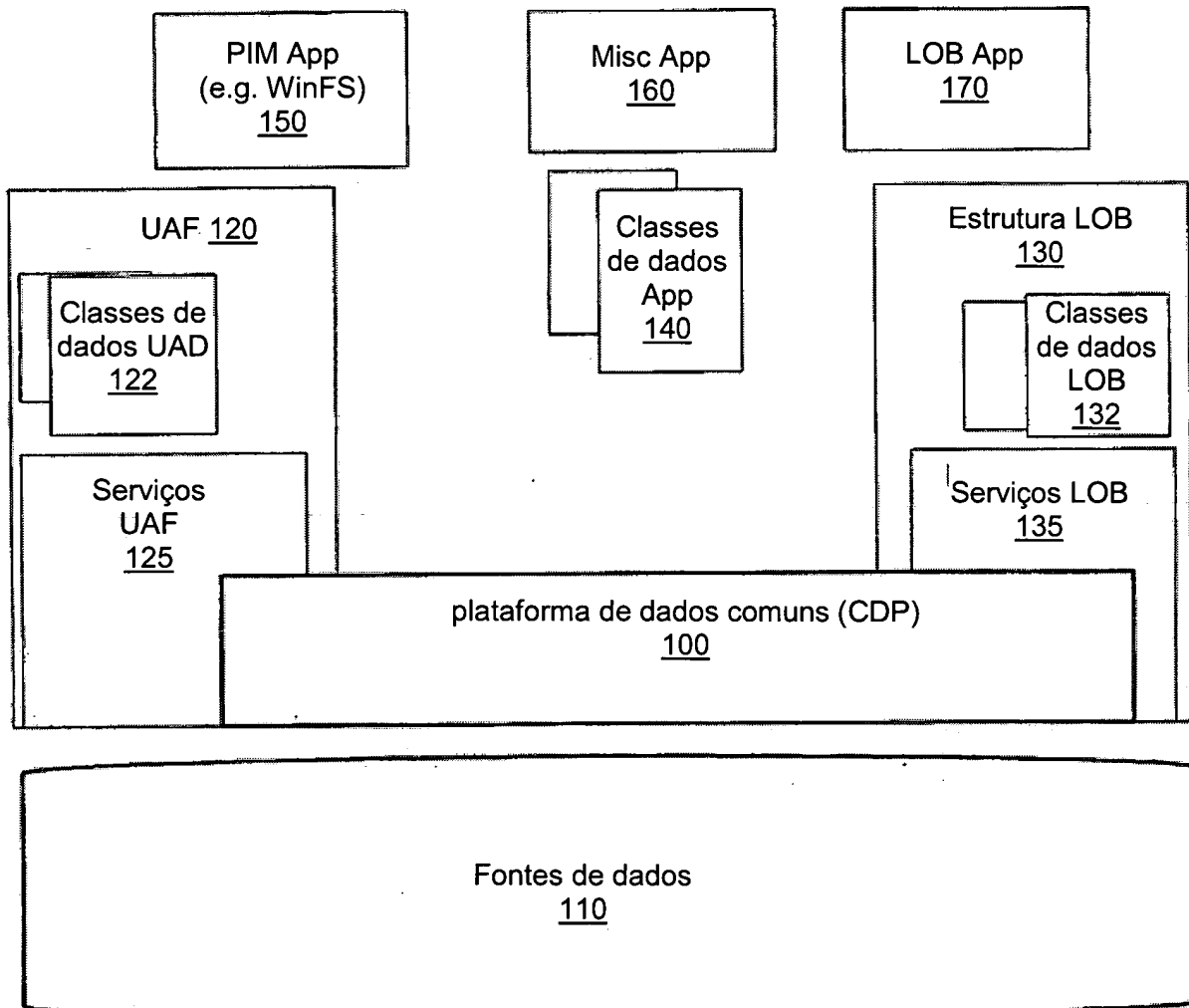


Fig. 1

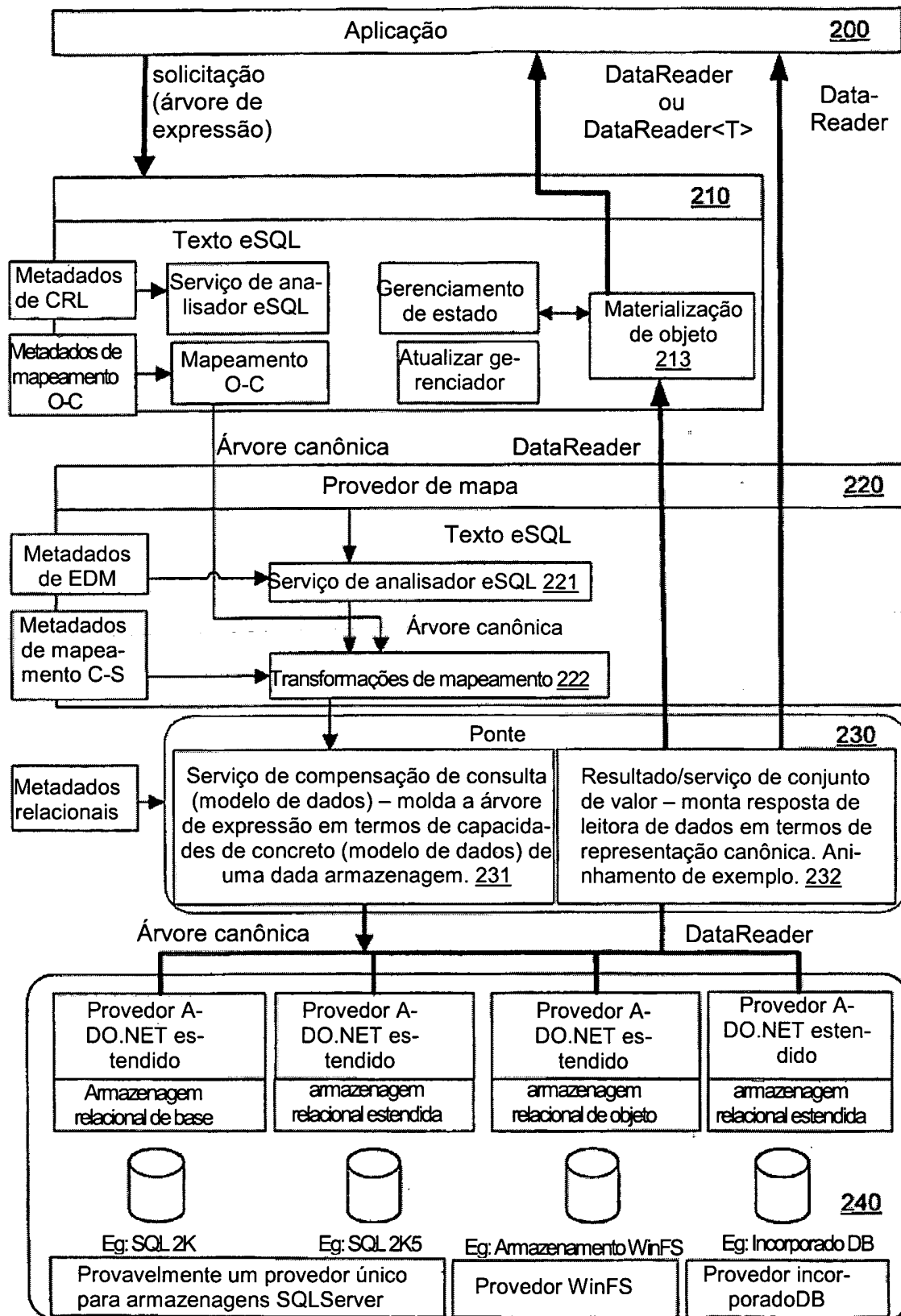


Fig. 2

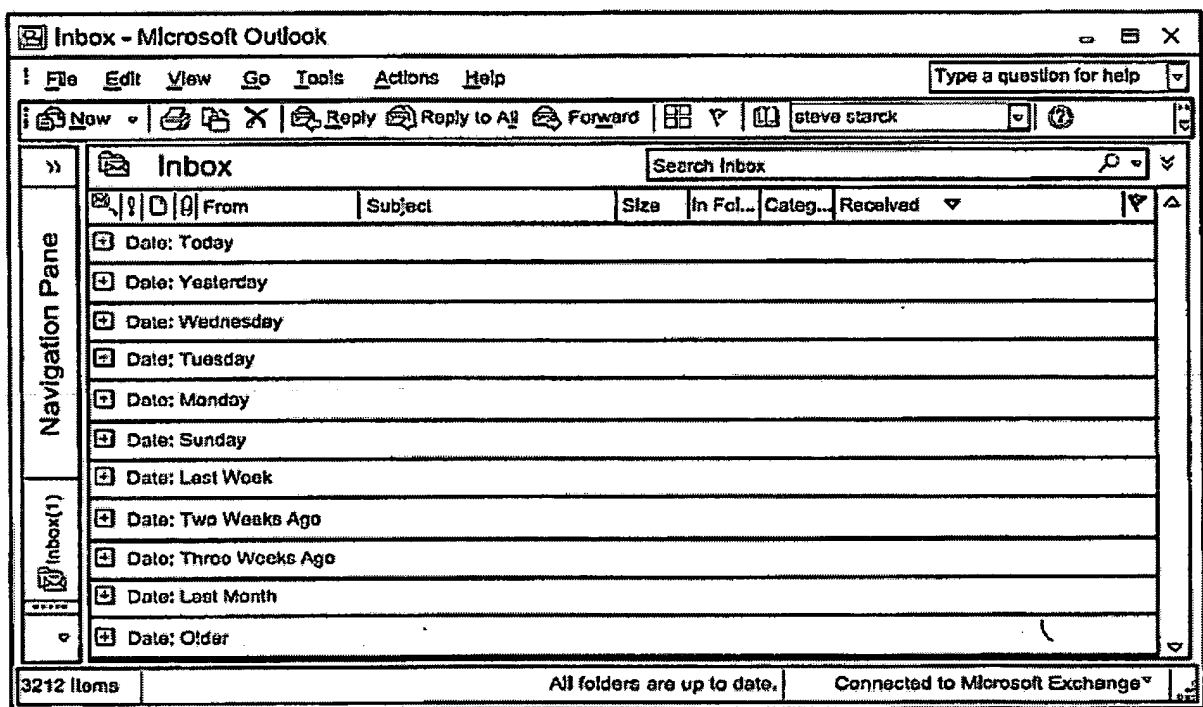


Fig. 3

PI 07083 24-2

RT / 15 2007 / 001 008

RESUMO

"LINGUAGEM DE CONSULTA EXTENSÍVEL COM SUPORTE PARA TIPOS DE DADOS RICOS"

É fornecida uma linguagem de consulta com suporte para tipos de dados ricos. Um sistema pode compreender uma aplicação configurada para gerar consultas de acordo com a linguagem de consulta. Também é fornecida uma plataforma para tratar de consultas que são geradas de acordo com a linguagem de consulta fornecida. A plataforma compreende genericamente uma camada de serviços de objeto, uma camada de provedor de mapa, e uma camada de ponte que pode realizar interface finalmente com uma variedade de bancos de dados comercialmente disponíveis.