MS-SQL Server, MySQL, Oracle e PostgreSQL

Linguagens Procedurais

O PostgreSQL permite que as funções definidas pelo usuário sejam escritas em outras linguagens além de SQL e C. Estas linguagens são chamadas genericamente de linguagens procedurais (PLs). No caso de uma função escrita em uma linguagem procedural, o servidor de banco de dados não possui nenhum conhecimento interno sobre como interpretar o texto do código fonte da função. Em vez disso, a tarefa é passada para um tratador especial que conhece os detalhes da linguagem. O tratador pode fazer todo o trabalho de análise gramatical e sintática, execução, etc., por si próprio, ou pode servir como um "elo de ligação" entre o PostgreSQL e a implementação existente de uma linguagem de programação. O tratador em si é uma função escrita na linguagem C, compilada como um objeto compartilhado, e carregado conforme necessário, como qualquer outra função escrita na linguagem C.

Atualmente existem quatro linguagens procedurais disponíveis na distribuição padrão Postgre SQL: PL/pgSQL (Capítulo 35), PL/Tcl (Capítulo 36), PL/Perl (Capítulo 37) e PL/Python (Capítulo 38). Os usuários podem definir outras linguagens. Os princípios básicos para o desenvolvimento de uma nova linguagem procedural estão descritos no Capítulo 46.

Estão disponíveis outras linguagens procedurais adicionais, mas não são incluídas na distribuição núcleo. O Apêndice H contém informações sobre como encontrá-las.

Instalação De Linguagem Procedural

A linguagem procedural deve ser "instalada" em cada banco de dados onde vai ser utilizada. Porém, as linguagens procedurais instaladas no banco de dados template1 ficam disponíveis automaticamente em todos os bancos de dados criados após sua instalação, uma vez que suas entradas em template1 são copiadas pelo comando CREATE DATABASE. Portanto, o administrador de banco de dados pode decidir quais linguagens ficarão disponíveis em quais bancos de dados, e pode tornar algumas linguagens disponíveis por padrão se assim o decidir.

Para as linguagens fornecidas na distribuição padrão, pode ser utilizado o programa createlang para instalar a linguagem em vez de executar os passos manualmente. Por exemplo, para instalar a linguagem PL/pgSQL no banco de dados template1 é utilizado:

O procedimento manual descrito abaixo somente é recomendado para a instalação de linguagens personalizadas que o programa createlang desconhece.

Instalação Manual de Linguagem Procedural

A linguagem procedural é instalada no banco de dados em quatro passos, que devem ser efetuados por um superusuário do banco de dados. O programa createlang automatiza tudo menos o passo 1.

O objeto compartilhado contendo o tratador da linguagem deve ser compilado e instalado em um diretório de biblioteca apropriado. Funciona da mesma maneira que a construção e instalação de módulos contendo funções C regulares definidas pelo usuário; consulte a Seção 31.9.6. Geralmente o tratador da linguagem depende de uma biblioteca externa que disponibiliza o mecanismo verdadeiro da linguagem de programação; se for assim, esta biblioteca também deve ser instalada.

O tratador deve ser declarado pelo comando:

CREATE FUNCTION nome_da_função_tratadora()

RETURNS language_handler

AS 'caminho_para_o_objeto_compartilhado'

LANGUAGE C;

O tipo especial retornado language_handler informa ao sistema de banco de dados que a função não retorna um dos tipos de dado SQL definidos, e que não pode ser utilizada diretamente nas declarações SQL.

Opcionalmente o tratador da linguagem pode disponibilizar uma função "validadora" para verificar se a definição da função está correta, sem na verdade executá-la. Caso exista, a função validadora é chamada pelo comando CREATE FUNCTION. Se o tratador disponibilizar uma função validadora, esta deve ser declarada por um comando como:

CREATE FUNCTION nome_da_função_validadora(oid)

RETURNS void

AS 'caminho_para_o_objeto_compartilhado'

LANGUAGE C;

A linguagem procedural deve ser declarada pelo comando:

CREATE [TRUSTED] [PROCEDURAL] LANGUAGE nome_da_linguagem

HANDLER nome_da_função_tratadora

[VALIDATOR nome_da_função_validadora] ;

A palavra opcional TRUSTED (confiável) especifica que é permitido aos usuários comuns do banco de dados, que não possuem privilégio de superusuário, utilizarem esta linguagem para criar procedimentos de funções e gatilhos.

Uma vez que as funções na linguagem procedural são executadas dentro do servidor de banco de dados, o sinalizador TRUSTED somente deve ser especificado para as linguagens que não permitem acesso às funcionalidades internas do servidor de banco de dados, nem ao sistema de arquivos.

As linguagens PL/pgSQL, PL/Tcl e PL/Perl são consideradas confiáveis; as lingua-gens PL/TclU, PL/PerlU e PL/PythonU foram projetadas para fornecer funcionalidades ilimita-das, não devendo ser marcadas como confiáveis.

O Exemplo 34-1 mostra como funciona o procedimento de instalação manual com a linguagem PL/pgSQL.

Instalação manual do PL/pgSQL

O comando abaixo informa ao servidor de banco de dados onde encontrar o objeto compartilhado da função tratadora de chamadas da linguagem PL/pgSQL:

CREATE FUNCTION plpgsql_call_handler() RETURNS language_handler AS

'$libdir/plpgsql' LANGUAGE C;

A linguagem PL/pgSQL possui uma função validadora, portanto esta também é declarada:

CREATE FUNCTION plpgsql_validator(oid) RETURNS void AS

'$libdir/plpgsql' LANGUAGE C;

Depois o comando

CREATE TRUSTED PROCEDURAL LANGUAGE plpgsql

HANDLER plpgsql_call_handler

VALIDATOR plpgsql_validator;

define que a função declarada anteriormente deve ser chamada para os procedimentos de função e gatilho onde o atributo de linguagem for plpgsql.

Na instalação padrão do PostgreSQL o tratador para a linguagem PL/pgSQL é construído e instalado no diretório de "biblioteca". Se o suporte à linguagem Tcl estiver configurado, os tratadores pa-ra PL/Tcl

e PL/TclU também serão construídos e instalados no mesmo local. Da mesma maneira, os tratadores para PL/Perl e PL/PerlU serão construídos e instalados se o suporte à linguagem Perl estiver configurado, e PL/PythonU será instalado se o suporte à linguagem Python estiver configurado.

Linguagem SQL

Linguagem de Manipulação, Consulta e Controle de Dados.

Histórico SQL

A linguagem SQL foi desenvolvida pela IBM no final dos anos 70, tendo sido a linguagem adotada em seu protótipo de banco de dados relacional denominado System R. O primeiro banco de dados comercial a utilizar a linguagem SQL foi apresentado em 1979, pela Oracle.

No início dos anos 80, o Instituto Americano de Padronização (ANSI – American National Standards Institute) iniciou os trabalhos de desenvolvimento de uma versão padronizada do SQL, a qual foi publicada nos anos de 1986 e 1987. Posteriormente outras revisões e acréscimos foram efetuados no SQL que sofreu importantes melhorias.

Como ela é padronizada pelo ANSI, assim como acontece, por exemplo com as linguagens C, C++ e Pascal, diversos fabricantes de bancos de dados sentiram-se seguros em adota-la como sendo a linguagem padrão para seus bancos de dados.

Outro fator, talvez mais significativo que o anterior e que levou à adoção do padrão SQL por diversos bancos de dados, foi o fato de que essa linguagem não é procedural. Isso pode ser traduzido da seguinte maneira: “não importa como uma tarefa vai ser executada pelo banco de dados; importa apenas que ela vai ser executada”. Em breve veremos as estruturas básicas do SQL, e certamente aqueles que nunca tiveram contato com ela irão se surpreender com as facilidades apresentadas.

Bancos de dados, como Access, SQL Server, Oracle, DB2 e Paradox, utilizam o SQL. Embora boa parte do SQL padrão ANSI seja utilizada por esses bancos, os programadores encontrarão algumas variações e uma série de acréscimos ao padrão desses bancos de dados.

Uma das grandes vantagens do SQL é que é uma linguagem universal para os bancos de dados profissionais. Uma boa parte dessa linguagem está padronizada. Evidentemente existem alguns acréscimos conforme o banco de dados utilizado (MSSQL Server, Oracle, DB2, etc). Entretanto, a base é a mesma.

Exemplo

Uma outra vantagem é que essas instruções são executadas pelo banco de dados e não pelo programa. Vamos imaginar o método sem SQL. Tendo uma lista com 80.000 nomes, desejo saber se existe um cliente como o nome de João Gomes Araújo. O usual seria verificar registro por registro, testando o nome de cada cliente cadastrado. Uma alternativa para otimizar a performance é dividir o banco de dados conforme as letras do alfabeto. Assim sendo, poderíamos ter uma tabela com os nomes iniciados em A, B, C,D outra com os nomes iniciados com E,F,G e assim por diante.

De qualquer forma, o programa teria que abrir o arquivo, ler registro por registro até receber o resultado. Então cabe uma pergunta: se ele encontrar o nome desejado, ele pára ou verifica se existem mais clientes com o mesmo nome? E se desejarmos mandar uma mala direta apenas para os clientes de uma determinada cidade, ou para aqueles que fazem aniversário em um determinado dia do mês.

Devemos abrir uma ou mais tabelas e ir filtrando registro por registro.

Bom, se você não tem pressa ou se seu banco de dados é pequeno (uns 500 registros no máximo) tudo bem.

Entretanto, pense na seguinte situação: um diretor deseja saber AGORA todos os clientes que moram no Recife e que tenham entre 25 e 35 anos, com renda mensal superior a R$ 1.500,00, com nível de instrução superior e que tenham comprado algum produto nos últimos três meses.

Se você for programar isso em código, talvez eles comecem a pensar em outro programador para ocupar seu lugar. Entretanto se você usar um banco de dados compatível com SQL bastará montar uma consulta para obter esse resultado no ato.

Isso porque as consultas (ou queries) podem ser escritas e testadas sem a necessidade de compilação. Você faz isso diretamente no servidor ou mesmo em uma estação. Simples, rápido e fácil.

Definição

O SQL é uma linguagem padrão, especificamente concebida para permitir que as pessoas a criem Bancos de Dados, adicionem novos dados a essas bases, manipulem os dados, e recuperem partes selecionadas dos dados.

Existem basicamente duas formas de manipulação de dados usando o software de banco de dados. Uma abordagem é interagir diretamente com o SGBD usando uma linguagem especial chamada linguagem de consulta. Na segunda abordagem, o utilizadorinterage com o programa de aplicação. O programa de aplicação envia instruções para o SGBD, que então executa as ações especificadas pelo programa. Esta lição irá se concentrar em usar linguagens de consulta para executar tarefas de processamento de dados.

Depois de ler esta lição, você deve ser capaz de:

Definir a linguagem de consulta de dados.

Descreva um exemplo de uma linguagem de consulta.

Discutir as funções e capacidades de uma linguagem de consulta.

Query Language (linguagem de consulta)

A Linguagem de consulta permite ao usuário interagir diretamente com o software de banco de dados, a fim de executar as tarefas de processamento de informações, usando dados em um banco de dados. É normalmente uma linguagem de computador fácil de usar, que se baseia em palavras básicas, tais como SELECT, DELETE ou ALTER. Usando linguagem de consulta e um teclado de computador, o usuário digita comandos que instruem o SGBD para recuperar dados de uma base de dados ou A atualização de um banco de dados.

Structured Query Language (SQL - linguagem de consulta estruturada)

É um tipo de linguagem de consulta que é amplamente utilizada para executar operações usando bancos de dados relacionais. Lembre-se que os bancos de dados relacionais são compostos por tabelas com linhas e colunas. A Linguagem SQL pode ser usado para recuperar informações de tabelas relacionadas em um banco de dados ou para selecionar e recuperar informações de linhas e colunas específicas em uma ou mais tabelas.

Uma das chaves para a compreensão de como funciona o SQL em um banco de dados relacional é perceber que cada tabela e coluna tem um nome específico associado a ele. Para consultar uma tabela, o usuário especifica o nome da tabela (indicando as linhas a ser exibido) e os nomes das colunas a serem exibidas.

Uma consulta SQL típica contém três elementos-chave:

SELECT (os nomes das colunas a serem exibidos)

FROM (indica o nome da tabela da qual os nomes das colunas será derivado)

WHERE (descreve a condição para a consulta)

Para ilustrar a aplicação deste tipo de consulta, vamos supor que um determinado usuário deseje consultar um banco de dados relacional contendo informações sobre os doadores para uma organização de caridade. Se o usuário quiser saber o nome e endereço de todos os indivíduos que doam US$ 100 ou mais, a seguinte consulta poderá ser usada:

SELECT Nome, Endereço

FROM Lista-de-Doadores

WHERE Doacao > 100

Uma vez que este comando foi executado, o computador irá exibir uma lista de doadores que atenda aos critérios pré-definidos. Neste caso, todos os dados são extraídos a partir de uma única tabela. Consultas semelhantes podem ser feitas para extrair dados de várias tabelas. Tal estratégia pode ser usada para analisar as informações do cliente, envolvendo dados de faturamento e dados de ordem, usando duas tabelas separadas.

Outras capacidades e linguagens de consulta

SQL tem muitas outras capacidades, uma das quais é a de ser capaz de atualizar e revisar um banco de dados relacional. Os usuários podem descobrir a necessidade de adicionar ou excluir colunas em um banco de dados.

Outros tipos de linguagens de consulta também estão disponíveis para manipulação de dados em bancos de dados relacional. Outro exemplo popular é chamado de consulta por exemplo (QBE). Esta linguagem utiliza uma abordagem gráfica e padrões de grade para permitir ao usuário especificar os dados a serem exibidos. SQL QBE não pode ser usado com Bancos de Dados hierárquicos e rede. Linguagens de consulta exclusivas foram projetados especificamente para esses bancos de dados.

O que pode fazer o SQL?

SQL pode executar consultas em um banco de dados

SQL pode selecionar e recuperar dados a partir de um banco de dados

SQL pode inserir registros em um banco de dados

SQL pode atualizar registros em um banco de dados

SQL pode excluir registros de um banco de dados

SQL pode criar novos Bancos de Dados

SQL pode criar novas tabelas em um banco de dados

SQL pode criar procedimentos armazenados (stored procedures) em um banco de dados

SQL pode criar visões (views) em um banco de dados

SQL pode definir permissões em tabelas, procedimentos, e visões

SQL é um padrão

Embora o SQL seja um padrão ANSI (American National Standards Institute), há muitas versões diferentes da linguagem SQL. No entanto, para ser compatível com o padrão ANSI, a linguagem deve possuir pelo menos, os comandos principais (como SELECT, UPDATE, DELETE, INSERT) de maneira semelhante.

Nota: A maioria dos programas de banco de dados SQL também têm suas próprias extensões proprietárias, além do padrão SQL!

Tenha em mente que...

SQL não é case sensitive (não é sensível a maiúsculas)

Alguns sistemas de banco de dados requerem um ponto e vírgula no final de cada instrução SQL.

DML, DDL e DCL

A linguagem SQL pode ser dividida em três partes:

DDL - linguagem de definição de dados (Data Definition Language) permite a criação, eliminação e alteração da estrutura fisica da Base de Dados. Ela também define os índices (chaves), especifica as ligações entre as tabelas, e impõe restrições entre tabelas. As declarações mais importantes DDL em SQL são:

CREATE DATABASE - cria um novo banco de dados

ALTER DATABASE - altera um banco de dados

CREATE TABLE - cria uma nova tabela

ALTER TABLE - modifica uma tabela

DROP TABLE - apaga uma tabela

CREATE INDEX - cria um índice (chave de busca)

DROP INDEX - exclui um índice

DML - linguagem de manipulação de dados (Data Manipulation Language) permite a Inserção, edição e exclusão dos dados das tabelas. A consulta e os comandos de atualização formam a parte DML de SQL são:

SELECT - extrai dados de um banco de dados

UPDATE - atualiza os dados em um banco de dados

DELETE - exclui dados de um banco de dados

INSERT - insere novos dados em um banco de dados

DCL - Linguagem de Controle de Dados (Data Control Language) atribui permissões aos objetos através dos comandos Grant, revoke e Deny. A Linguagem de Controle de Dados (DCL) é um subconjunto do Lanaguge SQL (Structured Query) que permite que os administradores de banco de dados para configurar o acesso de segurança para bancos de dados relacionais. DCL é o mais simples dos subconjuntos SQL, já que consiste em apenas três comandos: GRANT, REVOKE e DENY. Combinados, esses três comandos oferecem aos administradores a flexibilidade para definir e remover permissões de banco de dados de forma extremamente granular.

GRANT é utilizado para adicionar novas permissões para um usuário

REVOKE é utilizado para remover o acesso de banco de dados de um usuário

DENY é utilizado para impedir explicitamente que um usuário receba uma permissão especial

Lição de encerramento

Usuários confiam na linguagem de consulta para manipular dados em um banco de dados. O SQL, por exemplo, é largamente utilizadopara gerir Bancos de Dados relacionais. É importante entender o propósito de linguagens de consulta. Talvez um dia, você terá a oportunidade de usar essas línguagem para ajudar sua organização a manipular e extrair o significado de um dos seus muitos bancos de dados.

Agora que você completou esta lição, você deve ser capaz de:

Definir a linguagem de consulta.

Descrever um exemplo de uma linguagem de consulta estruturada.

Discutir as funções e capacidades de uma linguagem de consulta estruturada.

Tipos de Dados

Antes de criar uma tabela você precisa entender as diferenças entre os tipos de dados que você pode utilizar em uma coluna. Os tipos de dados SQL se classificam em 13 tipos de dados primários e de vários sinônimos válidos reconhecidos por tais tipos de dados. Os tipos de dados primários são:

Tipo de Dados	Longitude	Descrição
BINARY	1 byte	Para consultas sobre tabela anexa de produtos de banco de dados que definem um tipo de dados Binário.
BIT	1 byte	Valores Sim/Não ou True/False
BYTE	1 byte	Um valor inteiro entre 0 e 255.
COUNTER	4 bytes	Um número incrementado automaticamente (de tipo Long)
CURRENCY	8 bytes	Um inteiro escalável entre 922.337.203.685.477,5808 e 922.337.203.685.477,5807.
DATETIME	8 bytes	Um valor de data ou hora entre os anos 100 e 9999.
SINGLE	4 bytes	Um valor em ponto flutuante de precisão simples com uma classificação de - 3.402823*1038 a -1.401298*10-45 para valores negativos, 1.401298*10- 45 a 3.402823*1038 para valores positivos, e 0.
DOUBLE	8 bytes	Um valor em ponto flutuante de dupla precisão com uma classificação de - 1.79769313486232*10308 a -4.94065645841247*10-324 para valores negativos, 4.94065645841247*10-324 a 1.79769313486232*10308 para valores positivos, e 0.
SHORT	2 bytes	Um inteiro curto entre -32,768 e 32,767.
LONG	4 bytes	Um inteiro longo entre -2,147,483,648 e 2,147,483,647.
LONGTEXT	1 byte por caractere	De zero a um máximo de 1.2 gigabytes.
LONGBINARY	Segundo se necessite	De zero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por caractere	De zero a 255 caracteres.

A seguinte tabela recolhe os sinônimos dos tipos de dados definidos:

Tipo de Dado	Sinônimos	Descrição
BINARY	VARBINARY	Varbinary – dados binários com tamanho variável até 8 Kb
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO	
BYTE	INTEGER1	
COUNTER	AUTOINCREMENT	
CURRENCY	MONEY	Money - ± 922.337.203.685.447,5808 – 8 bytes

DATETIME	DATE TIME TIMESTAMP	Date– 1/1/1753 a 31/12/9999 – 8 bytes Timestamp – cria um valor único gerado pelo SQL Server, automaticamente atualizado em caso de inclusão ou alteração de uma linha.
SINGLE	FLOAT4 IEEESINGLE REAL	
DOUBLE	FLOAT FLOAT8 IEEEDOUBLE NUMBER NUMERIC	
SHORT	INTEGER2 SMALLINT	Smallint - ± 32.767 – 2 bytes
LONG	INT INTEGER INTEGER4	
LONGBINARY	GENERAL OLEOBJECT	
LONGTEXT	LONGCHAR MEMO NOTE	
TEXT	ALPHANUMERIC CHAR - CHARACTER STRING - VARCHAR	Char – para carácteres com tamanho fixo até 8 Kb Varchar – para carácteres com tamanho variável até 8 Kb

Veja no quadro abaixo o melhor tipo de dados para os exemplos

Nome da coluna	Descrição	Exemplo	Melhor tipo de dado
preco	Preço de um item a venda	5678.39	Money
cep	5 a 10 carácteres	11015-420	VARCHAR(10)
massa	Peso atômico de um elemento	4.066054	FLOAT
comentarios	Bloco de texto gigante	Joe, I'm at the shareholder's meeting. They just gave a demo and there were rubber duckies flying around the screen. Was this your idea of a joke? You might want to spend some time on Monster.com....	LONGCHAR
quantidade	Itens em estoque	239	INT
taxa	Uma porcentagem	3.755	FLOAT

titulo	Um texto	Guia de bolso do administrador	VARCHAR(50)
sexo	Um carácter M ou F	M	CHAR(1)
telefone	Dez dígitos sem pontuação	32027100	CHAR(10)
estado	Dois carácteres	SP	CHAR(2)
aniversário	Dia, mês e ano	28/10/1973	DATE

Cada SGBD possui variantes e particularidades em seus tipos de dados. Para maiores informações sobre os tipos de dados nos diferentes SGBDs clique nos links abaixo:

MS SQL Server

MySQL

Linguagem De Manipulação De Dados

Linguagem de Manipulação de Dados (ou DML, de Data Manipulation Language) é uma família de linguagens de computador utilizada para a recuperação, inclusão, remoção e modificação de informações em bancos de dados. Pode ser procedural, que especifica como os dados devem ser obtidos do banco; pode também ser declarativa (não procedural), em que os usuários não necessitam especificar o caminho de acesso, isto é, como os dados serão obtidos. O padrão SQL é não procedural. DMLs foram utilizadas inicialmente apenas por programas de computador, porém (com o surgimento da SQL) também têm sido utilizadas por pessoas.

Principais Comandos

As DMLs têm sua capacidade funcional organizada pela palavra inicial em uma declaração, a qual é quase sempre um verbo. No caso da SQL, estes verbos são:

Select

Insert

Update

Delete

Cada declaração SQL é um comando declarativo. As declarações individuais da SQL são declarativas, em oposição às imperativas, na qual descrevem o quê o programa deveria realizar, em vez de descrever como ele deveria realizar. Muitas implementações de banco de dados SQL estendem suas capacidades SQL fornecendo linguagens imperativas, isto é, procedurais. Exemplos destas implementações são o PL/SQL, da Oracle, e o SQL PL, da DB2. As linguagens de manipulação de dados tendem a ter muitos tipos diferentes e capacidades entre distribuidores de banco de dados. Há um padrão estabelecido para a SQL pela ANSI, porém os distribuidores ainda fornecem suas próprias extensões ao padrão enquanto não implementam o padrão por completo.

Há dois tipos de DMLs:

Procedural

Declarativa

Controle de Concorrência em Banco de Dados

Controle de concorrência é um método usado para garantir que as transações sejam executadas de uma forma segura e sigam as regras ACID. Os SGBD devem ser capazes de assegurar que nenhuma ação de transações completadas com sucesso (committed transactions) seja perdida ao desfazer transações abortadas (rollback).

Uma transação é uma unidade que preserva consistência. Requeremos, portanto, que qualquer escalonamento produzido ao se processar um conjunto de transações concorrentemente seja computacionalmente equivalente a um escalonamento produzindo executando essas transações serialmente em alguma ordem. Diz-se que um sistema que garante esta propriedade assegura a seriabilidade.

Técnicas de locking podem ser utilizadas para garantir seriabilidade e recuperabilidade nos escalonamentos das transações do banco de dados. Um escalonamento conflito-serializável é um escalonamento equivalente a alguma execução serial das transações.

Propriedades ACID de Transações

Vamos revisar aqui rapidamente os conceitos de ACID, que pode ser visto neste artigo.

Atomicidade: A execução de uma transação deve ser atômica, ou todas as ações são executadas, ou nenhuma é;

Consistência: Cada transação executada isoladamente deve preservar a consistência do banco de dados;

Isolamento: Cada transação deve ser isolada dos efeitos da execução concorrente de outras transações;

Durabilidade: Toda transação que for finalizada de forma bem-sucedida deve persistir seus resultados em banco mesmo na presença de falhas no sistema.

Consistência e Isolamento

É o usuário quem deve garantir a consistência da transação. Exemplo: Transferência de fundos entre contas não alteram a quantia total de dinheiro nas contas.

O isolamento deve garantir que duas transações, executadas concorrentemente, devem ter o mesmo resultado se executadas em ordem serial. Exemplo: T1 concorrente com T2 T1, T2 ou T2, T1.

Atomicidade das Transações

As falhas em uma transação podem ter como motivo:

Interrupção do SGBD;

Queda do sistema;

Detecção de uma situação inesperada.

Uma transação interrompida ao meio pode deixar o banco de dados em um estado inconsistente. O banco de dados deve prover recursos para remoção dos efeitos de transações incompletas para garantir a atomicidade.

O SGBD mantém um registro (log) das ações executadas pelo usuário para que estas possam ser desfeitas caso ocorra alguma falha em uma transação. O log também é utilizado para garantir a durabilidade. Se ocorrer queda do sistema antes que todas as mudanças tenham sido feitas em disco, o log é usado para restaurar o estado do banco de dados quando o sistema for reiniciado.

Escalonamento de Transações

Um escalonamento é uma lista de ações de um conjunto de transações. Representa uma sequência de execução que deve conservar a mesma ordem de execução das ações das transações presentes nele.

Escalonamento completo: O escalonamento insere, para todas as transações, as ações de abort ou commit;

Escalonamento serial: O escalonamento não intercala as ações de transações diferentes;

Escalonamento equivalente: Para qualquer estado da base de dados, o efeito de executar um primeiro escalonamento é idêntico ao efeito de executar um segundo escalonamento;

Escalonamento serializável: Escalonamento equivalente a alguma execução serial das transações.

Se cada transação preserva a consistência, todo escalonamento serializável preserva a consistência.

Recuperação de Falhas

O Gerenciador de Recuperação de Falhas garante a atomicidade e a durabilidade das transações.

Atomicidade: Desfazendo as ações das transações que não realizaram o commit.

Durabilidade: Todas as ações das transações que fizeram commit serão persistentes.

O Gerenciador é responsável por recuperar a consistência, após uma queda do sistema.

Há 3 fases no algoritmo de recuperação de ARIES:

Analisar: Percorre o log para frente (a partir do ponto de verificação mais recente) para identificar todas as transações que estavam ativas, e todas as “páginas sujas” no momento da falha.

Refazer: Refaz todas as atualizações das “páginas sujas”, quando necessário, para garantir que todas as atualizações registradas no log foram realizadas e escritas no disco.

Desfazer: As escritas de todas as transações que estavam ativas no momento da falha são desfeitas (recuperando o valor anterior à atualização registrado no log da atualização), varrendo o log de trás para frente.

Falhas no Sistema

Dada a complexidade dos equipamentos e programas modernos, é ponto pacífico que falhas ocorrerão, quer sejam problemas de “hardware”, quer sejam defeitos de “software”. Estas falhas têm como efeito indesejável comprometer a integridade do BD. Para que seja de alguma utilidade prática, O SGBD deve, portanto, incorporar mecanismos que garantam sua integridade, quando não pelo menos na presença daquelas falhas que ocorrem com mais frequência. Desta forma, o SGBD pode ser mantido em operação por longos períodos de tempo sendo, quando muito, interrompido por curtos intervalos para que os mecanismos de controle de integridade sanem inconsistências causadas por eventuais falhas.

Esta seção sevirá de introdução à área de controle de integridade em SGBD, onde os principais problemas e as soluções mais importantes serão mencionadas de forma simplificada. Em capítulo posterior, esta problemática será analisada em maiores detalhes.

Intuitivamente, a única maneira do SGBD se proteger contra falhas, que podem destruir parte dos dados, é criar e manter certa redundância no sistema.

Desta forma, quando parte do BD e danificado, sua “cópia redundante” pode ser revivida para recuperar os dados perdidos e restabelecer a operação normal. Inclusive, em sistemas que requeiram alta confiabilidade, as partes mais críticas do próprio “hardware” podem ser duplicadas de forma redundante.

Note que, se a “cópia” de um objeto não é recente, então deve-se manter também um histórico das operações efetuadas sobre este objeto, de tal modo que o SGBD possa refazer estas operações e trazer esta “cópia” ao estado mais recente, idêntico àquele da “cópia” original antes da falha. Caso contrário, transações executadas entre o instante de criação da “cópia” e o momento atual serão perdidas.

Tipos De Falhas

Um nó qualquer, quando em operação normal, depende de um padrão de interligações complexas entre vários elementos. Para efeito de examinar a ocorrência e danos causados por falhas nestes componentes é conveniente agrupá-los da seguinte forma

Procedimentos;

Processadores;

Memórias;

No caso distribuído, comunicação de dados;

Por procedimentos entende-se a totalidade dos módulos e programas aplicativos (“software”) que compõem o SGBD, podendo-se incluir aqui também os utilitários do sistema operacional usados pelo SGBD. Os processadores correspondem tanto ao processador, ou processadores, central como as demais unidades de controle de periféricos, terminais, “modems”, etc.

As memórias, onde reside o BD, aqui entendido como dados mais programas, são de crucial importância. É lá que será acomodada toda redundância introduzida para fins de controle de integridade. Todos os mecanismos de proteção contra falhas prestam especial atenção ao tratamento dispensado aos vários tipos de memórias com que o sistema interage e, em última análise, se fiarão na boa característica de resistência a falhas que tais elementos oferecem. Para que se possa conduzir uma análise mais detalhada, as memórias manipuladas pelo sistema serão subdivididas em:

Memória principal

Memória secundária imediatamente disponível

Memória secundária dormente

A memória principal corresponde a memória associada aos processadores, isto é, memória dos processadores centrais, memórias tipo cache, “buffers” de entrada/saída, espaço de paginação, etc. Há certos tipos de falhas às quais o conteúdo da memória principal não sobrevive, devendo ser considerado como irremediavelmente perdido. Estes defeitos serão cognominados de falhas primárias. Interrupção no fornecimento de energia elétrica, defeito nos processadores ou procedimentos do sistema podem causar este tipo de falha. Por não sobreviver a este tipo de falha mais comum, diz-se que a memória principal é volátil.

O termo memória secundária imediatamente disponível, ou memória secundária ativa, referem-se à memória de massa, geralmente discos magnéticos, onde o BD é residente e que está a disposição do SGBD a todo instante.

O conteúdo da memória secundária ativa não é afetado por falhas primárias que o sistema venha a sofrer. Porém, panes nos cabeçotes de leitura/escrita dos discos ou partículas de poeira que assentem sobre a superfície dos mesmos podem danificar os delicados mecanismos dos cabeçotes e provocar danos irrecuperáveis à superfície magnética destruindo, em todo ou em parte, o conteúdo da memória secundária ativa.

Isto se verificando, diz-se que o sistema sofreu uma falha secundária. Fitas magnéticas também podem ser usadas como memória secundária ativa, se bem que cuidados devem ser tomados para que a eficiência do sistema não seja por demais comprometida. Partes raramente usadas do BD, cópias antigas de parte ou da íntegra do sistema, além de aplicativos ativados com pouca frequência, são candidatos naturais a residirem em fita. Nunca dicionários, catálogos e outros elementos frequentemente acessados.

Como um último recurso, e em casos realmente catastróficos, o controle de integridade do SGBD pode apelar para a memória secundária dormente (“off-line”). Entende-se como memória secundária dormente toda memória fisicamente desconectada do sistema. Geralmente, devido à sua grande capacidade de armazenamento de dados, fitas magnéticas são empregadas para este fim. Em BD de grandes proporções, toda uma fitoteca pode ser necessária. É comum armazenar os componentes da memória secundária dormente em locais próprios, distantes do centro onde opera o sistema. A preocupação básica é evitar que catástrofes sobre um dos lugares, tais como incêndios ou furtos, não afete o outro. De qualquer maneira, eventos que destruam ou inutilizem o conteúdo da memória secundária dormente do sistema serão chamados de falhas terciárias.

Existem situações que exigem ações por parte do sistema de controle de integridade do BD, embora não se configurem propriamente como falhas em componentes do sistema.

O caso típico é quando, sob operação normal, surje a necessidade de se cancelar transações. Isto pode ocorrer tanto por erro ou a pedido do usuário, como podem ser ações forçadas pelo SGBD como última instância para evitar bloqueio na execução de transações que competem por certos recursos do sistema. Estes casos serão rotulados como pseudo-falhas do sistema.

Agora, não está em cheque o conteúdo de nenhuma das memórias ou a sanidade dos processadores ou procedimentos associados ao sistema.

A cooperação do controle de integridade, porém, é necessária para invocar a transação que inverte o efeito das ações elementares executadas em benefício da transação a ser cancelada. Deste modo, o BD permanece em um estado consistente, além do que é forçada a liberação dos recursos que foram sequestrados pela transação.

A discussão acima desloca-se a partir de falhas nos componentes mais nobres, isto é, com menor tempo de acesso, para os menos nobres, com tempos de acesso consideravelmente maiores. É importante ter em mente uma noção da frequência com que os vários tipos de falhas costumam ocorrer na prática, bem como do tempo necessário para que o controle de integridade restaure a operação normal do BD em cada caso.

Controle de Concorrência

Um SGBD, suportando bancos de dados com várias aplicações, deverá necessariamente permitir acesso concorrente aos dados. É intuitivo que, em tese, quanto maior for o nível de concorrência permitido, tanto melhor será o tempo de resposta do sistema como um todo. Em tese porque, forçosamente, os mecanismos que controlam o acesso concorrente ao banco impõem um ônus adicional sobre o desempenho do SGBD. Os procedimentos que harmonizam o paralelismo no seio do SGBD serão conhecidos por mecanismos de controle de concorrência.

Num cenário de BDs distribuídos, ou mesmo de BDs centralizados com acesso distribuído, a implementação de paralelismo torna-se uma necessidade imperiosa. Com relação ao caso centralizado, hoje são conhecidas técnicas que equacionam os problemas a contento, apoiadas em um tratamento teórico preciso e confirmadas por implementações reais. No que concerne ao caso distribuído, a situação é mais confusa. Isto é devido, em grande parte, ao fato de que os nós da rede operam de forma bastante independente, embora o controle de concorrência deva ser efetivado de forma global, abrangendo informação que pode estar espalhada por vários nós. Aqui, muitos dos aspectos do problema ainda se encontram em fase de pesquisa e experimentação.

Exemplificando O Problema

Quando transações manipulam dados concorrentemente, certos problemas, chamados de anomalias de sincronização, poderão ocorrer. Exemplos são acessos a dados inconsistentes, perdas de atualizações e perda da consistência do banco. Por exemplo, considere duas transações, T1 e T2, ambas debitando uma determinada quantia a um saldo S. Seja a sequência de ações:

T1 lê o saldo S;

T2 lê o saldo S;

T2 debita a quantia, escrevendo o novo valor de S;

T1 debita a quantia, escrevendo o novo valor de S;

O valor final do saldo neste caso refletirá apenas a quantia debitada por T1, sendo a atualização submetida por T2 perdida.

O exemplo acima também serve para ilustrar porque controle de concorrência, embora semelhante, não é equivalente ao dilema de gerenciar acesso a recursos partilhados em um sistema operacional. De fato, na sequência acima, cada transação respeita o princípio de acesso exclusivo ao objeto partilhado S. Porém, isto claramente não é suficiente pois uma atualização é perdida. Assim sendo, um mecanismo de controle de concorrência não deverá se limitar a implementar acesso exclusivo a objetos do banco de dados.

O problema fundamental a ser resolvido pelos métodos de controle de concorrência é colocado da seguinte forma. Assuma que todas as transações preservam a consistência lógica do banco de dados e terminam, quando executadas sequencialmente. Um método de controle de concorrência deverá, então, garantir que em toda execução concorrente das transações:

Cada transação termina;

Cada transação é executada sem interferência das outras, e sem que anomalias de sincronização ocorram.

Estes objetivos deverão ser atingidos permitindo-se um máximo de concorrência possível, de forma transparente para os usuários, e para qualquer conjunto de transações acessando qualquer parte do banco de dados.

Note que o controle de concorrência e a gerência de transações são tarefas que se complementam. Cabe ao gerente de transações escalonar as ações de uma transação de tal forma que esta seja processada corretamente, conforme a especificação do usuário. Por outro lado, cabe ao mecanismo de controle de concorrência arbitrar a intercalação das ações de transações diferentes de tal forma a que todas as transações terminem, sejam processadas sem que uma interfira com a outra e sem que anomalias de sincronização ocorram.

A idéia do critério de correção comumente aceito é bem simples. Seja T um conjunto de transações. Inicialmente observa-se que uma execução sequencial ou serial das transações em T, ou seja, uma execução em que as transações são processadas uma após a outra terminar, em uma ordem qualquer, é necessariamente correta. Isto é fácil ver pois em uma execução serial não há processamento concorrente.

O próximo passo é postular que uma execução concorrente E das transações em T será considerada correta se for computacionalmente equivalente a alguma execução serial das transações. A execução E será chamada neste caso de serializável. A noção de equivalência computacional usada aqui exige que o estado final do banco de dados seja o mesmo em E e S e que as transações leiam os mesmos dados em E e S (assumindo que E e S começam no mesmo estado inicial do banco de dados) e, portanto, executem a mesma computação em E e S.

A postulação deste critério de correção para execuções concorrentes é justificada pois, como S é serial, as transações em T são naturalmente executadas sem que uma interfira com a outra. É fácil justificar que anomalias de sincronização não ocorrem em S. Como E é computacionalmente equivalente a S, as transações são executadas em E sem que uma interfira com a outra e sem que anomalias de sincronização ocorram. Os próximos exemplos ilustrarão estes conceitos.

Suponha que P e C representem os saldos da poupança e da conta corrente de um determinado cliente (armazenadas em um banco de dados centralizado, por simplicidade). Considere duas transações cujos efeitos desejados são:

T1: se houver saldo suficiente na poupança, transfira $5.000,00 da poupança para a conta corrente;

T2: se houver saldo suficiente na conta corrente, debite um cheque de $10.000,00. Suponha que a sequência de ações elementares sobre o banco de dados gerada pela execução

da transação T1 (supondo que haja saldo suficiente) seja: T11: leia o saldo P para X;

X := X – 5.000; T12: leia o saldo C para Y;

Y := Y + 5.000; T13: escreva o novo saldo Y em C; T14: escreva o novo saldo X em P.

Apenas aquelas ações que acessam o banco de dados receberam rótulos pois são estas que nos interessarão a seguir. Suponha que a sequência de ações elementares sobre o banco de dados gerada pela execução da transação T2 (supondo que haja saldo suficiente) por sua vez seja:

T21: leia o saldo C para Z; Z := Z – 10.000;

T22: escreva o novo saldo Z em C.

Primeiro considere uma execução puramente serial em que T2 é processada antes de T1. Esta execução pode ser abstraída pela seguinte sequência de rótulos das operações sobre o banco de dados:

S = T21 T22 T11 T12 T13 T14

Considere agora uma execução concorrente abstraída pela seguinte sequência de rótulos:

L = T11 T21 T22 T12 T13 T14

Esta execução L não é serial, porém é serializável por ser equivalente a S. Isto é fácil de ver pois a ação T11 em L pode ser comutada com T21 e T22 sem que o resultado do processamento de T1 seja alterado e sem que o estado final do banco de dados seja modificado. De fato, T11 lê o saldo P enquanto as ações de T2 afetam apenas o saldo C. Logo, T11 lê o mesmo saldo P em L e em S.

Por fim, considere uma execução concorrente abstraída pela seguinte sequência de rótulos:

N = T11 T12 T21 T22 T13 T14

Esta execução N não é serial e também não é serializável por não ser equivalente nem a S, nem a uma execução serial S' em que T1 é processada antes de T2. De fato, T12 lê o saldo inicial C, que é posteriormente escrito por T13. Desta forma, a atualização expressa por T2 é perdida e o banco de dados final (em N) não reflete a execução de T2. Logo N não pode ser equivalente à S ou à S', ou seja, não é serializável.

Controle De Concorrência Entre Transações Em Bancos De Dados

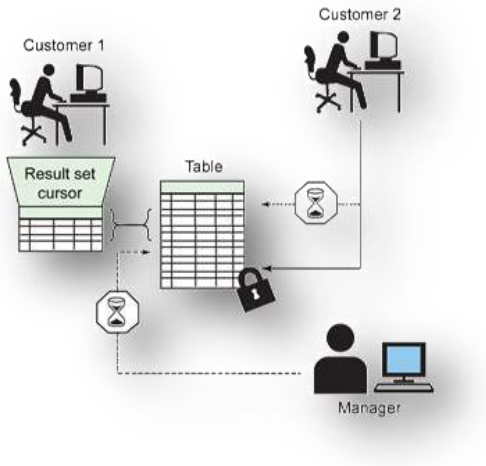

Figura 1: Controle de transações em bancos de dados

Qualquer banco de dados que seja utilizado por mais de um usuário, terá que administrar o controle de concorrência entre as informações que estão sendo acessadas pelos usuários. Controle de concorrência é quando, em um banco de dados, usuários distintos tentam acessar a mesma informação e então é feito um controle entre essas transações. E para a solução deste problema existem diversas técnicas de controle de concorrência que são utilizadas como forma de assegurar a propriedade de não interferência entre uma operação e outra, ou o isolamento das transações executadas ao mesmo tempo. Grande parte dessas técnicas garante a serialização, que é a execução das transações de forma serial. Para isso, é necessário saber que transações são todas as operações executadas entre o início e o fim da transação, e para gerenciar as transações é necessário conhecer as propriedades comumente chamadas de ACID (acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade) que devem ser usadas pelos métodos de controle de concorrência e recuperação do SGBD:

A Atomicidade é o princípio de que uma transação é uma unidade de processamento atômica, ou seja, a transação deve ser realizada por completo ou ela não deve ser realizada de forma alguma. Caso haja alguma falha durante a transação, os efeitos parciais desta transação no banco devem ser desfeitos.

Para que essa transação onde teve a falha seja desfeita, é necessário que o banco de dados emita o comando que desfaça tal transação, garantindo assim a integridade do banco.

A preservação da Consistência permite que uma transação seja executada desde o início até o fim sem que haja a interferência de outras transações durante sua execução, ou seja, é a execução de uma transação isolada, levando o banco de um estado consistente para outro.

O Isolamento garante que uma transação possa parecer isoladamente de outras transações, mesmo tendo várias transações sendo executadas simultaneamente, o sistema irá dar garantias de que, para cada conjunto de transações, uma transação seja encerrada antes do início da outra, ou seja, a execução de uma determinada transação não sofrerá a interferência de nenhuma outra transação que esteja acontecendo simultaneamente.

A Durabilidade (ou permanência) é a garantia de que as mudanças que ocorreram no banco de dados ao término de uma transação que foi finalizada com sucesso persistam no banco, ou seja, após uma operação com êxito ser encerrada, estes dados gravados devem consistir no banco de dados mesmo que ocorrem quaisquer tipos de falha no sistema.

A técnica de bloqueio em duas fases para controle de concorrência é baseado no bloqueio de itens de dados, sendo que, chamamos de bloqueio uma variável que fica atrelada ao item de dados. Este bloqueio pode ser binário (possui dois valores: 1 e 0), logo, o item de dados está bloqueado ou não está bloqueado. Permitindo que o item de dado só esteja acessível para uma transação apenas se a variável não estiver bloqueada (ou estiver com valor 0). São usadas duas operações para o bloqueio binário, são elas: lock(1) e unlock(0), quando o item de dados está sendo usado, o estado da variável é lock(1), assim que a transação encerra a utilização do item é emitida a operação unlock(0), então, o item já está disponível para outra transação. Duas maneiras de bloquear(lock) os dados são:

Bloqueio Compartilhado: quando uma transação recebe este tipo de bloqueio e a instrução é de leitura, então, mais de uma transação poderá acessar o mesmo dado. Se a instrução for de gravação, então ela não poderá participar de um bloqueio compartilhado, ou seja, é permitido que várias transações acessem um mesmo item "A" se todas elas acessarem este item "A" apenas para fins de leitura.

Bloqueio Exclusivo: quando uma transação recebe este tipo de bloqueio, ela fica exclusivamente reservada para a instrução que compõe a transação, não permitindo que outra transação faça uso do dado que está sendo utilizado, logo, um item bloqueado para gravação é chamado de bloqueado exclusivo, pois uma única transação mantém o bloqueio no item.

Uma transação precisa manter o bloqueio do item de dado durante o tempo em que estiver acessando aquele item, até mesmo porque nem sempre o desbloqueio imediato após o acesso final é interessante, pois pode comprometer a serialização em alguns casos.

Sempre haverá a necessidade de bloqueio e desbloqueio dos itens de dados, mas existem algumas situações em que a combinação dessas duas fases pode gerar um problema no banco dados. Caso não seja feito o desbloqueio do item de dado antes da solicitação de um bloqueio a outro item de dado, pode ocorrer um deadlock (ou impasse).

Consideremos que existem duas transações A e B, a transação A está esperando por algum item que está bloqueado na transação B, e a transação B está esperando por algum item que está bloqueado em A, então, ambas ficam na fila de espera, aguardando que seja liberado o bloqueio de um item, como isso não ocorre, as duas transações acabam nunca conseguindo ser concluídas.

Outro problema que pode ocorrer é o starverd (ou starvation). Consideremos que existem três transações A, B e C, a transação A faz a requisição de um bloqueio compartilhado de um item de dado, logo em seguida a transação B faz uma requisição de bloqueio exclusivo do mesmo item de dado, como a transação A está usando o bloqueio compartilhado do item, logo a transação B não poderá deter o acesso, o que faz com que ela fique na fila de espera, até a liberação do mesmo.

Enquanto a transação B está na fila, chega a transação C com o pedido de compartilhamento do mesmo item, como é possível fazer o compartilhamento do item que A está usando, então, a transação C passa na frente de B e consegue o compartilhamento do item de dado da transação A.

Quando a transação A terminar de desocupar, o item de dado continuará ocupado pela transação C, enquanto isso, a transação B continuará aguardando a liberação total do item de dado para que ela possa fazer o bloqueio exclusivo. Caso as próximas transações sejam sempre de acesso compartilhado

deste mesmo item, a transação B nunca conseguirá fazer um progresso, e então, ela é considerada estagnada.

E por aqui eu finalizo este artigo, no qual vimos como ocorre o controle de concorrência usando a técnica de bloqueio em duas fases e quais são alguns dos possíveis problemas que podem ocorrer.