concorrência mais relaxados têm sido propostos para manter a consistência do banco de dados.

21.6 Suporte para transação em SQL

Nesta seção, oferecemos uma rápida introdução ao suporte para transação em SQL. Existem muito mais detalhes, e os padrões mais novos têm mais comandos para processamento de transação. A definição básica de uma transação SQL é semelhante ao conceito já definido de uma transação. Ou seja, ela é uma unidade lógica de trabalho e tem garantias de ser atômica (ou indivisível). Uma única instrução SQL sempre é considerada atômica — ou ela completa a execução sem um erro, ou falha e deixa o banco de dados inalterado.

Com a SQL, não existe uma instrução Begin_Transaction explícita. O início da transação é feito implicitamente quando instruções SQL em particular são encontradas. Porém, cada transação precisa ter uma instrução de fim explícita, que é um COMMIT ou um ROLLBACK. Cada transação tem certas características atribuídas a ela. Essas características são especificadas por uma instrução SET TRANSACTION em SQL. As características são o *modo de acesso*, o *tamanho da área de diagnóstico* e o *nível de isolamento*.

O **modo de acesso** pode ser especificado como READ ONLY ou READ WRITE. O default é READ WRITE, a menos que o nível de isolamento de READ UNCOMMITTED seja especificado (ver a seguir), caso em que READ ONLY é assumido. Um modo READ WRITE permite a execução de comandos de seleção, atualização, inserção, exclusão e criação. Um modo READ ONLY, como o nome indica, serve simplesmente para a recuperação de dados.

A opção de **tamanho da área de diagnóstico**, DIAGNOSTIC SIZE *n*, especifica um valor inteiro *n*, que indica o número de condições que podem ser mantidas de maneira simultânea na área de diagnóstico. Essas condições fornecem informações de feedback (erros ou exceções) ao usuário ou programa nas *n* instruções SQL executadas mais recentemente.

A opção de **nível de isolamento** é especificada usando a instrução ISOLATION LEVEL <isolamento>, em que o valor para <isolamento> pode ser READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ ou SERIALIZABLE.[15] O nível de isolamento default é SERIALIZABLE, embora alguns sistemas usem READ COMMITTED como default. O uso do termo SERIALIZABLE aqui é baseado em não permitir violações que causam leitura suja, leitura não repetitiva e fantasmas,[16] e, assim, não é idêntico ao modo como a serialização foi definida anteriormente na Seção 21.5. Se uma transação é executada em um nível de isolamento inferior a SERIALIZABLE, então uma ou mais das três violações a seguir pode ocorrer:

1. **Leitura suja.** Uma transação T_1 pode ler a atualização de uma transação T_2, que ainda não foi confirmada. Se T_2 falhar e for abortada, então T_1 teria lido um valor que não existe e é incorreto.
2. **Leitura não repetitiva.** Uma transação T_1 pode ler determinado valor de uma tabela. Se outra transação T_2 mais tarde atualizar esse valor e T_1 ler o valor novamente, T_1 verá um valor diferente.
3. **Fantasmas.** Uma transação T_1 pode ler um conjunto de linhas de uma tabela, talvez com base em alguma condição especificada na cláusula SQL WHERE. Agora, suponha que uma transação T_2 insira uma nova linha que também satisfaça a condição da cláusula WHERE usada em T_1, na tabela usada por T_1. Se T_1 for repetida, então T_1 verá um fantasma, uma linha que anteriormente não existia.

A Tabela 21.1 resume as possíveis violações para os diferentes níveis de isolamento. Uma entrada *Sim* indica que uma violação é possível e uma entrada *Não* indica que ela não é possível. READ UNCOMMITTED é a mais complacente, e SERIALIZABLE é a mais restritiva porque evita todos os três problemas mencionados anteriormente.

Uma transação SQL de exemplo pode se parecer com o seguinte:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO FUNCIONARIO (Pnome, Unome, Cpf, Dnr, Salario) VALUES ('Roberto', 'Silva', '99100432111', 2, 35.000);
EXEC SQL UPDATE FUNCIONARIO
    SET Salario = Salario * 1.1 WHERE Dnr = 2;
EXEC SQL COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ... ;
```

[15] Estes são semelhantes aos *níveis de isolamento* discutidos rapidamente ao final da Seção 21.3.

[16] Os problemas de leitura suja e leitura não repetitiva foram discutidos na Seção 21.1.3. Fantasmas serão discutidos na Seção 22.7.1.

Tabela 21.1
Violações possíveis com base nos níveis de isolamento definidos na SQL.

	Tipo de violação		
Nível de isolamento	Leitura suja	Leitura não repetitiva	Fantasma
READ UNCOMMITTED	Sim	Sim	Sim
READ COMMITTED	Não	Sim	Sim
REPEATABLE READ	Não	Não	Sim
SERIALIZABLE	Não	Não	Não

Essa transação consiste em primeiro inserir uma nova linha na tabela FUNCIONARIO e, depois, atualizar o salário de todos os funcionários que trabalham no departamento 2. Se houver um erro em qualquer uma das instruções SQL, a transação inteira é cancelada. Isso implica que qualquer salário atualizado (por essa transação) seria restaurado a seu valor anterior e que a linha recém-inserida seria removida.

Conforme vimos, a SQL oferece uma série de recursos orientados à transação. O DBA ou programadores de banco de dados podem tirar proveito dessas opções para tentar melhorar o desempenho da transação ao relaxar a serialização, se isso for aceitável para suas aplicações.

Resumo

Neste capítulo, discutimos os conceitos de SGBD para processamento de transação. Apresentamos o conceito de uma transação de banco de dados e as operações relevantes ao processamento de transação. Comparamos sistemas monousuário com sistemas de multiusuário e, depois, apresentamos exemplos de como a execução não controlada de transações simultâneas em um sistema multiusuários pode gerar resultados e valores de banco de dados incorretos. Também discutimos os diversos tipos de falhas que podem ocorrer durante a execução da transação.

Em seguida, apresentamos os estados típicos pelos quais uma transação passa durante a execução e discutimos vários conceitos que são usados nos métodos de recuperação e controle de concorrência. O log do sistema registra os acessos do banco de dados, e o sistema utiliza essa informação para se recuperar de falhas. Uma transação tem sucesso ou atinge seu ponto de confirmação, ou falha e precisa ser cancelada. Uma transação confirmada tem suas mudanças gravadas permanentemente no banco de dados. Apresentamos uma visão geral das propriedades desejáveis das transações — atomicidade, preservação de consistência, isolamento e durabilidade — que normalmente são conhecidas como propriedades ACID.

Depois, definimos um schedule (ou histórico) como uma sequência de execução das operações de várias transações com possível intercalação. Caracterizamos os schedules em relação a sua facilidade de recuperação. Os schedules recuperáveis garantem que, quando uma transação é confirmada, ela nunca precisará ser desfeita. Os schedules sem cascata acrescentam uma condição para garantir que nenhuma transação cancelada exija o cancelamento em cascata de outras transações. Schedules estritos oferecem uma condição ainda mais forte que permite um esquema de recuperação simples, consistindo em restaurar os valores antigos dos itens que foram alterados por uma transação abortada.

Definimos a equivalência dos schedules e vimos que um schedule serializável é equivalente a algum schedule serial. Definimos os conceitos de equivalência de conflito e equivalência de visão, que levam às definições de serialização de conflito e serialização de visão. Um schedule serializável é considerado correto. Apresentamos um algoritmo para testar a serialização (conflito) de um schedule. Discutimos por que o teste de serialização é impraticável em um sistema real, embora possa ser usado para definir e verificar os protocolos de controle de concorrência, e mencionamos rapidamente definições menos restritivas de equivalência de schedule. Por fim, fornecemos uma breve visão geral de como os conceitos de transação são usados na prática dentro da SQL.

Perguntas de revisão

21.1. O que significa a execução concorrente de transações de banco de dados em um sistema multiusuário? Discuta por que o controle de concorrência é necessário e dê exemplos informais.

21.2. Discuta os diferentes tipos de falhas. O que significa uma falha catastrófica?

21.3. Discuta as ações tomadas pelas operações read_item e write_item em um banco de dados.

21.4. Desenhe um diagrama de estado e discuta os estados típicos pelos quais uma transação passa durante a execução.

21.5. Para que é usado o log do sistema? Quais são os tipos característicos de registros em um log do sistema? O que são pontos de confirmação da transação e por que eles são importantes?

21.6. Discuta as propriedades de atomicidade, durabilidade, isolamento e preservação da consistência de uma transação de banco de dados.

21.7. O que é um schedule (histórico)? Defina os conceitos de schedules recuperáveis, sem cascata e estritos, e compare-os em matéria de sua facilidade de recuperação.

21.8. Discuta as diferentes medidas de equivalência de transação. Qual é a diferença entre equivalência de conflito e equivalência de visão?

21.9. O que é um schedule serial? O que é um schedule serializável? Por que um schedule serial é considerado correto? Por que um schedule serializável é considerado correto?

21.10. Qual é a diferença entre as suposições de gravação restrita e gravação irrestrita? Qual é mais realista?

21.11. Discuta como a serialização é usada para impor o controle de concorrência em um sistema de banco de dados. Por que a serialização às vezes é considerada muito restritiva como uma medida da exatidão para os schedules?

21.12. Descreva os quatro níveis de isolamento em SQL.

21.13. Defina as violações causadas por cada um dos seguintes itens: leitura suja, leitura não repetitiva e fantasmas.

Exercícios

21.14. Mude a transação T_2 da Figura 21.2(b) para

read_item(X);
$X := X + M$;
if $X > 90$ then exit
else write_item(X);

Discuta o resultado final dos diferentes schedules na Figura 21.3(a) e (b), onde $M = 2$ e $N = 2$, em relação às seguintes questões: a inclusão da condição acima muda o resultado final? O resultado obedece à regra de consistência implícita (de que a capacidade de X é 90)?

21.15. Repita o Exercício 21.14, acrescentando uma verificação em T_1 de modo que Y não exceda 90.

21.16. Inclua o commit da operação ao final de cada uma das transações T_1 e T_2 na Figura 21.2, e depois liste todos os schedules possíveis para as transações modificadas. Determine quais dos schedules são recuperáveis, quais são sem cascata e quais são estritos.

21.17. Liste todos os schedules possíveis para as transações T_1 e T_2 na Figura 21.2 e determine quais são serializáveis de conflito (corretos) e quais não são.

21.18. Quantos schedules *seriais* existem para as três transações da Figura 21.8(a)? Quais são eles? Qual é o número total de schedules possíveis?

21.19. Escreva um programa para criar todos os schedules possíveis para as três transações da Figura 21.8(a) e para determinar quais desses schedules são serializáveis de conflito e quais não são. Para cada schedule serializável de conflito, seu programa deverá imprimir o schedule e listar todos os schedules seriais equivalentes.

21.20. Por que uma instrução de fim de transação explícita é necessária em SQL, mas não uma instrução de início explícita?

21.21. Descreva situações em que cada um dos diferentes níveis de isolamento seriam úteis para o processamento de transação.

21.22. Qual dos seguintes schedules é serializável (de conflito)? Para cada schedule serializável, determine os schedules seriais equivalentes.

a. $r_1(X)$; $r_3(X)$; $w_1(X)$; $r_2(X)$; $w_3(X)$;

b. $r_1(X)$; $r_3(X)$; $w_3(X)$; $w_1(X)$; $r_2(X)$;

c. $r_3(X)$; $r_2(X)$; $w_3(X)$; $r_1(X)$; $w_1(X)$;

d. $r_3(X)$; $r_2(X)$; $r_1(X)$; $w_3(X)$; $w_1(X)$;

21.23. Considere as três transações T_1, T_2 e T_3, e os schedules S_1 e S_2 a seguir. Desenhe os grafos de serialização (precedência) para S_1 e S_2 e indique se cada schedule é serializável ou não. Se um schedule for serializável, escreva o(s) schedule(s) serial(is) equivalente(s).

T_1: $r_1(X)$; $r_1(Z)$; $w_1(X)$;
T_2: $r_2(Z)$; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$;
T_3: $r_3(X)$; $r_3(Y)$; $w_3(Y)$;
S_1: $r_1(X)$; $r_2(Z)$; $r_1(Z)$; $r_3(X)$; $r_3(Y)$; $w_1(X)$; $w_3(Y)$; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$;
S_2: $r_1(X)$; $r_2(Z)$; $r_3(X)$; $r_1(Z)$; $r_2(Y)$; $r_3(Y)$; $w_1(X)$; $w_2(Z)$; $w_3(Y)$; $w_2(Y)$;

21.24. Considere os schedules S_3, S_4 e S_5 a seguir. Determine se cada schedule é estrito, sem cascata, recuperável ou não recuperável. (Determine a condição de facilidade de recuperação mais estrita que cada schedule satisfaz.)

S_3: $r_1(X)$; $r_2(Z)$; $r_1(Z)$; $r_3(X)$; $r_3(Y)$; $w_1(X)$; c_1; $w_3(Y)$; c_3; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$; c_2;
S_4: $r_1(X)$; $r_2(Z)$; $r_1(Z)$; $r_3(X)$; $r_3(Y)$; $w_1(X)$; $w_3(Y)$; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$; c_1; c_2; c_3;
S_5: $r_1(X)$; $r_2(Z)$; $r_3(X)$; $r_1(Z)$; $r_2(Y)$; $r_3(Y)$; $w_1(X)$; c_1; $w_2(Z)$; $w_3(Y)$; $w_2(Y)$; c_3; c_2;

Bibliografia selecionada

O conceito de serialização e as ideias relacionadas para manter a consistência em um banco de dados foram introduzidas em Gray et al. (1975). O conceito da transação de banco de dados foi discutido inicialmente em Gray (1981), que ganhou o cobiçado ACM Turing Award em 1998 por seu trabalho sobre transações de banco de dados e implementação de transações em SGBDs relacionais. Bernstein, Hadzilacos e Goodman (1988) focalizam as técnicas de controle de concorrência e recuperação em sistemas de banco de dados centralizados e distribuídos; trata-se de uma excelente referência. Papadimitriou (1986) oferece um ponto de vista mais teórico. Um grande livro de referência de mais de mil páginas, por Gray e Reuter (1993), oferece um ponto de vista mais prático dos conceitos e técnicas de processamento de transação. Elmagarmid (1992) oferece coleções de artigos de pesquisa sobre processamento de transação para aplicações avançadas. O suporte de transação em SQL é descrito em Date e Darwen (1997). A serialização de visão é definida em Yannakakis (1984). A facilidade de recuperação de schedules e a confiabilidade em bancos de dados são discutidas em Hadzilacos (1983, 1988).