9. Concurrency Control

Transações e controle de concorrência são descritos neste capítulo. Também é abordado como o DBMaker mantém o acesso simultâneo e a precisão dos dados em um ambiente multiusuário com o mecanismo de bloqueio. A Seção de Transações apresenta o conceito de transação e as funções usadas no gerenciamento de uma transação. A Seção Níveis de Isolamento de Transações descreve os quatro níveis de transação. A Seção Ambiente Multiusuário explica a necessidade de controle de concorrência em um sistema de banco de dados. Finalmente, a seção Locks explica as técnicas de controle de concorrência usadas pelo DBMaker.

9.1 Transactions

Em um banco de dados, uma transação é uma unidade de trabalho composta por uma ou mais instruções SQL. É uma operação atômica, o que significa que deve completar uma série de instruções inteiramente ou não fazer nada. As propriedades de uma transação são: serial, atômica, permanente, consistente e isolada.

Transaction States

Uma transação deve estar em um dos seguintes estados:

- **Ativa** — Quando uma transação começa a executar, ela imediatamente entra no estado ativo. No estado ativo, uma transação pode realizar várias operações no banco de dados.
- **Parcialmente Completada** — Quando uma transação alcança sua última instrução no DBMaker (como COMMIT WORK), ela entra no estado parcialmente completada. A transação completou sua execução e ainda pode ser abortada se ocorrer um erro durante a saída real. O resultado não pode ser gravado no disco e uma falha de hardware pode impedir sua conclusão bem-sucedida.
- **Completada** — Quando uma transação conclui sua execução com sucesso, ela entra no estado completada.
- **Falhada** — Quando uma transação não pode prosseguir para uma conclusão normal, ela entra no estado falhada. Isso pode ser causado por erros de hardware ou lógica, ou um aborto da transação pelo usuário durante um estado ativo.

- **Abortada** — Quando uma transação termina de forma não bem-sucedida, ela entra no estado abortada. Nesta situação, qualquer alteração ou efeito que a transação aplicou ao banco de dados deve ser revertido.

O diagrama de estados correspondente a uma transação é mostrado na Figura 9-1.

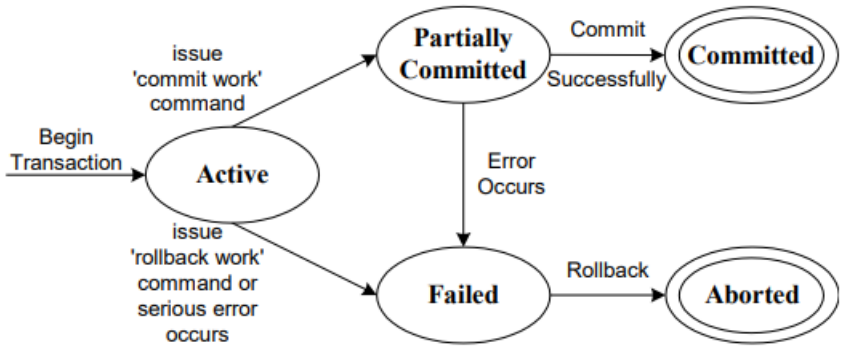

Figure 9-1 The transaction states

Managing a Transaction

Ao conectar-se ao DBMaker, uma transação começa automaticamente e entra no estado ativo. O DBMaker iniciará automaticamente uma nova transação após a transação anterior ter sido terminada.

Cada vez que uma instrução é executada, uma transação é automaticamente confirmada pelo DBMaker. Isso é conhecido como modo de autocommit. Nesse modo, a duração de uma transação é igual à duração de uma única instrução SQL. Isso significa que, quando uma transação é terminada no final de uma instrução SQL, outra começa com a próxima instrução SQL. Cada instrução SQL é uma transação independente.

Para forçar uma transação a permanecer não confirmada até que várias instruções SQL tenham sido executadas, mude para o modo de confirmação manual emitindo um comando SET AUTOCOMMIT OFF. Nesse modo, uma transação só pode ser confirmada usando o comando SQL COMMIT WORK. Quantas instruções SQL forem necessárias podem ser executadas antes de terminar a transação. Para terminar a transação, emita um comando COMMIT WORK para confirmar as alterações, ou emita um comando ROLLBACK WORK para abortar quaisquer alterações feitas e terminar a transação.

Para retornar ao modo de autocommit, emita um comando SET AUTOCOMMIT ON. O modo de transação padrão é AUTOCOMMIT ON.

NOTA: *Após a terminação de uma transação, todos os recursos alocados são liberados.*

Using a Savepoint

Um ponto de salvamento é um ponto intermediário que pode ser arbitrariamente declarado dentro do contexto de uma transação. Um ponto de salvamento é usado para reverter o trabalho realizado após a declaração de um ponto de salvamento dentro de uma transação.

Por exemplo, uma transação com uma série de instruções é executada, e ocorre um erro ao executar a vigésima instrução. Se um ponto de salvamento for marcado entre a décima quinta e a décima sexta instruções, as primeiras quinze instruções podem ser preservadas. Um usuário pode reverter para o ponto de salvamento e começar a emitir comandos a partir da décima sexta instrução SQL após corrigir o erro. A Figura 9-2 mostra um exemplo de como o usuário não precisa abortar a transação e reenviar todas as instruções.

No entanto, se o usuário não marcar um ponto de salvamento entre a décima quinta e a décima sexta instruções, a transação deve ser abortada e as primeiras quinze instruções devem ser reenviadas. Isso é inconveniente e desperdiça tempo. Um ponto de salvamento resolve esse problema.

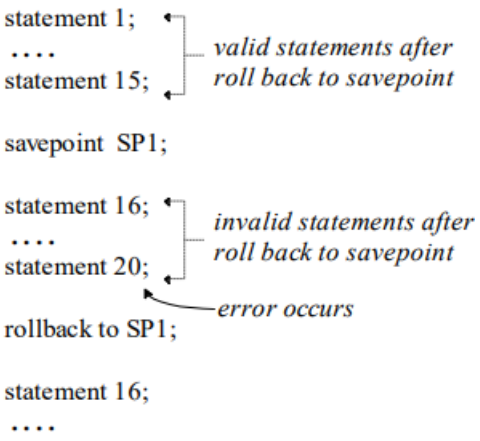

Figure 9-2 Using Savepoints

Os comandos SAVEPOINT e ROLLBACK TO ... marcam um ponto de salvamento e fazem a reversão para um ponto de salvamento específico.

Exemplo 1:

O comando SAVEPOINT:

```
dmSQL> SAVEPOINT <savepoint_name>;
```

Exemplo 2:

O comando ROLLBACK TO …

```
dmSQL> ROLLBACK TO <savepoint_name>;
```

O usuário especifica o <savepoint_name>. Após reverter para um ponto de salvamento, os recursos do sistema que foram alocados após o ponto de salvamento, como bloqueios, são liberados.

9.2 Transaction Isolation Levels

Transactions Concurrency Issues

Transações executando simultaneamente no mesmo banco de dados podem exibir certos fenômenos indesejados. Estes são comumente chamados de leitura suja, leitura não repetível e leitura fantasma.

A discussão a seguir é baseada nos seguintes dados (a menos que indicado de outra forma):

Uma tabela table1 com a coluna c1 contendo os valores 1, 3 e 5. Duas transações T1 e T2 executam simultaneamente nesta tabela.

DIRTY READ

Definição: Uma transação lê dados escritos por uma transação não confirmada concorrente.

```
T1 T2
------------ ------------
Insert 4
Select c1<5
… …
Commit or rollback
```

Quando T1 executa "select c1 <5", o seguinte resultado é retornado: c1 = 1, 3, 4. No entanto, esse resultado pode estar incorreto porque T2 pode fazer rollback posteriormente.

NON-REPEATABLE READ

Definição: Uma transação relê dados que tinha lido anteriormente e descobre que os dados foram modificados por outra transação.

Exemplo:

```
T1 T2
------------ -----------------
Select c1<5
 Update c1=2 where c1=1
 Commit
Select c1<5
```

Na primeira seleção, o resultado c1 = 1, 3 é retornado para T1. Em seguida, T2 atualiza o valor 1 para 2 e confirma essa atualização. Posteriormente, T1 executa a mesma consulta e o resultado 2, 3 é retornado. T1 executou a mesma instrução duas vezes, mas valores diferentes foram retornados a cada vez.

PHANTOM READ

Definição: Duas leituras do mesmo predicado retornam conjuntos diferentes de itens. A segunda leitura retorna pelo menos um item que não está no conjunto original.

Exemplo:

```
T1 T2
------------------ ------------
Select c1<5
Insert 4
Commit
Select c1<5
```

Primeiro, a instrução SELECT de T1 é executada e o resultado c1 = 1, 3 é retornado. Em seguida, T2 insere o valor 4 e confirma a alteração. Posteriormente, T1 executa a mesma instrução de consulta, e o resultado 1, 3, 4 é retornado. T1 executou a mesma instrução duas vezes, mas resultados diferentes foram retornados a cada vez. Alguns itens no segundo resultado não estão no primeiro resultado. Neste exemplo, o valor 4 no segundo resultado é o fantasma.

The Four Transaction Isolation Levels

Independentemente dos três problemas de concorrência, a ANSI/ISO SQL define quatro níveis de concorrência de transações:

ISOLATION LEVEL	DIRTY READ	NON-REPEATABLE READ	PHANTOM READ
Leitura confirmada	Possível	Possível	Possível
Leitura não confirmada	Não é possível	Possível	Possível
Leitura Repetível	Não é possível	Não é possível	Possível
Sinalizável	Não é possível	Não é possível	Não é possível

Set Transaction Isolation levels in DBMaker

O DBMaker oferece três métodos para definir os níveis de isolamento de transações usando a palavra-chave dmconfig, a função ODBC e a sintaxe SQL no dmsql:

DMCONFIG KEYWORD

A palavra-chave relacionada é **DB_IsoLv**.

```
DB_IsoLv {1,2,3,4}
1 : READ UNCOMMITTED
2 : READ COMMITTED
3 : REPEATABLE READ
4 : SERIALIZABLE
```

O valor padrão é 1.

O **DB_IsoLv** é configurado para indicar o nível de isolamento padrão de cada transação. Por exemplo, se **DB_IsoLv = 3**, o nível de isolamento padrão de cada transação é **Repeatable Read**.

ODBC FUNCTION

SQLSetConnectionOption é usado para definir o nível de isolamento de transação, e **SQLGetConnectionOption** é usado para obter o nível de isolamento de transação atual.

```
SQLSetConnectOption(     HDBC,     SQL_ATTR_TXN_ISOLATION,     level)
Level                              :                                  {
SQL_TXN_READ_UNCOMMITTED,
SQL_TXN_READ_COMMITTED,
SQL_TXN_REPEATABLE_READ,
SQL_TXN_SERIALIZABLE
}
SQLGetConnectOption(     HDBC,     SQL_ATTR_TXN_ISOLATION,     &level)
```

SQL SYNTAX

```
Type "SET TRANSACTION ISOLATION LEVEL [level]" in the command line.
 [level] :
{ READ COMMITTED
 | READ UNCOMMITTED
 | REPEATABLE READ
 | SERIALIZABLE
 }
Type "CALL GETSYSTEMOPTION('isolv',?);" in the command line in the
dmsql will get
information about isolation level.
```

Exemplo:

Para obter informações sobre o nível de isolamento de transação

```
dmSQL> SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
dmSQL> CALL GETSYSTEMOPTION('isolv',?);
OPTION_VALUE :SQL_TRANSACTION_READ_UNCOMMITTED
```

9.3 Multi-User Environment

Quando mais de um usuário está acessando um banco de dados, é importante considerar o que pode acontecer quando eles tentam acessar dados simultaneamente.

Sessions

Uma conexão é um caminho de comunicação entre um usuário e o DBMaker. Um caminho de comunicação é estabelecido usando memória compartilhada ou uma rede. Antes de usar os recursos do banco de dados, estabeleça uma conexão com o DBMaker usando a seguinte instrução SQL

Exemplo:

Para conectar um usuário a um banco de dados DBMaker:

```
dmSQL>  CONNECT  TO  database_name  user_name  password;
```

Quando um usuário se conecta a um banco de dados DBMaker, a conexão específica é chamada de sessão. Uma sessão dura desde o momento em que um usuário se conecta a um banco de dados DBMaker até o momento em que o usuário se desconecta. Uma sessão só pode ter uma transação ativa por vez.

The Necessity of Concurrency Control

Em um ambiente de sistema de banco de dados multiusuário, mais de um usuário pode se conectar a um banco de dados ao mesmo tempo. Isso pode resultar em várias transações atualizando o mesmo banco de dados simultaneamente. Se nenhum mecanismo de controle de concorrência for utilizado, várias situações podem resultar em inconsistência de dados:

- O problema da atualização perdida
- O problema da atualização temporária
- O problema do resumo incorreto

LOST UPDATE PROBLEM

O problema da atualização perdida ocorre quando duas transações atualizam um item de dados aproximadamente ao mesmo tempo.

Exemplo:

As transações T1 e T2 lêem e modificam o valor de X, mas utilizam cálculos diferentes para modificar o valor. Isso resulta em cada transação contendo um valor diferente para X. A transação T1 escreve o valor que possui para X no banco de dados após lê-lo, mas antes de T2 escrever seu valor para X. Em seguida, T2 escreve o valor que possui para X no banco de dados, sobrescrevendo o valor escrito por T1. O valor escrito por T1 é perdido.

```
T1 T2
-------------- --------------
read(X);
 read(X);
X = X - N;
 X = X + M;
write(X);
 write(X);
```

TEMPORARY UPDATE PROBLEM

O problema da atualização temporária ocorre quando uma transação atualiza um valor, mas é revertida após outra transação atualizar o mesmo valor.

Exemplo:

A transação T1 lê e modifica o valor de X, escreve-o de volta no banco de dados e depois continua com outros comandos. Enquanto a transação T1 continua executando, a transação T2 lê o valor de X, modifica-o para um novo valor e escreve-o de volta no banco de dados. A transação T1 então falha antes da conclusão e deve reverter todos os valores para restaurar o banco de dados ao seu status original. O sistema de gerenciamento de banco de dados restaura o valor original de X, sobrescrevendo o valor escrito pela transação T2. O valor de X calculado pela transação T2 existe apenas temporariamente.

```
**T1 T2
-------------- --------------
read(X); read(X);
X = X - N;
write(X); X = X + M;
 write(X);
rollback;**
```

INCORRECT SUMMARY PROBLEM

Um problema de resumo incorreto ocorre quando uma transação está calculando a soma agregada de vários registros enquanto outras transações estão atualizando esses registros.

Exemplo:

A transação T1 calcula a soma agregada usando os valores de X e Y ao mesmo tempo em que a transação T2 está modificando esses valores. A transação T2 atualiza o valor de X antes que a transação T1 o utilize para calcular a soma, e atualiza o valor de Y após a transação T1 usá-lo para calcular a soma. Isso resulta em a transação T1 usar alguns valores para calcular a soma antes de serem atualizados e outros depois de serem atualizados. Quando ambas as transações são concluídas, o valor da soma está incorreto em relação aos valores no banco de dados.

```
T1 T2
--------------- -------------
sum = 0;
 read(X);
 X = X - N;
 write(X);
read(X);
sum = sum + X;
read(Y);
sum = sum + Y;
 read(Y);
Y = Y + N;
 write(Y);
```

Existem várias técnicas para resolver problemas de concorrência, como bloqueios e carimbos de tempo. A próxima seção mostra como a técnica de bloqueio é aplicada no DBMaker para controlar a execução concorrente de transações.

9.4 Locks

Nesta seção, o conceito de bloqueio é apresentado primeiro. Em seguida, o mecanismo de bloqueio do DBMaker é introduzido, incluindo a granularidade do bloqueio e os modos de bloqueio. Finalmente, é demonstrado como lidar com o deadlock.

Lock Concept

Em geral, um sistema de banco de dados multiusuário utiliza várias formas de bloqueio para sincronizar o acesso de transações concorrentes. Antes de acessar os objetos de dados, como tabelas e tuplas, uma transação deve bloquear esses objetos de dados.

O bloqueio no DBMaker é totalmente automático e não requer nenhuma ação do usuário. O bloqueio implícito ocorre em todas as instruções SQL; os usuários não precisam bloquear explicitamente nenhum objeto de dados no banco de dados.

SHARED AND EXCLUSIVE LOCKS

Em geral, três tipos de bloqueio são usados para permitir operações de leitura múltipla com escrita única em um banco de dados multiusuário:

- **Share Locks (S)** — Uma transação envolvendo uma operação de leitura em um objeto de dados. Para suportar um maior grau de concorrência de dados, várias transações podem adquirir bloqueios de compartilhamento no mesmo objeto de dados ao mesmo tempo.
- **Update Locks (U)**— Uma transação envolvendo uma operação de atualização planejada ou uma operação de atualização em um objeto de dados. Este bloqueio é compatível com os bloqueios de compartilhamento, mas não é compatível com os bloqueios exclusivos. Um objeto pode ter apenas um bloqueio de atualização por vez.
- **Exclusive Locks (X)**— Uma transação envolvendo uma operação de atualização em um objeto de dados. Esta transação é a única que pode acessar o objeto até que o bloqueio exclusivo seja liberado.

TWO-PHASE LOCKING

O protocolo de bloqueio em duas fases é usado para garantir que as transações sejam serializadas. No protocolo de bloqueio em duas fases, cada transação deve emitir todos os pedidos de bloqueio antes de poder emitir qualquer pedido de desbloqueio.

O protocolo pode ser dividido em duas fases:

- **Fase de Expansão (ou Crescimento)** — Esta fase permite que a transação emita quaisquer novos pedidos de bloqueio necessários. Pedidos de desbloqueio não são permitidos nesta fase.

- **Fase de Encolhimento** — Esta fase permite que a transação libere os bloqueios adquiridos na fase de expansão. Novos pedidos de bloqueio não são permitidos nesta fase.

O protocolo de bloqueio em duas fases é atualmente utilizado pelo DBMaker para fornecer controle de concorrência por meio da serialização das transações.

DEADLOCK

Quando duas ou mais transações estão aguardando a liberação de dados bloqueados por outras transações antes de poderem prosseguir, ocorre um deadlock.

Exemplo:

A transação T1 está aguardando a transação T2 liberar o bloqueio de compartilhamento do X, enquanto a transação T2 está aguardando a transação T1 liberar o bloqueio de compartilhamento do Y. Portanto, ocorre um deadlock e o sistema ficará esperando indefinidamente.

```
T1 T2
------------------- --------------
share_lock(Y);
read(Y);
 share_lock(X);
 read(X);
exclusive_lock(X);
(T1 waits for T2) exclusive_lock(Y);
 (T2 waits for T1)
```

Lock Granularity

Existem três níveis de granularidade para bloqueios de dados no DBMaker: relação (tabela), página e tupla (linha). Uma relação contém várias páginas, e uma página contém várias tuplas.

Um bloqueio aplicado em um nível superior se estende para os níveis inferiores. Por exemplo, se um usuário obtiver um bloqueio exclusivo (X lock) em uma relação, todas as páginas e tuplas contidas nessa relação terão o bloqueio X aplicado a elas. Portanto, nenhum usuário poderá acessar qualquer tupla ou página dessa relação. No entanto, se um usuário obtiver um bloqueio X em uma tupla, outro usuário pode obter um bloqueio X em outra tupla simultaneamente. Não há interferência entre dois

objetos no mesmo nível ao usar o bloqueio X. A Figura 9-3 mostra a granularidade de bloqueio (níveis) no DBMaker.

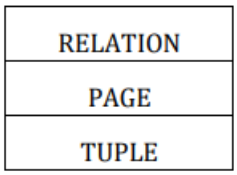

Figure 9-3: Lock granularity

Usar uma granularidade de bloqueio mais alta resulta em um menor grau de concorrência de dados; em contraste, a granularidade de bloqueio mais alta utiliza menos recursos do sistema (como memória compartilhada). A seleção do nível de granularidade de bloqueio é um trade-off entre concorrência e recursos. No DBMaker, o nível de granularidade de bloqueio padrão é linha, mas se uma granularidade de bloqueio diferente for necessária, ela pode ser especificada ao criar uma tabela. Consulte o Capítulo 5, Arquitetura de Armazenamento, para mais informações.

Lock Types

Os principais modos de bloqueio suportados no DBMaker são bloqueios compartilhados (S), bloqueios de atualização (U) e bloqueios exclusivos (X). Mais de um usuário pode ter um bloqueio S em um objeto de dados simultaneamente, mas apenas um usuário pode ter um bloqueio X ou U em um objeto de dados. Além dos bloqueios S, U e X, outro modo de bloqueio chamado bloqueio de intenção é suportado.

Quando um objeto de dados é bloqueado, o sistema atribuirá automaticamente um bloqueio de intenção ao próximo objeto de granularidade superior. Por exemplo, um bloqueio S especificado em uma tupla gerará um bloqueio de intenção S (IS) na página que inclui essa tupla, e um bloqueio IS na relação à qual a tupla pertence.

Os modos de bloqueio de intenção suportados são:

- **IS** — Indica que o bloqueio S é especificado em uma granularidade inferior.
- **IU** — Indica que o bloqueio U é especificado em uma granularidade inferior.
- **IX** — Indica que o bloqueio X é especificado em uma granularidade inferior.
- **SIX** — Indica que um bloqueio S é especificado na granularidade atual e um bloqueio X é especificado em uma granularidade inferior. Esta é uma combinação dos bloqueios S e IX.
- **SIU** — Indica que um bloqueio S é especificado na granularidade atual e um bloqueio U é especificado em uma granularidade inferior. Esta é uma combinação dos bloqueios S e IU.

- **UIX** — Indica que um bloqueio U é especificado na granularidade atual e um bloqueio X é especificado em uma granularidade inferior. Esta é uma combinação dos bloqueios U e IX.

O resultado da compatibilidade de cada um dos modos de bloqueio é listado na Tabela 9-1. T representa verdadeiro, o que significa que a matriz para cada um dos dois modos de bloqueio é compatível e pode existir simultaneamente em um objeto de dados. F representa falso, o que significa que a matriz para cada um dos dois modos de bloqueio não é compatível e não pode existir simultaneamente.

Se os pedidos de bloqueio em um objeto de dados entrarem em conflito com um bloqueio existente nesse objeto, o pedido não será executado até que o bloqueio existente seja liberado, ou até que o tempo de espera para o pedido de bloqueio expire. Se a mensagem de erro 'Tempo de espera do bloqueio' for retornada ao usuário, o tempo de espera para o bloqueio expirou. O tempo de espera padrão é de 5 segundos. No entanto, os usuários podem especificar um tempo de espera diferente configurando o valor da palavra-chave DB_LTimO no arquivo dmconfig.ini para outro valor, de acordo com suas necessidades individuais.

Exemplo:

Para definir o tempo de espera para 8 segundos

```
DB_LTimO = 8;
```

	N	**IS**	**S**	**UI**	**SIU**	**IX**	**U**	**SIX**	**UIX**	**X**
N	T	T	T	T	T	T	T	T	T	T
IS	T	T	T	T	T	T	T	T	T	F
T	T	T	T	T	T	F	T	F	F	F
UI	T	T	T	T	T	T	F	T	F	F
IX	T	T	T	T	T	F	F	F	F	F
U	T	T	T	F	F	F	F	F	F	F
SIX	T	T	F	T	F	F	F	F	F	F
UIX	T	T	F	F	F	F	F	F	F	F
X	T	F	F	F	F	F	F	F	F	F

Dealing with Deadlock

Ao analisar o gráfico de "espera por", o DBMaker detecta automaticamente uma situação de deadlock. Se um deadlock for detectado, uma transação vítima é abortada para resolver o problema de deadlock.

Exemplo:

O DBMaker detecta um deadlock quando a transação T2 emite um bloqueio X em Y. A transação T2 será abortada para resolver o problema de deadlock e o usuário que está executando a transação T2 receberá a mensagem de erro: "transação abortada devido a deadlock".

```
T1                              T2

-----------------           ---------------

share_lock(Y);

read(Y);

                                share_lock(X);

                                read(X);

exclusive_lock(X);

(T1 waits for T2)           exclusive_lock(Y);

                                (T2 waits for T1)

                                 T2 aborted by DBMaker
```