

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA  
COMPUTAÇÃO**

**Aluna: Izabel Cristina Mioranza**

**Confiabilidade em Banco de Dados Distribuído:  
Uma análise de soluções implementadas.**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

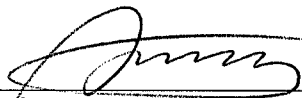
**Orientador: Murilo Silva de Camargo**

Florianópolis, Dezembro de 2001

# **Confiabilidade em Banco de Dados Distribuído: Uma análise de soluções implementadas.**

Aluna: Izabel Cristina Mioranza

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.



---

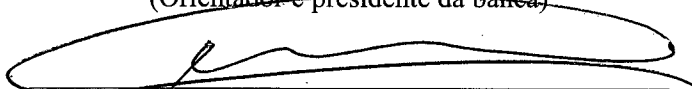
Prof. Dr. Fernando A. O. Gauthier  
(Coordenador do Curso)

Banca Examinadora



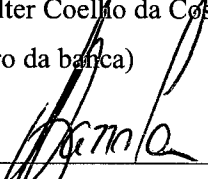
---

Prof. Dr. Eng. Murilo Silva de Camargo  
(Orientador e presidente da banca)



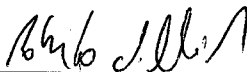
---

Prof. Dr. Rosvelter Coelho da Costa  
(Membro da banca)



---

Prof. Dr. Vitorio Bruno Mazolla  
(Membro da banca)



---

Prof. Dr. Roberto Willrich  
(Membro da banca)

Antes que os olhos possam ver,  
devem ser incapazes de lágrimas.

Antes que o ouvido possa ouvir,  
deve ter perdido a sensibilidade.

Antes que a voz possa falar  
em presença dos Mestres,  
deve ter perdido a possibilidade de ferir.

Antes que a alma possa erguer-se  
na presença dos Mestres,  
é necessário que seus pés tenham  
sido lavados no sangue do coração.

## AGRADECIMENTOS

A Deus que nunca me abandonou, e que  
permitiu que eu chegasse até aqui;  
A minha família, em especial aos meus pais,  
que sempre me apoiaram;  
Aos meus amigos pela força;  
Em especial ao meu orientador  
Murilo S. de Camargo pelo incentivo e pela ajuda.

## Sumário

Capítulo 1.....	1
Introdução .....	1
1.1 Apresentação .....	1
1.2 Metodologia.....	3
1.3 Estrutura da dissertação.....	3
Capítulo 2.....	5
Sistema de Banco de Dados Distribuído .....	5
2. Processamento distribuído dos dados .....	6
2.1 Armazenamento Distribuído dos Dados .....	7
2.1.1 Replicação .....	8
2.1.2 Fragmentação .....	8
2.2 Estruturas de Sistemas .....	9
2.3 Tipos de Falhas no Sistema .....	10
Capítulo 3.....	13
Recuperação em Banco de Dados.....	13
3. Tipos de Falhas .....	13
3.1 Tipos de Armazenamento.....	15
3.1.1 Armazenamento Volátil.....	15
3.1.2 Armazenamento Não-Volátil.....	15
3.1.3 Armazenamento Estável.....	15
3.2 Recuperação e Atomicidade .....	16
3.2.1 Paginação <i>Shadow</i> .....	16
3.2.2 Gerenciador de Transações e o Gerenciador de Log .....	17
3.2.2.1 Registro de Log .....	18
3.2.2.2 Registro de Log de Desfazer.....	20
3.2.2.3 Registro de Log de Refazer .....	25
3.2.2.4 Registro de log de desfazer/refazer.....	29
3.3 Arquivo de armazenamento.....	31
Capítulo 4.....	33
Confiabilidade em um Banco de Dados Distribuído .....	33
4. Sistema, estado e falhas .....	34
4.1 Particionamento de rede .....	36
4.1.1 Protocolo centralizado .....	37
4.1.2 Protocolo baseado em votação.....	37
4.1.3 Protocolo de controle de réplicas.....	38
4.2 Recuperação e Controle de Dados .....	38
4.3 Protocolos de confiabilidade distribuídos.....	39
4.3.1 Protocolo de <i>Commit Two-Phase</i> (2PC).....	40
4.3.1.1 Fase I .....	42
4.3.1.2 Fase II .....	43
4.3.1.3 <i>Commit Two-Phase</i> Distribuído.....	44
4.3.1.4 Recuperação utilizando o protocolo <i>Commit Two-Phase</i> - 2PC.....	48
4.3.2 Protocolo de <i>Commit Three-Phase</i> (3PC) .....	51
4.3.2.1 Fase I .....	52
4.3.2.2 Fase II .....	53
4.3.2.3 Fase III.....	53
4.3.2.4 Protocolo de Término .....	54
4.3.2.5 Manuseio de Falhas utilizando o protocolo 3PC .....	55
Capítulo 5.....	58
Banco de Dados - <i>Oracle</i> .....	58
5. <i>Oracle</i> .....	58
5.1 Estrutura do banco de dados <i>Oracle</i> .....	58
5.1.1 Pool Compartilhado.....	60
5.1.2 Cache do Buffer de Banco de Dados.....	61

5.1.3 Buffer do Registro <i>Redo</i> .....	61
5.1.4 Processos de Servidor (Server Processes) .....	62
5.1.5 Processos de Segundo Plano (Background Processes) .....	62
5.1.5.1 SMON.....	64
5.1.5.2 PMON.....	64
5.1.5.3 DBWR .....	64
5.1.5.4 LGWR .....	65
5.2. Tipos de Falhas.....	65
5.2.1 Erro de usuário .....	65
5.2.2 Falha de declaração e processo.....	66
5.2.3 Falha de Instância .....	66
5.2.4 Falha de mídia ou de disco .....	67
5.2.5 Falha de Comunicação.....	68
5.3 Estruturas usadas para a Recuperação da Base de Dados pelo <i>Oracle</i> .....	68
5.3.1 Cópias de segurança da Base de Dados .....	69
5.3.1.1 Backups do Sistema Operacional .....	69
5.3.1.2 Backups Lógicos.....	70
5.3.1.3 O <i>Redo</i> Log.....	70
5.3.2 Trabalhando usando o Gerente de Recuperação (RMAN) .....	72
5.3.3 Segmentos de <i>Rollback</i> .....	73
5.3.3.1 Rolando para Frente (Rolling Forward) e Rolando para Trás (Rolling Back).....	74
5.4 Banco de Dados <i>Hot Standby</i> .....	76
Capítulo 6.....	78
Banco de Dados - <i>Ingres</i> .....	78
6. <i>Ingres</i> .....	78
6.1 Banco de dados.....	78
6.1.1. Arquivo de Log de Transação.....	79
6.1.2. Processo de Recuperação.....	80
6.1.3. Processo de arquivar log de transações.....	80
6.2. Tipos de Backup.....	80
6.2.1 Backup de Sistema Operacional .....	82
6.2.2. Utilização do <i>Unloaddb</i> .....	82
6.3. Processo de Recuperação.....	83
6.3.1. Checkpoint.....	83
6.3.2. Journalling .....	84
6.3.3. Rollforwarddb.....	85
6.4. Transações Distribuídas.....	86
6.4.1. Protocolo Two-Phase <i>Commit</i> .....	86
Capítulo 7.....	88
Banco de Dados – <i>DB2</i> .....	88
7. <i>IBM/DB2</i> .....	88
7.1 <i>DB2</i> .....	88
7.2 Identificando qual o servidor do Banco de Dados que está com Falha .....	90
7.3 Recuperando uma Base de Dados.....	90
7.3.1 Utilizando o método de recuperação de versão .....	91
7.3.2 Utilizando o método de recuperação Roll-Forward.....	92
7.4 Arquivos de Log do <i>DB2</i> .....	93
7.4.1 Arquivo de log Circular.....	93
7.4.2 Arquivo de log Arquivado .....	94
7.5 Recuperação de falhas de transação utilizando o 2PC.....	95
7.6 Recuperação de Desastre.....	97
Capítulo 8.....	99
Banco de Dados – <i>SQL Server</i> .....	99
8. <i>SQL Server</i> .....	99
8.1 Bancos de Dados ( <i>databases</i> ).....	100
8.1.1 O banco de dados <i>Master</i> .....	100

8.1.2 O banco de dados <i>Model</i> .....	101
8.1.3 O banco de dados <i>Tempdb</i> .....	101
8.1.4 O banco de dados <i>MSDB</i> .....	102
8.2 Estrutura do Banco de Dados .....	102
8.3 Transações Distribuídas utilizando o 2PC .....	103
8.4 Arquivos de log de Transação .....	103
8.5 Backup dos Dados .....	104
8.5.1 Tipos de Backup .....	104
8.5.1.1 <i>Full</i> Backup (Backup Completo).....	105
8.5.1.2 <i>Differential</i> Backup (Backup Diferencial).....	105
8.5.1.3 Log Backup .....	106
8.5.1.4 Backup Snapshot Server-less.....	106
8.5.1.5 Failover Clustering .....	106
8.5.1.6 Backups de arquivo/grupos de arquivos .....	107
8.5.1.7 Backup Lógicos .....	107
8.6 Tipos de Recuperação.....	107
8.6.1 Recuperação <i>Full</i> .....	108
8.6.2 Recuperação <i>Bulk Logged</i> .....	108
8.6.3 Recuperação Simples.....	109
Capítulo 9.....	110
Comparação entre os Bancos de Dados .....	110
9. Banco de Dados .....	110
9.1 Tipos de Falhas.....	110
9.2 Tipos de Backup .....	112
9.3 Recuperação do Banco de Dados .....	114
Capítulo 10.....	117
Conclusão.....	117
10.1 Resumo do trabalho .....	117
10.2 Conclusões.....	119
10.3 Relevância do Trabalho .....	120
10.4 Perspectivas Futuras .....	121
Apêndice A .....	122
Banco de Dados .....	122
1. Definições e Conceitos de Banco de Dados .....	122
1.1 Instâncias e Esquemas .....	123
1.2 Linguagens de Banco de Dados.....	124
1.3 Arquiteturas de Sistemas de Banco de Dados .....	124
1.3.1 Sistemas Centralizados .....	125
1.3.2 Sistema Cliente-Servidor.....	126
1.3.2.1 Servidores de Transações .....	127
1.3.2.2 Servidores de Dados .....	127
1.3.3 Sistemas Paralelos .....	128
1.3.4 Sistemas Distribuídos .....	128
Apêndice B .....	131
Gerenciamento de Transações .....	131
1. Propriedades de uma Transação .....	131
1.1 Estados da Transação.....	132
1.2 A Execução de Transações .....	134
Referência Bibliográfica.....	136

## Índice de Figuras

Figura 2.1 Processamento de banco de dados. [Ozsu,2001].....	6
Figura 2.2 Sistema de Banco de Dados Distribuído.....	7
Figura 3.1. O gerenciador de Log e o gerenciador de Transações. [Molina,2001].....	17
Figura 4.1 Componentes de um ambiente Distribuído [Ozsu, 1999].....	33
Figura 4.2 Cadeia de Eventos que Conduzem a um Fracasso de Sistema [Ozsu, 1999]......	35
Figura 4.3 Fontes de faltas de um Sistema [Siewiork e Swarz,1982]......	35
Figura 4.4. Ações do Protocolo 2PC Centralizado [Ozsu, 2001].....	42
Figura 4.5. Estrutura de Comunicação de 2PC Distribuída [Ozsu, 2001]......	45
Figura 4.6. Transição de estado no protocolo 2PC [Ozsu,2001].....	46
Figura 4.7. Ações do protocolo 3PC – [Ozsu, 2001].....	52
Figura 4.8. Transições de estado no protocolo 3PC [Ozsu,2001]......	55
Figura 5.1. Esquema geral de um banco de dados <i>Oracle</i> . [ <i>Oracle</i> , 1].....	59
Figura 5.2. Fases de restauração e recuperação [Sarin,2000]......	75
Figura 5.3. Fases de Roll-forward e <i>Rollback</i> da recuperação de banco de dados [Sarin,2000].....	76
Figura 7.1. Restabelecimento de uma base de dados utilizando recuperação de versão– [ <i>DB2</i> , 1]..	92
Figura 7.2. Restabelecimento de uma base de dados utilizando Roll-forward – [ <i>DB2</i> ,1].....	93
Figura 7.3. Arquivo de log circular – [ <i>DB2</i> , 1].....	94
Figura 7.4. Arquivo de log arquivado – [ <i>DB2</i> , 1].....	95
Figura 8.1. Banco de Dados <i>SQL Server</i> – [ <i>SQL Server</i> , 3]......	100
Figura 1.1 Relacionamento dos Níveis de Abstração.....	123
Figura 1.2 Um sistema computacional Centralizado.....	125
Figura 1.3 Estrutura geral de um Sistema Cliente-Servidor [Korth,1999]......	126
Figura 1.4. Um sistema Distribuído. [Korth,1999].....	129
Figura 1.1 Diagrama de estado de uma transação [ Korth,1999].....	133



## Índice de Tabela

Tabela 3.1. Etapas de uma transação e seu efeito sobre a memória e o disco. [Molina,2001]. .....	19
Tabela 3.2. Ações e entradas de log em uma transação [Molina,2001] .....	21
Tabela 3.3. Conclusões consideradas utilizando a tabela anterior [Molina, 2001]. .....	24
Tabela 3.4. As ações e as entradas de log, usando um registro de log de refazer.[Molina,2001]. ....	26
Tabela 3.5. As ações e as entradas de log usando um registro de log de desfazer/refazer. [Molina,2001] .....	30
Tabela 5.1. Tipos de backups realizados pelo RMAN [Sarin,2000] .....	73
Tabela 9.1 Tipos de Backup .....	112
Tabela 9.2. Métodos de Recuperação.....	115

## Resumo

Em um Sistema Gerenciador de Banco de Dados somente a replicação dos dados não é suficiente para garantir um banco de dados mais confiável, é necessária a implantação de diversos protocolos que exploram a distribuição e replicação dos dados, executando com isso, operações mais confiáveis.

Confiabilidade pode então, ser definida como uma medida que será utilizada para que o sistema esteja em conformidade com as especificações a que foi submetido. Cada estado pelo qual o sistema de confiabilidade passa, é válido no momento em que o estado conhece sua especificação completamente. Porém, pode ocorrer um estado de não confiabilidade do sistema. Isso é possível porque o sistema pode adquirir um estado interno que não consegue obedecer a sua especificação.

O trabalho proposto tem como enfoque principal estudar confiabilidade dos dados em um banco de dados distribuído. Em seguida, são expostos os conceitos e definições de um banco de dados (distribuído e centralizado), as falhas que podem ocorrer com o mesmo, que podem ser tanto de hardware quanto de software, as técnicas utilizadas em métodos de recuperação (centralizada e distribuída), e, por último, são apresentados quatro (4) banco de dados (*Oracle, Ingres, DB2, SQL Server 2000*). Os bancos de dados em questão apresentam métodos e formas particulares de executar a recuperação dos dados, porém, todos utilizam os tipos de recuperação que são essenciais para tornar os dados novamente confiáveis ao usuário.

Palavras Chaves: Banco de Dados Distribuído, Confiabilidade, Métodos de Recuperação, Backup, Falhas, *Oracle, DB2, Ingres, SQL Server*.

## Summary

In a database management system, only replying to the data isn't enough to guarantee a more reliable data system. In fact, it is necessary to implant various types of protocols that explore the distribution and duplication of data. Because of that it can execute a more reliable operation.

Total reliability can be defined as a measure that will be used to make the system be equal to the specification that it has been submitted by. Each statement that the system passes is valid at the moment that the statement recognizes its complete specifications. But, a state of not system reliability can occur. That is possible because the system can acquire an internal state that may not obey its specification.

The proposed work has a special focus, that is, to study reliability of the data in a distributed kind of data. Next, the concept and definitions of a database (distributed and centralized) are exhibited. The failures that may occur can be caused by the hardware parts and or by the software parts. Also techniques used in methods of recovering (centralized and distributed). Lastly four (4) databases are presented (*Oracle, Ingres, DB2, SQL Server 2000*). The database in focus presents the methods and particular forms of recovery data but all use the kind of recovery that is essential to make sure that the data is again totally reliable.

Key words: Distributed database, Reliability, Recovery Methods, Backup, Failures, *Oracle, DB2, Ingres, SQL Server*.

# Capítulo 1

## Introdução

### 1.1 Apresentação

A arquitetura de sistemas utilizando banco de dados tem sido tradicionalmente centralizada na seguinte maneira: o banco de dados reside em apenas um único computador e todos os programas acessam o banco de dados através dessa máquina, sendo que os usuários podem estar ligados diretamente a este computador ou então estarem conectados através de uma rede.

Atualmente existem inúmeras razões para se trabalhar com um banco de dados distribuído, incluindo o compartilhamento de dados, confiabilidade, disponibilidade e rapidez no processamento de consultas. Porém, todas essas vantagens são acompanhadas de algumas desvantagens, entre elas o alto custo do desenvolvimento de *software*, alto potencial de erros do sistema e o aumento de *overhead*.

Os sistemas distribuídos têm como uma de suas características principais, a necessidade de garantir os aspectos relacionados à consistência dos dados em um Banco de Dados. As características de atomicidade, consistência, isolamento e durabilidade devem ser mantidas indiferentes a disposição física dos dados. Desta forma, devem ser consideradas as variáveis agregadas ao modelo, caracterizando-as e aplicando-as ao contexto criado, tornando-se possível a criação de métodos e funções que visem solucionar tais problemas.

Sistemas de gerência de bancos de dados distribuídos estendem as facilidades usuais de uma gerência de dados local, de tal forma, que o armazenamento de um banco de dados possa ser distribuído ao longo dos nós de uma rede de comunicação de dados, sem comprometer a visão integrada dos dados pelo usuário.

O aumento da confiabilidade do banco de dados através da replicação das partes críticas do banco em vários *sites*, e o aumento da eficiência através de um particionamento da rede e um armazenamento dos dados mais próximos aos locais

onde são mais utilizados, foram alguns fatores que fizeram com que os bancos de dados distribuídos sejam mais divulgados e utilizados nos dias atuais.

Outra forma de garantir que o banco de dados esteja em um estado de confiabilidade, é utilizar sistemas RAID, que garantem que a falha de um único disco não resulte na perda de dados. A forma mais simples e mais rápida de RAID é o disco espelhado, que mantém duas cópias de cada bloco em discos separados. Sistemas mais seguros mantêm uma cópia de cada bloco de armazenamento estável em um sistema remoto, enviando-a por uma rede de computadores, além de armazenar o bloco em um sistema de disco local. Os blocos são enviados ao sistema remoto e conseqüentemente para o armazenamento local, com isso, uma vez completas as operações de saída, essa saída nunca é perdida.

Porém, toda essa evolução trouxe consigo alguns inconvenientes, devida a complexidade de *hardware* e *software*, é certo afirmar que o banco de dados pode sofrer falhas que podem comprometer a integridade do banco, portanto, há a necessidade de incorporar mecanismos que garantam sua integridade. Seguindo esse princípio, o banco de dados pode ser mantido em operação por longos períodos de tempo, sofrendo pequenas interferências pelo controle de recuperação que está sanando pequenas falhas.

Analisando os diversos tipos de falhas (por exemplo, falhas de transação, queda do sistema, falhas de disco, particionamento de rede e outros), que pode ocorrer com o banco foram desenvolvidos algoritmos conhecidos como algoritmos de recuperação. Esses algoritmos estão divididos em duas partes. Na primeira parte, são executadas ações durante o processamento normal da transação a fim de garantir que haja informação suficiente para a recuperação de falhas se houverem, enquanto que na segunda parte são executadas ações em seguida a falha, recuperando o conteúdo do banco de dados para um estado que assegure sua consistência, a atomicidade, durabilidade e o isolamento desta transação.

O objetivo principal desta dissertação é tratar um dos aspectos relacionados a bancos de dados distribuídos, que é a confiabilidade do banco de dados, levando-se em conta que, diversas falhas podem ocorrer com o banco de dados tornando-o inconsistente, e a recuperação do banco que é executada necessita de alguns cuidados especiais para ser realizada em um ambiente distribuído, pois a confiabilidade dos dados

deve ser global e não apenas local em cada uma das máquinas que esta envolvida na transação.

A recuperação dos dados em um banco de dados distribuído irá depender do método de execução das atualizações. Existem duas possibilidades de executar as atualizações, a primeira "*In-Place*", onde os valores dos itens de dados atualizados são fisicamente modificados no banco de dados estável e a segunda, "*Out-of-Place*" onde os valores dos itens de dados atualizados não são fisicamente modificados no banco de dados estável, mas são mantidos como valores separados, sendo que, periodicamente esses valores são incorporados ao banco de dados estável.

## **1.2 Metodologia**

Esta dissertação foi realizada baseando-se nas referências bibliográficas, artigos e relatórios técnicos que foram coletadas no decorrer da pesquisa. Quando se trata dos bancos de dados *Oracle*, *Ingres*, *DB2* e *SQL Server* foi utilizada a documentação técnica de tais bancos para fazer o levantamento das informações, ou seja, foi realizado o estudo de como é o comportamento dos bancos quando há necessidade de recompor o banco, tornando-o estável independentemente do tipo de falha que possa ocorrer, com base nos manuais e *helps* de cada banco.

## **1.3 Estrutura da dissertação**

Este trabalho foi estruturado seguindo as etapas que são expostas a seguir. No capítulo 2 é apresentando o processamento distribuído dos dados, como é realizada a replicação e fragmentação dos dados pelos diversos *sites* que compõem a estrutura e os tipos de falhas que envolvem um banco.

O capítulo 3 descreve o sistema de recuperação em um banco de dados centralizado, tipos de armazenamentos dos dados, os tipos de recuperação que podemos aplicar quando ocorrerem falhas, a recuperação baseada em *Log*, a utilização dos *checkpoints* (pontos de controle), a paginação *shadown* e outros.

O capítulo 4 descreve a confiabilidade de um banco de dados em um sistema distribuído, os protocolos mais utilizados para a recuperação dos dados, o protocolo em

duas fases e o protocolo em três fases, e, também como é feita a recuperação do banco utilizando, esses dois protocolos.

Os capítulos seguintes apresentam os bancos de dados que foram analisados e estudados no decorrer da pesquisa. No capítulo 5 é descrito o primeiro banco que é o *Oracle*, com os tipos de falhas que podem ocorrer com o mesmo e quais as estruturas utilizadas para a recuperação do banco. No capítulo 6 é apresentado o banco de dados *Ingres* e seus principais processos em recompor o banco de dados. No capítulo 7 é apresentado o banco de dados *DB2* com os seus principais métodos em recuperação e backup dos dados, e, finalmente no capítulo 8 é descrito o último banco que é o *SQL Server*, também com suas principais características, principais backup e principais métodos de recuperação.

No capítulo 9 é apresentada uma comparação entre os bancos de dados descritos nos capítulos anteriores, fazendo uma análise quanto aos tipos de falhas que existem e como os bancos de dados se comportam em relação a elas, os tipos de backup que são implementados e as estruturas de recuperação que são utilizadas por cada banco.

No capítulo 10 é apresentada a conclusão da pesquisa desenvolvida, e finalmente é apresentada a bibliografia que foi utilizada para desenvolver este trabalho, desde livros, artigos e endereços da Internet que foram usados.

No Apêndice A é tratado as definições e conceitos gerais sobre banco de dados, com as principais arquiteturas utilizadas, no Apêndice B é realizada uma introdução ao gerenciamento de transações, os estados pelos quais ela pode passar, e, a execução de uma transação local ou distribuída.

## Capítulo 2

### Sistema de Banco de Dados Distribuído

Neste capítulo iremos considerar Computação Distribuída como um número de elementos de processamento autônomo (não necessariamente homogêneos) que são interconectados por uma rede de computadores e que cooperam na realização das tarefas que lhe são atribuídas [Ozsu,1999].

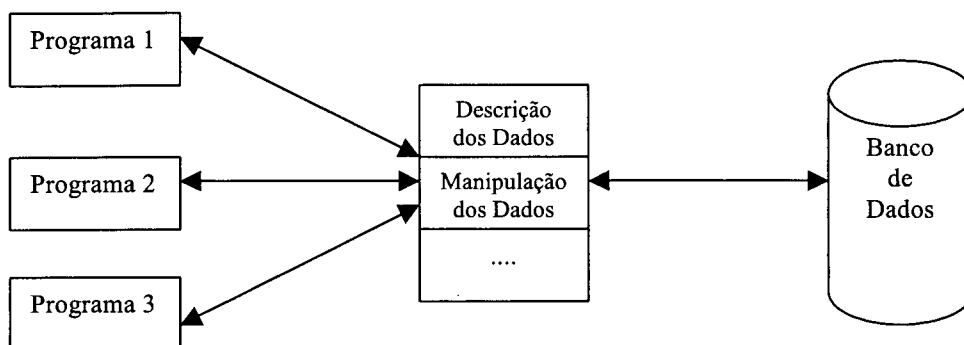
Em um sistema de banco de dados distribuído, o que é distribuído é o processamento lógico, as funções (arquitetura cliente-servidor), os dados (fragmentados) e o controle. A tecnologia utilizada em Sistemas de Banco de Dados Distribuídos é a união das tecnologias de sistemas de banco de dados e redes de computadores .

Os sistemas de banco de dados possuíam dados independentes para cada aplicativo, isto é, cada software aplicativo possui o seu próprio banco de dados. Com a evolução, foram criados os bancos de dados centralizados, que trouxeram o que foi chamado de independência dos dados, por meio do qual os programas aplicativos ficavam imunes às mudanças que ocorriam ao banco de dados, mudanças estas, tanto físicas quanto lógicas.

Com o surgimento da tecnologia de redes de computadores, o processamento centralizado dos dados deixou de existir, para que surgisse uma nova abordagem. Com essa união, surgiu então o que foi chamado de processamento distribuído, com o objetivo de que o fundamental é a integração dos dados e não a centralização desses dados. É necessário deixar bem claro, que o mais importante é alcançar a integração sem necessariamente utilizar o conceito de centralização [Ozsu, 2001].

A figura 2.1 a seguir, tem por objetivo demonstrar como é o processamento de um banco de dados.





**Figura 2.1 Processamento de banco de dados. [Ozsu,2001]**

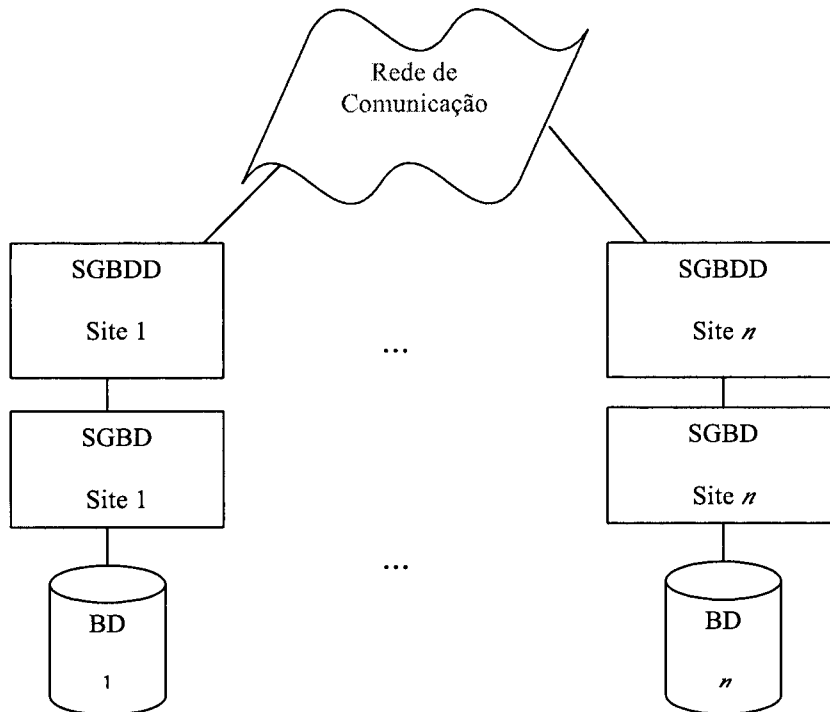
Neste capítulo, serão discutidos os conceitos fundamentais de um banco de dados distribuído, como é realizado o armazenamento dos dados, as estruturas de um sistema distribuído, e, por último quais são os tipos de falhas que podem ocorrer com esse tipo de banco de dados.

## 2. Processamento distribuído dos dados

Um sistema de banco de dados distribuído, consiste em *sites* fracamente acoplados, que compartilham componentes físicos. Cada *site* pode ou não participar da execução de uma transação, e essa transação pode acessar dados em um *site* local ou em diversos *sites*, tornando-se uma transação global.

Um banco de dados distribuído (BDD) é uma coleção de múltiplos bancos de dados que estão interrelacionados, distribuídos sobre uma rede de computadores. Um sistema de gerência de banco de dados distribuídos (SGBDD) é um software que gerencia o BDD e provê um mecanismo de acesso, que torna a distribuição dos dados transparente para o usuário, isto é, o usuário tem uma visão integral dos dados. Um sistema de banco de dados distribuído (SBDD) é a integração de um banco de dados distribuído (BDD) com um sistema gerenciador de banco de dados distribuído(SGBDD) [Ozsu, 2001].

A figura 2.2 apresenta um exemplo de uma arquitetura de um SBDD.



**Figura 2.2 Sistema de Banco de Dados Distribuído**

A interface com o usuário deve ser transparente, isto é, o usuário não deve ter conhecimento se uma operação está sendo realizada em um banco de dados local, ou em diversos bancos de dados espalhados em diversos *sites*. A recuperação desses dados pode ser local ou então, o administrador local pode solicitar ao SGBDD os outros dados que necessita. O SGBDD faz a solicitação ao SGBD que, por sua vez busca em seu BD os dados que lhe foram solicitados; o BD deve retornar ao SGBD os dados; o SGBD retorna os dados que lhe foram solicitados, e o SGBDD apresenta ao usuário os dados solicitados.

## 2.1 Armazenamento Distribuído dos Dados

Levando-se em conta que uma relação  $r$  está armazenada em um banco de dados distribuído, há diversos enfoques para o armazenamento dessa relação:

- **Replicação:** o sistema deverá manter réplicas (cópias) dessa relação e cada cópia ficará armazenada em *sites* diferentes, resultando na replicação dos dados. Uma solução para esse problema é armazenar somente uma cópia da relação  $r$ ;

- Fragmentação: a relação  $r$  é particionada em vários fragmentos e cada fragmento é armazenado em *sites* diferentes;
- Replicação e Fragmentação: a relação é particionada em vários segmentos. O sistema mantém diversas réplicas da cada fragmento [Korth, 1999].

### 2.1.1 Replicação

Considerando-se que uma relação  $r$  é replicada, então uma cópia da relação  $r$  é armazenada em dois ou mais *sites*, ou podemos ter replicação total da relação  $r$  em todos os *sites* que compõem o sistema. Existem algumas vantagens e desvantagens na replicação dos dados, a saber:

- Disponibilidade: se ocorrer falha em algum *site* que possua a relação  $r$ , essa mesma relação pode ser encontrada em um outro *site* do sistema, assim o sistema poderá processar a transação mesmo que ocorra falha em algum dos *sites* que possua a relação  $r$ ;
- Aumento do paralelismo: na maioria dos casos o acesso à relação  $r$  é para leitura da relação, então, diversos *sites* podem processar consultas à relação  $r$  em paralelo. Paralelismo intra-consultas, ocorre quando são executadas consultas diferentes de um mesmo dado em um mesmo *site*. Paralelismo entre-consultas, ocorre quando várias consultas do mesmo dado são realizadas ao mesmo tempo em *sites* diferentes;
- Aumento do *overhead* para atualização: todas as réplicas da relação  $r$  devem ser consistentes, pois de outra forma poderão ocorrer erros no processamento, dessa maneira, toda vez que uma relação  $r$  for atualizada, a atualização deve se propagar a todos os *sites* que contém réplicas de  $r$  [Korth, 1999].

### 2.1.2 Fragmentação

Considerando uma relação  $r$  como fragmentada, então  $r$  é dividida em fragmentos  $r_1, r_2, \dots, r_m$ . Esses fragmentos contém informações que possibilitam a

reconstrução da relação  $r$ . Existem dois tipos de fragmentação: a fragmentação horizontal e a fragmentação vertical.

Na fragmentação horizontal a relação  $r$  é dividida separando as tuplas de  $r$  em dois ou mais fragmentos. Na fragmentação vertical a relação é dividida pela decomposição do esquema  $R$  da relação  $r$ , isto é, a fragmentação vertical de  $r(R)$  implica na definição de vários subconjuntos de atributos  $R1, R2, \dots, Rn$ .

## 2.2 Estruturas de Sistemas

Em um sistema distribuído, cada *site* possui seu próprio gerenciador local de transação, cuja função é garantir as propriedades de atomicidade, consistência, isolamento e durabilidade (ACID) das transações que são executadas localmente. As transações globais, são gerenciadas por um gerente de transações globais. Cada *site* do sistema contém dois subsistemas:

- **Gerenciador de Transações:** as transações que mantêm acesso aos dados locais armazenados no *site* local, são administradas pelo Gerenciador de Transações. Essas transações podem ser uma transação local e somente será processada naquele *site* ou, parte de uma transação global, uma transação que é processada em diversos *sites*. O gerenciador de transação é responsável por:
  - Manter um *log* para propósitos de recuperação;
  - Participar de um esquema de controle de concorrência adequado para a coordenação da execução de transações concorrentes em um mesmo *site*.
- **Coordenador de Transações:** é responsável pela coordenação da execução de várias transações, tanto a nível local quanto global, que são iniciadas naquele *site*. É responsável por:
  - Dar início à execução da transação;
  - Dividir uma transação em um número de subtransações e distribuir essas subtransações pelos *sites* apropriados para execução;
  - Coordenar o fim das transações, que pode resultar na efetivação ou na interrupção de todos os *sites* [Korth, 1999].

## 2.3 Tipos de Falhas no Sistema

Devido à complexidade dos equipamentos e programas modernos, podem ocorrer falhas tanto de “*hardware*” quanto de “*software*”. Essas falhas comprometem a integridade do Banco de Dados (BD). O SGBD deve incorporar mecanismos que garantam sua integridade, com isso, o SGBD pode ser mantido em operação por longos períodos de tempo, sendo que, pode ser interrompido por curtos intervalos para que os mecanismos de controle de recuperação, sanem as inconsistências geradas por pequenas falhas.

A única forma que o SGBD tem para se proteger contra falhas, é criar e manter uma certa redundância de dados. Agindo assim, quando uma parte do BD for danificada, sua “cópia” pode ser ativada para recuperar os dados que foram perdidos e restabelecer a operação normal. O SGBD deve manter também um histórico das operações efetuadas sobre o banco de dados, 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, as transações executadas entre o instante de criação da “cópia” e o momento atual serão perdidas[Casanova,1985].

Um sistema distribuído pode sofrer os mesmos tipos de falhas que afligem um sistema centralizado. Porém, há tipos de falhas adicionais que precisam ser tratadas em um ambiente distribuído, como:

- Falha em um *site*,
- Perda de mensagens;
- Falha de comunicação;
- Problemas de particionamento da rede.

A perda de mensagens é comum em sistemas distribuídos, que utilizam protocolos de controle de transmissão como o TCP/IP para o tratamento desse tipo de falha.

As falhas de comunicação e particionamento da rede estão diretamente ligados à forma como os *sites* de um sistema distribuído estão conectados. Se uma ligação falha, as mensagens que deveriam ser transmitidas por meio dela deverão ser roteadas utilizando uma nova rota para se chegar ao *site* destino. Entretanto, uma falha em apenas um *site* pode particionar a rede, já que cada *site* precisa de, no máximo, duas

ligações para acessar os outros *sites* e, se ocorrer falha em um *site* central os demais *sites* serão desconectados do sistema.

Um sistema distribuído precisa detectar falhas, reconfigurar o sistema e recuperar a situação original durante o retorno de um processo que tenha sido interrompido. As falhas são tratadas de formas diferentes. Quando ocorre a falha perda de mensagens, as mensagens são retransmitidas, porém a retransmissão repetida de uma mensagem, sem que haja uma mensagem de reconhecimento, pode ocasionar a perda de comunicação. Falhas nesse sentido sugerem particionamento da rede.

Geralmente não há certeza se ocorreu uma falha na comunicação entre os *sites* ou se ocorreu um particionamento da rede, o sistema detecta a falha, mas não consegue identificar qual o tipo de falha. Suponhamos que o *site* S1 não consiga se comunicar com o *site* S2, pode ser que o S2 esteja com problemas, ou a ligação entre o S1 e S2 não esteja funcionando, ocorreu então, o que se chama de particionamento da rede. Se o S2 identificar uma falha, ele iniciará um procedimento para a reconfiguração do sistema:

- Se existem dados que estão replicados no *site* S2 e não estão funcionando, o catálogo deverá ser atualizado para que as consultas não solicitem acesso à cópia desse *site*,
- Se no *site* com problemas (S2) houver transações ativas no momento da falha, essas transações devem ser abortadas, para que não gerem bloqueios em *sites* que estão ativos;
- Se o *site* que não está funcionando (S2) é um servidor de algum subsistema, deverá ocorrer uma eleição para determinar um novo servidor [Molina, 2001].

Quando um *site* volta a integrar a rede, é necessário que um procedimento seja executado para a atualização das tabelas do sistema, de modo a refletir as alterações sofridas enquanto ele esteve fora da rede. Se o *site* possui réplicas de itens de dados, é preciso que ele obtenha os valores atualizados desses itens e que se possa garantir que ele passe a receber as atualizações futuras, garantindo desta forma, a confiabilidade dos dados.

Os mecanismos de confiabilidade são implementados seguindo os algoritmos de controle de concorrência, levando-se em conta também as localizações dos dados, pois a

existência de cópias duplicadas dos dados é uma proteção para manter a confiabilidade dos dados.

No capítulo 4 será discutido com mais detalhes a questão da confiabilidade em banco de dados distribuídos, onde serão esmiuçados os protocolos que tratam os problemas de falhas e as formas de corrigi-las, para manter o banco de dados sempre em um estado confiável.

## Capítulo 3

### Recuperação em Banco de Dados

O sistema de recuperação em um banco de dados é responsável pela restauração do banco de dados para um estado consistente que havia antes de ocorrer falhas. Em outras palavras, o modo de afirmar que o banco de dados de fato é recuperável, é garantir que toda porção de informação que ele contém, possa ser reconstruída a partir de alguma outra informação armazenada de modo redundante, em algum outro lugar do sistema.

Uma parte integrante de um sistema de banco de dados é o esquema de recuperação, utilizado na restauração do banco para um estado consistente que havia antes de ocorrer uma determinada falha.

Neste capítulo serão destacados os tipos de falhas que podem ocorrer em um banco de dados centralizado e distribuído, os tipos de armazenamento que podemos ter com esse banco de dados e o que eles causam quando temos uma falha. Serão apresentados os métodos de recuperação utilizados quando ocorrem falhas em um banco de dados centralizado, e o que cada um desses métodos em particular pode fazer para recuperar o banco de dados tornando-o novamente confiável, ficando reservado o capítulo 4 para a exposição dos protocolos de confiabilidade em bancos de dados distribuídos.

### 3. Tipos de Falhas

Vários tipos de falhas podem ocorrer em um sistema, as mais simples de tratar são as falhas que não resultam em perdas de informação, porém, as mais difíceis são as falhas que resultam em perdas de informações.

Levando-se em conta os seguintes tipos de falhas, teremos:

- **Falha de Transação:** dois tipos de erros podem ocasionar uma falha de transação:



- **Erro lógico:** a transação não consegue continuar com a execução normal devido a algum acontecimento interno, do tipo, entrada de dados inadequados, um dado que não foi encontrado, ou *overflow*.
- **Erro de sistema:** a transação não consegue prosseguir porque o sistema entrou em um estado inadequado, isto é, o processo foi interrompido. Mais tarde, porém, essa mesma transação poderá ser reexecutada.
- **Queda do Sistema:** um mal funcionamento de hardware ou um bug de software de banco de dados ou um problema no sistema operacional, causou a perda do conteúdo que estava no armazenamento volátil e fez com que o processamento da transação parasse. Esses erros fazem com que o sistema pare mas não altera o conteúdo do armazenamento não-volátil. Os erros em referência são conhecidos como: condição falhar-parar [Date, 2001].
- **Falha de disco:** se ocorrer uma quebra do cabeçote ou uma falha durante uma operação de transferência de dados, um bloco do disco poderá perder seu conteúdo .

Seguem abaixo alguns fatores que devem ser levados em conta, na recuperação das falhas:

- É necessário identificar os modos de falha possíveis, dos equipamentos que serão utilizados para armazenar os dados;
- Como essas possíveis falhas irão afetar o conteúdo do banco de dados.

Estudando esses fatores, foram desenvolvidos algoritmos conhecidos como algoritmos de recuperação. Esses algoritmos estão divididos em duas partes: na primeira parte, são executadas ações durante o processamento normal da transação, a fim de garantir que haja informação suficiente para a recuperação de falhas se houverem, enquanto que na segunda parte são executadas ações em seguida à falha, recuperando assim, o conteúdo do banco de dados para um estado que assegure sua consistência, atomicidade, durabilidade e isolamento da transação [Korth,1999].

### **3.1 Tipos de Armazenamento**

Os vários tipos de dados no banco de dados, podem ser armazenados e sofrerem acesso em diferentes meios de armazenamento. Os vários meios de armazenamento são distinguidos por sua velocidade relativa, capacidade e resistência à falha.

#### **3.1.1 Armazenamento Volátil**

As informações que estão guardadas nesse tipo de armazenamento, não sobrevive a quedas que possam ocorrer no sistema. Um exemplo desse tipo de armazenamento pode ser a memória principal e a memória cache, porém, o acesso a esse armazenamento é extremamente rápido.

#### **3.1.2 Armazenamento Não-Volátil**

As informações residentes nesse tipo de armazenamento, sobrevive a quedas que ocorram no sistema. Um exemplo, são os discos e fitas magnéticas. Os discos estão sendo utilizados para armazenamento on-line e as fitas para armazenamento de arquivo, porém ambos estão sujeitos a falhas (quebra de cabeçote por exemplo) que resultaria em perda da informação. O acesso ao armazenamento não-volátil é mais lento que ao armazenamento volátil [Korth, 1999].

#### **3.1.3 Armazenamento Estável**

A informação que reside em um armazenamento estável quase nunca é perdida. A utilização de um armazenamento estável, consiste em replicar a informação em vários meios de armazenamento não-volátil (mais usado discos), com modos possíveis de falhas independentes, e controlar a atualização das informações, e com isso, garantir que uma eventual falha que ocorra durante a transferência de dados não prejudique as informações.

Sistemas RAID garantem que a falha de um único disco não resulte em perda de dados. A forma mais simples e mais rápida de RAID é o disco espelhado, que mantém duas cópias de cada bloco em discos separados. Sistemas mais seguros mantêm

uma cópia de cada bloco de armazenamento estável em um sistema remoto, enviando-a por uma rede de computadores, além de armazenar o bloco em um sistema de disco local. Os blocos são enviados ao sistema remoto e conseqüentemente para o armazenamento local, com isso, uma vez completas as operações de saída, essa saída nunca é perdida.

### 3.2 Recuperação e Atomicidade

Sempre que ocorrerem falhas durante a execução de uma transação  $T$ , os valores das informações que estavam residentes nos blocos de buffer foram perdidos, com isso, pode-se executar duas operações: reexecutar ou não a transação  $T$ ; das duas formas o banco de dados fica no estado de inconsistência, pelo fato de ter sido modificado sem saber se a transação foi efetivada ou não.

Para atingir a propriedade de atomicidade em um banco de dados, deve-se usar algumas formas de recuperação de dados, descritas a seguir.

#### 3.2.1 Paginação *Shadow*

O banco de dados é particionado em um número de blocos de comprimento fixo, chamados de página. Cada página recebe uma numeração que pode ir de 1 até  $n$ , contudo, há necessidade de encontrar para um  $i$  qualquer a  $i$ -ésima página do banco de dados. Para atingir esse propósito é usada uma *tabela de página* que tem  $n$  entradas – uma para cada página do banco de dados, e, cada entrada contém um ponteiro para uma página no disco. A primeira entrada contém um ponteiro para a primeira página do banco de dados, a segunda entrada aponta para a segunda página e assim por diante.

A idéia básica da técnica de paginação *shadow* (sombra), é manter duas tabelas de página durante o processamento da transação: a tabela de página atual e a tabela de página *shadow*. Quando a transação inicia, ambas as tabelas são idênticas. A tabela de páginas *shadow* nunca é alterada durante a execução da transação, já a tabela de página atual, deve ser alterada quando a transação processa uma operação de escrita (*write*). Todas as operações de *input* (entrada) e *output* (saída) em memória principal, usam a tabela de páginas atuais para localizar páginas do banco de dados no disco. Quando a transação é parcialmente efetivada, a tabela e páginas *shadow* é descartada e a tabela

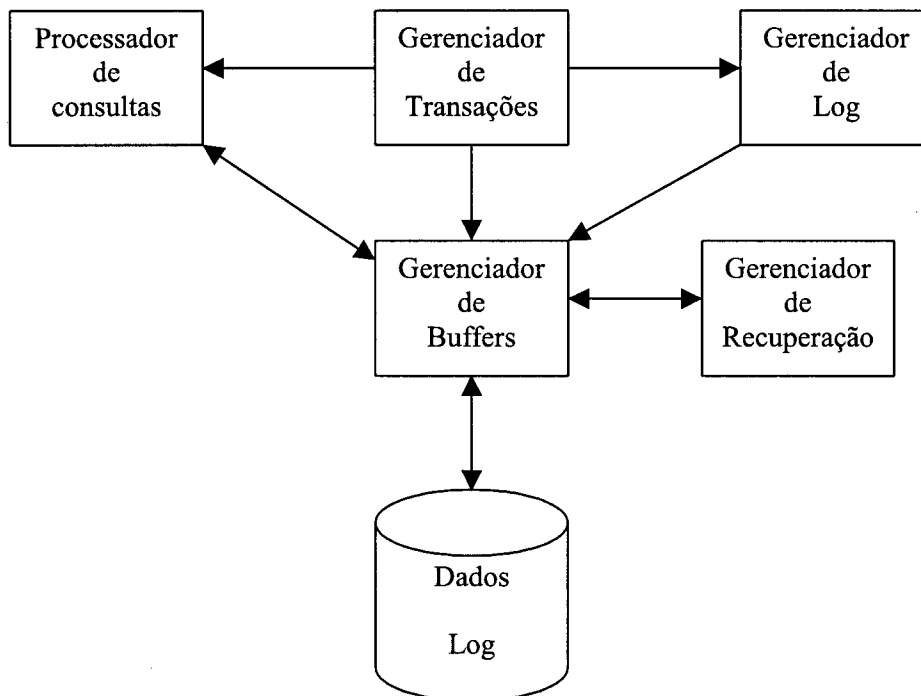
corrente torna-se a nova tabela de páginas, porém se a transação for abortada, a tabela de página atual é descartada [Finger e Ferreira, 2000] .

### 3.2.2 Gerenciador de Transações e o Gerenciador de Log

Assegurar que as transações serão executadas corretamente é o trabalho do *gerenciador de transações*, um subsistema que executa as seguintes funções:

1. Enviar sinais ao gerenciador de log para que as informações necessárias sejam armazenadas sob a forma de *registros de log*,
2. Assegurar que as transações concorrentes não irão interferir umas nas outras, de forma a produzir erros.

A figura 3.1 apresenta o funcionamento do gerenciador de log e gerenciador de transações.



**Figura 3.1. O gerenciador de Log e o gerenciador de Transações. [Molina,2001]**

O gerenciador de transações envia mensagens sobre ações de transações ao gerenciador de log; ao gerenciador de buffers envia mensagens informando quando é possível ou quando é necessário copiar o buffer de volta no disco; ao processador de

consultas, para que possa executar as operações no banco de dados que constituem a transação.

O gerenciador de log mantém o log e trabalha com o gerenciador de buffers, pois o espaço para o log aparece inicialmente em buffers de memória principal e em certos momentos há necessidade de copiar esses buffers para o disco.

Quando ocorre uma pane, o gerenciador de recuperação é ativado, examinará o log e o utilizará para reparar os dados, e o acesso ao disco sempre será feito através do gerenciador de buffers.

### 3.2.2.1 Registro de Log

Para uma transação ler um elemento (dado) do banco de dados, esse elemento deve primeiro ser trazido para um buffer da memória principal, isto é, se ele ainda não estiver lá, desta forma o conteúdo do buffer pode ser lido pela transação. A gravação de um novo valor para um elemento de banco de dados primeiro é criado no banco e somente depois é feita a cópia para o buffer.

A seguir, algumas primitivas que serão utilizadas nos algoritmos de registro de log:

1. *Input(X)*: o bloco de disco contendo o elemento do banco de dados (X) será copiado para um buffer da memória;
2. *Read(X, t)*: o conteúdo do elemento X será copiado para a variável local  $t$ ;
3. *Write(X, t)*: o conteúdo da variável local  $t$  será copiado para o elemento X no Buffer;
4. *Output(X)*: copia o buffer que contém o elemento X para o disco.

A instrução *Read* e *Write* são emitidas por transações e *Input* e *Output* são emitidos pelo gerenciador de transações.

Vamos demonstrar através do exemplo a seguir, como seria o relacionamento entre as operações primitivas e uma transação. Consideremos um banco de dados que tem dois elementos A e B, com os valores  $A = B = 8$ . Os valores das cópias da memória e do disco de A e B da variável local  $t$  no espaço de endereços da transação  $T$  estão indicados para cada etapa [Molina,2001].

Ação	t	Mem A	Mem B	Disco A	Disco B
<b>Read(A, t)</b>	8	8		8	8
<b>t := t * 2</b>	16	8		8	8
<b>Write(A, t)</b>	16	16		8	8
<b>Read(B, t)</b>	8	16	8	8	8
<b>t := t * 2</b>	16	16	8	8	8
<b>Write(B, t)</b>	16	16	16	8	8
<b>Output(A)</b>	16	16	16	16	8
<b>Output(B)</b>	16	16	16	16	16

Tabela 3.1. Etapas de uma transação e seu efeito sobre a memória e o disco. [Molina,2001].

Na primeira etapa,  $T$  lê A que gera um comando  $Input(A)$  para o gerenciador de buffers, isto é, se o bloco de A ainda não estiver em um buffer, executa também a cópia do valor de A através do comando  $Read$  na variável local  $t$  do espaço de endereços de  $T$ . Na segunda etapa,  $t$  é duplicado e não tem nenhum efeito sobre A, tanto em buffer como em disco. Na terceira etapa,  $t$  é gravado em A no buffer, não afetando A em disco. As três etapas seguintes fazem o mesmo para B. Nas últimas etapas A e B são copiados para o disco.

Se todas as etapas forem executadas, o banco de dados estará no estado de consistente. Porém, se ocorrer um erro do sistema antes do  $Output(A)$  ser executado, não haverá nenhum efeito sobre o banco de dados armazenado em disco, desta forma será como se  $T$  nunca tivesse sido executado e a consistência do banco de dados será preservada. Contudo, se houver um erro do sistema após  $Output(A)$  e antes do  $Output(B)$ , o banco se encontrará no estado de inconsistente.

Um log é uma seqüência de registro de log, cada um informando algo sobre o que alguma transação fez. As ações de várias transações podem ser “intercaladas” de tal forma que uma etapa de uma transação poderá ser executada e ter seu efeito anotado, e o mesmo ocorre para uma etapa de outra transação e em seguida para uma segunda etapa da primeira transação ou uma etapa de uma terceira transação e assim por diante [Molina, 2001].

### 3.2.2.2 Registro de Log de Desfazer

Sempre que ocorre uma pane no sistema, o log será consultado para que as transações existentes quando ocorreu a pane, possam ser refeitas. O log pode ser usado em conjunto com um arquivo de armazenamento, quando ocorrer uma falha de mídia que danifique o disco que contenha o arquivo de log.

Para consertar o efeito de uma pane, algumas transações terão seus efeitos refeitos e os novos valores que elas gravaram no banco de dados serão gravados novamente e, outras transações terão seus efeitos desfeitos, com isso, o banco de dados será restaurado, dando a impressão que as transações nunca foram executadas.

Esse registro de log é chamado de registro de log de desfazer, isto é, se ele não estiver absolutamente certo de que os efeitos de uma transação foram completados e o disco foi atualizado, qualquer alteração no banco de dados que a transação tenha feito, será desfeita e o banco de dados voltará a ficar no estado que se encontrava antes da transação.

O *gerenciador de log* tem a tarefa de registrar no log cada evento à medida que as transações estão sendo executadas. Um bloco do log é preenchido com registros de log, cada um representando um desses eventos. Os blocos de log são gravados inicialmente na memória principal e somente depois no armazenamento não volátil.

As formas de registro de log utilizadas em um registro de log:

- $\langle \textit{Start } T \rangle$  - esse registro indica que a transação  $T$  teve início;
- $\langle \textit{Commit } T \rangle$  - a transação  $T$  foi concluída com sucesso e não fará mais nenhuma modificação nos elementos do banco de dados, sendo que as mudanças feitas por essa transação devem aparecer em disco;
- $\langle \textit{Abort } T \rangle$  - a transação não foi completada com sucesso e, com isso nenhuma mudança que essa transação tenha efetuado poderá ser copiada para o disco. A função do gerenciador de transações é garantir que essas mudanças não apareçam em disco, ou que o seu efeito no disco seja cancelado quando ocorrer um  $\langle \textit{Abort } T \rangle$  [Molina, 2001].

Quando é utilizado o log de desfazer, o tipo de registro de log usado, é o *registro de atualização*, que é uma tripla  $\langle T, X, \nu \rangle$ , cujo significado é: a transação  $T$  mudou o elemento do banco de dados  $X$  e seu antigo valor era  $\nu$ . O log de desfazer não registra o novo valor de um elemento do banco de dados, somente registra o valor antigo. Caso a

recuperação seja necessária a única ação do gerenciador de recuperação será cancelar o possível efeito de uma transação em disco, restaurando o valor antigo.

Existem duas regras que são utilizadas junto com o log de desfazer, que permitem a recuperação de uma falha do sistema, a saber:

1. se a transação modificar o elemento  $X$  do banco de dados, então o registro de log na forma  $\langle T, X, v \rangle$  deve ser gravado em disco *antes* do novo valor de  $X$ ;
2. se uma transação é consolidada (efetuada), seu registro de log que contém *Commit*, deve ser gravado em disco somente *depois* que todos os elementos do banco de dados que sofreram modificações pela transação, estejam gravados em disco.

Resumindo as duas regras, teremos a seguinte seqüência de ações [Molina,2001]:

1. os registros de log que indicam elementos do banco de dados alterados;
2. os próprios elementos do banco de dados alterados;
3. o registro de log *Commit*.

O gerenciador de log executa um comando que esvazia o log e ao mesmo tempo ordena ao gerenciador de buffers para copiar os arquivos de log para o disco. Esse comando utilizado pelo gerenciador de log é o *Flush Log*.

Etapa	Ação	t	Mem A	Mem B	Disco A	Disco B	Log
1)							$\langle Start T \rangle$
2)	Read(A, t)	8	8		8	8	
3)	$t := t * 2$	16	8		8	8	
4)	Write(A, t)	16	16		8	8	$\langle T, A, 8 \rangle$
5)	Read(B, t)	8	16	8	8	8	
6)	$t := t * 2$	16	16	8	8	8	
7)	Write(B, t)	16	16	16	8	8	$\langle T, B, 8 \rangle$
8)	Flush Log						
9)	Output(A)	16	16	16	16	8	
10)	Output(B)	16	16	16	16	16	
11)							$\langle Commit T \rangle$
12)	Flush Log						

Tabela 3.2. Ações e entradas de log em uma transação [Molina,2001]

Analisando a tabela acima teremos:

- 1) a primeira ação a ser executada, é gravar no log o registro que marca o início da transação ( $\langle Start T \rangle$ );



- 2) na linha (2) a leitura do elemento A e o armazenamento na variável local  $\zeta$ ;
- 3) na linha (3) a alteração do conteúdo da variável  $\zeta$  que não afeta o conteúdo da memória A e nem o conteúdo do banco A;
- 4) não é armazenada nenhuma entrada de log para as linhas (2) e (3), porque não afeta o conteúdo do banco de dados;
- 5) na linha (4) ocorre a gravação do novo valor para o elemento A, essa mudança é refletida no log que armazena o registro  $\langle T, A, \delta \rangle$ , informando que o elemento A foi alterado por  $T$  e o antigo valor é  $\delta$ ;
- 6) as linhas (5) a (7) fazem o mesmo procedimento para o elemento B;
- 7) na etapa (8) o log é esvaziado e arquivado em disco;
- 8) nas etapas (9) e (10) o novo valor é copiado para o disco. Essas etapas são solicitadas pelo gerenciador de transações para que a transação  $T$  possa ser efetivada;
- 9) na linha (11) a transação é efetivada e o registro de  $\langle Commit T \rangle$  é armazenado no arquivo de log;
- 10) o arquivo de log é novamente esvaziado, para que fique armazenado em disco que a transação foi consolidada [Molina, 2001].

### 3.2.2.2.1 Recuperando o Banco de Dados utilizando o Registro de log de Desfazer

Supondo que ocorra uma pane no sistema, as transações não foram executadas de forma atômica e o banco de dados pode se encontrar em um estado de inconsistência. Nesse momento o gerenciador de recuperação entra em ação para efetuar a recuperação do banco de dados usando o arquivo de log. A primeira tarefa do gerenciador de recuperação é dividir as transações em: transações que foram efetivadas e as que não foram efetivadas.

Se houver um registro de log com a informação de  $\langle Commit T \rangle$ , todas as mudanças feitas pela transação  $T$  foram gravadas em disco, conclui-se então que a transação  $T$  não poderia ter deixado o banco de dados em um estado inconsistente. Porém, se for encontrado um registro de  $\langle Start T \rangle$  no log, mas não for encontrado um registro de  $\langle Commit T \rangle$ , pode ser que as mudanças efetuadas não tenham sido gravadas no disco antes da pane. Desta forma, a transação  $T$  é incompleta e deve ser

desfeita. Se  $T$  alterou o valor de  $X$  em disco antes da pane, então haverá um registro  $\langle T, X, v \rangle$  no log, registro esse, que foi copiado para o disco antes da pane.

O gerenciador de recuperação percorre o log do fim para o início. A medida que vai varrendo o log o mesmo terá uma visão de todas as transações, ou seja, quais tem o registro de  $\langle Commit T \rangle$  e quais tem o registro de  $\langle Abort T \rangle$ , além de encontrar também, um registro  $\langle T, X, v \rangle$ , desta forma [Molina, 2001]:

- se uma determinada transação  $T$  contém o registro de *Commit* armazenado no log, o gerenciador de recuperação não fará nada, porque a transação foi consolidada e não deve ser desfeita;
- porém, se  $T$  for uma transação incompleta ou uma transação abortada, o gerenciador de transação deve alterar o valor de  $X$  no banco de dados para  $v$ ;
- após efetuar as mudanças necessárias o gerenciador de recuperação deve armazenar um registro de log contendo um  $\langle Abort T \rangle$  para cada transação incompleta e depois esvaziar o arquivo de log.

Utilizando a tabela anterior 3.2, supondo que ocorra uma pane do sistema em diversos momentos distintos, teremos:

Momento de pane	Registro de Log em disco	Ação
Após a etapa (12)	$\langle Commit T \rangle$ gravado	Quando o processo de recuperação for executado, os registros referentes a transação $T$ serão ignorados.
Entre as etapas (11) e (12)	$\langle Commit T \rangle$ gravado $\langle Commit T \rangle$ não gravado	Igual ao processo anterior. A transação $T$ está incompleta; O processo de recuperação percorre o log e encontra armazenado o registro $\langle T, B, 8 \rangle$ , o valor 8 será armazenado no elemento $B$ em disco. Continua percorrendo o log e encontrará o registro $\langle T, A, 8 \rangle$ e tornará o valor de $A$ igual a 8; Será gravado o registro $\langle Abort T \rangle$ no log e este será esvaziado.
Entre as etapas (10) e (11)	$\langle Commit T \rangle$ não gravado	A transação $T$ está incompleta e precisa ser desfeita como o processo anterior.
Entre as etapas (8) e (10)	$\langle Commit T \rangle$ não gravado	A transação $T$ está incompleta e precisa ser desfeita como o processo anterior. A única diferença é que o novo valor para o

		elemento A ou B pode não ter alcançado o disco, mas da mesma forma o valor antigo será armazenado nos elementos correspondentes garantindo a consistência do banco de dados.
Antes da etapa (8)	Não é certo se $\langle Commit \rangle$ foi ou não gravado	O gerenciador executa a recuperação como no caso (2).

Tabela 3.3. Conclusões consideradas utilizando a tabela anterior [Molina, 2001].

### 3.2.2.2.2 Pontos de Verificação com o Registro de Log de Desfazer

Sempre que o Gerenciador de Recuperação é ativado, examina o log por inteiro. Se a transação tem seu registro de *Commit* gravado em disco, isto quer dizer que os registros de log dessa transação não são mais necessários durante a recuperação, com isso, podem ser eliminados.

Se os registros de log fossem eliminados logo após uma transação ter sido efetivada, os registros de log de alguma transação que ainda estiver ativa podem ser perdidos e, caso ocorra uma falha no sistema, a recuperação não seria executada. A solução para esse problema é utilizar os pontos de verificação simples ou pontos de verificação não quiescentes (*checkpoint*) junto com os registros de log. Ao utilizarmos os pontos de verificação simples, teremos:

- a não aceitação de novas transações;
- esperar até que todas as transações que estiverem ativas sejam efetivadas e gravem um registro de *Commit* ou *Abort* no log;
- esvaziar o log para o disco;
- gravar o registro de log  $\langle Ckpt \rangle$  e esvaziá-lo o log novamente;
- voltar a aceitar transações.

Durante uma recuperação, o log é lido a partir do final para a frente, identificando as transações que estão incompletas. Porém quando é encontrado um registro de  $\langle Ckpt \rangle$ , todas as transações incompletas já foram vistas, não havendo desta forma, necessidade de varrer o log antes do  $\langle Ckpt \rangle$ , podendo este ser eliminado ou sobrescrito e a restauração do banco de dados estará completa.

A diferença de trabalhar com os pontos de verificação não quiescentes é que as transações podem entrar no sistema durante o ponto de verificação. Desta forma teremos:

- um registro de log  $\langle \text{Start Ckpt } (T_1, \dots, T_k) \rangle$  gravado e em seguida o log sendo esvaziado, isto é, gravado em disco.  $T_1, \dots, T_k$ ; estas são transações que ainda não foram consolidadas e suas mudanças não foram gravadas em disco;
- que esperar todas as transações de  $T_1, \dots, T_k$  estiverem consolidadas ou abortadas, mesmo assim novas transações podem ser inicializadas;
- quando todas as transações de  $T_1, \dots, T_k$  estiverem completas, um registro de log será gravado  $\langle \text{End Ckpt} \rangle$  e o log será esvaziado.

Quando o log for varrido a partir do fim, poderá ser encontrado:

- um registro de  $\langle \text{End Ckpt} \rangle$ . Isto significa dizer que todas as transações incompletas começaram depois do registro  $\langle \text{Start Ckpt } (T_1, \dots, T_k) \rangle$  anterior. A varredura será feita para trás até o próximo registro de *Start Ckpt* onde serão encontradas as transações incompletas. Com isso, o log anterior será descartado;
- ou, um registro de  $\langle \text{Start Ckpt } (T_1, \dots, T_k) \rangle$ . Isto quer dizer que a pane ocorreu durante o ponto de verificação. As transações que são consideradas incompletas, são aquelas que foram encontradas na varredura feita de trás para frente antes de alcançar o registro de *Start Ckpt* e as transações dentre  $T_1, \dots, T_k$  que não se completaram antes da pane.

Como regra geral, após ser encontrado, um registro  $\langle \text{End Ckpt} \rangle$  é gravado em disco, o log mais antigo que o registro *Start Ckpt* anterior pode ser eliminado [Molina, 2001].

### 3.2.2.3 Registro de Log de Refazer

Enquanto que o efeito de um registro de log de desfazer é cancelar o efeito das transações que estão incompletas e ignorar as transações que estão consolidadas, o registro de log de refazer ignora as transações incompletas e refaz as mudanças das transações que foram efetivadas. O registro de log de refazer exige que o registro de

*Commit* seja gravado em disco antes que qualquer valor que foi alterado seja armazenado em disco.

O registro de log de refazer representa mudanças em elementos do banco de dados através de um registro que fornece um novo valor. Esses registros são idênticos aos registros do log de desfazer,  $\langle T, X, \nu \rangle$ , porém tem outro significado: uma transação  $T$  gravou um novo valor  $\nu$  para o elemento do banco de dados  $X$ .

A regra do registro de log de refazer é conhecida como *regra de registro de log de gravação antecipada*. Desta forma:

- antes de ocorrer uma modificação em qualquer elemento do banco de dados  $X$  em disco, todos os registros de log que fazem parte da modificação em  $X$ , inclusive o registro de  $\langle T, X, \nu \rangle$  e o registro de  $\langle \text{Commit } T \rangle$ , devem aparecer gravados no disco.

Quando o registro de log de refazer está em uso, as seguintes etapas são cumpridas em ordem, para que ocorra a gravação em disco:

- primeiro, os registros de log que indicam elementos do banco de dados alterados;
- em seguida o registro de log *Commit* é gravado;
- e por último, os elementos do banco de dados que foram alterados.

A tabela 3.4 a seguir, mostra os passos a serem seguidos, quando se utiliza um registro de log de refazer [Molina, 2001].

Etapa	Ação	T	Mem A	Mem B	Disco A	Disco B	Log
1)							$\langle \text{Start } T \rangle$
2)	Read(A, t)	8	8		8	8	
3)	$t := t * 2$	16	8		8	8	
4)	Write(A, t)	16	16		8	8	$\langle T, A, 16 \rangle$
5)	Read(B, t)	8	16	8	8	8	
6)	$t := t * 2$	16	16	8	8	8	
7)	Write(B, t)	16	16	16	8	8	$\langle T, B, 16 \rangle$
8)							$\langle \text{Commit } T \rangle$
9)	Flush Log						
10)	Output(A)	16	16	16	16	8	
11)	Output(B)	16	16	16	16	16	

Tabela 3.4. As ações e as entradas de log, usando um registro de log de refazer. [Molina, 2001].

Nas etapas (4) e (7) os registros de log que refletem as mudanças, tem os novos valores de A e B armazenados. Em seguida, na etapa (8), aparece o registro de  $\langle \text{Commit } T \rangle$  e somente na etapa (9) o log é esvaziado, isto é, gravado em disco, e após isso, todos os registros de log que envolvem as mudanças na transação  $T$  são armazenados em disco. Na etapa (10) e (11) os novos valores de A e B poderão ser gravados em disco.

### 3.2.2.3.1 Recuperação utilizando o Registro de Log de Refazer

Em um registro de log de refazer, as transações incompletas são tratadas como se nunca tivessem ocorrido, enquanto que com as transações consolidadas os novos valores podem ser gravados novamente em disco, independentemente do fato de eles já estarem armazenados. Para se recuperar de uma pane no sistema utilizando o log de refazer é necessário que:

- sejam identificadas as transações consolidadas;
- o log seja varrido desde o início e para cada registro de log  $\langle T, X, \nu \rangle$  encontrado:
  - se  $T$  não for uma transação consolidada, então não faça nada;
  - se  $T$  for uma transação consolidada, o novo valor de  $\nu$  será gravado para o elemento do banco de dados  $X$ .
- seja gravado um registro de  $\langle \text{Abort } T \rangle$  para cada transação  $T$  incompleta e o log seja esvaziado.

Supondo que ocorra uma pane em alguns momentos distintos no sistema, a recuperação seria executada na seguinte seqüência:

- se a pane ocorrer após a etapa (9), o registro de  $\langle \text{Commit } T \rangle$  será esvaziado para o disco e conseqüentemente o sistema de recuperação identifica,  $T$  como uma transação consolidada. Ao percorrer o log do início para o fim, os registros de log cujos valores são  $\langle T, A, 16 \rangle$  e  $\langle T, B, 16 \rangle$ , fazem com que o sistema de recuperação, grave os novos valores para o elemento A e B;
- se a pane ocorrer entre as etapas (10) e (11) a gravação do elemento A será redundante, enquanto que a gravação do elemento B será essencial para garantir a consistência do banco de dados;

- se a pane ocorrer após a etapa (11), as duas gravações são redundantes, porém inofensivas;
- se a pane ocorrer entre as etapas (8) e (9), mesmo que o registro de  $\langle Commit T \rangle$  já tenha sido gravado no log, mas não há certeza de que o registro de log tenha sido gravado em disco, é necessário então que ocorra a recuperação do banco de dados como foi realizado com a etapa (9);
- se ocorrer a pane antes da etapa (8), então há certeza que o registro de  $\langle Commit T \rangle$  não foi gravado em disco, deste modo  $T$  é considerado como uma transação incompleta e um registro de  $\langle Abort T \rangle$  é armazenado no log [Molina, 2001].

### 3.2.2.3.2 Utilizando o Registro de Log de Refazer junto com Pontos de Verificação

Independentemente do ponto de verificação permitir ou não que as transações tenham início no meio do mesmo, é necessário gravar em disco, todos os elementos do banco de dados que foram modificados por transações que foram efetivadas mas que ainda não foram armazenadas em disco. O gerenciador executa essa ação a partir da verificação de quais buffers estão *sujos* (buffers alterados e não gravados em disco), e também quais transações modificaram quais buffers.

As etapas a serem usadas para executar um ponto de verificação junto com o log de refazer é:

- armazenar um registro de log  $\langle Start Ckpt (T_1, \dots, T_k) \rangle$ , onde  $T_1, \dots, T_k$  são todas as transações não efetivadas e esvaziar o log em seguida;
- armazenar em disco os elementos do banco de dados que foram gravados em buffers, mas não gravados em disco por transações que já foram consolidadas quando o registro *Start Ckpt* foi gravado no log;
- armazenar um registro  $\langle End Ckpt \rangle$  no log e esvaziar o log [Molina, 2001].

### 3.2.2.3.3 Recuperação utilizando o log de refazer e pontos de verificação

Dependendo do último registro de log ter armazenado um *Start* ou *End*, pode haver as seguintes situações:

- se antes de ocorrer uma pane, o último registro de ponto de verificação no log foi  $\langle \text{End Ckpt} \rangle$ , todo valor gravado por uma transação que foi efetivada antes do  $\langle \text{Start Ckpt } (T_1, \dots, T_k) \rangle$  teve suas alterações gravadas em disco, porém as transações que foram inicializadas após o início do ponto de verificação podem ter mudanças que ainda não foram transferidas para o disco, mesmo que essas transações tenham sido efetivadas;
- se o último registro de ponto de verificação no log, for um registro de  $\langle \text{Start Ckpt } (T_1, \dots, T_k) \rangle$ , pode ser, ou não, que as transações que foram efetivadas antes do início deste ponto de verificação, tenham suas mudanças gravadas em disco. Desta forma, é necessário pesquisar de volta até o registro  $\langle \text{End Ckpt} \rangle$  anterior, encontrar seu registro de  $\langle \text{Start Ckpt } (S_i, \dots, S_m) \rangle$  correspondente e refazer todas as transações efetivadas que começaram após esse  $\text{Start Ckpt}$  ou que estão entre as transações  $S_i$  [Molina, 2001].

#### 3.2.2.4 Registro de log de desfazer/refazer

Um registro de log de desfazer/refazer tem os mesmos registros de log que os outros tipos de log, porém com uma diferença, o registro de log de desfazer/refazer que é gravado possui quatro componentes:  $\langle T, X, v, w \rangle$ , que significa que a transação  $T$  mudou o valor do elemento do banco de dados. Seu valor anterior era  $v$ , e o valor atual é  $w$ , seguindo a regra:

- antes de qualquer elemento do banco de dados  $X$  ser modificado em disco, devido a qualquer alteração causada por alguma transação  $T$ , é necessário que o registro de atualização  $\langle T, X, v, w \rangle$  apareça em disco.

Os registros de log que estão armazenados, tem agora tanto o valor antigo quanto o novo valor de A e B, como é apresentado na tabela 3.5 a seguir:



Etapa	Ação	T	Mem A	Mem B	Disco A	Disco B	Log
1)							$\langle \text{Start } T \rangle$
2)	Read(A, t)	8	8		8	8	
3)	$t := t * 2$	16	8		8	8	
4)	Write(A, t)	16	16		8	8	$\langle T, A, 8, 16 \rangle$
5)	Read(B, t)	8	16	8	8	8	
6)	$t := t * 2$	16	16	8	8	8	
7)	Write(B, t)	16	16	16	8	8	$\langle T, B, 8, 16 \rangle$
8)	Flush Log						
9)	Output(A)	16	16	16	16	8	
10)							$\langle \text{Commit } T \rangle$
11)	Output(B)	16	16	16	16	16	

Tabela 3.5. As ações e as entradas de log usando um registro de log de desfazer/refazer. [Molina,2001]

### 3.2.2.4.1 Recuperação utilizando os registros de log de desfazer/refazer

Quando há necessidade de recuperar o banco de dados usando um log de desfazer/refazer, as informações que são necessárias, estão armazenadas nos registros de atualização para desfazer uma transação  $T$ , restaurando os valores antigos dos elementos do banco de dados que a transação alterou, ou refazendo as alterações executadas por ela, seguindo as normas:

- refazer as transações que estão efetivadas, na ordem da mais antiga para a mais recente;
- desfazer as transações que são consideradas como incompletas, na ordem da mais recente para a mais antiga.

Supondo que ocorra uma pane de sistema em vários pontos:

- se a pane ocorrer após o registro  $\langle \text{Commit } T \rangle$  ser transferido para o disco,  $T$  será considerada como uma transação efetivada e o valor 16 é gravado em A e B;
- se a pane ocorrer antes do registro  $\langle \text{Commit } T \rangle$  ser transferido para o disco,  $T$  será considerada como uma transação incompleta, e os valores antigos de A e B (8) serão gravados em disco [Molina, 2001].

### 3.2.2.4.2 Usando log de desfazer/refazer com pontos de verificação

Para utilizar o log de desfazer/refazer com pontos de verificação é necessário:

- gravar um registro  $\langle \text{Start Ckpt } (T_1, \dots, T_k) \rangle$  no log,  $T_1, \dots, T_k$  são todas as transações ativas, e o log será esvaziado posteriormente;
- gravar em disco todos os *buffers* que estão sujos, isto é, *buffers* que contém elementos que foram alterados;
- gravar um registro  $\langle \text{End Ckpt} \rangle$  no log, e após, esvaziar o log.

## 3.3 Arquivo de armazenamento

Os registros de log protegem o banco de dados contra falhas no sistema, com isso, somente os dados que estão armazenados na memória principal são perdidos. As falhas de disco são consideradas como as mais graves e envolvem a reconstrução do banco de dados a partir dos registros de log que estão armazenados em um outro disco, e, esses arquivos são conhecidos como *arquivos de armazenamento*. Para recuperar o banco de dados até um ponto mais recente, é necessário utilizar os registros de log que devem ser armazenados em um local remoto, junto com os *arquivos de armazenamento*.

Existem dois níveis de arquivos de armazenamento:

- *despejo completo*, onde todo o banco de dados é copiado;
- *despejo incremental*, onde apenas os elementos do banco de dados que sofreram alterações desde o despejo anterior, são copiados.

A restauração do banco de dados pode ser feita a partir de um despejo completo e de seus despejos incrementais subsequentes, desta forma, o despejo completo é copiado de volta no banco de dados e em seguida na ordem do mais antigo para o mais recente, são efetuadas as mudanças registradas nos despejos incrementais posteriores [Molina, 2001].

O registro de log de refazer ou de desfazer/refazer são os mais apropriados para serem utilizados junto com o arquivo de armazenamento. O processo de criar um arquivo de armazenamento pode ser feito de acordo com as etapas seguintes:

- 1) gravar no registro de log um  $\langle \text{Start Dump} \rangle$ ;
- 2) criar um ponto de verificação apropriado segundo o registro de log a ser utilizado;

- 3) executar um despejo completo ou incremental do(s) disco(s) de dado(s);
- 4) verificar se foi copiado para um local remoto e seguro uma parte suficiente do log para que sobreviva a uma falha de mídia do banco de dados;
- 5) gravar no registro de log um *<End Dump>*;
- 6) quando o despejo for concluído, é seguro descartar o log antes do início do ponto de verificação anterior ao que foi executado na etapa (2).

Se ocorrer uma falha de mídia, o banco de dados deve ser reconstruído a partir de um arquivo de armazenamento mais recente, de acordo com as etapas seguintes:

- 1) restaurar o banco de dados, utilizando um arquivo de armazenamento:
  - a) encontrar o arquivo de despejo completo mais recente e copiar o seu conteúdo para o banco de dados;
  - b) se houver despejos incrementais posteriores ao despejo completo, reconstruir o banco utilizando cada um deles, começando pelo mais antigo.
- 2) modificar o banco usando o log armazenado mais recente, e o método de recuperação mais apropriado para o método de log que está armazenado [Molina, 2001].

O computador como um sistema qualquer está sujeito a falhas, que podem ser desde uma queda de disco, falha de energia e erros de software, sendo que em qualquer uma dessas ocorrências, a informação será perdida. Porém, além de falhas de sistemas, existem as falhas de transação, que podem ocorrer por várias razões, por exemplo: violação de integridade ou impasses.

Todo sistema de banco de dados, possui um esquema de recuperação, que é responsável pela detecção de falhas, e também, um esquema de restauração do banco de dados para o estado anterior à ocorrência da falha. Quando ocorre uma falha, o banco de dados não se encontra mais em um estado consistente, isto é, o banco não pode mais refletir o estado verdadeiro das informações contidas nele. O esquema de recuperação é responsável em assegurar a atomicidade de uma transação.

A implementação de forma eficiente de um esquema de recuperação, exige que o número de escritas no banco de dados seja de forma minimizada, isto é, os registros de log podem ser mantidos inicialmente em buffer de log volátil, mas devem ser escritos em armazenamento estável sempre que possível [Korth,1999].

## Capítulo 4

### Confiabilidade em um Banco de Dados Distribuído

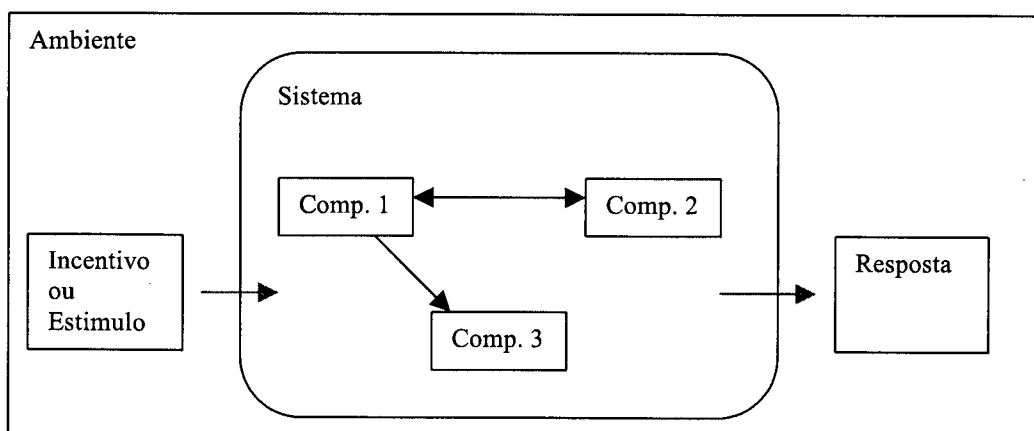
Somente a replicação de dados não é suficiente para garantir um banco de dados mais consistente. Dentro de um Sistema Gerenciador de Banco de Dados é necessária a implantação de diversos protocolos que exploram a distribuição e replicação dos dados, executando com isso, operações mais confiáveis.

Um SGBD distribuído confiável é um sistema que continua processando as requisições do usuário, mesmo que o sistema subjacente esteja no estado de incerto, isto é, mesmo que ocorram falhas, o SGBD continuará executando as transações sem violar a consistência do banco de dados [Casanova, 1985].

Confiabilidade pode ser definida como uma medida que será utilizada para que o sistema esteja em conformidade com as especificações a que foi submetido.

A confiabilidade de um SGBD distribuído, leva em conta a atomicidade e a durabilidade de transações.

No que se refere à confiabilidade, podemos definir sistema como um mecanismo que consiste na coleção de componentes que interage com seu ambiente respondendo a incentivos do ambiente com um padrão reconhecível de comportamento (Figura 4.1) [Ozsu,1999].



**Figura 4.1 Componentes de um ambiente Distribuído [Ozsu, 1999]**

Pode-se interpretar esta definição, como sendo o comportamento de resposta de um elemento de um sistema perante estímulos, externos e/ou internos, em um ambiente comum, onde o componente em questão deveria assumir um estado esperado.

O comportamento em questão, retrata uma resposta baseada em funções e métodos implementados, e todo e qualquer desvio comportamental pode ser considerado uma *falha*, visto que trata-se de um comportamento não esperado.

Cada componente do sistema é um sistema que faz parte do sistema geral. O ambiente de um componente é o sistema do qual ele é uma parte.

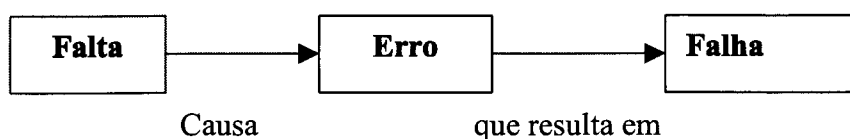
Neste capítulo será discutida a confiabilidade dos dados em um sistema de banco de dados distribuído, quais são os métodos utilizados para restaurar o banco tornando-o novamente confiável, e ainda, serão apresentado também os protocolos 2PC e 3PC, como eles trabalham e como lidam com falhas que ocorrem durante o seu processamento.

#### **4. Sistema, estado e falhas**

Em um sistema de banco de dados distribuído, o termo *falha*, refere-se às deficiências nos componentes que formam o sistema, isto é, o estado em que os componentes estão reunidos. Cada estado pelo qual o sistema de confiabilidade passa, é válido no momento em que o estado conhece sua especificação completamente. Porém, pode ocorrer um estado de não confiabilidade do sistema, isso é possível porque o sistema pode adquirir um estado interno que não consegue obedecer sua especificação [Ceri, 1984].

Transações que se encontram nesse estado, causam um fracasso eventual no sistema. São chamados de estados *internos errôneos*, a parte do estado que está incorreto é chamada de *erro* no sistema. Qualquer erro nos estados internos dos componentes de um sistema é chamada de *falta* do sistema. Assim, uma falta causa um erro que resulta em um fracasso de sistema [Ozsu, 1999].

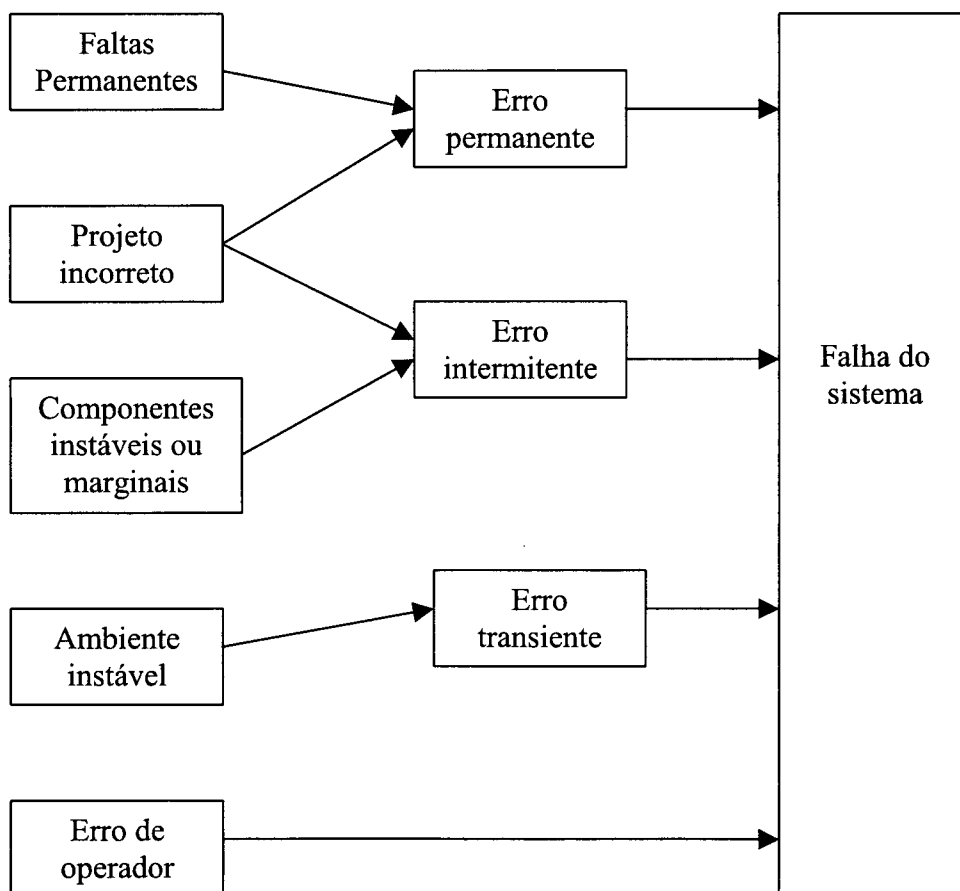
A figura a seguir 4.2 apresenta os estados pelo qual uma falta passa originando com isso uma falha no banco.



**Figura 4.2 Cadeia de Eventos que Conduzem a um Fracasso de Sistema [Ozsu, 1999].**

Os erros, faltas ou falhas, são diferenciados entre permanentes ou não permanentes. Uma falta permanente é chamada de “*hard faults*” [Ozsu,1999], que reflete uma mudança irreversível no comportamento do sistema. Faltas permanentes causam erros permanentes que resultam em falhas permanentes.

Os sistemas também passam por faltas intermitentes e transientes. Esses tipos de faltas, são caracterizadas pela recuperação que exige intervenções que consertam a falta, e, essas faltas são chamadas de “*soft faults*” [Ozsu,1999]; o que difere uma da outra é a confiabilidade do sistema que experimentou a falta [Siewiorek e Swarz,1982]. Siewiorek e Swarz classificam as fontes de faltas como é apresentada na figura 4.3 a seguir.



**Figura 4.3 Fontes de faltas de um Sistema [Siewiorek e Swarz,1982].**

Falhas de transação em um SGBDD podem ocorrer devido a um aborto de transações. Normalmente, uma média de 3% das transações são abortadas. As falhas de sistema ou *site*, ocorrem devido a problemas com o processador, memória principal, e outros fatores, sendo que o conteúdo da memória principal será perdido, mas o conteúdo dos dispositivos de armazenamento estável estarão seguros. Falhas de mídia ocorrem devido a falhas nos dispositivos de armazenamento secundário, onde os dados armazenados são perdidos. Falhas de comunicação podem ser ocasionadas devido a mensagens perdidas ou não entregues ou pelo particionamento da rede.

Estudos recentes indicam que a ocorrência de *soft faults* é significativamente mais alta que *hard faults* [Longbottom, 1980]. Gray [1987] menciona também que a maioria dos fracassos de *software* é transitente, sugerindo que uma descarga e um reinício podem ser suficientes para recuperar os dados sem qualquer necessidade de consertar o software.

A recuperação da informação que o sistema mantém, depende do método de executar as atualizações. Há duas possibilidades de executar as atualizações:

- Atualizações "*In-Place*": os valores dos itens de dados atualizados são fisicamente modificados no banco de dados estável.
- Atualizações "*Out-of-Place*": os valores dos itens de dados atualizados não são fisicamente modificados no banco de dados estável, mas mantidos como valores separados, sendo que, periodicamente esses valores são incorporados ao banco de dados estável.

#### **4.1 Particionamento de rede**

O problema do particionamento da rede é um aspecto de relevante importância, isto porque é algo até mesmo natural que uma rede seja dividida em dois segmentos independentes, e isto não deve ser uma restrição total ao funcionamento do sistema, mas mesmo assim deve ser considerada como uma falha aceitável.

Considerando-se que as duas sub-redes continuam trabalhando exatamente com as mesmas características e registros da rede original, com desvantagens quanto ao acesso a dados da outra rede e com vantagens, principalmente pelo número reduzido de participantes.

O problema da falta de informação pode ser suprido com uma alocação estratégica de réplicas, bem como o fator de integridade e consistência das réplicas basear-se em um sistema de apoio que trabalhe com a idéia de análise posterior da informação.

Características como a durabilidade e confiabilidade, são relativos pois dependerão em muito dos critérios impostos às sub-redes. Porém atomicidade, isolamento e consistência, permanecem inalterados.

Todos os protocolos devem trabalhar sob o enfoque pessimista, enfatizando a consistência do banco de dados, principalmente considerando-se que as partições deverão continuar trabalhando normalmente [Molina, 2001].

#### **4.1.1 Protocolo centralizado**

O protocolo centralizado baseia-se no controle de concorrência centralizado. O sistema para cada *site* pode ser de dois tipos: *site* primário e cópia primária. Sendo o acesso realizado pelo *site* primário, permanecem todas as regras de acesso, inclusive o bloqueio das tabelas. No entanto, quando trata-se de cópia primária, mais de um *site* pode oferecer a informação, porém, somente o *site* definido como primário é que receberá comandos de *write*, ou seja, poderá ser alterado.

#### **4.1.2 Protocolo baseado em votação**

A técnica de votação é amplamente difundida em sistemas de controle. A idéia é que todos os *sites* envolvidos votem perante uma questão, e se a maioria dos votos for a favor, a transação será executada. As votações poderão ocorrer para cada etapa do processo, seguindo algumas normas ou regras básicas [Banco de dados, 1]:

- a. uma regra não pode ser votada e abortada ao mesmo tempo;
- b. nenhum *Commit* ocorre sem uma soma de votos mínima;
- c. nenhum *abort* ocorre sem uma soma de votos mínima de *abort*.



### 4.1.3 Protocolo de controle de réplicas

A idéia de replicação tende a aumentar em muito a complexidade dos dados, isto se deve à necessidade de gerenciamento dos dados replicados. A idéia de replicação pode levar o sistema a estados de inconsistência e complicaria em muito a problemática da escrita da informação, ou seja, onde a informação ficará armazenada (*sites* primários).

Por outro lado, existem as vantagens relativas a menor concorrência para a leitura das informações, na necessidade de particionamento da rede e em questões de acesso múltiplo localizado, onde alguns *sites* lêem de um local e outros de um local predeterminado.

Um dos problemas da réplica é a necessidade de requerer que todas as cópias do banco de dados sejam mutuamente consistentes ao final de cada transação. A idéia central é a de que os comandos de escrita sejam executados em todas as cópias disponíveis e a transação seja finalizada. Sendo assim as cópias ficarão indisponibilizadas no instante de tempo em que estiverem sendo atualizadas [ Molina, 2001]

## 4.2 Recuperação e Controle de Dados

O relacionamento que existe entre a recuperação de falhas e o controle de dados é bidirecional. Inicialmente o controle de dados tem uma função determinante na recuperação de falhas, isto porque o esquema de recuperação baseia-se no modelo transacional, garantindo com isso que a propriedade de atomicidade seja a base dos procedimentos de recuperação: “transações confirmadas devem ter seus resultados mantidos, transações abortadas ou ativas no momento de uma falha de sistema devem ser abortadas pelo sistema” [Finger e Ferreira,2000].

O controle de concorrência fornece a informação ao Gerenciador de Recuperação de que os escalonamentos são estritos, isto é, para restabelecer a implementação são utilizadas imagens prévias. Desta forma, o log será percorrido no sentido do fim para o começo até ser encontrado o último valor do item de dados a ser refeito.

### 4.3 Protocolos de confiabilidade distribuídos

Da mesma forma que em protocolos de confiabilidade locais, as versões distribuídas apontam para manter a atomicidade e durabilidade de transações distribuídas que são executadas em cima de vários bancos de dados. Os protocolos enviam à execução distribuída o início-transação, leia, escreva, aborte, *Commit* e recupere comandos.

A diferença entre falhas locais e falhas distribuídas, é que ocorrendo um particionamento da rede, pode ser que algum nó fique no estado de bloqueio após o reinício de uma falha.

Quando ocorre uma falha em algum nó, ao ser reiniciado, este deve determinar o estado confirmado/abortado de todas as transações que estavam na lista de transações ativas quando a falha ocorreu. Para ocorrer uma confirmação de uma transação distribuída, é necessário que todos os nós participantes votem pela confirmação. Devido a essa necessidade, haverá um instante em que o nó não saberá qual é o estado de uma transação global, com isso, este nó se encontrará no estado *indefinido*.

Seguindo essa idéia e ocorrendo um particionamento da rede, o nó não sabe se a transação está confirmada ou bloqueada. Este nó não pode continuar o seu reinício enquanto a indefinição não for resolvida, neste momento, este nó se encontrará no estado de *bloqueado*.

Ao trabalhar com Confirmação Atômica de uma transação distribuída, todos os participantes devem atingir a mesma decisão, isto é, todos os participantes devem chegar à decisão de confirmação ou de aborto. Um desses participantes é considerado o *coordenador* da transação e os outros os *coordenados*, sendo que cada nó da rede possui um *log de transações distribuídas* que é mantido em memória secundária.

Cada nó deve votar em apenas uma das opções sobre a decisão de confirmar uma determinada transação distribuída: *Sim* ou *Não* e atingir uma de duas decisões: *Confirmação* ou *Aborto*, obedecendo as seguintes regras: [Finger e Ferreira,2000]

1. todos os participantes devem atingir a mesma decisão;
2. nenhum participante pode mudar de decisão a partir do momento que ela foi tomada;
3. a decisão de Confirmação só será atingida se todos os participantes votarem Sim;

4. se não ocorrer nenhuma falha e se todos os participantes votarem em Sim, a transação será confirmada;
5. se todas as falhas foram recuperadas e nenhuma nova falha ocorrer, então todos os participantes vão chegar a uma decisão.

Um nó encontra-se no estado de indefinido entre o instante que ele vota *Sim* e o instante em que ele recebe a decisão sobre confirmar ou abortar a transação, se nesse exato instante, ocorrer uma falha de comunicação, o nó estará bloqueado e somente poderá chegar a uma decisão, quando a falha de comunicação estiver reparada.

#### 4.3.1 Protocolo de *Commit Two-Phase* (2PC)

É um protocolo muito simples, que assegura o *Commit* de transações distribuídas. Estende os efeitos de um *Commit* local a transações distribuídas, e com isso, todos os locais envolvidos terão efeitos permanentes. Há vários momentos em que é necessária a sincronização entre dados locais:

Primeiro, dependendo do tipo de algoritmo de controle de concorrência que é usado, algum programa de controle pode não estar pronto para terminar uma transação. Por exemplo, se uma transação ler um valor de um artigo de dados, que é atualizado por uma outra transação que não tem *Commit*, o programa de controle associado, pode não querer dar um *Commit* no item anterior.

Claro que, os algoritmos de controle de concorrência, são rígidos para evitar abortos em cascata, não permitindo ler o valor que está sendo atualizado de um artigo de dados por qualquer outra transação até que a transação de atualização do item de dado que esta sendo realizado termine [Ozsu, 2001].

O protocolo *Commit Two-Phase* funciona da seguinte maneira: existe o coordenador, que é o local onde o processo de transação se origina e que controla a execução da transação, e, os participantes que são os outros locais que participam da execução da transação.

- Na fase 1: o coordenador seleciona os participantes que estão prontos para escrever os resultados no banco de dados.
- Na fase 2: todos escrevem os resultados no banco de dados.

Na regra global de *Commit*, o coordenador aborta uma transação, somente se pelo menos um dos participantes votar para abortar a transação, e, o coordenador

executa um *Commit* de uma transação, somente se todos os participantes votarem para executar o *Commit* da transação.

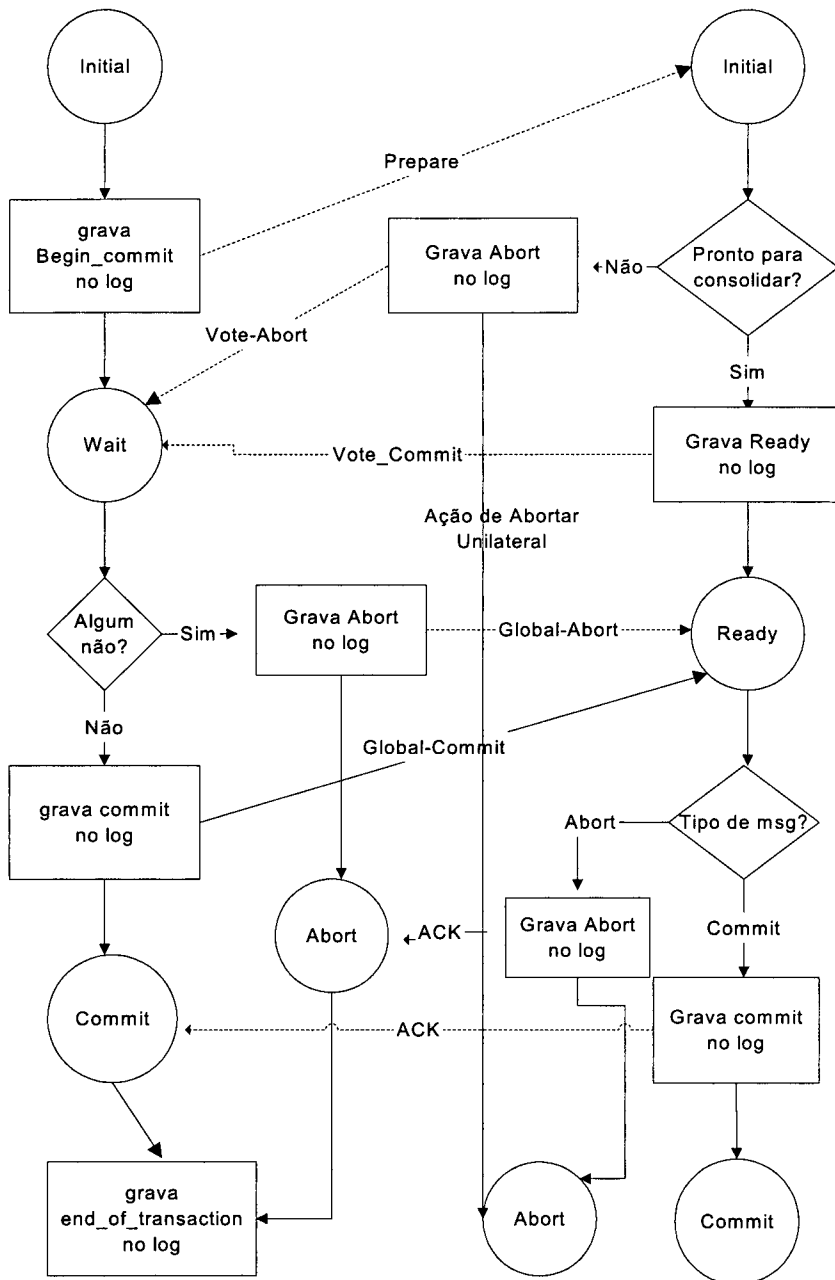
Há várias formas de comunicação diferentes, que podem ser implementadas em um protocolo 2PC. A figura 4.4 a seguir, relata o protocolo 2PC Centralizado, onde a comunicação ocorre somente entre o coordenador e os participantes, sendo que os participantes não se comunicam entre si [Ozsu, 2001].

Outra alternativa é a 2PC linear em que os participantes podem comunicar-se entre si. Outra estrutura de comunicação para implementação do protocolo 2PC, envolve a comunicação entre todos os participantes durante a primeira fase do protocolo, de forma que todos os *sites* possam tomar uma decisão independente, com respeito a uma transação.

O protocolo 2PC é implementado por uma comunicação entre o coordenador e os *sites* participantes, na qual a primeira fase é concluída, e por uma comunicação inversa dos participantes com o coordenador, concluído assim, a segunda fase do protocolo. Desta forma, teremos [Ozsu, 2001]:

- O coordenador enviando uma mensagem de “prepare” ao próximo *site* participante, (por exemplo, o *site* número 2). Se o *site* participante (2), não estiver pronto para consolidar a transação, ele envia uma mensagem de “vote-abort” (VA) ao próximo *site* participante (no caso o número 3) e nesse instante a transação é abortada;
- Porém, se o *site* participante (2) estiver pronto para efetuar a transação, ele envia uma mensagem de “vote-commit” (VC) ao *site* participante (3) e fica no estado de *Ready*. Esse processo é repetido até o VC chegar ao coordenador da transação, chegando ao fim, a primeira fase do protocolo;

Se o coordenador optar por consolidar a transação, ele envia uma mensagem de “global-commit” (GC) ao próximo *site* participante (2), caso contrário, o coordenador envia uma mensagem de “global-abort” (GA). Conseqüentemente os *sites* participantes entram no estado de *Commit* ou de *Abort*, e essa mensagem é repassada de um *site* para outro até a mesma voltar para o coordenador [Ozsu, 2001].



**Figura 4.4. Ações do Protocolo 2PC Centralizado [Ozsu, 2001]**

#### 4.3.1.1 Fase I

Na primeira etapa do protocolo *2PC*, o coordenador é que decide quando uma transação  $T$  deve ser efetivada ou abortada, ainda que algumas etapas não tenham sido executadas:

- 1) o coordenador insere um registro de log  $\langle \text{Prepare } T \rangle$  no log local;

- 2) o coordenador envia para todos os participantes da transação  $T$  a mensagem de  $\langle \text{Prepare } T \rangle$ ;
- 3) cada participante que recebe a mensagem  $\langle \text{Prepare } T \rangle$  decide se deve efetivar ou abortar seu componente da transação  $T$  e enviar a resposta;
- 4) se um local resolver consolidar seu componente, deve entrar em um estado chamado de pré-consolidado. Quando o local se encontra nesse estado, não pode abortar o seu componente sem uma autorização do coordenador. Estão abaixo descritas as etapas a serem seguidas quando o componente se torna pré-consolidado:
  - a) executar todas as etapas necessárias, para assegurar que o componente local de  $T$  não aborte, mesmo que ocorram falhas no sistema, seguidas de uma recuperação;
  - b) inserir um registro de  $\langle \text{Ready } T \rangle$  no log e esvaziar o log para o disco;
  - c) enviar ao coordenador a mensagem de  $\langle \text{Ready } T \rangle$ .
- 5) porém, se o participante resolver abortar o seu componente de  $T$ , ele deverá anotar um registro de  $\langle \text{Don't Commit } T \rangle$  e enviar a mensagem de  $\text{Don't Commit } T$  para o coordenador [Ozsu, 2001].

#### 4.3.1.2 Fase II

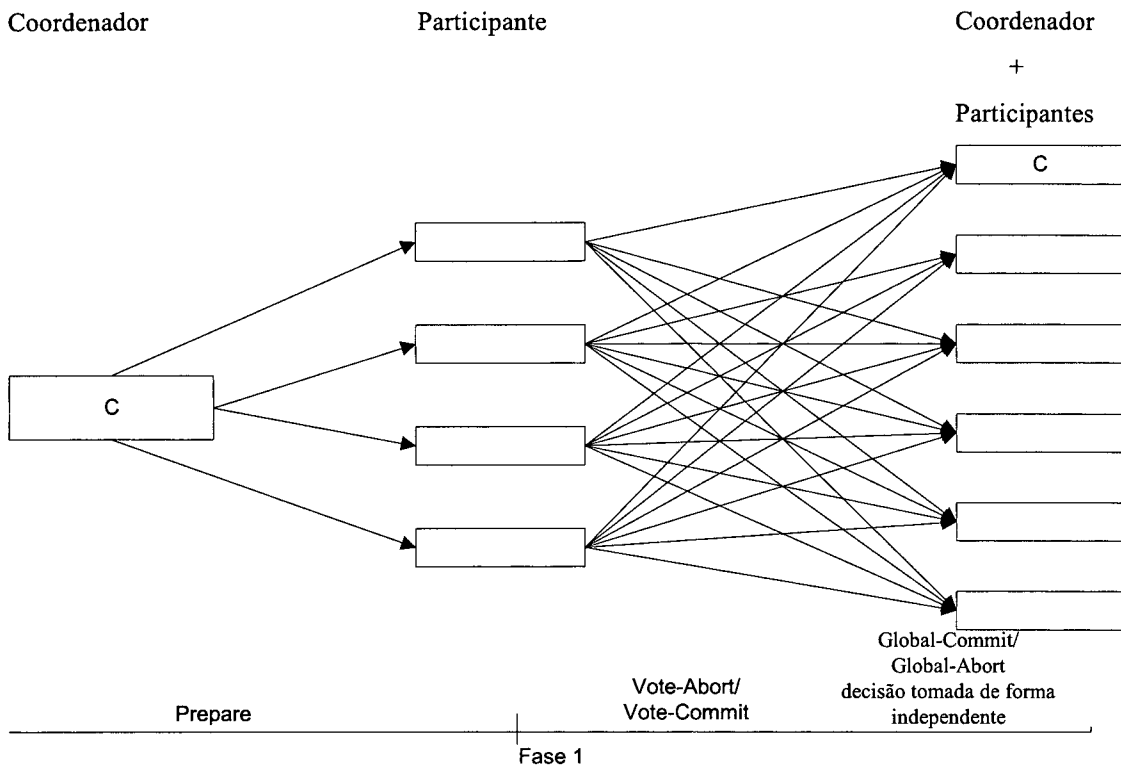
A segunda fase tem início, quando as respostas de cada local (participante) são recebidas pelo coordenador. Se algum local não enviar resposta, pelo fato deste estar fora de serviço ou desconectado da rede, após um tempo limite, o coordenador assumirá que o local enviou a resposta de  $\text{Don't Commit } T$ .

- 1) Se o coordenador receber a mensagem de  $\text{Ready } T$  de todos os locais participantes da transação  $T$ , então a transação será efetivada e o coordenador:
  - a) anota um registro de  $\langle \text{Commit } T \rangle$  em seu log local;

- b) envia uma mensagem de *Commit T* para todos os locais envolvidos na transação *T*.
- 2) Se o coordenador receber uma mensagem de *Don't Commit T* de um ou mais locais:
- a) anotar um registro de *<Abort T>* em seu log local;
  - b) enviará uma mensagem de *Abort T* para todos os locais envolvidos na transação *T*.
- 3) se um local receber a mensagem de *Commit T*, ele efetivará o componente de *T* nesse local e anotar um registro de *<Commit T>* no log;
- 4) se um local receber a mensagem de *Abort T*, ele abortará o componente de *T* e anotar um registro de *<Abort T>* no log [Ozsu, 2001].

#### 4.3.1.3 *Commit Two-Phase* Distribuído

A versão chamada de *Commit Two-Phase* Distribuída, elimina a necessidade da segunda fase do protocolo, pois os *sites* envolvidos na transação podem tomar a decisão sozinhos. Desta forma, o coordenador envia a mensagem de *prepare* a todos os participantes, por meio de uma mensagem: *vote-commit* ou *vote-abort*. Os participantes esperam por mensagens de todos os outros *sites* participantes e tomam a decisão de término, de acordo com a regra de consolidação global. Na Figura 4.5 a seguir, é apresentada a estrutura de comunicação da consolidação distribuída [Ozsu, 2001].



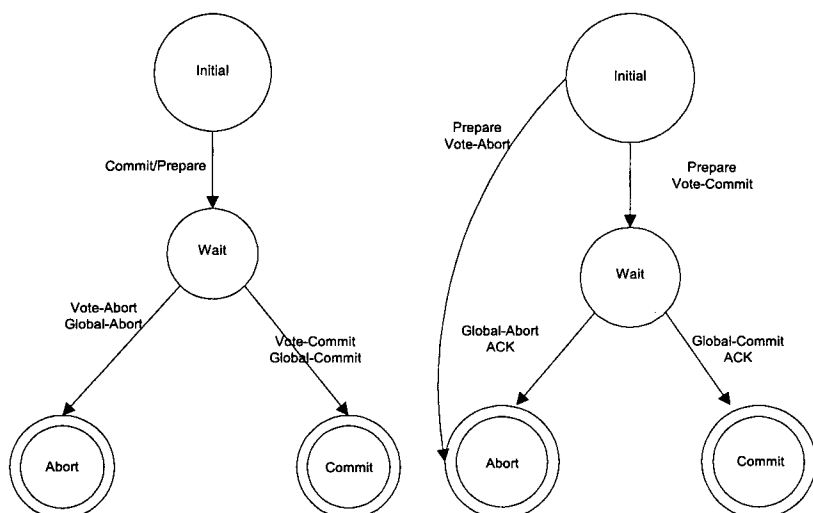
**Figura 4.5. Estrutura de Comunicação de 2PC Distribuída [Ozsu, 2001].**

#### 4.3.1.3.1 Protocolos de término

O protocolo de término é utilizado quando o tempo limite ocorre em um *site* de destino, isto é, quando ele não pode receber uma mensagem esperada de um *site* de origem, dentro de um certo período de tempo. O método utilizado para tratamento de tempo limite, depende de como são as falhas e do sincronismo que ocorre entre as mesmas.

Na figura 4.6 a seguir, são mostradas as transições de estado, que acontecem quando ocorrem intervalos de tempo limite entre os estados. No caso, os estados são denotados por círculos, e as arestas representam as transições de estado, os estados terminais são representados por círculos concêntricos, a transição de estado é dada na parte superior (mensagem recebida), e o resultado da transição é dada na parte inferior (mensagem enviada) [Ozsu, 2001].





**Figura 4.6. Transição de estado no protocolo 2PC [Ozsu,2001].**

O coordenador passa por três estados, quando ocorre um intervalo de tempo limite, a saber:

- Tempo limite no estado Espera (*Wait*): nesse estado, o coordenador está esperando por uma decisão dos *sites* participantes, desta forma, o coordenador não pode consolidar a transação de forma unilateral, porque a regra de consolidação global não foi satisfeita. A única ação que o coordenador pode tomar é a de abortar a transação, assim é gravado um registro de “*global-abort*” no arquivo de log e enviada uma mensagem a todos os *sites* participantes;
- Tempo limite no estado de Efetivação (*Commit*) ou Cancelamento(*Abort*): o coordenador não pode afirmar se os procedimentos de consolidação ou de cancelamento foram executados pelos *sites* participantes ou não. O coordenador fica enviando repetidamente as mensagens de “*global-commit*” e “*global-abort*” aos *sites* participantes da transação que ainda não enviaram resposta, e, espera pela confirmação dos mesmos [Ozsu, 2001].

Os *sites* participantes das transações podem passar por dois estados quando ocorre um intervalo de tempo limite, a saber:

- Tempo limite no estado Inicial (*Initial*): nessa fase o participante esta esperando pela mensagem “*prepare*”, provavelmente o coordenador tenha falhado no estado Inicial. O participante pode tomar a decisão de abortar a transação, e se receber mais tarde a mensagem “*prepare*”, ele irá consultar o

seu log e encontrará o registro de *abort* e responderá com um “*vote-abort*”, ou poderia simplesmente ignorar a mensagem de “*prepare*”;

- Tempo limite no estado de Pronto (*Ready*): nessa fase, o participante votou por consolidar a transação, mas ainda não conhece a decisão do coordenador. Desta forma, como o participante está no estado de espera e já votou por consolidar a transação, não pode mais tomar a decisão de abortar unilateralmente e terá que aguardar até que o coordenador envie a mensagem da decisão global ou então que algum outro *site* participante o avise [Ozsu, 2001].

Quando está se trabalhando com um protocolo de término distribuído, o participante que entrar em um tempo limite, pode pedir aos outros participantes que o ajudem a tomar uma decisão. Teremos então, o  $P_i$  como o *site* participante que entra em tempo limite e o  $P_j$  como os outros *sites* que responderão da seguinte forma:

- O  $P_j$  encontra-se no estado Inicial, isto significa que o  $P_j$  ainda não votou e talvez não tenha recebido a mensagem de “*prepare*”, podendo assim tomar a decisão de abortar de forma unilateral e irá responder à  $P_i$  com uma mensagem de “*vote-abort*”;
- O  $P_j$  encontra-se no estado de Pronto. Nesse caso o  $P_j$  votou por consolidar a transação, mas ainda não recebeu a mensagem de decisão global e portanto não pode ajudar a  $P_i$ ;
- O  $P_j$  encontra-se no estado de Consolidação ou de Cancelamento. Nesses casos o  $P_j$  tomou uma decisão unilateral de abortar a transação, ou recebeu uma mensagem do coordenador a respeito do término da transação, portanto pode enviar à  $P_i$  uma mensagem de “*vote-commit*” ou “*vote-abort*”.

Levando-se em conta, após receber essas respostas, o *site* participante  $P_i$  pode interpretar as respostas das seguintes maneiras:

- O *site*  $P_i$  recebeu mensagens “*vote-abort*” de todos os  $P_j$ , com isso, ele deduz que nenhum dos outros participantes votou ainda, mas que todos eles optaram por abortar a transação de forma unilateral, desta forma,  $P_i$  pode prosseguir até abortar a transação;

- O *site*  $P_i$  recebe mensagens “*vote-abort*” somente de alguns *sites*  $P_j$ , os outros participantes indicam que eles estão no estado de Pronto. Portanto, o  $P_i$  pode abortar a transação;
- O *site*  $P_i$  recebe de todos os *sites*, a mensagem que estão no estado de Pronto, assim, nenhum dos participantes sabe dizer sobre o destino da transação para encerrá-la corretamente;
- O *site*  $P_i$  recebe mensagens “*global-abort*” ou “*global-commit*” de todos os *sites*  $P_j$ , e, conclui que os outros *sites* receberam a decisão do coordenador. Portanto,  $P_i$  pode seguir em frente e tomar a decisão com base nas mensagens recebidas;
- O *site*  $P_i$  recebe mensagens de “*global-abort*” ou de “*global-commit*” somente de alguns  $P_j$ , enquanto que os outros indicam que estão no estado de Pronto, isto significa que alguns *sites* receberam a mensagem do coordenador e os outros estão esperando pela mensagem. Da mesma forma,  $P_i$  pode seguir como no caso anterior[Ozsu, 2001].

#### 4.3.1.4 Recuperação utilizando o protocolo *Commit Two-Phase - 2PC*

Em qualquer momento durante o processo de efetivação usando o protocolo *2PC*, um local pode falhar. Quando esse local se recuperar, a decisão a ser tomada por ele deve ser coerente com a decisão global tomada sobre a transação em questão. Há vários casos a considerar, dependendo do último registro armazenado no log para a transação  $T$ .

- 1) se o último registro de log foi  $\langle \text{Commit } T \rangle$  é porque  $T$  foi efetivada pelo coordenador;
- 2) se o último registro de log foi  $\langle \text{Abort } T \rangle$  é porque a decisão tomada pelo coordenador foi de abortar a transação;
- 3) se o último registro de log foi  $\langle \text{Don't Commit } T \rangle$ , então o local sabe que a decisão global deve ter sido a de abortar  $T$ ;
- 4) se o último registro de log foi  $\langle \text{Ready } T \rangle$ , o local de recuperação não sabe se a decisão global foi de efetivar ou abortar  $T$ . Desta forma, o local deve entrar em contato com pelo menos outro local, para saber qual foi a decisão

- global tomada com relação a  $T$ . Se nenhum local responder ao local solicitante, este deve esperar pela decisão do coordenador de abortar/efetivar;
- 5) pode ocorrer que o log não tenha nenhum registro sobre a transação  $T$ . Neste caso, o local deverá tomar uma decisão de abortar o seu componente de  $T$ .
  - 6) se o log contém um registro de  $\langle \text{Commit } T \rangle$  o *site* deverá executar o sistema de recuperação, utilizando o log de refazer;
  - 7) se o log contém um registro de  $\langle \text{Abort } T \rangle$  o *site* deverá executar o sistema de recuperação, utilizando o log de desfazer.

Os casos especificados acima, são referentes ao fato de que o *site* que falhou não é o coordenador. Quando o coordenador falha durante o protocolo  $2PC$ , surgem novos problemas. O primeiro deles é que os locais participantes, devem esperar que o coordenador se recupere ou então eleger um novo coordenador, o segundo e mais complexo, é a eleição deste coordenador. [Molina,2001] descreve que um método simples e seguro para eleger um novo líder é considerar que todos os locais participantes enviam mensagens informando o seu numero de identificação (IP), e o novo coordenador será o local cuja identificação seja a numeração mais baixa das que foram enviadas. Se houver inconsistência nas mensagens enviadas, ou algum local não enviar a sua mensagem, deverá esperar um tempo e a eleição será novamente reiniciada.

Após o novo líder ter sido escolhido, ele consulta os locais em busca de informações sobre cada transação distribuída  $T$ , e, cada local informa então, qual foi o último registro em seu log referente a  $T$ . Desta forma os casos possíveis são:

- se um dos locais tem  $\langle \text{Commit } T \rangle$  em seu log, significa então que o coordenador original deve ter enviado mensagens de  $\text{Commit } T$  para todos, desta forma é necessário efetivar  $T$ ;
- de modo semelhante, se um local tem  $\langle \text{Abort } T \rangle$  em seu log, o coordenador original decidiu abortar  $T$ , é seguro que o novo coordenador execute essa ação;
- supondo que nenhum local tenha  $\langle \text{Commit } T \rangle$  ou  $\langle \text{Abort } T \rangle$  em seu registro de log e também nenhum local tenha um registro de  $\langle \text{Ready } T \rangle$ , como as mensagens são arquivadas antes de serem enviadas, o antigo coordenador nunca recebeu  $\text{Ready } T$  desse local e então pode não ter

decidido consolidar  $T$ , desta forma é seguro que o novo coordenador aborte  $T$ ;

- o caso mais difícil é quando não existe nenhum registro de  $\langle Commit T \rangle$  ou  $\langle Abort T \rangle$ , e todos os locais possuem o registro de  $\langle Ready T \rangle$ , desta forma, dificulta descobrir qual foi a última ação do coordenador anterior. O novo coordenador não tem capacidade de decidir se deve abortar ou efetivar a transação de  $T$  e deve esperar até que o coordenador antigo se recupere, ou então o Administrador do Banco de Dados poderá intervir e forçar o término dos componentes da transação  $T$  que estão a espera [Molina, 2001].

A recuperação do sistema pode ser exigida ainda com outro tipo de falha, onde o problema reside no particionamento da rede:

- o coordenador e todos os *sites* participantes da transação, permanecem em uma partição, desta forma o protocolo de efetivação não é afetado pela falha;
- o coordenador e os participantes da transação se separam entre as partições da rede, desta forma os *sites* de uma partição vê que os *sites* da outra estão fora do ar. Os *sites* que não estão na mesma partição do coordenador, executarão o protocolo de recuperação do coordenador, enquanto que os *sites* que se encontram na mesma partição do coordenador, seguem a efetivação normal, assumindo que os outros *sites* estão fora do ar.

A maior desvantagem do protocolo 2PC é que uma falha do coordenador pode gerar uma obstrução, isto é, os demais *sites* ficam na espera para poderem efetivar ou abortar uma transação até que o coordenador volte a funcionar.

Se a recuperação é realizada conforme foi descrito acima, o processamento normal das transações em um *site* não podem ser inicializadas até que as transações em dúvida sejam efetivadas ou desfeitas. O processo de descobrir o status de uma transação, pode ser demorado por ter que consultar vários *sites* até descobrir qual é o status da transação. Se o coordenador estiver fora do ar e nenhum outro *site* possuir o status da transação, a recuperação ficará obstruída, deste modo, o *site* que esta reiniciando pode ficar ocioso por um longo período.

Para resolver esse tipo de impasse, os algoritmos de recuperação possuem um suporte para notificação no log de informações de bloqueio. Em vez de escrever somente um registro de  $\langle Ready T \rangle$  no log, é escrito um registro de  $\langle Ready T, L \rangle$ , onde o  $L$  representa uma lista de todos os bloqueios realizados pela transação  $T$  quando o registro de log é escrito. Durante o processo de recuperação, para toda transação  $T$  com dúvida, todos os bloqueios notificados no registro  $\langle Ready T, L \rangle$  do log são reassumidos.

Após readquirir os bloqueios de todas as transações em dúvida, o processamento da transação poderá começar no *site*, mesmo que o status de uma transação que se encontra em dúvida não tenha sido determinado (efetivar ou abortar). A efetivação ou o aborto de uma transação em dúvida pode ser realizada concorrentemente com a execução de novas transações, desta forma, a recuperação é mais rápida e nunca fica obstruída [Ozsu, 2001].

#### 4.3.2 Protocolo de *Commit There-Phase* (3PC)

Esse protocolo foi criado para evitar a possibilidade de paralisação na ocorrência de falhas utilizando o protocolo *2PC*. Foi acrescentando aos estados de *Wait* (e *Ready*) e *Commit*, onde o processo está pronto para *Commit* (se esta for a decisão final) mais um processo. São descritos os diagramas de transição estatais para o coordenador e o participante neste protocolo. Isto é chamado de Protocolo de *Commit There-Phase* porque há três transições de estado, do estado *Initial* para um estado de *Commit*.

O protocolo *3PC*, consegue impedir a paralisação, porque foi acrescentada uma fase extra na qual é criada uma pré-decisão sobre o destino da transação. Essa pré-decisão é passada aos *sites* participantes, então, mesmo que ocorra uma falha do coordenador, a decisão pode ser tomada com base na pré-decisão [Ozsu, 2001]. Na figura 4.7 é apresentada a estrutura do protocolo 3PC, assim teremos:

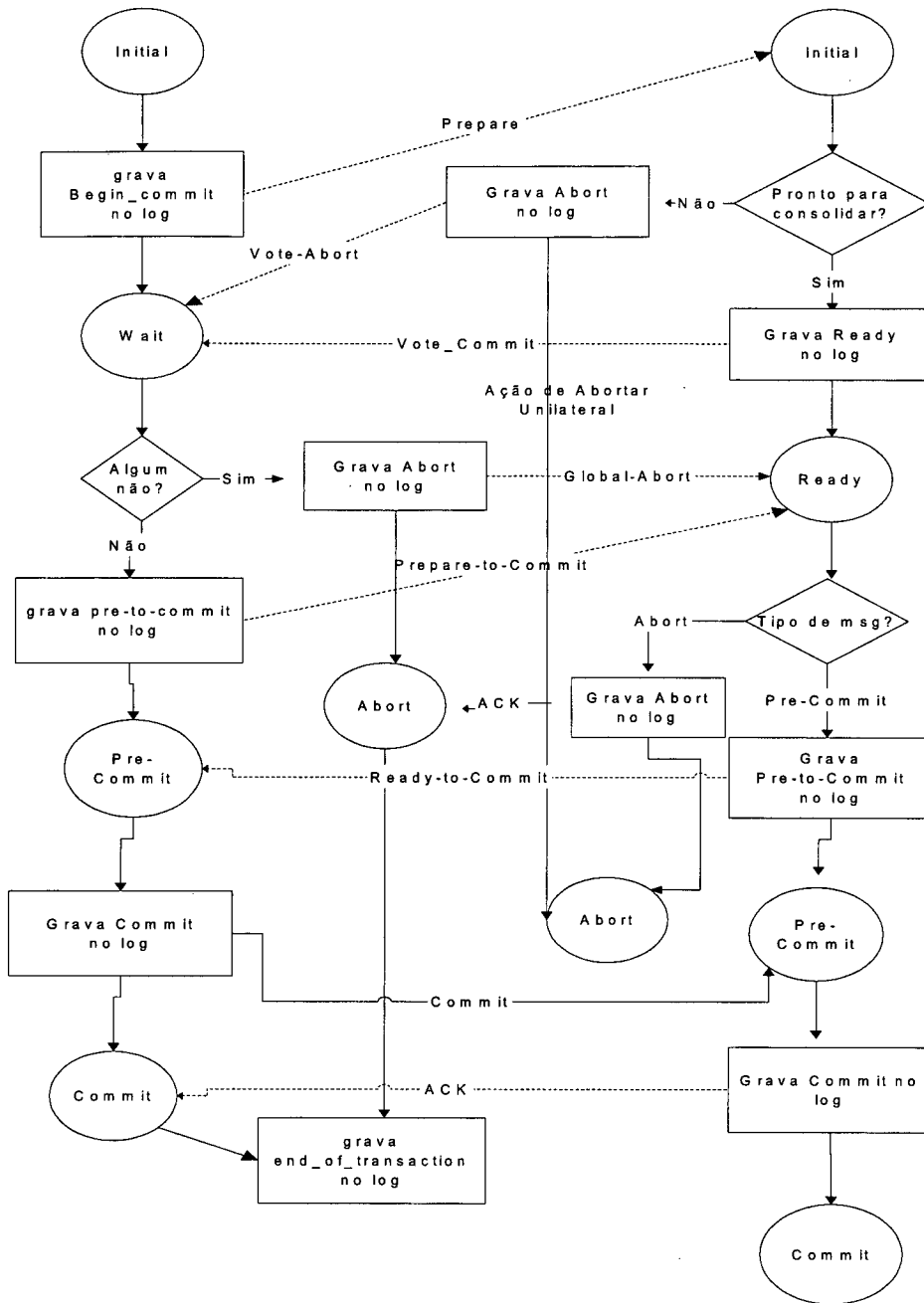


Figura 4.7. Ações do protocolo 3PC – [Ozsu, 2001].

#### 4.3.2.1 Fase I

Idêntica ao protocolo 2PC.

#### 4.3.2.2 Fase II

- Quando o coordenador recebe uma mensagem de *Abort T* de um dos participantes, ou se o mesmo não recebe nenhuma mensagem dos participantes por um certo período de tempo, a decisão a ser tomada pelo coordenador é a de *Abort T*.
- Se o coordenador receber uma mensagem de *Ready T* de todos os *sites* que estão participando da transação, toma a decisão preliminar de efetivação de *T* (*pre-commit T*). A pré-efetivação difere da efetivação completa, pelo fato de que ainda pode haver um cancelamento ou um aborto da transação *T*;
- A decisão da pré-efetivação, permite que o coordenador transmita aos *sites* participantes, a informação de que todos os *sites* participantes estão prontos. O coordenador adiciona um registro de *<precommit T>* ao log, forçando a escrita do log em disco e envia aos *sites* participantes a mensagem de *<precommit T>*;
- Quando o *site* recebe a mensagem (*Abort* ou *Precommit T*), grava esse registro no log e descarrega o log em disco, enviando uma mensagem de *Acknowledge T* (reconhecimento) para o coordenador [Ozsu, 2001];

#### 4.3.2.3 Fase III

Essa fase somente é executada se na fase II ocorrer uma decisão de pré-efetivação.

- Após a mensagem de *Pre-Commit T* ser enviada aos *sites* participantes, o coordenador espera que alguns *sites* enviem o retorno de *acknowledge T* para poder tomar a decisão de efetivar a transação *T*.
- É adicionado o registro de *<Commit T>* ao log e descarregado em disco, enviando uma mensagem de *Commit T* a todos os *sites* participantes.
- Quando o *site* participante recebe essa mensagem, registra a informação no log.
- O *site* que está executando a transação pode abortá-la a qualquer momento, mesmo antes de enviar a mensagem de *Ready T* ao coordenador. A



mensagem de *Ready T* significa que o *site* executará a ordem do coordenador.

- A mensagem enviada pelo coordenador de *Pre-Commit T*, significa que o coordenador deverá efetivar a transação *T*. Desta forma, quando é utilizado o protocolo *3PC* é assegurado que sempre a transação será efetivada [Molina, 2001].

#### 4.3.2.4 Protocolo de Término

Da mesma forma que no protocolo 2PC, o protocolo 3PC também trabalha com os intervalos de tempo limite. Quando o tempo limite é o do coordenador, existem quatro (4) estados descritos a seguir:

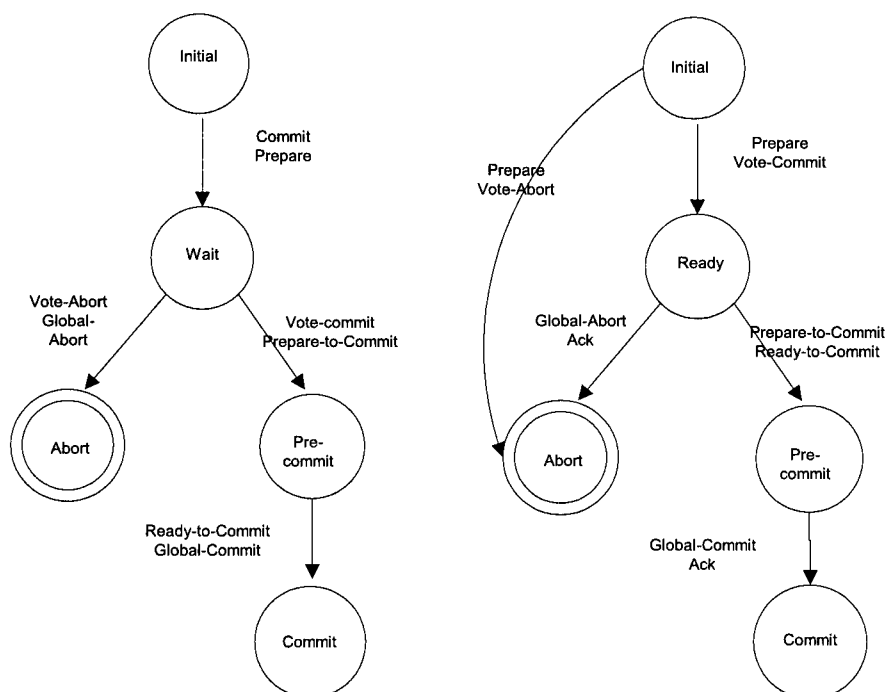
- Tempo limite no estado de Espera (*Wait*): ocorre da mesma forma que no protocolo 2PC. Desta forma, ele decide unilateralmente abortar a transação, gravando no log um registro de *abort* e envia uma mensagem de “*global-abort*” a todos os participantes que votaram por consolidar a transação;
- Tempo limite no estado de *Precommit*: o coordenador não tem a informação de que os *sites* participantes responderam ou então estão no estado *Precommit*. Porém ele sabe que os *sites* participantes estão no estado de Pronto, concluindo então, que eles devem ter votado por consolidar a transação. O coordenador passa todos os *sites* participantes para o estado de *Precommit*, enviando uma mensagem de “*prepare-to-commit*”, e após isso, consolida a transação de forma global, gravando no log o registro de *Commit* e enviando uma mensagem de “*global-commit*”;
- Tempo limite no estado de Consolidação (*Commit*) ou Cancelamento (*Abort*): nessa fase o coordenador não sabe qual a ação tomada pelos *sites* participantes, mas sabe que eles estão no estado de *Precommit*, desta forma, ele não precisa executar qualquer ação especial.

O tempo de intervalo limite entre os participantes podem ocorrer em três (fases), a saber:

- Tempo limite no estado Inicial (*Initial*): é tratado de maneira idêntica ao protocolo 2PC;

- Tempo limite no estado de Pronto (*Ready*): nesse estado, o *site* participante votou por consolidar a transação, mas não conhece a decisão global do coordenador. Levando-se em conta, que a comunicação com o coordenador não existe mais, é eleito um novo coordenador que dará continuidade no processo a partir deste ponto;
- Tempo limite no estado de *Precommit*: o *site* participante recebeu a mensagem de “*prepare-to-commit*” e está aguardando a mensagem final de “*global-commit*” do coordenador.

A figura 4.8 a seguir, mostra as transições de estado que ocorrem em um protocolo 3PC.



**Figura 4.8. Transições de estado no protocolo 3PC [Ozsu,2001].**

#### 4.3.2.5 Manuseio de Falhas utilizando o protocolo 3PC

Na ocorrência de um *site* participante da transação falhar:

- Se um *site* falha, o coordenador toma as providências semelhantes ao protocolo *2PC*. Se ocorrer de um *site* falhar antes de responder ao coordenador com *Ready T*, a resposta a ser considerada é a de *Abort T*, caso contrário, o resto do protocolo continua normalmente.

- Após o *site* participante se recuperar da falha, ele examina o log de algum *site* vizinho, para descobrir qual foi a decisão tomada com relação à transação quando ocorreu a falha. No log poderá ser encontrado:
  - a) um registro de  $\langle \text{Commit } T \rangle$ , o *site* executará o protocolo de refazer  $T$ ;
  - b) um registro de  $\langle \text{Abort } T \rangle$ , o *site* executará o protocolo de desfazer  $T$ ;
  - c) um registro de  $\langle \text{Ready } T \rangle$  e nenhum registro de  $\langle \text{Abort } T \rangle$  ou de  $\langle \text{pre-commit } T \rangle$ . Nesse caso, o *site* deverá consultar o coordenador para saber qual o destino da transação. Caso o coordenador responda que a transação foi efetivada, o *site* executará o protocolo de recuperação *Redo* ( $T$ ). Se o coordenador responder que a transação foi abortada, o *site* executará o protocolo de recuperação *Undo* ( $T$ ) e se o coordenador responder com uma mensagem de *Precommit*  $T$ , o *site* armazena essa informação e enviará o retorno de *acknowledge*  $T$  para o coordenador, e caso o coordenador não responda, será executado o protocolo de falha do coordenador;
  - d) um registro de  $\langle \text{precommit } T \rangle$  e nenhum registro de  $\langle \text{abort } T \rangle$  ou  $\langle \text{commit } T \rangle$ , o *site* consultará o coordenador e dependendo da resposta se a transação foi abortada ou efetivada, o *site* executa o protocolo de recuperação *Redo* ( $T$ ) ou *Undo* ( $T$ ), caso o coordenador responde que está no estado de pré-efetivação, o *site* continua a execução a partir deste ponto e, caso o coordenador não responda, será executado o protocolo de falha do coordenador.

O protocolo de falha do coordenador sempre é ativado por um *site* que está participando da transação, à medida que o *site* não recebe uma resposta do coordenador dentro de um determinado intervalo de tempo [Ozsu, 2001].

Quando ocorre uma falha do coordenador:

- a) os *sites* participantes elegem um novo coordenador através de um protocolo de eleição;
- b) o novo coordenador envia uma mensagem a todos os *sites* que estão participando, pedindo que estes informem o status local da transação em vigor;
- c) cada *site* envia ao coordenador, inclusive ele próprio, o status de  $T$ .

- efetivada, o log possui um registro de  $\langle Commit T \rangle$ ;
  - abortada, o log possui um registro de  $\langle Abort T \rangle$ ;
  - pronta, o log possui um registro de  $\langle Ready T \rangle$ , mas não possui nenhum registro de  $\langle Precommit T \rangle$  ou de  $\langle Abort T \rangle$ ;
  - pré-efetivada, o log possui um registro de  $\langle precommit T \rangle$ , mas não contém nenhum registro de  $\langle Commit T \rangle$  ou de  $\langle Abort T \rangle$ ;
  - não-pronta, o log não possui nenhum registro de  $\langle Ready T \rangle$  e nenhum registro de  $\langle Abort T \rangle$ .
- d) após o coordenador receber a resposta, irá decidir se deve efetivar ou abortar a transação T, ou ainda, se reinicia o protocolo de 3PC, desta forma:
- se ao menos um dos *sites* tem o status local como *Committed*, o coordenador efetiva T;
  - se ao menos um dos *sites* tem o status local de *aborted*, o coordenador aborta T;
  - se nenhum dos *sites* possui o status de *aborted* e nem de *Committed*, mas pelo menos um dos *sites* possui o status de *precommitted*, o coordenador reinicializa o protocolo 3PC, enviando uma mensagem de *precommit* a todos os *sites*;
  - fora essas condições, o coordenador sempre irá abortar a transação [Ozsu, 2001].

Com a utilização do protocolo de falha do coordenador, é possível que o novo coordenador possa tomar uma decisão baseado no status que o coordenador antigo deixou quando falhou.

Este capítulo apresentou alguns aspectos relacionados com a confiabilidade dos dados em um sistema de gerenciamento distribuído. Quais são os tipos de falhas que podem ocorrer em um sistema distribuído, os protocolos de confiabilidade 2PC e 3PC, e, como é o comportamento destes dois protocolos quando ocorre uma falha e eles estão sendo executados por alguma transação, sendo que o objetivo principal dos protocolos em questão é garantir a atomicidade e durabilidade das transações distribuídas.

## Capítulo 5

### Banco de Dados - *Oracle*

#### 5. *Oracle*

A *Oracle* é uma companhia que foi criada em 1977, com matriz em Redwood Shores, na Califórnia, e é considerada a única companhia capaz de implementar soluções completas e globais de e-business que se estendem do CRM (gerenciamento do relacionamento com o cliente) no front office às aplicações operacionais dos sistemas internos de apoio (back office) e à infra-estrutura da plataforma.

O software *Oracle* foi criado para rodar em PCs, estações de trabalho, minicomputadores, mainframes e computadores de processamento paralelo maciço, bem como em PDAs e dispositivos portáteis [*Oracle*, 2].

O servidor *Oracle* é um sistema de gerenciamento de banco de dados relacional que fornece uma abordagem aberta, abrangente e integrada, utilizada no gerenciamento de informações, é constituído por um banco de dados e uma instância.

#### 5.1 Estrutura do banco de dados *Oracle*

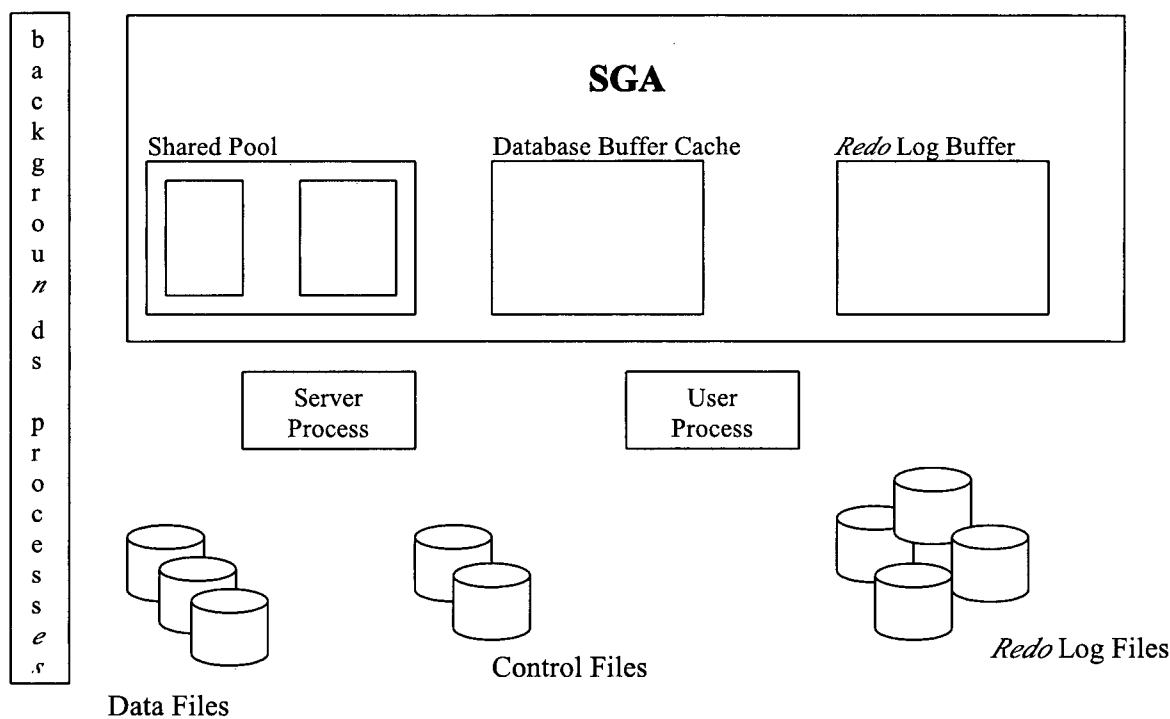
A arquitetura de um banco de dados *Oracle* possui uma estrutura lógica e física:

- Estrutura Física: é formado pelos arquivos do Sistema Operacional e mais quatro tipos de arquivos, a saber:
  - Um ou mais arquivos de dados;
  - Dois ou mais arquivos de registro *Redo*;
  - Um ou mais arquivos de controle;
  - Arquivo de parâmetro.
- Estrutura Lógica:
  - É formada por uma ou mais tablespaces, que são unidades lógicas que possuem os relacionamentos lógicos, isto é, possuem todos os objetos de um aplicativo, para simplificar as operações administrativas;

- é formada também por esquemas, que é uma coleção de objetos em forma de tabelas, visões, índices, links de banco de dados e outros.

A tabela 5.1 a seguir mostra um esquema geral da arquitetura de um banco de dados *Oracle*:

*backgrounds processes*



**Figura 5.1. Esquema geral de um banco de dados *Oracle*. [Oracle, 1]**

Sempre que um banco de dados é inicializado, uma Área Global do Sistema (SGA – System Global Área) é alocada junto com os processos de segundo plano (*backgrounds processes*) do *Oracle*. A área global do sistema, é uma área na memória, utilizada para armazenar informações sobre o banco de dados, para que os usuários possam utilizar durante o processamento das aplicações. Essa combinação entre os processos do segundo plano e SGA é chamada de Instância do *Oracle* [Oracle, 1].

A Instância do *Oracle* é formada também por dois tipos de processos: processos de usuário e os processos do servidor do *Oracle*.

- Processos de Usuário: executa os aplicativos próprios do usuário, dentre os quais se destacam o *Oracle Forms* ou *Oracle Tool*, *Enterprise Manager* e outros;

- Processos do *Oracle*. são os processos que executam o trabalho para os processos do usuário em de segundo plano, executam o trabalho do servidor *Oracle*.

A SGA é um grupo de estruturas de memória compartilhadas que contém os dados e controlam as informações para um sistema de banco de dados do *Oracle*. Quando uma instância, é iniciada os buffers de memória da SGA são alocados e os processos de segundo plano são iniciados. Primeiro a instância é carregada para a memória e somente depois o banco de dados é montado por ela. Os usuários que estão conectados no momento a um servidor *Oracle*, compartilham dos dados na Área Global do Sistema.

A SGA está dividida em três partes principais:

- Pool Compartilhado (*Shared Pool*);
- Cache do Buffer de Banco de Dados (*Database Buffer Cache*);
- Buffer do Registro *Redo* (*Redo Log Buffer*).

### 5.1.1 Pool Compartilhado

É uma área de memória da SGA que contém estruturas compartilhadas, como:

- Áreas compartilhadas de comandos SQL (*Shared SQL Áreas*);
- Cache do Dicionário de Dados (*Data Dictionary Cache*)

Cada Shared SQL Área contém informações para executar um único comando SQL, sendo que uma única área SQL compartilhada é usada por vários aplicativos que emitem a mesma declaração, lembrando, que somente comandos completamente idênticos serão encaminhados para a mesma posição de memória.

O cache do dicionário de dados, contém um subconjunto do dicionário de dados, ou seja, as informações sobre as tabelas (nome) e visões utilizadas pelos comandos SQL, os nomes e os tipos de dados de colunas nas tabelas do Banco de Dados e os privilégios de todos os usuários do *Oracle* [*Oracle*, 1].

### 5.1.2 Cache do Buffer de Banco de Dados

Os buffers de banco de dados da SGA, armazenam os blocos de dados do banco de dados usados mais recentemente, enquanto que o conjunto dos buffers de banco de dados de uma instância é o cache do buffer de banco de dados.

O acesso pode ser feito de duas formas:

- Cache *miss*;
- Cache *hit*.

O cache *Miss*, ocorre quando um processo de usuário, acessa pela primeira vez uma parte do dado. O processo fará a cópia do dado do disco para o cache antes de acessá-lo. O cache *Hit*, ocorre quando o processo de usuário acessa um dado que já está no cache, assim, o dado será acessado diretamente na memória.

Sempre que um usuário requisita dados, o Processo do Servidor verifica se o dado já está no Cache do Buffer de Banco de Dados, se o dado não for localizado, o processo de servidor lê os blocos apropriados nos arquivos de dados para o cache do buffer de banco de dados, onde será armazenado em buffers. Se os usuários modificarem os dados requisitados, as mudanças ocorrerão nos blocos de dados do cache. Esta nova versão dos dados, é conhecida como imagem posterior (*after image*), por ser a imagem do dado que foi alterado.

O *Oracle* salva a versão original dos dados num bloco de reconstrução (*Rollback*), essa versão é chamada então de imagem anterior (*before image*), por ser a imagem dos dados antes das alterações executadas pelo usuário. É utilizada sempre que ocorrer uma falha, ou então sempre que o usuário quiser desfazer a alteração. Essas imagens são armazenadas no Buffer do Registro *Redo* [ Sarin, 2000].

### 5.1.3 Buffer do Registro *Redo*

O Buffer do Registro *Redo* é um buffer circular, que contém informações sobre mudanças ocorridas no banco de dados. As entradas de *Redo* armazenadas nos buffers de registro *Redo*, são gravadas em um arquivo de registro *Redo* on-line, o qual é usado quando a recuperação do banco de dados for necessária.

O Buffer do Registro *Redo* pode ser ignorado pelo *Oracle Data Loader*, ou então usando a palavra reservada *unrecoverable* nos comandos *Create Table* e *Create Index*.



### 5.1.4 Processos de Servidor (Server Processes)

O *Oracle* cria os processos de servidor, para lidar com as solicitações dos processos de usuários conectados. A responsabilidade do processo de servidor é a de se comunicar com os processos de usuário e interagir-se com o *Oracle*, executando as solicitações feitas pelo usuário. Em alguns sistemas, os processos de usuário e os processos de servidor são separados, mas podem ser combinados para se tornar um único processo.

### 5.1.5 Processos de Segundo Plano (Background Processes)

Cada instância do *Oracle*, possui um conjunto de processos de segundo plano, que consolidam as funções que seriam tratadas por vários outros programas do *Oracle*, executando e monitorando de forma assíncrona a E/S, e, os outros processos para fornecer um melhor desempenho e confiabilidade.

Os processos de segundo plano (*background processes*), são processos utilizados para maximizar o desempenho e acomodar os usuários num sistema *Oracle*. Dependendo da configuração, um sistema *Oracle* pode ter os seguintes processos de segundo plano:

- *System Monitor* (SMON)
- *Process Monitor* (PMON)
- *Database Writer* (DBWN) – a função deste processo, é gravar os blocos modificados do cache de buffer do banco de dados para o datafiles. Outra função deste processo, é gravar apenas quando muitos dados precisam ser lidos na SGA e poucos buffers de dados estão livres, sendo que os dados mais recentes são armazenados primeiro no datafiles.
- *Log Writer* (LGWR) – este processo grava as entradas do registro *Redo* no disco, isto é, primeiro as entradas são armazenadas no buffer do registro na SGA e somente depois o processo LGWR grava seqüencialmente em um arquivo de registro *Redo* online.
- *Checkpoint* (CKPT) – após um certo período de tempo especificado, o processo DBWN grava todos os buffers de banco de dados que sofreram modificações na SGA em datafiles. O processo CKPT é responsável por

atualizar os datafiles e os arquivos de controle do banco de dados, para indicar qual é o ponto de controle mais recente.

- *Archiver* (ARCH) – este processo é responsável pela cópia dos arquivos de registro *Redo* online para o armazenamento de arquivamento. Isso acontece quando os arquivos de registro *Redo* online estão cheios ou quando ocorre uma troca de registro.
- *Recoverer* (RECO) – é usado para determinar as transações distribuídas que estão pendentes devido a ocorrência de algum tipo de falha. Em intervalos determinados o Reco local tenta se conectar aos bancos de dados remotos e completar automaticamente o *Commit* ou o *Rollback* da parte local de quaisquer transações distribuídas pendentes.
- *Lock* (LCKn) - é usado para o bloqueio entre instâncias do *Oracle Parallel Server*.
- *Snapshot Refresh* (SNPN) – quando se trabalha com banco de dados distribuído podem ser criados até 36 processos de fila de tarefa, que podem atualizar automaticamente os snapshots de tabela, isto é, a transferência dos dados da memória para uma tabela arquivada.
- *Dispatcher* (Dnnn) - cada processo destes, é responsável pelo roteamento das solicitações dos processos de usuários conectados para os processos de servidor compartilhado disponível, e por retornar as respostas aos processos de usuários apropriados.
- *Parallel Query* (Pnnn)

Os processos de segundo plano SMON, PMON, DBWR e LGWR, são obrigatórios para uma *instância Oracle* executar. Uma instância *Oracle* é composta pela SGA e pelos processos de segundo plano. Se qualquer um destes quatro falharem, a instância sofrerá um *crash* e será reiniciada (*restart*) [Sarin, 2000].

O SMON, PMON, DBWR e LGWR não podem ser controlados através de variáveis de inicialização no arquivo de parâmetros do *Oracle* (o *init.ora*). Eles retomam dos usuários, os recursos do banco de dados que não estão sendo utilizados por um longo período de tempo.

A seguir serão listadas algumas características de cada um dos quatro *background processes* obrigatórios para uma instância *Oracle* executar [Oracle, 1].

### 5.1.5.1 SMON

- executa a recuperação (*recovery*) automática da instância;
- retoma espaços usados por segmentos temporários com muito tempo sem uso;
- reorganiza áreas contíguas (*merge*) de espaço livre nos arquivos de dados (*datafiles*) [Sarin, 2000].

### 5.1.5.2 PMON

- limpa as conexões terminadas anormalmente;
- desfaz (*rollback*) as transações não gravadas (*uncommitted*) no banco de dados;
- libera bloqueios (*locks*) mantidos por um processo finalizado;
- libera recursos da SGA retidos pelos processos que falharam;
- reinicia *server process* e *dispatcher process* compartilhados que falharam.

### 5.1.5.3 DBWR

- Grava todos os *buffers* modificados nos *Datafiles*;
- Utiliza um algoritmo LRU (*Least Recently Used*);
- Atrasa a gravação para otimizar a E/S.

O DBWR gerencia o *Database Buffer Cache*, para que o *server process*, sempre possa encontrar *buffers* livres. O DBWR gravará os *buffers* ocupados para o disco, nos seguintes casos:

- quando a lista de *buffers* ocupados (*dirty list*) atingir um tamanho limite;
- quando um processo percorrer um número específico de *buffers* na lista de blocos menos utilizados (*LRU list*) e não encontrar um *buffer* livre;
- quando atingir um tempo limite entre a execução de dois *DBWR process* (*time-out*);
- quando ocorrer um *checkpoint* (*background process* que força a execução de uma tarefa) para executar um DBWR [Sarin, 2000].

#### 5.1.5.4 LGWR

Grava entradas no *Redo Log Buffer* para o disco. O LGWR é executado nas seguintes situações:

- quando ocorre um *Commit* (operação para atualizar o banco);
- quando o *Redo Log Buffer* atinge um terço de ocupação;
- quando um DBWR termina de executar após um *checkpoint*,
- quando ocorre um *time-out*.

O LGWR possui as seguintes propriedades:

- existe somente um LGWR por instância;
- uma confirmação de *Commit*, não ocorre enquanto a transação não for gravada em disco (no *Redo Log File*);
- durante transações muito longas, o *Redo Log Buffer* pode atingir mais de um terço de sua ocupação, antes do LGWR gravar no *Redo Log File* [Sarin, 2000].

## 5.2. Tipos de Falhas

Vários tipos de falhas podem ocorrer quando se utiliza um banco de dados, algumas são descritas a seguir:

### 5.2.1 Erro de usuário

- Ocorre quando um usuário, por exemplo, apaga acidentalmente uma tabela do banco de dados, ou quando um processo de usuário é abortado;
- Apenas o processo de usuário é afetado, mesmo assim é necessário desfazer as modificações que este processo tenha gerado no banco de dados;
- Periodicamente, o PMON verifica se existem processos abortados, caso encontre, desfaz a transação e libera os bloqueios adquiridos;
- A base de dados, deve ser recuperada para o momento antes de ocorrência do erro.

### 5.2.2 Falha de declaração e processo

- Ocorre quando existir uma falha lógica na manipulação de uma declaração em *Oracle*, por exemplo, a declaração não é uma construção de SQL válido;
- Quando ocorre esse tipo de falha, os efeitos (se tiver) da declaração estarão automaticamente cancelados pelo *Oracle* e o controle será dirigido novamente para o usuário;
- Uma falha de processo, pode ser considerada como uma desconexão anormal do usuário, ou então, o fim anormal de um processo;
- O PMON soluciona o problema desfazendo a transação que originou o problema;
- O *Oracle* executa automaticamente a recuperação necessária de transações e desfaz mudanças que tenham sido originadas destas transações [Sarin, 2000].

### 5.2.3 Falha de Instância

- Ocorre quando um problema impede a instância de continuar a trabalhar normalmente. Estes problemas podem ser originários de uma falha de hardware ou de software, afetando processos e usuários;
- Os dados que estão localizados nos buffers da SGA não estão escritos no arquivo de dados, e a recuperação só pode ser feita utilizando arquivos de log;
- O processo SMON de outra instância, executa a recuperação da instância afetada assim que a mesma é reiniciada.
- Os seguintes passos são executados quando ocorre uma falha de instância:
  - Recuperar os dados que não tenham sido registrados no arquivo de dados ou, que ainda não tenham sido registrados no arquivo de *Redo Log*, inclusive o conteúdo dos segmentos de *Rollback*. Esse tipo de recuperação é conhecido como *cache recovery*.
  - Tornar a base de dados disponível, em vez de esperar que todas as transações efetuem *Rollback*. O *Oracle* permite que isso seja executado, assim que a recuperação de *cache recovery* é completada.

- Marcar todas as transações que estavam ativas no momento da falha como “morta”, e, marcar o segmento de *Rollback* com transações contendo “partes disponíveis”.
- Forçar o SMON a executar *Rollback* em transações marcadas como “mortas”. Esse tipo de recuperação é conhecida como *transaction recovery*.
- Solucionar quaisquer transações distribuídas que estiverem pendentes no protocolo *Two-Phase Commit* no momento da falha.
- As novas transações encontram filas bloqueadas pelas transações “mortas”, forçando o *Rollback* destas transações, se estiver sendo usada a recuperação *Fast-Start*, somente será executado o *Rollback* no bloco de dados que estiverem com problema, ao invés de ser feito na transação inteira [Oracle, 1].

#### 5.2.4 Falha de mídia ou de disco

- Ocorre quando um disco é corrompido, isto é, quando ocorre uma falha física do disco;
- Arquivos de dados, de log e arquivos de controle podem ser danificados com esse tipo de falha;
- Falhas de disco, exigem que sejam restabelecidos arquivos de dados de uma base que possua informações mais recentes, de antes da ocorrência da falha, pra que isso aconteça, é necessário que existam as cópias de segurança;
- Se algum arquivo de dados foi danificado mas não compromete toda a base de dados, a base pode continuar funcionando enquanto algumas *tablespaces* são recuperadas individualmente;
- A operação da base de dados, após a ocorrência de uma falha de mídia, depende de se ter ou não um arquivo *multiplexed*, que é uma segunda cópia do arquivo. Se a falha danifica um único disco e existe um arquivo *multiplexed*, a base de dados pode continuar operando normalmente.
- A falha de mídia pode ser dividida em duas categorias: erro de leitura e erro de gravação.

- O *Oracle* descobre que ocorreu um erro de leitura quando não consegue ler um arquivo de dados. Um erro do Sistema Operacional é retornado ao aplicativo e um erro do *Oracle* indicando que o arquivo não pode ser achado ou aberto. O *Oracle* continua a executar, mas o erro é retornado até ser corrigido.
- No próximo *checkpoint*, ocorrerá um erro de gravação quando o *Oracle* tenta escrever o cabeçalho de um arquivo como parte do *checkpoint*, retornando um erro no DBW passando o arquivo de dados off-line automaticamente, porém, somente o arquivo de dados que está com erro se torna off-line, o resto da tablespace continua on-line. Se o erro continuar persistindo, o *Oracle* fecha a instância [Sarin,2000].

### 5.2.5 Falha de Comunicação

- Quando o sistema usa rede local ou linha telefônica para a comunicação entre estações de clientes e servidores de base de dados, ou ainda, para conectar vários servidores de base de dados para formar um banco de dados distribuído, podem ocorrer falhas. Estas podem ser de linha telefônica ou de software de comunicação, interrompendo a operação normal de um sistema de base de dados;
- Quando ocorre uma falha de comunicação que interrompe a execução de um aplicativo de cliente e causa uma falha no processo que foi gerado, o processo PMON descobre e soluciona o processo que foi abortado;
- Esse tipo de falha, pode interromper o protocolo de *Two-Phase Commit* em uma transação distribuída. Após o problema ser corrigido, o processo do *Oracle Reco* de cada servidor da base de dados que está envolvido, soluciona automaticamente qualquer transação que ainda não tenha sido resolvida em todos os nós do sistema distribuído [*Oracle*, 2] .

### 5.3 Estruturas usadas para a Recuperação da Base de Dados pelo *Oracle*

Várias estruturas de proteção contra possíveis falhas são utilizadas pelo *Oracle*. A seguir serão descritas algumas delas:

### 5.3.1 Cópias de segurança da Base de Dados

Uma das formas de garantir a segurança dos dados contra possíveis falhas é a cópia de segurança dos arquivos físicos. O *Oracle* usa esse tipo de proteção, para restabelecer arquivos de dados ou arquivos de controle que foram danificados.

#### 5.3.1.1 Backups do Sistema Operacional

O *Oracle* trabalha com alguns backups a nível de sistema operacional, isto é, são backups realizados através de comandos executados no próprio ambiente do sistema operacional, comandos esses, executados pelo DBA do banco de dados. A seguir serão descritos alguns backups utilizados pelo *Oracle*.

1. *Cold Backups* – Esse tipo de backup realiza cópia de todo o banco de dados, porém, para que esse tipo de backup possa ser efetuado, é necessário que o banco de dados tenha sido fechado no modo normal, isto é, sem que alguma instância tenha sofrido algum tipo de pane;
2. *Hot Backups* – esse backup realiza a cópia do banco, enquanto o banco de dados está no modo aberto, isto é, quando ainda tem usuário utilizando o banco. Esse backup somente pode ser utilizado quando o banco estiver no modo *Archivelog*, isto é, quando os registros *Redo* on-line estiverem armazenados em disco. Um problema encontrado no *Hot backup*, é quando ocorre uma pane com uma tablespace que está nesse modo, pois o banco não pode ser aberto na reinicialização, porque o *Oracle* vai dizer que os arquivos de dados daquele tablespace, precisam ser recuperados porque estão no modo de backup. Pelo motivo exposto acima, sempre que houver necessidade de realizar backups do tipo *Hot*, é necessário repetir o processo para cada tablespace individual [Sarin,2000].



### 5.3.1.2 Backups Lógicos

Esse tipo de backup, contém uma cópia dos elementos de dados, porém não possui os blocos físicos que compõem o banco de dados. Existem vários tipos de backups lógicos que são utilizados, alguns serão descritos a seguir:

- Backup em arquivos de texto – são backups realizados de linhas ou de colunas que são selecionadas através de comandos do SQL Plus e gravadas em arquivos em formato texto;
- Backup do arquivo de controle lógico – esse tipo de backup envolve a geração do texto do comando para recriar o arquivo de controle, ao contrário do backup do arquivo físico, que faz a copia do próprio arquivo de controle físico.

### 5.3.1.3 O *Redo* Log

Os dados das estruturas lógicas são armazenados fisicamente em arquivos de dados ou *datafiles*. O *Oracle* é responsável por controlar o acesso a estes arquivos, tanto a nível de requisições de bloqueio quanto a restrições de segurança, sendo que esse controle é realizado sobre as estruturas lógicas. Uma estrutura lógica pode ocupar mais de um *datafile*, por exemplo, uma *tablespace* ou um índice podem corresponder a vários arquivos físicos. Os dados que estão contidos nos arquivos físicos não são mais os atuais, estando alguns segundos atrasados em relação aos arquivos de *log*.

Sempre que um usuário requisita dados, o *Server Process* verifica se o dado já está no *Database Buffer Cache*. Se o dado não estiver nele, o *Server Process* lê os blocos apropriados nos arquivos de dados para o *Database Buffer Cache*, onde é armazenado em *buffers*. Se os usuários modificam os dados requisitados, as mudanças ocorrem nos blocos de dados do cache, sendo que a nova versão dos dados é conhecida como *after image* ou imagem posterior, pelo fato de ser a imagem do dado após a alteração [*Oracle*, 3].

O *Oracle* salva a versão original dos dados num bloco de reconstrução (*rollback*). Esta versão é conhecida como *before image* (imagem anterior), por conservar os dados como estavam antes da modificação feita pelos usuários. Se o usuário desistir da alteração antes de executar uma operação de gravar as alterações no

banco (*commit*), a transação pode ser desfeita (*rolled back*), utilizando a informação no bloco de *Rollback*.

Durante o processamento da transação, o *Server Process* grava os blocos *before image* e *after image* no *Redo Log Buffer*. Esta informação é utilizada para desfazer uma transação ou para a recuperação do banco de dados em situações de falha (*recovery*).

O *Database Buffer Cache* é organizado em duas listas: a *dirty list* e a *Least Recently Used list (LRU list)*. A *dirty list* contém *buffers* modificados que ainda não foram gravados no disco, e a partir desta lista, que o DBWR copia os dados para os arquivos em disco. Esse processo é executado periodicamente [Charles, 1999].

Conteúdo da *LRU list*.

- *Free buffers* (buffers livres) – contém dados que não foram modificados desde que foram trazidos ao *buffers cache*;
- *Pinned buffers* (buffers em uso) – *buffers* que estão sendo acessados no momento. Após o acesso podem continuar livres ou tornar-se “sujos”;
- *Dirty buffers* (buffers “sujos”) – contem dados que já foram modificados. Para retirá-los da memória é necessário atualizar os dados correspondentes no disco [Sarin,2000] .

O arquivo de *Redo log* no *Oracle*, registra todas as mudanças ocorridas nos dados. Uma base de dados do *Oracle* consiste em pelo menos dois ou mais arquivos log: o *Redo log online* e *Archived Redo log*. Sempre que um deles estiver cheio, o servidor começa a escrever no outro. Um único arquivo de log causaria problemas, porque não haveria a possibilidade de sobrescrever as entradas de log antes que estas fossem comprometidas, ou então, haveria a necessidade de esperar até que essas entradas fossem levadas para os arquivos de dados.

#### **5.3.1.3.1 O Redo log on-line**

A base de dados do *Oracle* tem um arquivo de *Redo log on-line* associado a ela. O processo do *Oracle LGWR* usa esse arquivo, para registrar as mudanças que foram feitas e qual foi a instância que realizou as mudanças. Cada registro do *Redo log on-line* contém ambos os velhos e os novos valores, sendo que o registro velho é armazenado em um bloco de *Rollback*.

O *Redo* log on-line, possui dois ou mais arquivos que são reusados para registrar mudanças que ocorrem na base de dados de forma contínua [Oracle, 3].

#### 5.3.1.3.2 O *Redo* log Buffer

É um buffer circular contendo as informações sobre mudanças ocorridas no banco de dados. Ele é utilizado na reconstrução das mudanças feitas ao banco de dados quando uma recuperação do banco é requisitada. A recuperação do banco é possível devido aos segmentos de *Rollback*, que são áreas de memória que contém a informação anterior e a informação atualizada de todos os blocos de memória alterados.

O *Redo Log Buffer* pode ser ignorado (*bypassed*) pelo *Oracle Data Loader* (aplicativo para conversão de dados de outros bancos de dados para uma base *Oracle*), ou usa a palavra reservada (*keyword*) *UNRECOVERABLE* nos comandos *CREATE TABLE* e *CREATE INDEX* da linguagem SQL do *Oracle* [Oracle, 3].

#### 5.3.1.3.3 O Archived *Redo* Log

A base de dados do *Oracle*, pode ser configurada para armazenar arquivos de *Redo* Log On-line sempre que eles estiverem cheios. Esses arquivos são identificados por números de seqüência de log.

Esses arquivos são mais utilizados em recuperação de mídia da seguinte maneira:

- quando o Archived *Redo* Log está ativo e um arquivo de log fica cheio, isto faz com que o membro mais antigo do grupo de log seja transferido para outro arquivo, se possível outro disco;
- caso isso não aconteça, provavelmente os registros de log de transações mais novas, sobrescrevem os registros antigos. Se houver uma perda de dados, fica impossível recuperar o estado do Banco de Dados, tendo que retornar à última cópia de segurança.

### 5.3.2 Trabalhando usando o Gerente de Recuperação (RMAN)

O Recovery Manager (RMAN) é uma ferramenta de DBA fornecida pela *Oracle*, usada para gerenciar as operações de recuperação e backup. O RMAN pode ser

utilizado para fazer o backup de todo o banco de dados ou de apenas alguns componentes do banco de dados, por exemplo, os tablespaces, os arquivos de dados, os arquivos de controle e dos registros arquivados [Sarin,2000].

A função mais importante exercida pelo RMAN é que ele possibilita que sejam executados backups incrementais do banco de dados, isto é, são realizados backups apenas dos blocos de dados que sofreram mudanças desde o último backup completo. O RMAN possui um catálogo de recuperação que é o local onde ficam armazenadas informações sobre os backups realizados, restaurações e recuperações efetuadas.

Os diferentes tipos de backups são demonstrados na tabela 5.1 a seguir:

Tipo de Backup	Descrição
Full	Realiza o backup de todos os blocos de dados.
Incremental	Realiza o backup apenas dos blocos de dados que sofreram alterações desde o último backup realizado.
Open	O backup é feito com o banco de dados aberto
Closed	O backup é realizado com o banco de dados montado, mas não aberto.
Consistent	Esse tipo de backup é feito com o banco de dados montado, mas não aberto e com o banco de dados fechado de forma normal, isto é, sem ter sofrido nenhum tipo de pane antes da montagem.
Inconsistent	Esse backup é realizado quando o banco de dados está aberto ou montado depois de ter sido fechado de forma anormal ou ter sofrido algum tipo de pane. Backups deste modo, necessitam de recuperação.

Tabela 5.1. Tipos de backups realizados pelo RMAN [Sarin,2000].

### 5.3.3 Segmentos de *Rollback*

Em geral armazenam os valores velhos dos dados que foram mudados por alguma transação que não realizou *Commit*. São usados para várias funções na operação de uma base de dados do *Oracle*, principalmente para desfazer qualquer transação que não efetuou *Commit*, voltando dessa forma o banco de dados a ficar de forma consistente após a ocorrência de uma falha.

### 5.3.3.1 Rolando para Frente (Rolling Forward) e Rolando para Trás (Rolling Back)

O SMON grava os buffers de base de dados no disco, somente quando houver necessidade, por isso, dois problemas podem resultar de um fracasso de instância:

1. Os blocos de dados modificados por uma transação, não podem ser escritos no arquivo de dados e somente estão presentes no arquivo de *Redo Log*. Portanto o arquivo de *Redo Log*, será utilizado para reaplicar as mudanças durante o processo de recuperação.
2. Depois da fase Rolling Forward, o arquivo de dados pode conter mudanças, as quais não sofreram *Commit* durante o momento de fracasso. Estas mudanças devem ser desfeitas, para assegurar a consistência da transação.

Dois passos são utilizados pelo *Oracle* para resolver esse impasse, primeiro é efetuado o Rolling Forward juntamente com a *cache recovery* e depois o Rolling Back com a *transaction recovery* [Sarin,2000].

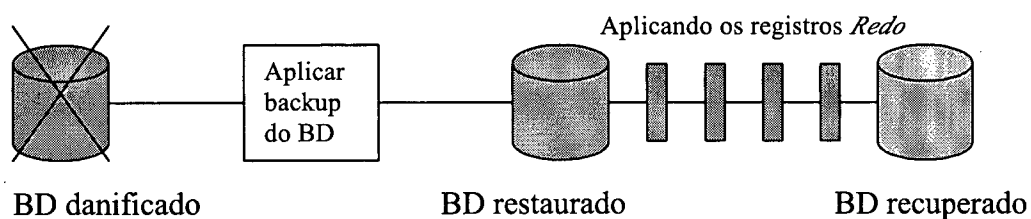
1. O primeiro passo consiste em reaplicar todas as mudanças registradas no arquivo de *Redo Log* para o datafiles.
2. Em seguida o Rolling Forward prossegue através de todos os arquivos de *Redo Log*, transformando a base de dados em um estado consistente, utilizando normalmente o Online *Redo Log* e quando necessário o Archived *Redo Log*.
3. Quando é utilizado o segmento de *Rollback*, as transações que foram alteradas são desfeitas, isto é, depois de ser usado o Rolling Forward se ainda existirem transações que não foram efetivadas. Este processo de desfazer mudanças é chamado de Rolling Back.

Em uma operação normal do banco de dados, as alterações realizadas são feitas através de declarações SQL ou PL/SQL. A partir do momento que uma transação é disparada, o processo de servidor analisa e executa a declaração, lendo o bloco de dados que foi requerido nos buffers de memória e grava as alterações nos buffers do bloco de dados. Esses buffers que foram modificados, são marcados como “Dirty” e são movidos para a “Dirty List”. O processo DBWR gerencia os buffers de banco de dados, com base em um algoritmo chamado de LRU, garantindo com isso, que os blocos de dados mais

utilizados permaneçam na memória, enquanto que os menos usados sejam tirados do disco [Charles, 1999].

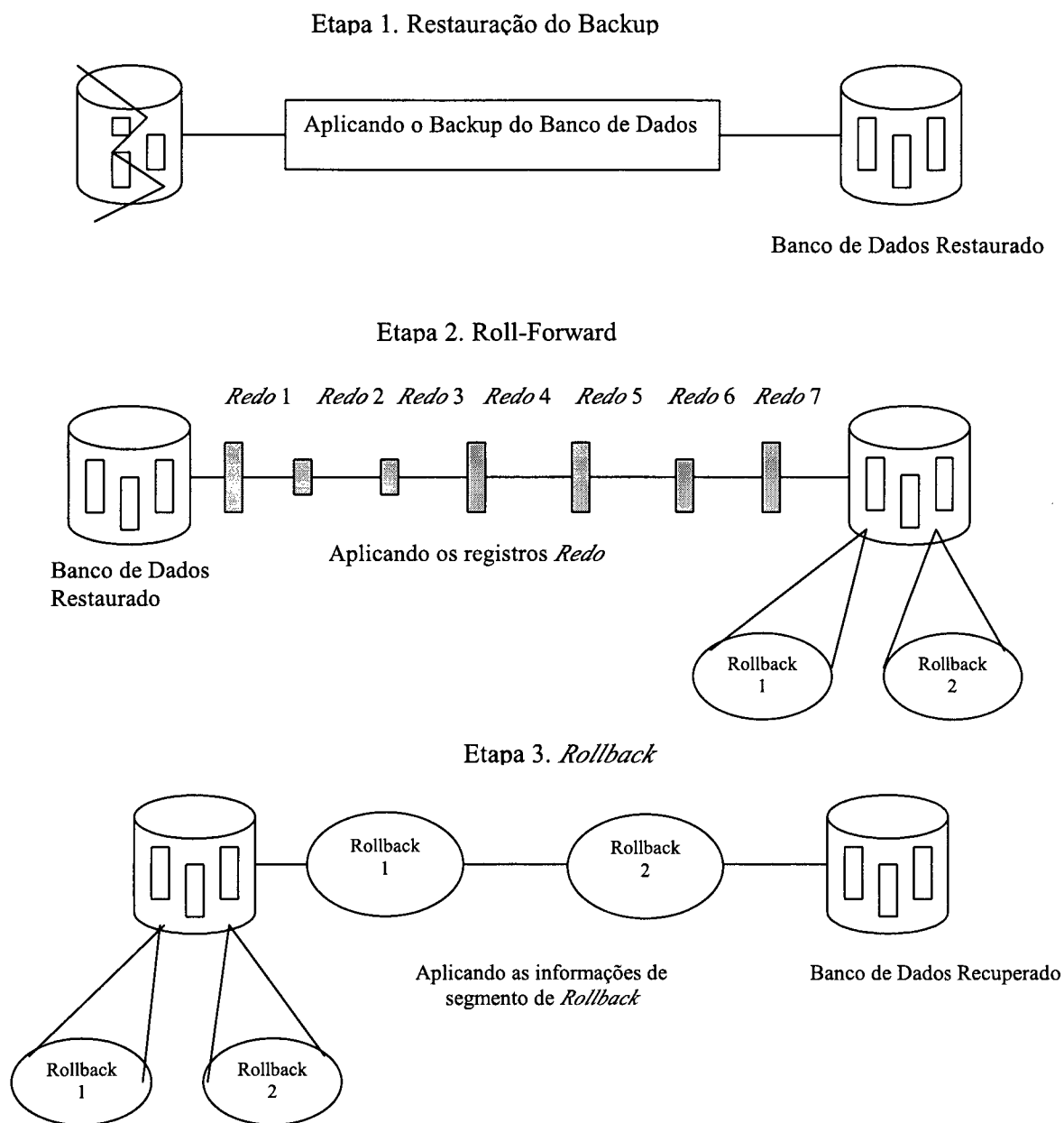
São também armazenadas as alterações nos buffers do registro *Redo* na SGA. O *Oracle* grava os dados *Undo* dos segmentos de *Rollback*. Desta forma, quando o usuário faz o *Commit* das alterações ou quando uma declaração faz o *Commit* implícito, elas se tornam permanentes. O processo LGWR grava as alterações feitas nos arquivos de registro *Redo* a partir dos buffers do registro *Redo*. Sempre que uma alteração é feita, é armazenado um número que ajuda o *Oracle* a controlar as alterações e a seqüência na qual elas foram realizadas [Sarin, 2000].

Se ocorrerem falhas que ocasionem perda dos dados, é necessário primeiro, restaurar o backup dos dados e segundo, utilizar os registros *Redo* para executar a recuperação. Essas duas fases distintas, restauração e recuperação, são apresentadas na figura 5.2 a seguir.



**Figura 5.2. Fases de restauração e recuperação [Sarin,2000].**

Durante a execução da operação de recuperação, todas as alterações que foram ou não efetivadas e que estão armazenadas nos registros de *Redo*, são aplicadas aos arquivos de dados através do processo de Roll-forward. As alterações feitas nas transações que não foram efetivadas, são desfeitas durante a fase de *Rollback*, usando para isso as informações de *Undo* que estão armazenadas nos segmentos de *Rollback*. As fases de roll-forward e *Rollback* da recuperação do banco de dados são descritos na figura 5.3 a seguir.



**Figura 5.3. Fases de Roll-forward e *Rollback* da recuperação de banco de dados [Sarin,2000].**

#### **5.4 Banco de Dados *Hot Standby***

A capacidade de um sistema alternar para um banco de dados *Oracle* alternativo quando a instância primária se torna indisponível é chamada de *Failover*. É uma forma

de manter uma cópia de backup sempre atualizada do banco de dados de produção em um servidor separado, em *standby*.

Quando ocorrer de a instância de produção falhar, o banco de dados de *standby* pode ser recuperado e tornar-se disponível para os usuários continuarem utilizando o banco em um curto período de tempo, isto é, o banco de dados *standby* precisa de um tempo para se recompor e ficar igual ao banco que estava em produção.

O banco de dados *Hot Standby* é uma cópia *standby* do banco de dados que está sendo utilizado, desta forma o banco de dados *standby* é mantido atualizado através dos registros dos arquivos de *Redo* que são aplicados constantemente ao banco. Assim sendo, o banco de dados *standby* estará em um modo de recuperação constante até que ele seja ativado e colocado a disposição dos usuários após um failover [Sarin, 2000].

Neste capítulo foi apresentada algumas características referentes ao *Oracle*, como a sua arquitetura, os principais processos que são utilizados quando o banco de dados precisa efetuar a recuperação da base independente do tipo de falha que aconteça.

Normalmente nas falhas que são menos graves, a recuperação é feita automaticamente pelo banco no momento em que a instância é reinicializada, porém, existem algumas falhas que precisam ser recuperadas utilizando alguns recursos que o *Oracle* disponibiliza, como por exemplo a recuperação utilizando o *RMAN*, ou então a recuperação utilizando o método de *Roll-forward*, que possibilita que sejam executados os métodos *Redo* e *Undo*, isto é, desfazer as transações que não efetuaram *Commit* e refazer as transações que efetuaram *Commit*. O *Oracle* disponibiliza ainda, a recuperação de transações distribuídas utilizando o protocolo *2PC*.



## Capítulo 6

### Banco de Dados - *Ingres*

#### 6. *Ingres*

*Ingres* é considerado como um banco de dados completo, possibilitando que os clientes possam utilizar os dados em um vasto campo de empreendimento, acesso distante e disponibilidade de replicação alta. As novas versões do *Ingres*, como o *Ingres II* oferece soluções através de uma variedade de plataformas integrais para eBusiness como *Windows*, *Unix*, *OpenVMS* e *Linux*. *Ingres II* voltado para o eBusiness possui capacidades e ferramentas de desenvolvimento de aplicativo que aumentam o desempenho, dando ao usuário a velocidade que é necessário para integrar fontes de dados e aplicativos diversos sem sacrificar integridade de informações.

O banco de dados *Ingres* foi desenvolvido pela *Computer Associates (CA)* para solucionar os problemas de recursos de informação, controlando os empreendimentos que requerem uma infra-estrutura de tecnologia sólida e soluções de administração de banco de dados flexíveis que constantemente apóiam aplicações críticas e recursos de dados [*Ingres*,1].

#### 6.1 Banco de dados

O backup da base de dados do *Ingres* é essencial para manter a integridade dos dados e a segurança dos mesmos. Desta forma, dados valiosos podem ser recuperados caso ocorram falhas ou um desastre. O backup é utilizado para recuperar o banco de dados para um ponto, antes de ter ocorrido o fracasso ou a falha.

O *Ingres* possui o recurso de executar uma recuperação completa ou uma recuperação parcial. A recuperação completa permite que toda a base de dados seja recuperada de algum dispositivo, ou que sejam recuperadas algumas tabelas individuais, já a recuperação parcial permite que sejam restaurados alguns dados de uma ou mais tabelas.

O *Ingres* Star é um gerente de dados distribuídos que inclui armazenamento, acesso e processamento distribuído, permitindo que diversas bases de dados separadas criem uma visão única dos seus dados para o usuário [*Ingres*, 2].

O banco de dados *Ingres* Star recebe pedidos de dados de múltiplos clientes e passa os pedidos para o servidor local (SGBD). O servidor local sustenta acessos simultâneos à base de dados e questões locais do servidor Star. O SGBD local, acessa a base de dados e obtém os resultados desejados, passando essa informação para o servidor Star, que devolve a informação para a aplicação distribuída.

O *Ingres* possui um sistema que é chamado de Logging. Esse sistema, armazena as transações que ocorreram no banco de dados automaticamente e, é formado por:

- Arquivo de log de transação;
- Processo de recuperação (dmfrcp);
- Processo de arquivar (dmfacp);

### **6.1.1. Arquivo de Log de Transação**

O *Ingres* possui dois arquivos de log alternados e habilitados. Se ocorrer uma falha de mídia em um dos logs, isso não resultará na perda de dados ou na interrupção de serviço, isto é, se um dos discos que contém os arquivos de log falhar, o sistema automaticamente interrompe o acesso a esse disco e passa a acessar o outro arquivo de log sem que a aplicação seja interrompida [*Ingres*, 2].

O sistema de logging assegura que os registros de log fiquem descritos de uma forma que torne fácil o acesso para a recuperação e o processo de gravar o arquivo de log no disco. Sempre que uma transação efetua um *Commit*, o arquivo de log move o buffer de log, que está residente em uma memória compartilhada, para o arquivo de log de transação.

Durante o processamento normal, isto é, on-line, é reservado um espaço para o arquivo de log de transação, para efetuar um *Rollback* se for necessário. Nesse espaço, são armazenados os registros de log de compensação, que descrevem os passos executados durante o *Rollback*.

### 6.1.2. Processo de Recuperação

O processo de recuperação (dmfrcp) permite o acesso e a restauração de sistemas ou servidores que tenham sofrido algum tipo de falha. O sistema de logging escreve pontos de consistência no arquivo de log de transações, para assegurar que todas as bases de dados estejam consistentes até o ponto em que ocorreu a falha. Desta forma, quando estiver sendo efetuado o *Rollback* da transação, o usuário poderá continuar a trabalhar na base de dados [Ingres, 3].

O processo de recuperação é semelhante a um SGBD normal, porém não sustenta as conexões dos usuários, o processo permanece ativo sempre que o usuário estiver ativo.

A recuperação on-line é executada quando o servidor para de trabalhar anormalmente, neste caso, os usuários que estão conectados a outros servidores não são atingidos.

### 6.1.3. Processo de arquivar log de transações

O processo de arquivar (dmfacp) remove as transações que efetuaram *Commit* do log de transações e escreve em tabelas individuais os arquivos de log, que são chamados de arquivos de logs diários, correspondentes à base de dados.

Desta forma cada base de dados tem seu próprio arquivo diário, que contém um registro de todas as mudanças que foram realizadas na base de dados desde o último checkpoint.

O processo de arquivar, somente é acionado quando o log de transações está pronto para ser arquivado, ou então, quando o último usuário se desconecta do banco de dados.

## 6.2. Tipos de Backup

O banco de dados *Ingres* trabalha com a seguinte metodologia em se tratando de Backup e Recuperação [Ingres, 4]:

- a) *Backup S.O.* – é um método utilizado pelo sistema operacional, que salva a estrutura do *filesystem* em algum dispositivo em local remoto;

- b) *Unloaddb* – é um comando do *Ingres* utilizado para gerar cópias da base de dados em arquivos do tipo simples, isto é, um arquivo que é formado de registros de mesmo tipo sem que exista uma estrutura interna definindo as relações entre os registros;
- c) *Checkpoint* – é utilizado um comando do *Ingres* chamado *ckpdb* que serve para criar pontos de controle (checkpoint) juntamente com os arquivos de log;
- d) *Rollforward* – é o ato de recuperar uma base de dados que se encontra no estado de inconsistência até o ponto em que a base se encontrava antes de ocorrer a falha;
- e) *Pontos de consistência* – são marcas colocadas no arquivo de log, que serve para ajudar no momento de efetuar uma recuperação;
- f) *Log de transação* – é usado para assegurar que as transações foram ou não efetivadas. Todas as transações são “anotadas” no arquivo de log, permitindo que o *Ingres* possa executar, se houver necessidade, a recuperação da base. É utilizado um arquivo circular que à medida que vai ficando sobrecarregado é descarregado pelo *archiver* em um dispositivo qualquer;
- g) *Journals* (registro cronológico/diários) – é criado pelo processo *archiver*, o qual permite que a recuperação seja executada no caso de um fracasso do dispositivo de armazenamento da base de dados causando falha, permitindo assim que a base de dados possa ser recuperada até a última transação que foi efetuado comit ;
- h) *Archiver* – é um processo do *Ingres* que permite ler o arquivo de log de transação, eliminando as transações que foram retiradas da memória, assim o *archiver* copia as informações das transações para um arquivo de *Journals*, para mais tarde ser utilizado em caso de uma auditoria de recuperação;
- i) *Backup Parcial* - o *Ingres* possui um método de backup, que é chamado de backup da base de dados parcial, e, é utilizado quando há necessidade de executar um backup de tabelas individuais.

### 6.2.1 Backup de Sistema Operacional

Um backup do S.O. permite que sejam efetuadas cópias de arquivos em determinados diretórios, para que mais tarde possam ser recuperados para uso dos usuários, porém, o *Ingres* não permite que sejam restabelecidos arquivos individuais, pelos seguintes motivos a saber:

- 1) as tabelas do banco de dados do *Ingres* e seus arquivos de índices, devem estar em sincronização a todo instante;
- 2) durante a execução de um backup do S.O., o *Ingres* pode alterar o conteúdo de uma tabela e do índice, e isso não será refletido na cópia que está sendo realizada;
- 3) o *Ingres* refaz o relacionamento entre as tabelas sempre que uma tabela é reorganizada pelo DBA, sendo que este relacionamento é armazenado em outra tabela.

O backup do S.O. só pode ser utilizado para recuperar uma base de dados do *Ingres* quando esta for executada e o *Ingres* estiver “fechado”, isto é, não tiver mais nenhum usuário utilizando o banco.

A única forma de recuperar um banco de dados é recuperando completamente todos os relacionamentos que existem com aquela base de dados, inclusive os *Journals*, *Checkpoint*, *Dump* e outros [*Ingres*, 4].

### 6.2.2. Utilização do *Unloaddb*

É um método do *Ingres* que possibilita a execução de cópias da base de dados para arquivos em um determinado diretório, e que mais tarde possam ser utilizadas pelo DBA caso haja necessidade de recarregar a base.

Esse procedimento de recarregar a base, pode ser feito apenas com tabelas individuais, ou então com toda a base, inclusive arquivos de índice, procedimentos, regras e outros [*Ingres*, 1].

### 6.3. Processo de Recuperação

A maioria dos locais utilizam os comandos “*ckpdb*” e “*rollforwarddb*” para efetuar o backup e recuperar completamente a base de dados.

O *Ingres* mantém um caminho das últimas 16 tentativas de Checkpoint, utilizando um arquivo de configuração (aaaaaaa.cnf) para fazer isso. É um arquivo binário que não é visualizado pelo usuário [*Ingres*, 3].

#### 6.3.1. Checkpoint

O *Ingres* possui um comando *ckpdb* que é utilizado para recuperar a base de dados, controlando os eventos que acontecem durante a operação, sendo que é o único comando que pode ser utilizado para recuperar a base de dados, enquanto a mesma está sendo utilizada pelos usuários. Pelo menos 99% das plataformas do *Ingres*, utilizam esse método de recuperação [*Ingres*, 2].

A seguinte sequência de eventos, ocorre quando é utilizado o comando *Ckpdb*:

1. o *checkpoint* utiliza um *lock* na base de dados, para prevenir que dois pontos de *checkpoint* possam ocorrer de uma vez;
2. quando é feito um checkpoint off-line, um lock exclusivo é imposto à base de dados, impossibilitando dessa forma o acesso à base, e esperando que todos os usuários da base de dados, se desconectem para prosseguir;
3. se o checkpoint é off-line, o comando *archiver* ACP é sinalizado para descarregar (*dump*) todas as transações que efetuaram *Commit* do arquivo de log para o arquivo de *Journals*, e, se por um acaso o ACP não estiver ativo o checkpoint falhará;
4. se o checkpoint é on-line, o sistema de logging não permite que novas transações sejam inicializadas, e somente as transações que já estavam em andamento podem continuar para efetuar um *Rollback* ou um *Commit*. O comando checkpoint ficará aguardando até que todas as transações ativas estejam completas, podendo ficar nesse estado por tempo indeterminado;
5. quando o checkpoint é on-line, é aplicado um lock de controle às tabelas da base de dados, não permitindo com isso, que ocorram modificações nas tabelas (criação, alteração, drop, etc) enquanto o checkpoint é criado;

6. junto com o checkpoint on-line é criada uma marca no arquivo de log de transação, que informa o “início do checkpoint”, criando um ponto de consistência. O ACP então, descarrega todas as transações que efetuaram *Commit*, e após isso, o sistema de logging permite que novas transações sejam inicializadas;
7. se o checkpoint é on-line, o ACP escreve as transações que foram inicializadas antes da marca de início do checkpoint, em um arquivo de *Journals* chamado de “arquivo cronológico/diário velho”. Um novo arquivo de diário é aberto, para receber as transações que foram inicializadas após a marca do início do checkpoint;
8. o controle é passado para o comando *ckptmpl.def*, que executa a cópia dos arquivos para um dispositivo de armazenamento que pode ser fita, disco ou outro qualquer. Cada arquivo possui um “alias”, para que a localização possa ser executada de forma mais rápida;
9. quando o checkpoint é on-line, uma marca de “fim de checkpoint” é escrita no arquivo de log de transação, e o checkpoint espera o ACP terminar de escrever as transações no arquivo de *Journals*;
10. o arquivo (aaaaaaa.cnf) é copiado com as novas informações do checkpoint, para a área de *II\_Dump*; caso ocorram falhas, o arquivo está salvo;
11. todos os locks que foram gerados pelo checkpoint são liberados;
12. se o checkpoint for no modo off-line, o lock da base de dados exclusiva é liberado permitindo que os usuários possam acessar a base de dados [*Ingres*, 4].

### 6.3.2. Journalling

*Journalling* pode ser definido como a gravação de “mutações” que acontece quando existe um padrão de comparação, por exemplo o checkpoints, em arquivos cronológicos. O *archiver* (ACP) é responsável por escrever os registros da imagem do arquivo de log nos arquivos de *Journals* [*Ingres*, 2].

### 6.3.3. Rollforwarddb

*Rollforwarddb* é um método de recuperação, que permite o *Ingres* controlar os eventos que acontecem durante a operação de recuperação da base de dados. É o único método disponível para ser utilizado em conjunto com o comando *ckpdb* [*Ingres*, 4].

A seguinte seqüência acontece quando o comando *Rollforwarddb* é usado:

1. o *rollforwarddb* atribui um lock exclusivo à base de dados;
2. o arquivo de configuração (aaaaaaa.cnf), presente na área da base de dados, é renomeado para aaaaaaa.rfc;
3. todos os arquivos presentes nessa área são apagados, com exceção do arquivo aaaaaaa.rfc;
4. se a recuperação que estiver sendo executada for de um checkpoint on-line, o *archiver* ACP começa a escrever as transações que foram executadas antes da marca de início do checkpoint para um arquivo de “diário velho” (*journals*). Um novo arquivo de diário é aberto, para receber as transações que foram inicializadas após a marca de início do checkpoint;
5. o arquivo aaaaaaa.rfc é lido para achar as informações referentes ao último checkpoint, e esse último ponto é tomado como referência para executar a recuperação;
6. nesse momento, o controle é passado para o comando *ckptmpl.def* onde o programa de backup/restore esvazia o conteúdo do backup que foi feito em um dispositivo qualquer e restaura para o diretório apropriado;
7. é aplicado o arquivo de log que contém as estruturas das cópias que foram realizadas antes de esvaziar os arquivos, e estas, são aplicadas para evitar qualquer problema de inconsistência após a baixa do backup;
8. o arquivo aaaaaaa.rfc é renomeado novamente para aaaaaaa.cnf e o lock exclusivo da base de dados é liberada.



## 6.4. Transações Distribuídas

Uma transação distribuída, abrange mais de um local onde está situada a base de dados, porém, para o usuário, uma transação distribuída é como se fosse uma transação local única [*Ingres*, 3].

O *Ingres* Star, coordena as transações que englobam locais múltiplos, de modo que todos os locais envolvidos votem para efetivar a transação ou a transação não é efetivada, usando para isso o protocolo *Two Phase Commit*.

### 6.4.1. Protocolo Two-Phase *Commit*

O protocolo 2PC consiste nas duas seguintes fases [*Ingres*, 1]:

- 1) todos os *sites* em comum acordo, para efetuar um *Commit*,
- 2) efetuação do *Commit* propriamente dito.

O *Ingres* Star administra estas duas fases da seguinte maneira:

- Na primeira Fase:
  - o usuário envia uma declaração de *Commit*,
  - o *Ingres* Star envia um pré-commit para os *sites* que estão envolvidos na transação;
  - os *sites* devolvem a mensagem, informando que estão preparados;
  - o *Ingres* Star decide efetivar a transação;
  - se os *sites* devolvem a mensagem, informando que não estão preparados, o *Ingres* Star cancela a transação decidindo por abortá-la.
- Na segunda Fase:
  - O *Ingres* Star envia uma mensagem para os *sites*, informando que podem efetivar a transação;
  - A transação é efetivada.

Quando acontece qualquer tipo de falha, a transação é abortada independentemente da mensagem que foi enviada.

Existem alguns casos em que o 2PC não é utilizado. É o caso de transações que atualizam somente uma base de dados, ou de bases de dados que não conseguem

suportar o protocolo 2PC, isto é, todas as bases de dados envolvidas na transação estão no mesmo local.

Este capítulo apresentou o banco de dados *Ingres*, com suas principais características, os processos utilizados quando é necessário fazer a recuperação dos dados do banco, e, também os métodos de backups que são utilizados quando o banco é recomposto.

Quando as transações são consideradas como transações distribuídas, o *Ingres Star* coordena as transações que envolvem locais múltiplos, de modo que todos os locais que estão envolvidos na transação votam em efetivar ou não a transação, usando para isso o protocolo *Two Phase Commit (2PC)*.

## Capítulo 7

### Banco de Dados – *DB2*

#### 7. *IBM/DB2*

Em consequência do constante e rápido desenvolvimento, a *International Business Machines Corporation* criou em 1949 a *IBM World Trade Corporation*, uma subsidiária inteiramente independente, cujo objetivo era aumentar vendas, serviços e produção fora dos Estados Unidos [*DB2*, 2].

As atividades da *IBM World Trade Corporation* se estendem hoje por mais de 150 países. As fábricas e laboratórios da *IBM* funcionam em 15 diferentes países. São 29 laboratórios de desenvolvimento que, juntamente com os 5 dos centros de pesquisa pura onde são realizadas as mais sofisticadas pesquisas tecnológicas, onde foi desenvolvido o banco de dados *DB2*, que hoje é considerado um dos mais completos bancos de dados existente no mercado, [Banco de dados, 2].

Este capítulo tem por objetivo apresentar o banco de dados *DB2*, sua estrutura, seus métodos e recursos utilizados quando se trata de recompor o banco de dados para um estado de consistência. A recuperação de versão e a recuperação Roll-Forward, são dois métodos que o *DB2* utiliza para recuperar a base de dados.

Quando se trata de transações distribuídas, o banco de dados *DB2* implementa o protocolo Two-Phase Commit para restaurar as transações que sofreram algum tipo de pane e não puderam ser efetivadas.

#### 7.1 *DB2*

Comandos utilizados em transações no banco de dados e as aplicações, podem falhar por diversas razões. Uma falha de uma transação não é um fracasso de uma ação executada pelo banco de dados quando é utilizado um parâmetro incorreto no banco, ou um erro que fez com que o banco de dados terminasse de forma anormal, mas sim, uma

falha de hardware ou de *Commit/Rollback*, pelo fato do arquivo de log estar cheio e não existir um arquivo de log adicional para escrever o registro de *Commit/Rollback*.

Quando houver necessidade de efetuar o *Rollback* automático pelo gerente do banco de dados, em unidades incompletas de trabalho, na ocorrência de uma falha será necessário configurar o banco com o parâmetro de reinicialização automático, isto é, o *autorestart*. Se foi habilitado o *autorestart*, quando ocorrer uma falha, a recuperação do banco será inicializada automaticamente juntamente com os arquivos de log onde ficam armazenadas as informações referentes à recuperação.

A recuperação de falha será então executada pelo utilitário do banco de dados, sempre que ocorrer uma finalização anormal do banco. Qualquer transação que estiver efetuando um *Commit*, estando no pool de buffers do banco de dados, significa que não foi gravada no momento da falha [DB2, 3].

Quando o *autorestart* não estiver habilitado, qualquer aplicação que tentar se conectar ao banco de dados e necessitar de uma recuperação de falha executada, isto é, que precise ser reinicializada, receberá um erro: SQL1015N. Desta forma será necessário selecionar a ferramenta do banco que faz a reinicialização automática da ferramenta de recuperação.

Os arquivos de log são usados para recuperar o banco de dados de erros de aplicativos ou de sistema. Quando são utilizados em combinação com as cópias de segurança, servem para recuperar a consistência do banco de dados, até o ponto onde o erro aconteceu. Os arquivos de história da recuperação, armazenam um resumo das informações das cópias de segurança, que podem ser utilizadas quando houver necessidade de recuperar todo ou uma grande parte do banco de dados.

Os arquivos de índices podem se tornar inválidos, na ocorrência de problemas de disco fatais. Se isto ocorrer, é necessária a recuperação desses arquivos, executando a recriação dos mesmos no momento em que o banco de dados é reinicializado, isto é, a hora que é executado o *Database Restart* [DB2, 3].

## 7.2 Identificando qual o servidor do Banco de Dados que está com Falha

Quando o servidor do Banco de Dados falha, a aplicação receberá um erro do SQLCODEs. Através desse erro, é possível identificar qual servidor está com falhas [documentação *DB2*]:

- SQL0279N – esse código é emitido, quando um servidor estiver envolvido em uma transação e ocorrer a falha durante o processo de *Commit*,
- SQL1224N – esse código é emitido, quando o servidor que falhou é o coordenador da transação;
- SQL1229N – esse código é emitido, quando o servidor que falhou não é o coordenador da transação.

Após ser determinado qual o servidor que está com problemas, é realizado um processo de recuperação do servidor, onde é necessário:

1. Corrigir o problema que causou a falha;
2. Reinicializar o gerente do banco de dados, utilizando o comando *DB2Start*;
3. Reinicializar o banco de dados, utilizando o comando de *Restart Database* [*DB2,1*].

## 7.3 Recuperando uma Base de Dados

Quando o banco de dados é finalizado com um erro, automaticamente é feita a recuperação da base de dados. As bases de dados que não podem ser recuperadas possuem um parâmetro de configuração (*logretain* e *userexit*), que afirma que a mesma está inválida. Isto significa que os únicos arquivos de log que existem, são os arquivos de log ativos, que contêm dados sobre a transação atual.

Porém, já com as bases de dados que podem ser recuperáveis, o parâmetro de configuração informa que as mesmas podem ser recuperadas, desta forma, os arquivos de log ativos, podem ser utilizados junto com os arquivos de log armazenados, que possuem informações sobre as transações que efetuaram *Commit* e a base pode ser recuperada até o ponto onde ocorreu o erro, com uma ressalva, a recuperação só pode acontecer quando a base de dados estiver off-line.

Se ocorrer de algumas transações serem inesperadamente interrompidas, a base de dados é encerrada com erro e conseqüentemente se encontrará em um estado de

inconsistência. É necessário mover o estado da base para um estado consistente e utilizável através dos algoritmos de *Redo* e *Undo*, isto é, desfazer as transações que não foram efetivadas e refazer as transações que foram efetivadas.

Quando a base de dados estiver em um estado consistente e utilizável, significa que atingiu o que é conhecido como “ponto de consistência”. Assim, todas as transações ficam resolvidas e os dados disponíveis para outros usuários ou aplicativos.

Existem dois métodos de recuperar uma base de dados danificada: a recuperação de versão e recuperação *Roll-forward* [DB2, 1].

### 7.3.1 Utilizando o método de recuperação de versão

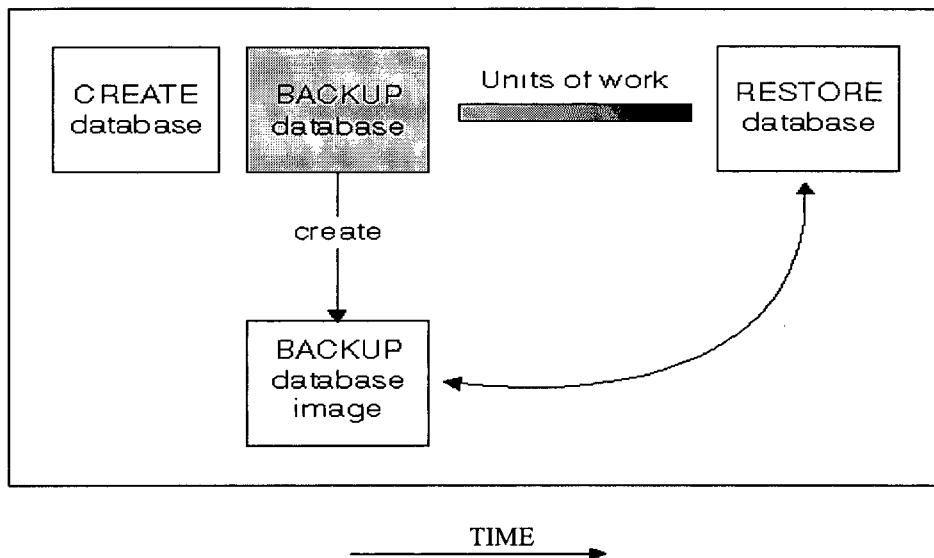
A recuperação de versão é a restauração de uma versão prévia da base de dados, usando para isso, uma imagem que é criada durante uma operação de cópia de segurança. A operação de reconstrução da base de dados inteira, utiliza uma cópia do Backup que foi feita anteriormente, permitindo assim, que a base de dados seja estabelecida para o estado idêntico ao momento em que era realizada a cópia de segurança.

O método de recuperação de versão, utiliza o comando de Backup junto com o comando de Restore, fazendo com que o banco volte ao estado que foi salvo previamente, usando para isso, os arquivos de log que estão armazenados. Ao se executar uma cópia do banco de dados, é utilizado o comando de Backup, ou pode-se selecionar o banco de dados a ser realizada a cópia, ou então as tabelas onde será executado o backup.

Em um sistema de banco de dados que está particionado, pode-se determinar quais são os servidores que vão realizar o backup e quais as tabelas a serem armazenadas. Em um sistema de banco de dados distribuído, o comando de Backup e *Restore* são aplicados às transações distribuídas [DB2, 1].

A recuperação de versão, utiliza as cópias de segurança que estão em off-line, para recuperar a base de dados que está no estado de irrecuperável. Esse tipo de recuperação, não permite que nenhum outro aplicativo possa usar a base enquanto a cópia de segurança estiver sendo realizada. A base de dados é recuperada até o ponto que estava quando foi realizada a última cópia de segurança. A figura 7.1 a seguir mostra como é feito a recuperação de uma base utilizando a recuperação de versão.

Sempre que esse método de recuperação for utilizado deve-se programar e executar cópias completas da base de dados, em uma base regular auxiliar.



**Figura 7.1. Restabelecimento de uma base de dados utilizando recuperação de versão— [DB2, 1]**

### 7.3.2 Utilizando o método de recuperação Roll-Forward

Esse método de recuperação utiliza o comando de Backup, mais o comando de Restore, junto com o comando de Roll-forward (refazer as transações que efetuaram *Commit*), permitindo que o banco ou uma tabela específica seja restabelecida, a um ponto específico, antes de ter ocorrido a falha.

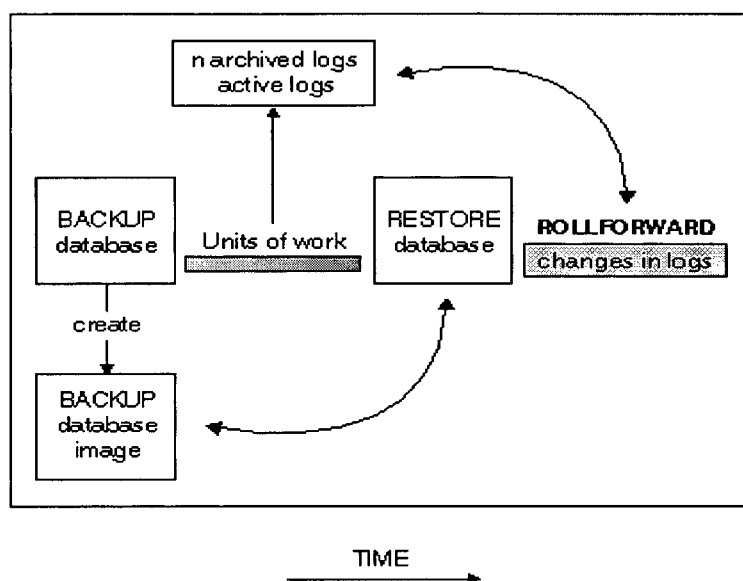
Quando o banco de dados é criado, e é informado que o arquivo de log é circular, os arquivos de log podem ser reusados e não são arquivados, o método de Roll-forward não pode ser utilizado. A única forma de recuperação que pode ser utilizada é a recuperação de versão.

Quando o arquivo de log arquivado é executado, significa que a recuperação Roll-forward pode ser executada, pelo fato de poder aplicar os arquivos de log armazenados, que contém os registros das últimas mudanças que ocorreram após o Backup ter sido executado.

Em um ambiente de banco de dados particionado, a base de dados é localizada através das partições que ocorreram. Quando é utilizado esse método de recuperação

junto com checkpoint, em todas as partições da base de dados deve ser executado o algoritmo de *Redo*, para assegurar que todas as partições encontram-se no mesmo nível [DB2, 1].

A figura 7.2 a seguir mostra como é feito o restabelecimento de uma base utilizando o método de recuperação Roll-forward.



**Figura 7.2. Restabelecimento de uma base de dados utilizando Roll-forward – [DB2,1]**

## 7.4 Arquivos de Log do DB2

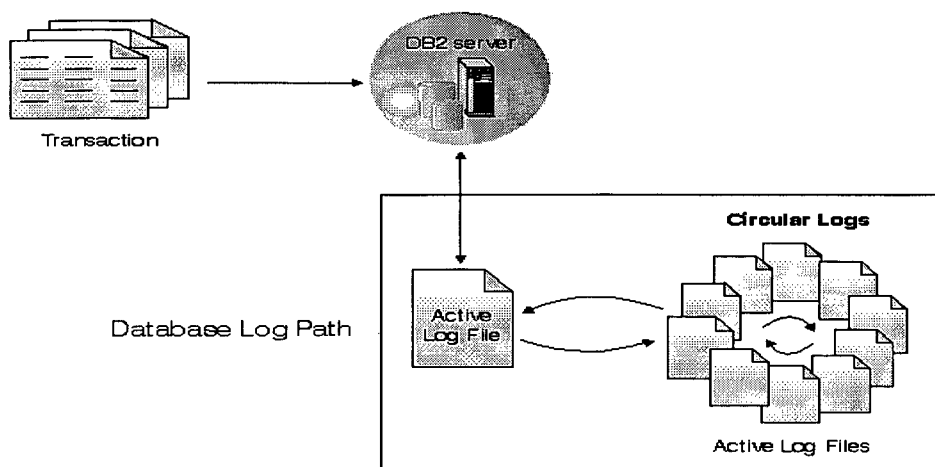
Todas as bases de dados tem arquivos de log associadas a elas. Esses arquivos são responsáveis em manter os registros de mudanças que ocorrem na base. Existem dois tipos de log no *DB2*: arquivo de log circular e arquivo de log arquivado.

### 7.4.1 Arquivo de log Circular

Sempre que uma nova base de dados é criada, o default é o log ser utilizado na forma circular. Como o próprio nome sugere log circular usa um “anel” de arquivos de log on-line, para fornecer a recuperação de falhas de transação ou falhas de aplicação. Os arquivos de log são usados e retidos somente para assegurar a integridade de transações atuais. O log circular não permite que seja utilizado o algoritmo de *Redo* em



transações anteriores a última cópia completa realizada, como é demonstrado na figura 7.3, [DB2, 1].



**Figura 7.3. Arquivo de log circular – [DB2, 1]**

Os arquivos de log ativos, são usados quando houver necessidade de uma recuperação para restabelecimento de uma falha ou para deixar a base em um estado compatível. As mudanças que foram efetivadas mas não foram fisicamente escritas da memória para o disco são refeitas, assegurando dessa forma a integridade da base de dados.

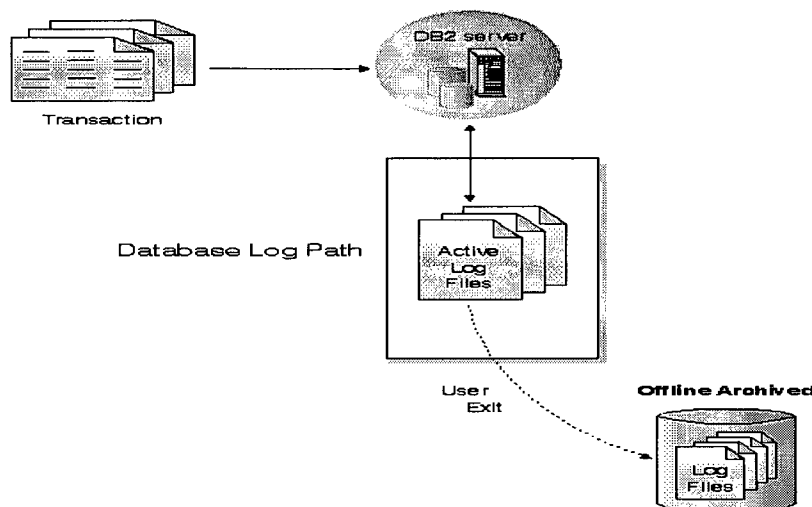
#### 7.4.2 Arquivo de log Arquivado

Os logs arquivados são utilizados quando a recuperação Roll-forward é executada, na ocasião em que mudanças do log ativo não são mais necessárias em um processo normal, com isso o log se torna um arquivo de log arquivado, podendo ser:

- **Logs arquivados on-line:** esse tipo de log é dito on-line, quando é armazenado no diretório de trabalho do log da base de dados.
- **Logs arquivados off-line:** esse tipo de arquivo é dito off-line, quando não mais se encontra no diretório de trabalho do log da base de dados.

A recuperação Roll-forward pode utilizar os arquivos de log ativos e arquivados para reconstruir uma base de dados, fazendo com que as mudanças efetuadas que estão armazenadas nos arquivos de log arquivados, sejam aplicadas nos arquivos de log ativos. A figura 7.4 apresenta como é trabalhado com o arquivo de log arquivado.

Quando é feita a recuperação através de Backup, deve ser aplicado nos arquivos de log o comando de *Roll-forward*, até alcançar o último checkpoint em que o Backup foi realizado [DB2, 1].



**Figura 7.4. Arquivo de log arquivado – [DB2, 1]**

### 7.5 Recuperação de falhas de transação utilizando o 2PC

Quando ocorre uma falha de comunicação de rede, o banco de dados fica particionado, isto é, fica dividido com um coordenador em cada bloco da divisão. Juntamente com o coordenador é eleito um *site* que trabalha como um gerente para a aplicação, e, é esse gerente, que distribui os serviços para os outros coordenadores, armazenando a informação de quem está envolvido na transação.

Utilizando o 2PC, o coordenador envia uma mensagem de *Commit* para os *sites* participantes, inclusive para os outros *sites* que estão sob a coordenação de um outro coordenador. Seguindo as regras do 2PC, os *sites* enviam a resposta, e se algum dos *sites* enviar a resposta com “Não”, é executado o *Rollback* imediatamente. Caso contrário, o coordenador entra na segunda fase do 2PC. O coordenador envia a mensagem de *Commit* aos *sites* e armazena esse registro no arquivo de log, e a transação é dita efetuada quando o coordenador recebe uma mensagem de todos os *sites* participantes, dizendo que o *Commit* foi realizado [DB2, 3].

Quando o servidor de uma determinada partição, descobre que o servidor de outra partição sofreu uma pane, o trabalho que envolve transações que são comuns a ambos os servidores é parada imediatamente. Desta forma:

- a) se o servidor que não está com falhas, for o coordenador da partição de uma transação, e esta transação ainda não está efetuada, isto é, não foi efetuado *Commit* da transação, o coordenador interrompe a transação imediatamente, para que possa ser efetuada a recuperação da falha. Se a transação já se encontra na segunda fase do 2PC, a aplicação que disparou essa transação recebe uma mensagem de erro do SQL, perdendo sua conexão com o banco de dados. Caso contrário, o coordenador envia uma mensagem de *Rollback* para todos os servidores que estão participando da transação;
- b) se o servidor que está com falhas, for o coordenador da transação, o processo é interrompido em todos os servidores que estão executando a transação, para que possa ser executado o processo de recuperação de falha. É utilizado o algoritmo de *Undo*, que desfaz as alterações que foram efetivadas pela transação;
- c) se o coordenador da transação não é o servidor que está com falhas, nem o servidor local, os gerentes que auxiliam os coordenadores são interrompidos e uma mensagem de *Rollback* é enviada a todos os servidores das partições do banco de dados, desfazendo os efeitos que foram causados pela transação.

Sempre que ocorrer um fim anormal, o servidor vai se encontrar em um estado inativo. Para que o servidor volte ao estado ativo é necessário executar uma recuperação de Crash que é ativada quando o banco de dados é reinicializado. Essa recuperação aplica o arquivo de log que está armazenado em cima dos arquivos de log que estão ativos, garantindo que todas as transações locais que não foram efetuadas sejam desfeitas, com exceção das transações que estão em um estado de incerteza, transações estas que não foram nem efetivadas nem abortadas.

Transações que estão em um estado de incerteza, são transações que talvez tenham sido efetivadas, mas não armazenaram a informação de *Commit* no registro de log. Isso acontece, porque o coordenador da transação não recebeu a mensagem de *Commit* de todos os *sites* participantes da transação [DB2, 3].

A recuperação de Crash, tenta solucionar esse problema de transações que estão incertas, da seguinte maneira:

1. Se o servidor que foi reinicializado não é o coordenador da aplicação, ele envia uma mensagem ao coordenador da aplicação para que descubra o resultado da transação;
2. Se o servidor que foi reinicializado for o coordenador da aplicação, ele envia uma mensagem aos outros coordenadores, informando que ele ainda está esperando pelo reconhecimento da resposta de *Commit*.

## 7.6 Recuperação de Desastre

Recuperação de desastre é o termo usado para descrever as atividades que precisam ser realizadas para restabelecer a base de dados no caso de fogo, terremoto, vandalismo e outros. Um plano de recuperação de desastre, pode incluir uma ou mais estratégias:

- Um *site* pode ser usado no caso de uma emergência;
- Uma máquina diferente em que a base de dados possa ser recuperada;
- Armazenamento em discos, separados da base de dados e dos arquivos de log.

Quando houver necessidade de recuperar a base de dados inteira em outra máquina, deve existir uma cópia da base completa e uma cópia de todos os arquivos de log, arquivados para aquela base de dados.

Geralmente é utilizado um vetor de disco para que possa se prevenir contra danos causados nos discos. Um vetor de disco, consiste em uma coleção de unidades de disco que aparecem como uma unidade de disco única para um aplicativo.

Os vetores de disco são conhecidos como RAID (Redundant Array of Independent Disks). O termo específico, é utilizado para vetores de disco de hardware, mas também podem ser fornecidos pelo software no nível de sistema operacional ou de aplicativo.

O ponto de distinção entre os vetores de disco de hardware e de software, é como a CPU manipula os pedidos de I/O. Para os vetores de disco de hardware, a atividade de I/O é administrada pelos controladores de disco, enquanto que para os vetores de disco de software, isso é feito pelo sistema operacional ou por um aplicativo [DB2, 3].

Este capítulo teve como objetivo mostrar como o banco de dados *DB2* se comporta quando é necessário utilizar recursos para torná-lo novamente confiável. O *DB2* implementa alguns métodos utilizados na recuperação do banco, entre eles o protocolo 2PC, que é usado quando se trata de transações distribuídas.

Existe ainda o problema de transações que estão no estado de incerteza, isto é, que não foram nem efetivadas e nem abortadas, e como ocorreu uma falha não sabem dizer qual foi a última ordem enviada pelo coordenador. O *DB2* soluciona esse tipo de impasse usando a recuperação de Crash, que irá verificar qual é o coordenador que está liderando a transação e qual foi a última ordem passada por ele, desta forma, no instante em que a instância é reinicializada, as transações que estavam no estado de incerteza, já podem tomar uma decisão, ou efetivam a transação ou tornam ela inválida, isto é, abortam a transação.

## Capítulo 8

### Banco de Dados – *SQL Server*

#### 8. *SQL Server*

O banco de dados *SQL Server* pode ser definido como um Sistema Gerenciador de Bancos de Dados Relacionais (SGBDR), que funciona sob sistema operacional *Windows NT*. O *SQL Server*, tem as seguintes características em sua arquitetura:

1. **Multiprocessamento simétrico (SMP)** - O SMP permite ao *SQL Server*, aumentar seu desempenho pelo uso de processadores adicionais. Ele suporta até 16 processadores, com tudo isso acontecendo sem interação do usuário, aliviando também os administradores das complexidades de administrar processadores múltiplos;
2. **Portabilidade** - O *SQL Server*, pode rodar em diferentes sistemas operacionais e em diversas plataformas de hardware;
3. **Independência de rede** - O *SQL Server* suporta diferentes tipos de protocolos de rede, tais como: TCP/IP, IPX/SPX, etc;
4. **Confiabilidade** - O *SQL Server* provê que se possa administrar o banco de dados a distância. Com essa característica o *SQL Server* pode ser utilizado sem interrupções [*SQL Server*, 4].

Este capítulo irá apresentar o banco de dados *SQL Server* no que se refere a recuperação dos dados quando ocorre falhas no sistema. Inicialmente será apresentado a arquitetura do banco de dados *SQL Server*, e logo em seguida será exposto como o *SQL Server* lida com transações distribuídas.

Em um segundo momento será tratado os tipos de backup que são implementados por esse banco, e, finalmente, os tipos de recuperação que poderemos ter quando houver necessidade de recompor o banco de dados para um estado consistente.

## 8.1 Bancos de Dados (*databases*)

O *SQL Server* trabalha com dois tipos de bancos de dados: os bancos de dados que são utilizados pelo *SQL Server* para operar e gerenciar o sistema e os bancos de dados do usuário, que é utilizado para armazenar os seus próprios dados.

Uma vez instalado o *SQL Server*, são criadas automaticamente quatro *databases*:

- a) *master*
- b) *model*
- c) *tempdb*
- d) *msdb*

Embora ambos os tipos de bancos de dados (sistema e usuário) armazenem dados, o *SQL Server* utiliza os bancos de sistema para operar e gerenciar o sistema. O catálogo de sistema, por exemplo, consiste unicamente de tabelas armazenadas no banco de dados *master*, como é demonstrado na figura 8.1 a seguir.

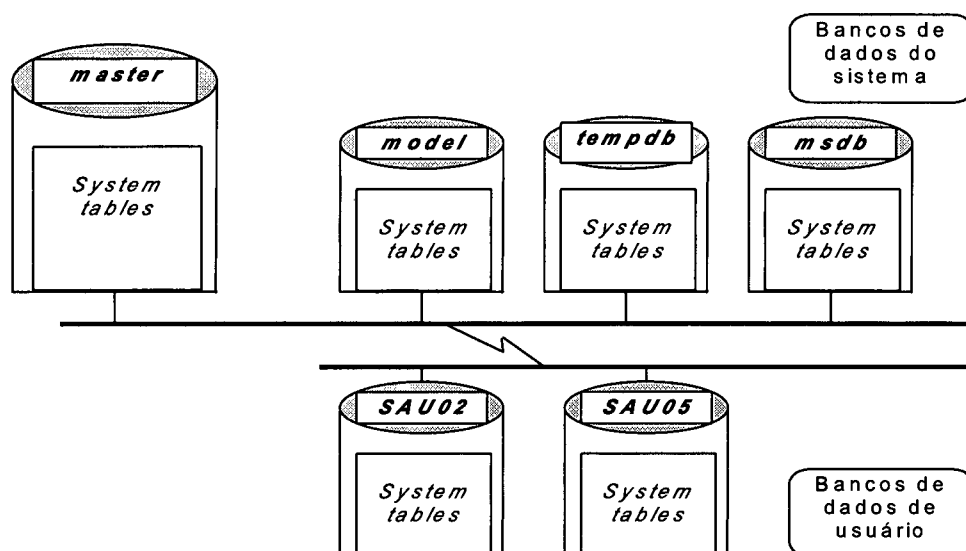


Figura 8.1. Banco de Dados *SQL Server* – [*SQL Server*, 3].

### 8.1.1 O banco de dados *Master*

Controla os bancos de dados de usuários e a operação do *SQL Server*, por isso os dados armazenados em suas tabelas são críticos e deve-se sempre ter *backup* atualizado, mantendo:

- a) contas de *login*;
- b) processos em andamento;
- c) mensagens de erro do sistema;
- d) *databases* armazenados no servidor;
- e) espaço alocado a cada *database*;
- f) *locks* ativos;
- g) *databases* disponíveis e dispositivos de *dump*;
- h) procedimentos de sistema, que são primariamente utilizados para administração.

O banco de dados *master*, contém 13 tabelas de uso compartilhado com o sistema, conhecido como Catálogo do Sistema ou Dicionário de Dados, [Gunderloy, 2001].

### 8.1.2 O banco de dados *Model*

Fornecer um protótipo (*template*) para um novo banco de dados. Contém as tabelas de sistema que serão inseridas em cada banco de dados de usuário. As seguintes implementações podem ser realizadas neste *database*.

- a) tipos definidos pelo usuário (*user datatypes*), regras (*rules*), padrões (*defaults*), *stored procedures*;
- b) usuários que terão acesso a todos os bancos adicionados ao sistema (administradores);
- c) privilégios padrões, notadamente aos usuários *guest* (*guest accounts*);

Sendo que este banco de dados possui um conjunto de 18 tabelas é conhecido como Catálogo do Banco de Dados.

### 8.1.3 O banco de dados *Tempdb*

Providencia um espaço de armazenamento para tabelas e outras ações temporárias ou intermediárias, tais como resultados que envolvam a cláusula *GROUP BY*, *ORDER BY*, *DISTINCT* e cursores (*CURSORS*). Possui as seguintes características:

- a) criado automaticamente no *DEVICE MASTER*;
- b) seu conteúdo é apagado quando o usuário fecha a conexão, exceto para tabelas temporárias globais;



- c) quando o banco é parado (*stopped*) seu conteúdo é apagado completamente;
- d) seu tamanho padrão é de 2 *Mbytes*.
- e) pode ser colocado em memória *RAM*.

#### 8.1.4 O banco de dados *MSDB*

Providencia suporte ao serviço *SQL Executive Service* (o qual fornece serviços de *schedulle* de tarefas, replicação, gerenciamento de alertas). Possuem as seguintes tabelas de sistema:

- a) *sysalerts* - armazena informações sobre todos os alertas definidos por usuários;
- b) *sysoperators* - informações sobre os operadores;
- c) *sysnotifications* - relaciona quais operadores devem receber quais alertas;
- d) *systasks* - mantém informações sobre todas as tarefas definidas por usuários;
- e) *syshistory* - informações a respeito de quando um alerta e uma tarefa foram executados, se com sucesso ou com falha, identificação do operador, data e hora da execução;
- f) *sysservermessages* - mensagens sobre as operações relacionadas ao servidor [*SQL Server*, 3].

## 8.2 Estrutura do Banco de Dados

O banco do *SQL Server* é constituído de dois ou mais arquivos físicos de sistema operacional, a saber:

- arquivos primários: é utilizado para armazenar todos os objetos do banco de dados, como as tabelas e índices, sendo utilizado também, para apontar para o resto dos arquivos que constituem o banco de dados;
- arquivo secundário: esse arquivo só existe quando o arquivo primário não é suficiente para armazenar todos os dados, podendo ser criada a medida que houver necessidade [Vieira, 2000];

- arquivo de log: esse arquivo é utilizado, para registrar todas as informações, antes das mesmas serem armazenadas nos arquivos primários ou secundários. Sendo utilizado para ajudar na recuperação caso o banco de dados encontrar-se no estado de inconsistência, podendo existir vários arquivos de log caso o arquivo de log original ficar sem espaço.

### 8.3 Transações Distribuídas utilizando o 2PC

Transações Distribuídas, são consideradas transações que modificam os dados armazenados em mais de um banco de dados. As mesmas instruções que são utilizadas para gerenciar transações locais, também são usadas também em transações distribuídas.

Quando é emitido um comando de *Commit Transaction* em uma transação distribuída, o *SQL Server* dispara automaticamente o protocolo 2PC. Na primeira fase, o *SQL Server* pede para que todos os *sites* participantes da transação se preparem. Os *sites* verificam podem efetivar a transação e reservam os recursos necessários para que isso aconteça. Após todos os *sites* enviarem a resposta ao *SQL Server* informando que estão prontos para efetivar a transação, é que a segunda fase se inicia.

Nessa fase, o *SQL Server* solicita aos bancos de dados envolvidos, para efetuar o *Commit* da transação, porém se algum *site* não puder efetuar a transação, é enviada uma mensagem de *Abort*, isto é, para desfazer a transação em todos os *sites* envolvidos.

As transações distribuídas são gerenciadas por um componente do *SQL Server* chamado de *Distributed Transaction Coordinator – DTC* (coordenador de transações distribuídas).

As transações locais normalmente são expandidas para transações distribuídas automaticamente quando é executada uma transação que ativa um procedimento armazenado remoto [Gunderloy, 2001].

### 8.4 Arquivos de log de Transação

Os arquivos de log de transação são associados aos bancos de dados individuais, por isso é necessário então, que o banco de dados seja configurado de tal forma, que os arquivos de log cresçam automaticamente a medida que ocorra o aumento das informações, para que sejam armazenadas todas as informações referentes às transações.

Durante as transações, os arquivos de dados do banco de dados, são lidos e gravados juntamente com os logs de transação que são gravados em paralelo. Se ocorrer uma falha de mídia, é necessário ter os arquivos de dados e os arquivos de log para que o banco possa ser recuperado.

Os arquivos de log, precisam ser armazenados em discos diferentes (separado fisicamente do disco que contém os dados), isto é, normalmente os arquivos são lidos apenas para backups ou então para realizar um *Rollback* de uma transação, mas não são constantemente gravados. [*SQL Server*, 1].

## 8.5 Backup dos Dados

Um backup pode ser definido como uma cópia de segurança dos dados que estão armazenados em algum outro lugar que não seja o disco rígido do mesmo computador. Geralmente é um disco rígido de alguma outra máquina que está conectada por meio de uma rede local.

Os backups que são realizados pelo *SQL Server*, podem ser backups off-line ou on-line, ou seja, os usuários do banco podem continuar acessando os dados enquanto o backup está sendo realizado, e, isto é feito através dos registros de transações.

### 8.5.1 Tipos de Backup

O *SQL Server*, estabelece pontos de controle nos bancos de dados, para copiar as transações efetivadas do registro de transações para o banco de dados. Esse registro é semelhante a um diário onde é informada a data ao lado do que está acontecendo. No registro de transações, também é armazenado um número de seqüência de registro (LSN) ao lado de cada linha do registro.

Quando o backup é iniciado, o *SQL Server* grava o LSN atual. Uma vez terminado o Backup, o *SQL Server* copia todas as entradas do registro de transações desde o LSN gravado no início até o LSN atual. Após isso o *SQL Server* copia todas as páginas do banco de dados que contêm realmente dados, e finalmente o SQL reúne todas as partes do registro de transações, que foram gravadas durante o processo de backup, ou seja, todas as linhas do registro de transações com LSN é maior que o LSN

gravado no início da sessão de backup, assim, os usuários podem continuar trabalhando normalmente com o banco de dados enquanto ele está sendo copiado [Vieira, 2000].

O *SQL Server* trabalha com os tipos de backup descritos a seguir.

#### **8.5.1.1 Full Backup (Backup Completo)**

Esse tipo de backup, faz cópia de todas as tabelas que formam o banco de dados, sendo que essa cópia pode ser realizada em uma fita ou em um disco local ou remoto.

O backup completo contém:

- uma estrutura de esquema e arquivo;
- dados;
- copia o LSN gravado no início do backup até o LSN final do backup;
- uma parte do log de transação, que contém todas as atividades do banco de dados desde o início do processo de backup.
- Os arquivos de índices, podem ser criados em um grupo de arquivos e a cópia é feita do grupo de arquivos, facilitando a recuperação, pois, para recriar um índice, o *SQL Server* exige que todos os arquivos de tabela básica e de índice, estejam na mesma condição em que estavam quando o índice foi criado inicialmente [Vieira, 2000].

#### **8.5.1.2 Differential Backup (Backup Diferencial)**

Esse tipo de backup, realiza cópias somente dos dados que foram alterados desde o último *Full Backup*, capturando atividades do banco de dados que ocorrem durante o processo de backup, possibilitando dessa forma, que o backup seja realizado enquanto os usuários continuam trabalhando no sistema [*SQL Server*, 2].

Os backups diferenciais, permitem que o banco de dados seja restaurado apenas até o momento em que o backup diferencial foi criado, e não até o ponto exato da falha. Uma forma de evitar esse tipo de problemas, é criar vários backups de log de transação após cada backup diferencial ter sido realizado [Gunderloy, 2001].

### 8.5.1.3 Log Backup

Esse tipo de backup faz cópia dos arquivos de log, que foram alterados desde o último backup completo. Trunca o log de transação até o início da sua parte ativa (a partir do ponto da transação aberta mais antiga até o final do log) e descarta a parte inativa.

Antes de realizar a recuperação do banco de dados utilizando o backup de log, é necessário realizar um backup do banco de dados. O ideal seria realizar uma combinação do backup diferencial e do log de transação [*SQL Server*, 3].

### 8.5.1.4 Backup Snapshot Server-less

São backups considerados funcionalmente equivalentes a backups de banco de dados completos ou backups de arquivos/grupos de arquivos. Esse tipo de backup, requer o uso de uma aplicação VDI (*virtual device interface*), que pode se comunicar diretamente com um avançado sistema de armazenamento corporativo.

A aplicação de backup, pode tirar do ar um dos espelhos e ainda realizar o backup daquele espelho para uma fita ou então torná-lo disponível em outro sistema. Nesse caso, o backup efetivamente se torna um banco de dados stand by imediatamente disponível. Esse tipo de backup, pode ser restabelecido muito rapidamente utilizando os backups de log e backups diferenciais [*SQL Server*, 3].

### 8.5.1.5 Failover Clustering

O *SQL Server* 2000, permite o failover e failback para ou a partir de um nó em um cluster. Uma instância de *SQL Server*, é executada em uma máquina primária, enquanto que uma instância secundária em uma segunda máquina é mantida disponível até o failover.

Em uma configuração “Ativa-Ativa”, o *SQL Server* roda múltiplos servidores simultaneamente em bancos de dados diferentes, permitindo que as organizações possam habilitar o failover para ou a partir de qualquer nó. Sendo que o *SQL Server* suporta clusters de failover até de quatro nós. Quando um do nós falha, o *SQL Server*

usa os recursos do sistema operacional mudando o failover para qualquer outro nó sobrevivente [*SQL Server*,3].

#### **8.5.1.6 Backups de arquivo/grupos de arquivos**

São cópias auxiliares de um ou mais arquivos de uma base de dados específica, que permite que a base de dados seja recuperada em unidades pequenas: em arquivos ou em nível de grupos de arquivos. Dessa forma, um banco de dados não fica limitado a um único disco rígido. Usando um grupo de arquivos, a cópia pode ser feita somente daqueles arquivos específicos ao invés de fazer a cópia de todo o banco de dados.

#### **8.5.1.7 Backup Lógicos**

Esses tipos de backups, são utilizados para extrair do banco apenas dados lógicos, como uma tabela específica, ou uma linha de alguma tabela específica que tenha sofrido modificações.

Quando é executada a recuperação do banco, usando esse tipo de backup, os dados são apresentados em pequenas partes, uma tabela ou uma linha de uma tabela, normalmente é usada quando acontece erros por parte do usuário [*SQL Server*, 1].

### **8.6 Tipos de Recuperação**

O *SQL Server*, marca cada transação no arquivo de log com um LSN e com a hora da ocorrência. Combinada com isso, a cláusula STOPAT da instrução de restauração pode retroceder os dados até um determinado momento.

O *SQL Server*, executa a recuperação automática toda vez que o banco de dados é inicializado. Primeiro é feita a conferência do banco de dados mestre e após isso, a checagem das outras bases do sistema. Para cada base de dados, o mecanismo de recuperação automática confere o log de transação, e, se houver qualquer transação que não foi efetuada *Commit*, automaticamente é feito o *Rollback* dessas transações.

O mecanismo de recuperação, confere o arquivo de log para as transações que efetuaram *Commit*, e, se elas ainda não foram gravadas no banco de dados, serão executadas novamente.

Os tipos de recuperação podem ser definidos em três: o *Full*, *Bulk\_logged* ou o *Simple*. A velocidade de recuperação, o tamanho dos arquivos de log e o grau de confiabilidade dos dados, depende do tipo de recuperação que é escolhido.

### 8.6.1 Recuperação *Full*

Esse tipo de recuperação, é o que mais traz segurança no caso de um arquivo de dados danificado. Quando o banco de dados está nesse modo, todas as operações são anotadas no arquivo de log de transações, e se o backup é realizado no modo *Full backup*, quando ocorrer uma falha de mídia e o banco necessitar ser recuperado por completo, a recuperação *Full* pode restabelecer o banco até a última transação que foi realizada antes da falha.

O *SQL Server*, utiliza junto com os arquivos de log, o *checkpoint* que permite colocar pontos de referência no log de uma transação, permitindo que o banco seja recuperado até o último checkpoint encontrado antes da falha [*SQL Server*, 1].

### 8.6.2 Recuperação *Bulk\_Logged*

Utilizando essa forma de recuperação, o banco de dados pode ser restabelecido por completo caso ocorra uma falha de mídia, utilizando o melhor desempenho e menos espaço de armazenamento dos arquivos de log. A recuperação *Bulk\_Logged*, utiliza o mesmo tipo de armazenamento da recuperação Full, com a vantagem: que as operações são anotadas de forma mínima, isto é, o servidor do SQL anota somente que a operação aconteceu.

A recuperação é possível, porque o servidor mantém informações sobre o que foi modificado pelo tamanho da operação. O servidor SQL, mantém um local (página) adicional chamado de BCM. Se estiver armazenado nessa página o número 1, é porque ocorreu uma mudança nessa tabela desde o último *Full Backup* que foi realizado.

O tempo necessário para efetuar uma recuperação, usando o backup do tipo *Bulk\_logged* é semelhante ao tempo que leva para recuperar uma base no modo completo, e as operações não precisam ser executadas novamente, porque todas as informações necessárias para restaurar os dados, estão disponíveis nos backups dos arquivos de log [*SQL Server*, 3].

### 8.6.3 Recuperação Simples

É um modelo que possui uma estratégia mais simples de restabelecer uma base de dados. O arquivo de log de transação, é copiado em intervalos regulares e freqüentes, por isso, somente é permitido o backup do tipo completo e Backup Diferencial [*SQL Server*, 2].

Este capítulo apresentou algumas características principais referentes ao banco de dados *SQL Server*, como a sua arquitetura, os principais processos que são utilizados quando o banco de dados precisa efetuar a recuperação da base independente do tipo de falha que aconteça.

Deu-se um destaque maior aos tipos de backup que são implementados pelo banco, pois, a recuperação depende totalmente do tipo de backup que foi utilizado.

O *SQL Server* possibilita que o banco de dados seja restaurado na sua totalidade, usando para isso um backup *Full*, ou que sejam restaurados apenas algumas tabelas, ou então apenas alguns arquivos que foram danificados, isto é, que se encontram no estado de inconsistência. Possibilitando com isso uma agilidade maior em tornar o banco de dados disponível para o usuário novamente.



## Capítulo 9

### Comparação entre os Bancos de Dados

#### 9. Banco de Dados

Um Banco de Dados Distribuído, pode ser considerado como um conjunto de dados que estão espalhados por diversos *sites*, sendo que esses *sites* estão todos interligados através de uma rede de comunicação.

Podemos ter com isso ainda transações que podem em um momento serem consideradas locais e em um momento seguinte uma transação global. Transação pode ser definida como uma coleção de ações que fazem com que os estados consistentes de um banco de dados, sofram alterações para novos estados consistentes. No início de uma transação, o banco de dados está em um estado consistente, durante a execução da transação, o banco de dados pode transitar por estados de inconsistência, porém, ao finalizar a transação, o banco de dados estará novamente em um estado consistente.

Sempre que ocorrer uma falha ou uma pane, o banco de dados precisa estar preparado para se recuperar, e solucionar esse tipo de problema, para isso, são utilizados diversos métodos de backup e recuperação.

Neste capítulo, será realizada uma comparação entre os bancos de dados descritos anteriormente, levando-se em conta alguns fatores que tem maior relevância quando se trata de Confiabilidade dos dados em um sistema distribuído, e também os métodos mais utilizados em restauração do banco de dados para um estado confiável que existia antes da ocorrência de falhas.

#### 9.1 Tipos de Falhas

Vários tipos de falhas podem ocorrer em um sistema, e cada uma exige um tratamento diferente.

A seguir, serão descritos os tipos de falhas mais comuns e como os bancos de dados se comportam com relação a elas:

## 1. Erro de usuário

- *Oracle* – através do processo PMON, o *Oracle* desfaz as alterações que esse processo tenha gerado;
- *DB2* – o banco de dados é restaurado, utilizando o comando de *Restore*,
- *Ingres* – o banco de dados é restaurado através do processo de recuperação *dmfrcp*,
- *SQL Server* - é utilizado o backup lógico para restaurar esse tipo de falha.

## 2. Falha de instância

- *Oracle* – o processo SMON de uma outra instância, executa a recuperação da instância afetada, assim que a mesma é reiniciada;
- *DB2* – as transações que não foram efetivadas quando ocorreu erro, são refeitas, utilizando os algoritmos de *Redo* e *Undo*, desta forma, as transações que foram efetivadas, serão refeitas e as transações que não foram efetivadas serão desfeitas;
- *Ingres* – é utilizado um processo do *Ingres* chamado de *Rollforward*, que permite que a base de dados seja recuperada até um certo ponto, antes de ocorrer uma falha;
- *SQL Server* - é utilizado o backup físico pra resolver esse tipo de falha.

## 3. Falhas de mídia ou de disco

- *Oracle* – o *Oracle* utiliza as cópias de segurança (backup), para restabelecer a base do banco de dados, junto com o backup dos arquivos de log;
- *DB2* – o *DB2* utiliza o backup completo que foi feito anteriormente para restabelecer o banco de dados, sendo que a base deve estar em off-line, isto é, os usuários não podem estar acessando a base;
- *Ingres* – o *Ingres* utiliza o backup (parcial ou *Full*), para realizar o restabelecimento do banco de dados;
- *SQL Server* – é utilizado o backup completo ou diferencial, para restaurar o banco de dados.

## 4. Falhas de Comunicação

- *Oracle* – o processo PMON, descobre qual foi o processo abortado e soluciona o problema. Se esse tipo de falha interromper o processo 2PC, o

processo RECO de cada base de dados local, soluciona automaticamente qualquer transação que ainda não tenha sido resolvida em todos os *sites* envolvidos;

- *DB2* – é utilizado o processo de recuperação chamado de *dmfrecp*, que permite restaurar o sistema ou o servidor, junto com o protocolo 2PC;
- *Ingres* – é utilizado o método de recuperação chamado de recuperação de versão, o qual permite que sejam utilizados os comandos de backup e restore;
- *SQL Server* – é utilizado o backup completo ou diferencial para restaurar o banco de dados

## 9.2 Tipos de Backup

Um backup pode ser definido como uma cópia de segurança dos dados que estão armazenados em algum outro lugar que não seja o disco rígido do mesmo computador. Normalmente é um disco rígido de alguma outra máquina, que está conectada por meio de uma rede local.

No caso de ocorrer alguma pane com o banco de dados, é necessário utilizar o backup para restaurar a base do banco, ou seja, deixar o banco novamente em um estado consistente, isto é, no mesmo estado que estava antes de ocorrer a falha.

A tabela 9.1 a seguir, apresenta os tipos de backups que existem e que são mais utilizados pelos bancos de dados *Oracle*, *DB2*, *Ingres* e *SQL Server*.

<b>Tipos de Backup</b>	<i>Oracle</i>	<i>DB2</i>	<i>Ingres</i>	<i>SQL Server</i>
Backup Lógico	X			X
Backup <i>Full</i> (Completo)	X	X	X	X
Backup Differential ou Incremental (Parcial)	X	X	X	X
Backup dos arquivos de Log	X	X	X	X
Backup <i>Snapshot</i>				X
Backup <i>Failover</i> ou <i>Hot Standby</i>	X			X
Backup de arquivo e grupo de arquivos				X
Backup de Sistema Operacional	X		X	
<i>Unloaddb</i>			X	
Backup realizado pelo RMAN	X			

Tabela 9.1 Tipos de Backup

O backup lógico é um backup incrementado pelo *Oracle* e pelo *SQL Server*. É utilizado para fazer cópias dos elementos dos dados, excluindo a parte física dos dados. Normalmente esse backup é usado como exportação e importação dos dados, através do recurso de Export no *Oracle* e do DTS no *SQL Server*.

O backup *Full* é um backup incrementado por todos os bancos de dados, pois se trata de um backup completo de toda a base de dados, que normalmente é feito quando o banco está no modo fechado, isto é, todos os usuários estão fora do sistema.

Backup *Differential* ou Incremental, também chamado por alguns bancos como parcial, é o tipo de backup que somente realiza cópias dos dados que foram alterados desde o último backup *Full* realizado. É um tipo de backup que pode ser feito enquanto o banco está no modo aberto, isto é, os usuários continuam trabalhando no sistema durante a realização do mesmo.

Backup dos arquivos de log, é um backup incrementado por todos os bancos de dados, pois é utilizado sempre que houver necessidade de recuperar o banco de dados. Tem-se os arquivos de log on-line que são criados durante o andamento das transações, e a medida em que o banco foi configurado, são armazenados no disco através do uso do *Archived* log.

O backup *Snapshot*, é usado pelo *SQL Server*, e a funcionalidade dele é equivalente aos backups completos que são realizados pelos outros bancos de dados. Esse backup, é usado para a recuperação do banco junto com os backups de log e os backups diferenciais.

Backup *Failover* ou *Hot Standby*, é a capacidade que o banco tem de alternar para um banco de dados alternativo, quando a instância primária se torna indisponível. Quando ocorre da instância primária falhar, o banco de dados *Standby* precisa de um tempo para se recompor e ficar disponível para que os usuários possam voltar a usar o banco de dados. Desta forma, um banco de dados *Standby*, sempre está no modo de recuperação através dos registros dos arquivos de *Redo*.

Backup de arquivo/grupo de arquivos é incrementado pelo *SQL Server*, e é usado para realizar cópias auxiliares de um ou mais arquivos de uma base de dados específica, permitindo desta forma, que a base de dados seja recuperada em pequenas unidades, isto é, em arquivos ou grupos de arquivos.

O backup de Sistema Operacional, é incrementado pelo *Oracle* e pelo *Ingres*. No banco de dados *Ingres*, o backup usando o sistema operacional, é um método utilizado para armazenar a estrutura do *filesystem* em algum dispositivo local remoto. Já o *Oracle* trabalha com alguns backups a nível de sistema operacional, isto é, são backups realizados através de comandos executados no próprio ambiente do sistema operacional, comandos esses, executados pelo DBA do banco de dados.

Backup utilizando *Unloaddb*, é um método utilizado pelo *Ingres*, o qual possibilita que sejam executadas cópias da base de dados para arquivos em um determinado diretório e que mais tarde, se houver necessidade, são utilizadas pelo DBA para executar a recuperação da base.

Quando se utiliza o gerenciador de recuperação do *Oracle*, é possível que o RMAN gere as operações de backup e de recuperação. O RMAN pode ser utilizado para efetuar cópias de segurança de todo o banco ou de alguns componentes do banco, porém o RMAN, não faz backup dos arquivos de inicialização. Desta forma é necessário que sejam feitas cópias através do S.O.

### **9.3 Recuperação do Banco de Dados**

Quando ocorre de algumas transações serem inesperadamente interrompidas, a base de dados é encerrada com erro e conseqüentemente ficará no estado de inconsistência. Com isso, o banco de dados ao ser reinicializado, faz automaticamente a recuperação da base de dados.

Quando a base de dados encontrar-se em um estado consistente e utilizável, significa que ela atingiu o que é conhecido como “ponto de consistência”, isto é, todas as transações que estavam inconsistentes foram resolvidas e agora os dados estão disponíveis para que possam ser acessados pelos usuários ou pelos programas aplicativos.

A tabela 9.2 a seguir, mostra os principais métodos de recuperação utilizados pelos bancos de dados.

<b>Tipos de Recuperação</b>	<i>Oracle</i>	<i>DB2</i>	<i>Ingres</i>	<i>SQL Server</i>
Recuperação de Versão		X		
Recuperação Roll-Forward	X	X	X	
Recuperação utilizando o 2PC	X	X	X	X
Recuperação de Desastre		X		
Recuperação Full	X	X	X	X
Recuperação Bulk-Logged				X
Recuperação Simples				X
Processo de Recuperação – <i>Dmfrcp</i>			X	
Recuperação usando Checkpoint	X	X	X	X
Recuperação usando arquivos de Log	X	X	X	X

Tabela 9.2. Métodos de Recuperação.

O método de recuperação de versão é implementado pelo *DB2*, e é utilizada uma versão prévia da base de dados, que foi criada durante uma operação de cópia de segurança. Quando é executada a recuperação da base de dados utilizando esse método de recuperação, é usado o comando de Backup junto com o comando de *Restore* e com os arquivos de log que estão armazenados.

O método de recuperação *Roll-Forward* é implementado pelos bancos *Oracle*, *DB2* e *Ingres*. Em particular, o *DB2* usa esse método, junto com os comandos de backup, *Restore* e comando *Roll-Forward*, que tem por objetivo refazer as transações que efetuaram *Commit*, utilizando para isso, os arquivos de log que estão armazenados em disco; já no *Oracle*, o *Roll-forward*, utiliza os arquivos de *Redo Log Online* e se houver necessidade os arquivos de *Redo Log Archived*.

O protocolo de recuperação 2PC é implementado por todos os bancos de dados, sendo que na primeira fase, todos os *sites* votam para efetuar o *Commit* das transações e na segunda fase todos os *sites* efetuam propriamente a transação. Maiores detalhes nos capítulos referentes a cada banco de dados.

O método de recuperação de desastre é implementado pelo banco de dados *DB2*, e é usado quando ocorre uma falha (fogo, terremoto, vandalismo e outros) na base de dados. Desta forma, quando houver necessidade de restaurar o banco, é imprescindível ter a base de dados em outra máquina diferente da que sofreu a pane, ou então ter uma cópia dos dados e dos arquivos de log armazenados em discos separados da base atual.

A recuperação *Full*, é um método utilizado por todos os bancos de dados, e é realizada através da restauração do último backup *Full* juntamente com os arquivos de log que estão armazenados em disco, ou seja, os logs arquivados.

A recuperação *Bulk-Logged*, é um método implementado pelo banco de dados *SQL Server*. Esse método, utiliza o mesmo tipo de armazenamento da recuperação *Full*, com a vantagem que somente as operações que foram modificadas são anotadas. Quando há necessidade de usar esse tipo de recuperação, o *SQL Server* pesquisa em uma página chamada de BCM se consta armazenado o número 1, se esse número for encontrado, é porque ocorreu uma mudança nessa tabela desde o último backup *Full* realizado.

O método de recuperação simples é implementado pelo *SQL Server* e somente pode ser utilizado junto com o backup do tipo completo ou diferencial, e é estabelecido um intervalo regular e freqüente onde os arquivos de log são copiados para o disco.

O processo de recuperação *Dmfrcp*, é um comando utilizado pelo *Ingres*, que permite o acesso e a restauração de sistemas ou de servidores que tenham sofrido algum tipo de falha.

O método de recuperação utilizando *checkpoint*, é implementado por todos os bancos de dados, e é utilizado sempre que é feita a recuperação usando arquivos de log que estão armazenados em disco. Em particular, o banco de dados *Ingres* utiliza esse método, para executar a recuperação da base enquanto o banco está aberto, isto é, enquanto os usuários estão utilizando o banco.

A recuperação usando arquivos de log é implementada por todos os bancos, pois sempre que é feita uma recuperação do banco, existe a necessidade de restaurar o mesmo sempre para o ponto que ele se encontrava antes da ocorrência da falha. Para maiores detalhes, em cada capítulo específico dos bancos, foi descrito como cada um trata os arquivos de log e como eles são utilizados na recuperação dos dados.

Nesse capítulo foi apresentada uma comparação dos principais paradigmas envolvendo a confiabilidade em banco de dados distribuído. Os tópicos abordados foram os que tem maior relevância quando se trata de recuperação em banco de dados, que são os fatores que envolvem os tipos de falhas que podem ocorrer em um sistema, os tipos de backup que podem ser realizados pelo banco ou pelo administrador do banco de dados, e quais os métodos mais utilizados por cada banco, e aquilo que eles tem em comum, quando há necessidade de recuperar o banco para um estado confiável, isto é, tornar os dados novamente confiáveis após o sistema ter sofrido algum tipo de pane.

# Capítulo 10

## Conclusão

### 10.1 Resumo do trabalho

Um banco de dados que é armazenado em diversos computadores e, a comunicação entre esses computadores é realizada através de redes de comunicação, pode ser definido como um banco de dados distribuídos. Devido a sua arquitetura, a organização dos dados e a estrutura de como o banco é criado, ele está passível de sofrer alguns tipos de falhas.

As falhas que podem ocorrer em um banco podem ser tanto de hardware quanto de software, comprometendo com isso a integridade do Banco de Dados (BD). O SGBD deve incorporar mecanismos que garantam sua integridade, desta forma, o SGBD pode ser mantido em operação por longos períodos sendo que, pode ser interrompido por curtos intervalos para que os mecanismos de recuperação sanem as inconsistências geradas por pequenas falhas.

O sistema de recuperação em um banco de dados é responsável pela restauração do banco de dados para um estado consistente que havia antes de ocorrer falhas. Em outras palavras, o modo de garantir que o banco de dados de fato é recuperável é garantir que toda porção de informação que ele contém possa ser reconstruída a partir de outra informação armazenada de modo redundante em algum outro lugar do sistema.

Dependendo da pane que ocorrer no banco, é necessário usar algum método de recuperação que possa utilizar um dos backups que foram gerados anteriormente. Normalmente, em uma recuperação, é utilizado o backup *full* que foi realizado por último, juntamente com os registros de log que estão armazenados em algum dispositivo, e, os pontos de controle que estão armazenados nos arquivos de log. Desta forma, é possível restaurar o banco para um estado confiável, existente antes de ocorrer a falha.

Outra forma de executar a recuperação, ou de consertar o efeito de uma pane, é que algumas transações terão seus efeitos refeitos e os novos valores que elas gravaram no banco de dados serão gravados novamente, e, outras transações terão seus efeitos



desfeitos. Com isso, o banco de dados será restaurado, dando a impressão que parecerá que as transações nunca foram executadas.

Esse método é chamado de registro de log de desfazer, isto é, se ele não estiver absolutamente certo de que os efeitos de uma transação foram completados e o disco foi atualizado, qualquer alteração no banco de dados que a transação tenha feito, será desfeita e o banco de dados voltará a ficar no estado que se encontrava antes da transação.

O registro de log de refazer ignora as transações incompletas e refaz as mudanças das transações que foram efetivadas. O registro de log de refazer, exige que o registro de *Commit* seja gravado em disco antes que qualquer valor que foi alterado seja armazenado em disco. O registro de log de refazer, representa mudanças em elementos do banco de dados através de um registro que fornece um novo valor.

O protocolo de recuperação que é mais utilizado pelos bancos de dados em transações distribuídas, é o protocolo de *Commit Two-Phase* (2PC). É considerado um protocolo com duas fases: na primeira fase o coordenador geral da transação, envia uma mensagem pedindo, que todos os *sites* que estão envolvidos na transação se preparem para efetuar o *Commit* e fica esperando a resposta dos *sites*. É dado início então, à segunda fase do protocolo, e dependendo da resposta que os *sites* mandaram, é efetuado o *Commit* da transação ou então é executado um comando de abort que irá cancelar a transação. A regra geral do protocolo *Two-Phase Commit* (2PC) é que uma transação somente será efetivada se todos os *sites* participantes votarem por efetuar a transação, e ao contrario, uma transação poderá ser abortada se pelo menos um dos *sites* decidir por cancelar a transação.

Para evitar a possibilidade de paralisação quando ocorre falhas utilizando o protocolo 2PC. Foi acrescentando aos estados de *Wait* (e *Ready*) e *Commit*, onde o processo está pronto para *Commit* (se esta for a decisão final) mais um processo. São descritos os diagramas de transição estatais para o coordenador e o participante neste protocolo. Isto é chamado de Protocolo de *Commit Three-Phase* porque há três transições de estado, do estado *Initial* para um estado de *Commit*.

O protocolo 3PC segue a mesma regra geral do 2PC, porém com uma vantagem, o 3PC consegue impedir a paralisação porque foi acrescentado uma fase extra na qual é criado uma pre-decisão sobre o destino da transação. Sendo que essa pre-decisão é

passada aos *sites* participantes, então, mesmo que ocorra uma falha do coordenador a decisão pode ser tomada com base na pre-decisão.

Desta forma, este trabalho ficou dividido em três partes, na primeira é apresentado os tipos de falhas que podemos encontrar em um banco de dados, sendo este centralizado ou distribuído, e, em uma segunda parte, os métodos que podem ser utilizados para recompor o banco de dados.

Na terceira parte, foi apresentado os bancos de dados *Oracle*, *DB2*, *Ingres* e o *SQL Server*, com uma breve introdução sobre cada banco, os tipos de falhas que podem ocorrer em cada um deles, e, os métodos que cada um possui para se recompor caso ocorra uma falha, sendo que alguns deles possuem métodos particulares de trabalhar com o problema de falhas, mas geralmente todos eles implementam o protocolo 2PC (*Two-Phase Commit*) o qual garante que as informações geradas pelo banco, são informações corretas, isto é, confiáveis.

## 10.2 Conclusões

Durante o decorrer do trabalho, foram abordados alguns temas que vem conflitar diretamente com o estado de “confiabilidade” do banco: os tipos de falhas que podem ocorrer no sistema, independente destas originarem ou não a perda da informação; e os métodos de backups que são implementados pelos bancos de dados, de forma que, mais tarde possam ser utilizados pelo processo de recuperação.

Com base no levantamento das informações que foi realizado sobre banco de dados distribuído, mais específico, o assunto Confiabilidade dos dados em banco de dados distribuído, estudando as diversas formas de falhas que podem ocorrer, os tipos de backup que cada banco implementa e os métodos de recuperação que podemos utilizar, chegou-se à conclusão, que é necessário implementar e utilizar os diversos métodos que os bancos possuem para garantir que o mesmo possa sempre estar em um estado confiável.

Dependendo da transação que esta sendo realizada, pode-se utilizar o protocolo 2PC, para transações distribuídas, juntamente com alguma recuperação local: o método de Refazer (*Redo*), o método de Desfazer (*Undo*), ou o método de Desfazer/Refazer os efeitos de uma transação.

A respeito dos bancos de dados estudados, cada um possui métodos e formas particulares de executar a recuperação, porém, todos implementam os tipos de recuperação que são essenciais para tornar os dados novamente confiáveis ao usuário.

Particularmente, o banco de dados *Oracle* e o banco de dados *SQL Server*, possuem um método de backup, que se destaca dos outros bancos, pelo fato, que se ocorrer uma pane, em apenas alguns instantes o usuário estará usando novamente o banco. Esse método é conhecido como *Failover* ou *Hot Standby*, que é a capacidade que o banco tem de alternar para um banco de dados alternativo quando a instância primaria se torna indisponível. Quando ocorre da instância primaria falhar, o banco de dados *Standby* precisa de um tempo para se recompor e ficar disponível para que os usuários possam voltar a usar o banco de dados. Desta forma um banco de dados *Standby* sempre esta no modo de recuperação através dos registros dos arquivos de *Redo*.

### 10.3 Relevância do Trabalho

Somente a replicação de dados não é suficiente para garantir um banco de dados mais consistente. Dentro de um Sistema Gerenciador de Banco de Dados é necessária a implantação de diversos protocolos que exploram a distribuição e replicação dos dados, executando com isso, operações mais confiáveis.

Um SGBD distribuído confiável é um sistema que continua processando as requisições do usuário, mesmo que o sistema subjacente esteja no estado de incerto, isto é, mesmo que ocorram falhas, o SGBD continuará executando as transações sem violar a consistência do banco de dados [Casanova, 1985].

Seguindo esse principio, de que é necessário manter a confiabilidade do banco independente da pane que ocorrer com o mesmo, este trabalho vem para contribuir no esclarecimento dos métodos de recuperação que podem ser implementados para solucionar o problema de restauração do banco de dados. Utilizando os banco de dados *Oracle*, *SQL Server*, *DB2* e *Ingres*, podemos ter uma idéia mais concreta de como aplicar esses métodos, e, também a forma que esses bancos se comportam com relação aos mesmos, juntamente com a estrutura que é utilizada por cada um, para garantir a confiabilidade das suas informações.

#### **10.4 Perspectivas Futuras**

A necessidade de manter a confiabilidade do banco de dados, independente do tipo de pane que possa ocorrer, faz com que, cada vez mais, as empresas adquiram softwares que tenham competência em garantir, que os dados possam ser recuperados de uma tal forma, que é como se a pane não tivesse ocorrido. Para o desenvolvimento deste trabalho, foi realizado uma análise técnica de alguns softwares gerenciadores de bancos de dados com base no material encontrado sobre os mesmos. Para dar andamento nesse trabalho será desenvolvido um protótipo, onde será realizado algumas simulações, com o acompanhamento de como os bancos de dados irão se comportar, com relação as falhas, e, ao processo de recuperação implementado por cada banco.

## **Apêndice A**

### **Banco de Dados**

Banco de dados (BD) pode ser definido como um conjunto de dados. Um conjunto de dados associados a um conjunto de programas para acessar esses dados é chamado de Sistema Gerenciador de Banco de Dados (SGBD), sendo que o objetivo principal de um SGBD é proporcionar um ambiente que seja conveniente e eficiente para a recuperação e armazenamento das informações em um BD.

Normalmente um SGBD é criado para proporcionar o manuseio de um grande volume de informações, devendo garantir a segurança das informações que estão armazenadas, contra eventuais problemas que possam ocorrer com o banco de dados.

Neste capítulo será exposto uma introdução aos conceitos e definições de Banco de dados, e também serão apresentadas as arquiteturas de um sistema de banco de dados.

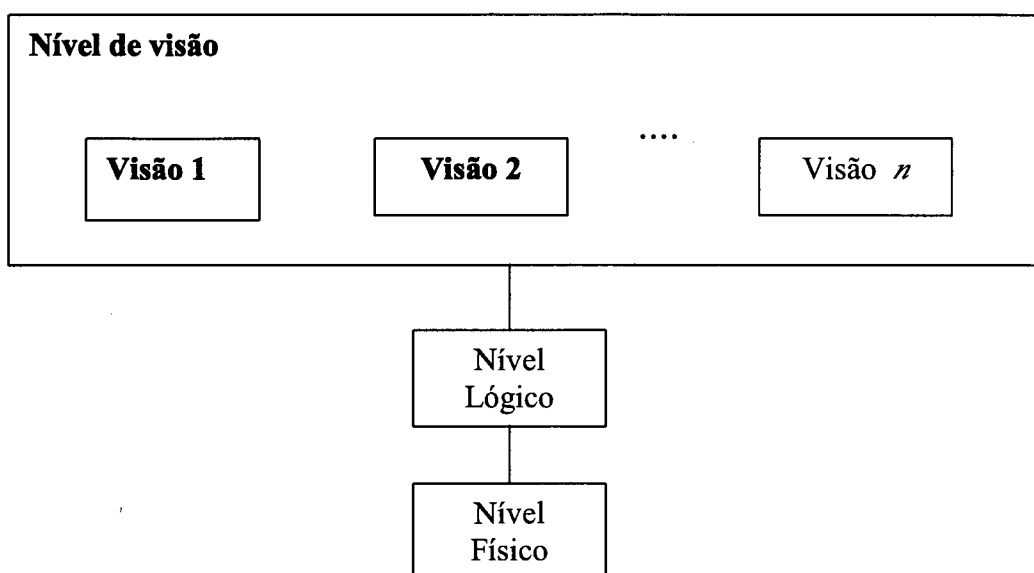
#### **1. Definições e Conceitos de Banco de Dados**

Um SGBD é uma coleção de arquivos e programas inter-relacionados que permitem ao usuário o acesso para consultas e alterações desses dados [Korth,1999]. Um SGBD permite que o usuário tenha uma visão abstrata dos dados, isto é, o sistema oculta detalhes como forma de armazenamento, recuperação e manutenção desses dados.

- **Nível físico:** esse nível descreve como os dados estão de fato armazenados. É considerado o nível mais baixo de abstração;
- **Nível lógico:** esse nível descreve quais dados estão armazenados no BD e quais os relacionamentos que existem entre eles. É considerado um nível de abstração médio e, é utilizado pelos administradores do BD;
- **Nível de visão:** é considerado o nível mais alto de abstração, porque descreve apenas uma parte do BD. Os usuários do banco de dados utilizam apenas

parte do banco, com isso, é criada uma visão de uma parte do BD, porém, o sistema pode proporcionar diversas visões do BD.

O relacionamento entre os níveis de abstração esta representado na Figura 1.1. [Korth,1999].



**Figura 1.1 Relacionamento dos Níveis de Abstração**

### 1.1 Instâncias e Esquemas

Um BD sofre alterações através de informações que são inseridas ou excluídas. O conjunto de informações que estão contidas no BD, em um determinado instante, são chamadas de instância do banco de dados.

O projeto geral do banco de dados é conhecido como Esquema, sendo que o esquema é pouco alterado. Os sistemas de BD possuem vários esquemas, considerando os níveis de abstração, temos o esquema físico (nível físico), o esquema lógico (nível lógico) e os sub-esquemas (nível de visão).

A capacidade de modificar a definição dos esquemas em um determinado nível, sem alterar o esquema de nível superior é chamada de independência dos dados. Existem dois níveis de independência dos dados:

- Independência de dados física: o esquema físico sofre modificações sem a necessidade de reescrever qualquer programa de aplicação;

- Independência de dados lógica: ocorrem modificações no esquema lógico, sem que seja necessário reescrever programas de aplicação. Essas modificações ocorrem sempre que uma estrutura lógica do BD é alterada.

## 1.2 Linguagens de Banco de Dados

Um sistema de banco de dados proporciona dois tipos de linguagens: uma específica para os esquemas do banco de dados e outra para realizar consultas e atualizações.

- Linguagem de Definição de Dados (DDL): em um esquema de dados é especificado um conjunto de definições, que são expressas por uma linguagem especial chamada de DDL. O resultado desse conjunto de definições é armazenado em um arquivo especial conhecido como *metadados*, e, em um sistema de banco de dados, esse arquivo é consultado antes que o dado real seja alterado.
- Linguagem de Manipulação dos Dados (DML): o objetivo principal é viabilizar o acesso ou a manipulação dos dados de forma compatível ao modelo de dados apropriado. Uma DML procedural exige que o usuário especifique *quais* dados são necessários e *a maneira de* obter esses dados. Em uma DML não procedural o usuário especifica *quais* dados são necessários, *sem* a necessidade de especificar a forma de obter esses dados.

## 1.3 Arquiteturas de Sistemas de Banco de Dados

A Arquitetura de um sistema de banco de dados é influenciada diretamente pelo sistema básico computacional, sobre qual o sistema de banco de dados está sendo executado.

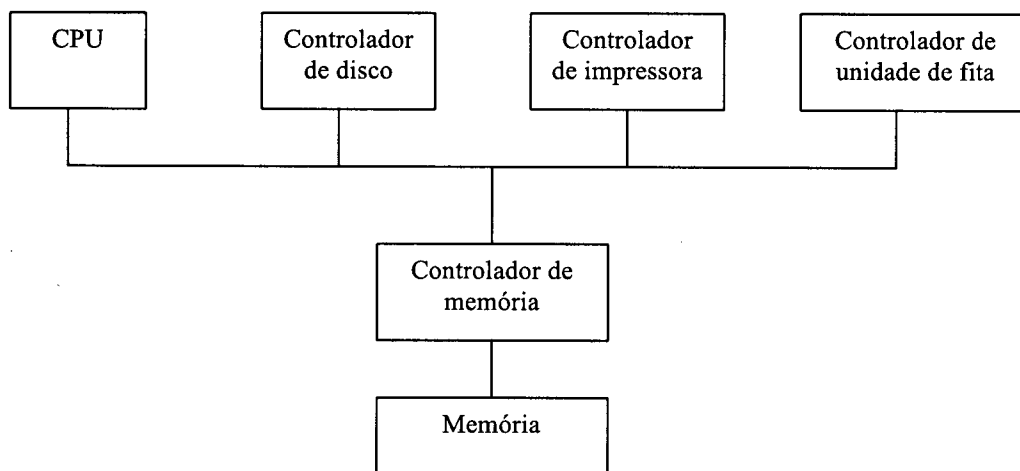
- Rede de Computadores: algumas tarefas são executadas no servidor do sistema e outras são executadas no cliente, com isso, criou-se o sistema de banco de dados cliente-servidor.
- Processamento paralelo: permite que atividades do sistema de banco de dados sejam realizadas com maior rapidez, reduzindo o tempo de respostas das transações e, com isso, aumentando o número de transações a serem

processadas por segundo. Surgiu então, o sistema de banco de dados paralelo.

- Distribuição de dados: diversas cópias do banco de dados são espalhadas pelos nós da rede, permitindo que os dados residam onde são gerados ou onde são mais utilizados. As grandes organizações podem manter operações em seus bancos de dados mesmo que ocorram falhas no sistema, como por exemplo, inundações, incêndios, problemas de *hardware* ou de *software*. Sistemas de banco de dados distribuídos, foram desenvolvidos para lidar com dados distribuídos geograficamente ou administrativamente, em diversos sistemas de banco de dados.

### 1.3.1 Sistemas Centralizados

Sistemas Centralizados são sistemas de banco de dados executados sobre um único sistema computacional e não interagem com outros sistemas. Tais sistemas podem ter um único usuário trabalhando em um computador pessoal até sistemas de alto desempenho em sistemas de grande porte.



**Figura 1.2 Um sistema computacional Centralizado**

Sistemas de banco de dados que são desenvolvidos para serem mono-usuário, não tem a preocupação com o controle de concorrência, pois existe somente um usuário gerando atualizações. A recuperação de dados é quase inexistente, pois antes de qualquer atualização é realizado um backup do banco de dados.

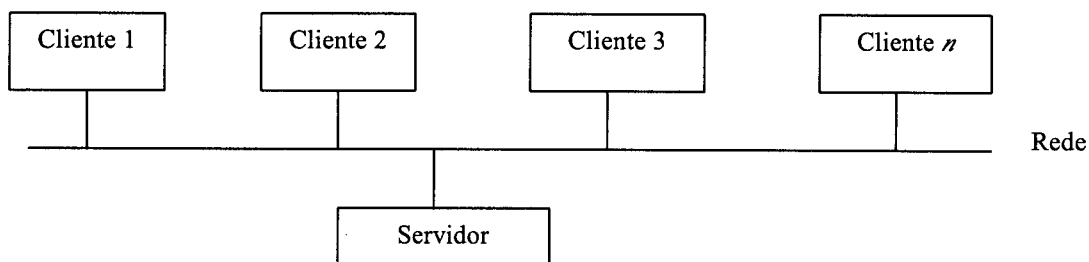


Os sistemas de computadores possuem vários processadores que compartilham a memória principal, mesmo assim, as consultas que são realizadas nesse tipo de banco de dados, não particionam essa mesma consulta entre os vários processadores, ao contrário, cada consulta é executada em um único processador, com isso, várias consultas podem ser executadas concorrentemente. Um grande número de transações são processadas por segundo, isso não quer dizer que uma transação individual possa ser executada com maior rapidez.

Bancos de dados que são processados em equipamentos que utilizam um único processador, já dispõem de recursos de multitarefa, dando a ilusão ao usuário de que diversos processos são executados em paralelo.

### 1.3.2 Sistema Cliente-Servidor

Os computadores pessoais estão sendo utilizados nos sistemas centralizados. Como resultado dessa nova tecnologia, os sistemas centralizados agem como *sistemas servidores* que atendem as solicitações de *sistemas clientes*.



**Figura 1.3 Estrutura geral de um Sistema Cliente-Servidor [Korth,1999].**

As funções de um banco de dados podem ser divididas em duas categorias: *front-end* e *back-end*. O *back-end* gerencia as estruturas de acesso, o desenvolvimento e otimização de consultas, o controle de concorrência e recuperação, enquanto que o *front-end* consiste nas ferramentas utilizadas por uma DML, como formulários, gerador de relatórios e recursos de interface gráfica.

Os sistemas servidores são caracterizados como:

- Sistemas Servidores de transações: disponibilizam uma interface onde os clientes enviam pedidos para executar uma determinada

ação, em resposta, o sistema servidor executa a ação e devolve o resultado para o cliente.

- Sistemas Servidores de dados: permite que os servidores interajam com os clientes que fazem solicitações de leitura e atualização de dados em unidades como arquivos ou páginas, proporcionando meios para indexação de dados e transações, de forma que os dados não se tornem inconsistentes se um equipamento cliente ou processo falhar.

### 1.3.2.1 Servidores de Transações

Em um sistema centralizado, o *front-end* e o *back-end* são executados dentro de um único sistema, ao contrário, a arquitetura de servidores de transação faz com que os computadores pessoais ajam como clientes de sistemas servidores, nos quais são armazenados grandes volumes de dados, dando suporte aos recursos de *back-end*. Os clientes enviam solicitações ao sistema servidor, onde as transações são executadas, e, o resultado obtido é enviado de volta ao cliente que tem como responsabilidade exibir esses dados.

Padrões utilizados do tipo ODBC (*open database connectivity*), são padrões de programas de aplicação de interface que possibilitam aos clientes, criarem comandos utilizando uma DML e enviá-los para o servidor, onde são executados. Um cliente qualquer, que trabalhe com uma interface ODBC, pode conectar-se a um servidor que forneça essa interface, sendo que nas primeiras gerações de sistemas de banco de dados, como não existia esse padrão, era utilizado o mesmo *software* para *back-end* e *front-end*.

### 1.3.2.2 Servidores de Dados

São utilizados em redes locais, onde não existe uma conexão de alta velocidade entre clientes e servidores, os equipamentos utilizados pelos clientes podem ser comparados aos equipamentos dos servidores, em nível de processamento. Em tal ambiente o processamento é feito localmente e os dados são enviados ao servidor. Este tipo de arquitetura é bastante utilizado em sistemas de banco de dados orientado a objeto.

Os dados que navegam para um cliente durante uma transação podem ser armazenados no equipamento local do cliente, mesmo depois de completada a transação. Se os dados forem particionados entre os clientes que solicitaram a transação, e um dos clientes raramente solicita um dado ao mesmo tempo que o outro, os bloqueios também podem ser armazenados localmente, isto é, no equipamento do cliente.

### **1.3.3 Sistemas Paralelos**

A utilização de sistemas paralelos impõem velocidade ao processamento e a entrada/saída de dados através do uso paralelo de diversas CPUs e discos. Ao contrário do processamento serial, no processamento paralelo muitas operações são realizadas simultaneamente. Esse tipo de arquitetura é utilizado em banco de dados muito volumosos, ou que processam um volume enorme de transações por unidade de tempo.

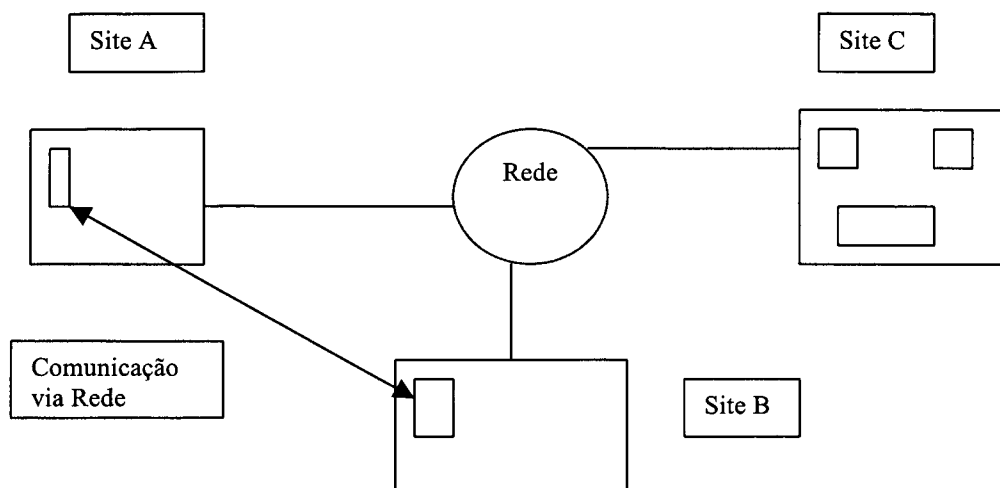
Duas formas são utilizadas para avaliar o desempenho de um sistema de banco de dados: o número de tarefas que podem ser realizadas em um dado intervalo de tempo e o tempo total que o sistema leva para completar uma única tarefa. Quando se utiliza a arquitetura de Sistemas Paralelos, é importante levar em consideração duas metas:

- **Aceleração:** refere-se a redução do tempo de execução de uma tarefa por meio do aumento do grau de paralelismo.
- **Escalabilidade:** refere-se ao manuseio de um maior número de tarefas por meio do aumento do grau de paralelismo.

### **1.3.4 Sistemas Distribuídos**

O banco de dados é armazenado em diversos computadores e, a comunicação entre esses computadores é realizada através de redes de comunicação de dados. Não compartilhando memória principal ou discos, os computadores de um sistema de banco de dados distribuído, pode variar quanto ao tamanho e funções, podendo ser estações de trabalho até sistemas de grande porte.

Os computadores em um sistema de banco de dados distribuído recebem várias denominações, como *nó* ou *site*. A estrutura geral de um sistema distribuído é mostrado na figura 1.4 a seguir:



**Figura 1.4. Um sistema Distribuído. [Korth,1999]**

A principal diferença que existe entre um banco de dados distribuído e um banco de dados paralelo, é que no banco de dados distribuído, os dados são distribuídos fisicamente, a administração dos dados é feita separadamente e a intercomunicação é menor, outra importante diferença é que existem transações *locais* e transações *globais*. Sendo que uma transação é dita *local* quando acessa um único *site*, justamente aquele na qual foi iniciada e a transação *global* ao contrário, faz acesso a vários *sites*, ou a um *site* além daquele acessado quando foi iniciada.

Existem diversas razões para a utilização de um banco de dados distribuído, entre essas razões, estão o compartilhamento de dados, a autonomia e a disponibilidade.

- **Compartilhamento de dados:** os usuários de um *site* podem ter acesso a dados que estão localizados em outros *sites*.
- **Autonomia:** cada *site* mantém um certo nível de controle sobre os dados que estão armazenados naquele *site*. Em sistema centralizado, o administrador geral é responsável pelo banco como um todo, já em um sistema distribuído, existe um administrador local para cada *site*, sendo essa a maior vantagem de se utilizar um banco de dados distribuído.
- **Disponibilidade:** na ocorrência de um *site* sair do ar, os demais *sites* poderão continuar com a operação, se os dados estiverem replicados em diversos *sites*. Uma transação que necessita de um determinado dado, poderá buscar esse item em algum outro *site*, com isso, a queda de um *site* não implicará

necessariamente na queda do sistema. A queda de um *site* precisa ser detectada pelo sistema, sendo que o sistema deve determinar ações que devem ser executadas, para que o *site* seja recuperado. A recuperação de um *site* é mais complexa em sistemas distribuídos do que em sistemas centralizados.

Com o desenvolvimento dos computadores pessoais e das redes locais de computadores, os sistemas de banco de dados passaram de centralizados para sistemas de banco de dados distribuídos. Houve desta forma, uma evolução na tecnologia utilizada em banco de dados, criando-se os recursos de *front-end*, que são recursos utilizados pelo cliente do banco de dados, e recursos de *back-end* que são utilizados pelos desenvolvedores de sistemas de banco de dados.

Um banco de dados distribuído pode ser considerado como um conjunto de banco de dados parciais, que compartilha de um esquema que é comum a diversos *sites* e que possui um coordenador local e um coordenador global. Os processadores comunicam-se uns com os outros através das redes de comunicação que tratam do roteamento e das estratégias de comunicação. Esse assunto será discutido com maiores detalhes no capítulo 2.

## Apêndice B

### Gerenciamento de Transações

As operações que ocorrem em um banco de dados, formando uma única unidade lógica de trabalho, do ponto de vista do usuário são conhecidas como transações.

Transação é uma coleção de ações que fazem com que os estados consistentes de um banco de dados, sofram alterações para novos estados consistentes. No início de uma transação, o banco de dados está em um estado consistente, durante a execução da transação o banco de dados pode transitar por estados de inconsistência, porém, ao finalizar a transação o banco de dados estará novamente em um estado consistente.

Uma transação é o resultado da execução de um programa pelo usuário, programa esse, escrito em uma linguagem de manipulação de dados (DML).

Neste capítulo será apresentada uma breve noção sobre transações, as suas definições e conceitos, os estados pelos quais uma transação pode passar durante a execução da mesma.

#### 1. Propriedades de uma Transação

Algumas propriedades (ACID) que deve-se levar em conta quando se fala sobre transações:

- **Atomicidade:** todas ou nenhuma das operações que envolvem a transação são realizadas. Se uma das transações for interrompida por falha, os resultados parciais que foram obtidos devem ser desfeitos. A atividade de preservação da atomicidade da transação quanto do aborto de uma transação devido a erros de entrada, sobrecarga do sistema ou *deadlocks* (estado de bloqueio geral do sistema), é chamada de recuperação da transação.
- **Consistência:** as transações executadas não podem violar as restrições de integridade, isto é, ao término da transação o banco de dados precisa estar em um estado consistente.

- Isolamento: o resultado das transações que são executadas em concorrência, deve ser o mesmo se as transações fossem executadas isoladamente, seguindo uma ordem pré-definida. Uma transação que não completou toda a operação, não deve revelar os valores obtidos a outra transação até a mesma ser efetivada, isto é *Committed*.
- Durabilidade: a partir do momento que a transação foi realizada, o sistema deve assegurar que os resultados obtidos, devem permanecer ignorando falhas que possam ocorrer.

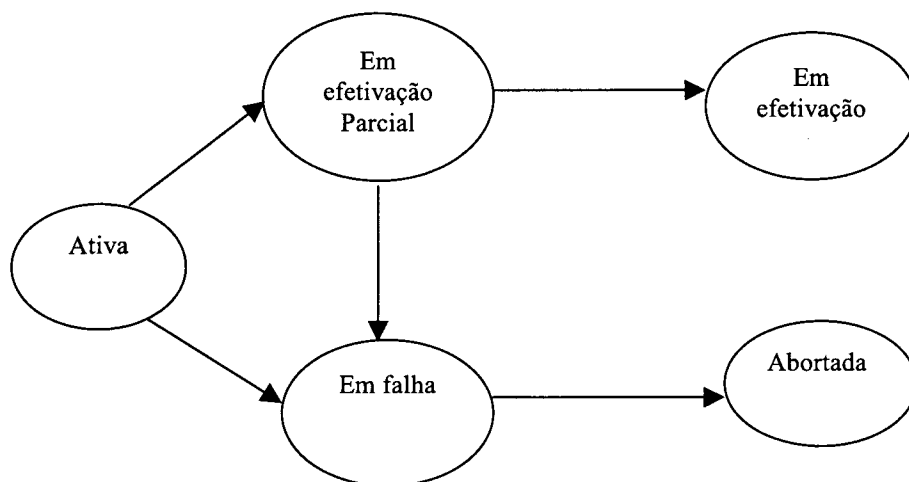
### 1.1 Estados da Transação

Se não ocorrerem falhas, todas as transações são realizadas com sucesso, entretanto nem sempre isso é conseguido, pois as falhas são possíveis em uma transação. Quando ocorrem falhas, dizemos que a transação foi abortada ou desfeita (*rolled back*) [Korth,1999], assim, quaisquer atualizações que a transação tiver feito no banco de dados deverão ser desfeitas, respeitando a propriedade de atomicidade.

No entanto, se não ocorrerem falhas, a transação é realizada com sucesso, dizemos que ela foi efetivada (*commit*) [Korth,1999]. O banco de dados é transformado em um novo estado consistente que deve permanecer mesmo que ocorram falhas no sistema. Uma vez que a transação atingir o estado de *Commit*, não pode-se desfazer seus efeitos, mesmo que a transação seja abortada. O único modo de reverter este quadro é executando uma transação de compensação, que é realizada pelo usuário do banco de dados.

Para que a transação seja realizada com sucesso, ela deve estar em um dos seguintes estados:

- Ativa (estado inicial): a transação permanece nesse estado enquanto estiver sendo executada;
- Em efetivação parcial: após a última declaração ter sido executada;
- Em falha: após a descoberta de que a execução normal não pode ser executada;
- Abortada: a transação foi abortada e o banco de dados voltou ao estado inicial de antes da execução da transação;
- Em efetivação: quando a transação foi executada com sucesso.



**Figura 1.1 Diagrama de estado de uma transação [ Korth,1999]**

Uma transação está efetivada somente quando ela tiver passado pelo estado de efetivação, entretanto se a transação for interrompida ela estará no estado de abortada, e finalmente se a transação for concluída, ela se encontrará no estado de efetivação ou abortada.

Uma transação tem seu início no estado ativo, quando ela acaba a última operação está no estado de efetivação parcial, nesse momento, a transação foi completada e pode ocorrer uma falha de hardware impedindo que a transação seja concluída com êxito.

A partir do momento em que o sistema determina que a transação não pode prosseguir sua execução normal, ela se encontrará no estado de falha, devendo ser desfeita e com isso passa para o estado de abortada e o sistema pode então reiniciar ou matar a transação.

O componente do gerenciamento de recuperação em um banco de dados, usa um esquema simples, onde apenas uma transação ficará ativa por vez, utilizando cópias do banco de dados que são conhecidas como *shadow* (sombra), levando em conta que o banco de dados é simplesmente um arquivo no disco, onde um ponteiro aponta para a cópia corrente do banco de dados.

Uma transação que deseja realizar operações no banco de dados, cria primeiro uma cópia completa dele. Todas as operações são feitas nesta nova cópia, não interferindo nos dados da cópia original. Se ocorrerem falhas de transação e, a mesma



for abortada, a cópia nova é apagada, e com isso, a cópia original do banco de dados não será afetada.

Se a transação foi completada com sucesso, o ponteiro é atualizado e aponta para a nova cópia do banco de dados que se transforma na cópia atual, e, a cópia antiga é apagada. Assim, ou todas as atualizações da transação são efetivadas ou nenhum de seus efeitos estarão refletidos no banco de dados, garantindo a propriedade de Atomicidade [Korth, 1999].

## 1.2 A Execução de Transações

A nível lógico, um banco de dados distribuído é descrito por um esquema conceitual global, e, os objetos descritos neste esquema, serão chamados de objetos lógicos. A nível físico, o banco de dados é definido através de uma série de esquemas internos, um para cada *site* em que o banco é armazenado. Os objetos descritos nos esquemas internos serão chamados de objetos físicos[Casanova,1985].

Um SGBDD é uma coleção de SGBDs locais interligados pelo SGBD global. Cada *site* na rede, possui uma cópia do SGBD global, que é dividido em três componentes:

- Diretório de dados globais (DDG): contém as descrições dos objetos físicos, lógicos e dos mapeamentos entre estes;
- Gerente de transações (GT): faz o controle e a interpretação do processamento de consultas e transações que são executadas pelo sistema;
- Gerente de dados (GD): executa a interface com o SGBD local.

Quando uma transação T é chamada, o gerente de transações no *site* de origem assume o controle do processamento desta transação T e aciona os mecanismos necessários antes do início da transação, durante a transação e após o termino da transação.

No início da transação, o GT identifica as ações que serão executadas nessa transação, deste modo, se houver necessidade de desfazer parte das ações o GT terá condições de fazê-lo sem problemas.

Durante a execução da transação, o gerente de transações interage com os mecanismos de controle de concorrência e com os mecanismos de controle de

integridade. O gerente de transações, precisa estar atento ao fato de que os recursos do sistema podem ser requisitados pelas inúmeras transações que estão sendo realizadas. À medida que o gerente de transações concede o uso dos recursos, ele aciona os procedimentos de controle de concorrência, evitando assim que mais de uma transação tenha acesso, simultaneamente, a recursos que não podem ser compartilhados. Pelo fato de cancelar transações, o gerente de transações mantém um histórico da seqüência de ações que estão sendo executadas em favor de uma transação, desta forma, é possível realizar o controle de integridade. Este histórico é tarefa realizada pelo controle de integridade, com o qual o gerente de transações deve manter contato direto durante a execução da transação [Korth, 1999].

Quando ocorre o término da transação, é tomada uma decisão no sentido de tornar público todos os efeitos das ações que foram executadas em benefício da transação. Se os efeitos não forem divulgados, as ações devem então ter seus efeitos removidos do BD.

Uma transação pode ter seu término de forma natural ou devido a causas excepcionais. Quando ocorre o término por causa excepcional, a transação sempre será cancelada, entrando em ação nesse instante os mecanismos de controle de integridade, que invertem a transação, restaurando o BD a um estado consistente, de forma que possa recomeçar suas operações normais.

Pode ocorrer também, de transações serem canceladas mesmo estando em processamento normal. Isso acontece, quando o usuário tem a opção de cancelar uma operação antes que ela possa se completar, nesse caso, também é acionado o mecanismo de integridade para que possa garantir a consistência do BD.

O gerente de transações, ao término da transação, deve garantir que todos os recursos utilizados temporariamente pela transação, voltem ao controle do SGBD, sendo postos à disposição de outras transações.

Chega-se a conclusão que ao se trabalhar com sistemas distribuídos, o término de uma transação não é assim tão simples. Todos os *sites* que estiveram envolvidos com a transação, devem refletir ou remover todos os efeitos das ações executadas pela transação. Protocolos especiais devem ser ativados para garantir a atomicidade da transação como um todo, mesmo na presença de falhas que podem afetar a operação normal de vários dentre os *sites* participantes [Bernstein, 1997].

## Referência Bibliografia

[Bernstein, 1997] Philip A. Bernstein and Eric Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers, 1997.

[Banco de dados, 1] <http://www.uri.com.br/~mzp/mestrado/distribuido/distrifim.html>

[Banco de dados, 2] [http://www.ibsuper.net/artigoib\\_bdperfeito.htm](http://www.ibsuper.net/artigoib_bdperfeito.htm)

[Casanova, 1985] Marco Antônio Casanova e Arnaldo Vieira Moura. *Princípios de sistemas de gerencia de banco de dados distribuídos*. Rio de Janeiro: Campus, 1985.

[Ceri, 1984] Stefano Ceri and Giuseppe Pelagatti. *Ditributed Databases Principles & Systems*. McGraw-Hill computer science series. 1984.

[Charles, 1999] Charles Dye. *Oracle Distributed Systems*. 1999.

[Date, 2001] C. J. Date. *Introdução a Sistemas de Banco de dados*. Rio de Janeiro: Campus, 2001.

[DB2, 1]

<http://www-4.ibm.com/cgi->

[bin/DB2www/data/DB2/udb/winos2unix/support/document.d2w/report?fn=DB2v7d0DB2d015.htm](http://www-4.ibm.com/cgi-bin/DB2www/data/DB2/udb/winos2unix/support/document.d2w/report?fn=DB2v7d0DB2d015.htm)

[DB2, 2] <http://www.ibm.com/br/ibm/history/>

[DB2, 3]

<http://www-4.ibm.com/cgi->

[bin/DB2www/data/DB2/udb/winos2unix/support/document.d2w/report?fn=DB2d0DB2d0.htm#ToC](http://www-4.ibm.com/cgi-bin/DB2www/data/DB2/udb/winos2unix/support/document.d2w/report?fn=DB2d0DB2d0.htm#ToC)

[Finger e Ferreira, 2000] Ferreira, João Eduardo & Finger, Marcelo. Controle de Concorrência e distribuição de dados: teoria clássica, suas limitações e extensões modernas. São Paulo, IME-USP, 2000.

[Gunderloy, 2001] Mike Gunderloy e Joseph L. Jordan. *Dominando o SQL Server 2000*. Trad. Lávio Pareschi. São Paulo: Makron Books, 2001.

[Gray, 1987] J. Gray. *Why Do Computers Stop and What Can Be Done About It*. Tutorial Notes, CIPS (Canadian Information Processing Society) Edmonton '87 Conf., Edmonton, Canada, November 1987.

[Ingres, 1] [http://www.ca.com/products/Ingres/pd\\_Ingres2.pdf](http://www.ca.com/products/Ingres/pd_Ingres2.pdf)

[Ingres, 2] [http://www.cai.com/products/Ingres/documentation\\_set.htm](http://www.cai.com/products/Ingres/documentation_set.htm)

[Ingres, 3] <http://www.cs.mu.oz.au/~yuan/Ingres/Ingres.html>

[Ingres, 4] <http://www.cai.com/products/Ingres.htm>

[Korth, 1999] Abraham Silberschatz, Henry F. Korth e S. Sudarsham. *Sistema de Banco de Dados*. São Paulo: Makron Books, 1999.

[Longbottom, 1980] R. Longbottom. *Computer System Reliability*. Chichester, England: Wiley, 1980.

[Molina, 2001] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. *Implementação de sistemas de banco de dados*. Trad. Vandenberg D. de Souza. Rio de Janeiro: Campus, 2001.

[Oracle, 1] <http://www.profissionalOracle.com.br/>

[Oracle, 2] <http://www.Oracle.com/br/>

[Oracle, 3] <http://otn.Oracle.com>

[Oracle, 4] <http://technet.Oracle.com/docs/content.html>

[Ozsu,1999] M. Tamer Ozsu and Patrick Valduriez. *Principles of Distributed Database System*. 2nd ed. p.cm. 1999.

[Ozsu,2001] M. Tamer Ozsu and Patrick Valduriez. *Princípios de Sistemas de Banco de Dados Distribuídos*. Trad. Vandenberg D. de Souza. Rio de Janeiro: Campus, 2001.

[Sarin, 2000] Sarin, Sumit. *Oracle DBA: dicas e técnicas*. Trad. Kátia Roque. Rio de Janeiro: Campus, 2000.

[Siewiorek and Swarz, 1982] D. P. Siewiorek and R. S. Swarz. *The Theory and Practice of Reliable System Design*. Bedford, Mass.: Digital Press, 1982.

[SQL Server, 1] <http://www.microsoft.com/brasil/sql/productinfo/reliable.stm>

[SQL Server, 2] <http://www2.uol.com.br/info/aberto/forum/sqlserver.shl>

[SQL Server, 3] <http://www.microsoft.com/sql/techinfo/administration/2000/default.asp>

[SQL Server, 4] <http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/sql>

[Vieira, 2000] trad. Daniel Vieira. *Microsoft SQL Server*. Rio de Janeiro: Campus, 2000.