# Paradox X InterBase

## Introdução

A primeira vista, as tabelas Paradox não apresentam muitas diferenças das tabelas criadas no InterBase e as seguintes semelhanças são evidentes:

- o acesso pode ser feito através de um *Alias*;
- os tipos dos campos possíveis são similares, apesar de possuirem nomes diferentes;
- as tabelas podem ser criadas com o *DataBase Desktop*;
- são utilizados os mesmos componentes *TTable* e *TQuery* para acesso;

Na realidade, o BDE (*Borland DataBase Engine*) cria uma ilusão de que tabelas InterBase e Paradox comportam-se da mesma maneira.

Para alguns desenvolvedores, entretanto esta ilusão termina logo. A primeira decepção vem com a utilização do *Database Desktop* para manipulação de tabelas InterBase. Enquanto o *Database Desktop* é a ferramenta ideal para criação e reestruturação de tabelas Paradox, é deficiente com relação ao Interbase, onde a restruturação e a utilização de características mais avançadas só podem ser alcançadas através da montagem de scripts que serão executados no Interbase Windows ISQL.

As pesquisas e índices no InterBase diferenciam maiúsculas e minúsculas, enquanto que no Paradox esta diferenciação é configurável. Ainda no InterBase a definição de chaves primárias e estrangeiras é realizada facilmente, porém a alteração destas chaves não é tão trivial. Algumas operações utilizando o InterBase são mais lentas do que no Paradox. Torna-se rapidamente claro que o InterBase não é automaticamente melhor que o Paradox.

A idéia de que o InterBase não é sempre mais adequado do que o Paradox é verídica.Os dois produtos apresentam diferenças significativas e a escolha de qual utilizar é plenamente dependente das condições e dos objetivos do aplicativo final. Na realidade, a única coisa que eles tem em comum é que ambos armazenam dados em tabelas. Cada um possui pontos fortes e fracos dependendo da realidade a ser trabalhada. O importante é decidir qual deles é apropriado para uma situação particular. A decisão tomada, usar Paradox ou InterBase, afetará de forma significativa o desenvolvimento do seu projeto.

## Paradox: Baseado em Arquivos

O Paradox é um sistema de banco de dados baseado em arquivos. Os arquivos de dados contêm registros de dados que possuem uma ordem fixa. Em outras palavras, o registro de número 106 será sempre o mesmo registro, até que seja movido fisicamente dentro do arquivo, através de uma operação de ordenação, por exemplo. O que é mais importante, ele será sempre o sucessor do registro 105 e o antecessor do 107, até que esta ordem seja explicitamente alterada. Isto permite uma fácil navegação entre os registros, através do cursor, já que é possível localizar um registro em uma tabela através da sua posição, sem necessidade de referenciar seu conteúdo.

Esta ordenação explícita de registros em uma tabela, apresenta algumas vantagens: a movimentação para o início ou para o final de um arquivo de dados é simples e os registros podem ser facilmente relidos quando um cursor é posicionado sobre eles. A concepção de navegação (*browsing*) pode ser conveniente para usuários e desenvolvedores. Isto permite que os registros sejam manipulados um de cada vez e em uma ordem previsível. Este comportamento de navegação nas tabelas Paradox é uma das características mais difíceis de serem adaptadas ao sistema InterBase, e muitos desenvolvedores em InterBase têm grandes problemas em efetuar esta transição.

**InterBase: Baseado em Conjuntos**

O InterBase é na realidade um sistema de banco de bados relacional. As tabelas não são armazenadas em arquivos individuais e o que é mais interessante: os registros não encontram-se ordenados. Matematicamente falando, os conjuntos são desordenados. A ordem é "descoberta" somente quando o conjunto for representado, como por exemplo em uma consulta ao banco de dados. Você não pode contar duas vezes com o fato de o mesmo registro ser o registro número 105, a menos que explicitamente seja imposta uma certa ordenação na consulta. Uma vez que não podemos identificar positivamente um registro pela sua posição dentro de uma tabela devemos nos referir a valores internos a este registro. Entretanto, para identificações positivas de um registro, pelo menos um campo ou a combinação de alguns campos devem conter um valor único para cada registro, o que é conhecido por Chave Primária.

É possível que mais de um campo ou combinações de campos possam fornecer valores únicos. Estes campos - ou combinações - são denominados Chaves Candidatas. É imprescindível para o gerenciamento de um sistema de banco de dados a identificação de Chaves Primárias. Uma tabela que possui uma Chave Primária é chamada de *R-table* e todos os *Data Sets* (*tables, query e stored-procedures*) devem ser *R-tables* para que o modelo relacional possa funcionar corretamente.

A vantagem deste conceito de dados baseado em conjunto é que os conjuntos e as operações neles realizadas possuem a propriedade de "encapsulamento". Isto significa que quando realizamos operações em um conjunto sempre geraremos um outro conjunto, que poderá, então, ter uma nova operação com ele realizada, produzindo um novo conjunto, e assim suscessivamente. Este é um poderoso conceito lógico que, se corretamente implementado, tornará o desenvolvimento de aplicações independente das características físicas de armazenamento. Esta é a base dos modelos relacionais.

Os sistemas "baseados em cursor" (*cursor-based*) como o Paradox permitem que você trabalhe somente com um registro de cada vez. Sistemas "baseados em conjuntos" (*set-based*) permitem a manipulação de conjuntos de dados como se fossem uma entidade única. Esta característica permite a simplificação do desenvolvimento de aplicações, além de melhor garantir a integridade dos dados.

**Projetos Físicos e os Impactos de Velocidade**

O Paradox é um sistema-cliente no qual os dados são completamente manipulados por clientes individuais. Quando você quiser que os dados sejam lidos ou manipulados, eles devem ser transportados para a aplicação. Cada aplicação manipula todo o processamento por si só. Se múltiplos usuários estiverem acessando a tabela simultaneamente sobre uma rede, cada aplicação usuária transporta os dados requeridos para a máquina usuária.

Cada instância de aplicação Paradox não tem retorno de outra. Se uma instância precisa garantir a estabilidade dos dados por alguma razão, ela deve proibir outras instâncias de alterarem os dados através de um esquema de travamento (*locking scheme*).

Quando, em uma aplicação Paradox, precisamos procurar em uma tabela, a seguinte rotina será efetuada: (1) os dados brutos e índices necessários devem ser carregados na memória da máquina cliente; (2) a atividade física da procura é conduzida; (3) os resultados são gerados; e (4) os dados tornados desnecessários após a operação são então descartados. Esta rotina é repetida para cada operação sobre cada cliente. O servidor de arquivos que contem os arquivos de dados não faz nada além de enviar os dados brutos requeridos, através da rede, para as máquinas clientes, sem realizar quaisquer esforços de processamento dos dados. Funciona, assim, como um disco rígido remoto.

Em suma, o Paradox é veloz em um disco rígido local mas a sua performance diminui drasticamente em uma rede. A rede fica sobrecarregada, com um grande volume de processamentos

sendo realizados repetidamente na máquina cliente. Mesmo em uma rede pequena a performance é rapidamente afetada quando um novo usuário é conectado.

O InterBase é um sistema "baseado em servidores" (*server based*). Em vez de diferentes processos manipulando fisicamente os dados armazenados, somente um processo central, "rodando" na máquina servidora, possui acesso direto aos dados. Todas as aplicações clientes desenvolvem políticas de requerimento ao processo servidor, o qual é processado diretamente na máquina servidora enquanto o cliente aguarda.

Quando o servidor termina ele transmite apenas os resultados de volta ao cliente, que retoma então suas atividades. O impacto mais direto deste esquema é que a rede não necessita ficar congestionada com grandes volumes de dados redundantes sendo enviados pelos clientes. Alem disso, as tarefas frequentemente complexas de processamento de dados podem ser delegadas de máquinas clientes, normalmente menos poderosas, para as normalmente mais poderosas máquinas servidoras.

Deve-se perceber que tudo sobre o desenvolvimento em InterBase implica em um ambiente multi-usuário. O InterBase foi desenvolvido desde o princícpio como um sistema multi-usuário. O Paradox, por outro lado, foi basicamente desenvolvido para um único usuário, sendo que características que visavam permitir o atendimento a múltiplos usuários foram depois sendo adicionadas.

**InterBase Não é Para Navegação**

Conforme foi mencionado anteriormente o InterBase constitui-se em um verdadeiro RDBMS (Sistema Relacional de Gerenciamento de Banco de Dados) mas a performance de algumas operações são realmente mais lentas do que em tabelas Paradox. Esta questão torna-se evidente quando temos uma tabela com uma quantidade muito grande de registros. Se você utilizar o método *Last* com o objetivo de mover o ponteiro para o último registro da tabela certamente perceberá uma grande diferença na performance dos dois sistemas. Portanto, para estas operações, a utilização de tabelas Paradox torna a aplicação mais veloz do que em tabelas InterBase.Em tabelas Paradox o índice aponta para um endereço físico onde se encontra o registro.

O cursor na tabela Paradox moverá quase que instantaneamente para a posição determinada porque ele conhece a localização física (endereço) deste registro na tabela. Esta operação na tabela InterBase poderá apresentar alguns problemas. Primeiro porque os dados encontram-se desordenados e existe uma questão sobre o que constitui o "Último". A operação deve impor uma ordem antes de decidir qual registro é o último. Por *default* o último registro será considerado aquele registro cuja Chave Primária possui o maior valor.

Portanto, é necessário determinar o valor da maior Chave Primária. Uma vez que este valor foi determinado o registro que possui este valor pode ser recuperado. Se for necessário recuperar os últimos registros (como no caso da apresentação dos dados em um grid) o processo deve ser repetido algumas vezes, tornando-se um pouco complicado. Para recuperar o penúltimo registro, o InterBase deve agora encontrar o maior valor da chave primária que seja inferior ao valor máximo desta. Para o antepenúltimo registro, ele deve procurar o valor mais próximo ao mais próximo do valor máximo da chave primária, e assim consecutivamente, até o número necessário de registros ser recuperado. Uma operação que é uma "moleza" para o Paradox torna-se uma tremenda "dor-de-cabeça" para o InterBase. Em geral, a navegação de trás para frente através de tabelas SQL é ineficiente. O BDE armazena grupos de registros para minimizar este problema, mas se você avançar muito para trás em tabelas relativamente extensas, acabará tendo que aguardar por pausas significativas.

**Travamento (*locking*)**

Quando dois usários tentam acessar a mesma faixa de dados podem surgir problemas ameaçando a integridade do banco de dados. O Paradox e o InterBase lidam com estes problemas de maneiras diferentes.

Como o Paradox não possui nenhum conhecimento do que outro processo Paradox está fazendo, ele utiliza um esquema pessimista de travamento *(Pessimistic Locking Scheme)*. Assim que um usuário tenta mudar um registro o registro é travado. Nenhum outro usuário pode fazer alterações neste registro até que o primeiro usuário termine as alterações ou cancele a operaçao. Por um lado, este esquema de travamento é positivo se considerarmos que o usuário que obteve o travamento poderá concluir com sucesso a sua operação de alteração. Por outro lado, isto é negativo no sentido em que um usuário poderá mononopolizar um registro indefinidamente. Isto seria um problema, por exemplo, num sistema de reserva de viagens. Um viajante indeciso poderia "trancar" um assento por um longo período, levando os outros a acreditarem que assento já foi tomado, sendo que no final o indeciso pode acabar desistindo do assento.

O InterBase, por outro lado, lida com a manipulação de todos os dados através de um esquema de concorrência otimista *(Optimistic Concurrency Scheme)*. Na verdade, quando um usuário processa uma alteração o registro não impedirá que outras pessoas tentem também alterar este registro. Quando um usuário começa a alterar um registro InterBase uma cópia do registro original é salva. O usuário executa seu serviço, mas os outros usuários não estão sob nenhuma forma impedidos de acessar o mesmo registro.

Quando algum usuário termina sua alteração e efetua um *post* (atualização no banco) a cópia do registro original é comparada com o registro corrente. Se os valores são diferentes (provavelmente porque outro usuário alterou mais rapidamente o mesmo registro) as alterações efetuadas pelo usuário no registro são rejeitadas.

Isto significa que usuários individuais não podem travar o acesso de outros usuários ao mesmo registro. No exemplo anterior de reservas de viagens, o primeiro passageiro a confirmar a reserva ficaria com o lugar, mesmo se vários outros usuários estivessem simultaneamente considerando o mesmo lugar. O lado ruim da estória, porém, é que as alterações são rejeitadas apenas no momento em que se tenta efetuar a atualização no banco, após todo o trabalho de alteração ter sido efetivado. Isto pode ser corrigido relendo-se (pelo método *refresh*) os campos alterados e tentando-se novamente efetuar a atualização. Os componentes *TTable* ou *TQuery* tornam esta operação transparente através da propriedade *UpdateMode*. A verificação, quando enviamos comando de atualização no banco, pode ser feita dependendo do valor desta propriedade, em todos as campos, somente nos campos alterados ou somente nas chaves primárias.

Estas duas visões refletem as diferenças básicas nas duas filosofias. O modelo pessimista Paradox assume que as colisões serão frequentes e deverão ser tratadas de uma forma rigorosa, enquanto o modelo otimista InterBase assume que as colisões serão ocasionais e maximiza a habilitação dos usuários para o compartilhamento de dados sem interferência de um com o outro, enquanto estiver sendo mantida a integridade.

Um importante benefício do modelo do InterBase é que um usuário que quer visualizar um conjunto de dados estáveis, talvez para gerar uma série de relatórios que precisam refletir a situação atual dos dados, não sofrerá interferencia de outro usuário que estiver naquele momento alterando os dados. Em tabelas Paradox, o gerador de relatórios deverá colocar um "*write lock*" na tabela para garantir que os dados não sejam alterados e ninguém poderá efetuar transações na tabela até que este *lock* tenha deixado de existir.

No InterBase as versões dos registros mais antigos, são retidas durante o tempo em que o usuário estiver algum interesse sobre elas. Isto significa que nem os leitores dos dados impedem a ação dos "editores" e nem os últimos comprometem o resultado dos primeiros. InterBase é o único banco SQL que torna isto transparente. Quando os adeptos do InterBase são questionados sobre as

vantagens do banco, este controle da disponibilidade do registro é normalmente a primeira vantagem a ser mencionada.

**Processamento de Transações**

Como você já sabe, o modelo "baseado em conjuntos" constitui-se em um conjunto de dados que podem ser tratados como entidades individuais, levando em consideração o conteúdo específico destes conjuntos. O processamento de transações é uma extensão destas idéias. Uma transação é um grupo de operações onde ou todas elas são bem ou mal sucedidas. Nunca deverá ocorrer a situação onde apenas algumas sejam bem ou mal sucedidas.

Por exemplo, um caixa automático realiza transações com bancos de dados. Sempre que você retira dinheiro, duas operações devem ser realizadas para que o banco possa contabilizar seu ativo apropriadamente: o saldo da sua conta deve ser reduzido e o saldo de dinheiro disponível no banco deve ser reduzido na mesma quantia. Obviamente, a situação preferível é aquela na qual as duas operações sejam bem sucedidas, mas se acabar a energia no meio de uma operação, é totalmente inaceitável que uma conta seja atualizada e a outra não - ambas operações deveriam "falhar" para manter uma contabilização apropriada.

O processamento de transações permite que isto aconteça. As operações em uma transação não são permanentes até que seja efetuado o *commit* na transação. Até que isto aconteça, as operações podem ser desfeitas, retornando-se ao ponto de partida. Um *Rollback* pode ser implementado explicitamente, utilizando-se o método *Rollback*, ou automaticamente, quando ocorre uma falha do sistema.

O InterBase suporta totalmente o conceito de transações. Na realidade, todas as operações ocorrem dentro de um contexto de transação. Na ausência de um controle explícito do programador, o BDE automaticamente envolve todas as operações na sua própia transação. Por exemplo, toda vez que você atualiza um registro, a transação é inicializada (*started*) e finalizada (*commited*) automaticamente após cada confirmação (*post*). Usando o componente *TDataBase* você pode explicitamente controlar uma única transação, e também fazê-la conter quantas operações você quiser.

O Paradox porém não suporta transações. Sempre que um registro é atualizado no banco, as mudanças são permanentemente gravadas na tabela. Será necessária uma nova alteração para desfazer manualmente as alterações anteriores se um *Rollback* for desejado. Além disto, o sistema não irá garantir que, em um grupo de operações, todas elas serão bem sucedidas ou todas elas falharão. É possível simular algumas destas características através de certos truques de programação e tabelas temporárias, mas eventualmente, haverá a necessidade de se modificarem os registros um de cada vez, em lote, o que deixaria aberturas para a ocorrência de falhas. Além disto, é impossível de se programar um aplicativo Paradox para se recuperar de uma falha de sistema como queda de força ou danos na memória secundária.

**Gatilhos (*triggers*) e procedimentos (*procedures*)**

Uma *Stored Procedure* consiste em um "pedaço" de código armazenado em um banco de dados junto aos dados que este contém. Isto permite ao servidor efetuar manipulações complexas de dados inteiramente no próprio servidor. As vantagens principais disto são que processos ainda mais complexos podem ser delegados ao servidor, e qualquer número de diferentes aplicações clientes podem chamar os mesmos procedimentos. Se o procedimento for modificado no servidor, nenhuma das aplicações necessitará ser reescrita, desde que a interface do procedimento continue a mesma.

Um *trigger* consiste em uma *Stored Procedure* que não é explicitamente chamado por uma aplicação, mas é executado em resposta a uma ação ocorrida com determinados dados, como por

exemplo, uma inserção de novos registros. A utilização de *triggers* permite que você realize validações de dados extremamente complexas, sendo que as operações planejadas ocorrerão garantidamente no interior da mesma transação que disparou o "engatilhamento". Se alguma das operações falhar, todas as alterações feitas por *triggers* associados a esta operação serão também desfeitas (*rolled back*).

O InterBase suporta *Stored Procedures* que retornam *Result sets*, os quais são tratados exatamente como tabelas somente de leituras, assim como *triggers* que simplesmente executam transações de dados e não retornam nenhum resultado. O InterBase suporta essencialmente um número ilimitado de *triggers* para cada tabela os quais podem ocorrer antes ou depois das operações de inserção, alteração e remoção de registros. Se mais de um *trigger* é associado a uma operação a ordem de execução pode ser especificada. *Triggers* podem efetuar mudanças que disparem outros *triggers*, numa reação em cadeia, mas estas ações em cascata continuarão contidas numa única transação.

O Paradox não suporta nenhum destes conceitos. Todos os processamentos de dados devem ser realizados no cliente. Cada aplicação deve conter a mesma codificação para manutenção dos dados, e cada uma delas deve também ser modificada caso o método de manipulação de dados necessitar ser alterado.

Não existe uma garantia que uma aplicação será completada uma vez que tenha sido iniciada. Por exemplo a operação de se efetuar uma deleção em cascata de registros "detalhe" após a deleção de um registro "mestre" pode falhar no seu meio, deixando alguns registros detalhe sem serem apagados. Se esta "cascata" fosse implementada por meio de *triggers* no InterBase, ou todos os registros ou nenhum deles seriam apagados. Além disto, numa aplicação Paradox, a codificação para perpetuar a deleção deveria ser escrita em cada uma das aplicações diferentes que utilizassem os dados do sistema Paradox. Utilizando-se InterBase, por outro lado, a codificação necessita ser escrita apenas uma vez, no trigger. A aplicação simplesmente deleta o registro "mestre" e o InterBase cuida da deleção dos registros "detalhe".

## Fazendo a melhor escolha

Escolher entre Paradox e InterBase pode ter implicação importante para o seu projeto. Portanto, é essencial saber o que é mais adequado em cada situação. A figura 1 mostra alguns princípios gerais que podem ajudá-lo a tomar a melhor decisão.

Isto são sugestões e não devem ser encaradas como regras. A maioria pesume que uma rede estará envolvida. Se você está implementando um sistema mono-usuário, o Paradox é usualmente a melhor escolha. O servidor InterBase local pode ser indicado para um sistema mono-usuário, mas sem os aspectos de concorrência, as vantagens básicas do InterBase não estarão sendo utilizadas.

## Conclusão

É importante relembrar que o Paradox e o InterBase são sistemas consideravelmente diferentes, mesmo que o BDE crie uma ilusão de semelhança. Esta é uma sedutora mas perigosa idéia pois converter uma aplicação de um mecanismo para outro (InterBase em Paradox ou vice e versa) envolve muito mais do que uma simples mudança de *Alias*. Requer na verdade, significativas diferenças de conceito e projeto.

Escolher qual sistema utilizar é uma decisão crucial que deve ser tomada no início do projeto. Esta decisão terá um impacto profundo no desenvolvimento da sua aplicação.

| ***Paradox é melhor quando...*** | ***InterBase é melhor quando...*** |
|---|---|

| | |
|---|---|
| a aplicação é utilizada com menos de 10 usuários concorrentemente | a aplicação é utilizada com mais de 10 usuários concorrentemente |
| dados e estruturas de dados devem ser facilmente modificados por usuários finais | dados devem ser centralizados, mantidos e protegidos |
| a máquina cliente é proporcionalmente mais potente que a máquina servidora | a máquina servidora é muito mais potente que a máquina cliente |
| largura de banda da rede satisfatória | rede está carregada |
| velocidade e "conveniencia" são mais importantes que integridade | integridade de dados é crucial |
| baixa disponibilidade de administradores de rede e banco de dados qualificados | disponibilidade de administradores de rede e banco de dados qualificados |
| somente uma aplicação acessará rotineiramente os dados | várias aplicações poderão acessar os dados |
| as aplicações serão as responsáveis pela manutenção da integridade de dados | o banco será o responsável pela integridade de dados independentemente das aplicações |
| pequena ou moderada quantidade de dados (< 100 MB) | moderada a grande quantidade de dados (> 100 MB) |

Figura 1

**Adaptação livre do artigo "InterBase vs. Paradox"**
Delphi Informant Magazine - volume 2, número 6, Junho de 1996, Pag. 28-35

Danielle Fortes Lopes - dani@wim.com.br
**WIM Informática Ltda.**