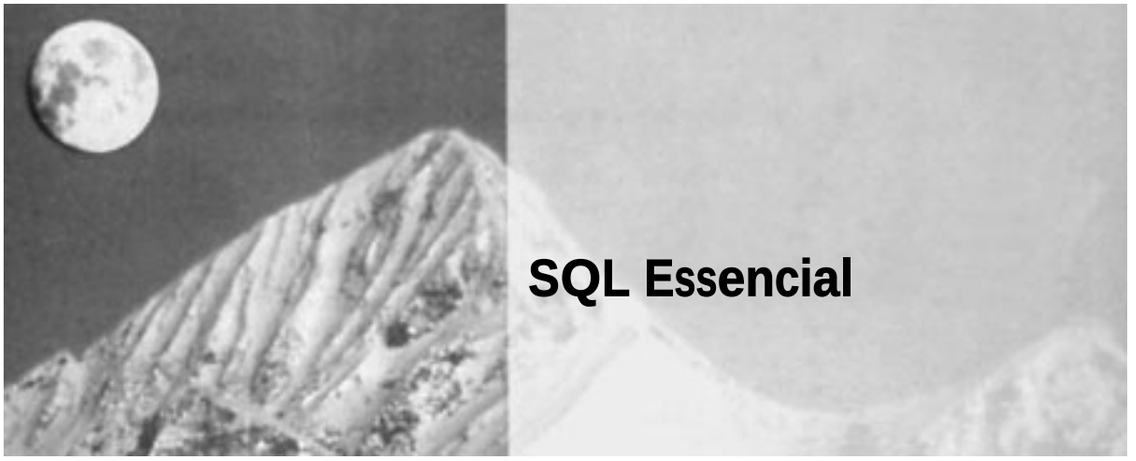

SQL Essencial

Este capítulo de bônus é fornecido com o livro Dominando o Delphi 6. Trata-se de uma introdução básica ao SQL para acompanhar o Capítulo 14, "Programação Cliente/Servidor".

Os pacotes de SGBDR (sistema gerenciador de bancos de dados relacional) geralmente são tão baseados no SQL (Structured Query Language, linguagem de consulta estruturada) que freqüentemente são chamados de servidores SQL. O padrão SQL é definido por um comitê ANSI/ISO, embora muitos servidores usem extensões personalizadas do último padrão oficial (chamado SQL-92 ou SQL2). Recentemente, muitos servidores começaram a adicionar extensões de objeto, que devem fazer parte do futuro padrão SQL3.

Ao contrário do que seu nome parece implicar, o SQL é usado não apenas para consultar um banco de dados e manipular seus dados, mas também para defini-lo. O SQL consiste em duas áreas: uma DDL (Data Definition Language, linguagem de definição de dados), incluindo os comandos para criação de bancos de dados e tabelas; e uma DML (Data Manipulation Language, linguagem de manipulação de dados), incluindo os comandos de consulta. As duas próximas seções explorarão essas duas áreas distintas.

NOTA Todos os trechos de código SQL apresentados neste capítulo foram testados com InterBase 5 e 6.

SQL: a Linguagem de Definição de Dados

Os comandos da DDL geralmente são usados apenas ao se projetar e manter um banco de dados; eles não são usados diretamente por um aplicativo cliente. O ponto de partida é o comando create database, que tem uma sintaxe muito simples:

```
create database "mddb.gdb";
```

Esse comando cria um novo banco de dados (na prática, um novo arquivo GDB do InterBase) no diretório corrente ou no caminho indicado. Na instrução anterior, observe o ponto-e-vírgula final, usado como um terminador de comando pelo console InterBase. A operação oposta é eliminar um banco de dados e você também pode modificar alguns dos parâmetros de criação com alter database.

NOTA Em geral, os programas clientes não devem operar sobre metadados, uma operação que na maioria das organizações concorre com as responsabilidades do administrador de banco de dados. Adicionamos essas chamadas em um programa Delphi simples (chamado DdlSample) apenas para permitir que você crie novas tabelas, índices e triggers em um exemplo de banco de dados. Você pode usar esse exemplo, enquanto lê as seções a seguir. Como alternativa, você pode digitar os comandos no aplicativo Interactive SQL do Windows.

Tipos de Dados

Após criar o banco de dados, você pode começar a adicionar tabelas nele, com o comando create table. Na criação de uma tabela, você precisa especificar o tipo de dados de cada campo. O SQL inclui vários tipos de dados, embora seja menos rico do que o Paradox e outros bancos de dados locais. A Tabela 11.1 lista os tipos de dados padrão do SQL e alguns outros tipos disponíveis na maioria dos servidores.

TABELA 11.1 Os tipos de dados usados pelo SQL

Tipo de dados	Padrão	Utilização
char, character (n)	Sim	Indica uma string de n caracteres. Servidores ou drivers específicos podem impor um limite de tamanho (32.767 caracteres no caso do InterBase).
int, integer	Sim	Um número inteiro, normalmente quatro bytes, mas dependente de plataforma.
smallint	Sim	Um número inteiro pequeno, geralmente de dois bytes.
float	Sim	Um número de ponto flutuante.
double *(precisão)*	Sim	Um número de ponto flutuante de alta precisão.
numeric (*precisão*, *escala*)	Sim	Um número de ponto flutuante, com a precisão e a escala indicadas.
date	Não	Uma data. A implementação desse tipo de dados varia de um servidor para outro.
blob	Não	Um objeto que contém um grande volume de dados binários (BLOB significa objeto binário grande).
varchar	Não	Uma string de tamanho variável usada para evitar o consumo de espaço de uma string fixa grande.

NOTA No Delphi, a classe TStringField pode distinguir entre os tipos char e varchar, indicando o tipo real em uma propriedade e corrigindo alguns problemas de uso de um char que não é preenchido com espaços à direita na cláusula where de uma instrução de atualização.

Os programadores que estão acostumados ao Paradox e outros mecanismos locais provavelmente notarão a ausência de um tipo lógico, ou booleano, de campos de data e hora (no InterBase, o tipo date contém data e hora) e de um tipo AutoInc, que oferece uma maneira comum de configurar um ID exclusiva em uma tabela. A ausência de um tipo lógico pode gerar alguns problemas, ao se fazer upsize de um aplicativo já existente. Como alternativa, você pode usar um campo smallint com valores 0 e 1 para verdadeiro e falso, ou pode usar um domínio, conforme explicado na próxima seção. Um tipo AutoInc está presente em alguns servidores, como o Microsoft SQL Server, mas não no InterBase. Esse tipo pode ser substituído pelo uso de um gerador, conforme discutido no livro.

Domínios

Os domínios podem ser usados para definir uma espécie de tipo de dados personalizado em um servidor. Um domínio é baseado em um tipo de dados já existente, possivelmente limitado a um subconjunto (como em um tipo de sub-intervalo do Pascal). Um domínio é uma parte útil de uma definição de banco de dados, pois você pode evitar a repetição da mesma verificação de intervalo em vários campos e pode tornar a definição mais legível, simultaneamente.

Como um exemplo simples, se você tiver várias tabelas com um campo de endereço, pode definir um tipo para esse campo e depois usar esse tipo onde um campo de endereço for usado:

```
create domain AddressType as char(30);
```

A sintaxe dessa instrução também permite que você especifique um valor padrão e algumas restrições, com a mesma notação usada ao se criar uma tabela (conforme veremos na próxima seção). Esta é a definição completa de um domínio booleano:

```
create domain boolean as smallint default 0 check (value between 0 and 1);
```

O uso e a atualização de um domínio (com a chamada alter domain) torna particularmente fácil atualizar simultaneamente o padrão e as verificações de todos os campos baseados nesse domínio. Isso é muito mais fácil do que chamar alter table para cada uma das tabelas envolvidas.

Criando Tabelas

No comando create table, após o nome da nova tabela, você indica a definição de um número de colunas (ou campos) e algumas restrições de tabela. Toda coluna tem um tipo de dados e mais alguns parâmetros:

- not null indica que um valor para o campo sempre deve estar presente (esse parâmetro é obrigatório para chaves primárias ou campos com valores exclusivos, conforme descrito a seguir).
- default indica o valor padrão do campo, que pode ser qualquer um dos seguintes: determinado valor constante, null ou user (o nome do usuário que inseriu o registro).
- Uma ou mais restrições podem ser incluídas, opcionalmente com um nome indicado pela palavra-chave constraint. As restrições possíveis são primary key, unique (que indica que todo registro deve ter um valor diferente para esse campo), references (para se referir a um campo de outra tabela) e check (para indicar uma verificação de validade específica).

Aqui está um exemplo do código que você pode usar para criar uma tabela com informações de cliente simples:

```
create table customer (
  cust_no       integer     not null primary key,
  firstname     varchar(30) not null,
  lastname      varchar(30) not null,
  address       varchar(30),
  phone_number  varchar(20)
);
```

Nesse exemplo, usamos not null para a chave primária e para o primeiro e último campos de nome, que não podem ser deixados vazios. As restrições de tabela podem incluir uma chave primária usando vários campos, como em:

```
create table customers (
  cust_no   integer     not null,
  firstname varchar(30) not null,
```

```
  ...
  primary key (cust_no, name)
);
```

NOTA A restrição mais importante é references, que permite definir uma chave estrangeira para um campo. Uma chave estrangeira indica que o valor do campo se refere a uma chave em outra tabela (uma tabela mestra). Essa relação torna a existência do campo na tabela mestra obrigatória. Em outras palavras, você não pode inserir um registro que se refira a um campo mestre inexistente; nem pode destruir esse campo mestre, enquanto outras tabelas estão fazendo referência a ele.

Uma vez que você tenha criado uma tabela, pode removê-la com o comando drop table, uma operação que pode falhar, se a tabela tiver algumas relações restritas com outras tabelas.

Finalmente, você pode usar alter table para modificar a definição da tabela, removendo ou adicionando um ou mais campos e restrições. Entretanto, você não pode modificar o tamanho de um campo (por exemplo, um campo varchar) e ainda manter o conteúdo corrente da tabela. Você deve mover o conteúdo do campo redimensionado para um armazenamento temporário, eliminar o campo, adicionar um novo com o mesmo nome e um tamanho diferente, e finalmente mover os dados de volta.

Índices

O mais importante a ser lembrado a respeito dos índices é que eles não são relevantes para a definição do banco de dados e não se relacionam com o modelo relacional matemático. Um índice deve ser considerado simplesmente como uma sugestão para o SGBDR sobre como acelerar o acesso aos dados.

Na verdade, você sempre pode executar uma consulta indicando a ordem do ordenamento, que estará disponível independentemente dos índices (embora o SGBDR possa gerar um índice temporário). É claro que definir e manter muitos índices poderia exigir muito tempo; se você não souber exatamente como o servidor será afetado, basta deixar o SGBDR criar os índices necessários.

A criação de um índice é baseada no comando create index:

```
create index cust_name on customers (name);
```

Posteriormente, você pode remover o índice, chamando drop index. O InterBase também permite que você use o comando alter index para desativar um índice temporariamente (com o parâmetro inactive) e reativá-lo (com o parâmetro active).

Visões

Além de criar tabelas, o banco de dados permite que você defina visões, ou vistas (views), de uma tabela. Uma visão é definida usando-se uma instrução select e permite criar tabelas virtuais persistentes, mapeadas em tabelas físicas. No Delphi, as visões têm exatamente a mesma aparência das tabelas.

As visões são uma maneira prática de acessar muitas vezes o resultado de uma junção, mas eles também permitem que você limite os dados que usuários específicos podem ver (restringindo o acesso a dados sigilosos). Quando a instrução select que define uma visão é simples, a visão também pode ser atualizada, atualizando na verdade as tabelas físicas que estão por trás dela; caso contrário, se a instrução select for complexa, o visão será somente de leitura.

Migrando Dados Já Existentes

Existem duas alternativas para definir um banco de dados escrevendo manualmente as instruções de DDL. Uma opção é usar uma ferramenta CASE para projetar o banco de dados e permitir que ele gere o código de DDL. A outra é portar um banco de dados já existente de uma plataforma para outra, possivelmente de um banco de dados local para um servidor SQL. A versão Enterprise do Delphi inclui uma ferramenta para automatizar esse processo, o Data Pump Wizard.

O objetivo dessa ferramenta é extrair a estrutura de um banco de dados e recriá-la para uma plataforma diferente. Antes de iniciar o Data Pump, você deve usar o BDE Administrator para criar um apelido (alias) para o banco de dados que deseja gerar. Usar o Data Pump é muito simples: você seleciona o apelido de origem e o apelido de destino; em seguida, você seleciona as tabelas a serem movidas.

Quando você selecionar uma tabela (por exemplo, a tabela EMPLOYEE) e clicar em Next, o Data Pump Wizard verificará se a operação de "upsize" é possível. Após alguns segundos, o assistente apresentará uma lista das tabelas e permitirá que você modifique algumas opções.

Se a conversão de campo não for simples, o Data Pump mostrará a mensagem "Has Problems" ou "Modified". Após modificar as opções, se necessário, você pode clicar no botão Upsize para realizar a conversão real. Selecionando um campo, você pode verificar como o assistente pretende transformá-lo; então, clicando no botão Modify Table Name ou Field Mapping Information For Selected Item, você pode mudar a definição.

Uma alternativa ao uso do Data Pump Wizard (disponível apenas no Delphi Enterprise) é o componente BatchMove, que realiza uma conversão padrão das tabelas e não pode ser otimizado. Finalmente, você pode simplesmente usar o Database Desktop, criar uma nova tabela no servidor e clicar no botão Borrow Struct para extrair a definição de tabela de uma tabela local já existente.

SQL: a Linguagem de Manipulação de Dados

Os comandos SQL dentro da Data Manipulation Language são comumente usados pelos programadores; portanto, os descreveremos com mais detalhes. Existem quatro comandos principais: select, insert, update e delete. Todos esses comandos podem ser ativados usando-se um componente Query, mas apenas select retorna um conjunto de resultados. Para os outros, você deve abrir a consulta usando o método ExecSQL, em vez de Open (ou a propriedade Active).

Select

A instrução select é o comando SQL mais comum e conhecido; ela é usada para extrair dados de uma ou mais tabelas (ou visões) de um banco de dados. Em sua forma mais simples, o comando é:

```
select <campos> from <tabela>
```

Na seção *<campos>*, você pode: indicar um ou mais campos da tabela, separados por vírgulas; usar o símbolo * para indicar todos os campos da tabela simultaneamente; ou até especificar uma operação para aplicar em um ou mais campos. Aqui está um exemplo mais complexo:

```
select upper(name), (lastname || "," || firstname) as fullname
from customers
```

Nesse código, upper é uma função de servidor que converte todos os caracteres para letras maiúsculas, o símbolo de encadeamento duplo ("double-pipe", ||) é o operador de encadeamento de strings e a palavra-chave opcional as fornece um novo nome para a expressão global que envolve o nome e o sobrenome.

Adicionando uma cláusula where, você pode usar a instrução select para especificar quais registros vair recuperar, assim como em quais campos está interessado:

```
select *
from customers
where cust_no = 100
```

Esse comando seleciona um único registro, o correspondente ao cliente cujo número de ID é 100. A cláusula where é seguida de um ou mais critérios de seleção, os quais podem ser unidos usando-se os operadores and, or e not. Aqui está um exemplo:

```
select *
from customers
where cust_no=100 or cust_no=200
```

Os critérios de seleção podem conter funções disponíveis no servidor e usar operadores padrão, incluindo +, -, >, <, =, < >, > = e < =. Também existem alguns outros operadores SQL especiais:

`is null`	Testa se o valor do campo está definido.
`in <lista>`	Retorna verdadeiro se o valor estiver incluído em uma lista após o operador.
`between <mínimo> and <máximo>`	Indica se o valor está incluído no intervalo.

Aqui está um exemplo desses operadores:

```
select *
from customers
where address is not null and cust_no between 100 and 150
```

Outro operador poderoso, usado para realizar correspondência de padrão em strings, é like. Por exemplo, se você quiser procurar todos os nomes que começam com a letra B, pode usar a seguinte instrução:

```
select *
from employee
where last_name like "B%"
```

O símbolo % indica qualquer combinação de caracteres e também pode ser usado no meio de uma string. Por exemplo, a instrução a seguir procura todos os nomes que começam com B e terminam com n:

```
select *
from employee
where upper(last_name) like "B%N"
```

O uso de upper faz com que a busca não diferencie entre maiúsculas e minúsculas e é exigido porque o operador like realiza uma correspondência que leva em consideração letras maiúsculas e minúsculas. Uma alternativa a like é o uso dos operadores containing e starting with. Usar like em um campo indexado com o InterBase pode produzir uma busca muito lenta, pois o servidor nem sempre usará o índice. Se você estiver procurando uma correspondência na parte inicial de uma string, é melhor usar a expressão starting with, que permite o uso do índice e é muito mais rápida.

Outra opção é ordenar as informações retornadas pela instrução select, especificando uma cláusula order by, usando um ou mais dos campos selecionados:

```
select *
from employee
order by lastname
```

Os operadores asc e desc podem ser usados para ordenação ascendente e descendente; o padrão é ascendente.

Uma variação importante do comando select é dado pela cláusula distinct, que remove as entradas duplicadas do conjunto de resultados. Por exemplo, você pode ver todas as cidades onde tem clientes, com a seguinte expressão:

```
select distinct city
from customer
```

O comando select também pode ser usado para extrair valores agregados, calculados por funções padrão:

`avg`	Calcula o valor médio de uma coluna do conjunto de resultados (funciona apenas em campos numéricos).
`count`	Calcula o número de elementos no conjunto de resultados; isto é, o número de elementos que satisfaz determinada condição.
`max e min`	Calculam os valores mais alto e mais baixo de uma coluna no conjunto de resultados.
`sum`	Calcula o total dos valores de uma coluna do conjunto de resultados (funciona apenas em campos numéricos).

Essas funções são aplicadas no conjunto de resultados, normalmente em uma coluna específica, excluindo os valores nulos. Esta instrução calcula o salário médio:

```
select avg(salary)
from employee
```

Outra cláusula importante é group by, que permite agregar os elementos do conjunto de resultados de acordo com algum critério, antes de calcular valores agregados com as funções listadas anteriormente. Por exemplo, talvez você queira determinar o salário máximo e médio dos funcionários de cada departamento:

```
select max (salary), avg (salary), department
from employee
group by department
```

Note que todos os campos não calculados devem aparecer na cláusula group by. O seguinte não é válido:

```
select max (salary), lastname, department
from employee
group by department
```

DICA Quando você extrai valores agregados, é melhor usar um apelido para o campo de resultado, com a palavra-chave as. Isso torna mais fácil fazer referência ao valor resultantes em seu código Delphi.

Os valores agregados também podem ser usados para determinar os registros no conjunto de resultados. As funções agregadas não podem ser usadas na cláusula where, mas elas são colocadas em

uma seção having específica. A instrução a seguir retorna o salário mais alto de cada departamento, mas apenas se o valor estiver acima de 40.000:

```
select max(salary) as maxsal, department
from employee
group by department
having max(salary) > 40000
```

Outra possibilidade interessante é aninhar uma instrução select dentro de outra, formando uma sub-consulta. Aqui está um exemplo, usado para retornar o funcionário (ou funcionários) de salário mais alto:

```
select firstname, lastname
from employee
where salary = (select max(salary) from employee)
```

Não poderíamos ter escrito esse código com uma única instrução, pois adicionar o nome no resultado da consulta teria implicado em adicioná-lo também na seção group by.

Junções Internas e Externas

Até agora, nossos exemplos de instruções select trabalharam em tabelas individuais — uma séria limitação para um banco de dados relacional. A operação de combinar dados de várias tabelas-fonte é chamada de junção (join) de tabelas. O padrão SQL suporta dois tipos de junções, chamadas interna (inner join) e externa (outer join).

Uma junção interna pode ser escrita diretamente na cláusula where:

```
select *
from <tabela1>, <tabela2>
where <tabela1.campo_chave>=<tabela2.chave_externa>
```

Esse é um exemplo típico de junção interna usada para extrair todos os campos de cada tabela envolvida. Uma junção interna é útil para tabelas com uma relação de um-para-um (um registro de uma tabela correspondente a apenas um registro da segunda tabela). Na verdade, a sintaxe padrão deve ser a seguinte, embora as duas estratégias normalmente gerem o mesmo efeito:

```
select *
from <tabela1> left join <tabela2>
on <tabela1.campo_chave>=<tabela2.chave_externa>
```

Uma junção externa, em vez disso, pode ser especificamente solicitada com a instrução:

```
select *
from <tabela1> left outer join <tabela2>
on <tabela1.campo_chave>=<tabela2.campo_chave>
```

A principal diferença em relação a uma junção interna é que as linhas selecionadas de uma junção externa não considerarão os campos nulos da segunda tabela. Existem outros tipos de junções, incluindo os seguintes: a autojunção (self-join), em que uma tabela se une a ela mesma; a multijunção (multi-join), que envolve mais de duas tabelas; e o produto cartesiano, uma junção sem nenhuma cláusula where, que combina cada linha de uma tabela a cada linha da segunda, normalmente produzindo um conjunto de resultados enorme. A junção interna certamente é a forma mais comum.

Insert

O comando insert é usado para adicionar novas linhas em uma tabela ou a uma visão atualizável. Quando você insere um novo registro em um controle DBGrid conectado a uma tabela de servidor SQL, o BDE gera um comando insert e o envia para o servidor. Além desse uso implícito, existem vários casos em que você desejará escrever chamadas de insert SQL explícitas (incluindo o uso de atualizações colocadas em cache, que discutiremos posteriormente neste capítulo).

A não ser que adicione um valor para cada campo, você deve listar os nomes dos campos que está realmente fornecendo, como no código a seguir:

```
insert into employee (empno, lastname, firstname, salary)
values (0,"brown", "john", 10000)
```

Você também pode inserir em uma tabela o conjunto de resultados de uma instrução select (se os campos da tabela de destino tiverem a mesma estrutura dos campos selecionados), com esta sintaxe:

```
insert into <tabela> <instrução select>
```

Update

O comando update modifica um ou mais registros de uma tabela ou visão. O Delphi gera uma chamada de update sempre que você edita dados com controles visuais conectados a uma tabela ou a uma consulta ativa em um servidor SQL. Novamente, também existem casos onde você desejará usar a instrução update diretamente.

Em uma instrução update, você pode indicar qual registro vai modificar, usando uma cláusula where semelhante a de uma instrução select. Por exemplo, você pode mudar o salário de um funcionário específico, com esta chamada:

```
update employee
set salary = 30000
where emp_id = 100
```

Alerta

Uma única instrução update pode atualizar todos os registros que satisfazem determinada condição. Uma cláusula where incorreta poderia atualizar muitos registros, sem se querer isso, e nenhuma mensagem seria apresentada.

A cláusula set pode indicar vários campos, separados por vírgulas e pode usar os valores correntes dos campos para calcular os novos valores. Por exemplo, a instrução a seguir dá um bom aumento para os funcionários contratados antes de 1 de janeiro de 1990:

```
update employee
set salary = salary * 1.20
where hiredate < "01-01-1990"
```

Delete

O comando delete é igualmente simples (embora seu uso errôneo possa ser muito perigoso). Novamente, você geralmente remove registros usando um componente visual, mas também pode executar um comando SQL como o seguinte:

```
delete from employee
where empid = 120
```

Você simplesmente indica uma condição identificando os registros a serem excluídos. Se você executar esse comando SQL com um componente Query (chamando ExecSQL), poderá então usar a propriedade RowsAffected para ver quantos registros foram excluídos. O mesmo se aplica aos comandos update.

Construindo Consultas Visualmente no Delphi (com o SQL Builder)

Como vimos, o SQL tem muitos comandos, particularmente em relação às instruções select. E, na verdade, ainda não vimos todos eles! Enquanto os comandos da DDL são geralmente usados por um administrador de banco de dados ou apenas para a definição inicial do banco de dados, os comandos da DML são normalmente usados no trabalho diário de programação com Delphi.

Para ajudar no desenvolvimento de instruções SQL corretas, o Delphi Enterprise inclui uma ferramenta chamada SQL Builder, a qual infelizmente só pode ser usada através de uma conexão BDE (mesmo que você possa copiar posteriormente a consulta e usá-la, por exemplo, com um conjunto de dados dbExpress). Você a ativa facilmente, dando um clique com o botão direito do mouse em um componente Query.

Usar o SQL Builder é muito simples. Você escolhe o banco de dados em que deseja trabalhar e, em seguida, seleciona uma ou mais tabelas, colocando-as na área de trabalho. Após selecionar os parâmetros corretos, como explicado a seguir, você pode usar o comando Query | Run Query (ou F9) para ver o resultado da consulta, ou o comando Query | Show SQL (F7), para ver o código-fonte da instrução select gerada.

Nas tabelas selecionadas, você pode simplesmente marcar os campos que deseja ver no conjunto de resultados. A caixa de seleção ao lado do nome da tabela seleciona todos os seus campos. Mas o real poder do SQL Builder reside em dois recursos. Primeiro, você pode arrastar um campo de uma tabela para outra, para uni-las.

O outro recurso poderoso é o Query Notebook, o controle de várias páginas na parte inferior da janela do SQL Builder. Aqui está uma descrição curta de cada uma das páginas:

A página Criteria indica os critérios de seleção da cláusula where. Selecionando um dos campos da tabela de resultado, você pode indicar uma comparação em relação a um valor fixo ou outro campo, e pode-se usar like, is null, between e outros operadores. Usando o menu local da grade presente nessa página, você também pode ativar o operador exist ou uma expressão SQL inteira. Essa página permite combinar várias condições com os operadores and, or e not, mas ela não permite especificar uma precedência entre esses operadores através da adição de parênteses.

A página Selection lista todos os campos do conjunto de resultados e permite que você forneça um apelido a eles. Com o menu local, você também pode introduzir funções agregadas (sum, count etc.). Finalmente, a caixa de seleção na parte superior esquerda indica a condição distinct.

A página Grouping corresponde à cláusula group by. O SQL Builder agrupa automaticamente todos os campos usados nas funções agregadas, conforme exigido pelo padrão SQL.

A página Group Criteria corresponde a uma cláusula having, que está disponível em conjunto com as funções agregadas. As operações são semelhantes àquelas da página Selection e são ativadas usando-se o menu local.

A página Sorting corresponde à cláusula order by. Você simplesmente seleciona o campo que deseja ordenar e, em seguida, seleciona a ordenação ascendente ou descendente.

A página Joins é a última, mas provavelmente a mais poderosa, pois ela permite definir condições de junção, além do arrasto simples de um campo de uma tabela para outra na área de trabalho. Essa página permite otimizar o pedido de junção, indicando-se seu tipo (INNER ou OUTER) e selecionando-se condições diferentes do teste de igualdade.

Para entender melhor como se usa o SQL Builder, podemos construir um exemplo baseado no exemplo de banco de dados do InterBase instalado pelo Delphi (e correspondente ao apelido LocalIB). O exemplo está no diretório SqlBuilder e seu formulário tem um componente Query, um DataSource e um DBGrid, conectados como sempre. A propriedade DatabaseName do componente Query é configurada como IBLocal e dar um clique com o botão direito do mouse no componente ativa o SQL Builder.

Queremos criar uma consulta incluindo o nome e o sobrenome, departamento, título e salário de cada funcionário. Essa operação exige duas junções. Escolha as tabelas Employee, Department e Job. Clique no campo Dep_No da tabela Department e arraste o cursor sobre o campo Dep_No da tabela Employee. Analogamente, conecte a tabela Job à tabela Employee, usando os três campos, Job_Code, Job_Grade e Job_Country.

Após criar as junções, selecione os campos que você deseja ver no conjunto de resultados: First_Name, Last_Name e Salary da tabela Employee; Department da tabela Department; e Job_Title da tabela Job. Finalmente, vá para a página Sorting do Query notebook e selecione Department.Department da lista Output Fields, para ordenar o conjunto de resultados por departamento.

Deve ser gerado o código SQL a seguir:

```
select employee.first_name, employee.last_name, department.department,
  job.job_title, employee.salary
from employee employee
  inner join department department
  on (department.dept_no = employee.dept_no)
  inner join job job
  on (job.job_code = employee.job_code)
  and (job.job_grid = employee.job_grid)
  and (job.job_country = employee.job_country)
order by department.department
```

Poderíamos adicionar uma cláusula where extra para escolher apenas o funcionário com um salário alto. Basta ir para a página Selection, selecionar o campo Employee.Salary, ir para a coluna Operator >= e digitar o valor 100.000. Executando a consulta, você verá um número limitado de registros e observando o código-fonte SQL, verá a instrução extra:

```
where employee.salary >= 100000
```

Finalmente, note que é possível exportar e importar o código SQL de um arquivo de texto simples. Simplesmente fechando o SQL Builder, você também salvará o texto da consulta na propriedade SQL do componente Query relacionado.