Padrão SQL e sua Evolução

Allyn Grey de Almeida Lima
Rua Sud Menucci, 65 – apto 112-A
Jardim Aurélia
13033-055 – Campinas – São Paulo – Brasil
e-mail: allyn@ccuec.unicamp.br

Resumo

SQL é a linguagem comercial de banco de dados mais utilizada no mercado. Devido à sua popularização, organizações como ANSI e ISO resolveram padronizá-la. A padronização iniciou-se em 1986 com a versão SQL-86 e evoluiu até a versão SQL:2003, em fase de publicação.
O objetivo deste trabalho é mostrar alguns novos recursos adicionados à linguagem nestes anos, em especial nas versões mais recentes (SQL:1999 e SQL:2003), além de discutir o tipo de suporte dado a estes recursos e a integração entre as organizações de padronização e os vendedores de bancos de dados.

1. Introdução

Segundo [Ramakrishnan,2003], *Structured Query Language* (SQL) foi originalmente chamada de SEQUEL e desenvolvida pela IBM como parte do Sistema R no início dos anos 70, quando os bancos de dados relacionais estavam sendo desenvolvidos e linguagens foram criadas para manipulá-los. Ao contrário de linguagens procedurais como C e Pascal, SQL é uma linguagem de programação declarativa, ou seja, descreve o problema ao invés da solução, especificando o que deve ser feito e não como. [Wikipedia].

Com o sucesso da linguagem, o Instituto Americano Nacional de Padrões (ANSI) padronizou as implementações do produto em 1986, e no ano seguinte o padrão foi adotado pela Organização Internacional de Padronização (ISO). A partir deste ano, outras organizações foram envolvidas e o padrão foi evoluindo até chegar ao recente SQL:2003.

Porém, existem variações da linguagem (chamadas de extensões) que foram incorporadas para complementar as capacidades da linguagem. Tais variações podem comprometer a portabilidade, pois para obtê-la, o padrão deve ser rigorosamente seguido, não utilizando algumas facilidades incorporadas nas extensões. Neste contexto, surge uma discussão entre a importância do padrão e o mercado exigente que pede soluções rápidas e fáceis.

O artigo abrangerá a evolução do padrão SQL, os principais recursos adicionados nas versões mais recentes (que vão desde novos tipos de dados até partes novas como SQL/XML), e quais deles são suportados por alguns gerenciadores de bancos de dados. Ele não será um guia SQL e nem compreenderá muitos detalhes da linguagem. Para isto, veja [Ramakrishnan,2003]. É um trabalho de curso para a disciplina Banco de Dados I do Instituto de Computação da Universidade Estadual de Campinas (Unicamp), sob orientação do Professor Dr. Geovane Cayres Magalhães.

Os exemplos utilizados no decorrer do trabalho foram retirados de outros artigos [Melton,1999], [Melton,2003] e [Guimarães,2003] e adaptados ao contexto de uma vídeo locadora.

2. Evolução da Linguagem

Segundo [Melton,1999], as duas organizações atualmente envolvidas na padronização do SQL são o comitê americano ANSI e o comitê internacional ISO. Mais especificamente, a

comunidade internacional trabalha através da ISO/IEC/JTC1 (*Joint Technical Committee 1*), responsável por desenvolver e manter padrões relacionados à Tecnologia da Informação. Dentro da JTC1, um subcomitê denominado SC32 – *Data Management and Interchange* – foi formado para reunir vários padrões relacionados à banco de dados e metadados. SC32, por sua vez, é formado por um número de *Working Groups* que realmente fazem o trabalho técnico onde o WG3 (*Database Languages*) é responsável pelo padrão SQL. Nos Estados Unidos, os padrões relacionados à banco de dados, incluindo o SQL, são tratados pela ANSI NCITS H2 (*National Committee for Information Technology Standards – Technical Comittee on Database*), na qual IBM, Microsoft e Oracle têm seus representantes.

O primeiro padrão (SQL-86) foi definido em 1986 pelo ANSI e no ano seguinte a ISO adotou o SQL como padrão. Esta primeira geração consistia basicamente na implementação da IBM. Em 1989 surgiu uma nova versão (SQL-89) em que foram adicionados recursos importantes como chave primária, chave estrangeira e valores nulos.

No ano em o SQL-89 foi publicado, a H2 tornou-se muito ativa nas propostas escritas para a especificação que gerou o SQL-92, o padrão até hoje mais seguido pelos gerenciadores de bancos de dados. Alterações significativas como domínios, tabelas temporárias, novos tipos de junção (*left, right e natural join*), expressões nomeadas (cláusula AS), valores únicos (*unique)*, expressões na cláusula *from*, entre outras, foram incluídas neste padrão.

Mas desde 1993, foi sendo desenvolvido um trabalho para atualizar o padrão SQL de modo que atendesse às características das últimas versões lançadas pelos bancos de dados comerciais. Como resultado, surgiu o padrão SQL:1999 (SQL3). Esta versão foi muito mais ambiciosa introduzindo novos tipos de dados, consultas recursivas, gatilhos e o conceito de orientação a objetos.

A versão mais recente, ainda em fase de publicação, é o SQL:2003 (ou SQL:200n) que traz recursos para XML, instrução *merge* e colunas identidade. Estas versões mais recentes serão comentadas com maiores detalhes neste artigo.

2.1. Importância do Padrão

Algumas implementações existentes no mercado adotam instruções SQL diferentes do padrão, gerando extensões. Uma instrução SQL raramente pode ser transportada entre sistemas de banco de dados sem maiores modificações. Alguns acreditam que a incompatibilidade é intencional para que clientes dependam de um banco de dados específico [Wikipedia,2004].

Então, como garantir que as instruções SQL de uma aplicação funcionarão da mesma forma se ocorrer migração de banco de dados? As instruções básicas não são críticas, mas as avançadas sim. Se esta garantia for desejada, é necessário seguir o padrão rigorosamente, o que pode comprometer facilidade e desempenho.

Os vendedores de bancos de dados criam extensões para melhorar o desempenho e tornar mais fácil a manipulação dos dados. Eles se preocupam em: criar soluções rápidas para atender a demanda do mercado, tornar a implementação mais atrativa e ter vantagens com relação à concorrência.

Porém, atualmente, os vendedores estão mais preocupados com a padronização. As novas versões dos produtos já estão suportando grande parte dos recursos. Há também o inverso, soluções extremamente eficientes criadas pelos vendedores foram padronizadas (como por exemplo, o conceito de papéis).

3. SQL:1999

Segundo [Melton,1999], este suporte foi planejado para ser lançado em 1996, mas não aconteceu, porque ele não é simplesmente um SQL-92 acrescido de orientação a objetos, envolve recursos adicionais.

O uso do nome SQL:1999, ao invés de SQL-99, começou devido à preocupação com a próxima geração que não poderia ser SQL-02, pois causaria confusão com SQL2 (mais conhecido como SQL-92).

Após o SQL-92, as organizações começaram a publicar o padrão SQL como um documento base (*foundation*) e várias partes independentes. O SQL:1999 foi dividido em cinco partes [Oracle,1999].

Parte 1: SQL/Framework: descrição não técnica de como o documento está estruturado;

Parte 2: SQL/Foundation: a base da especificação;

Parte 3: SQL/CLI (*Call Level Interface*): especifica uma interface para o SQL que pode ser usada por um programa de aplicação. Implementação mais conhecida é o ODBC;

Parte 4: SQL/PSM (*Persistent Stored Modules*): especifica *stored procedures*, incluindo integridade computacional;

Parte 5: SQL/Bindings: especifica como comandos podem ser embutidos em programas hospedeiros e como podem ser preparados para execução.

O foco será na parte SQL/Foundation, que pode ser grosseiramente dividida em "recursos relacionais" (relacionadas à parte tradicional do SQL) e "recursos de orientação a objetos".

3.1. Novos tipos de dados

- BOOLEAN: utilizado para especificar valores como verdadeiro, falso e desconhecido.

- LOB (*Large Object*): utilizado para representar grande volume de dados, como áudio e dados de imagem. Possui dois subtipos: BLOB (*Binary Large Object*) e CLOB (*Character Large Object*).

LOB tem restrições quanto a seu uso, como por exemplo, atributos deste tipo não podem ser usados como chave primária, estrangeira ou em cláusulas como GROUP BY e ORDER BY. Este tipo de dados requer grande quantidade de armazenamento, por isso o SQL:1999 provê localizadores (*locators*) para gerenciá-los e fazer com que o valor seja transferido aos poucos para a aplicação. Estes localizadores podem ser utilizados para recuperar dados de outros tipo (array, UDT).

O tamanho do LOB pode ser especificado na definição do atributo. O exemplo 1 mostra a sua utilização.

```
Exemplo 1:

CREATE TABLE Filmes (codFilme VARCHAR (10), codGenero VARCHAR(2),
codCategoria VARCHAR(2), nome VARCHAR(50), duracao TIME,
anoLancamento INTEGER, sinopse CLOB(15K), foto BLOB(1M),...)
```

- ARRAY: estrutura de dados composta por um número definido de elementos do mesmo tipo. Útil para atributos multivalorados, conforme exemplo 2.

```
Exemplo 2:

telefone VARCHAR(15) ARRAY[2]

Para inserir dados:
INSERT INTO Clientes (telefone) VALUES (array['32198765',
'91231234'])

Para acessar dados:

telefone[1], retorna como resultado: '91231234'
```

Há uma crítica com relação a este tipo afirmando que ele viola a Primeira Forma Normal: "Toda relação está na Primeira Forma Normal se não possui atributos multivalorados nem relações aninhadas [Guimarães, 2003]". Porém, existem argumentos afirmando que este tipo existe apenas para armazenar informações que podem ser decompostas. Mas alguns novos recursos, como orientação a objetos, abandonaram a Primeira Forma por ser muito restrita.

- ROW: formado por uma seqüência de atributos, permitindo armazenar valores estruturados em uma única coluna. O exemplo 3 mostra os atributos nome e endereço sendo deste tipo.

```
Exemplo 3:

CREATE TABLE Clientes (codCliente VARCHAR(10),
nome ROW (primeiro VARCHAR(30), sobrenome VARCHAR(30)),
endereco ROW (rua VARCHAR(50), numero INTEGER, bairro VARCHAR(20),
cidade VARCHAR (30), estado VARCHAR(2)),email VARCHAR(40),...)

Para acessar dados:

SELECT C.endereco.rua FROM Clientes C WHERE C.codCliente = 1
```

- Tipos distintos definidos pelo usuário (UDTs): consiste em criar um novo tipo baseado em outro já suportado. No exemplo 4, a aplicação necessita do tipo “Real” ao invés do tipo inteiro.

```
Exemplo 4:

CREATE TYPE Real AS DECIMAL(8,2)

CREATE TABLE Categoria_Preco (valorDiaria  Real)

Para acessar os dados (como 3 é um número inteiro, é necessário
fazer um cast para comparar com o tipo Real):

SELECT * FROM Categoria_Preco WHERE valorDiaria > Real(3)
```

3.2. Novos predicados

- SIMILAR TO: o operador *like* é muito restrito comparado ao poder de expressões regulares utilizadas no sistema operacional Unix. Com isso, foi acrescentado o operador *similar to* para comparações mais sofisticadas, conforme exemplo 5.

```
Exemplo 5:

SELECT P.desconto FROM Promocoes P WHERE P.diaSemana SIMILAR TO
'((Segunda|Terça|Quarta|Quinta|Sexta)-feira)|Sábado|Domingo'
```

- DISTINCT FROM: tem semântica semelhante ao predicado *unique* (retorna verdadeiro se a subconsulta não contém linhas duplicadas), mas considera dois valores nulos como não diferentes. O *not distinct from* não implica necessariamente em igualdade. O exemplo 6 compara o conteúdo dos atributos codFilme, dataLocacao e dataDevolucao da tabela Locacao.

```
Exemplo 6:

('1','23/05/2004',null) IS DISTINCT FROM ('1','23/05/2004',null)
retorna falso
```

3.3. Novas semânticas

- Consultas recursivas: existem consultas que necessitam de uma estrutura em árvore e não podem ser expressas apenas com os recursos do SQL-92.

Um exemplo clássico deste tipo de consulta é o problema de processador de lista de materiais (*bill of material processor*) que procura relacionar todos os componentes e sub-componentes de um conjunto complexo, como por exemplo, um avião. A geração de tais listas, em

diferentes níveis de hierarquia, é extremamente importante para planejamento de produção, cálculo de custos, gerenciamento de estoques.

Foi adicionado no SQL:1999 uma nova cláusula para tratar consultas recursivas. Para entender o seu funcionamento será utilizado o exemplo 7, retirado do livro [Guimarães,2003]. Este exemplo não está no contexto da vídeo locadora, devido à complexidade de se utilizar recursão.

O exemplo 7 exibe o conteúdo de uma tabela de Vôos contendo a identificação do vôo, companhia aérea, cidade origem e cidade destino (por simplicidade, foram omitidos os horários de saída e chegada). Esta tabela representa um grafo orientado, conforme figura 1, onde os vértices são as cidades e as arestas correspondem a um vôo ou trecho de vôo de uma cidade origem para uma cidade destino. O objetivo da consulta é percorrer o grafo descobrindo várias alternativas para voar de uma cidade para outra.

Exemplo 7:

Tabela 1. Tabela de Vôos

tr_id	cia_aérea	origem	destino
1	UA	SF	DEN
2	UA	SF	DAL
3	UA	DEN	CHI
4	UA	DEN	DAL
5	UA	DAL	NY
6	UA	CHI	NY

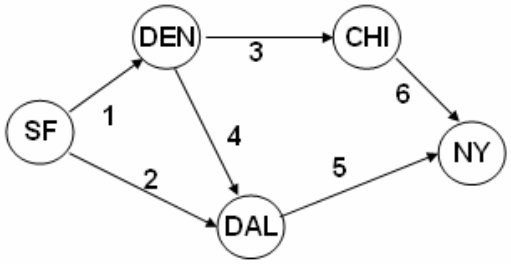

Figura 1. Grafo dos Vôos da companhia 'UA'

Supondo que deseja-se: "exibir todos os possíveis vôos ou trechos de vôos da companhia aérea 'UA', com origem em SF e destino final NY". Esta consulta necessita de recursão e a instrução SQL correspondente é mostrada no exemplo 8.

Exemplo 8:

```
1. WITH RECURSIVE Trechos (tr_id, cia, origem, destino) AS
2. SELECT tr_id, cia_aerea, origem, destino FROM Voos)
3. UNION
4. SELECT Voos.tr_id, Voos.cia_aerea, Voos.origem, Voos.destino
5. FROM Trechos,Voos
6. WHERE Trechos.destino = Voos.origem
7. AND Trechos.cia_aerea = Voos.cia_aerea)
8. SELECT tr_id,destino FROM Trechos
9. WHERE cia = 'UA' AND origem = 'SF'
```

A consulta é composta pela união de duas consultas: consulta base (linha 2) e a parte indutiva (linhas 4 a 7). A busca é feita em profundidade para obter o resultado, mas o padrão SQL possui cláusulas adicionais para a escolha do tipo de busca.

O resultado é exibido no exemplo 9. Observando o resultado, pode-se verificar que, quando a cidade origem = 'DEN', há duas alternativas: rotas 3 e 6 (2ª e 3ª linhas) ou 4 e 5 (4ª e 5ª linhas), da mesma forma que, quando cidade origem = 'SF', pode utilizar rotas 1, 3, 6 ou 2, 5.

```
Exemplo 9:
```

Tabela 2. Resultado da consulta recursiva

tr_id	destino
1	DEN
3	CHI
6	NY
4	DAL
5	NY
2	DAL
5	NY

E ainda, para obter todos os possíveis trechos da companhia 'UA' de qualquer cidade origem, basta omitir a condição "AND origem = 'SF'".

- SAVEPOINTS: são utilizados para dividir uma transação em partes menores, definindo um ponto de retorno quando a transação é abortada. Não terminam a transação, deixam recurso alocado. Qualquer alteração pendente pode ser desfeita até aquele ponto, utilizando ROLLBACK TO SAVEPOINT, como no exemplo 10.

```
Exemplo 10:

UPDATE Categoria_Preco SET valorDiaria = valorDiaria * 1.1
WHERE codCategoria = 1

SAVEPOINT update_ponto

INSERT INTO Genero (codGenero,nome) VALUES (1,'Comédia')

SELECT G.nome FROM Genero G WHERE G.codGenero = 1
(retorna 'Comédia')

ROLLBACK TO UPDATE_PONTO

SELECT G.nome FROM Genero G WHERE G.codGenero = 1
(nenhuma linha é retornada)

COMMIT (limpa os savepoints)
```

3.4. Aumento de Segurança

Um dos recursos padronizados, mas que já era utilizado pelos bancos de dados comerciais, foi o conceito de "papel" (*role*). Ele é utilizado para agrupar vários usuários numa unidade administrativa, a qual se pode aplicar permissões.

As permissões concedidas ou retiradas a um papel também se aplicam automaticamente a todos os membros existentes ou futuros. Ele serve para facilitar a representação de funções desempenhadas por uma classe de funcionários, refletindo como as empresas trabalham.

À medida que indivíduos mudam, só é preciso adicionar ou remover membros do papel. Há também restrições para não permitir excesso de poder a um único usuário (ele pertencer ao papel de gerente de contas e ao papel de auditor ao mesmo tempo). O exemplo 11 mostra a sintaxe.

```
Exemplo 11:

CREATE ROLE Gerente
CREATE ROLE Atendente
GRANT ALL ON Promocoes TO Gerente
GRANT SELECT ON Promocoes TO Atendente
GRANT Gerente TO Julio
GRANT Atendente TO Pedro
```

3.5. Bases de Dados Ativa

Apesar de estar disponível na maioria dos bancos de dados comerciais, o conceito de gatilhos (*triggers*) ainda não fazia parte do padrão SQL. Segundo [Ramakrishnan,2003], gatilhos são procedimentos iniciados automaticamente se mudanças especificadas ocorrerem no banco de dados. Eles são compostos por três partes: evento (ativa o gatilho), condição (testa se o gatilho deveria ser executado) e ação (o que acontece se o gatilho for executado).

São utilizados para manter integridade dos dados, controles de auditoria, estatísticas, disparar ações específicas (enviar e-mail quando estado de um atributo chega num valor crítico) ou disparar ações específicas (emitir aviso quando um filme reservado foi devolvido). Mas este mecanismo deve ser utilizado com cuidado, pois pode gerar gatilhos recursivos.

O uso de gatilhos é um assunto profundo. Para maiores informações consulte [Ramakrishnan,2003] e [Guimarães,2003]. O exemplo 12 mostra do uso de gatilhos para controle estatístico.

Exemplo 12:

```
CREATE TRIGGER controleEstatisticas
/* quantas vezes cada filme foi locado */
AFTER INSERT ON Locação  /* evento */
REFERENCING NEW TABLE AS FilmesLocados
FOR EACH STATEMENT
/* ação será executada uma única vez para todas as tuplas que
foram modificadas pelo comando INSERT */
     INSERT
           INTO Estatisticas (Tabela,Operacao,codFilme,Qtde)
           SELECT 'Locacao','Insert',F.codFilme,count(*)
           FROM FilmesLocados F
```

3.6. Recursos de Orientação a Objetos

O suporte ao gerenciamento de dados orientados a objetos foi adicionado e o recurso principal desta área é a introdução de tipos de dados estruturados definidos pelo usuário (*structured UDT*). Estes tipos contêm as seguintes características:

- Podem conter vários atributos de qualquer tipo SQL (*integer*, *array*) ou tipos estruturados aninhados;
- Atributos são encapsulados e acessados através de funções *get* e *set* ou através de métodos e funções definidos para o tipo de dados estruturado. Métodos e funções podem ser escritos tanto no padrão SQL/PSM quanto em linguagens como C ou Java, se forem suportadas;
- Suportam herança simples. Tipos estruturados podem conter subtipos herdando atributos e métodos, conforme exemplo 13.

Exemplo 13:

```
CREATE TYPE tipo_DVD
UNDER tipo_Itens
AS (codDVD INTEGER, regiao INTEGER, formatoTela VARCHAR(20),
idioma VARCHAR(10) ARRAY[5])
INSTANTIABLE
FINAL
REF (CodDVD)
INSTANCE METHOD alteraRegiao (regiaoAux INTEGER)
RETURNS INTEGER
```

O novo tipo tipo_DVD é um subtipo do tipo_Itens que descrevem os itens em geral existentes na loja (DVD,VHS, jogos, bebidas), incluindo atributos em comum como o nome. Porém, o subtipo DVD tem características diferentes de uma fita VHS, por exemplo, como região e formato de tela.

Este subtipo não pode ter outros subtipos (FINAL) e ele pode ser instanciável (INSTANTIABLE). Se NOT INSTANCIABLE fosse especificado, este tipo seria abstrato.

A palavra REF é usada para permitir referenciar uma instância deste tipo através do atributo CodDVD.

O tipo tipo_DVD pode ser usado como atributo ou na criação de tabelas denominadas "tipadas", conforme exemplo 14.

```
Exemplo 14:

CREATE TABLE Itens_Alugaveis (CodItem VARCHAR(10),...,tipo
tipo_DVD)

Ou

CREATE TABLE DVDs OF tipo_DVD
```

Cada coluna da tabela tipada DVDs corresponde a um atributo definido em tipo_DVD e os métodos definidos para este tipo (como alteraRegiao) podem operar sobre as linhas desta tabela. Para acessar um atributo do tipo estruturado, deve-se utilizar a sintaxe do exemplo 15.

```
Exemplo 15:

SELECT D.nome,D.tipo->formatoTela FROM DVDs WHERE D.codItem = 1
```

4. SQL:2003

O padrão SQL:2003, ou melhor chamado de SQL:200n, está sendo publicado, com revisões em todas as partes do SQL:1999. Ainda não existem muitos artigos referentes a este padrão, portanto apenas alguns recursos serão comentados aqui. Ele foi estruturado em 9 partes:

Parte 1: SQL/Framework

Parte 2: SQL/Foundation

Parte 3: SQL/CLI

Parte 4: SQL/PSM

Parte 9: SQL/MED (Management of External Data): especifica como SQL pode ser usado para gerenciamento de dados externos ao banco de dados;

Parte 10: SQL/OLB (Object Language Binding): interface para linguagem orientada a objetos;

Parte 11: SQL/Schemata: informações e definições de esquemas;

Parte 13: SQL/JRT (Java Routines and Types): rotinas e tipos usando Java;

Parte 14: SQL/XML: especificação relacionada à linguagem XML.

Algumas partes já tinham sido definidas desde do SQL:1999 e as partes 5, 6, 7, 8, and 12 não existem mais.

O foco será nas partes SQL/Foundation e SQL/XML.

4.1. Novos tipos de dados

O padrão continua com os tipos existentes no SQL:1999 (com exceção dos tipos BIT e VARYING BIT não mencionados neste artigo) e acrescentou três novos tipos:

- BIGINT: parecido com os tipos *smallint* e *integer*, porém com maior precisão. Todas as operações do *integer* são suportadas por este tipo. Inteiro de 8 bytes com sinal.

- MULTISET: parecido com o tipo array, é uma coleção de elementos do mesmo tipo não ordenados e que podem ter valores duplicados.
  Valores deste tipo podem ser criados numerando os elementos individualmente ou através de uma consulta como no exemplo 16.

```
Exemplo 16:

MULTISET ['Português','Espanhol']

MULTISET(SELECT linguas FROM Idiomas)
```

Este tipo suporta operações de conversão, remover duplicidade, retornar número de elementos, além de novas operações de agregação: COLLECT (cria um *multiset* de um conjunto de elementos), FUSION (cria um *multiset* contendo a união de todas as linhas de um grupo) e INTERSECTION (cria um *multiset* contendo a intersecção – elementos em comum – de todas as linhas de um grupo). O exemplo 17 mostra o uso dessas operações.

```
Exemplo 17:
```

Tabela 3. Tabela de DVDs

codDVD	idioma
1	MULTISET[‘Português’,’Espanhol’]
2	MULTISET[‘Português’,’Inglês’,’Espanhol’]
3	MULTISET[‘Português’ ,‘Francês’]

```
SELECT COLLECT(codDVD) AS todos_dvds, FUSION(idioma) as
todos_idiomas, INTERSECTION (idioma) as idiomas_comuns from DVDs
```

Tabela 4. Resultado da consulta

todos_dvds	todos_idiomas	idiomas_comuns
MULTISET [‘1’,’2’,’3]	MULTISET[‘Português’, ‘Espanhol’, Português, ’Inglês’, ’Espanhol’, ‘Português’,’Francês’]	MULTISET[‘Português’]

- tipo XML: pertence a parte SQL/XML que será comentada posteriormente.

4.2. CREATE TABLE LIKE

Já em SQL:1999 um usuário poderia criar uma nova tabela com a mesma estrutura de uma ou de várias tabelas já existentes, utilizando a cláusula CREATE TABLE LIKE. Entretanto a cópia era restrita, informações adicionais sobre as colunas, como valores padrão, colunas identidade (explicadas neste artigo mais tarde) não eram copiadas.

Um ponto importante desta instrução é que não há dependência entre a nova tabela e as tabelas já existentes que serviram como base. O exemplo 18 mostra a sintaxe.

```
Exemplo 18:

CREATE TABLE Filmes (codFilme INTEGER GENERATED ALWAYS
AS IDENTITY (START WITH 1, INCREMENT BY 1),sinopse CLOB(15K) NOT
NULL, situacao VARCHAR(12) DEFAULT 'disponível')

CREATE TABLE FilmesTemp (LIKE Filmes, codFranquia VARCHAR(2))

CREATE TABLE Filmes_Temp2 (
LIKE Filmes
```

```
INCLUDING COLUMN DEFAULTS
INCLUDING IDENTITY,codFilial VARCHAR(2));
```

A tabela Filmes contém informações sobre coluna identidade e valor padrão. Ao criar Filmes_Temp estas informações seriam perdidas.

No SQL:2003 foi adicionado novas opções para copiar estas informações. No exemplo, para criar a tabela Filmes_Temp2, foi colocado o INCLUDING. O NOT NULL, mesmo não especificado, é copiado também.

4.3. CREATE TABLE AS

Há circunstâncias em que é necessário copiar apenas um subconjunto das estruturas de uma ou mais tabelas existentes. Com isso, surgiu a instrução CREATE TABLE AS que permite criar tabela a partir da estrutura de uma consulta.

Não existe dependência entre a tabela criada e a consulta base. Após a nova tabela ter sido preenchida, atualizações nas tabelas da consulta base não refletirão na nova tabela. Veja no exemplo 19 a criação de uma tabela de filmes antigos que ainda são locados.

```
Exemplo 19:

CREATE TABLE FilmesAntigosLocados (nomeFilme,
anoLancamento,dataLocacao) AS
(SELECT DISTINCT B.nome,B.anoLancamento,L.dataLocacao
FROM Locacao L, Filmes B
WHERE L.codFilme = B.codFilme AND B.anoLancamento < 1970))
WITH DATA
```

4.4. MERGE

SQL:2003 adiciona uma quarta instrução, denominada MERGE, às já existentes (INSERT, UPDATE e DELETE). Esta instrução combina operações de inserção e atualização. O exemplo 20 mostra a tabela de filmes e uma tabela de filmes recém-chegados na locadora.

Para cada linha da tabela Filmes, procura-se um registro correspondente na tabela Filmes_RecemChegados, através do atributo codFilme. Se encontrar, atualiza linha somando as quantidades, senão, adiciona linha da tabela Filmes_RecemChegados.

```
Exemplo 20:
```

Tabela 5. Tabela de Filmes

codFilme	nome	quantidade
1	Todo Poderoso	7
2	Piratas do Caribe	6
3	O Último Samurai	10
4	O Júri	4

Tabela 6. Tabela de Filmes_RecemChegados

codFilme	nome	quantidade
2	Piratas do Caribe	5
1	Todo Poderoso	5

```
Instrução:

MERGE INTO Filmes AS A
USING (SELECT codFilme,nome,quantidade FROM Filmes_RecemChegados)
AS B
ON (A.codFilme = B.CodFilme)
WHEN MATCHED THEN
UPDATE SET quantidade = A.quantidade + B.quantidade
```

```
WHEN NOT MATCHED THEN
INSERT(codFilme,nome,quantidade)
VALUES (B.codFilme,B.nome,B.quantidade)
```

Tabela 7. Resultado da instrução MERGE

codFilme	Nome	quantidade
1	Todo Poderoso	12
2	Piratas do Caribe	11
3	O Último Samurai	10
4	O Júri	4

4.5. Geradores de seqüência

Há um novo recurso para gerar automaticamente números seqüenciais para atributos. A instrução responsável por isso é chamada de CREATE SEQUENCE, podendo especificar os valores mínimo e máximo, valor inicial do atributo, incremento, ciclo. O exemplo 21 mostra como utilizar este recurso.

```
Exemplo 21:

CREATE SEQUENCE filmeSeq AS INTEGER
START WITH 1 INCREMENT BY 1
MINVALUE 1
MAXVALUE 10000
NO CYCLE

INSERT INTO Filmes (codFilme, nome,...)
VALUES (NEXT VALUE FOR filmeSeq,'Todo Poderoso',...)
```

A função NEXT VALUE FOR aciona a criação da seqüência automática. Se a cláusula CYCLE não for especificada, quando o valor da seqüência ultrapassar um dos limites, será retornado um aviso de erro. Caso o contrário, o valor é reiniciado.

Para alterar as propriedades do gerador de seqüência é utilizado o comando ALTER SEQUENCE e para remover o gerador é utilizado o comando DROP SEQUENCE.

4.6. Colunas Identidade

Os geradores de seqüência são importantes, mas sobrecarregam os usuários com tarefas adicionais para criá-los e invocá-los através da função NEXT VALUE FOR. Para resolver isto, foi adicionado um novo recurso chamado coluna identidade. Este recurso tem como objetivo criar, num único comando, *surrogates keys* (chaves artificiais que não representam conteúdo do registro) e sem o usuário ter que executar comandos adicionais. O exemplo 22 exibe a sintaxe:

```
Exemplo 22:

CREATE TABLE Filmes (codFilme INTEGER GENERATED ALWAYS
AS IDENTITY (START WITH 1 INCREMENT BY 1
MINVALUE 1
MAXVALUE 10000
NO CYCLE),
nome VARCHAR (50))
```

Como os valores são gerados automaticamente, os usuários não precisam especificar o valor da coluna identidade, conforme exemplo 23.

```
Exemplo 23:

INSERT INTO Filmes (nome,...) VALUES ('Todo Poderoso',...)
```

O atributo codFilme não é especificado na instrução INSERT. Seu valor é gerado automaticamente chamando implicitamente a função NEXT VALUE FOR.

Se for especificado GENERATED ALWAYS, o primeiro valor da coluna identidade será o valor definido no START WITH. No exemplo 23 o primeiro valor seria 1.

Há outra opção, GENERATED BY DEFAULT, e se ela for definida, o valor assumido será o fornecido pelo usuário. Se o usuário não fornecer algum valor, este é gerado automaticamente.

4.7. Colunas Geradas

Principalmente na área de *data warehousing*, a performance pode ser melhorada se expressões forem avaliadas uma vez e seu resultado armazenado para uso futuro. Seguindo esta idéia, o SQL:2003 adiciona o recurso de colunas geradas. Elas são associadas a expressões conforme exemplo 24.

```
Exemplo 24:

CREATE TABLE Categoria_Preco (codCategoria VARCHAR(2),
valorUnitario DECIMAL(7,2), porcentagem DECIMAL(3,2),
valorDiaria GENERATED ALWAYS AS (valorUnitario * porcentagem))

INSERT INTO Categoria_Preco (codCategoria, valorUnitario,
porcentagem)
VALUES (1, 5.00, 1.1)
```

Ao inserir uma linha na tabela, o valor é da coluna valorDiaria é gerado automaticamente. No exemplo 24, houve um aumento de 1% no preço do filme da categoria 1, portanto a coluna valorDiaria terá o valor 5.50. Se os valores das colunas valorUnitario e porcentagem forem atualizados, automaticamente será atualizado o valor da coluna valorDiaria.

Colunas geradas podem conduzir a uma performance melhor não apenas porque reduzem a computação, mas também porque podem ser adicionadas no índice da tabela.

4.8. SQL/XML

Como a linguagem XML (*Extensible Markup Language*) está se tornando um padrão para troca de dados, surgiu a necessidade de adaptar os bancos de dados para garantir o armazenamento e manipulação dos dados XMl de forma eficiente. Em 2000, foi aprovada a inclusão de uma nova parte no padrão, o SQL/XML. A partir deste momento um grupo informal de companhias (incluindo IBM, Oracle e Microsoft) se reuniu para desenvolver propostas que são enviadas para H2 e WG3 aprovarem. Este grupo é chamado de “SQLX”.

Nesta nova parte é definido como o SQL pode ser utilizado em conjunto com o XML. Especifica como armazenar documentos XML no banco de dados e consultar este documentos através de uma linguagem de consulta denominada XQuery e exibir os dados no formato XML.

O armazenamento de dados XML em um banco de dados relacional exige um mapeamento entre o esquema do documento XML e o esquema do BD.

Um novo tipo de dado (tipo XML) foi adicionado para aumentar performance, pois até então um documento XML era armazenado em tipos VARCHAR e CLOB.

A geração de resultados no formato XML é feita através de novos operadores adicionados nas expressões SQL. No exemplo 27 será mostrado um destes operadores: XMLELEMENT que cria um elemento XML. Para tornar o resultado legível, ele será mostrado em formato de linhas, pois ainda não há um mapeamento do tipo XML para contexto de SQL puro.

```
Exemplo 27:
```

```
SELECT C.codCliente, XMLELEMENT(NAME "Cliente", C.nome.primeiro
|| '' || C.nome.sobrenome) AS "resultadoXML" FROM Clientes C
```

Tabela 8. Resultado da consulta em formato XML

codCliente	resultadoXML
1	<Cliente>Maria José</Cliente>
2	<Cliente>Pedro Herinque</Cliente>

A parte SQL/XML não será abordada neste artigo com detalhes. Para melhor entendimento, consultar as referências [SQLX], [Melton,2001] e [Melton,2002].

5. Recursos do padrão SQL suportados pelos gerenciadores de banco de dados

Nesta seção serão mostrados quais recursos do SQL:1999 e SQL:2003 são suportados pelos três maiores gerenciadores de banco de dados: DB2, SQL Server e Oracle.

Nas tabelas 9 e 10 foram colocados os recursos mencionados neste artigo, as versões dos três vendedores e a indicação dizendo se o recurso é ou não suportam, ou se é suportado, mas com sintaxe diferente. Esta classificação foi baseada em artigos [Gulutzan,2003] e na interpretação dos manuais dos respectivos vendedores, porém não foram realizados testes com todos os recursos.

Tabela 9. Recursos do SQL:1999

	DB2 8.1	SQL Server 2000	Oracle 9i
LOB	Suporta	Não suporta	Suporta
BOOLEAN	Não suporta	Não suporta	Não suporta
ARRAY	Não suporta	Não suporta	Sintaxe diferente
ROW	Não suporta	Não suporta	Não suporta
Tipos Distintos	Sintaxe diferente	Não suporta	Não suporta
SIMILAR TO	Não suporta	Não suporta	Não suporta
DISTINCT FROM	Não suporta	Não suporta	Não suporta
WITH RECURSIVE	Não suporta	Não suporta	Não suporta
Savepoints	Suporta	Sintaxe diferente	Suporta
Roles	Não suporta	Sintaxe Diferente	Suporta
Gatilhos	Suporta	Sintaxe diferente	Sintaxe diferente
Tipos estruturados	Suporta	Não suporta	Suporta

Tabela 10. Recursos do SQL:2003

	DB2 8.0	SQL Server 2000	Oracle 9i
BIGINT	Suporta	Suporta	Não suporta
MULTISET	Não suporta	Não suporta	Versão 10g – maior suporte
CREATE TABLE LIKE	Suporta	Não suporta	Não suporta
CREATE TABLE AS	Suporta	Não suporta	Suporta
MERGE	Suporta	Não suporta	Suporta
Geradores de seqüência	Suporta	Não suporta	Suporta
Colunas identidade	Suporta	Sintaxe diferente	Não suporta
Colunas geradas	Suporta	Sintaxe diferente	Não suporta
SQL/XML	Suporta	Suporta	Suporta

A Microsoft lançará a nova versão do SQL Server, chamada até então de Yukon. O lançamento oficial no Brasil está previsto para o segundo semestre de 2004, contendo a implementação de vários recursos do padrão SQL:1999.

Nesta seção não foram mencionados bancos de dados de código aberto, pois eles ainda não suportam a maioria dos recursos do padrão SQL. Eles estão trabalhando para seguir padrão (PostgreSQL já suporta gatilhos, predicados *similar to*, tipo *boolean*), mas afirmam que não comprometerão desempenho e facilidade.

6. Conclusão

Nos últimos anos foram adicionados recursos interessantes, tanto no padrão SQL:1999 quanto no SQL:2003. A versão SQL-92 é a mais utilizada, a SQL:1999 adicionou muitos recursos e uma grande revisão. A versão SQL:2003 é a mais recente em que o amplo uso de XML, como um formato de representação e transferência de dados, justifica as atuais pesquisas da comunidade de banco de dados nesta área.

Alguns recursos já existentes nos bancos de dados comerciais foram padronizados, mostrando que os comitês estão se integrando com a realidade comercial. Os vendedores de bancos de dados estão se preocupando com a padronização, mas extensões são criadas para obter facilidade e desempenho.

Referências Bibliográficas

1. [DB2] DB2 Universal Database InformationB2 Universal Database Information Center, http://publib.boulder.ibm.com/infocenter/db2help/index.jsp.
2. [Guimarães, 2003] Guimarães, Célio Cardoso: Fundamentos de Banco de Dados – modelagem, projeto e liguagem SQL, Editora da Unicamp, ed. 1, 2003.
3. [Gulutzan,2003] Gulutzan, Peter: Standard SQL, The Data Administration Newsletter, Janeiro de 2003, http://www.tdan.com/edatt1_archive.htm.
4. [IBM,1999] Whitemarsh Information Systems Corporation: IBM's SQL 1999 Presentation. Disponível em: http://www.wiscorp.com/SQLStandards.html.
5. [Melton,1999] Andrew Eisenberg , Jim Melton: SQL: 1999, formerly known as SQL3, SIGMOD Record, vol. 28, n°. 4, Março de 1999, pp. 131-138, http://www.acm.org/sigmod/record/issues/0403.
6. [Melton,2001] Eisenberg, Andrew, Melton, Jim: SQL/XML and the SQLX Informal Group of Companies, ACM SIGMOD Record, vol. 30, n°.3, Setembro de 2001, http://www.acm.org/sigmod/record/issues/0109/.
7. [Melton,2002] Eisenberg, Andrew, Melton, Jim: SQL/XML is Making Good Progress, ACM SIGMOD Record, vol. 31, n°.2, Junho de 2002, http://www.acm.org/sigmod/record/issues/0206/.
8. [Melton,2003] Andrew Eisenberg, Krishna Kulkarni, Jim Melton, Jan-Eike Michels, Fred Zemke: SQL:2003 has been published, Março de 1999, http://www.acm.org/sigmod/record/issues/9903/.
9. [Oracle,1999] Whitemarsh Information Systems Corporation: Oracle's SQL 1999 Presentation, http://www.wiscorp.com/SQLStandards.html.
10. [Oracle9i] Oracle 9i SQL Reference - Release 9.2, http://sqlzoo.napier.ac.uk/big/B/s/a/toc.htm.
11. [Ramakrishnan,2003] Ramakrishnan, Raghu, Gehrke, Johanes: Database Management Systems, McGraw-Hill, 3 ed., 2003.
12. [Oracle10g] Oracle Technology Networking: SQL 2003 Standard Support in Oracle Database 10g, http://otn.oracle.com/products/database/application_development/sqlxml/index.html
13. [SQLServer], SQL Server Developer Center, http://msdn.microsoft.com/sql/default.aspx.
14. [SQLX] SQLX Group, http://www.sqlx.org.
15. [Wikipedia] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/SQL.
16. [Yukon] Microsoft Windows Server System: Introducing SQL Server 2005, Code-Named "Yukon", http://www.microsoft.com/sql/yukon/.