

ORDBMS

Object Relational Database Management Systems

Overview

- Incluem novas capacidades de armazenamento de objetos aos sistemas relacionais
 - gerenciamento de dados tradicionais
 - gerenciamento de objetos complexos
 - dados espaciais
 - dados temporais
 - áudio, vídeo, imagens, ...
 - Através do encapsulamento de métodos com as estruturas de dados, um servidor ORDBMS pode executar operações complexas de manipulação de dados e transformar dados multimídia e outros dados complexos
 - Gerenciamento de transações e de performance dos sistemas relacionais
- +
- Flexibilidade dos sistemas orientados a objetos
 - estruturas tabulares e linguagem de definição de dados
 - Todas as informações persistentes permanecem (banco de dados) em tabelas
 - as linguagens são extensões daquelas utilizadas nos sistemas relacionais
 - Ex.: SQL3
- Algumas entradas tabulares podem ter estruturas de dados mais “ricas”
 - Tipo Abstratos de Dados (Abstract Data Types - ADTs)
 - ADTs são tipos de dados construídos pela combinação de tipos de dados simples com as operações associadas com o tipo em construção
 - O suporte a ADTs é atrativo porque as operações associadas com o novo tipo podem ser usadas para indexar, armazenar e recuperar os registros baseados na nova definição

ORDBMS

- Não existe um único Modelo Relacional Extendido
- Cada produto implementa características diferentes
- Comum a todos:
 - utilizam as tabelas do relacional
 - linguagem de consulta
 - incorporam algum conceito de objeto
 - alguns tem a habilidade de armazenar métodos e dados no BD
- Antes se chamavam Extended Relational DBMS
- Mais recentemente Object Relational DBMS
- Atualmente Universal Server DBMS ou Universal DBMS

Universal Servers

- IBM DB2 Universal Database (IBM)
- Informix-Danamic Server (Informix)
- Oracle 8 (Oracle)
- UniSQL/X (UniSQL)
- OSMOS (Unisys)

Padronização: SQL3

- Ainda não é um padrão de fato
- É uma evolução do SQL/92
- Inclui todo SQL/92
- Tentativa de definição nas referências a seguir:
 - International Organization for Standardization (ISSO): Database Language SQL. Document ISSO/IEC 9075:1992.
 - American National Standards Institute (ANSI). Document ANSI X3.135-1992.
- Referências
 - International Organization for Standardization (ISSO): Information Technology-Database Languages - SQL - Technical Corrigendum 1. Document ISSO/IEC 9075:1992/Cor.1:1994(E).
 - International Organization for Standardization (ISSO): Information Technology-Database Languages - SQL - Technical Corrigendum 2. Document ISSO/IEC 9075:1992/Cor.2:1996(E).
- Vantagens
 - Reusabilidade
 - Capacidade de estender o DBMS Server para realizar funcionalidades padrão de forma centralizada, em vez de Ter isto codificado em cada aplicação

- Desvantagens
 - complexidade
 - alto custo
 - *alguns acreditam que se perde a simplicidade do relacional*
 - *outros acreditam que SQL3 irá apenas servir para uma minoria de aplicações que não tem boa performance com o relacional*
 - OO pura não é o foco destas extensões

Características da linguagem

ROW TYPE

- é uma seqüência de pares de nome campo/tipo de dado que disponibiliza um novo tipo de dado que pode representar linhas de tabelas
- linhas complexas podem ser armazenadas em variáveis passadas como argumento e retonadas em funções
- permitem que uma tabela possa conter um atributo com os valores de uma linha inteira
- Ex.:

User Defined Types (UDTs)

- ADTs
- Distinct types
 - Permitem a diferenciação entre os mesmos tipos reais básicos

Exemplos:

```
CREATE TABLE owner_number_type AS CHAR(5);
CREATE TABLE staff_number_type AS CHAR(5);
```

- ao tratar uma instância de um tipo como uma instância de outro, um erro será gerado
- Em seu caso mais geral, um UDT consiste de uma ou mais definições de atributos e, adicionalmente, de rotinas (encapsulamento)
- Ainda existe discussão a respeito do uso das definições público, privado e protected

- público
 - os componentes são visíveis para todos os usuários autorizados de um UDT
- privado
 - são visíveis dentro da definição do UDT que o contém
- protected
 - são parcialmente encapsulados, sendo visíveis tanto dentro do seu UDT como dentro de toda a hierarquia de subtipos
- O valor do atributo de um UDT é acessado pelo ponto ('.')
- para cada atributo é definido automaticamente
 - *observer*
 - retorna o valor corrente do atributo
 - *mutator*
 - seta o valor do atributo para um valor passado como parâmetro
 - *observer* e *mutator* podem ser redefinidos
- *function constructor*
 - tem o mesmo nome e tipo que o UDT
 - não recebe parâmetros

- retorna uma nova instância com o atributos setados com o valor default
- pode ser redefinido

- *Ex.:*
- *stored attributes*
 - tipo comum de atributo com um nome e um tipo de dados (comum ou UDT)
- *virtual attributes*
 - derivados do uso de funções
 - *Ex.:* Age no exemplo anterior

User Defined Routines

- Definem os métodos para manipular os dados
- podem ser definidas como parte de um UDT ou como parte do *schema*
- pode ser uma procedure ou função
- podem ser providas em uma linguagem de programação como C/C++ ou definidas totalmente em SQL
- são invocadas de um comando SQL CALL
- podem receber parâmetros do tipo: IN, OUT ou INOUT

Polimorfismo

- rotinas diferentes podem Ter o mesmo nome, ou seja, o nome das funções podem ser sobrecarregados (*overloading*)
- duas funções no mesmo *schema* não podem ter a mesma assinatura: o mesmo número de args, os mesmos tipos de dados para cada arg e os mesmo tipo de retorno
- duas *procedures* no mesmo *schema* não podem Ter o mesmo nome e o mesmo número de parâmetros
- existe um estudo para usar apenas o tipo do primeiro argumento na decisão de qual método escolher

Supertipos e Subtipos

- Herança simples
- Herança múltipla
- uma instância de um subtipo é considerada uma instância de todos os seus supertipos
- *Substitutability*
 - sempre que uma instância de um supertipo é esperada, uma instância de um subtipo pode ser usada em seu lugar
 - o tipo de um UDT pode ser testado: TYPE
 - Ex.:
- Cada instância de um UDT deve estar associada com exatamente um tipo mais específico, que corresponde ao subtipo mais abaixo na hierarquia

• Herança Múltipla

- se um atributo em mais de um supertipo é herdado de um supertipo comum mais alto na hierarquia, então somente o atributo do supertipo comum é herdado
- se um atributo com o mesmo nome em cada um dos supertipos não é herdado de um supertipo comum mais alto na hierarquia então a definição deste atributo no subtipo é inválida a menos que os atributos herdados sejam renomeados

Criação de Tabelas

- Compatibilidade em SQL92
- É necessário usar o comando CREATE TABLE mesmo que a tabela consista de um único UDT
- Uma instância de um UDT somente pode tornar-se persistente se é armazenada como um valor de coluna em uma tabela
- Ex.:
- Acesso aos elementos do 1º exemplo: `staff.info.sno`
- Acesso aos elementos do 2º exemplo: `staff.sno`
- Herança em tabelas
 - Ex.:
 - Quando inserimos uma tupla nas tabelas *Staff* e *Renter*, os valores das colunas herdadas são inseridas na tabela *Person*

- Quando acessamos todas as tuplas de *Person*, isto inclui todos os detalhes de *Staff* e *Renter*
- Restrições
 - Cada linha da supertabela *Person* pode corresponder a no máximo uma linha em cada subtabela
 - Cada linha nas subtabelas *Staff* e *Renter* podem corresponder a apenas uma linha na tabela *Person*
- Comportamento de *INSERT*, *UPDATE* e *DELETE*:
 - quando uma linha é inserida em uma subtabela, os valores de qualquer atributo herdado são inseridos nas supertabelas correspondentes, em cascata para cima (cascading upwards) na hierarquia de tabelas
 - quando uma linha é alterada em uma subtabela, um procedimento similar ao descrito acima é executado para alterar os valores de atributos herdados nas correspondentes supertabelas
 - quando uma linha é alterada em uma supertabela, então todas as tabelas que tiveram atributos herdados, direta ou indiretamente, desta supertabela terão também seus valores alterados
 - quando uma linha é deletada em uma subtabela ou supertabela, as linhas correspondentes na hierarquia de tabelas também são deletadas

Consultas

- Ex1.:

Selecionar os nomes de todos os gerentes

```
SELECT s.name
FROM   staff s
WHERE  s.position = 'Manager'
```

Este exemplo invoca a função *observer* pré-definida *Position* na cláusula *WHERE* para acessar o atributo *Position* da tabela *staff*.

- Ex2.:

Selecionar os nomes e as idades de todos os gerentes

```
SELECT s.name, s.get_age
FROM   staff s
WHERE  s.is_manager;
```

Este exemplo utiliza a função definida pelo usuário *Is_Manager* como um predicado para a cláusula *WHERE*. Também é invocada a função *Get_Age* herdada de *Person*.

- Ex3.:

Selecionar os nomes e endereços de todas as pessoas com idade maior que 65 anos

```
SELECT p.name, p.address
FROM   person p
WHERE  p.get_age > 65;
```

Esta consulta irá listar, além das tuplas inseridas explicitamente em *Person*, todos os nomes e endereços de quaisquer registros que tenham sido inseridos em qualquer subtabela, direta, ou indireta, de *Person*.

- Ex4.:

Para listas apenas os detalhes das instâncias específicas de Person, deve-se utilizar ONLY.

```
SELECT p.name, p.address
FROM ONLY person p
WHERE P.GET_AGE > 65;
```

Tipo Referência e Identidade de Objeto

- No SQL92 a única forma de se definir relacionamentos entre as tabelas era o uso de **CONSTRAINTS** do tipo **PRIMARY KEY** e **FOREIGN KEY**, juntamente com a cláusula **REFERENCES**
- SQL3 permite a definição de **REFERENCE TYPE**:
 - definir relacionamentos entre tuplas
 - identificar unicamente uma tupla dentro do banco de dados inteiro
 - o seu valor pode ser armazenado em uma tabela e usado como uma referência direta a uma tupla específica em outra tabela
 - funcionalidade similar a de um OID em OODBMS
 - Ex.:

Estender o tipo Staff_Type definido pelo usuário para armazenar informações de um parente próximo (esposa, pai, mãe, ...)

```
CREATE TABLE person OF PERSON_TYPE(
    oid REF(PERSON_TYPE)
    VALUES ARE SYSTEM GENERATED);
CREATE TABLE staff_type UNDER(person_type
AS(
```

```
snum VARCHAR(5),
position VARCHAR(10),
salary
...
relative REF(PERSON_TYPE),
...);
```

```
CREATE TABLE staff OF STAFF_TYPE (
    PRIMARY KEY snum);
```

O tipo referência REF(PERSON_TYPE) pode referenciar uma tupla em qualquer tabela que seja do tipo Person_Type:

```
CREATE TABLE another_person_table OF
    PERSON_TYPE(
        oid REF(PERSON_TYPE)
        VALUES ARE SYSTEM
        GENERATED);
```

Para limitar as referências para uma tabela específica, pode-se adicionar a cláusula SCOPE:

```
CREATE TABLE staff OF STAFF_TYPE (
    PRIMARY KEY snum
    SCOPE FOR relativ IS person);
```

Consultas utilizando o operador '→':

```
SELECT s.relative→fname, s.relative→lname,
s.relative→tel_no
FROM staff s
where s.lname = 'Silva' and s.fname = 'João';
```

- **O TIPO REFERÊNCIA** não garante a **INTEGRIDADE REFERENCIAL**.
- **A INTEGRIDADE REFERENCIAL** é mantida através de **CONSTRAINTS**

Collection Types

- são construtores usados para definir coleções de outros tipos
- são usados para armazenar múltiplos valores em uma única coluna de uma tabela e podem resultar em tabelas aninhadas onde uma tabela contém outra
- o tipo da coleção pode ser um tipo pré-definido, um UDT, um *row type*, ou outra coleção
- o tipo de uma coleção não pode ser um tipo referência ou um UDT contendo uma referência
- cada coleção deve ser homogênea
- ARRAY:
 - um array unidimensional com um número máximo de elementos
- LIST:
 - coleção ordenada que permite duplicatas
- SET:
 - coleção não ordenada que não permite duplicatas
- MULTSET
 - coleção não ordenada que permite duplicatas

- Ex1.:

Estender a modelagem do item parente mais próximo (*relative*) da tabela *Staff* para conter os detalhes sobre mais de um parente

```
CREATE TABLE staff_type UNDER(person_type
AS(
    snum      VARCHAR(5),
    position  VARCHAR(10),
    salary
    ...
    relative  SET(PERSON_TYPE),
    ...);
SELECT n.fname, n.lname, n.tel_no
FROM   staff s, TABLE(s.relative) n
WHERE  s.lname = 'Silva' and
s.fname='João';
```

- Ex2.:

Encontrar quantos parentes um determinado membro de *Staff* possui

```
SELECT snum, fname, lname, COUNT(relative)
FROM   staff;
```

Como o atributo *Relative* é do tipo SET, é possível utilizar a função de agregação COUNT

Persistent Stored Modules

- Novos comandos adicionados ao SQL3
- atribuição, seleção, repetição, CALL e RETURN

ORDBMS X OODBMS

- Modelagem de Dados

característica	ORDBMS	OODBMS
OID	REF Type	SIM
encapsulamento	UDTs	SIM
herança	Hierarquias de tabelas e UDTs	SIM
polimorfismo	SIM	SIM
objetos complexos	UDTs	SIM
relacionamentos	Integridade Referencial	SIM

- Acesso aos Dados

Característica	ORDBMS	OODBMS
criação e acesso	SIM (não transparente)	SIM (transparência varia)
consultas	SIM (SQL3)	SIM (ODMG 2.0)
navegação	SIM(REF Type)	SIM
restrições integridade	SIM	NÃO

- Compartilhamento entre os dados

Característica	ORDBMS	OODBMS
ACID transactions	SIM	SIM (em estudo)
recovery	SIM	SIM (em estudo)
novos modelos de transações	Não	SIM (em estudo)
segurança, integridade, visões	SIM	Limitado

EXEMPLOS

ROW TYPE

- Ex1.:

```
CREATE TABLE office (  
    onum VARCHAR(25),  
    address ROW(  
        street VARCHAR(25),  
        area VARCHAR(15),  
        city VARCHAR(15),  
        pcode ROW(city_id VARCHAR(4),  
                subpart VARCHAR(4))) );
```

User Defined Types (UDTs)

- *observer e mutator*

```
FUNCTION fname(p person_type)  
    RETURNS VARCHAR  
    RETURN p.fname;
```

```
FUNCTION fname( p person_type,  
               new_value VARCHAR)  
    RETURNS person_type
```

```
BEGIN  
    p.name := new_value;  
    RETURN p;  
END
```