# Programação SQL via linguagem hospedeira (Cap 8 do livro texto, com extensões)

Via biblioteca de funções:

**CLI** (Call Level Interface): padronizada no SQL92:

- Permite abrir conexões com o SGBD (podem ser remotas),
- Passar parâmetros,
- Executar comandos SQL,
- Usar *cursores* para varrer as linhas do resultado de uma consulta SQL
- Permite implementar o conceito de *SQL dinâmico:*
  Cadeias com comandos SQL são criadas dinamicamente pela aplicação e submetidos ao SGBD
- Permite comunicação remota com SGBD via protocolo específico do SGBD

# Programação SQL via linguagem hospedeira

**Problemas:**

- O padrão SQL CLI não é seguido pela maioria dos SGBDs
- Uma biblioteca distinta para cada SGBD é implementada em cada linguagem hospedeira (!),
- Dificulta aplicações que usam Bancos de Dados heterogêneos e legados (comum em grandes empresas)
- Veja exemplos no apêndice 1 do livro texto (para PHP)

- **Solução**:
- API (biblioteca) genérica independente do SGBD:
  - Feita para cada linguagem hospedeira
  - Funciona como *middleware* para as bibliotecas específicas
  - Exemplos: PEAR DB e ADOdb para PHP, JDBC para Java

# Programação SQL via linguagem hospedeira

## Linguagens hospedeiras mais usadas:

**Antes do advento da Web**:
- Visual Basic, Visual C++, C++,

**Após o advento da Web:**
- Java, Php, Python, Perl, C# (todas interpretadas)
- A aplicação agora é um módulo disparado pelo Servidor Web:
  - pode ser um processo, uma thread, ou interpretador da linguagem embutido no Servidor Web
- Usuário (*cliente*) interage com o sistema via navegador
- Três camadas de SW no *servidor* :
  - Servidor Web: comunica-se com cliente (*browser*) via http
  - Aplicação: escrita numa das linguagens hospedeiras
  - SGBD: pode estar numa máquina remota ao Servidor Web

# Programação SQL via linguagem hospedeira

**Como funciona:**
- Usuário (browser) clica num link dirigido para o Servidor Web, (browser abre conexão TCP na porta 80 do Servidor Web),
- Browser exibe página vinda do Servidor, p. ex. um formulário
- Usuário preenche formulário no browser e o "submete" (browser envia dados via mensagem *http* do tipo *POST)*
- Servidor Web obtem *path* do script tratador da mensagem:
  - Servidor Web redireciona o socket da conexão para a entrada/saída padrão do script e dispara a sua execução
- Script obtem os dados do formulário e faz conexão com o SGBD
- Script prepara e envia comando SQL para o SGBD
- Script recebe resultado do comando SQL, prepara páginaHTML, escreve página HTML na "saída padrão" que é redirecionada para o socket da comunicação com o browser!
- Browser recebe e exibe página com resultado do comando SQL

# Programação SQL via linguagem hospedeira

**Exemplo:** Script pata acesso a uma BD MySql (p. 245 livro texto)
(requer o módulo driver para o SGBD MySql)

```
$server=”myserver.ic.unicamp.br”; $dbase=”mydbase); #$user.etc
$conn= mysql_connect($server, $user, $password);
$mysql_select($dbase);  # usa conexão p/ acessar BD específica
$query= “select * from lista_torneios”;  # exemplo de consulta
$rs= mysql_query($query); # envia comando select para executar
        # retorna cursor ($rs) para o resultado da consulta SQL
$nc= mysql_num_fields($rs) # número de colunas no resultado
$meta= mysql_fetch_field($rs);     # obtem descrição de 1 coluna
$line= mysql_fetch_row($rs);       # obtem uma linha do resultado,
                                   # retorna null qdo todas linhas lidas
```

# Programação SQL via linguagem hospedeira

**Soluções alternativas ao uso de um SGBD “full fledged”**

SGBD “leve”, funciona como módulo da linguagem hospedeira.
Exemplos:
**SQLite**: pode ser usado com Java,PHP, Python, Perl, C, etc
**HSQLDB (HyperSQL):** implementado em Java

## SQLite:

- Nenhum procedimento de instalação/configuração
- Executado como um módulo pelo processo da aplicação
- Permite acesso concorrente à BD: multiple readers/1writer lock
- Implementa a BD como um arquivo comum no disco
- BD possui formato independente da plataforma (little endian/ big endian, 32/64 bits: pode ser transportada num *pen drive*
- A biblioteca do Sqlite ocupa apenas 275KB

# Programação SQL via linguagem hospedeira

- Código fonte em domínio público  (nenhum copyright)
- Modo interativo (linha de comando) facilita testes/depuração
- Limites da BD, tabelas, colunas, etc grandes e parametrizáveis
- Implementa a maioria dos recursos do SQL2 + extensões:
  - Chaves estrangeiras e chaves alternativas
  - Triggers
  - Colunas *varchar* ocupam apenas o espaço usado
  - Colunas possuem “tipos manifestos” (dinâmicos) x estáticos:
    - Podem conter tipos distintos do da declaração
    - Ex: uma cadeia pode ser inserida onde o tipo definido era inteiro
  - Chave Primária do tipo integer – faz autoincrement
  - Suporta BLOBs
  - Implementa transações atômicas: BEGIN …  COMMIT
  - Suporta índices

# Programação SQL via linguagem hospedeira

**Um exemplo minimal de SQLite com Python:**

```
import sqlite
con = sqlite.connect('mydatabase.db')  # cria BD se não existe
cur = con.cursor()    # cria um cursor para acessar a BD
cur.execute('CREATE TABLE foo (o_id INTEGER PRIMARY
KEY, fruit VARCHAR(20), veges VARCHAR(30))')
con.commit()            # força escrita no disco
cur.execute('INSERT INTO foo (o_id, fruit, veges)
VALUES(NULL, "apple", "broccoli")') # Chave Primária autoincr
con.commit()            # força escrita no disco
print cur.lastrowid
cur.execute('SELECT * FROM foo')
print cur.fetchall()    # exibe as linhas retornadas pelo select
```

# Programação SQL via linguagem hospedeira

**HSQLDB** – Java SQL Database

- Usa a API *JDBC* de comunicação com Bases de Dados
- Armazena BD em memória ou “cacheada” em disco
- Suporta SQL92 e recursos de SQL-1999 e SQL-2008
- Suporta acesso concorrente (*multi-threaded*)
- Velocidade comparável à de SGBDs open source
- *Redo log* para implementar recuperação de transações
- Suporta controle de concorrência via *locks* ou *multiversão*, e os níveis *read committed* e *serializable* do padrão SQL2
- Limites grandes para tabelas (16 GB)
- Instalação requer JRE ou JDK (este já vem com JDBC)

# Programação SQL via linguagem hospedeira

**Um exemplo de uso de JDBC para acesso a BD MySql:**
Três etapas:
(i)carrega o driver JDBC do MySql,(ii) faz conexão ao SGBD, (iii)prepara, executa comando(s) SQL e obtem o resultado

```
public class LoadDriver {              // carrega o driver JDBC
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {    // handle the error
        }
Connection conn = null;            // faz conexão com SGBD
       try{
DriverManager.getConnection(jdbc:mysql://titi.lab.ic.unicamp.br/te
nis?" +  "user=scott&password=tiger");
       }
```

# Programação SQL via linguagem hospedeira

## Exemplo de uso de JDBC (continuação)

```
Statement stmt = null;
ResultSet rs = null;
try {
    stmt = conn.createStatement();
    stmt.executeQuery("SELECT * FROM jogadores");
    rs = stmt.getResultSet();
    }
    // Now do something with the ResultSet ....
if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { } // ignore
        stmt = null;
}
```

# Programação SQL via linguagem hospedeira

## O problema do "casamento de impedância"

- "Persistência de dados" requer armazenamento confiável, eficiência de atualização, recuperação, backup
- SGBDs Relacionais provêm esses recursos
- SGBDs orientados a objetos não "decolaram" até hoje

**Por outro lado**,

- Objetos de linguagens OO não são facilmente mapeados para os tipos de dados do SQL
- É o conhecido problema de "descasamento de impedância" ("*impedance mismatch*")

## Programação SQL via linguagem hospedeira

**O problema do "casamento de impedância" (cont):**
**Solução:**

- Biblioteca para fazer o "mapeamento Objeto-Relacional":
  F*ramework* para mapear objetos em tabelas/linhas de uma BD Relacional convencional
- Várias existem para o ambiente JAVA:
  **TopLink** (1998 - hoje da Oracle)
  **Hibernate** (2001 - JBoss)
  - Mapeia classes Java para tabelas.
  - Mapeia tipos de dados de Java para os tipos de SQL
  - Mapeamento feito via configuração de arquivo XML, ou via *Java Annotations*
  - Provê linguagem para consultas sobre objetos(classes):
    - HQL – faz consultas SQL-like sobre objetos

## Programação SQL via linguagem hospedeira

**Hibernate (cont):**

- Compatível com .NET
- Versão 3.5 adere ao padrão JPA (Java Persistence API)

- **JPA - Java Persistence API**:
  3 componentes:
- API especifica sintaxe p/ mapear classes para tabelas
- JPQL – Java Persistent Query Language – SQL like
- Metadados para mapeamento Objeto- Relacional
- Apenas classes lightweight (POJO) são mapeáveis:
  - Sintaxe do mapeamento usa *anotações,*
  - Parâmetros da conexão (servidor,porta, BD) usa arquivo XML

# Programação SQL via linguagem hospedeira JPA (cont)

Um exemplo simples: definição de uma classe (entidade) com anotações informando o mapeamento para SQL:

```
@Entity
@Table(name = "t_customer")
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(nullable = false)
    private String firstname;
    @Column(nullable = false, length = 30)
    private String lastname;
    @Column(length = 15)
    private String telephone;
    @Column(name = "e_mail")
    private String email;
    private Integer age;
        // incluir aqui constutores, getters, setters
}
```

# Programação SQL via linguagem hospedeira JPA (continuação)

Esquema da tabela SQL correspondente (observe a chave primária gerada via auto-incenent):

```
Create table t_customer
( id            bigint(20)  not null   auto_increment primary key,
 firstname  varchar(255) not null,
  lastname  varchar(30) not null,
  telephone varchar(15),
  e_mail       varchar(255),
  age          int(11)
)
```