# Laboratorio di
# Sistemi Informativi

[ tomo 2 ]

Master in Tecnologie del Software Libero
e Open Source

Stefano Zacchiroli
zack@cs.unibo.it

# DBMS per sviluppatori

astrazione

# Eterogeneità dei DBMS

- l'eterogeneità dei DBMS è problematica per gli sviluppatori
  - l'estremo (da evitare se possibile) è scrivere backend per ogni DBMS che si intende supportare
    - caso non infrequente, ma solo per grossi applicativi che necessitano di feature avanzate
- eterogeneità da considerare:
  1. eterogeneità di feature
     - induce eterogeneità a livello di ling. DML/DDL/DCL
  2. eterogeneità di accesso
     - diversi client (per shell script)
     - diverse librerie (per lo sviluppo in vari linguaggi)

# Astrazione sui DBMS

- soluzione informatica proverbiale:

  *« just add an extra level of indirection »*

- per l'eterogeneità di feature (1.)
  - soluzioni solo parziali :-(
  - gli standard non sono sufficienti: sono implementati parzialmente e male nei DBMS
  - scrivere query nell'intersezione dei linguaggi dei DBMS che si vogliono supportare
  - scrivere in linguaggi intermedi (non SQL) che vengono poi "compilati" verso DBMS specifici
    - e.g. ORM (Object Relational Mapper)

# Astrazione sui DBMS (cont.)

- per l'eterogeneità di accesso (2.)
  - soluzioni “accettabili”
  - ODBC (language independent, ma bloated)
  - librerie language-specific di astrazione sui DBMS
    - una API comune
      - definita con processi simili alla standardizzazione, con la descrizione di feature opzionali e obbligatorie
    - una implementazione per ogni DBMS supportato
    - codice scritto per tali librerie è solitamente portabile da un DBMS ad un altro cambiando “poco” (i.e. una stringa o l'import di un modulo)
    - esempi notevoli:
      - DBI (Perl), DB-API 2.0 (Python), JDBC (Java), ...

# DBI in a nutshell

- What is DBI?

"*The DBI is the standard database interface module for Perl. It defines a set of methods, variables and conventions that provide a consistent database interface independent of the actual database being used.*"

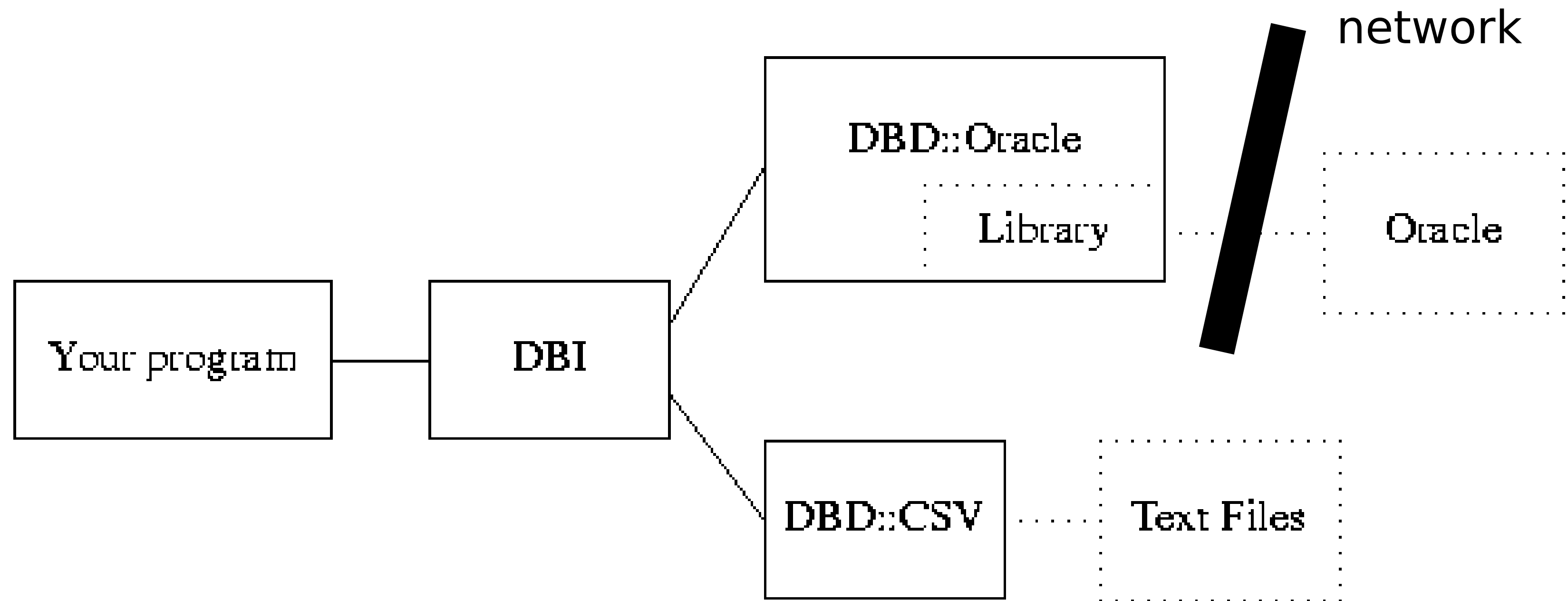
-- Tim Bunce

- riferimenti
  - http://dbi.perl.org/
  - DBI FAQ, http://dev.isystek.com/dbi/fom.cgi
  - *“Programming the Perl DBI“*, O'Reilly

# DBI - overview

- cos'è?
  - l'”extra level of indirection” per scrivere applicazioni che effettuano query SQL in Perl
- feature overview
  - *DataBase Driver (DBD)* come layer d'astrazione sul DBMS in uso
  - un driver DBD per ogni *data source* supportato
  - lista (non esaustiva) di driver DBD:
    - MySQL, PostgreSQL, SQL Server, Oracle, Firebird, ...
    - SQLite, CSV, Excel, ...

# DBI - architettura




- DBI non sa come comunicare con alcun DBMS
  - sa però come trovare il DBD driver del caso
- una query effettuata a DBI viene forwardata al driver che (eventualmente) usa una libreria DBMS-specific e ritorna il risultato a DBI

# DBI - pacchetti

- libdbi-perl
  - main DBI abstraction layer, API
  - modulo Perl “DBI”
  - documentazione generica

    `man 3pm DBI`
- libdbd-*driver*-perl
  - DBD driver, query engine DBMS-specific
    - e.g.: libdbd-{pg,mysql,sqlite3}-perl
  - documentazione del driver

    `man 3pm DBD::`*`driver`*
    - usuali cookbook-like manpage dei moduli Perl

# DBI – query reuse

- necessità frequente: iterare query SQL al DBMS
  - rationale: SQL non è Turing completo, i “loop” vanno gestiti esternamente
- è particolarmente inefficiente ricreare le query SQL ad ogni iterazione
  - garbage collection delle query (viste come stringhe)
  - impossibilità di risuare le ottimizzazioni lato DBMS
- DBI (& co) permette di fattorizzare le query
  - distinzione tra *preparazione* ed *esecuzione*
  - il DBMS *può* pre-compilare le query: risparmia parsing, rete e può riusare le ottimizzazioni

# DBI – query reuse (cont.)

- l'uso è semplice
  - *placeholder* nelle query a prepare-time
  - riempimento degli stessi a execution-time
- e.g.

```
my $s = $d->prepare("SELECT * FROM foo WHERE bar = ?");
$s->execute("baz");
$s->execute("quux");
...
```

- “prepare” può essere emulata a livello DBD se non supportato dal DBMS

# DBI - uso

- working example

```
use DBI;
my $dbh = DBI->connect(
    'DBI:mysql:database=db;host=localhost')
  or die "Couldn't connect to database: " . DBI->errstr;
my $sth =
  $dbh->prepare("SELECT * FROM foo WHERE bar = ?");
$sth->execute("quux");
while (@row = $sth->fetchrow_array()) { print "@row\n"; }
$sth->finish();
$dbh->disconnect();
```

# DBI - esercizi

- scrivere i seguenti script Perl:
  1. esame.pl <matricola> <corso> <voto>
     - verifica che voto sia >= 18 e aggiunge una riga alla tabella esami, voto può avere “L” come suffisso (lode)
  2. media.pl <cognome>
     - cerca tutti gli studenti che hanno il cognome fornito e ne stampa la media voti a video
- testarli sui database “studenti” precedentemente creati su MySQL e Postgres
  - i dettagli per l'uso dei driver BDB si trovano sulle manpage DBD::Mysql e DBD::Pg

# ORM (Object-Relational) Mapping

- Reprisal sull'astrazione (deja-vu?):
  - soluzione (parziale) per l'eterogeneità di feature
    1. scrivere query (SQL) nell'intersezione dei DBMS che si vogliono supportare *oppure*
    2. scrivere in *linguaggi intermedi* (non SQL) che vengono poi “compilati” verso DBMS specifici

    Perl DBI è un esempio di (1.)
- Caso particolare di (2.):
  - il *linguaggio intermedio* coincide con (o è un sottoinsieme di) il linguaggio di sviluppo dell'applicazione

# ORM (cont.)

- ORM è una tecnica di programmazione per convertire dati tra il sistema di tipi dei linguaggi ad oggetti e quello dei DBMS
- concetti chiave di ORM:
  - restrizione al paradigma di *programmazione ad oggetti* (non applicabile ad altri paradigmi)
  - inferire *entità* e *relazioni* (dell'universo relazionale) da *classi* e *attributi* (dell'universo della programmazione ad oggetti)
- il linguaggio intermedio nel caso di ORM è il linguaggio di definizione delle classi

# ORM (cont.)

- risultato per il programmatore
  - un database “virtuale” di oggetti che usa un DBMS come data storage
  - molte delle azioni esprimibili in SQL sono nascoste da azioni su oggetti/classi, e.g.:
    - creazione di tabelle -> definizione di classe
    - inserimento di tuple -> istanziazione
    - proiezione/update -> accesso ad attributi
    - join -> definizione di classe + accesso ad attributi
    - cancellazione -> uso di class method
    - selezione di tuple -> uso di class method

# [ Digressione su *DBMS ]

( i.e.: tantissime buzzword in 2 slide )

- bestiario:
  - RDBMS
    - ormai li conosciamo :-)
  - ORDBMS (poco accordo sul termine...)
    - DBMS estensibile che permette l'integrazione di tipi di dati e metodi forniti dall'utente. Non c'entra nulla con la programmazione OO
  - ODBMS (Object-oriented DBMS)
    - DBMS in cui l'informazione è rappresentata come istanze (nel senso della programmazione ad oggetti)

# [ Digressione su *DBMS (cont.) ]

- in un ODBMS il programmatore sviluppa ad oggetti e stabilisce quali oggetti sono *persistenti*
  - i.e. quali vivono nel data storage del ODBMS
- esempi di ODBMS
  - db4o (database for objects), per Java e .NET
  - Zope Object Database (ZODB), per Python
- una implementazione di ORM *non è* un ODBMS. È piuttosto un ODBMS “virtuale”, in quanto l'astrazione è la stessa, ma il data storage è un RDBMS, per nulla OO-specific

# ORM (cont.)

- effetti collaterali:
  - astrazione sul DBMS: essendo il linguaggio di definizione degli oggetti scorrelato dal DBMS, il mapping verso il mondo relazionale può essere istanziato più volte per diversi DBMS
  - diventa inoltre possibile (grazie al maggiore livello di astrazione) implementare a livello ORM feature non disponibili nel DBMS sottostante
    - e.g. integrità referenziale, trigger, ...
    - case study: SQLObject implementa l'integrità referenziale per SQLite

# ORM (cont.)

- ORM è una *tecnica* di programmazione language-neutral ; le sue *implementazioni* sono librerie language-specific
- idee comuni a molte implementazioni
  - classi specifiche da cui ereditare per stabilire un mapping tra una classe ed una tabella del DBMS
  - convenzioni su nomi / metodi di classe per definire campi di una tabella e join
  - uso dell'introspezione

# SQLObject

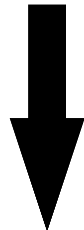
- SQLObject è un diffuso ORM open-source per il linguaggio di programmazione Python
- feature
  - database independence
    - MySQL, Postgres, SQLite, Firebird, MS SQL Server ...
  - non tenta di “nascondere” completamente il DBMS
  - sqlbuilder: astrazione ad AST per costruire query SQL
    - mappabili anche verso DBMS non-SQL
- success stories
  - framework web: Turbogears, Pylons, ...
- http://www.sqlobject.org

# SQLObject – dichiarazioni

```
from sqlobject import *

sqlhub.processConnection = \
  connectionForURI('sqlite:/:memory:')


class Person(SQLObject):

    fname = StringCol()

    mi = StringCol(length=1, default=None)

    lname = StringCol()

    ...


Person.createTable()
```

# SQLObject – SQL

```
class Person(SQLObject):
    fname = StringCol()
    mi = StringCol(length=1, default=None)
    lname = StringCol()
```



```
CREATE TABLE person (          --- sintassi: SQLite / MySQL
    id INT PRIMARY KEY AUTO_INCREMENT,
    fname TEXT,
    mi CHAR(1),
    lname TEXT
);
```

# SQLObject – tipi di colonne

- alcuni tipi di colonne supportate e tipi equivalenti Python
  - BLOBCol (string), BoolCol (boolean), DecimalCol / CurrencyCol (float), StringCol / UnicodeCol (string)
  - DateCol / DateTimeCol (datetime.{date/datetime})
  - EnumCol (string), SetCol (set)
- il mapping automatico tra tipi di colonne e tipi del linguaggio rimuove la necessità di marshalling/unmarshalling

# SQLObject - INSERT

```
>>> p = Person(fname="John", lname="Doe")
>>> p
<Person 1 fname='John' mi=None lname='Doe'>
>>> p.fname
'John'
>>> p.mi = 'Q'
>>> p2 = Person.get(1)
>>> p2
<Person 1 fname='John' mi='Q' lname='Doe'>
>>> p is p2
True
```

# SQLObject - SELECT

```
>>> peeps = Person.select(Person.q.firstName=="John")
>>> list(peeps)
[<Person 1 fname='John' mi='Q' lname='Doe'>]
peeps = Person.select(
            AND(Address.q.personID == Person.q.id,
                Address.q.zip.startswith('504')))
...
peeps = Person.selectBy(firstName="John", lastName="Doe")
...
bob = User.byUsername('bob')
# richiede “alternateID=True”
```

# SQLObject – relazioni 1-*

```
class Address(SQLObject):
    street = StringCol()
    city = StringCol()
    state = StringCol(length=2)
    zip = StringCol(length=9)
    person = ForeignKey('Person')

class Person(SQLObject):
    addresses = MultipleJoin('Address')

# l'accesso all'attributo “addresses” di un oggetto
# Person restituisce una lista di oggetti di tipo Address
```

# SQLObject – relazioni *-*

```
class Utente(SQLObject):
    nome = StringCol(alternateID=True, length=20)
    ruoli = RelatedJoin('Ruolo')

class Ruolo(SQLObject):
    nome = StringCol(alternateID=True, length=20)
    utenti = RelatedJoin('Utente')

# la tabella extra per il join viene creata implicitamente
# da SQLObject e non ha alcuna classe corrispondente
```

# SQLObject - esercizi

- esercizio Python-specific, sorry :-(
- reimplementare gli esercizi precedentemente svolti su Perl DBI (slide 14) utilizzando lo ORM SQLObject

# DBMS embedded

SQLite

# DBMS più “facili”

- installare / configurare / istanziare un DBMS non è un task triviale ...
  - richiede competenze che esulano dalle normail competenzi di utenti e sviluppatori
- ... ma SQL ed il modello relazionale sono diffusi
  - know-how degli sviluppatori in materia
  - l'implementazione di un motore di query base SQL è un requisito di molti applicativi
    - e.g.: molte applicazioni LAMP o più in generale web-based
- i DBMS embedded affrontano questo problema

# DBMS embedded

- a differenza dei DBMS stand-alone, i DBMS embedded:
  - non vengono eseguiti in un processo separato
    - nessuna necessità di configurare i canali di accesso
  - sono implementati come librerie linkate nell'applicazione che li usa
    - nessuna necessità di configurare/istanziare un servizio di sistema, ci si basa su competenze usuali degli sviluppatori: l'uso di librerie
  - usano file ordinari come data storage
    - nessuna necessità di configurare reami aggiuntivi di permessi, ci si base sui permessi del filesystem

# SQLite

- SQLite è un DBMS embedded public-domain
  - il più diffuso TTBOMK
- scritto in C (65kloc)
  - codice elegante e ben commentato, API semplice
- usato come libreria (shared / static)
  - footprint 150-250 Kb (minimale-completo)
- ottime prestazioni
  - nei casi d'uso appropriati ! ...
  - ... nei quali spesso è più veloce dei fratelli maggiori stand-alone !!

# SQLite - feature

- self-contained
- zero-configuration
- DB su file singolo
- transazioni ACID
- scala a terabyte di db
  - supporta i BLOB
- è disponibile un top-level interattivo
- implementa SQL-92, eccezioni:
  - no foreign keys
  - no 100% ALTER TABLE
  - no 100% trigger
  - no RIGHT/OUTER join
  - no nested transaction
  - no VIEW scrivibili
  - no GRANT/REVOKE (ma non serve!)

# SQLite – usi appropriati

- non è un DBMS “one size does fit all”
  - a sinistra nel tradeoff tra semplicità e ammontare di feature disponibili
    - velocità e basso footprint sono effetti collaterali
- usi appropriati:
  - website, rimpiazzo per application-specific file format, sistemi embedded, db temporanei, analisi fire-and-forget di dati dalla cmdline, sviluppo RAD, sperimentazione di estensioni, didattica
- usi *non* appropriati
  - dataset molto grandi (petabyte), architetture inerentemente distribuite, alta concorrenza (locking), query SQL eccessivamente complesse

# SQLite – pacchetti

- pacchetto sorgente “sqlite3”, binari:
  - sqlite3
    - top-level interattivo per query (“sqlite3”)
  - libsqlite3-0 / libsqlite3-dev
    - librerie shared (dipendenze per i runtime) / librerie di sviluppo (dipendenze per la compilazione)
  - sqlite3-doc
    - documentazione (mirror del website)
    - ... ma (ovviamente) manca l'utilissimo wiki!
- reference: http://www.sqlite.org

# SQLite – installation HOWTO

- facile (per il toplevel):

```
aptitude install sqlite3
```

- le librerie shared vengono installate automaticamente come dipendenze di pacchetti
- non c'è necessità di setup post-installazione
  - ogni applicazione crea il db di cui necessità invocando la libreria shared
- le librerie -dev vanno installate on-demand:

```
aptitude install libsqlite3-dev
```

# SQLite – gestione dei DB

- uso dei database

```
$ sqlite3 filename
```

- creazione di database
  - vengono creati in maniera lazy alla prima scrittura su di un database non esistente

```
$ sqlite3 foo.sqlite
sqlite> CREATE TABLE foo (id INT);
sqlite> CTRL-D
```

- rimozione di database

```
rm -f filename   # :-)
```

# SQLite – shell

- “sqlite3” è il top-level interattivo per l'accesso a DB sqlite
  - GNU readline support
  - script-friendly
- convenzioni
  - i comandi extra-SQL iniziano con “.”
  - i comandi SQL sono terminati da “;”

# SQLite – shell (cont.)

```
sqlite> .help    -- riformattato / accorciato
.dump ?TABLE? ...      Dump the database in an SQL text format
.explain ON|OFF        Turn output mode suitable for EXPLAIN on or off.
.help                  Show this message
.import FILE TABLE     Import data from FILE into TABLE
.indices TABLE         Show names of all indices on TABLE
.load FILE ?ENTRY?     Load an extension library
.mode MODE ?TABLE?     Set output mode where MODE is one of:
                         csv, column, html, insert, line, list, tabs, tcl
.output FILENAME       Send output to FILENAME
.read FILENAME         Execute SQL in FILENAME
.schema ?TABLE?        Show the CREATE statements
.tables ?PATTERN?      List names of tables matching a LIKE pattern
```

# SQLite - esercizi

- create un database SQLite come da esercizi della slide 101
  http://www.bononia.it/~zack/courses/labsimfosset0607/Lezione%201.pdf
- cosa non è enforced dei vincoli espressi nello schema?
  - come si confronta SQLite a livello di vincoli supportati con MySQL e Postgres
- creare e popolare il db della slide 106
- “portare” gli script Perl/Python realizzati in precedenza (usando rispettivamente DBI e SQLObject) a SQLite

# SQLite – utenza

- il controllo di accesso è delegato al file system
- non esistono quindi equivalenti degli statement GRANT/REVOKE
- locking lettori/scrittori implicito ed effettuato a livello di transazione

# SQLite - miscellanea

... alcuni link come spunti di discussione ...

- benchmark / conronti di performance
  - http://www.sqlite.org/cvstrac/wiki?p=SpeedComparison
- success stories
  - http://www.sqlite.org/cvstrac/wiki?p=BigNameUsers
- amalgamation
  - http://www.sqlite.org/cvstrac/wiki?p=TheAmalgamation
- supporto per full text search
  - http://www.sqlite.org/cvstrac/wiki?p=FtsTwo

# SQLite – core API

```
#include <stdio.h>
#include <sqlite3.h>
int main(int argc, char **argv){
  sqlite3 *db;
  char *zErrMsg = 0;
  int rc;
  if( argc!=3 ){
    fprintf(stderr, "Usage: %s DATABASE SQL-STATEMENT\n", argv[0]);
    exit(1);
  }
  rc = sqlite3_open(argv[1], &db);
  if( rc ){
    fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    sqlite3_close(db);
    exit(1);
  }
  rc = sqlite3_exec(db, argv[2], callback, 0, &zErrMsg);
  if( rc!=SQLITE_OK ){
    fprintf(stderr, "SQL error: %s\n", zErrMsg);
    sqlite3_free(zErrMsg);
  }
  sqlite3_close(db);
  return 0;
}
```

# SQLite – core API (cont.)

```
static int callback(void *NotUsed, int argc, char **argv, char **azColName){
  int i;
  for(i=0; i<argc; i++){
    printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
  }
  printf("\n");
  return 0;
}
```

(fonte: http://www.sqlite.org/quickstart.html)

- riferimenti
  - http://www.sqlite.org/capi3.html (manuale)
  - http://www.sqlite.org/capi3ref.html (API reference)

# SQLite - estensioni

- dalla versione 3.3.6 SQLite permette di caricare estensioni da shared library
  - sintassi (alternative):

```
sqlite> .load filename
sqlite> SELECT load_extension('filename');
```

- una estensione può definire nuove funzioni (aggregate o meno) e nuove collating sequence
- al momento del caricamento la funzione entry point `sqlite3_extension_init` delle estensioni viene invocata
- riferimenti: http://www.sqlite.org/cvstrac/wiki?p=LoadableExtensions

# SQLite – estensioni (cont.)

- prototipo dell'entry point:

```
int sqlite3_extension_init(
   sqlite3 *db,          /* database connection */
   char **pzErrMsg,      /* Write error msgs here */
   const sqlite3_api_routines *pApi  /* API methods */
);
```

  - *db* è analogo a quanto ritornato da sqlite3_open, è da passare a sqlite3_create_{function,collation}
  - *pzErrMsg* può essere riempito con sqlite3_mprintf per ritornare messaggi di errore
  - *pApi* è il punto di accesso alla API di SQLite per l'estensione

# SQLite – estensioni (cont.)

- problematiche di sicurezza
  - (purtroppo) non sono infrequenti applicazioni che permettono all'utente finale di inserire statement SQL che vengono passati verbatim ad un DBMS
  - ciò non è particolarmente problematico con SQL standard
    - si rischia di compromettere “solo” il database
    - esistono meccanismi di blacklisting statement per statement
  - è problematico per le estensioni
    - una estensione può eseguire codice C arbitrario
    - è possibile bypassare il blacklisting

# SQLite – estensioni (cont.)

- problematiche di sicurezza (cont.)
  - SQLite prevede un master lock per le estensioni
    - è attivo di default (i.e. non è possibile caricarne)
    - è disattivabile via `sqlite3_enable_load_extension();`
  - nei pacchetti Debian è disattivato per default
  - se usate altre distribuzioni di SQLite può essere necessario ricompilarlo per potere caricare le estensioni dal top-level
  - maggiori info su http://www.sqlite.org/cvstrac/wiki?p=LoadableExtensions

# SQLite – esempio di ext.

```
#include <sqlite3ext.h>               // Notazione: boilerplate, highlight
SQLITE_EXTENSION_INIT1

/* The half() SQL function returns half of its input value. */
static void halfFunc(
  sqlite3_context *context,
  int argc,
  sqlite3_value **argv
){
  sqlite3_result_double(context, 0.5*sqlite3_value_double(argv[0]));
}
int sqlite3_extension_init(
  sqlite3 *db,
  char **pzErrMsg,
  const sqlite3_api_routines *pApi
){
  SQLITE_EXTENSION_INIT2(pApi)
  sqlite3_create_function(db, "half", 1, SQLITE_ANY, 0, halfFunc, 0, 0);
  /* ... other user-defined functions/collation here ... */
  return 0;
}                            // Fonte: http://www.sqlite.org/cvstrac/wiki?p=LoadableExtensions
```

# SQLite – esempio di ext. (cont.)

- compilazione

```
$ ls -l my_sqlite3_ext.c
-rw-r--r-- 1 zack zack 548 2007-07-06 15:03 my_sqlite3_ext.c
$ gcc -shared my_sqlite3_ext.c -o half.so
$ ls -l half.so ; file half.so
-rwxr-xr-x 1 zack zack 5629 2007-07-06 15:04 half.so
half.so: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), not
   stripped
```

- loading & testing

```
$ export LD_LIBRARY_PATH=`pwd`
$ sqlite3
SQLite version 3.3.17
Enter ".help" for instructions
sqlite> select half(3);
SQL error: no such function: half
sqlite> .load half.so
sqlite> select half(3);
1.5
```

# SQLite – extension API

- punti di partenza per sopravvivere alla API reference:
  - sqlite3_result_*
    - e.g. sqlite3_result_text
    - insieme di funzioni usabili da un'estensione per restituire valori all'engine SQLite
    - effettuano la conversione da tipi C a tipi SQLite
  - sqlite3_value_*
    - e.g. sqlite3_value_text
    - duali alle funzioni di cui sopra
    - insieme di funzioni per ottenere dall'engine SQLite i parametri passati ad una funzione
    - effettuano la conversione da tipi SQLite a tipi C

# SQLite - esercizi

- utilizzando il meccanismo delle loadable extension di SQLite implementare le 5 tracce dei progetti di Laboratorio Sistemi Informativi :-)