SQL for the applications

Call Level Interface (CLI)

Call Level Interface

- Requests are sent to the DBMS through functions of the host language
  - solution based on predefined interfaces
    - API, Application Programming Interface
  - SQL instructions are passed as parameters to the functions of the host program
  - there is no concept of precompilation
- The host program directly contains the calls to the functions made available by the API

Call Level Interface

- There are different solutions of type Call Level Interface (CLI)
  - standard SQL/CLI
  - ODBC (Open DataBase Connectivity)
    - proprietary solution by Microsoft of SQL/CLI
  - JDBC (Java Database Connectivity)
    - solution for the Java World
  - OLE DB
  - ADO
  - ADO.NET

Mode of use

- Indipendently from the CLI solution adopted, there is a common structure of interaction with the DBMS
  - opening the connection with the DBMS
  - execution of SQL instructions
  - closing the connection

Interaction with DBMS

1. Call to an API primitive to create a connection with the DBMS

Interaction with DBMS

1. Call to an API primitive to create a connection with the DBMS
2. Send an SQL instruction over the connection

Interaction with DBMS

1. Call to an API primitive to create a connection with the DBMS
2. Send an SQL instruction over the connection
3. Receive a result in response to the sent instruction
   - In case of a SELECT, a set of tuples

Interaction with DBMS

1. Call to an API primitive to create a connection with the DBMS
2. Send an SQL instruction over the connection
3. Receive a result in response to the sent instruction
   - In case of a SELECT, a set of tuples
4. Elaborate the result obtained
   - There are predefined functions to read the result

Interaction with DBMS

1. Call to an API primitive to create a connection with the DBMS
2. Send an SQL instruction over the connection
3. Receive a result in response to the sent instruction
   - In case of a SELECT, a set of tuples
4. Elaborate the result obtained
   - There are predefined functions to read the result
5. Close the connection at the end of the work session

Interaction with DBMS

- ODBC (Open DataBase Connectivity)
  - Standard access method towards a database
  - Goal: making the access protocol to the database indipendent from the type of database used
  - PHP makes available to the programmer a library that allows to access via ODBC to a database
- Access method aimed at a specific DBMS
  - MySQL, Postgres, Microsoft SQL server, …
  - PHP makes available to the programmer specific libraries for most DBMS

SQL for the applications

MySQL functions for PHP

MySQLi extension

- MySQLi (MySQL improved) is an extension of PHP that allows to interface with a MySQL DB in an efficient way
- Supported functionalities
  - Connection to the DB
  - Execution: immediate or prepared (SQL instructions previously used and kept in cache for successive calls) of SQL query
  - Acquisition and reading of data
  - Support for stored procedures, multiple queries and transactions

Creation of a connection

- Call to the mysqli_connect() function
  - Needs four parameters: "**hostname**" (name of the machine that hosts the DBMS MySQL to which you want to connect), "**username**", "**password**", "**dbname**" (name of the DB)
  - In case of success it returns a MySQL **connection ID**, otherwise it returns **FALSE**

- Example:

```
//Connection to MySQL through mysqli_connect()
$con = mysqli_connect('localhost','joe','xyz','dbname');
```

Creation of a connection

- Example with possible connection error handling
  - die(): arrests the execution of the script and prints a message
  - mysqli_connect_errno(): returns the connection error code
  - mysqli_connect_error(): returns the connection error

```
if (mysqli_connect_errno())
  {
  die ('Failed to connect to MySQL: ' . mysqli_connect_error());
  }
```

Closing a connection

- Must be executed when interacting with the DBMS is not necessary anymore
  - Closes the connection with the DBMS and releases the relative resources
- Call of the mysqli_close() function
  - Parameter (optional): ID of the connection
  - If no parameter is specified, the most recently opened connection is closed

```
//closing the connection
mysqli_close($con);
```

Execution of SQL instructions

- Immediate execution of the instruction
  - The server compiles and executes immediately the SQL instruction received
- "Prepared" execution of the instruction
  - The SQL instruction
    - is compiled (prepared) only once and its execution plan is stored by the DBMS
    - is executed many times during the session

      Useful when you need to execute the same SQL instruction multiple times in the same work session
    - changes only the value of some parameters

Immediate execution

- Call to the mysqli_query() function
  - Needs the **connection ID** and the **query** to execute as parameters, in a string format
  - In case of success, it returns the **result of the query**, otherwise it returns **FALSE**
  - mysqli_error(): returns the text of the error related to the Mysql function most recently executed

- Example:

```
/* QUERY SQL */
$sql   = "SELECT author.surname, paper.name,
          FROM author, paper
          WHERE author.paper = paper.author";
$result = mysqli_query($con,$sql);

if( !$result )
  die('Query error: ' . mysqli_error($con));
```

"Prepared" execution

- Logical steps
  1. Preparation of the query
  2. Assignment of the values to the parameters of the query
  3. Execution of the query
  4. Possible repetition of steps 2. and 3. with different values

Preparation of the query

- Call to the *mysqli_prepare()* function
  - Needs the **connection ID** and the **query** to execute as parameters, in string format
  - The parameters inside the query are labeled with a '?'
  - The function sends the query to MySQL that checks its validity and correctness
  - In case of success returns an **objcet of type mysqli_stmt**, otherwise it returns **FALSE**

Binding of parameters to the query

- Before executing the query you need to link each parameter to its corresponding variable ("binding" operation)
- Call to the *mysqli_stmt_bind_param()* function
  - Needs as parameters the object returned by msqli_prepare(), the data types and the variables that need to be assigned to the parameters of the query
  - In case of success it returns TRUE, otherwise it returns FALSE

Example of binding

```
//preparation of the query
$stmt = mysqli_prepare($con, "INSERT INTO Forniture VALUES (?,?,?)");

$CodF= "F1";
$CodP= "P1"
$Qta= 100;

//parameter binding
mysqli_stmt_bind_param($stmt, "ssi", $CodF, $CodP, $Qta);
```

- Type of parameter
  - "s": string
  - "i": integer number
  - "d": real number

Execution of the "prepared" query

- Call to the *mysqli_stmt_execute()* function
  - Needs the object returned by mysqli_prepare() as parameter
  - In case of success it returns TRUE, otherwise it returns FALSE

```
//preparation of the query
$stmt = mysqli_prepare($con, "SELECT Province FROM City WHERE name=?");
if (!stmt)
        die ('Query error; ' .mysqli_error());

$name= "Turin";

//parameter binding
mysqli_stmt_bind_param($stmt, "s", $name );

//execution of the query
mysqli_stmt_execute($stmt);
```

Reading the result

- The result of the mysqli_query() function is stored in a variable of type "resource"
  - A special variable, that contains the reference to an external resource
- Reading the result is done row by row: a cycle made of two phases
  - Acquisition of a row from the table (using a cursor)
  - Reading of the acquired line

Name	Num
Andrew	2
Gabriel	2

Cursor

Acquisition of a row

- There are many possibilities to acquire a row from the table
- Call to the *mysqli_fetch_row()* function
  - Needs as parameter the resource returned by mysqli_query()
  - Returns the array corresponding to the current row, or FALSE in case there are no rows available
  - Each column of the result is stored in an element of the array, starting from index "0"

```
while ($row = mysqli_fetch_row($result)) {
      ...
      }
```

Acquisition of a row

- Call to the *mysqli_fetch_assoc()* function
  - Needs as parameter the resource returned by mysqli_query()
  - Returns the associative array corresponding to the current row, or FALSE in case there are no rows available
  - Each column of the result is stored in an element of the associative array, with the keys defined by the field names
  - A numeric index is not defined

Acquisition of a row

- Call to the *mysqli_fetch_array()* function
  - It's the most generic function
  - Needs as parameter the resource returned by mysqli_query() and the type of array to be returned (scalar, associative or both)
  - MYSQLI_ASSOC: the resulting array is associative
  - MYSQLI_NUM: the resulting array is associative
  - MYSQLI_BOTH: the resulting array is accessible both with a numeric index and with a key corresponding to the field name

Examples

Name	Num
Andrew	2
Gabriel	2

```
while ($row = mysqli_fetch_row($result)) {
        echo "<tr>";
        echo "<td>$row[0]</td><td>$row[1]</td>";
        echo "</tr>";
}
```

```
while ($row = mysqli_fetch_row($result)) {
        echo "\t<tr>\n";
        foreach ($row as $cell) {
                echo "\t\t<td>$cell</td>\n";
        }
        echo "\t</tr>\n";
}
```

Examples

Name	Num
Andrew	2
Gabriel	2

```
while ($row = mysqli_fetch_assoc($result)) {
        echo "<tr>";
        echo "<td>" . $row["Name"] . "</td><td>" . $row["Num "] . "</td>";
        echo "</tr>";
}
```

Other useful functions

- int *mysqli_num_rows(resource $result)*
  - Returns the number of rows of the $result resource, or FALSE otherwise

```
if ( mysqli_num_rows($result) <=0) {
        echo "<h5>    No result!    </h5>";
}
else {
// access to the rows of the result
...
}
```

Other useful functions

- int mysqli_num_fields(resource $result)
  - Returns the number of fields (attributes) of the $result resource, or FALSE otherwise
- string mysqli_fetch_field(resource $result)
  - Returns the next column as an object. To obtain its name you need to select the "name" property

```
for ($i = 0; $i < mysqli_num_fields($result); $i++){
  $title  =  mysqli_fetch_field($result);
  $name = $title->name;
  echo "<th> $name </th>";
}
```

Other useful functions

```
if( mysqli_num_rows($result) > 0 ){
  // Table heading
  echo "<table border=1 cellpadding=10>";
  echo "<tr>";
  for ($i = 0; $i < mysqli_num_fields($result); $i++){
    $title  =  mysqli_fetch_field($result);
    $name = $title->name;
    echo "<th> $name </th>";
  }
  echo "</tr>";
  // Table filling
  while ($row = mysqli_fetch_row($result)) {
  ...
  }
```

Transactions

- Connections implicitly happen in auto-commit mode
  - After the successful execution of each SQL instruction, a commit is automatically executed
- When it's necessary to execute commits only after the successful execution of a sequence of SQL instructions
  - The commit needs to be managed in a non-automatic way
  - Only a single commit is executed at the end of the execution of all instructions

Transaction management

- bool mysqli_autocommit (mysqli $link , bool $mode)
  - Enables or disables the auto-commit mode
  - Needs as parameters the **connection ID** and **TRUE or FALSE** depending on whether you want to enable or disable auto-commit mode
  - In case of success it returns TRUE, otherwise it returns FALSE

Transaction management

- If autocommit is disabled the operations of commit and rollback must be executed explicitly
- bool mysqli_commit (mysqli $link)
  - Executes the commit of the current transaction
  - In case of success returns TRUE, otherwise it returns FALSE
- bool mysqli_rollback (mysqli $link)
  - Executes the rollback of the current transaction
  - In case of success returns TRUE, otherwise it returns FALSE