



The Kalmanovitz Library and  
The Center for Knowledge Management

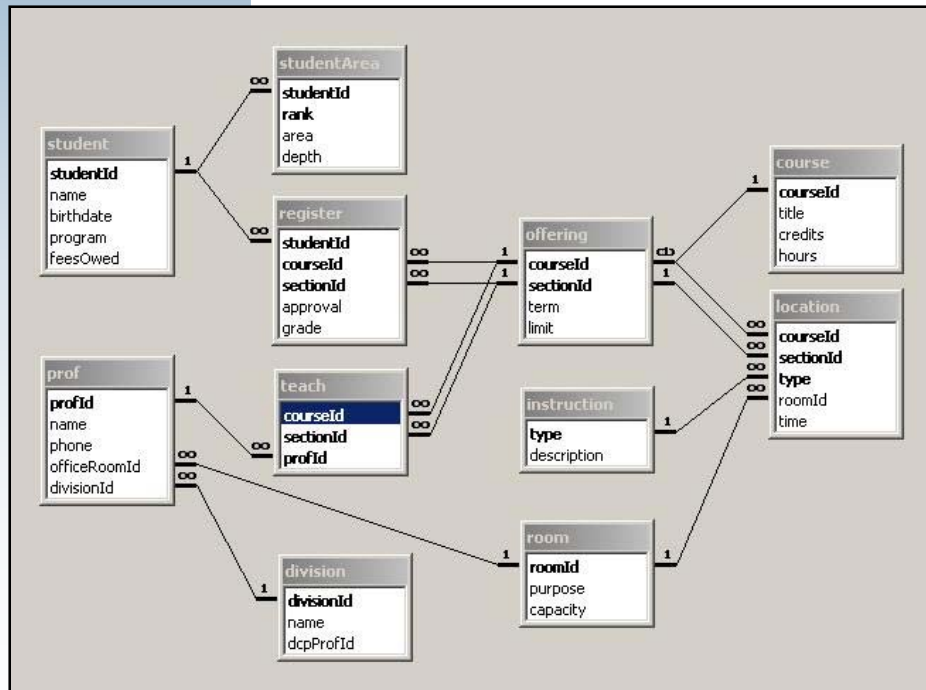
# *Perl DBI: Database Interface*

Gilberto da Gente  
Bioinformatics Resource Specialist  
Thursday, April 10<sup>th</sup> 2008  
University of California, San Francisco



# Relational Databases

- In 1970, E. F. Codd published a now classic paper, "A Relational Model of Data for Large Shared Data Banks,"



- The relational database model revolves around data storage units called tables,
- Tables have a number of attributes associated with them, called columns.

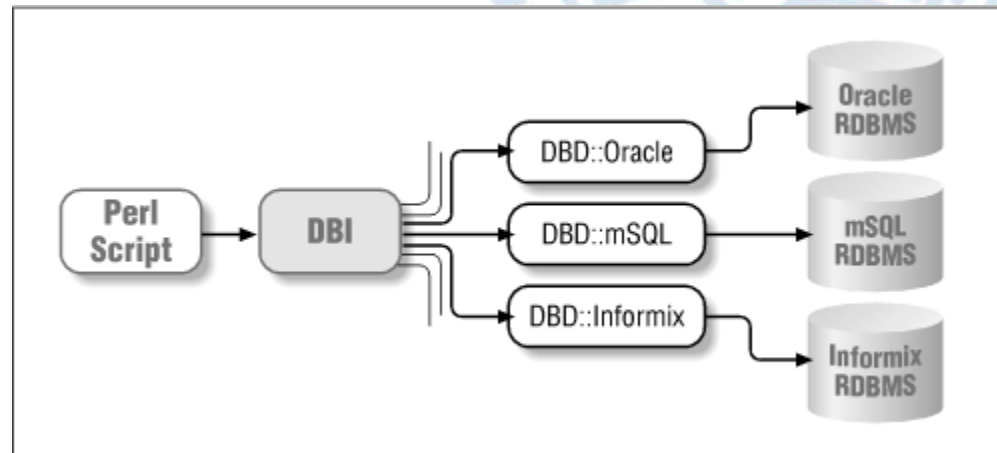
# *SQL: Sequence Query Language*

- Language that makes interacting with relational databases simple.
- All query languages implement four main operations with which you can manipulate the data.
  - Fetching
    - Select Statement
  - Storing
    - Insert Statement
  - Updating
    - Update Statement
  - Deleting
    - Delete Statement

# DBI

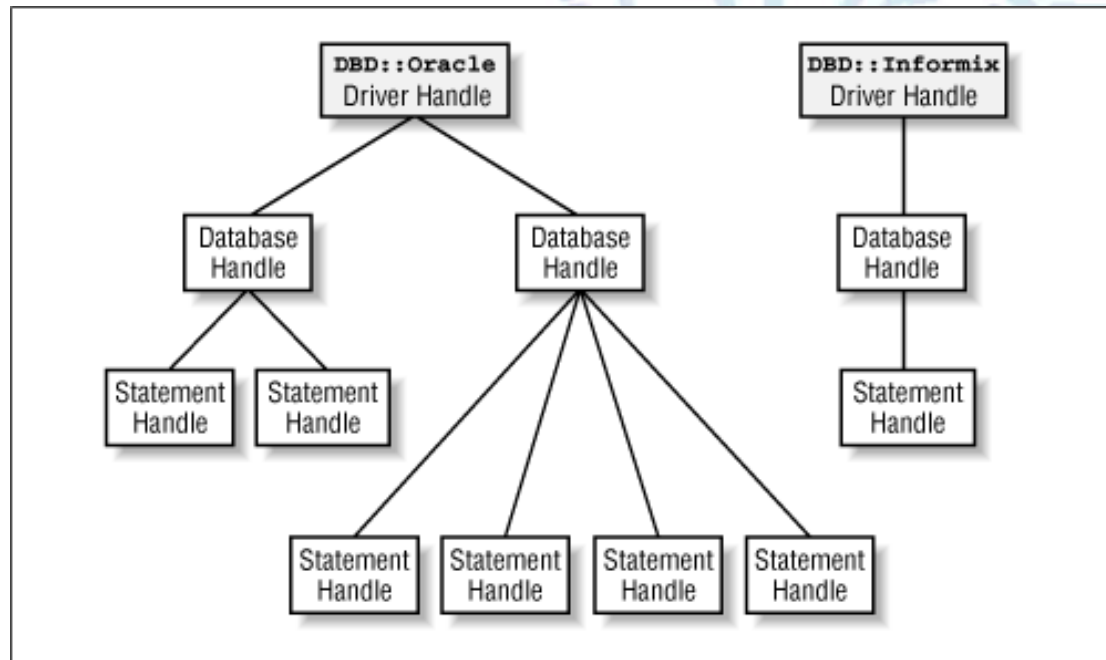
- "The DBI is the standard **database interface** module for Perl. It defines a set of methods, variables and conventions that provide a consistent database interface **independent of the actual database being used.**"

-- Tim Bunce



# DBI Handles

- DBI centers around three objects called handles
- Each object has its own functions
  - **Driver handles**
  - **Database handles**
  - **Statement handles**



# *Driver Handles*

- Although rarely seen or used overtly they are the main interface to particular databases and are initially called as a data source in the DBI connect method
- Driver handles represent loaded drivers and are created when the driver is loaded and initialized by the DBI.
- There is exactly one driver handle per loaded driver.
- There are two functions associated with driver handles
  - `data_sources()` , to enumerate what can be connected.
  - `connect()`, to actually connect to a database.

# *Database Handles*

- Database handles are the first step towards actually doing work with the database.
- They encapsulate a single connection to a particular database.
- `$dbh` is generally used to represent a database handle object

```
$dbh = DBI->connect  
( $data_source, $username, $password, \%attr );
```

# *Database and Driver functions*

- There are functions associated with driver and database handles that can elucidate the \$data source field

```
@drivers = DBI->available_drivers();
```

```
foreach $driver (@drivers) {  
  print DBI-  
  >data_sources($driver );}
```

# *Connect attributes*

```
$dbh = DBI->connect( $data_source, $username, $password, \%attr );
```

```
$data_source = "dbi:mSQL:database:hostname:port_number"
```

```
$data_source = "dbi:Oracle:connection_descriptor"
```

```
$data_source = "dbi:ODBC:description"
```

```
%attr = (  
  PrintError => 1,  
  RaiseError => 0);
```

```
$dbh->{PrintError} = 1;  
$dbh->{RaiseError} = 0;  
$dbh->{AutoCommit} = 1;
```

# *quote()*

- This method correctly quotes values as literal strings within your SQL statement before it is issued to the database.
- Correctly quotes and escapes SQL statements in a way that is suitable for a given database engine.
- A good DBI method to remember when building SQL statements.

```
### Escape the string quotes ...  
my $quotedString = $dbh->quote( $string );
```

```
my $sth = $dbh->prepare($quotedString );  
$sth->execute();
```

```
exit;
```

```
SELECT *  
FROM media  
WHERE description = $quotedString
```

# *Statement Handles*

- Statement handles are the final type of object that DBI defines for database interaction and manipulation.
- These handles encapsulate individual SQL statements to be executed within the database.
- `$sth` is used as a statement handle object

# *Perl DBI Process*

- Connect to a Database
- Process SQL statements
- Disconnect from a Database

# Connecting

- The data source name, a string containing information specifying the driver to use, what database you wish to connect to, and possibly its whereabouts.
  - This argument is highly database-specific.
- The username that you wish to connect to the database as.
- The password associated with the supplied username.
- The final argument, `\%attr`, is optional and may be omitted.
- The syntax for connecting to databases using DBI is to use the `connect()` call, defined as follows:

```
$dbh = DBI->connect( $data_source, $username, $password);
```

# *Processing SQL Select statements*

Statements are handles in a three step process

- Prepare
  - Input SQL statement in `$dbh->prepare()` function which returns a statement handle `$sth` object
- Execute
  - Run statement handle function `execute()` on `$sth` object produced from previous prepare function
  - `$sth->execute();`
- Fetch
  - Retrieve data one row at a time from `$sth` object using `$sth>fetchrow_array`

# *\$dbh->prepare()*

```
$sth = $dbh->prepare("SQL Select Statement");
```

## Simple Queries

```
SELECT column, column, ..., column  
FROM table
```

```
SELECT *  
FROM table
```

## Condition Queries

```
SELECT column, column, ..., column  
FROM table  
WHERE column
```

```
WHERE column =  
WHERE column !=  
WHERE column >  
AND <  
WHERE column IN  
WHERE column LIKE
```

# *\$sth->execute();*

- Informs the database to go ahead and execute the SQL statement that you have prepared.
- This execution stage will actually tell the database to perform the query and begin to collect the result set of data.
- A true value will be returned from a correct execute() call. Otherwise, a value of undef is returned, signifying that the execution has failed.
- If PrintError is enabled an error message will be generated.

# *\$sth->fetchrow\_array*

- The general way in which we fetch data from the database's result set is to loop through the records returned via the statement handle.
- Each row is processed until no rows are left to fetch.
- The fundamentally important thing to remember is that the fields in the result set are in the order in which you asked for the columns in the SQL statement.

```
while ( ( $name, $type ) = $sth->fetchrow_array ) {  
    print "the name field is $name and the type is $type\n";}
```

```
while (@row = $sth->fetchrow_array ) {  
    print "$row[0] and $row[1]\n";}
```

# *Other Methods of Access*

- In addition to fetching a row in the form of a simple list of values, You can also fetch with a reference to an array of values, or a reference to a hash of field-name/value pairs.
- All essentially retrieve the current row from the cursor, but return the data to your Perl program in different formats.

```
$array_ref = $sth->fetchrow_arrayref
```

```
$hash_ref = $sth->fetchrow_hashref;
```

# *dump\_results()*

- For fetching all of the rows in a statement handle's result set and printing them out.

```
$sth->dump_results( );
```

```
$rows = $sth->dump_results( 80, "\n", ':', \*FILE );
```

1:	Maximum Field Length	-	35
2:	Line Separator	-	"\n"
3:	Field Separator	-	","
4:	Output file handle	-	STDOUT

# *Processing other SQL statements*

- Insert

```
INSERT INTO table VALUES ( 'String1', 'String2', 'String3',)
```

- Delete

```
DELETE FROM table
```

```
DELETE FROM table WHERE  
column LIKE '%%'
```

- Update

```
UPDATE table  
SET column = 'String'  
WHERE column2 = 'String2'
```

# *\$dbh->do()*

- The do( ) method supplied by the DBI makes executing non-SELECT statements much simpler than repeatedly preparing and executing statements.
- This is achieved by simply wrapping the prepare and execute stages into one composite method.
- In most cases you will be inputting data into the database so make sure to use the DBI quote() function
- Ideally the do() function should not be used in loops.

```
$dbh->do( "INSERT INTO table VALUES ( String )" );
```

# *Disconnecting*

- disconnect() is invoked against a specific database handle.

```
### Perform the connection using the Oracle driver
my $dbh = DBI->connect( "dbi:Oracle:archaeo", "username",
"password" , {
    PrintError => 0
})
or die "Can't connect to Oracle database: $DBI::errstr\n";
```

```
### Now, disconnect from the database
$dbh->disconnect
or warn "Disconnection failed: $DBI::errstr\n";
```