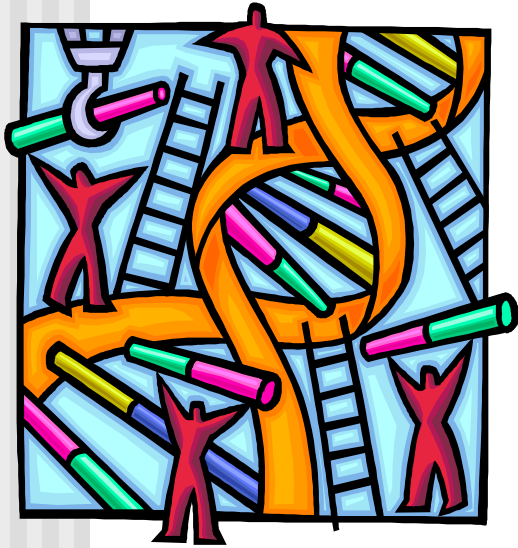


# Computing Concepts for Bioinformatics

---



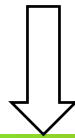
- More SQL
- Interacting with your database using a web interface
- **Integrating PERL & MySQL**

<http://amadeus.biosci.arizona.edu/~nirav>

# Where do we stand ...



You are here



Good Practices

Bad Practices

We will be compromising on quality and style  
to get the job done !

# For the lazy ones !

- Its painful to type `mysql -u -p blah blah` (lazy folks raise your hands)
- Lets create a file `.my.cnf` with connection details in it.
- Make sure you are in your "home" directory (how do you do that ?)
- Use `nedit` to create the file:  
**nedit .my.cnf**
- **Add the following lines:**  
`# Mysql defaults`  
`# for my db account`  
`[client]`  
`user=igertXX`  
`database=igertXX`  
`password=XXXXXX`
- `chmod og-r .my.cnf`  
(Why are we doing this)



# More tables, select

- We need to import a dataset which consist of:
  - ID (number)
  - Sequence Name (alpha numeric)
  - Notes (alpha numeric)
  - Date
- View the file (use more)  
`/home/student/eeb/sample/data/notes.csv`
- Design a table for it and load the data
- **create table notes ( id INT NOT NULL auto\_increment, seq\_name char(30), details char(200), worked\_on date, PRIMARY KEY (id));**
- **mysql> load data local infile**  
**"/home/student/eeb/sample/data/notes.csv"**  
**into table notes fields terminated by ',';**

# Autoincrement and date

---

- The AUTO\_INCREMENT attribute can be used to generate an unique identity for new rows
- Date field is given as YYYY-MM-DD
- `mysql> select max(id) from notes;`
- `mysql> insert into notes  
(seq_name,details,worked_on) values  
( 'seq-001', 'Test Sequence', '2003-11-6');`
- `mysql> select max(id) from notes;`  
(what answer did you get ?)

# More Select

---

- `mysql> select seq_name from notes;`
- `mysql> select distinct seq_name from notes;`  
(Avoids duplicate seq\_name)
- `mysql> select distinct seq_name,id from notes;`  
(why is the result different ?)
- `mysql> select seq_name,worked_on from notes;`
- `select seq_name,worked_on from notes order by worked_on;`  
(Organizes the rows !)
- `mysql> select seq_name,worked_on from notes ORDER BY worked_on DESC;`  
(Reverse order ..descending)

# More Date SQL

---

- `select seq_name,worked_on from notes where worked_on > '1999-01-01' ORDER BY worked_on DESC;`
- `select seq_name,worked_on,month(worked_on) from notes where worked_on > '1999-01-01' ORDER BY worked_on DESC;`
- `mysql> select seq_name, worked_on, month(worked_on) from notes where year(worked_on) > 1995 ORDER BY worked_on DESC;`

# Select (like)

- `mysql>select seq_name,details from notes where details LIKE '%A%'\G`
- `mysql> select seq_name,details from notes where seq_name LIKE 'A20%'\G`
- `mysql>select seq_name,details from notes where details LIKE '%4'\G`  
(What do you get ?)
- `%A%` will match anything before `A` and after it
- MySQL supports REGEXP
- To find a something ending in 4  
`mysql>select seq_name,details from notes where seq_name REGEXP '4$'\G`

# Getting data from multiple tables

- We have 2 tables: analysis and notes
- The common element is the seq\_name
- **mysql> select** p1.seq\_name , p1.start ,  
p1.stop , p2.details  
**from** *analysis* **as** p1 , *notes* **as** p2  
**where** p1.seq\_name = p2.seq\_name  
**and** p1.gene\_present = 'yes';
- We alias the table so that we do not have to call each column as analysis.seq\_name we call it p1.seq\_name
- More about Joins later

# Updating data

---

- Updates everything in the table analysis
- `mysql> update analysis set start =start +2 ,stop=stop -2 ;`
- Updates specific rows
- `mysql> update analysis set start = start +2 , stop = stop -2 where seq_name = 'W13950';`

# SQL Exercise

---



- I have performed GC% analysis on a set of sequences
- Load the resultant analysis into a table called `gc`
- Will have 2 columns  
`seq_name`  
`gc_content`
- The data file is on a website and is in csv format
- Find sequences where `gc > 75%`

# How are you going to do it ?

- Yes you can use the web interface <http://amadeus.biosci.arizona.edu/mysql/>

- [Print view](#)
- [Data Dictionary](#)
- Create new table on database nirav :

Name :

Fields :

## Database *nirav* - Table *gc* running on *localhost*

Field	Type <a href="#">[Documentation]</a>	Length/Values*	Attributes	Null	Default**	Extra	Primary
seq_name	VARCHAR	30		not null			<input checked="" type="radio"/>
gc	FLOAT			not null			<input type="radio"/>

Table comments :

Table type :

# Loading the data

---

- Using browser go to
- <http://amadeus.biosci.arizona.edu/~nirav/class-2005/eeb/gc.csv>
- Save the file to disk
- Click on the table name “gc” (not the icon) and scroll in the right panel all the way down to “insert data from text file” and select the link

- [Insert data from a textfile into table](#)

# Importing

GC content

Location of the textfile	<input type="text" value="f:\gc.vsv"/> <input type="button" value="Browse..."/>	
Replace table data with file	<input type="checkbox"/> Replace	The contents of the file replaces the contents of the selected table for rows with identical primary or unique key.
Fields terminated by	<input type="text" value=","/> (circled in red)	The terminator of the fields.
Fields enclosed by	<input type="text" value=""/> <input type="checkbox"/> OPTIONALLY	Often quotation marks. OPTIONALLY means that only char and varchar fields are enclosed by the "enclosed by"-character.
Fields escaped by	<input type="text" value="\"/>	Optional. Controls how to write or read special characters.
Lines terminated by	<input type="text" value="\r\n"/>	Carriage return: \r Linefeed: \n
Column names	<input type="text"/>	If you wish to load only some of a table's columns, specify a comma separated field list.
LOAD method	<input type="radio"/> ...DATA <input checked="" type="radio"/> ...DATA LOCAL	The best method is checked by default, but you can change it if it fails.

[\[Documentation\]](#)

# Finding $gc > 0.75$ (search)

## Database *nirav* - Table *gc* running on *localhost*

Structure

Browse

SQL

Search

Insert

Export

GC content

Select fields (at least one): :

seq\_name  
gc

- Number of rows per page
- Add search conditions (body of the "where" clause):

[\[Documentation\]](#)

Or Do a "query by example" (wildcard: "%")

Field	Type	Function	Value
seq_name	varchar(30)	LIKE	<input type="text"/>
gc	float	>=	<input type="text"/>

- Display order:  
  Ascending  Descending

Go

# PERL and MySQL: Why ?

---



- For fun
- Need to spice up available Friday nights !
- SQL is limited
- To connect data outside of mysql is hard without perl
- Many reasons !

# DBI/DBD

---

- DBI is “Database Interface”
- DBD is “Database Driver”
- We will be using the MySQL DBD to connect to our database and manipulate data stored in the tables
- The **DBI** is a 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
- You can get DBD drivers for Oracle, Sybase, Postgres, MySQL to name a few

# DBI: Connecting to a database options

---

- This works like a file handle (do you remember ?)  
Once the handle is created you can read and write to it.
- Create a database handle called \$dbh  

```
$dbh = DBI->connect( "DBI:mysql:igert59_db", "igert59", "12D3f4",  
{ RaiseError => 1 } ) or die("Connect error: $DBI::errstr");
```
- **DBI:mysql:igert99** - Data Source Name (DSN)  
DBI - DBI (necessary first part of the DSN)  
mysql - Name of DBD  
igert99 - Name of database we are connecting to  
igert59 - User we are connecting to this database as  
12D3f4 - Password for the user  
RaiseError , error: \$DBI::errstr – Checks for connection

# DBI: Prepare, Execute

---

- Now that you have the DBH and you are connected you can read and write to it.
- Define the SQL statement  
`$sql = "SELECT seq_name,details FROM notes WHERE details LIKE '%A% '";`
- Prepare the statement for execution  
`$sth = $dbh->prepare( $sql );`
- `$sth` is called the statement handle
- Now execute the handle  
`$sth->execute;`
- Once you execute the handle you can read results from it

# DBI: Prepare, Execute

---

- What is the use for PREPARE and EXECUTE
- This very important in handling large volumes of data.
- You can *prepare* a statement once and *execute* it many times with different parameters/value

- Example: A text file containing gene names,start,stop

```
$sql = "INSERT INTO gene_details  
  ( gene_names,start,stop) VALUES ( ?, ?, ? )";  
$sth = $dbh->prepare( $sql );  
open( INP, "gene.txt" ) or die "Cannot open gene file: $!";  
while ( $in=<INP> ) {  
    chomp ($in);  
    ( $gene_names, $start, $stop ) = split (/,/,$in);  
    $sth->execute($gene_names, $start, $stop );  
}  
close(INP);
```

# DBI: one step prepare & execute

---

- You can prepare and execute in one step using the do statement
- `$sql = "update genes set stop='12' where gene_name = 'ILK9' " ;`  
`$run = $dbh->do($sql) ;`
- `$run` will have a non zero value if sql statement was successful (undef if an error occurred)
- For most databases `$run = number of rows modified`
- `if ( $dbh->do( $sql ) )`  
`{ ...continue on with next action... }`  
`else { ...error with $sql... }`

# DBI: Accessing returned data

---

- To determine number of rows returned:  
`$rc = $sth->rows;`
- Fetchrow methods get the next row of data and returns a reference to an array of field values. If there are no more rows to fetch then it returns undef.
- There are 4 fetchrow methods
  - **fetchrow\_array**
  - **fetchrow\_arrayref**
  - **fetchrow\_hashref**
  - **fetchall\_arrayref**

# DBI: Fetchrow

---

- `while(@row = $sth->fetchrow_array) {  
 print "$row[0] $row[1]\n"; }  
# Above method is efficient but hard to read`
- `while($row = $sth->fetchrow_arrayref) {  
 print "$row->[0] $row->[1]\n"; }  
# Above method is efficient but hard to read`
- `while($row_hash = $sth->fetchrow_hashref) {  
 print "$row_hash->{gene} $row_hash->{start}\n"; }  
# This is NOT efficient but easy to read`
- ✓ To make the first two methods more readable use `bind_columns`  
`$sth->bind_columns(undef, \ $gene, \ $stop);`  
`$sth->execute;`  
`while($sth->fetchrow_arrayref) { print " $gene $stop\n"; }`
- ✓ Since you know **number of variables coming back** from your query ...always `bind_columns` accordingly

# DBI: Closing handles

---

- When done with your prepared statement  
`$sth->finish;`
- When done with your database handle  
`$dbh->disconnect;`
- Very important when you are doing 1000's of iterations (do it even if you have 1 !)

# DBI: hands on

---

- Write a program to query tables you have created earlier
- On amadeus create directory **class9**
- In class11 create file called **dbi.pl**
- The PERL code is at (don't type blindly):  
<http://amadeus.biosci.arizona.edu/~nirav/dbi.html>
- Save and execute it (chmod +x dbi.pl)
- Once it works, try other SQL statements (URL is listed on the page)
- Print number of rows fetched at the end (make it readable)

# Practice

---

Please Go through tutorial at:  
<http://sqlzoo.net/>