Intro to Databases

A database is an integrated collection of data

A database management system (DBMS) is the software that les you use your data by allowing you to insert, retrieve, modify, or delete records

Structured Query Language (SQL) is used to make queries, or to request specific information from the database

Open Database Connectivity (ODBC) is a technology to allow generic access to different database systems

Perl and databases

A programming language connects to a relational database with an interface – software that allows communications between a DBMS and a program

DBI (Perl Database Interface) is a database-independent interface that allows Perl to interact with databases

Relational databases

A relational database is made up of tables, each with a connection, or relationship to one or more tables

Tables are made up of rows and columns of information

- Each row of data is called a record
- Each column can be referred to as a field

A primary key is a field (or fields) in a table that contain unique data that cannot be in other records

Queries are a set of commands that allow the programmer to manipulate, alter or combine data to achieve different sets of data

SQL

SQL provides a set of commands that enable a programmer to define queries that select and return data from a table. These commands are called queries.

Queries can

- Return all or part of the data from a table
  - All or some of the fields
  - All or some of the records
- Add data into a table
- Update/alter existing data in a table
- Complex queries can join two or more tables together

Select statement

Select statements allow the user to get information from one or more tables

- Allows the programmer to select any or all fields
- Allows the programmer to set criteria to determine which records are returned
- Allows the programmer to sort the returned data

SELECT field1, fields2, … FROM table

SELECT field1, fields2, … FROM table

SELECT fields FROM table ORDER BY field1, field2, …

If you use * instead of specifying specific fields all fields in the table will be returned

- SELECT * from table

If you specify fields the info is returned in the order requested regardless of the order of the fields in the database. If you use the * the fields are returned in the order in which they appear in the database. If the structure of the database changes using the * for the fields could cause unpredictable results

WHERE clause

The WHERE clause in a SELECT query is used to specify selection criteria for the query. The WHERE clause will physically be written after the FROM clause

SELECT fields FROM table WHERE criteria

The WHERE clause uses boolean conditions that can be checked against each record to determine whether to include that particular record. The boolean conditions can contain the <, >, <=, >=, =, <> and LIKE operators. The boolean condition will generally compare a table field against some value.

The boolean condition must be in parenthesis

LIKE operator

When searching for string, or text data the LIKE operator can be used. The LIKE operator is used for pattern matching with the wildcard characters % and ?.

- The % symbol in the pattern matches zero or more characters at that position in the pattern.
- In some versions of SQL the * is used instead of the % symbol.
- The ? symbol matches any single character at that position in the pattern.

Text data must be in single quotes

- Some versions of SQL will allow text data to be in double quotes
- Some versions of SQL support regular expressions to different degrees
  - MySQL and MS Access do not support regex's

Fields used in a WHERE clause do not have to be included in the fields returned (but must be fields within the database)

Two or more boolean conditions can be included in a single WHILE clause if each individual boolean condition is in parenthesis. The individual conditions are joined by a AND or and OR

- WHERE ((LastName LIKE 's*') AND (State LIKE "ND"))
- WHERE ((State LIKE "ND") OR (State LIKE "MN"))

ORDER BY clause

The results of a query can be sorted into ascending or descending order by using the ORDER BY clause. The ORDER BY

SELECT fields FROM table ORDER BY field ASC
SELECT fields FROM table ORDER BY field DESC

ASC specifies an ascending sort order
DESC specifies a descending sort order
ASC is the default order if no order is specified

Multiple sort fields can be specified to determine the sort order

Each field must be separated by commas
Each field can be followed by ASC or DESC

SELECT fields FROM table ORDER BY field1 ASC, field2 DESC, field3, field ASC

Fields used in a ORDER BY clause do not have to be included in the fields returned (but must be fields within the database)

INSERT statement

Insert statements allow the user to add information to a table

INSERT INTO table (comma delimited field names) VALUES (information to be added to corresponding field)

INSERT INTO tblStateInfo (state, abbreviation, capital) VALUES (‘North Dakota’, ‘ND’, ‘Bismarck’);

To insert without listing the field names

INSERT INTO table VALUES (comma separated values for ALL fields in the order that the fields appear in the table);

- Null may be used if no value is needed or should be listed (such as auto-increment fields)

Text must be placed in single quotes, or double quotes depending upon the implementation of SQL

UPDATE statement

Update statements allow the user to change existing information in a table

UPDATE table SET (field1 = value1, field2 = value2) WHERE criteria

UPDATE tblnames SET LastName = ‘Whatever’ WHERE (ID = 100);

Text must be placed in single quotes, or double quotes depending upon the implementation of SQL

Perl DBI

The Perl Database Interface (DBI) module is a package that allows the programmer to access relational databases from with a Perl script.

The DBI provides the means for writing code that is database vendor independent

- DBI modules hide a certain level of function calls from the programmer

The DBI module sends requests to a particular database driver (DBD) module. The DBD module handles the processing of requests to/from the database. Each database requires its own driver.

- Some of the Perl DBD's are
- DBD::Oracle
- DBD::Sybase
- DBD::MySQL

Common Perl DBI variables

Conventional Perl DBI handle variable names

- $dbh – A handle to a database object
- $sth – A handle to a statement (query) object
- $fh – A handle to an open file
- $h – A “generic” handle; the meaning depends on context

Conventional Perl DBI non-handle variable names

- $rc – The return code from operations that return true or false
- $rv – The return value from operations that return an integer
- $rows – The return value from operations that return a row count
- $ary – An array (list) representing a row of values returns by a query

Telling the compiler about DBI

Before invoking anything DBI-related in the script the line

use DBI;

This must be in the script before any other DBI-related code

- use DBI tells the Perl interpreter that it needs to use the DBI module
- You don’t have to specify which DBD-level module you want

Connecting to the database

To actually connect to a database use the connect method

```
#connecting to the database
$sbh = DBI->connect($dataSource, $userName, $password, {
    PrintError => 0} );
```

Or

```
$dbh = DBI->connect($dataSource, $userName, $password) or die
    “Can’t connect to database\n”;
```

If the connect call succeeds it returns a database handle, which is assigned to $dbh

If the connect call fails it normally returns undef

- If the { PrintError => 0 } parameter is used and the connect fails nothing is returned to $dbh but no error message will be printed – the script will not exit

Setting up the data source

The data source (often called a data source name, or DSN) formats are determined by the requirements of the particular DBD module you are using

For the MySQL driver formats include either of the following

- “DBI:mysql:databaseName”
- “DBI:mysql:databaseName:hostName”

In the first format the host name defaults to localhost.

The case of DBI is irrelevant

The case of mysql must be lower case

If there isn’t a user name or password use empty strings for the arguments

The optional argument controls DBI’s error-handling behavior

Preparing/executing a query

prepare( ) is database method that returns a statement handle that can be executed

$sth = $dbh->prepare ($sql)

To actually run the query use the execute( ) method. This executes a prepared statement.

- Returns true if the statement executed successfully
- Returns undef if an error occurred

$sth->execute( );

Getting the data

@array = $sth->fetchrow_array ( )

Returns an array containing column values for the next row of the result, or an empty array if there are no more rows or if an error occurs

In a scalar context it returns the value of the first element of the array (first column of the row) or undef if there are no more rows or if an error occurs

$arrayRef = $sth->fetchrow_arrayref ( )

$arrayRef = $sth->fetch ( ) - same as $sth->fetchrow_arrayref ( )

Returns a reference to an array containing column values for the next row of the result. It returns undef if there are no more rows or an error occurs

Getting the data

$arrayRef = $sth->fetchall_arrayref ( )

Fetches all rows from the statement handle $sth and returns a reference to an array that contains one reference for each row fetched

Each element in $arrayRef is a reference to an array containing the values for one row of the result

$hashRef = $sth->fetchrow_hashref ( )

Returns a reference to a hash containing column values for the next row of the result set. It returns undef if there are no more rows or an error occurs. Hash index (key) values are the column names, and elements of the hash (values) are the column values

Accessing database metadata

Additional information about a table may be accessed

Array with the names of all tables in the database

```
@tables = $dbh->tables();
```

Number of columns in a table

```
$num = $sth->{NUM_OF_FIELDS};
```

Getting field names returned by the SQL statement

```
#array ref with all field names
$fieldNameRef = $sth->{NAME};
#scalar with the first field name
$fieldName = $sth->{NAME}->[0]
```

Accessing more metadata

Getting the type of the fields

```
#array ref with all field types
$fieldNameRef = $sth->{TYPE};
#scalar with the type of the first field
$fieldName = $sth->{TYPE}->[0]
```

The standard values for common types are:

SQL_CHAR	1	SQL_LONGVARCHAR	-1
SQL_NUMERIC	2	SQL_BINARY	-2
SQL_DECIMAL	3	SQL_VARBINARY	-3
SQL_INTEGER	4	SQL_LONGVARBINARY	-4
SQL_SMALLINT	5	SQL_BIGINT	-5
SQL_FLOAT	6	SQL_TINYINT	-6
SQL_REAL	7	SQL_BIT	-7
SQL_DOUBLE	8	SQL_WCHAR	-8
SQL_DATE	9	SQL_WVARCHAR	-9
SQL_TIME	10	SQL_WLONGVARCHAR	-10
SQL_TIMESTAMP	11		
SQL_VARCHAR	12		

Changing the SQL statement

If a basic SQL statement is used more than once with just the search criteria changed bind_param can be used to avoid additional prepare statements

The ? can be used as a placeholder for a piece of data to be set after the prepare statement (to avoid additional executions of the prepare function). The data is set using bind_param. The first parameter is the column (column numbers are 1 based), the second parameter is the data

```
$sql = q { SELECT * FROM tblnames WHERE LastName LIKE ?};
$sth = $dbh->prepare ($sql) or die ("db Error: ", $dbh->errstr(), "\n");
$sth->bind_param (1, 's%');
$sth->execute() or die ("db Error: ", $sth->errstr(), "\n");
#deal with s% results
$sth->bind_param (1, 't%'); #or $sth->bind_param (1, 's%' SQL_VARCHAR);
$sth->execute() or die ("db Error: ", $sth->errstr(), "\n");
#deal with t% results
```

More on changing the SQL statement

Databases tend to do well in determining the type of information passed to bind_param but a type can be given by adding a third parameter. All the types mentioned earlier can be used. To use the types ‘use DBI qw(:sql_types);’ must be specified. Otherwise the numeric value must be used

```
#if use DBI qw (:sql_types)
$sth->bind_param (1, 's%', SQL_VARCHAR);  # if use DBI qw(:sql_types);
$sth->bind_param (1, $searchData, SQL_VARCHAR);
$sth->bind_param (1, 100, SQL_INTEGER);  # if substituting in the numeric value 100
#if use DBI
$sth->bind_param (1, 's%' 1);  # if use DBI;
```

When substituting text into a query the quotes around the text do not need to be added to the parameter, that is handled automatically by the bind_param function; that’s why the method needs to know the type - text data will be surrounded by single quotes, numeric data will not

Binding output columns

One of the more efficient ways of retrieving data from a database is to associate a Perl variable with a fetched value. You do this by binding fields to variables using the bind_col method or the bindcolumns method

bind_col binds a given column for a SELECT query to a Perl variable, which should be passed as a reference. The column number should be in the range from 1 to the number of columns returned by the query. Each time a row is fetched, the variable is updated automatically with the column value.

```
$sth->bind_col (1, \$id);
$sth->bind_col (2, \$lastname);
$sth->bind_col (3, \$firstname);
$sth->bind_col (4, \$city);
$sth->bind_col (5, \$state);
```

bind_columns binds a list of variables to columns returned by a SELECT query

```
$sth->bind_columns (\$id, \$lastname, \$firstname, \$city, \$state);
```

Both bind_col and bind_columns should only be called after a prepare( ) and execute( ) statement have been performed.

do statement

$rows = $dbh->do ($sql);

The do statement prepares and executes the query. The return value is the number of row affected; -1 is the number of rows if unknown and undef is returned if an error occurs. If the number of rows affected is zero the return value is the string “0E0” (which evaluates as zero but is considered true)

do() is used primarily for statements that do not retrieve rows, such as DELETE, INSERT, REPLACE, or UPDATE. If you try to use it for a SELECT statement you won’t get back a statement handle and you won’t be able to fetch any rows

Finishing up

disconnect( ) is a database method that terminates the connection associated with the handle. If a connection is still active when the script exits a warning is printed and the connection is terminated automatically

```
$dbh->disconnect( );
```

finish( ) is a statement method that frees any resources associated with the statement handle. You usually don't need to invoke this method explicitly but it lets DBI know you are done fetching data

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

General error methods

$handle->err( )

- Returns the numeric error code for the most recently invoked driver operation
- Returns 0 for no error

$handle->errstr( )

- Returns the string error messge for the most recently invoked driver operation
- Returns an empty string for no error