MySQL and SQL

Relational databases (RDB) are based on relational model (RM).

RM handles data as relations among strings, numbers and dates.

SQL is widely supported *Standard Query Language* for RDBs.

MySQL is popular RDB server supporting call level interface for SQL.

MySQL on *mysql-server-1* (anubis) can be remotely queried by clients

```
linux% mysql --user=hamish --database=hamish --password
Enter password: ?????????
Welcome to the MySQL monitor.  Commands end with ; or \g.

mysql> help;
```

Select all attributes of tuples from *STD* relation:

```
mysql> select * from STD;
+----+------------+------------+----------------------+-----+
| S  | SNAME      | BORN       | DEP                  | IQ  |
+----+------------+------------+----------------------+-----+
| S1 | P Smythe   | 1996-03-12 | Computer Science     | 119 |
| S2 | J Totten   | 1997-09-19 | Computer Science     | 125 |
| S3 | A Armitage | 1996-08-04 | Business Management | 130 |
| S4 | K Black    | 1996-03-30 | Physics              | 152 |
+----+------------+------------+----------------------+-----+
4 rows in set (0.01 sec)
```

Unique identifier of row of table is *primary key*.

Set of (allowed) values for attribute is its *domain*.

relation	STD
tuple	(S3, A Armitage, 1996-08-04, Business Management, 130)
attributes	{S, SNAME, BORN, DEP, IQ}
datatypes	{string, string, date, string, integer}
primary key	S
domain of DEP	{Business Management, Computer Science, Physics}

Selecting and Creating Tables

To retrieve some attributes of STD relation satisfying condition:

```
mysql> select SNAME, DEP from STD where DEP='Computer Science';
+----------+------------------+
| SNAME    | DEP              |
+----------+------------------+
| P Smythe | Computer Science |
| J Totten | Computer Science |
+----------+------------------+
2 rows in set (0.00 sec)
```

create table creates tables by

- defining names and types of attributes
- declaring whether values must be supplied or not

STD relation could be created by:

```
create table STD (
  S      char(5) not null,
  SNAME  char(20) not null,
  BORN   date,
  DEP    char(22) not null,
  IQ     integer,
  primary key (S) );
```

insert into populates tables. *STD* relation is given by

```
insert into STD values
  ('S1', 'P Smythe', '1996-03-12', 'Computer Science', 119);
insert into STD values
  ('S2', 'J Totten', '1997-09-19', 'Computer Science', 125);
insert into STD values
  ('S3', 'A Armitage', '1996-08-04', 'Business Management', 130);
insert into STD values
  ('S4', 'K Black', '1996-03-30', 'Physics', 152);
```

drop table removes a table from database:

```
mysql> drop table STD;
Query OK, 0 rows affected (0.00 sec)
```

File of SQL commands *cmd.sql* can be executed as batch job

```
mysql> source cmd.sql;
```

Backup data with command file - drop table, create table, inserts.

Updating and Joining Tables

update alters attribute values matching query:

```
mysql> update STD set BORN = '1996-02-29', IQ = 190
    -> where S = 'S4' ;
Query OK, 1 row affected (0.00 sec)
```

delete erases records such as those on Computer Science students:

```
mysql> delete from STD where DEP = 'Computer Science';
Query OK, 2 rows affected (0.00 sec)
```

Joins create new relations from 2 or more relations by

- naming relations in *from* clause
- *select*ing attributes from joined relations
- equating joined attributes with *where* clause

Department details might be stored as a *DEPT* relation:

DNAME	HEAD	SCHOOL
Physics	W.MacPherson	EPS
Computer Science	A.Ireland	MACS
Business Management	U.Bititci	SoSS

STD and *DEPT* relations can be joined

- to create new relation with attributes *SNAME* and *SCHOOL*
- whenever *DEP* field in *STD* equals *DNAME* field in *DEPT*

SQL query to join 2 relations and filter the result:

```
mysql> select SNAME, SCHOOL from STD, DEPT
    -> where STD.DEP = DEPT.DNAME and SCHOOL != 'MACS';
+------------+--------+
| SNAME      | SCHOOL |
+------------+--------+
| A Armitage | SoSS   |
| K Black    | EPS    |
+------------+--------+
2 rows in set (0.00 sec)
```

Node.js and MySQL

Install mysql module for driving and calling MySQL.

```
unix% npm install mysql
```

sqlquery.js queries a MySQL server about stored book data

```
var mysql = require('mysql');
var express = require('express');
var http = require('http');
var app = express();
var nl  = "\n";
var head = "<!DOCTYPE html>" + nl + "<html>" + nl + "<head>" + nl;
head += "<title>Books</title>" + nl + "</head>" + nl + "<body>" + nl;

var conn = mysql.createConnection({
  host: "mysql-server-1.macs.hw.ac.uk",
  user: "hamish", password: "?????????", database: "hamish"
});

conn.connect(function(error) {
  if (error) { console.log('error: ' + error.stack); }
});

app.get('/search', function (req, res) {
  var s = req.query.search.trim();
  if ( s.indexOf(";") != -1 ) s = "*"; // stop injection attack
  var q = "select * from book";
  if ( s != "*" ) q += " where title='" + s + "' or author='" + s + "'";
  conn.query(q, function (err, results, fields) {
    if (err) { res.send('error querying: ' + err); return; }
    var s = head + "<table border='1'>" + nl + "<tr>";
    for (var i in fields) s += "<th>" + fields[i].name + "</th>";
    s += "</tr>" + nl;
    for (var row in results) {
      s += "<tr>" + nl;
      for (var col in results[row])
        s += "<td>" + results[row][col] + "</td>" + nl;
      s += "</tr>" + nl;
    }
    s += "</table>" + nl + "</body>" + nl + "</html>" + nl;
    res.send(s);
  });
});

var svr = http.createServer(app);
svr.on('error', function(err) {console.log('Server: ' + err);});
svr.listen(8080, function() {console.log("Node: linux06 port 8080");});
```

Web app can be run (and killed) remotely on *linux06*.

Node.js and MySQL

To support the adding of books augment *sqlquery.js* with

```
app.get('/add', function (req, res) {
  var title = req.query.title.trim();
  var author = req.query.author.trim();
  var year = req.query.year.trim();
  var q = "insert into book values ";
  if ( title.indexOf(";") == -1 && author.indexOf(";") == -1 &&
       year.indexOf(";") == -1 )
       q += "('" + title + "', '" + author + "', '" + year + "')";
  else q = "";
  conn.query(q, function (err, results, fields) {
    if (err) { res.end('error adding: ' + err); return; }
    res.end(head + "<p>" + results.affectedRows +
            " record added</p>\n</body>\n</html>\n");
  });
});
```

Exercise this Add functionality with form on online page.

To support the deletion of books augment *sqlquery.js* with

```
app.get('/delete', function (req, res) {
  var title = req.query.title.trim();
  var q = "delete from book where title='" + title + "'";
  if ( title.indexOf(";") != -1 ) q = "";
  conn.query(q, function (err, results, fields) {
    if (err) { res.end('error deleting: ' + err); return; }
    res.end(head + "<p>" + results.affectedRows +
            " record deleted</p>\n</body>\n</html>\n");
  });
});
```

Exercise this Delete functionality with form on online page.

Updating of year of books is enabled by adding to *sqlquery.js*

```
app.get('/update', function (req, res) {
  var title = req.query.title.trim();
  var year = req.query.year.trim();
  var q = "update book set year='" + year +
          "' where title='" + title +"'";
  if ( title.indexOf(";") != -1 || year.indexOf(";") != -1 ) q = "";
  conn.query(q, function (err, results, fields) {
    if (err) { res.end('error updating: ' + err); return; }
    res.end(head + "<p>" + results.changedRows +
            " record changed</p>\n</body>\n</html>\n");
  });
});
```

Exercise this Update functionality with form on online page.

JavaScript Object Notation

JSON is lightweight, language neutral, data-interchange format.

JSON is built on 7 types of value including 2 types of structure

string	double quoted string of characters
number	(signed) integer or real number e.g. 42, 6.022e23, -273.15
special	true, false, null
array	ordered sequence of comma separated values between [and]
object	unordered bag of string/value pairs between { and }

JSON objects and values are cut down versions of JavaScript ones

objects	can only have properties with double quoted names cannot have properties with function values
values	strings, numbers, true, false, null, array, JSON object

Strings can contain

- any unicode character except " or \ or control character
- escaped letters such as \", \\, \/, \b, \f, \n, \r, \t
- 4 digit hex sequences preceded by backslash \ and letter u

Examples of objects and arrays

{}	{"age" : 18, "sex" : "male", "occupation" : null }
[]	[{"alive":true}, [1], null] [[], [[]], [[[]]]]

JavaScript like many other programming languages has API for JSON.

JSON.parse() extracts JavaScript objects from JSON syntax data.

```
var txt = '{ \"bible\" : 2, \"whisky bottle\" : 3}';
var obj = JSON.parse(txt);
```

Value of *obj.bible* or *obj["bible"]* would then be 2.

MongoDB

Key-Value Store holds *key-value* pairs e.g. hash table, dictionary.

JSON KVS extends KVS to sets of KV pairs (objects) and arrays.

JSON KVS is popular type of NoSQL database that now rivals RDB.

JSON KV set is called a *document* and its keys are called *fields*.

MongoDB is JSON KVS that

- stores *collections* of *documents* in *databases*
- offers good availability, high performance, ready scalability

MongoDB shell can interact with *anubis* server on port 27017

```
unix% mongo --host anubis -u hamish -p ??????????
      --authenticationDatabase hamish
MongoDB shell version: 3.2.9 connecting to: mongo-server-1:27017/test
> help
 ...
> db.help
 ...
> db.getName()
test
> exit
```

Users start in *test* DB and switch to own DB to do things.

Basic interactions - creating, deleting and enumerating collections.

```
> use hamish
switched to db hamish
> db.createCollection('things')
{ "ok" : 1 }
> db.getCollectionNames()
[ "books", "mystore", "system.indexes", "things", "users" ]
> db.things.drop()
true
> show collections
books
mystore
system.indexes
users
```

MongoDB

Collections are listed with *find()* and *pretty()* functions

```
> db.system.indexes.find().pretty()
  {
     "v" : 1,
     "key" : {
             "_id" : 1
     },
     "name" : "_id_",
     "ns" : "hamish.mystore"
  }
...
```

Script of commands in file *jsfile.js* such as

```
db.createCollection('mystore')
db.mystore.insert( {name: 'Brian', sex: 'male', age: 21} )
db.mystore.insert( {name: 'Cindy', sex: 'female', age: 19} )
db.mystore.insert( {name: 'Pandora', sex: 'intersex', age: 31} )
```

creates and populates a collection with some initial data

```
> load("jsfile.js")
true
```

Collection can be queried by specifying a key value

```
> db.mystore.find( { age: 21 } )
{ "_id": ObjectId(".."), "name": "Brian", "sex": "male', "age": 21 }
```

Results can be filtered (*_id* field removed) and prettified with

```
> db.mystore.find( { sex: intersex }, { _id: 0 } ).pretty()
{
  "name" : "Pandora",
  "sex" : "intersex",
  "age" : 31
}
```

Query can involve regular expression match with key value

```
> db.mystore.find( { sex: /male/ }, { _id: 0 } )
{ "name" : "Brian", "sex" : "male", "age" : 21 }
{ "name" : "Cindy", "sex" : "female", "age" : 19 }
```

Queries can also use boolean expressions and comparison operators

```
> db.mystore.find( {$or: [{sex: 'male'}, {age: {$gt: 30}}]}, {_id:0} )
{ "name": "Brian", "sex": "male", "age": 21 }
{ "name": "Pandora", "sex": "intersex", "age": 31 }
```

MongoDB

Updates are performed with *update()* function

```
> db.mystore.update( { name: 'Pandora' }, { $set: {'sex': 'male'} } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.mystore.find( { sex: 'intersex' } )
>
```

For each matching document, set operation is performed on it.

Matching documents are deleted with *remove()* function

```
> db.mystore.remove( { name: 'Brian' } )
WriteResult({ "nRemoved" : 1 })
> db.mystore.find( { name: 'Brian' } )
>
```

Mongoose is widely used schema-based MongoDB driver for Node.js.

Users compile mongoose schema models for easier handling.

It supports MongoDB URI references

```
mongodb://user:password@server-host:port/database?options
```

Components of URI are

mongodb://	required prefix
user:password@	optional authentication credentials
server-host	required domain name or IP address of DB server
:port	optional port - default is 27017
/database	optional database name
?options	optional configuration opportunities ssl=true - default is false authSource=db - database db is linked to *user* credentials

To use, install mongoose module to interact with MongoDB database.

```
unix% npm install mongoose
```

Node.js and Mongoose

Use of mongoose schemas in *mongo.js* eases handling of data

```
var mongoose = require('mongoose');
var express = require('express');
var http = require('http');
var app = express();

mongoose.connect("mongodb://hamish:????@mongo-server-1:27017/hamish");
var conn = mongoose.connection;
conn.once('open', function() { console.log('MongoDB connected '); });

var head = "<!DOCTYPE html>\n<html>\n";
head += "<head><title>Books</title></head>\n";

var booktemplate = {title: String, author: String, year: Number};
var bookschema = mongoose.Schema(booktemplate);
bookschema.methods.show = function () {
  var entry = "<tr>";
  for (var i in booktemplate) entry += "<td>" + this[i] + "</td>";
  return entry + "</tr>\n";
}
var book = mongoose.model('book', bookschema);

app.get('/search', function (req, res) {
  var qstring = req.query.search.trim();
  if ( qstring == "*" )
       query = {};
  else query = {$or: [{"title":  qstring}, {"author": qstring}]};
  book.find( query, function(err, books) {
    if ( err ) { return console.log(err); }
    page = head + "<body>\n<table border='1'>\n<tr>";
    for (var i in booktemplate)
      page += "<th>" + i + "</th>";
    page += "</tr>\n";
    for (var i in books)
      page += books[i].show();
    page += "</table>\n</body>\n</html>\n";
    res.end(page);
  });
});

var sv = http.createServer(app);
sv.on('error', function(err){ console.log('Node error: ' + err); });
sv.listen(8880, function(){ console.log("Node: linux07 on port 8880"); });
```

Schema model *book* creates collection *books*. (**"s" at end!**)

Put methods acting on each book in schema before compiling model.

Web service can be run (and killed) remotely on *linux07*.

Node.js and Mongoose

To support the adding of books augment *mongo.js* with

```
app.get('/add', function (req, res) {
  var title = req.query.title.trim();
  var author = req.query.author.trim();
  var year = req.query.year.trim();
  var b = new book({title: title, author: author, year: year});
  b.save( function (err, reply) {
    if (err) { return console.log(err); }
    res.end(head + reply + "</body>\n</html>\n");
  });
});
```

Exercise this Add functionality with form on online page.

To support the deletion of books augment *mongo.js* with

```
app.get('/delete', function (req, res) {
  var t = req.query.title.trim();
  var query = book.find().remove({"title": t});
  query.remove( {"title": t}, function(err, reply) {
    if (err) { return console.log(err); }
    res.end(head + reply + "</body>\n</html>\n");
  });
});
```

Exercise this Delete functionality with form on online page.

Query to find and delete document is created then run with callback.

Query deletes all documents in collection with the given title.

Updating of year of books is enabled by adding to *mongo.js*

```
app.get('/update', function (req, res) {
  var t = req.query.title.trim();
  var y = req.query.year.trim();
  book.where({"title": t}).update({"year": y},
             function(err, reply) {
    if (err) { return console.log(err); }
    var s = "";
    for (var i in reply) s += i + ": " + reply[i] + ", ";
    res.end(head + s + "</body>\n</html>\n");
  });
});
```

Exercise this Update functionality with form on online page.

Things To Do

Browse MongoDB Tutorial
https://www.tutorialspoint.com/mongodb/

Browse mysql module, Douglas Wilson
https://github.com/mysqljs/mysql

Browse mongoose module
http://mongoosejs.com/docs/

Try *Quiz on Lecture*

Do *Exercise 8*

Key Points

- Node is part of the MEAN stack (MongoDB, Express, AngularJS, Node) that's *supposedly* replacing the LAMP web software stack (Linux, Apache, MySQL, PHP).

- NoSQL databases offer an alternative to relational databases like MySQL. Popular among them are *key-value* stores like MongoDB. Its efficient implementation in C++, key-value data model, and suitability for handling JSON data suit use with Node.js for storing data for web service applications.