Web Storage

INFSCI2560

University of Pittsburgh

Dr. Alawami

Next Class : TUE 10/16

Notes

- Tutorial (optional)
- *You have the right to*
  - Stop me if I am too fast for you.
  - Ask for more clarification.
  - Slides/points unclear.
  - *Speak up or email me*
- Full stack internships
- Front-end internships

  https://www.peersight.com/job/ignitus-front-end-development-internship?utm_campaign=google_jobs_apply&utm_source=google_jobs_apply&utm_medium=organic

  Much more on the web. Start looking and applying now for Summer 2019

Agenda- moving to back-end

- Web Storage
    - Local
    - Databases
        - Structured
        - Key-value pair
- Focus on NOSQL

Web storage- Why?

- HTTP is a **stateless** protocol.
  - When you use an application and then close it, its state will be reset the next time you open it
- As a developer, you need to store the state of your interface somewhere.

Scenario 1

- Each time a user visit your site
- You keep certain info
  - Links they clicked
  - Navigation behavior
- You don’t want the users to sign in
- You store the information of the behavior on your server
- Each user is associated with an ID
- How would you revert the state of your site when this user comes back?

Scenario 1

- One way to store small values?
- Cookies
  - Name, value pairs with properties
  - Lifetime independent of request/response
  - Passed between client and server during HTTP transactions
  - You can store information in them, read them out and delete them
- Hidden fields, URL rewriting
  - Form controls (input type="hidden") added dynamically to pages, containing name/value that should be associated with client.
  - Hardcoded links (href) contain name/value data in query

Scenario 1

- Sessions
  - Pass a single cookie (or fallback to URL rewriting) containing a session ID
  - Server maintains a mapping between session ID and associated data stored on the server.
- Is this good enough? Can you think of ways this scenario is not enough?

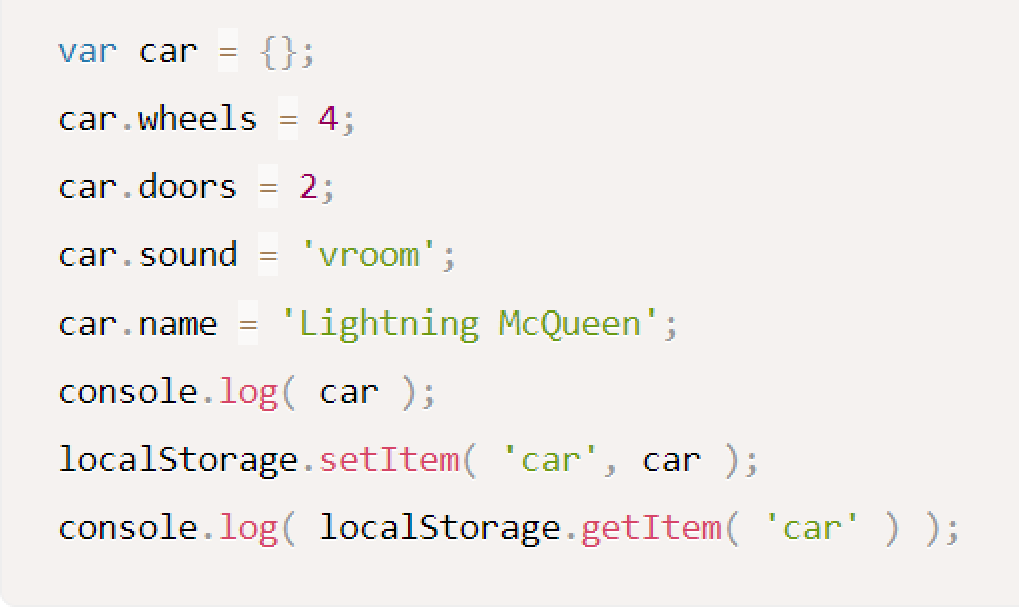
Drawback of this approach

- Cookies Limitation
  - Add to the load of accessing documents.
  - Only 4 KB of data storage.
  - Security issues

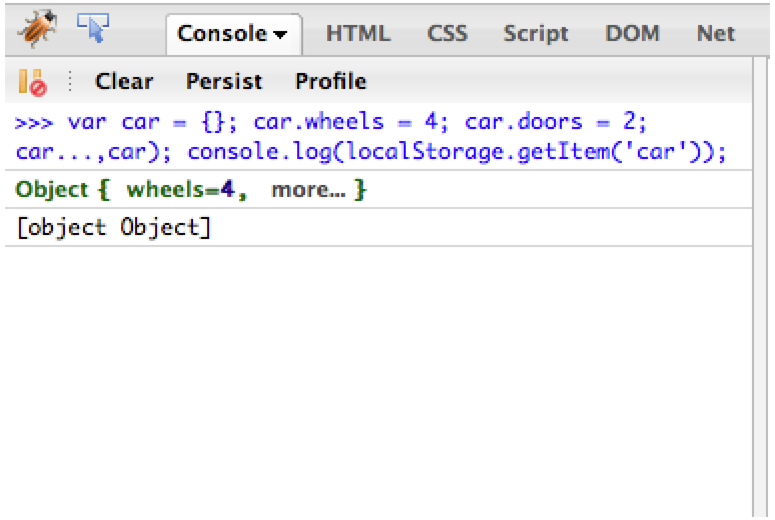
- Can you think of one more *Local* method?

HTML5 local storage

- Discussed on Lecture 2
- One annoying shortcoming of local storage is that you can only store strings in the different keys.

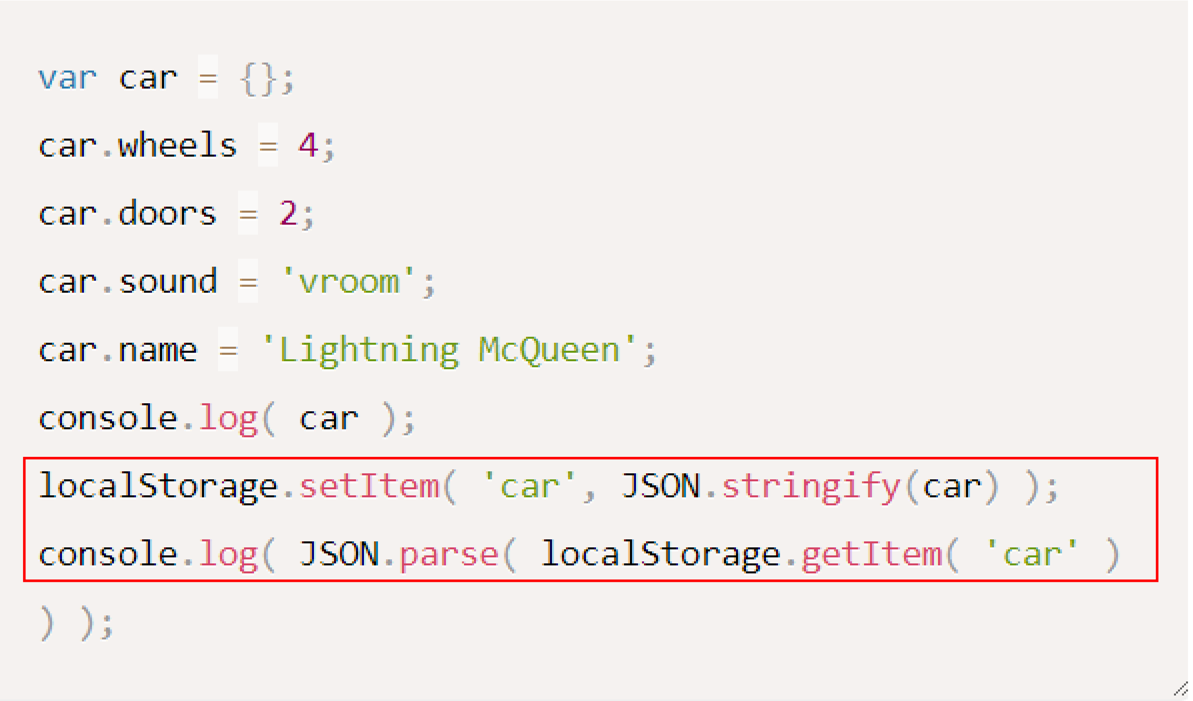


```
var car = {};
car.wheels = 4;
car.doors = 2;
car.sound = 'vroom';
car.name = 'Lightning McQueen';
console.log( car );
localStorage.setItem( 'car', car );
console.log( localStorage.getItem( 'car' ) );
```



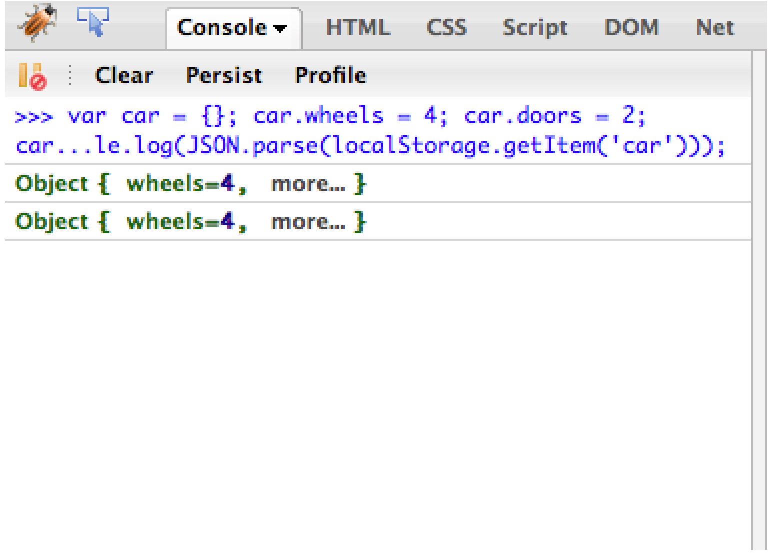
```
>>> var car = {}; car.wheels = 4; car.doors = 2;
car...,car); console.log(localStorage.getItem('car'));
Object { wheels=4,  more... }
[object Object]
```

A trick!



```
var car = {};
car.wheels = 4;
car.doors = 2;
car.sound = 'vroom';
car.name = 'Lightning McQueen';
console.log( car );
localStorage.setItem( 'car', JSON.stringify(car) );
console.log( JSON.parse( localStorage.getItem( 'car' )
) );
```

A trick!



```
>>> var car = {}; car.wheels = 4; car.doors = 2;
car...le.log(JSON.parse(localStorage.getItem('car')));
Object { wheels=4,  more... }
Object { wheels=4,  more... }
```

How to view local storage?

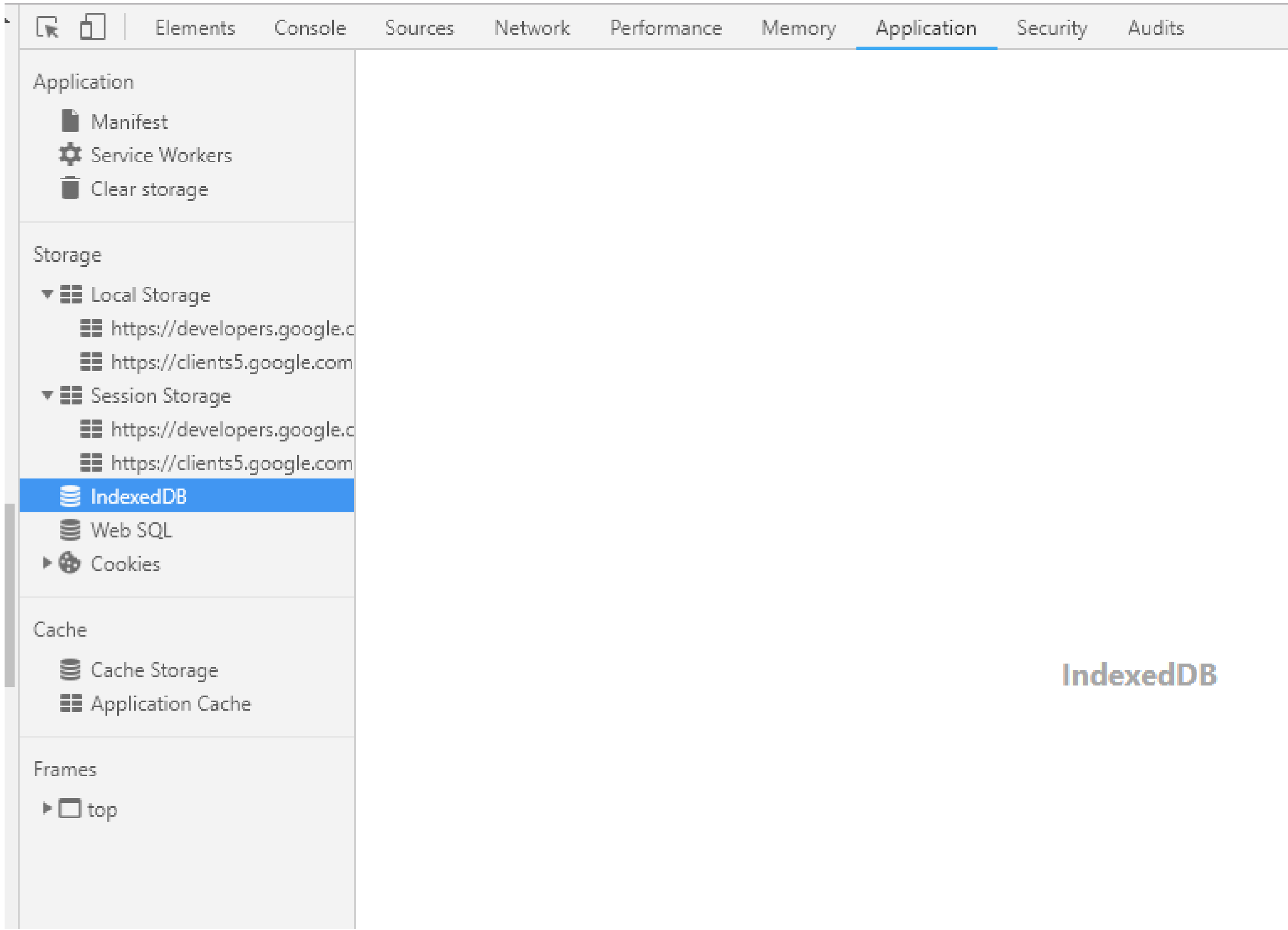
- Chrome

  Settings->Advance->

  content settings>cookies

- During development

  Developers tools-Application tab



Final note on Local storage

- Of course, any powerful technology comes with the danger of people abusing it for darker purposes. Samy, the man behind the “Samy is my hero” MySpace worm, recently released a rather scary demo called Evercookie, which shows how to exploit all kind of techniques, including local storage, to store information of a user on their computer even when cookies are turned off. This code could be used in all kinds of ways, and to date there is no way around it.

- Source: https://www.smashingmagazine.com/2010/10/local-storage-and-how-to-use-it/

Two type of storage

- Type
  - Local
  - Cloud based storage server
- Data Model
  - Structured (relational model)
  - Key/value (NOSQL)
  - Byte Streams

Data Storage Persistence

Classify
Shopping Cart

- **Session Persistence:**

  Ex: Session API. sessionStorage.setItem('key', 'value’);

- **Device Persistence:**
  - Data in this category is retained across sessions and browser tabs/windows, within a particular device.

  Ex:Cache API.

- **Global Persistence:** Data in this category is retained across sessions and devices.
  - most robust form of data persistence.
  - An example of a storage mechanism with global persistence is Cloud Storage.

Our focus

- Relational and NOSQL

What is a Database?

- Structured collection of data.
  - Tables
  - Fields
  - Query
  - Reports
- Essentially a much more sophisticated implementation of the flat files.

Relational Databases

Source: http://www.massey.ac.nz/~nhreyes/MASSEY/159339/Lectures/Lecture%2015%20-%20MySQL-%20PHP%201.ppt

Relational Databases

- Stores data in separate tables instead of a single store.
- Relationships between tables are set
- In theory, this provides a faster, more flexible database system.

Example

- We wish to maintain a **database** of student names, IDs, addresses, and any other information.
- Will be **updated frequently** with new names and information.
- Will want to **retrieve data** based on some predicate.
  - **e.g, ‘give me the names of all first year students who live in Albany’.**
- Will want to update database with new information about students, not previously recorded.
  - **e.g., may decide we want to include IRD nos.**
- Very difficult to manage using ‘flat file’ systems

Databases

- **Fast, Efficient back end storage**
  - Easier to manage than file system based approach
- **Relational Database structure**
  - Well developed theory and practise
- **Multi-user capable**
  - Multithreaded, multiprocessor, sometimes cluster based systems
- **Standards based queries**
  - **Structured Query Language (SQL**)

MYSQL Database

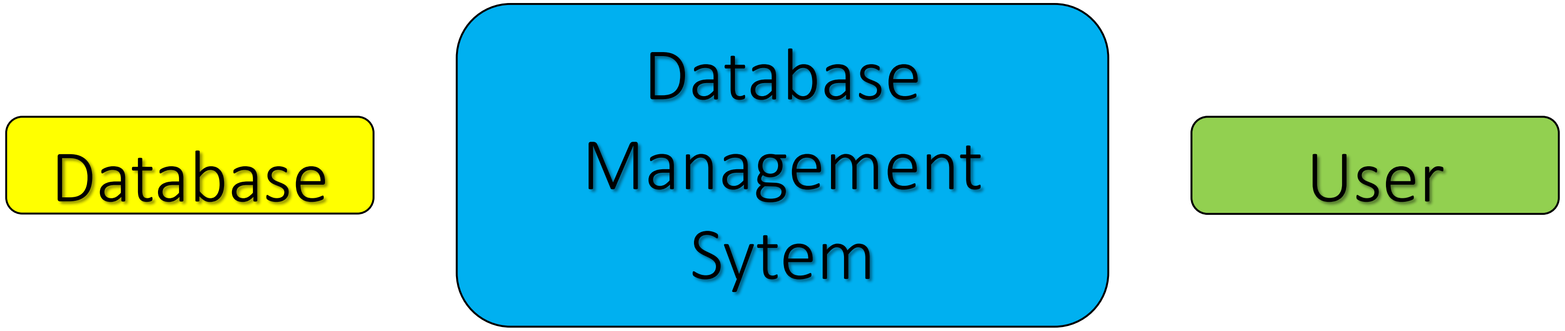
- World's most popular open source database because of its consistent **fast performance, high reliability and ease of use**
- Open Source License:- free
  - GNU General Public License
  - Free to modify and distribute but all modification must be available in source code format
- Commercial:- not free

Basic Database Server Concepts

- **Database runs as a server**
  - Attaches to either a default port or an administrator specified port
- **Clients connect to database**
  - For secure systems
    - authenticated connections
    - usernames and passwords
- **Clients make queries on the database**
  - Retrieve content
  - Insert content
- **SQL (Structured Query Language)** is the language used to insert and retrieve content

Database Management System

- Manages the storage and retrieval of data to and from the database and hides the complexity of what is actually going on from the user



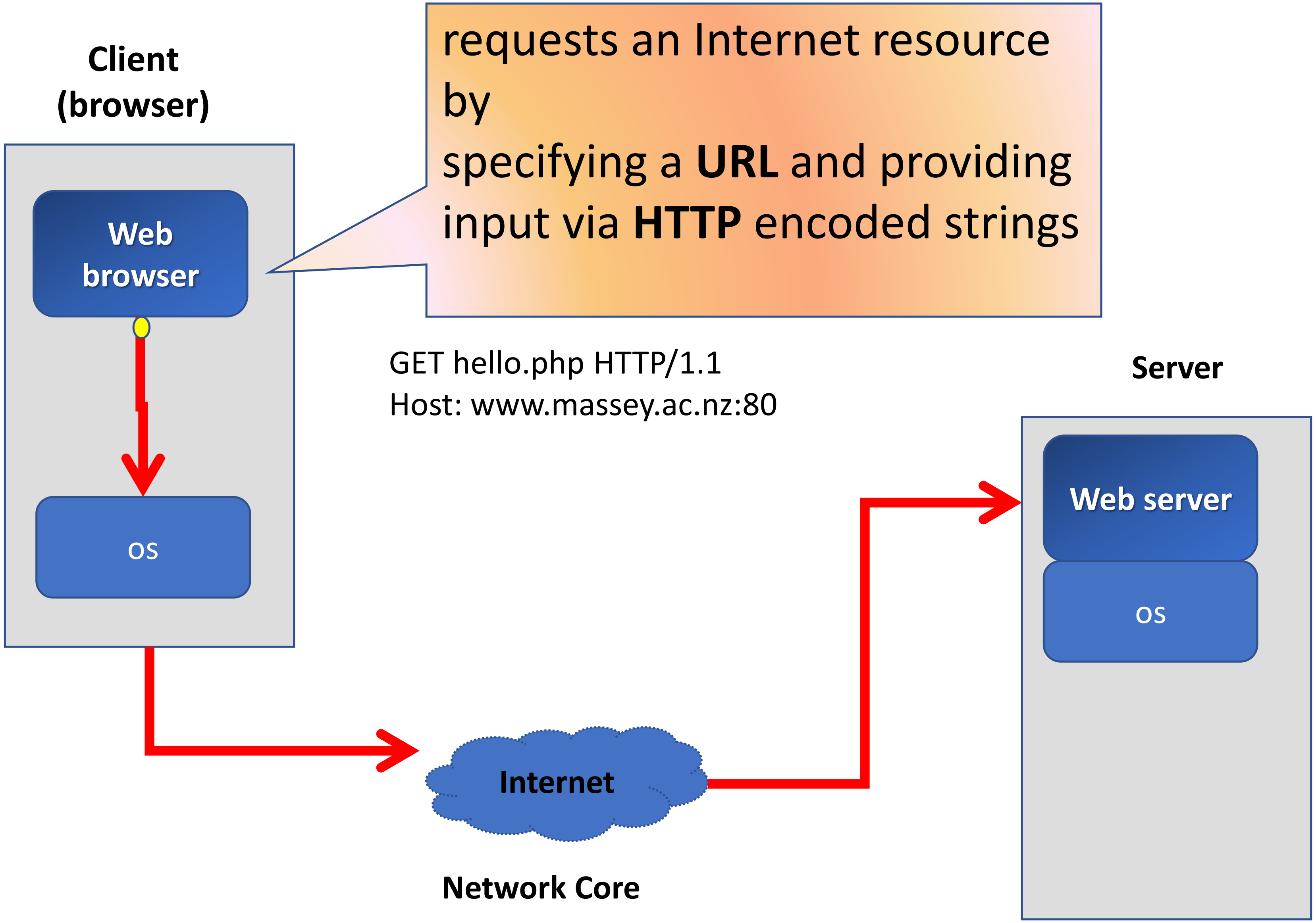
Database

Database Management Sytem

User

- MySQL is a relational database management system

Client: makes a request

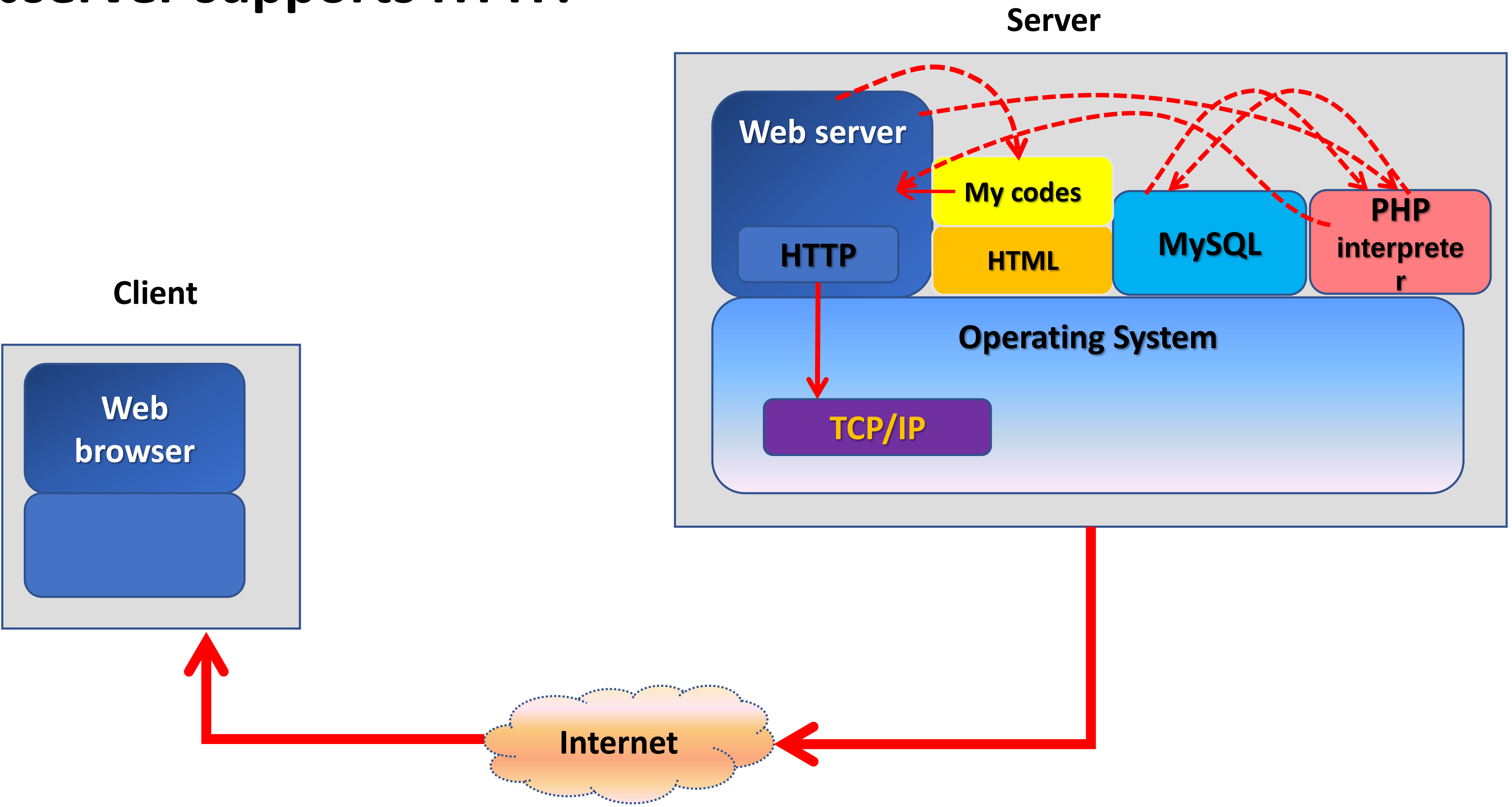


requests an Internet resource by specifying a **URL** and providing input via **HTTP** encoded strings

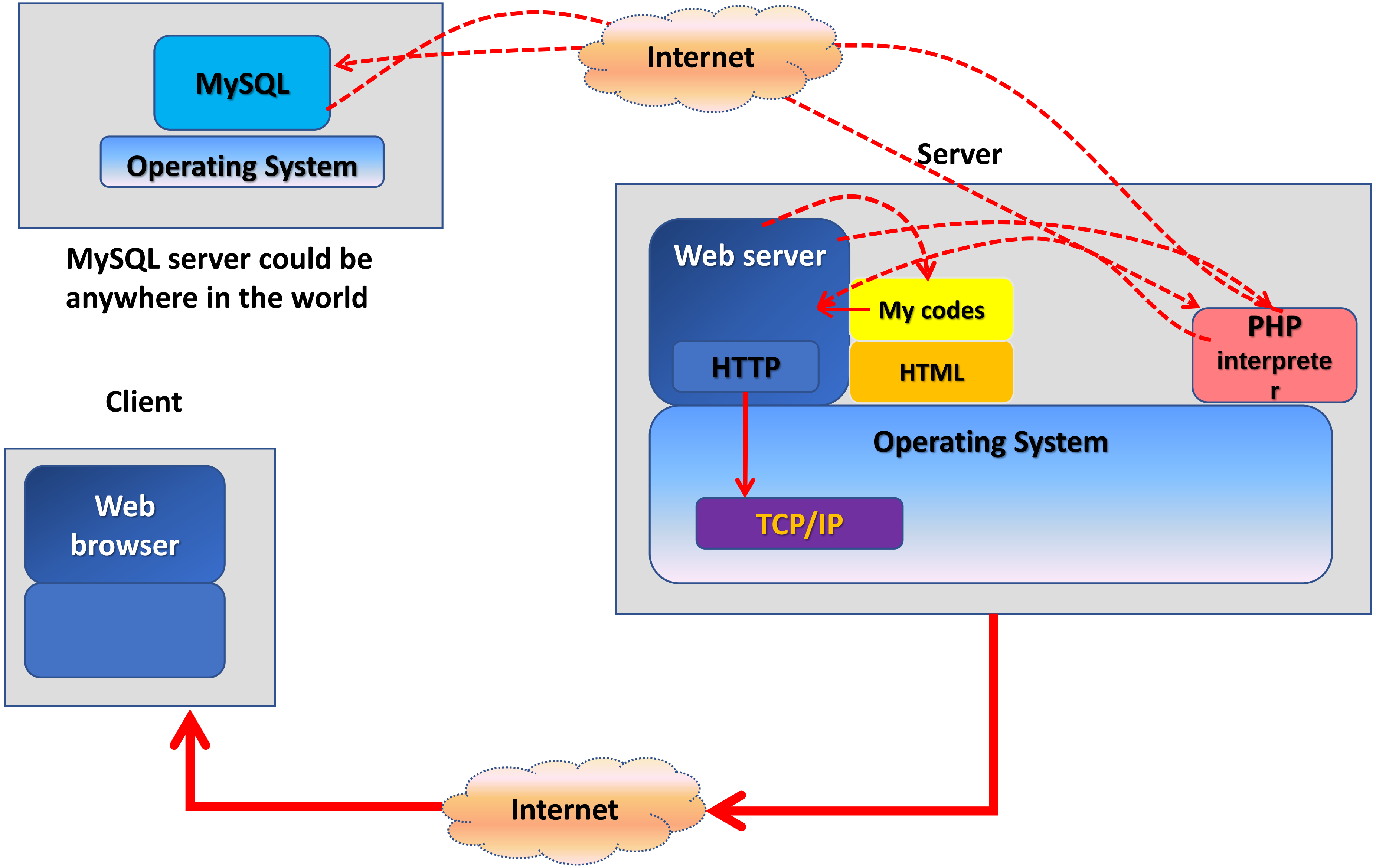
```
GET hello.php HTTP/1.1
Host: www.massey.ac.nz:80
```

Server: responds

- **Webserver supports HTTP.**



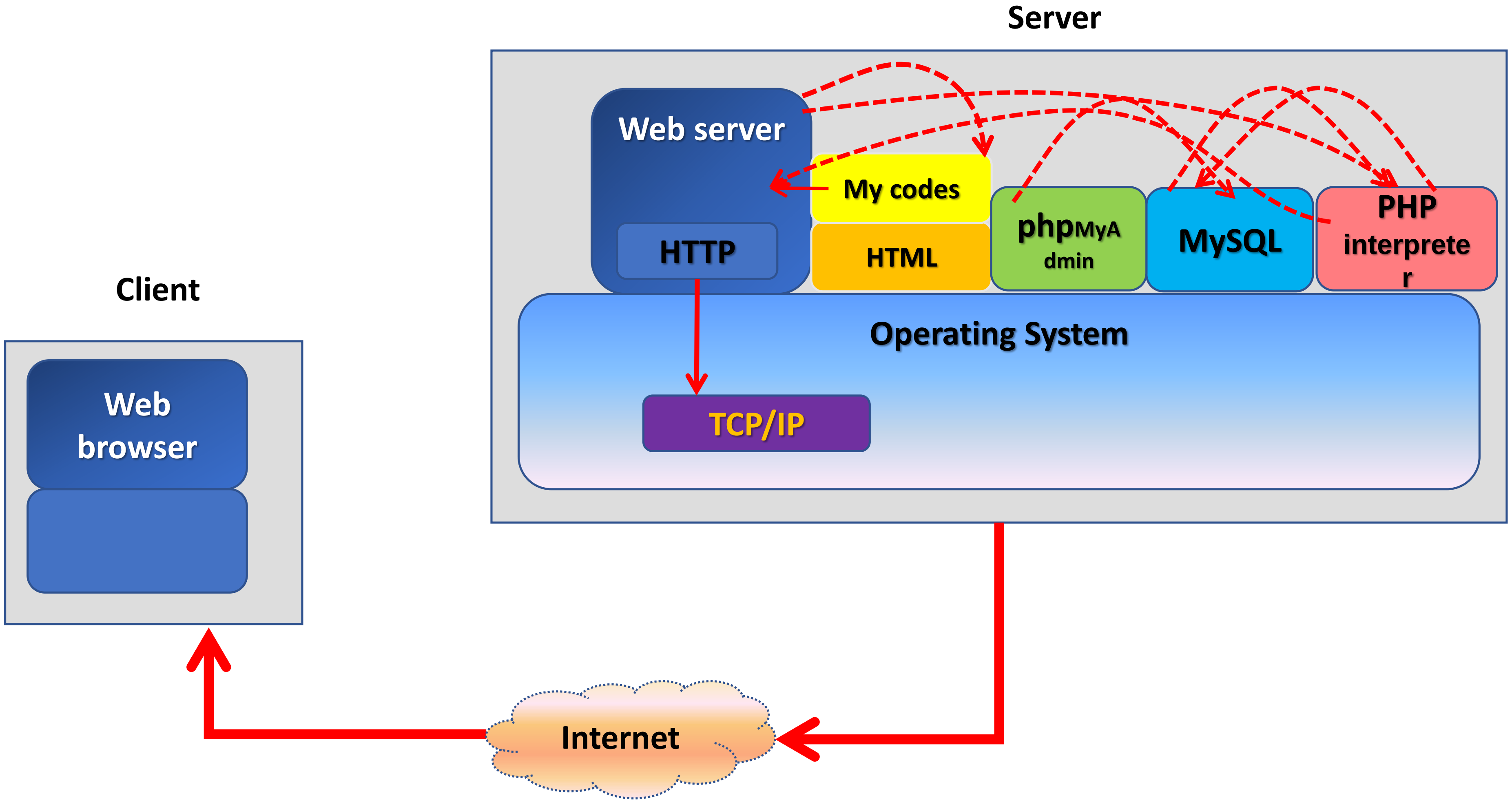
Server: responds



MySQL server could be anywhere in the world

Server: responds

- Webserver supports HTTP.



MYSQL

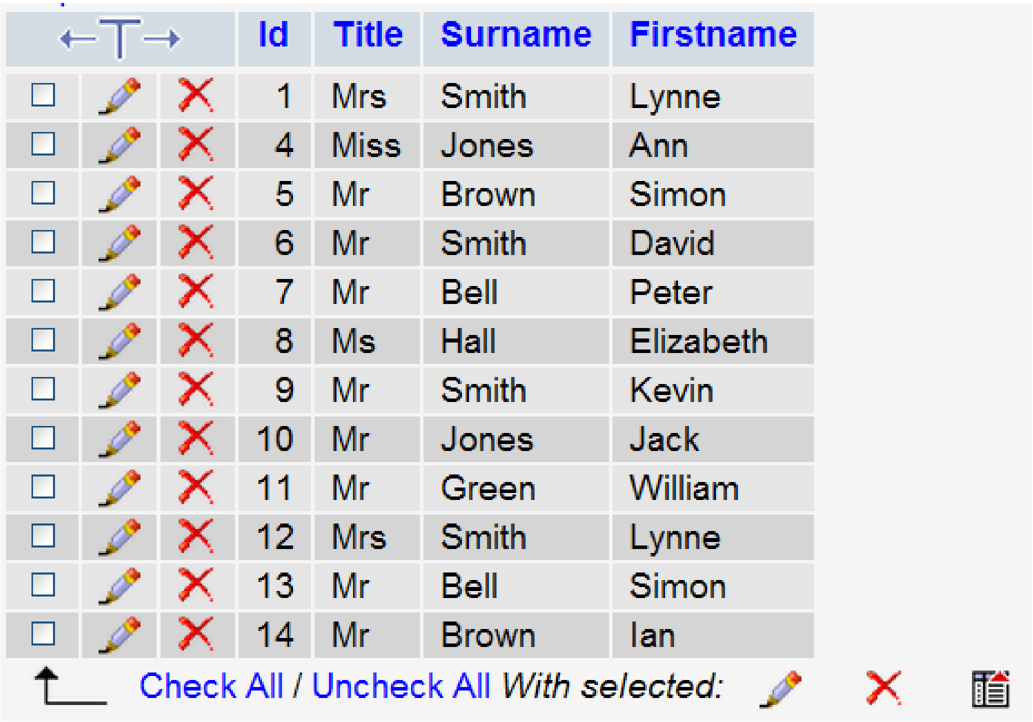
- Standalone
- Integrated within your web development environment (PHPMyAdmin, XAMPP, Apache).
- Pitt server don’t have database storage.

Install mysql in your machine. Use mysqldriver engine to connect

ex: java mysqlconnector.jar.

Database Quick tour

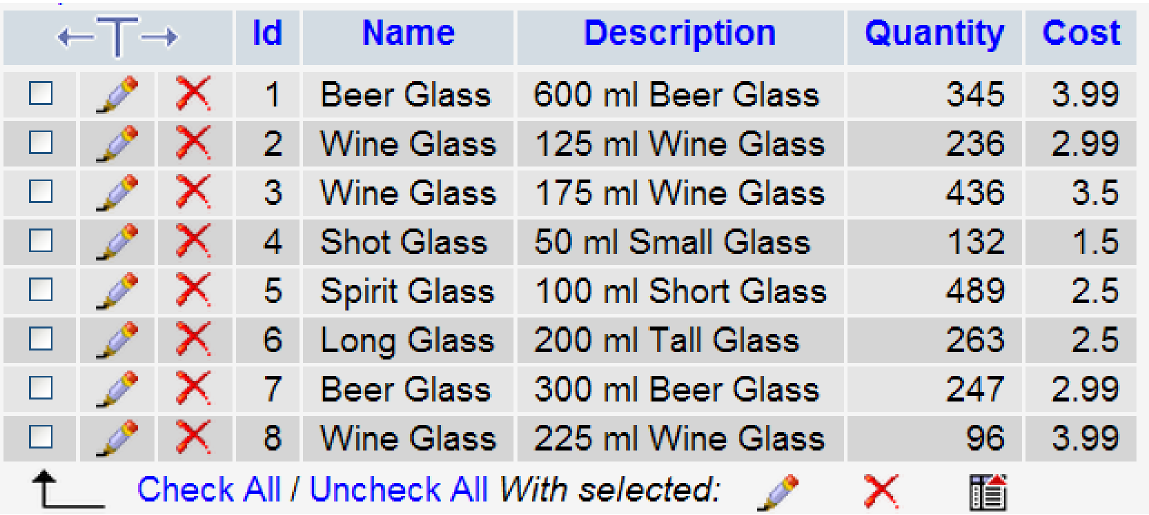
Table: Customers (data)



←T→			Id	Title	Surname	Firstname
□	✎	✗	1	Mrs	Smith	Lynne
□	✎	✗	4	Miss	Jones	Ann
□	✎	✗	5	Mr	Brown	Simon
□	✎	✗	6	Mr	Smith	David
□	✎	✗	7	Mr	Bell	Peter
□	✎	✗	8	Ms	Hall	Elizabeth
□	✎	✗	9	Mr	Smith	Kevin
□	✎	✗	10	Mr	Jones	Jack
□	✎	✗	11	Mr	Green	William
□	✎	✗	12	Mrs	Smith	Lynne
□	✎	✗	13	Mr	Bell	Simon
□	✎	✗	14	Mr	Brown	Ian

Check All / Uncheck All *With selected:* ✎ ✗

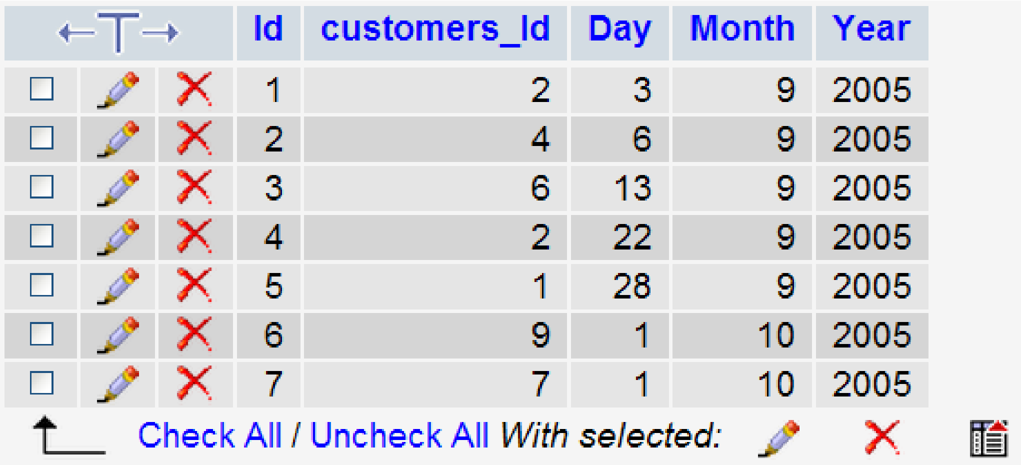
Table: Products (data)



←T→			Id	Name	Description	Quantity	Cost
□	✎	✗	1	Beer Glass	600 ml Beer Glass	345	3.99
□	✎	✗	2	Wine Glass	125 ml Wine Glass	236	2.99
□	✎	✗	3	Wine Glass	175 ml Wine Glass	436	3.5
□	✎	✗	4	Shot Glass	50 ml Small Glass	132	1.5
□	✎	✗	5	Spirit Glass	100 ml Short Glass	489	2.5
□	✎	✗	6	Long Glass	200 ml Tall Glass	263	2.5
□	✎	✗	7	Beer Glass	300 ml Beer Glass	247	2.99
□	✎	✗	8	Wine Glass	225 ml Wine Glass	96	3.99

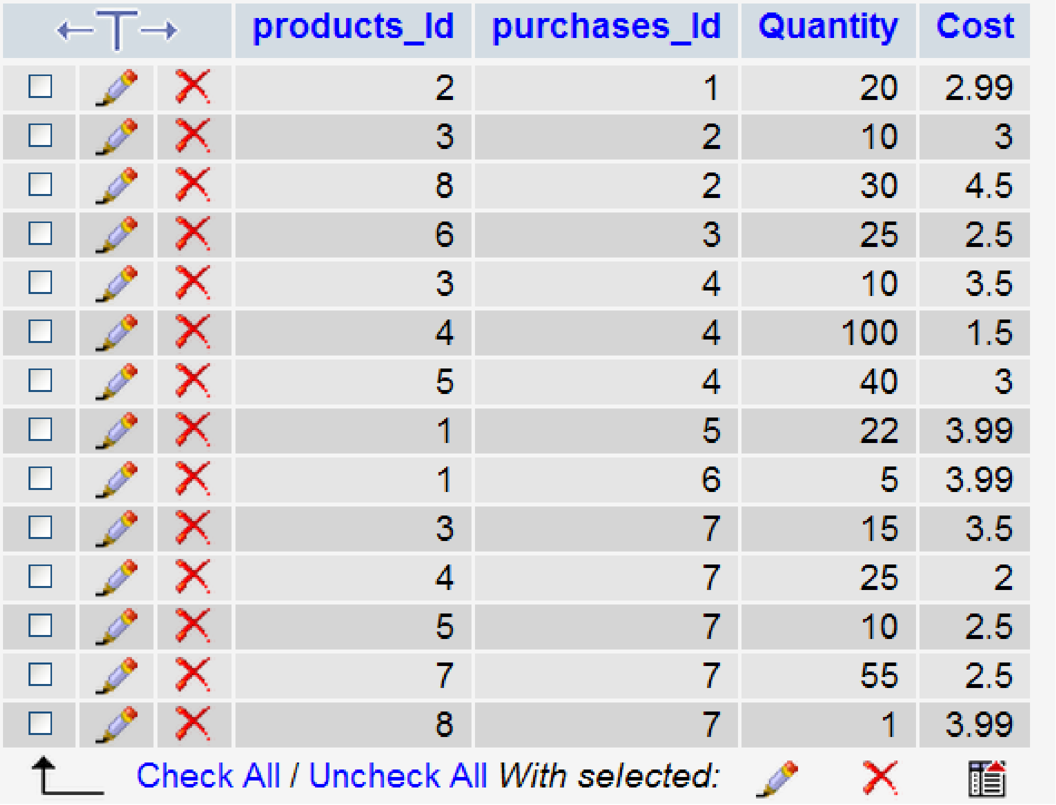
Check All / Uncheck All *With selected:* ✎ ✗

Table: Purchases (data)



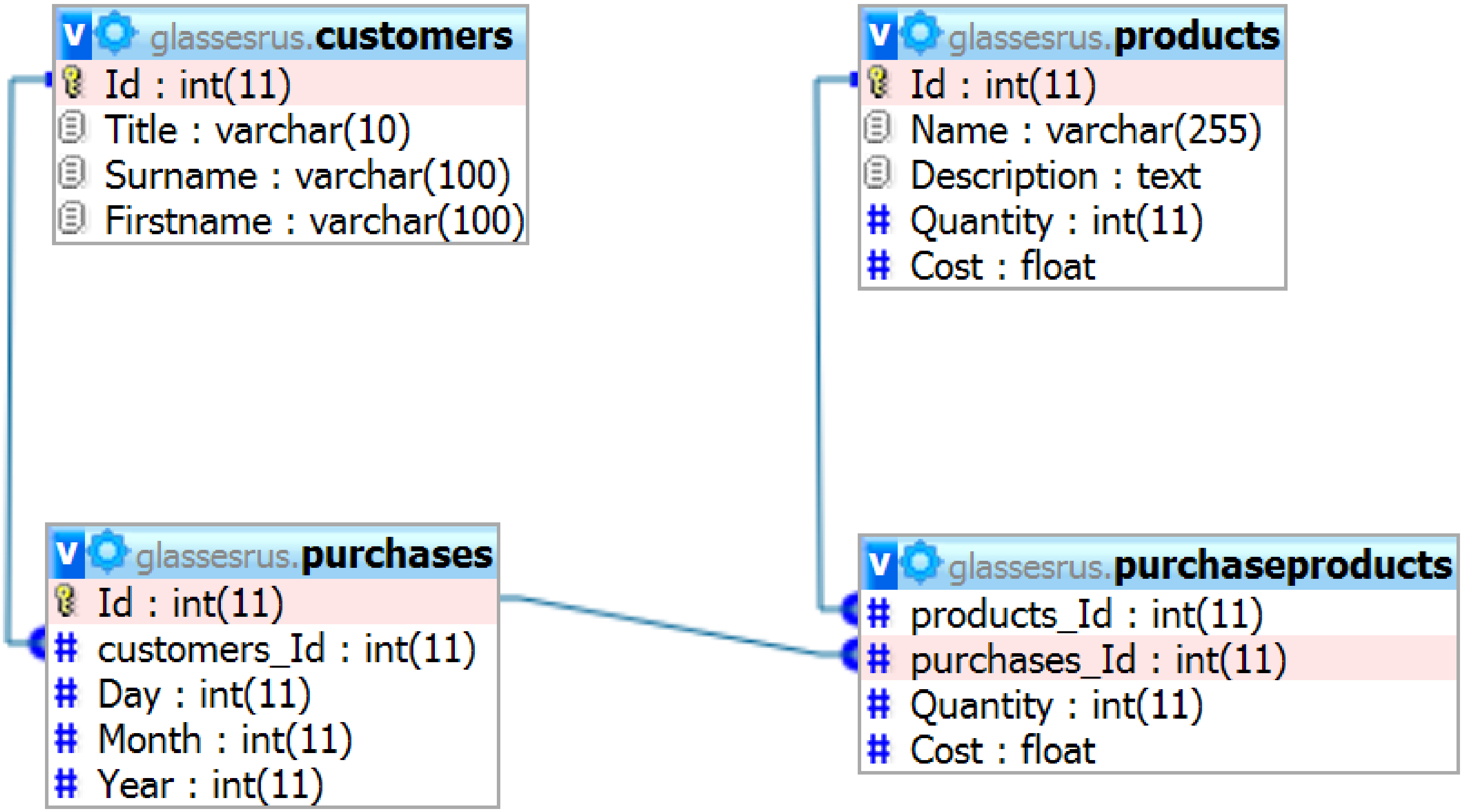
←T→			Id	customers_Id	Day	Month	Year
□	✎	✗	1	2	3	9	2005
□	✎	✗	2	4	6	9	2005
□	✎	✗	3	6	13	9	2005
□	✎	✗	4	2	22	9	2005
□	✎	✗	5	1	28	9	2005
□	✎	✗	6	9	1	10	2005
□	✎	✗	7	7	1	10	2005

Table: PurchaseProducts (data)



←T→			products_Id	purchases_Id	Quantity	Cost
□	✎	✗	2	1	20	2.99
□	✎	✗	3	2	10	3
□	✎	✗	8	2	30	4.5
□	✎	✗	6	3	25	2.5
□	✎	✗	3	4	10	3.5
□	✎	✗	4	4	100	1.5
□	✎	✗	5	4	40	3
□	✎	✗	1	5	22	3.99
□	✎	✗	1	6	5	3.99
□	✎	✗	3	7	15	3.5
□	✎	✗	4	7	25	2
□	✎	✗	5	7	10	2.5
□	✎	✗	7	7	55	2.5
□	✎	✗	8	7	1	3.99

Database Design



glassesrus.customers

Id : int(11)
Title : varchar(10)
Surname : varchar(100)
Firstname : varchar(100)

glassesrus.products

Id : int(11)
Name : varchar(255)
Description : text
Quantity : int(11)
Cost : float

glassesrus.purchases

Id : int(11)
# customers_Id : int(11)
Day : int(11)
Month : int(11)
Year : int(11)

glassesrus.purchaseproducts

products_Id : int(11)
purchases_Id : int(11)
Quantity : int(11)
Cost : float

SQL

- SQL - Structured Query Language, a special-purpose programming language designed for managing data held in a relational database
- SQL is almost English; it's made up largely of English words, put together into strings of words that sound similar to English sentences.

Query Types

- The first word of each query is its name, which is an action word (a verb) that tells DMBS what you want to do.
  - CREATE – creates a new table or a schema
  - DROP – drops an existing table or a schema
  - SELECT – retrieves data from a table or a set of tables
  - INSERT – creates a new record in a single table
  - UPDATE – updates in a single table
  - DELETE – deletes/removes a record from a single table

CREATE DATABASE

CREATE DATABASE [database name];

CREATE DATABASE

```sql
CREATE DATABASE movie_tracker;
```

USE DATABASE

USE [database name] statement will tell MySQL that all of your queries should be executed against the specified database.

USE DATABASE

```
USE movie_tracker;
```

SHOW TABLES

SHOW TABLES statement will give you a list of all tables in your database;

CREATE TABLE

CREATE TABLE statement is used to specify the logical layout of a table and to create a database table.

CREATE TABLE

```
CREATE TABLE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]
```

http://dev.mysql.com/doc/refman/5.1/en/create-table.html

CREATE TABLE

```
CREATE TABLE  movie (
        movie_id INT,
        title VARCHAR(200),
        budget DOUBLE,
        release_date DATETIME
);
```

CREATE TABLE

```
CREATE TABLE  movie (
        movie_id INT NOT NULL,
        title VARCHAR(200) NOT NULL,
        budget DOUBLE NOT NULL,
        release_date DATETIME NOT NULL
);
```

CREATE TABLE

```sql
CREATE TABLE  movie (
	movie_id INT PRIMARY KEY NOT NULL,
	title VARCHAR(200) NOT NULL,
	budget DOUBLE NOT NULL,
	release_date DATETIME NOT NULL
);
```

CREATE TABLE

```
CREATE TABLE  movie (
        movie_id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
        title VARCHAR(200) NOT NULL,
        budget DOUBLE NOT NULL,
        release_date DATETIME NOT NULL
);
```

USE DATABASE

USE [database name] statement will tell MySQL that all of your queries should be executed against the specified database.

SHOW TABLES

SHOW TABLES statement will give you a list of all tables in your database;

NUMERIC DATA TYPES

- Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT
- Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC
- Floating-Point Types (Approximate Value) - FLOAT, DOUBLE
- Bit-Value Type - BIT

http://dev.mysql.com/doc/refman/5.6/en/numeric-types.html

STRING DATA TYPES

- CHAR and VARCHAR Types
- BINARY and VARBINARY Types
- BLOB and TEXT Types

http://dev.mysql.com/doc/refman/5.6/en/string-types.html

DATE/TIME DATA TYPES

- DATE, DATETIME, and TIMESTAMP Types
- TIME Type
- YEAR Type

http://dev.mysql.com/doc/refman/5.6/en/date-and-time-types.html

INSERT

```
INSERT INTO table_name ( field1, field2,...fieldN )
                         VALUES
                        ( value1, value2,...valueN );
```

INSERT

```
INSERT INTO classicmodels.payments
(customerNumber, checkNumber, paymentDate, amount)
VALUES
(103, 1, ‘2014-10-10’, 4000);
```

UPDATE

UPDATE table_name **SET** field1=new-value1, field2=new-value2 [WHERE Clause]

UPDATE

```
UPDATE classicmodels.payments
SET amount = 10000
WHERE customerNumber = 103
AND checkNumber = 1;
```

UPDATE

```
UPDATE classicmodels.payments
SET amount = 10000, checkNumber = ‘XXXXXX’
WHERE customerNumber = 103
AND checkNumber = 1;
```

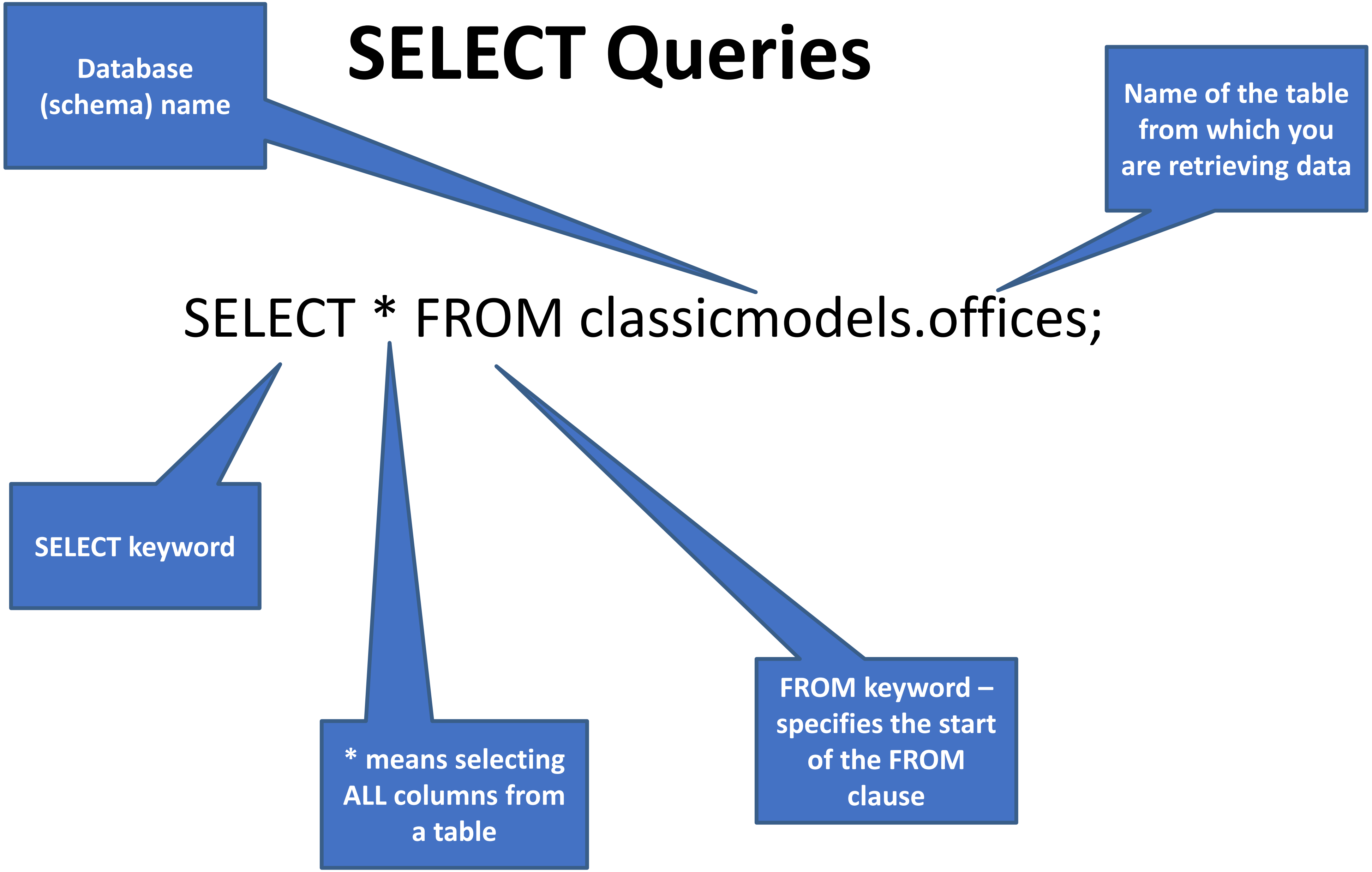
DELETE

DELETE FROM table_name [WHERE Clause]

DELETE

```
DELETE FROM classicmodels.payments
WHERE customerNumber = 103
AND checkNumber = 1;
```

SELECT Queries



Database (schema) name

Name of the table from which you are retrieving data

```
SELECT * FROM classicmodels.offices;
```

SELECT keyword

* means selecting ALL columns from a table

FROM keyword – specifies the start of the FROM clause

SELECT Queries

Selecting a list of columns

```
SELECT officeCode, city
FROM classicmodels.offices;
```

It's a good practice to end a query with a semicolon

Query Clauses

Clauses - constituent components of statements and queries.

- **FROM**
- **WHERE**
- GROUP BY
- HAVING
- **ORDER BY**
- **LIMIT**

FROM

- Indicates the table(s) from which data is to be retrieved.

SELECT * **FROM classicmodels.offices**

WHERE

- Includes a comparison predicate, which restricts the rows returned by the query.
- The WHERE clause eliminates all rows from the result set for which the comparison predicate does not evaluate to True.

```
SELECT * FROM classicmodels.offices
WHERE city = ‘Boston’;
```

AND OPERATOR

- ***AND*** operator is used in WHERE clauses
- Allows to limit query results be comparing values against multiple fields

```
SELECT * FROM classicmodels.offices
WHERE city = 'Boston' AND territory = 'NA';
```

ORDER BY

- Identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending).
- Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

```
SELECT * FROM classicmodels.offices
ORDER BY city DESC
```

ORDER OF CLAUSES

- CLAUSES must appear in the following order
  - **FROM**
  - **WHERE**
  - **GROUP BY**
  - **HAVING**
  - **ORDER BY**
- Not all clauses must appear in a query – **FROM** clause is the only one that's required

Operators

Operator	Description	Example
=	Equal to	Author = 'Alcott'
<>	Not equal to (most DBMS also accept != instead of <>)	Dept <> 'Sales'
>	Greater than	Hire_Date > '2012-01-31'
<	Less than	Bonus < 50000.00
>=	Greater than or equal	Dependents >= 2
<=	Less than or equal	Rate <= 0.05
BETWEEN	Between an inclusive range	Cost BETWEEN 100.00 AND 500.00
LIKE	Match a character pattern	First_Name LIKE 'Will%'
IN	Equal to one of multiple possible values	DeptCode IN (101, 103, 209)
IS *or* IS NOT	Compare to null (missing data)	Address IS NOT NULL

LIKE + WILDCARDS

- LIKE statement allows you to search for matches within character fields.
- % (percent) is a wildcard

```
SELECT * FROM Employees
WHERE lastName LIKE ‘%Sm’;
```

LIMIT

- Limits the number of records (table rows) returns by an SQL query
- Always the last clause in the query
- Note that LIMIT is specific to MySQL and Oracle – might not work with other database systems

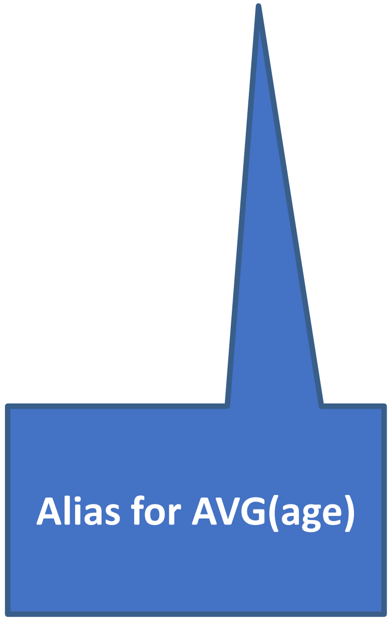
```
SELECT * FROM Employees WHERE lastName = ‘Smith’
LIMIT 5;
```

Aggregate Functions

- An aggregate function performs a calculation on a set of values and returns a single value.
- Most common MySQL aggregate functions are
  - AVG
  - COUNT
  - SUM
  - MIN
  - MAX

AVG(expression)

```
SELECT AVG(age) averagePatientAge FROM Patients
```



Alias for AVG(age)

http://www.mysqltutorial.org/mysql-avg/

COUNT Function

```
SELECT COUNT(*) patientCount
FROM Patients
WHERE patientAge > 10
```

http://www.mysqltutorial.org/mysql-count/

SUM Function

```sql
SELECT SUM(medicationPrice)
FROM Prescription p JOIN Medication m
ON p.medicationID = m.medicationID
WHERE patientID = 5
```

http://www.mysqltutorial.org/mysql-sum/

MAX Function

```
SELECT MAX(medicationPrice)
FROM Medication
```

http://www.mysqltutorial.org/mysql-max-function/

MIN Function

```
SELECT MIN(medicationPrice)
FROM Medication
```

http://www.mysqltutorial.org/mysql-min/

Reference on MySQL

- https://www.w3schools.com/sql/default.asp
- https://www.w3schools.com/sql/sql_ref_mysql.asp
- https://dev.mysql.com/doc/refman/8.0/en/ (More details and thorough)

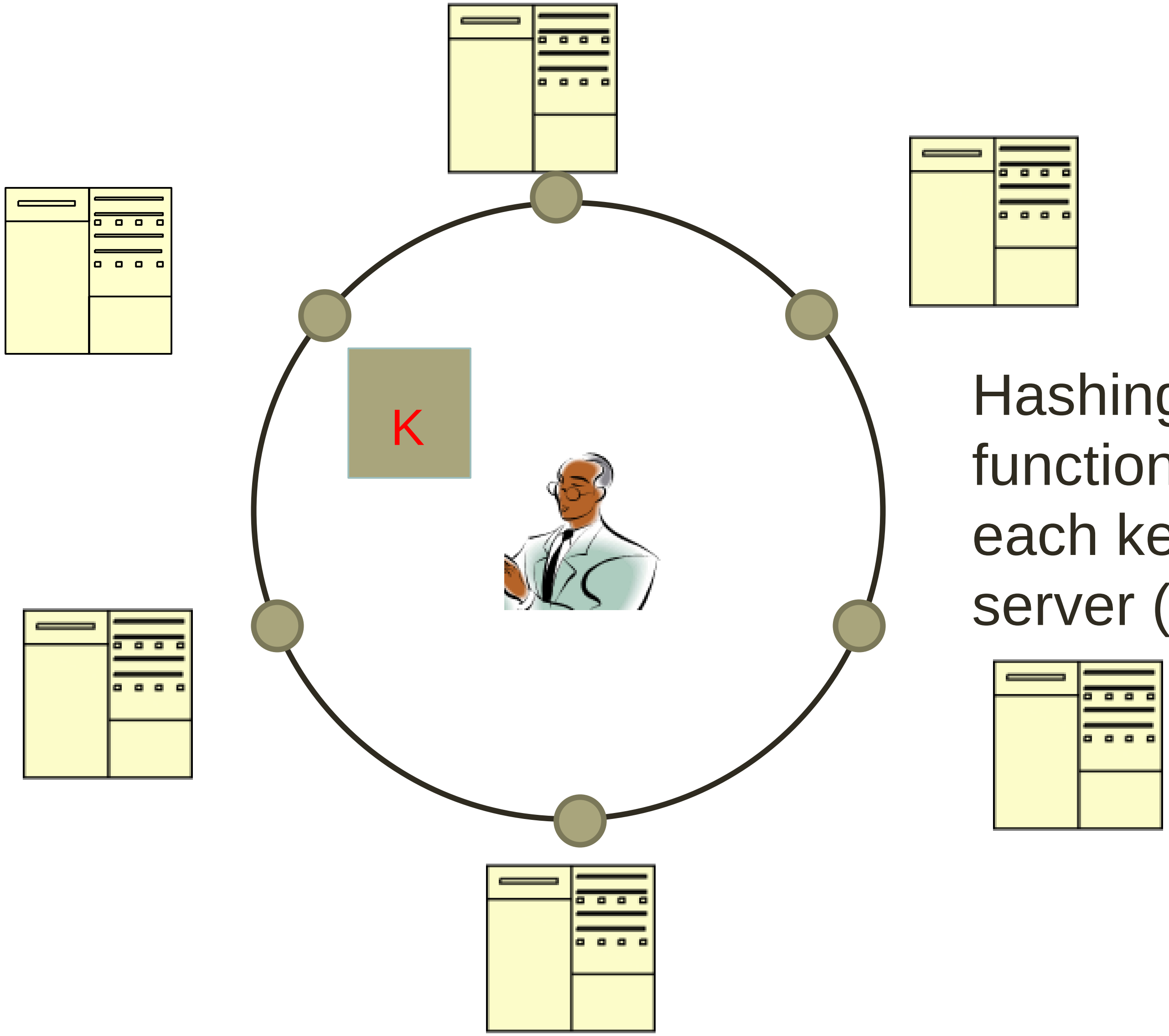
No SQL- MongoDB Intro

Modified from Kathleen Durant from Northeastern University

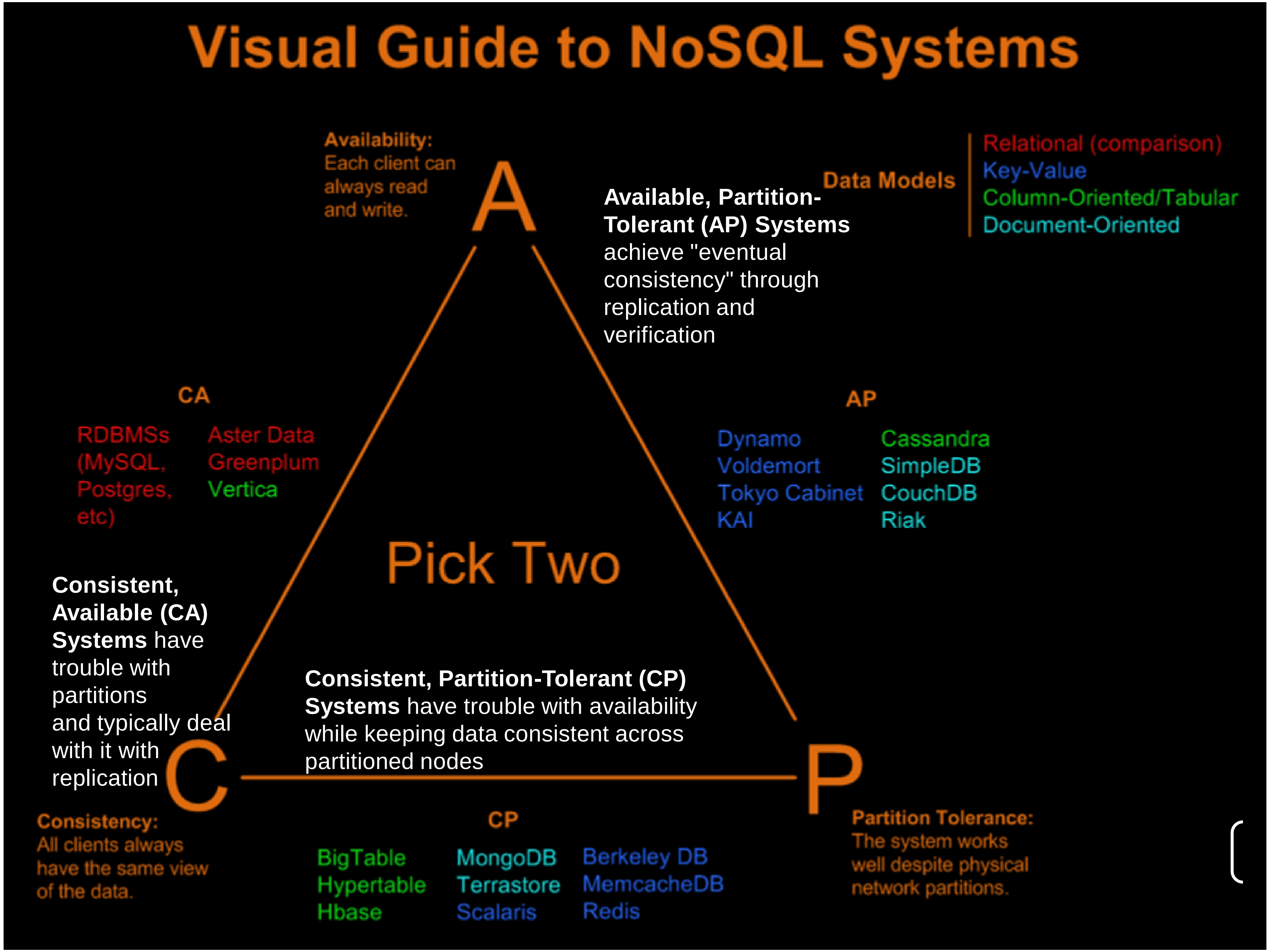
Taxonomy of NoSQL

- **Key-value** redis riak
- **Graph database** Neo4j the graph database HyperGraphDB
- **Document-oriented** mongoDB CouchDB relax
- **Column family** Cassandra HBASE

TypicalNoSQL architecture



Hashing function maps each key to a server (node)



Visual Guide to NoSQL Systems

Availability: Each client can always read and write.

Available, Partition-Tolerant (AP) Systems achieve "eventual consistency" through replication and verification

Data Models

- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Pick Two

Consistent, Available (CA) Systems have trouble with partitions and typically deal with it with replication

Consistent, Partition-Tolerant (CP) Systems have trouble with availability while keeping data consistent across partitioned nodes

Consistency: All clients always have the same view of the data.

CP

BigTable
Hypertable
Hbase

MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

Partition Tolerance: The system works well despite physical network partitions.

http://blog.nahurst.com/visual-guide-to-nosql-systems

ACID Properties

- **Atomicity**: This property ensures that either all the operations of a transaction reflect in database or none
- **Consistency**: To preserve the consistency of database, the execution of transaction should take place in isolation (that means no other transaction should run concurrently when there is a transaction already running).
- **Isolation**: For every pair of transactions, one transaction should start execution only when the other finished execution.
- **Durability**: Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure.

How doesNoSQL vary from RDBMS?

- Looser schema definition
- Applications written to deal with specific documents/ data
    - Applications aware of the schema definition as opposed to the data
- Designed to handle distributed, large databases
- Trade offs:
    - No strong support for ad hoc queries but designed for speed and growth of database
        - Query language through the API
    - Relaxation of the ACID properties

Benefits of NoSQL

Elastic Scaling

RDBMS scale up – bigger load , bigger server

NO SQL scale out – distribute data across multiple hosts seamlessly

DBA Specialists

RDMS require highly trained expert to monitor DB

NoSQL require less management, automatic repair and simpler data models

Big Data

- Huge increase in data RDMS: capacity and constraints of data volumes at its limits
- NoSQL designed for big data

Benefits of NoSQL

Flexible data models

- Change management to schema for RDMS have to be carefully managed
- NoSQL databases more relaxed in structure of data
  - Database schema changes do not have to be managed as one complicated change unit
  - Application already written to address an amorphous schema

Economics

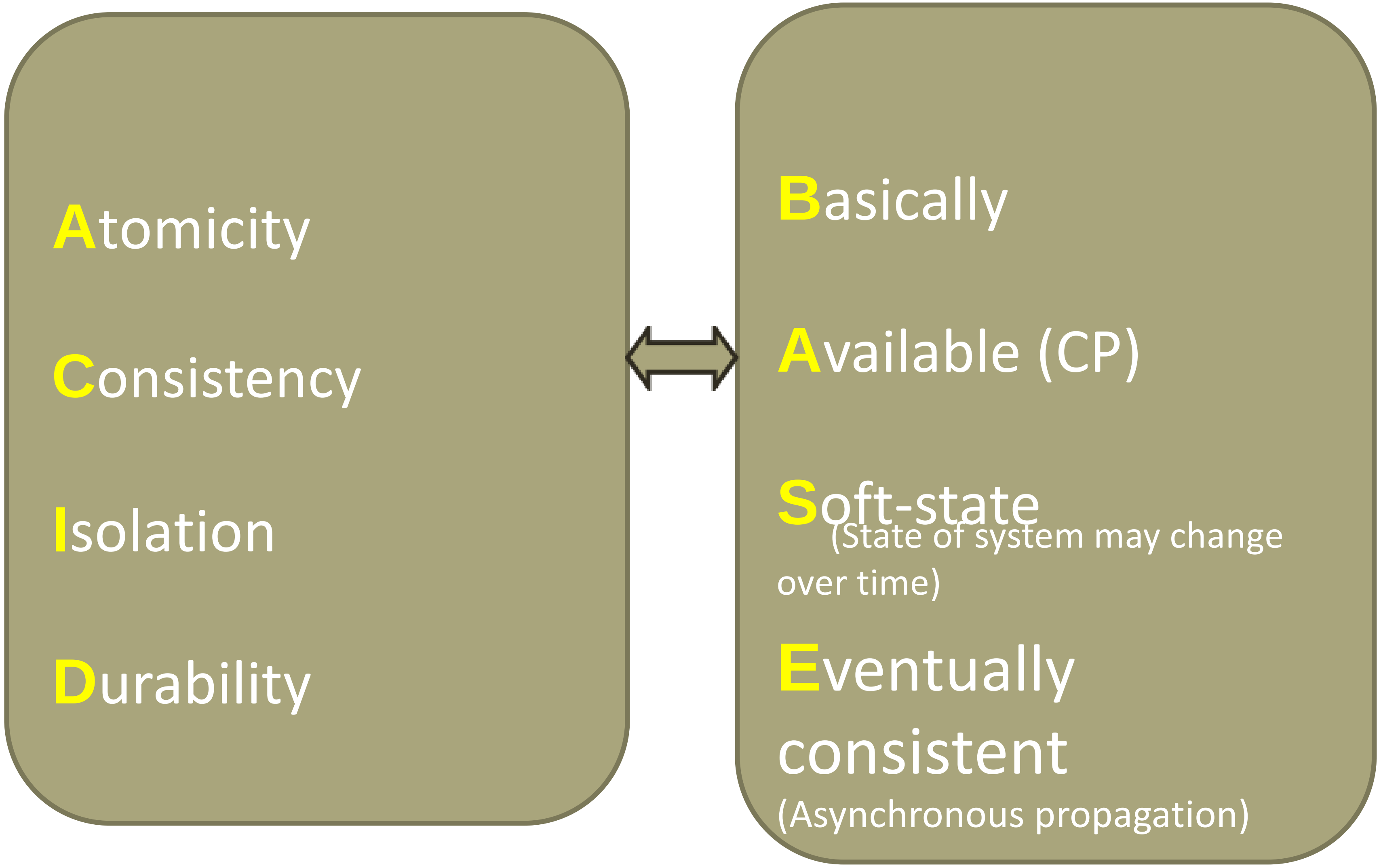
- RDMS rely on expensive proprietary servers to manage data
- No SQL: clusters of cheap commodity servers to manage the data and transaction volumes
- Cost per gigabyte or transaction/second for NoSQL can be lower than the cost for a RDBMS

Drawbacks of NoSQL

- Support
  - RDBMS vendors provide a high level of support to clients
    - Stellar reputation
  - NoSQL – are open source projects with startups supporting them
    - Reputation not yet established

- Maturity
  - RDMS mature product: means stable and dependable
    - Also means old no longer cutting edge nor interesting
  - NoSQL are still implementing their basic feature set

RDB ACID to NoSQL BASE



Atomicity

Consistency

Isolation

Durability

Basically

Available (CP)

Soft-state (State of system may change over time)

Eventually consistent (Asynchronous propagation)

Pritchett, D.: BASE: An Acid Alternative (queue.acm.org/detail.cfm?id=1394128)

First example: mongoDB

What is MongoDB?

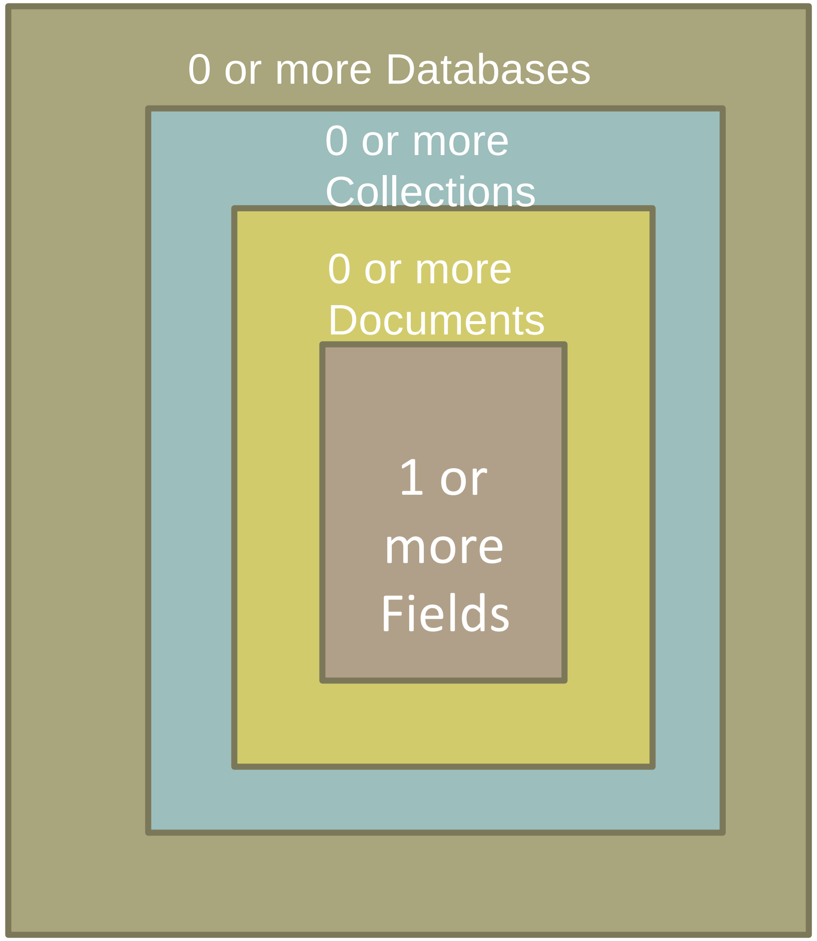
- Developed by 10gen
  - Founded in 2007
- A document-oriented, NoSQL database
  - Hash-based, s*chema-less database*
    - No Data Definition Language
    - In practice, this means you can store hashes with any keys and values that you choose
      - Keys are a basic data type but in reality stored as strings
      - Document Identifiers (_id) will be created for each document, field name reserved by system
    - Application tracks the schema and mapping
    - Uses BSON format
      - Based on JSON – B stands for Binary
- Written in C++
- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

Why use MongoDB?

- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
  - No ERD diagram
- Not well suited for heavy and complex transactions systems

MongoDB: HierarchicalObjects

- A MongoDB instance may have zero or more 'databases'
- A database may have zero or more 'collections'.
- A collection may have zero or more 'documents'.
- A document may have one or more 'fields'.
- MongoDB 'Indexes' function much like their RDBMS counterparts.



RDB Concepts to NO SQL

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Collection is not strict about what it Stores

Schema-less

Hierarchy is evident in the design

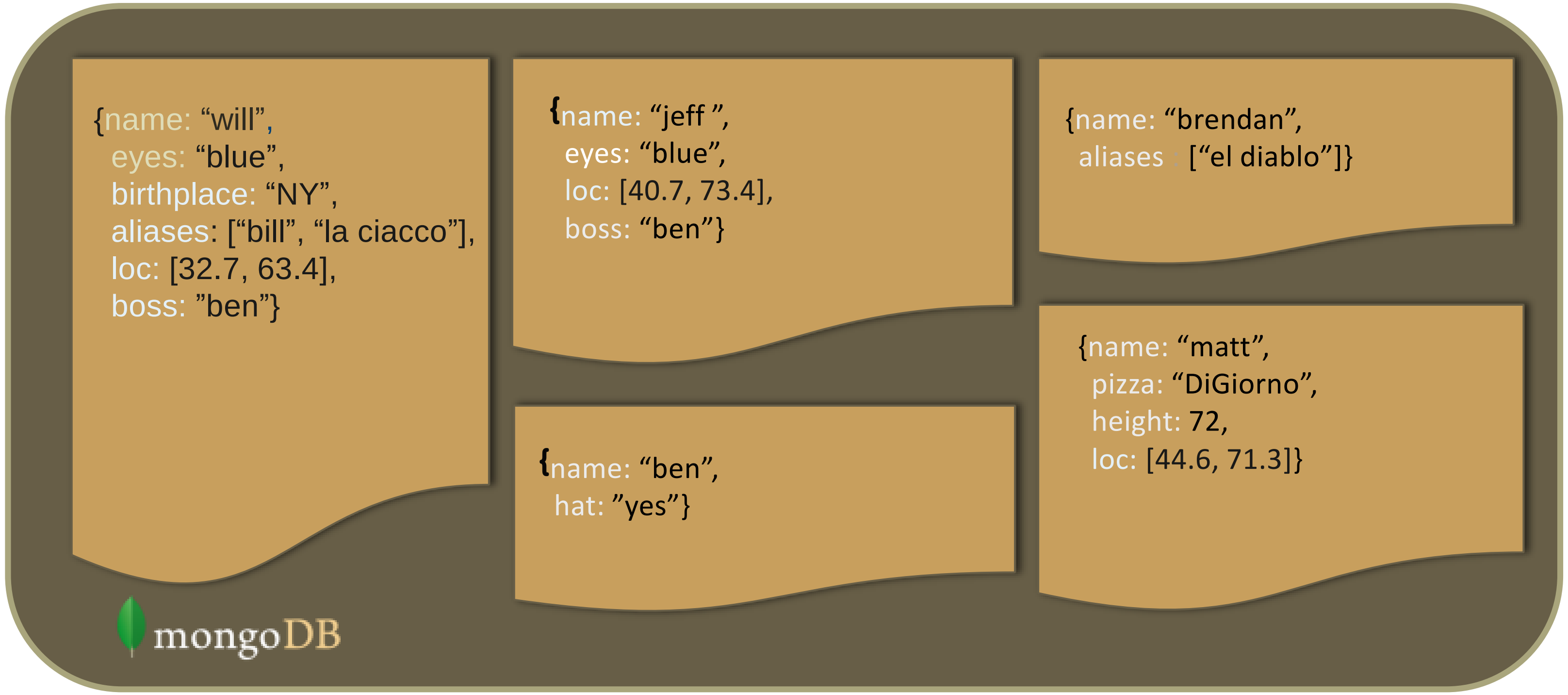
Embedded Document ?

MongoDB Processes and configuration

- Mongod – Database instance
- Mongos - Sharding processes
  - Analogous to a database router.
  - Processes all requests
  - Decides how many and which *mongod*s should receive the query
  - *Mongos* collates the results, and sends it back to the client.
- Mongo – an interactive shell (a client)
  - Fully functional JavaScript environment for use with a MongoDB

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
  - Addresses NULL data fields



```
{name: “will”,
  eyes: “blue”,
  birthplace: “NY”,
  aliases: [“bill”, “la ciacco”],
  loc: [32.7, 63.4],
  boss: ”ben”}
```

```
{name: “jeff ”,
  eyes: “blue”,
  loc: [40.7, 73.4],
  boss: “ben”}
```

```
{name: “ben”,
  hat: ”yes”}
```

```
{name: “brendan”,
  aliases   [“el diablo”]}
```

```
{name: “matt”,
  pizza: “DiGiorno”,
  height: 72,
  loc: [44.6, 71.3]}
```

- Data is in name / value pairs
- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”
- Data is separated by commas
  - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
  - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
  - Example [{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”},
  - {“name”: “Yoda”, affiliation: “rebels”}]

JSON format

- Data is in name / value pairs
  - A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”
- Data is separated by commas
  - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
  - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
  - Example

    [{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”},
    {“name”: “Yoda”, affiliation: “rebels”}]

CRUD operations

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

Collection specifies the collection or the ‘table’ to store the document

Create Operations

Db.collection specifies the collection or the ‘table’ to store the document

- db.collection_name.insert(<document>)
  - Omit the _id field to have MongoDB generate a unique key
  - Example db.**parts**.insert({{type: “screwdriver”, quantity: 15 })
  - db.**parts**.insert({_id: 10, type: “hammer”, quantity: 1 })
- db.collection_name.update(<query>, <update>, { upsert: true })
  - Will update 1 or more records in a collection satisfying query
- db.collection_name.save(<document>)
  - Updates an existing record or creates a new record

Read Operations

- db.collection.find(<query>, <projection>).cursor modified
  - Provides functionality similar to the SELECT command
    - <query> where condition , <projection> fields in result set
  - Example: var PartsCursor = db.parts.find({parts:“hammer”}).limit(5)

  - Has cursors to handle a result set
  - Can modify the query to impose limits, skips, and sort orders.
  - Can specify to return the ‘top’ number of records from the result set
- db.collection.findOne(<query>, <projection>)

Query Operators

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a loginal AND
$not	Inverts the effect of a query expression

https://docs.mongodb.org/manual/reference/operator/query/

Update Operations

- db.collection_name.insert(<document>)
    - Omit the _id field to have MongoDB generate a unique key
    - Example db.**parts**.insert({{type: “screwdriver”, quantity: 15 })
    - db.**parts**.insert({_id: 10, type: “hammer”, quantity: 1 })
- db.collection_name.save(<document>)
    - Updates an existing record or creates a new record
- db.collection_name.update(<query>, <update>, { upsert: true })
    - Will update 1 or more records in a collection satisfying query
- db.collection_name.findAndModify(<query>, <sort>, <update>,<new>, <fields>,<upsert>)
    - Modify existing record(s) – retrieve old or new version of the record

Delete Operations

- db.collection_name.remove(<query>, <justone>)
  - Delete all records from a collection or matching a criterion
  - <justone> - specifies to delete only 1 record matching the criterion
  - Example: db.parts.remove(type: /^h/ }) - remove all parts starting with h
  - Db.parts.remove() – delete all documents in the parts collections

CRUD examples

```
> db.user.insert({
        first: "John",
        last : "Doe",
        age: 39
})
```

```
> db.user.find ()
{ "_id" : ObjectId("51"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

```
> db.user.update(
        {"_id" : ObjectId(“51")},
        {
            $set: {
                age: 40,
                salary: 7000}
        }
)
```

```
> db.user.remove({
        "first": /^J/
})
```

SQL vs. Mongo DB entities

My SQL

```
START TRANSACTION;
INSERT INTO contacts VALUES
    (NULL, ‘joeblow’);
INSERT INTO contact_emails
VALUES
   ( NULL, ”joe@blow.com”,
      LAST_INSERT_ID() ),
   ( NULL,
“joseph@blow.com”,
      LAST_INSERT_ID() );
COMMIT;
```

Mongo DB

```
db.contacts.save( {
    userName: “joeblow”,
    emailAddresses: [
      “joe@blow.com”,
      “joseph@blow.com” ] }
);
```

DIFFERENCE: MongoDB separates physical structure from logical structure

Designed to deal with large &distributed

Demo

- Install MongoDB community edition
- MYSQL workbench + MYSQL server

References

- https://docs.mongodb.com/manual/reference/database-references/
- https://www.guru99.com/mongodb-tutorials.html

For next week

- Review today’s slides and topics
- Try a hands on MySQL and MongoDB
  - Create database in MYSQL and MongoDB
  - Insert records into your database
  - Query your database
- We didn’t cover how to connect this to your site yet!
- If you feel challenged: Create one page app using Angularjs + Mongodb (hint: you will need Nodejs+ Express to do that)
- For next week
  - https://www.geeksforgeeks.org/introduction-java-servlets/
  - https://www.tutorialspoint.com/servlets/
  - Review MVC from last week lecture

Enjoy your fall break !