The background features a light blue and yellow illustration of a data storage and retrieval system. It includes a yellow folder icon on the left, a globe in the upper center, a computer monitor in the center, and a stack of four yellow server units on the right. Faint lines connect these elements, suggesting a network or data flow.

# Biomedical Data Storage and Retrieval with SQL, NoSQL databases

# Learning Objectives

- To that end, upon completion of the seminar you should be able to:
  - Understand database concepts and terminology
  - Design and create tables
  - Use SQL / NoSQL to retrieve and analyze information
  - Use SQL / NoSQL to enter and manipulate data

# Outline

## Concepts

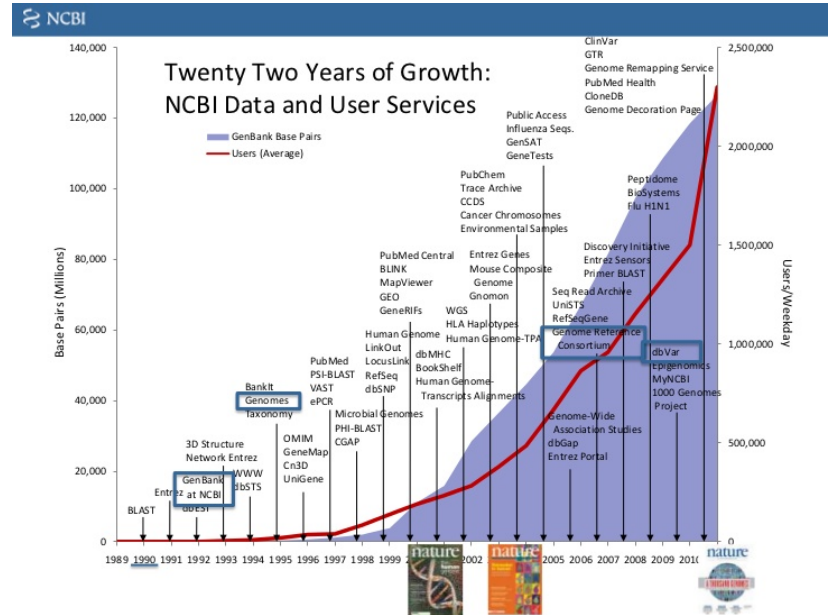
- Databases And Systems
- Relational Databases & SQL
- NoSQL Databases

## Hands-on

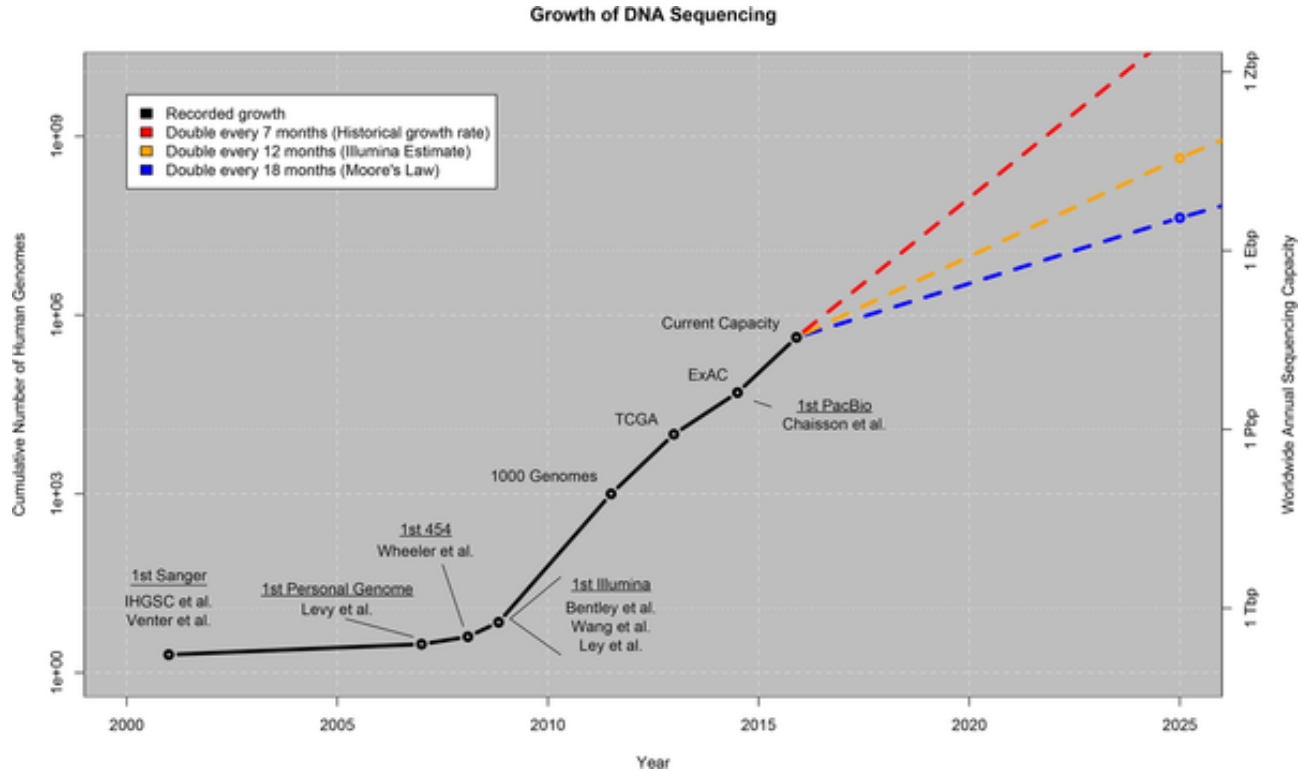
- Relational Database (MySQL)
- SQL
- No SQL Database (MongoDB)

# Biomedical Data

- Types of biomedical data
  - NGS reads
  - Sequences
  - 3D Protein Structure
  - Biological Networks
  - Expression data
  - Variants
  - Images
  - Geographical location
  - Electrical signals
  - Etc



## Growth of DNA sequencing

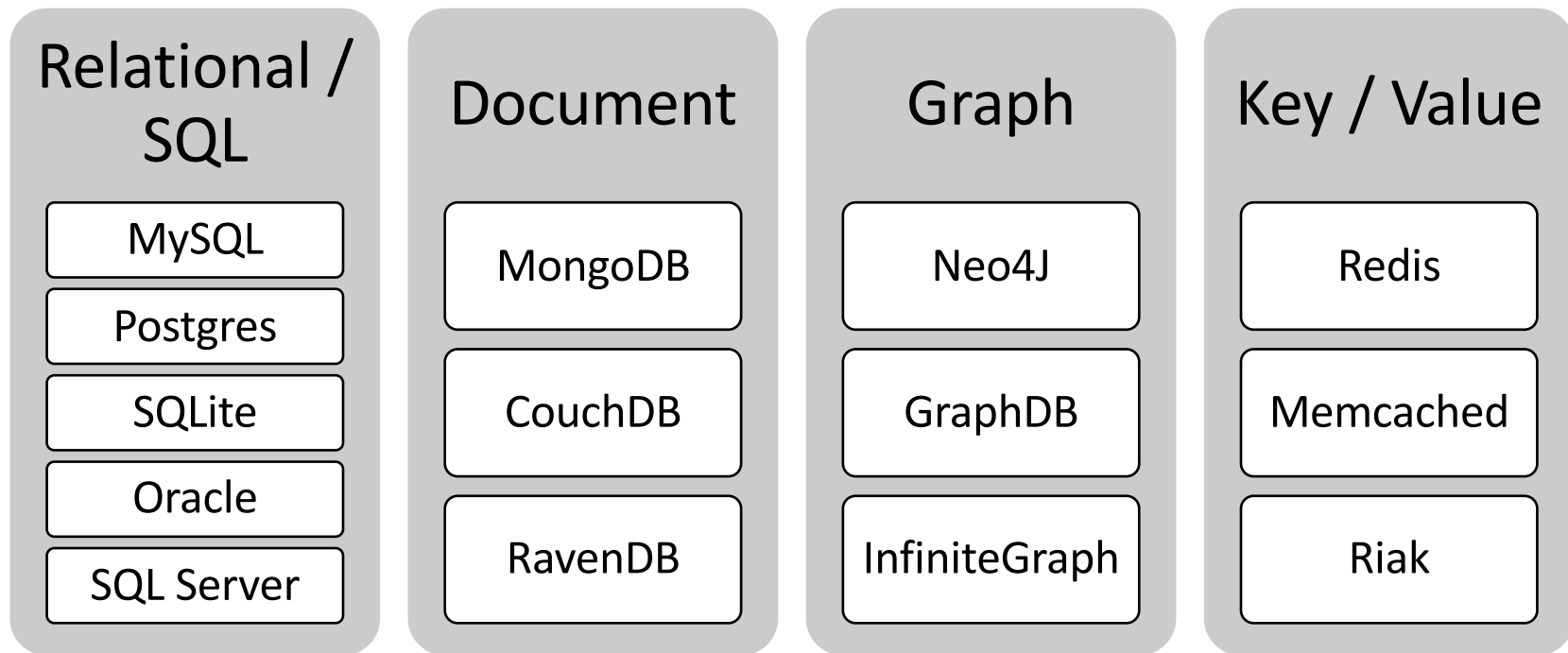


Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, et al. (2015) Big Data: Astronomical or Genomical?. PLOS Biology 13(7): e1002195.

doi:10.1371/journal.pbio.1002195

<http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002195>

# How Do Databases Store Information?



# ...But I just Want to Store MY Data...

- Understood!
- We will look at how to setup and use both relational databases and NoSQL or document databases

# Relational Databases



# What Is A Relational Database

- Structured collection of data
- Database (many tables)
- Table (many columns)
- Fields (columns)



	t_tax	rank	rank
		no rank	no rank
	57	superkingdom	no rank
	28	genus	family
		species	genus
	32199	species	genus
10	1706371	genus	family
11	1707	species	genus
13	203488	genus	family
14	13	species	genus
16	32011	genus	family

# Database Properties

- Scalability: large amounts of information/data
- Concurrency: support multiple users
- Persistency: data is always preserved
- Security: robustness to system failures and malicious users/programs
- Data independence: computer program is independent of the physical storage structure of the data files

# What is a table?

- Tables are the fundamental component of any relational database.
- A table stores the data corresponding to a specific type of object
- For example:
  - A Students table would store student information
  - An Orders table would store customer order information

tax_id	parent_tax_id	rank	rank
1	1	no rank	no rank
2	131567	superkingdom	no rank
6	335928	genus	family
7	6	species	genus
9	32199	species	genus
10	1706371	genus	family
11	1707	species	genus
13	203488	genus	family
14	13	species	genus
16	32011	genus	family

# What is a *table*?

- Tables are made up of *rows* and *columns*
- The ***columns*** of a table describe the characteristics of the information stored in that table
- The data in each ***row*** belongs to a given instance of the type of data stored in a particular table

tax_id	parent_tax_id	rank	rank
1	1	no rank	no rank
2	131567	superkingdom	no rank
6	335928	genus	family
7	6	species	genus
9	32199	species	genus
10	1706371	genus	family
11	1707	species	genus
13	203488	genus	family
14	13	species	genus
16	32011	genus	family

# What is a *table*?

- Each row contains one *data value* per column.
- The range of values that can go into a particular column of a row is called the *domain* of that column, and is generally restricted to data of a specific type (integers, character data, dates, etc.)

tax_id	parent_tax_id	rank	rank
1	1	no rank	no rank
2	131567	superkingdom	no rank
6	335928	genus	family
7	6	species	genus
9	32199	species	genus
10	1706371	genus	family
11	1707	species	genus
13	203488	genus	family
14	13	species	genus
16	32011	genus	family

# Example Relational Database Table

tax_id	parent_tax_id	rank	rank
1	1	no rank	no rank
2	131567	superkingdom	no rank
6	335928	genus	family
7	6	species	genus
9	32199	species	genus
10	1706371	genus	family
11	1707	species	genus
13	203488	genus	family
14	13	species	genus
16	32011	genus	family

# What Are They Called “Relational?”

- The term “relational database” comes from the mathematical definition of a *relation*, or set. All objects in a relation must have the same properties or characteristics
- The point? Tables group similar data or objects, with use one table per set of objects
- (BUT there can also be relations of one table to another)

# How Do I Get Started?

- By installing and setting up a Relational Database Management System (RDBMS or just DBMS)
- What is a DBMS?
  - A software package or system that facilitates the creation and maintenance of a computerized database, allowing data to persist over long periods of time
- Do I have to?
  - No...you can also use the SQLite database for personal tasks!



# When Should You Use A (Relational) Database?

- Three criteria to consider:
  - Does it need to be ACID compliant?
  - How complex the data is
  - how many things need to read/write it
- Atomicity
  - Each transaction be "all or nothing": if one part of a transaction fails, then the entire transaction fails, and the database state is left unchanged.
- Consistency
  - Ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules.
- Isolation
  - Ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially, i.e., one after the other.
- Durability
  - Ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors, i.e. the results need to be stored permanently.

# When to Use RDBMS, NoSQL, files?

- For when to use a RDBMS:
  - You have relational data, i.e. you have a customer who purchases your products and those products have a supplier and manufacturer
  - You have large amounts of data and you need to be able to locate relevant information quickly
  - You need to start worrying about the previous issues identified: scalability, reliability, ACID compliance
  - You need to use reporting or intelligence tools to work out business problems
- As for when to use a NoSQL
  - You have lots of data that needs to be stored which is unstructured
  - Scalability and speed needs
  - You generally don't need to define your schema up front, so if you have changing requirements this might be a good point
- Finally, when to use files
  - You have unstructured data in reasonable amounts that the file system can handle
  - You don't care about structure, relationships
  - You don't care about scalability or reliability (although these can be done, depending on the file system)
  - You don't want or can't deal with the overhead a database will add
  - You are dealing with structured binary data that belongs in the file system, for example: images, PDFs, documents, etc.

# Why Use A DBMS?

- Database manipulations: insertions, deletions, and modifications
- Efficient querying
- Concurrent processing and sharing by multiple users
- Consistent and valid data
- Recovery after crashes
- Security and user authorization

# What Relational Database To Choose?

## **SQLite Works Well For...**

- One user writing, one or more users reading
- Application file format
- Websites
- Data analysis
- Cache for enterprise data
- Server-side database
- File archives
- Replacement for ad hoc disk files
- Internal or temporary databases
- Education and Training

## **Client/Server DBMS Work Well For...**

- Client/Server Applications
- High-volume Websites
- Very large datasets
- High Concurrency

# Database Design

# Database Design

- Consider the nouns or entities you will want to store data about
- Consider the characteristics of each entity
- Consider how each entity relates to any other entity
- Are the relationships one-to-one, many-to-one, or many-to-many?
- This is typically referred to as Entity-Relationship (or *E-R Modeling*)

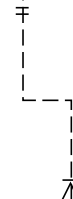
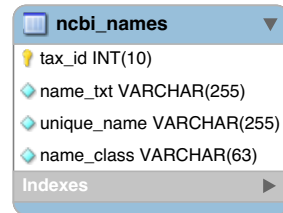
# Database Design

- NCBI (Taxonomy) Names

- tax\_id
- name\_txt
- unique\_name
- name\_class

- NCBI (Taxonomy) Nodes

- tax\_id
- parent\_tax\_id
- rank
- embl\_code
- division\_id
- ...



# Creating Tables

- With an E-R Model in hand, the actual creation of a database is very straightforward
- Each *entity* will generally become a *table*
- Any *relationships* with one or more *attributes* will also become a *table*
- *Relationships* without attributes *may sometimes* be modeled as a table



# Ground Rules for Relational Database Design

- No multi-part or multi-value fields
- Eliminate redundant information
- Avoid designs that call for the addition of columns to store new data
- Avoid “anomalies”:
  - Update
  - Delete
  - Insert

# Keys

- A *key* is a field or set of fields that can be used to uniquely identify a particular record in a database table.
  - Examples:
    - Name, Department
    - SSN
- A Primary Key is a field or set of fields chosen by the database designer to be used to define relationships between tables
  - • Primary Key: A primary key is unique. A key value can not occur twice in one table. With a key, you can find at most one row.
  - • Foreign Key: A foreign key is the linking pin between two tables.

# Primary Keys

- Primary Key fields:
  - Must contain unique, non-duplicated values (or sets of values in the case of multi-field keys)
  - Cannot be NULL
- In the event that no one column in a table is an appropriate Primary Key, an artificial primary key column is usually generated

# Foreign Keys

- When the values of a Primary Key column in a table are shared by a common column in a child table, that column is called the Foreign Key of the relationship
- Foreign Key fields must have the same data type and size as their related Primary Key field
- Foreign Keys are used in other tables in order to maintain relationships between data values

# Cardinality

- *Cardinality* is the term used to describe the character of a relationship
- Three types of cardinality exist:
  - *One to Many*
  - *Many to Many*
  - *One to One*

# What Can SQL do?

- SQL can:
  - execute queries against a database
  - retrieve data from a database
  - insert records in a database
  - update records in a database
  - delete records from a database
  - create new databases
  - create new tables in a database
  - create stored procedures in a database
  - create views in a database
  - set permissions on tables, procedures, and views

# SQL – Two Roles

- SQL can be primarily divided into two parts:
  - Data Manipulation Language (DML)
  - Data Definition Language (DDL)
- The query and update commands form the DML part of SQL:
  - SELECT - extracts data from a database
  - UPDATE - updates data in a database
  - DELETE - deletes data from a database
  - INSERT INTO - inserts new data into a database

# SQL – Two Roles

- The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose constraints between tables. The most important DDL statements in SQL are:
  - CREATE DATABASE - creates a new database
  - ALTER DATABASE - modifies a database
  - CREATE TABLE - creates a new table
  - ALTER TABLE - modifies a table
  - DROP TABLE - deletes a table
  - CREATE INDEX - creates an index (search key)
  - DROP INDEX - deletes an index



# Data Types

- Each column of a table must have a specified data type supported by the database software.
- The database will enforce the requirement that data entered into a field must comply with the field's data type
- Choose the smallest data type appropriate for each column, leaving room for future expansion

# Data Types

- INTEGER
- FLOAT (floating point numbers)
- NUMERIC (real numbers)
- CHAR
- VARCHAR
- TEXT
- TIMESTAMP
- DATE

# Creating Tables

- Choose good field and table names
  - No spaces or special characters (other than the underscore “\_” )
  - Use meaningful names

# Create Table

- You can build a table in a SQL-compatible database with the CREATE TABLE statement

```
CREATE TABLE 'ncbi_names' (  
    'tax_id' int unsigned NOT NULL,  
    'name_txt' varchar(255) NOT NULL,  
    'unique_name' varchar(255) NOT NULL DEFAULT "",  
    'name_class' varchar(63) NOT NULL );
```

# NULL Values

- Relational databases have a special value, called the NULL, that can be used as a placeholder when information is unknown, missing, or to be filled in later.
- NULL values are different than
  - Zero
  - Empty Strings
- A column of any data type can be NULL, though PRIMARY KEY columns can never be NULL

# Default Values

- Each column in a table may have a default value specified.
- The database will set the column to that value if no value is passed during insert
- If no default value is specified, the column will be set to NULL if possible.

# Working with MySQL

# MySQL Installation

- MySQL installation
  - <https://dev.mysql.com/doc/refman/5.7/en/installing.html>
- MySQL Workbench
  - <https://dev.mysql.com/downloads/workbench/>



# MySQL Workbench

MySQL Workbench interface showing the execution of a SQL query and the resulting data set.

**Execute SQL** (Red arrow pointing to the Execute button in the toolbar)

**SQL Editor** (Red arrow pointing to the SQL Editor window)

**Schema & Tables** (Red arrow pointing to the Schema browser on the left)

**Result Set** (Red arrow pointing to the Result Grid)

**Query 1** (SQL statement in the editor):

```
SELECT * FROM ncbi_taxon.ncbi_names;
```

**Result Grid** (Table of results):

tax_id	name_txt	unique_name	name_class
1	Bacteria	Bacteria <prokaryotes>	scientific name
2	not Bacteria Haeckel 1894		synonym
6	Azorhizobium		scientific name
6	Azorhizobium Dreyfus et al. 1988 emend. Lang et al. 2013		authority
7	Azorhizobium caulinodans		scientific name
7	Azorhizobium caulinodans Dreyfus et al. 1988		authority
9	Buchnera aphidicola		scientific name
9	Buchnera aphidicola Munson et al. 1991		authority
10	"Cellvibrio" Winogradsky 1929		synonym

**Action Output** (Log of the executed query):

Time	Action	Response	Duration / Fetch Time
16:07:59	SELECT * FROM ncbi_taxon.ncbi_names LIMIT 0, 1000	1000 row(s) returned	0.00056 sec / 0.000...

# NoSQL Databases

# NoSQL (Versus SQL) Databases

## **Advantages**

- Non-Relational means table-less
- Mostly Open Source and Low-Cost
- Easier scalability through support for Map Reduce
- No need to develop a detailed database model

## **Disadvantages**

- Community not as well defined
- Lack of reporting tools
- Lack of standardization















# No SQL Overview

- The problems with Relational Database:
  - Overhead for complex select, update, delete operations
    - Select: Joining too many tables to create a huge size table
    - Update: Each update affects many other tables
    - Delete: Must guarantee the consistency of data
  - Not well-supported the mix of unstructured data
  - Not well-scaling with very large size of data
- NoSQL is a good solution to deal with these problems

# Overview – NoSQL Family

- Data stored in four types:

- Document
- Graph
- Key-value
- Wide-column

Document Database	Graph Databases
  	 
Wide Column Stores	Key-Value Databases
   	    

# Overview – MongoDB

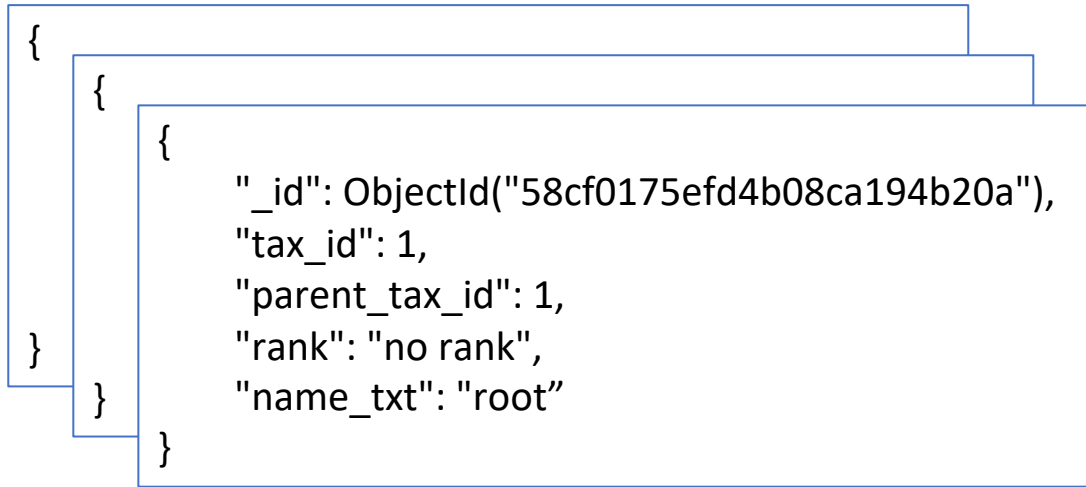
- MongoDB is:
  - An open source and document-oriented database
  - Data is stored in JSON-like documents
  - Designed with both scalability and developer agility
  - Dynamic schemas

# SQL vs MongoDB

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document
column	field
index	index
table joins (e.g. select queries)	embedded documents and linking
Primary keys	_id field is always the primary key
Aggregation (e.g. group by)	aggregation pipeline

# MongoDB Data Model

A *collection* includes *documents*.





# MongoDB Data Model

Structure of a JSON-document:

```
{
  "_id": ObjectId("58cf0175efd4b08ca194b20a"),
  "tax_id": 1,
  "parent_tax_id": 1,
  "rank": "no rank",
  "name_txt": "root"
}
```

The value of **field**:

- Native data types
  - Arrays
- Other documents

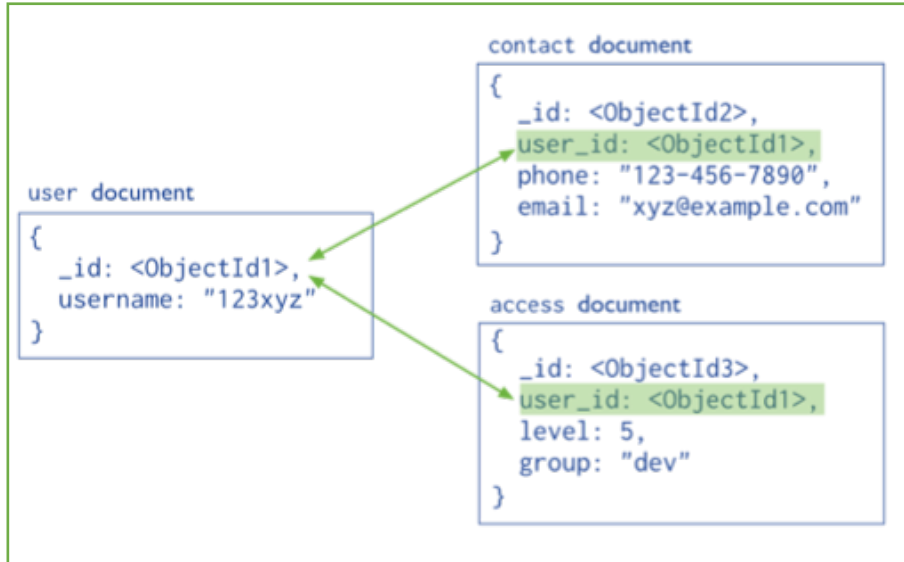
# MongoDB Data Model

Embedded documents:



# MongoDB Data Model

Reference documents or linking documents



# MongoDB Queries:

- CRUD (Create – Update – Delete)
  - Create a database: use database\_name
  - Create a collection: `db.createCollection(name, options)`
    - options: specify the number of documents in a collection etc.
  - Insert a document:
    - `db.<collection_name>.insert({“name”: “nguyen”, “age”: 24, “gender”: “male”})`
  - Query [e.g. select all]
    - `db.<collection_name>.find().pretty()`
  - Query with conditions:
    - `db.<collection_name>.find( { “gender”: “female”, “age”: {$lte:20} }).pretty()`

# MongoDB Queries:

- CRUD (Create – Update – Delete)
  - `db.<collection_name>.update(<select_criteria>,<updated_data>)`
  - `db.students.update({'name':'nguyen'}, { $set: {'age': 20 } } )`
  - Replace the existing document with new one: save method:
    - `db.students.save({_id:ObjectId('string_id'), "name": "ben", "age": 23, "gender": "male"})`

# MongoDB Queries:

- CRUD (Create – Update – Delete)
  - Drop a database
    - Show database: `show dbs`
    - Use a database: `use <db_name>`
    - Drop it: `db.dropDatabase()`
  - Drop a collection:
    - `db.<collection_name>.drop()`
  - Delete a document:
    - `db.<collection_name>.remove({"gender": "male" })`

# Installation - Mac

- <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>

# Installation - Windows

- <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>



# Sources

- David Lou, NIAID
- Brian Panulla, Web 2004 Pre-Conference presentation
- Nguyen Vo - MongoDB

# Recommended Resources

- *Fundamentals of Database Systems (latest edition)*, by Ramez Elmasri & Shamkant B. Navathe
- Database Engine Ranking
  - <https://db-engines.com/en/ranking>
- Data Management in Bioinformatics
  - [https://en.wikibooks.org/wiki/Data\\_Management\\_in\\_Bioinformatics](https://en.wikibooks.org/wiki/Data_Management_in_Bioinformatics)
- Current Protocols in Bioinformatics
  - <http://onlinelibrary.wiley.com/book/10.1002/0471250953/homepage/Archive.html>

# Resources:

## Data Management in Bioinformatics

- [https://en.wikibooks.org/wiki/Data\\_Management\\_in\\_Bioinformatics](https://en.wikibooks.org/wiki/Data_Management_in_Bioinformatics)
- Chapters
  - [Chapter 1: E/R Theory](#)
  - [Chapter 2: Normalization](#)
    - [E/R Theory and Normalization Exercises](#)
  - [Chapter 3: Data Querying](#)
  - [SQL Query Exercises](#)
  - [Integrating SQL and Programming Languages](#)
  - [Scratch Notes for Class](#)
  - [Server Programming: Add programming language to PostgreSQL](#)

# Resources:

## Bioinformatics Database Schemas

### Bioinformatics Related Schema

- BioSQL
- Chado
  - [http://www.gmod.org/wiki/Chado\\_-\\_Getting\\_Started](http://www.gmod.org/wiki/Chado_-_Getting_Started)
- ENSEMBL Schema
  - [http://useast.ensembl.org/info/docs/api/core/core\\_schema.html](http://useast.ensembl.org/info/docs/api/core/core_schema.html)
- BioMart

### Global Alliance for Genomics and Health

- Genomics API
  - <https://github.com/ga4gh/ga4gh-schemas>
  - Just retired (jan 2018); no replacement yet
- New technical specifications are being developed in the following areas:
  - Clinical & Phenotypic Data Capture
  - Cloud
  - Data Use & Researcher Identities
  - Discovery
  - Genomics Knowledge Standards
  - Large Scale Genomics

# Resources:

## SQL Editors

- <https://popsql.io>
- <https://www.araelium.com/queries>
- <https://sqlectron.github.io>
- <https://www.sqlprostudio.com>
- <https://teamsql.io>
- <https://www.sequelpro.com>

## SQL Web Tools

- <http://sqlfiddle.com>